Πανεπιστήμιο Κρήτης
Σχολή Θετικών Επιστημών
Τμήμα Επιστήμης Υπολογιστών

Σχήματα Κωδικοποίησης για το Σημασιολογικό Ιστό

Σωτήριος Β.Τουρτούνης
Μεταπτυχιακή Εργασία

Ηράκλειο, Φεβρουάριος 2003

Σχήματα Κωδικοποίησης για το Σημασιολογικό Ιστό

Σωτήριος Β. Τουρτούνης

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Περίληψη

Η μεγάλη ανάπτυξη του Σημασιολογικού Ιστού καθιστά όλο και πιο ευρύτερη την χρήση εφαρμογών που στηρίζονται σε αυτόν. Η πιο ενδεικτική κι ευρύτερα αποδεκτή κατηγορία εφαρμογών του είναι οι Διαδικτυακές Πύλες, με πλέον χαρακτηριστικό παράδειγμα τον Κατάλογο *Netscape Open Directory* (*ODP*). Η δομή της πύλης αυτή συνίσταται σε εκτεταμένες ιεραρχίες υποκλάσεων (*isA*) από διάφορα επιμέρους θέματα. Συνεπώς, η βελτιστοποίηση της πλοήγησης σε αυτού του είδους τις πύλες που μεταφράζεται σε επερωτήσεις μεταβατικού κλεισίματος, είναι ένα ανοικτό πρόβλημα. Συγκεκριμένα, θα συνηγορήσουμε στην χρήση σχημάτων κωδικοποίησης για την μοντελοποίηση των θεματικών αυτών ιεραρχιών στοχεύοντας στην απάντηση βασικών επερωτήσεων όπως είναι ο έλεγχος υπαλληλίας, απόγονοι, πρόγονοι, "κοντινότερος" κοινός πρόγονος κλπ. Βασικός στόχος μας με την επιλογή των σχημάτων για την κωδικοποίηση των ιεραρχιών είναι η αποφυγή του υπολογισμού των χρονοβόρων μεταβατικών κλεισιμάτων, λαμβάνοντας υπόψη ότι α) οι ιεραρχίες υπαλληλίας των

κλάσεων των πυλών μπορεί να κυμαίνονται μεταξύ απλών δέντρων και σύνθετων DAGs κι άρα θα πρέπει το σχήμα κωδικοποίησης να μπορεί να ανταποκριθεί σε μια λογική αύξηση του χώρου αποθήκευσης και του χρόνου εκτέλεσης των επερωτήσεων, και β) η πλοήγηση στα σχήματα των πυλών βασίζεται στην ανάκτηση κλάσεων με την χρήση σύνθετων συνθηκών φιλτραρίσματος στις σχέσεις υπαλληλίας κι συνεπώς χρειάζεται η δημιουργία ετικετών για τις κλάσεις, να είναι κατάλληλη ώστε να μπορούν ικανοποιητικά να χρησιμοποιηθούν από δεδομένους δείκτες μιας βάσης

Αρχικά, θα παρουσιάσουμε μια ποιοτική σύγκριση των τριών βασικών κατηγοριών σχημάτων κωδικοποίησης, που είναι α) τα σχήματα με χρήση διανυσμάτων - δηφίων, β) τα σχήματα με χρήση προθέματος και γ) τα σχήματα με χρήση διαστημάτων. Κατόπιν, θα προχωρήσουμε στην παρουσίαση δυο επιμέρους σχημάτων κωδικοποίησης ως βέλτιστες επιλογές κρίνοντάς τα ως προς την ικανοποιητική υλοποίηση των επερωτήσεων με ετικέτες, χρησιμοποιώντας ένα καθιερωμένο σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων. Αναφορικά, πρόκειται για το σχήμα του *Dewey* με χρήση προθέματος και για το σχήμα με χρήση διαστήματος των *Agrawal*, *Borgida* και *Jagadish*. Η επιμέρους σύγκριση των δυο σχημάτων αυτών θα γίνει με μία πειραματική αξιολόγηση τους με βάση τα κριτήρια του χρόνου εκτέλεσης των προαναφερθέντων βασικών επερωτήσεων αλλά και του αναγκαίου χώρου αποθήκευσης των κωδικοποιημένων ιεραρχιών χρησιμοποιώντας το σύστημα διαχείρισης βάσεων δεδομένων της *PostgreSQL*. Ως πεδίο δοκιμής της αξιολόγησης αυτής θα χρησιμοποιήσουμε κατ' αρχήν τις 16 θεματικές ιεραρχίες του *ODP* καταλόγου που έχουν την μορφολογία δέντρου, αλλά εν συνεχεία με την παραγωγή ενός συνθετικά δημιουργούμενου γράφου εκ των ιεραρχιών αυτών θα επεκτείνουμε την αξιολόγηση αυτή. Τέλος η σύγκριση αυτή παράλληλα θα μελετήσει και την επίδραση της χρήση ή μη της δυνατότητας συμπίεσης των ετικετών στην έκφραση κι υλοποίηση των επερωτήσεων αυτών.

Τα αποτελέσματα από την πειραματική αξιολόγηση της χρήσης των δυο σχημάτων κωδικοποίησης είναι αρχικά ότι οι επερωτήσεις τους είναι 2 μέχρι και 3 φορές πιο γρήγορες έναντι των αντίστοιχων στα μεταβατικά κλεισίματα. Για την περίπτωση του δέντρου τα δυο σχήματα κωδικοποίησης έχουν σχεδόν παρόμοιες επιδόσεις στις επερωτήσεις, με τον *Dewey* να απαιτεί κατά μέσο όρο 8 φορές περισσότερο έναντι του

Agrawal. Στην περίπτωση του γράφου έχουμε μια επιβάρυνση στο χώρο αποθήκευσης 2,24 φορές περισσότερο έναντι του δέντρου στον *Agrawal* (με συμπίεση) και 2,76 φορές περισσότερο (χωρίς συμπίεση), και 2,71 φορές περισσότερο στον *Dewey* (με συμπίεση). Από την άλλη πλευρά ο χρόνος εκτέλεσης των επερωτήσεων είναι κατά μέσο όρο 5 φορές μεγαλύτερος από ότι την περίπτωση του δέντρου για τον *Agrawal*, ενώ του *Dewey* είναι 38 φορές μεγαλύτερος. Τέλος, συγκρίνοντας και τα δυο σχήματα κωδικοποίησης για τις περιπτώσεις για με αλλά και χωρίς συμπίεση, έχουμε παρόμοιους χρόνους εκτέλεσης.

Επόπτης : Δημήτριος Πλεξουσάκης
Αναπληρωτής Καθηγητής,
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Labeling Schemes for the Semantic Web

Sotirios V. Tourtounis

Master of Science Thesis

Computer Science Department
University of Crete

Abstract

The great evolution of Semantic Web signifies continuously even more the use of its applications. Definitely, the most indicative and accepted category of it are the Portals, *Netscape Open Directory* (*ODP*) Catalogue being the most representative paradigm of them. The structure of that portal consists of voluminous subclass hierarchies (*isA*) from many distinctive topics. Therefore, the optimization of portal navigation, which is translated to transitive closure queries, is an open problem. So, we advocate for the use of labeling schemes for modeling topics hierarchies in order to efficiently answer questions such as subsumption check, descendants, ancestors, near common ancestor etc. Our basic objective with the choice of labeling schemes is the avoidance of costly transitive closure computations, by taking into account that a) portals *isA* hierarchies of classes may range from simple trees to complex *DAGs* and therefore we need a labeling scheme that corresponds to a reasonable storage and querying cost, and b) navigation to portal schemes relies on class retrieval using complex filtering conditions on subsumption

relationships and thus we need the generation of class labels to be proper in order to be used by standard index structures.

Initially, we introduce a qualitative comparison of the three basic categories of labeling schemes, which are a) schemes with bit-vector b) prefix-based schemes and c) interval-based schemes. Afterwards, we present two labeling schemes as optimal choices based on their satisfactory implementation of queries with labels, using a standard DBMS. Relatively, are the schema of Dewey (prefix-based) and the interval-based schema of Agrawal, Borgida and Jagadish. The comparison of those two schemes will be done on the criteria of the execution times of the aforementioned basic queries and the necessary storage space using *PostgreSQL DBMS*. The testbed of our evaluation are the 16 topic hierarchies of the *ODP* Catalogue, which are trees and then with the production of a synthetic graph from those initial hierarchies in order to expand this evaluation. This comparison will also study the effect of the use or not of label compression to the representation of those queries.

The results of the experimental evaluation of the use of the two labeling schemes are initially that their queries are 2 or even 3 times faster than those of transitive closures. In the case of tree those two labeling schemes have almost akin performance to the queries, with *Dewey* demanding 8 times more than *Agrawal*. In the case of graph, we have an encumberment to the storage space 2,24 times more than in the case of trees for *Agrawal* (with compression) and 2,76 times more (with no compression) and 2,71 times more in *Dewey* (with compression). On the other side, the queries execution time is on the average 5 times more than the case of tree for *Agrawal*, while for *Dewey* is 38 times bigger. Finally, by comparing the two labeling schemes for the case of with or without label compression, we have akin execution times.

Supervisor : Dimitrios Plexousakis
Associate. Professor,
Computer Science Department
University Of Crete

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τους επιβλέποντες της εργασίας μου αυτής, κ. Βασίλη Χριστοφίδη και κ. Δημήτρη Πλεξουσάκη, που ως άμεσοι επόπτες της εργασίας μου , με καθοδήγησαν καθ' όλη την διάρκειά της σε σημαντικό βαθμό στην επιλογή, καθορισμό του θέματος, αντιμετώπιση προβλημάτων και εν τέλει επιτυχή διεκπεραίωση της εργασίας μου.

Επίσης θα ήθελα να ευχαριστήσω τους Σοφία Αλεξάκη και Γρηγόρη Καρβουναράκη, των οποίων οι μεταπτυχιακές εργασίες αποτελούν επιμέρους τμήματα-προπομποί του γνωστικού αντικείμενου αλλά και συγκεκριμένα του συνολικού συστήματος μέσα στο οποίο οργανώθηκε και διεκπεραιώθηκε η εργασία αυτή.

Επίσης να ευχαριστήσω την ομάδα του εργαστηρίου των Πληροφοριακών Συστημάτων του Ινστιτούτου Πληροφορικής –Ι.Τ.Ε. καθώς και το Τμήμα Επιστήμης Υπολογιστών του Πανεπιστημίου Κρήτης. Τον μεν πρώτο για όλον τον αναγκαίο εργαστηριακό εξοπλισμό που χρειάστηκα , και τον δε δεύτερο για την απαραίτητη γνωστική κατάρτιση που μου πρόσφερε σε προπτυχιακό και μεταπτυχιακό επίπεδο.

Ιδιαίτερες ευχαριστίες αξίζουν και στους Χάρη Μαρινάκη, Αιμιλία Μαγκαναράκη, Νίκο Δημόκα, Χρήστο Παπαχρήστο, Πελαγία Κωνσταντάκη και Γιώργο Σαπουντζή που όχι μόνο με βοήθησαν πολλάκις σε διάφορες στιγμές που είχα ανάγκη την βοήθεια τους, αλλά περισσότερο που με στήριξαν με την φιλία τους καθ' όλη την διάρκεια ολοκλήρωσης της εργασίας αυτής.

Ολοκληρώνοντας, την μεγαλύτερη ευγνωμοσύνη μου την αξίζουν οι γονείς μου, Βασίλης και Ελένη, οι οποίοι έχουν υποστεί και θυσιάσει πάρα πολλά για την όλη πνευματική μου πρόοδο και καλλιέργεια, καθώς και για την αμέριστη συμπαράστασή και στήριξή τους στην δύσκολη αυτή περίοδο της ζωής μου.

Περιεχόμενα

ΠΕΡΙΕΧΟΜΕΝΑ 1

ΚΕΦΑΛΑΙΟ 1 10

ΕΙΣΑΓΩΓΗ 13

1.1 ΔΙΑΔΙΚΤΥΑΚΕΣ ΠΥΛΕΣ 13
1.2 ΠΛΑΊΣΙΟ ΠΕΡΙΓΡΑΦΉΣ ΠΌΡΩΝ (RESOURCE DESCRIPTION FRAMEWORK – RDF) 17
1.3 ΑΝΤΙΚΕΙΜΕΝΟ ΜΕΛΕΤΗΣ ΤΗΣ ΕΡΓΑΣΙΑΣ 19
1.4 ΟΡΓΑΝΩΣΗ ΚΑΙ ΣΥΝΕΙΣΦΟΡΑ ΤΗΣ ΕΡΓΑΣΙΑΣ 21

ΚΕΦΑΛΑΙΟ 2 24

ΠΥΛΕΣ ΔΙΑΔΙΚΤΥΟΥ: ΜΟΝΤΕΛΟΠΟΊΗΣΗ ΣΕ RDF/S ΚΑΙ ΣΤΑΤΙΣΤΙΚΉ ΑΝΆΛΥΣΗ 24

2.1 ΤΟ ΠΡΟΤΥΠΟ RDF-RDF/S 25
2.2 ΠΑΡΑΔΕΙΓΜΑ ΠΥΛΗΣ ΔΙΑΔΙΚΤΥΟΥ - ODP ΚΑΤΑΛΟΓΟΣ *25*
2.3 ΑΝΑΓΚΑΙΟΤΗΤΑ ΒΑΣΙΚΩΝ ΕΠΕΡΩΤΗΣΕΩΝ 42

ΚΕΦΑΛΑΙΟ 3 50

ΟΙΚΟΓΕΝΕΙΕΣ ΣΧΗΜΆΤΩΝ ΚΩΔΙΚΟΠΟΊΗΣΗΣ ΔΈΝΤΡΩΝ ΚΑΙ ΓΡΆΦΩΝ 50

3.1 ΑΛΓΟΡΙΘΜΟΙ ΜΕ ΑΝΑΠΑΡΑΣΤΑΣΗ ΚΩΔΙΚΩΝ ΣΕ ΔΙΑΝΥΣΜΑ ΔΗΦΙΩΝ (BIT-VECTOR) 51
3.2 ΑΛΓΟΡΙΘΜΟΙ ΚΩΔΙΚΟΠΟΙΗΣΗΣ ΜΕ ΧΡΗΣΗ ΠΡΟΘΕΜΑΤΟΣ (PREFIX-BASED) 64
3.3. ΑΛΓΟΡΙΘΜΟΙ ΚΩΔΙΚΟΠΟΊΗΣΗΣ ΜΕ ΧΡΉΣΗ ΔΙΑΣΤΗΜΆΤΩΝ /ΠΕΔΊΩΝ (INTERVAL-BASED) 69
3.4 ΣΥΓΚΡΙΣΗ ΣΧΗΜΑΤΩΝ ΚΩΔΙΚΟΠΟΙΗΣΗΣ ΓΙΑ DAGS 85
3.5 ΣΥΝΟΛΙΚΑ ΣΥΜΠΕΡΑΣΜΑΤΑ 90

ΚΕΦΑΛΑΙΟ 4 91

ΑΞΙΟΛΌΓΗΣΗ ΣΧΗΜΆΤΩΝ ΚΩΔΙΚΟΠΟΊΗΣΗΣ - ΜΕ ΠΡΌΘΕΜΑ ΚΑΙ ΜΕ - ΔΙΆΣΤΗΜΑ - ΠΕΡΊΠΤΩΣΗ ΤΩΝ ΔΈΝΤΡΩΝ 91

4.1 ΧΡΟΝΙΚΗ ΔΙΑΡΚΕΙΑ ΚΑΤΑΣΚΕΥΗΣ ΚΩΔΙΚΩΝ 92
ODP RDF SCHEMAS 94
4.2 ΣΧΕΣΙΑΚΗ ΑΝΑΠΑΡΑΣΤΑΣΗ ΚΑΙ ΜΕΓΕΘΟΣ ΑΠΟΘΗΚΕΥΣΗΣ 95
4.3 ΔΕΙΚΤΕΣ (INDICES) 98
4.4 ΧΩΡΟΣ ΚΑΙ ΧΡΟΝΟΣ ΑΠΟΘΗΚΕΥΣΗΣ 104
4.5 SQL ΕΡΩΤΗΣΕΙΣ 107
4.6 ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΕΠΕΡΩΤΗΣΕΩΝ 118
4.7 ΚΑΘΟΡΙΣΜΟΣ ΠΛΗΘΟΥΣ ΕΝΤΑΜΙΕΥΤΩΝ (BUFFERS) 125
4.8 ΣΥΜΠΕΡΑΣΜΑΤΑ 127

ΚΕΦΑΛΑΙΟ 5 129

ΑΞΙΟΛΌΓΗΣΗ ΣΧΗΜΆΤΩΝ ΚΩΔΙΚΟΠΟΊΗΣΗΣ - ΜΕ ΠΡΌΘΕΜΑ ΚΑΙ ΜΕ ΔΙΆΣΤΗΜΑ - ΠΕΡΊΠΤΩΣΗ ΤΩΝ DAGS 129

5.1 ΔΙΑΔΙΚΑΣΙΑ ΚΑΤΑΣΚΕΥΗΣ DAGS 130
5.2 ΧΡΟΝΙΚΗ ΔΙΑΡΚΕΙΑ ΚΑΤΑΣΚΕΥΗΣ ΚΩΔΙΚΩΝ 132
5.3 ΣΧΕΣΙΑΚΗ ΑΝΑΠΑΡΑΣΤΑΣΗ ΚΑΙ ΜΕΓΕΘΟΣ ΑΠΟΘΗΚΕΥΣΗΣ 134
5.4 ΧΩΡΟΣ ΚΑΙ ΧΡΟΝΟΣ ΑΠΟΘΗΚΕΥΣΗΣ 136
5.5 SQL ΕΡΩΤΗΣΕΙΣ 143
5.6 ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ 165
5.7 ΣΥΜΠΕΡΑΣΜΑΤΑ 176

ΚΕΦΑΛΑΙΟ 6 179

ΕΠΙΛΟΓΟΣ 179

6.1 Μελλοντικη Εργασια.....180

ΒΙΒΛΙΟΓΡΑΦΙΑ..... 183

ΠΑΡΑΡΤΗΜΑ.....i

ΠΛΑΝΑ ΕΚΤΕΛΕΣΗΣ ΒΑΣΙΚΩΝ ΕΠΕΡΩΤΗΣΕΩΝ - ΠΕΡΙΠΤΩΣΗ ΔΕΝΤΡΩΝ ΚΑΙ DAGs.....i

ΛΙΣΤΑ ΣΧΗΜΑΤΩΝ

Σχημα 1: Κατάλογος Open Directory (Dmoz) 25
Σχημα 2: RDF Γράφος 28
Σχημα 3: Παράδειγμα RDF γράφου (με τιμή ιδιότητα ένα Literal) 30
Σχημα 4: Σύνθετο Παράδειγμα RDF γράφου (με τιμή ιδιότητα έναν πορο) 31
Σχημα 5: Ιεραρχία RDF/S κλάσεων 33
Σχημα 6 : Παράδειγμα της ιεραρχιας του RDF καταλόγου της πυλης του Open Directory 37
Σχημα 7 : α) Κατανομή Κλάσεων/ Πλήθος Απογόνων, β) Κατανομή Κλάσεων/ Πλήθος Προγόνων 40
Σχημα 8 : Κατανομή θεμάτων (topics) κατά βάθος για τι ιεραρχιες α) Regional, β) Reference και c) Society 41
Σχημα 9: Αναλυτική Κατανομή των κλάσεων / επιπεδο βάθους 42
Σχημα 10 : Παράδειγμα Βασικών Σχεσεων Διασυνδεσης στην RDF/S ιεραρχία MetaNet 46
ΣΣχημα 11: Κλασσικός Δυαδικός Πίνακας – Αλγόριθμος Wirth 54
ΣΣχημα 12: Αλγόριθμος Hasse – Κωδικοποίηση Απλού Μεταβατικού Κλεισίματος 56
Σχημα 13: Αλγόριθμος Κωδικοποίησης Ait-Kaci 58
Σχημα 14: Κωδικοποίηση Αλγορίθμου Caseau 59
Σχημα 15: Near Optimal Hierarchical Encoding (NHE) 63
Σχημα 16: Κωδικοποίηση Αλγορίθμου Dewey 65
Σχημα 17: Κωδικοποίηση Αλγορίθμου Dietz 70
Σχημα 18: Συσχέτιση minimum post τιμής (u'') και μέγιστης pre τιμής (u') δεδομένης ταξινόμησης από το u 72
Σχημα 19 : Κωδικοποίηση Αλγορίθμου Zhang 74
Σχημα 20: Κωδικοποίηση Αλγορίθμου Li & Moon 75
Σχημα 21: Κωδικοποίηση Αλγορίθμου Agrawal, Borgida, Jagadish–Περίπτωση Δέντρου 77
Σχημα 22: Κωδικοποίηση Αλγορίθμου Agrawal, Borgida, Jagadish–Περίπτωση Πλέγματος 78
Σχημα 23 : Κωδικοποίηση Αλγορίθμου Schubert 84
Σχημα 24: Παράδειγμα Γράφου DAG προς κωδικοποίηση 87
Σχημα 25: Συμπιεση Κωδικων Γράφων : α) Agrawal et Al β) Li & Moon c) Dewey 87
Σχημα 26 : Αλγόριθμος του Agrawal et al – σε μορφή ψευδοκώδικα 93
Σχημα 27 : Αλγόριθμος του Unicode Dewey – σε μορφή ψευδοκωδικα 94
Σχήμα 28: Διάταξη τυπικού κόμβου του B+-tree 100
Σχήμα 29 : Λειτουργία ενός B+-tree δείκτη 101
Σχημα 30 : Παράδειγμα α) ισορροπημένου δέντρου β) μη-ισορροπημένου δέντρου 102
Σχημα 31 : Clustered Index 104
Σχημα 32 : Unclustered Index 104
Σχημα 33 : Μέγεθος Βάσης Αποθήκευσης και Δεικτών Agrawal et al και Unicode Dewey 106
Σχημα 34 : Χρόνος Κατασκευής των δέντρων των ιεραρχιών του ODP καταλόγου 107
Σχημα 35 : Ερωτηση Απογόνων στο PInterval 109
Σχημα 36 : Ερωτηση Απογόνων στο UDewey 109
Σχημα 37 : Ερωτηση "Φύλλων"- Κόμβων στο Unicode Dewey (βέλτιστη μορφή) 112
Σχημα 38 : Ερωτηση Precedings 113
Σχημα 39 : Ερωτηση Followings 114
Σχημα 40 : Ερωτηση Siblings 115
Σχημα 41 : Αλγόριθμος Ενσωματωμένης Συνάρτησης nca στον Unicode Dewey 116
Σχημα 42 : Ερωτηση του ελάχιστου κοινού προγόνου δυο κόμβων στο PInterval 117
Σχημα 43 : Ερωτηση του ελάχιστου κοινού προγόνου δυο κόμβων στο Uprefix 117
Σχημα 44 : Κατανομή των Κόμβων Επερωτήσεων στην ιεραρχία του ODP καταλόγου 121
Σχημα 45 : Προτεινομενη Κατανομή Κόμβων ODP Καταλόγου ως προς το πλήθος των προγόνων τους 131
Σχημα 46 : Χαρακτηρισμός Προστιθέμενων non-spanning-tree ακμών 132

Σχημα 47: Παράδειγμα Γράφου DAG προς κωδικοποίηση 136
Σχημα 48 : Παράδειγμα σχηματισμού των πινάκων DPInteval α) με συμπιεση και β) χωρίς συμπιεση για τον γράφο του σχήματος 47 136
Σχημα 49: Πλήθος Propagations ανά βήμα προσθήκης (5%) non-spanning tree ακμων 138
Σχημα 50 : Πλήθος Συμπιεσεων ανά βήμα προσθήκης (5%) non-spanning tree ακμων 138
Σχημα 51 : Κατανομή Πλήθους Propagations σχήματος DPInteval ως προς τις κατηγορίες των non-spanning tree ακμων ανά βήμα προσθήκης (5%) 139
Σχημα 52 : Κατανομή Πλήθους Propagations σχήματος DUPrefix ως προς τις κατηγορίες των non-spanning tree ακμων ανά βήμα προσθήκης (5%) 140
Σχημα 53 : Παράδειγμα εκτελεσης redundant computations με την προσθήκη της non-tree ακμής N_2 N_1 (περίπτωση μη ύπαρξη της δυνατοτητας συμπιεσης των πεδίων –ετικετων) 148
Σχημα 54 : Πίνακες α) PInterval (μόνο spanning-tree), β) DPInterval (με συμπιεση ετικετων) και γ) DPInterval (χωρίς συμπιεση ετικετων) για το γράφο του σχήματος 24. 149
Σχημα 55 : Αλγόριθμος Dancestors για το σχημα Unicode Dewey – με τη μορφή ψευδοκωδικα 155
Σχημα 56 : Παράδειγμα "Φύλλων"- κόμβων του κόμβου u, που έστω συνδέεται με μια non-tree ακμή με τον κόμβο u' 157
Σχημα 57 : Παράδειγμα γράφου με την ύπαρξη πολλαπλων nca's 161
Σχημα 58 : Αλγόριθμος NCA για το σχήμα Unicode Dewey – με την μορφή ψευδοκωδικα 164

ΛΙΣΤΑ ΠΙΝΑΚΩΝ

ΠΙΝΑΚΑΣ 1: ΣΤΑΤΙΣΤΙΚΆ ΤΩΝ ΘΕΜΑΤΙΚΩΝ ΙΕΡΑΡΧΙΩΝ ΤΟΥ ODP ΚΑΤΑΛΌΓΟΥ 39
ΠΙΝΑΚΑΣ 2: ΠΟΛΥΠΛΟΚΌΤΗΤΑ ΜΕΓΕΘΩΝ ΔΗΜΙΟΥΡΓΊΑΣ ΕΤΙΚΕΤΩΝ ΣΧΗΜΆΤΩΝ ΚΩΔΙΚΟΠΟΊΗΣΗΣ BIT-VECTOR 64
ΠΙΝΑΚΑΣ 3: ΣΥΝΘΉΚΕΣ ΕΠΕΡΩΤΗΣΕΩΝ ΣΧΗΜΆΤΩΝ ΚΩΔΙΚΟΠΟΊΗΣΗΣ DEWEY 66
ΠΙΝΑΚΑΣ 4: ΣΥΝΘΉΚΕΣ ΕΠΕΡΩΤΗΣΕΩΝ ΣΧΗΜΆΤΩΝ ΚΩΔΙΚΟΠΟΊΗΣΗΣ DIETZ 71
ΠΙΝΑΚΑΣ 5: ΣΥΝΘΉΚΕΣ ΕΠΕΡΩΤΗΣΕΩΝ ΣΧΗΜΆΤΩΝ ΚΩΔΙΚΟΠΟΊΗΣΗΣ LI & MOON 75
ΠΙΝΑΚΑΣ 6: ΣΥΝΘΉΚΕΣ ΕΠΕΡΩΤΗΣΕΩΝ ΣΧΉΜΑΤΟΣ ΚΩΔΙΚΟΠΟΊΗΣΗΣ AGRAWAL ET AL 77
ΠΙΝΑΚΑΣ 7 : ΧΡΌΝΟΣ ΚΑΤΑΣΚΕΥΉΣ ΚΩΔΙΚΩΝ (ΣΕ SECONDS) 94
ΠΙΝΑΚΑΣ 8 : ΧΡΌΝΟΙ ΕΚΤΈΛΕΣΗΣ ΤΩΝ ΒΑΣΙΚΩΝ ΕΠΕΡΩΤΗΣΕΩΝ ΣΤΟ ΔΈΝΤΡΟ ΥΠΟΚΛΆΣΕΩΝ ΤΟΥ ODP ΚΑΤΑΛΌΓΟΥ 121
ΠΙΝΑΚΑΣ 9 : ΕΠΕΡΩΤΗΣΗ ΑΠΟΓΌΝΩΝ – DESCENDANTS 126
ΠΙΝΑΚΑΣ 10 : ΕΠΕΡΩΤΗΣΗ PRECEDINGS 126
ΠΙΝΑΚΑΣ 11 : ΕΠΕΡΩΤΗΣΗ FOLLOWINGS 127
ΠΙΝΑΚΑΣ 12 : ΧΡΌΝΟΣ ΚΑΤΑΣΚΕΥΉΣ ΚΩΔΙΚΩΝ (ΣΕ SECONDS) 133
ΠΙΝΑΚΑΣ 13 : ΦΟΡΜΑΛΙΣΜΌΣ ΒΑΣΙΚΩΝ ΕΠΕΡΩΤΗΣΕΩΝ ΓΙΑ DAGS ΣΤΑ ΣΧΉΜΑΤΑ ΚΩΔΙΚΟΠΟΊΗΣΗΣ Α) PINTERVAL ΚΑΙ Β) UPREFIX 144
ΠΙΝΑΚΑΣ 14 : ΧΡΌΝΟΙ ΕΚΤΈΛΕΣΗΣ (ΣΕ SECONDS) ΤΩΝ ΒΑΣΙΚΩΝ ΕΠΕΡΩΤΗΣΕΩΝ ΣΤΟ DAG ΥΠΟΚΛΆΣΕΩΝ ΤΟΥ ODP ΚΑΤΑΛΌΓΟΥ (ΜΕ ΣΥΜΠΊΕΣΗ ΤΩΝ ΕΤΙΚΕΤΩΝ) 167
ΠΙΝΑΚΑΣ 15 : ΧΡΌΝΟΙ ΕΚΤΈΛΕΣΗΣ (ΣΕ SECONDS)ΤΩΝ ΒΑΣΙΚΩΝ ΕΠΕΡΩΤΗΣΕΩΝ ΣΤΟ DAG ΥΠΟΚΛΆΣΕΩΝ ΤΟΥ ODP ΚΑΤΑΛΌΓΟΥ (ΧΩΡΊΣ ΣΥΜΠΊΕΣΗ ΤΩΝ ΕΤΙΚΕΤΩΝ) 174

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Στη σύγχρονη εποχή υπάρχουν διάφορες εφαρμογές του Σημασιολογικού Διαδικτύου (*Semantic Web*) [BLHL01], όπως τέτοια χαρακτηριστικά παραδείγματα είναι το ηλεκτρονικό εμπόριο (*e-commerce*), ηλεκτρονική μάθηση (*e-learning*), η ηλεκτρονική επιστήμη (*e- science*), οι διαδικτυακές πύλες (*portals*) αλλά και τόποι (*sites*), στα οποία απαιτούνται πλέον εξαιρετικά εξελιγμένα εργαλεία για την διαχείριση των μεταδεδομένων. Με τον όρο μεταδεδομένα θεωρούμε το σύνολο διάφορων περιγραφών γύρω από την σημασία , την χρηστικότητα, την προσβασιμότητα και την ποιότητα των πληροφοριακών πηγών (π.χ. δεδομένα, έγγραφα, υπηρεσίες), οι οποίες είναι διαθέσιμες σε διάφορα επιχειρησιακά ενδοδίκτυα, καθώς και στον Παγκόσμιο Ιστό.

Για την σημασιολογική περιγραφή πόρων πληροφορίας από διάφορες κοινότητες χρηστών, χρησιμοποιούνται ευρέως σήμερα διάφορα δομημένα λεξιλόγια (π.χ. θησαυροί), θεματικές ταξινομήσεις, ή ακόμα και σύνθετα εννοιολογικά σχήματα (πχ οντολογίες). Συγκεκριμένα, υπάρχει ένα ευρύτατο φάσμα από τέτοια υφής περιγραφικά σχήματα που είναι διαθέσιμα, και τα οποία αντιπροσωπεύουν σήμερα αναμφισβήτητα ένα σημαντικό μέρος επί του συνολικού πλήθους των ιεραρχικών δεδομένων που υπάρχουν στον Παγκόσμιο Ιστό [MACP02].

1.1 Διαδικτυακές Πύλες

Με τον όρο πύλη εννοούμε μια έννοια η οποία δεν μπορεί να οριστεί απόλυτα, αλλά περιγραφικά με διακριτό τρόπο ως προς την οργάνωση και τις υπηρεσίες που προσφέρει. Γενικά ως πύλη θεωρούμε ένα σημείο ή εναλλακτικά μια τοποθεσία, όπου συγκεντρώνει και προσδίδει με ολοκληρωμένο τρόπο πληροφορία, επιμέρους εφαρμογές και διακριτές

προσωπικότητες, αλλά και κυρίως τα προσφέρει με ευκολία, περιεκτικότητα και με την δέουσα χρησιμότητα που τους αρμόζει [S99]. Το πλέον απτό και χαρακτηριστικό γνώρισμα των πυλών είναι η ίδια η οργάνωσή τους με την μορφή ενός καταλόγου περιεχομένων. Συνήθως, ο κατάλογος περιεχομένων έχει δενδρική μορφή, γεγονός που της επιτρέπει την όποια αναζήτηση πληροφορίας μέσω της κατάλληλης πλοήγησης και διερεύνησης του καταλόγου αυτού. Βέβαια κι η παρουσίαση της πληροφορίας, που προσφέρεται από διάφορους προμηθευτές πληροφορίας, και που πραγματοποιεί εν γένει η κάθε πύλη, δεν μπορεί να θεωρηθεί πως καθιστά την πύλη με αυτόν τρόπο την ίδια ως επιμέρους αυτόνομο προμηθευτή.

Η λειτουργικότητα που διακρίνει μια πύλη έγκειται, όπως και σε διάφορες μηχανές αναζήτησης, η δυνατότητα άσκησης επερωτήσεων μέσω της κατάλληλης απόδοσης και συνδυασμού λέξεων – κλειδιών μεταξύ τους. Επιπρόσθετα, μια άλλη σημαίνουσα δυνατότητα των πυλών είναι ότι μπορούν να προσαρμόζονται τόσο ως προς την μορφή παρουσίασης αλλά και του περιεχομένου στις εκάστοτε καταγραφείσες επιλογές και προτιμήσεις του χρήστη (*personalization*). Τέλος, πέραν της δυνατότητας πληροφόρησης μια πύλη μπορεί να παρέχει και δευτερεύουσας σημασίας υπηρεσίες όπως είναι η παροχή ηλεκτρονικών διευθύνσεων, η δυνατότητα επικοινωνίας με άλλους χρήστες κάτω από ένα κοινό πλαίσιο επαφής σε πραγματικό χρόνο κλπ.

Αναφορικά με την *ποικιλία* του *περιεχομένου* τους καθώς και το *εύρος* της *κοινότητας* που απευθύνονται οι πύλες μπορούν να διαχωριστούν και να ενταχθούν σε μία από τις κάτωθι κατηγορίες : [Delphi99]

1. **Εκδοτικές** (**Publishing)** πύλες, οι οποίες απευθύνονται σε σχετικά μεγάλες κοινότητες χρηστών, με διακριτά χαρακτηριστικά και με ποικίλα ενδιαφέροντα. Η ουσιαστική δυνατότητα που παρέχουν είναι, όπως και στις περισσότερες μηχανές αναζήτησης, αυτή της επερώτησης με επιμέρους λέξεις - κλειδιά. Στην κατηγορία αυτή ανήκουν όλοι οι μεγάλοι προμηθευτές πληροφορίας π.χ., *CNN, ABC News*.
2. **Εμπορικές** (**Commercial)** πύλες, που αποτελούν την πλέον διάσημη από άποψη αποδοχής κοινού κατηγορία πυλών, καθώς απευθύνονται σε πολλαπλές κοινότητες, όπως και οι *Εκδοτικές* που προαναφέραμε. Όμως ξεχωρίζουν από την μοναδική τους δυνατότητα για προσαρμογή του περιεχομένου τους στα ενδιαφέροντα και τις προτιμήσεις του χρήστη. Επίσης παρέχουν και πλήθος από

δευτερεύοντες υπηρεσίες που θέλγουν τους απλούς χρήστες, και που δεν διαθέτουν κάποιο άλλο κοινό χαρακτηριστικό τους γνώρισμα που να τους προσδιορίζει κατά απόλυτο τρόπο. Συνεπώς, οι πύλες αυτές είναι κι οι πιο δημοφιλείς. Παραδείγματα αυτών των πυλών είναι οι Yahoo, Lycos κλπ

3. **Προσωπικές** (**Personal)** πύλες , οι οποίες έχουν πιο περιορισμένο περιεχόμενο καθώς προσαρμόζονται στις προτιμήσεις του κάθε χρήστη. Για παράδειγμα μια λειτουργία που παρέχουν είναι η δυνατότητα ο κάθε χρήστης με την εισαγωγή του σε μια τέτοια πύλη, να παρεμβαίνει απόλυτα στην εξωτερική μορφή της κάθε περιοχής της κάθε πύλης και να συγκεντρώνει υπηρεσίες από διαφορετικούς φορείς υπηρεσιών στο διαδίκτυο.
4. ***Τέλος είναι οι*** **Εταιρικές** (**Corporate)** πύλες [FA99], που έχουν ένα εξαιρετικά πλούσιο περιεχόμενο όμως για ένα πολύ περιορισμένο αντικείμενο γνώσης σε ένα επίσης πολύ συγκεκριμένο σύνολο από άτομα. Το περιεχόμενο τους είναι ευρύτερο από το περιεχόμενο των πυλών Commercial εφόσον υπάρχει μεγάλη ποικιλία και πολυπλοκότητα στην πληροφορία που διαχειρίζονται. Σκοπός των πυλών αυτής της κατηγορίας είναι να παρέχουν την πρόσβαση, την αλληλεπίδραση και την διανομή γνώσης που διαπνέει και σχετίζεται με το αντικείμενο που χαρακτηρίζει μια εταιρεία. Γενικά, η κατηγορία αυτή πυλών είναι ένα σύνολο κατάλληλα συνεργαζόμενων εφαρμογών που σχετίζονται με καθήκοντα όπως είναι η συγκέντρωση, η διαχείριση, η ανάλυση και η διανομή τελικώς της πληροφορίας μέσα και έξω από την εταιρεία.

Παράλληλα οι πύλες κατηγοριοποιούνται σε πύλες *δεδομένων* (*data*), *πληροφορίας* (*information*), *συνεργατικές* (*collaborative*) και *γνώσης* (*knowledge*) (βλέπε εικόνα 1.2) [W99], [T99]. Οι πύλες δεδομένων διαχειρίζονται κυρίως δομημένα δεδομένα τα οποία είναι αποθηκευμένα σε βάσεις δεδομένων. Οι πύλες πληροφορίας διαχειρίζονται αδόμητα δεδομένα όπως ηλεκτρονικά μηνύματα, απλά κείμενα, εικόνες κ.λ.π. Οι συνεργατικές πύλες προσφέρουν δυνατότητες αλληλεπίδρασης μεταξύ των χρηστών της κοινότητας. Τέλος, οι πύλες γνώσης παρέχουν τις δυνατότητες που προσφέρουν οι παραπάνω κατηγορίες πυλών. Δεν θα πρέπει να αναφέρουμε πως όλες οι

προαναφερθέντες κατηγορίες πυλών περιέχουν ένα κατάλογο γνώσης (*knowledge catalog*).

Ο κατάλογος θεμάτων κάθε διαδικτυακής πύλης, ο οποίος έχει εξ αρχής κατασκευαστεί και δομηθεί με βάση μία ή περισσότερες θεματικές ιεραρχίες (*schemas*), δημοσιοποιείται στην πραγματικότητα στο διαδίκτυο σαν ένα σύνολο από στατικά δομημένες και διασυνδεδεμένες HTML ιστοσελίδες[1] : κάθε επιμέρους σελίδα περιέχει όλους τους πληροφοριακούς αυτούς πόρους (*αντικείμενα - objects*) ορισμένους κάτω από ένα συγκεκριμένο θέμα - topic(*class*), όπως και από διάφορα είδη σχέσεων μεταξύ θεμάτων. Συγκεκριμένα, η σχέση *subtopic* αναπαριστά την σχέση υπόστασης (σημασιολογικά αναφέρεται και ως *isA*) μεταξύ των κλάσεων ενός σχήματος διαδικτυακής πύλης. Στην περίπτωση αυτή το οριζόμενο σχήμα μιας διαδικτυακής πύλης διαμορφώνει εν τέλει είτε ένα δέντρο (στην περίπτωση που έχουμε μονές *isA* συνδέσεις), είτε εναλλακτικά ενός *κατευθυνόμενου άκυκλου γράφου - DAG -* (στην περίπτωση που έχουμε πολλαπλές *isA* συνδέσεις) κλάσεων, στην καλύτερη περίπτωση των ημιπλεγμάτων (*semi-lattices*), το οποίο χρησιμοποιείται για την υποβοήθηση της επιθυμητής πλοήγησης του τελικού χρήστη. Αναλυτικά, μέσω ενός συγκεκριμένου θέματος μπορεί ο οιοσδήποτε χρήστης εν συνεχεία να οδηγηθεί μέσω τη πλοήγησής του σε επιμέρους υποθέματά του (*subtopics*) και τελικώς να προσεγγίσει τις πηγές αυτές, οι οποίες είναι άμεσα ορισμένες κάτω από αυτές.

Αναφορικά, θεωρούμε ότι ο κατάλογος γνώσης είναι μια 'αποθήκη' μεταδεδομένων που υποστηρίζει την οργάνωση του περιεχομένου με πολλούς διαφορετικούς τρόπους, παρέχοντας την δυνατότητα ταξινόμησης σε πολλαπλές ιεραρχίες κλάσεων οντοτήτων. Συνεπώς, με βάση όλα τα προαναφερθέντα η διαχείριση αυτού του καταλόγου γνώσης καθίσταται ένα σημαντικό σημείο στη διαμόρφωση και περαιτέρω αναπροσαρμογή μιας πύλης. Δεδομένου ότι το πλήθος των επιδιωκόμενων λειτουργιών είναι αρκετά μεγάλο καλείται η πύλη να διαχειρίζεται μια εξαιρετικά σύνθετη δομή. Στο σημείο αυτό υπάρχει διαχωρισμός των πυλών ως προς το μοντέλο δεδομένων στο οποίο βασίζεται ο κατάλογος μιας πύλης, όπου θα πρέπει πλέον στις μέρες μας να μπορεί να διαχειριστεί τις πολλές συσχετίσεις μεταξύ αντικειμένων

[1] Να παρατηρηθεί ότι το RDF μπορεί να χρησιμοποιηθεί σαν ένας κατ' επέκταση εξαγώγιμος τρόπος ακατέργαστης ενημέρωσης του καταλόγου μιας πύλης

διαφορετικών κατηγοριών, με ενδεχόμενη εμφάνιση πολλαπλών ταξινομήσεων, την ύπαρξη ημιδομημένων περιγραφών κλπ. Ένα βέλτιστο τέτοιο μοντέλο αναπαράστασης των δεδομένων είναι και το RDF/S, το οποίο και θα μελετήσουμε αναλυτικά στη συνέχεια.

1.2 Πλαίσιο Περιγραφής Πόρων (Resource Description Framework – RDF)

Σε αυτήν την περιρρέουσα κατάσταση, μια βέλτιστη πρόταση για την μοντελοποίηση του εκάστοτε διαχειριζόμενου καταλόγου γνώσης σύμφωνα με τα πρότυπα του RDF/S (W3C) για την διευκόλυνση κι ανταλλαγή μεταδεδομένων σαν απλά δεδομένα που ανταλλάσσονται στο διαδίκτυο είναι και το *Πλαίσιο Περιγραφής Πόρων* (*Resource Description Framework – RDF*). Μάλιστα το *RDF* συνεχώς με καθ' όλα αυξητικό τρόπο αποκτά έγκριση δημιουργίας μεταδεδομένων και ανταλλαγή τους παρέχοντας με αυτόν τρόπο τα εξής χαρακτηριστικά γνωρίσματα :

α) μία *Δεδομένη Γλώσσα Αναπαράστασης* (*Standard Representation Language*) για το σύνολο περιγραφών που στηρίζονται σε κατευθυνόμενους με ετικέτα γράφους (*directed labeled graphs*)

β) μία *Γλώσσα Περιγραφής Σχημάτων* (*Schema Definition Language - RDFS*) για την μοντελοποίηση θησαυρών χρηστών ή ταξινομήσεων από επιμέρους ιεραρχίες κλάσεων /ιδιοτήτων (class /property) με συνυφασμένη την έννοια της σχέσης *υπόστασης* - *subsumption*- (τέτοια χαρακτηριστικά .παραδείγματα είναι τα δέντρα και οι άκυκλοι κατευθυνόμενοι γράφοι – *DAGs*)

γ) ένα XML συντακτικό εξίσου για περιγραφές σχημάτων κι πόρων. Αναλυτικά, ολοένα και περισσότερες διαδικτυακές πύλες (π.χ. *Netscape Dmoz* ή *Chefmoz*, *MusicBrain*, *CNET*, *XMLTree*[1] κλπ), εξάγουν με την μορφή *RDF/S* τις θεματικές κατηγορίες ή τα θέματα εν γένει τα οποία αναπτύσσουν για να περιγράψουν διάφορους πόρους σε αλληλοπροσδιοριζόμενο κοινό - ακροατήριο.

Το *RDF* είναι μια καινοτόμα πρόταση μοντελοποίησης των δεδομένων, οργανώνοντάς τα με βάση την σημασιολογία τους. Μέσα στο πλαίσιο περιγραφής που παρέχει οιοδήποτε στοιχείο (είτε αυτό είναι ένα επιμέρους διαδικτυακός τόπος, είτε ένα *XML* έγγραφο, είτε μια *HTML* σελίδα κλπ) θεωρείται ως ένας ανεξάρτητος πόρος που αυτοπροσδιορίζεται από ένα μοναδικό *URI*. Οι περιγραφές που σχετίζονται με τους πόρους αναπαρίστανται μέσω αντιστοίχως ιδιοτήτων που μπορούν να αποδοθούν στους πόρους. Συγκεκριμένα, το *RDF* ορίζει ένα μοντέλο γράφων, όπου οι κόμβοι τους αποτελούνται από τους μεμονωμένους πόρους και οι ακμές αποτελούνται από τις ιδιότητές τους αντιστοίχως. Με το *RDF* επιτρέπεται ο ορισμός κατάλληλων σχημάτων που επιτρέπουν την ερμηνεία των μεταδεδομένων και την επαναχρησιμοποίησή τους μέσα στα πλαίσια διαφορετικών κοινοτήτων. Τα *RDF* σχήματα ορίζουν τις δυνατές ετικέτες που μπορούν να αποδοθούν στους κόμβους και τις ακμές ενός γράφου περιγραφών *RDF*. Με βάση το πρότυπο του *RDF* οι οριζόμενες ιδιότητες είναι καθ' όλα προαιρετικές και πλειότιμες, κι επομένως οι ετερογενείς περιγραφές που υπάρχουν για τους πόρους της πύλης μπορούν να αναπαρασταθούν με την μορφή *RDF* μεταδεδομένων. Επομένως, συνολικά μπορούμε να συμπεράνουμε πως με το *RDF* μπορούμε να αναπαραστήσουμε τις περιγραφές που αποδίδονται σε κάθε είδους πληροφοριακό πόρο μιας κοινότητας και μάλιστα να αποδοθούν σε αυτούς διαφορετικοί τύποι μεταδεδομένων.

Στην εργασία αυτή θα σχετιστούμε με το σύνολο των υπαρχόντων κατηγοριών σχημάτων κωδικοποίησης που απευθύνονται σε αυτού του είδους τα ιεραρχικά δεδομένα που προέρχονται από τις διαδικτυακές πύλες που προείπαμε. Θα επικεντρωθούμε στην προσπάθεια βελτιστοποίησης διάφορων θεμελιωδών επερωτήσεων που έχουν καθ' όλα σημαντική, λειτουργική σημασία στους καταλόγους των διαδικτυακών πυλών. Επίσης, λόγω του μεγάλου όγκου δεδομένων που έχουμε να διαχειριστούμε από μεταδεδομένα του καταλόγου γνώσης που έχουμε να διαχειριστούμε θα χρησιμοποιήσουμε συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων.

1.3 Αντικείμενο Μελέτης της Εργασίας

Υπάρχει ένα σύνολο από επιμέρους δηλωτικών γλωσσών επερωτήσεων, οι οποίες χρησιμοποιούνται για *RDF/S* σχήματα, και οι οποίες μπορούν να υποστηρίξουν εν συνεχεία τη δυναμική πλοήγηση μέσω δυναμικώς ανανεούμενων διεπαφών χρήσης, αλλά και την προσωποποίηση, κάτι που επιτυγχάνεται τόσο μέσω των χρησιμοποιούμενων σχημάτων των διαδικτυακών πυλών, όσο και μέσω των περιγραφών των ίδιων των πηγών [AKCPT01, KACP02, STC02].

Η εν λόγω εργασία επικεντρώνεται στο θέματα της βελτιστοποίησης των προαναφερθέντων αυτών επερωτήσεων, με απώτερο κριτήριο την αποφυγή του εξαιρετικά μεγάλου σε κόστος υπολογισμό των μεταβατικών κλεισιμάτων, ιδιαιτέρως στην περίπτωση εκτεταμένων ιεραρχιών κλάσεων. Ένα τέτοιο αντιπροσωπευτικό παράδειγμα εκτεταμένης ιεραρχίας κλάσεων είναι αναμφισβήτητα και ο κατάλογος του *Open Directory Portal / Dmoz*, το οποίο περιλαμβάνει περίπου ανά πλήθος από 200M θέματα, τα οποία βρίσκονται εξαγόμενα με την μορφή *RDF /XML* αρχείων και τον οποίο θα μελετήσουμε εν συνεχεία με μεγαλύτερη λεπτομέρεια.

Αναλυτικά, θα δώσουμε ιδιαίτερη έμφαση στην υιοθέτηση εναλλακτικών σχημάτων κωδικοποίησης, τα οποία θα μπορέσουν επιμέρους με τα διακριτά χαρακτηριστικά τους γνωρίσματα να κωδικοποιήσουν κατάλληλα ιεραρχίες από κλάσεις (ή εναλλακτικά ιδιότητες – *properties*). Ως σχήμα κωδικοποίησης προσδιορίζουμε ένα τρόπο αναπαράστασης της διασύνδεσης των κόμβων μιας ιεραρχίας σε γραμμική διάσταση, αποσκοπώντας στην ικανοποιητική διασφάλιση χώρου και στη λειτουργική απεικόνιση και διαχείριση των συσχετίσεων μεταξύ τους. Η καταλληλότητα που πρέπει να επιδείξουν τα σχήματα στην παραγωγή των κωδικών τους, είναι ώστε με βάση αυτούς να μπορούμε με βάση αυτούς να υπολογίσουμε - μέσω σχετικών επερωτήσεων - τους "απογόνους"/ "προγόνους", συνεχόμενους / ομοεπίπεδους (adjacent /sibling), καθώς και τον πιο κοντινό, κοινό πρόγονο (*nearest common ancestor - nca*) δυο κόμβων μιας ιεραρχίας, αποκλειστικά και μόνο με τους κωδικούς αυτούς και χωρίς την όποια άλλη γνώση του υπό εξέταση κάθε φορά σχήματος. Με αυτόν τον τρόπο μπορούμε να ξεκινήσουμε από έναν κόμβο – θέμα σε μία ταξινομημένη ιεραρχία και να υπολογίσουμε όχι μόνο τους άμεσα πατρικούς / θυγατρικούς του, αλλά επίσης και τους κόμβους

"φύλλα" του από το σύνολο των απογόνων του στην κατηγορία αυτή των "φύλλων" κόμβων μπορούμε να ορίσουμε και να εντάξουμε στη συνέχεια τις περισσότερες πηγές πληροφόρησης που χρησιμοποιούνται σήμερα στο διαδίκτυο. Μπορούμε επίσης παράλληλα να ανακαλύψουμε τους ομοεπίπεδους κόμβους ενός κόμβου σε μία ιεραρχία, και με αυτόν τον τρόπο να προσδιορίσουμε το σύνολο άμεσα σχετικών ως προς το περιεχόμενό τους διαδικτυακές πηγές πληροφόρησης. Εδώ θα πρέπει όμως να διασαφηνιστεί ότι επί του παρόντος η αναφερόμενη αυτή δυνατότητα της προχωρημένης πλοήγησης και η ικανότητα της επανακατασκευής του υφιστάμενου αυτού σχήματος αναπαράστασης δεν υποστηρίζεται από κανένα από τις υπάρχοντες διεπιφάνειες χρήσης των διαδικτυακών πυλών.

Εν γένει έχουν κατά καιρούς προταθεί πολλά επιμέρους σχήματα κωδικοποίησης δεδομένων που βρίσκονται αναπαριστάμενα είτε με την μορφή δέντρου ή τη μορφή γράφου για ένα πλήθος εφαρμογών με καθ' όλα διακριτό περιεχόμενο μεταξύ τους, όπως είναι η δρομολόγηση δικτύων [GP03], ο αντικειμενοστραφής προγραμματισμός [D82, DS87, W88, KBL89, C93, KVH97], ή τα συστήματα αναπαράστασης γνώσης [ABJ89, E97], και προσφάτως μηχανές αναζήτησης, οι οποίες βασίζονται στην XML [CSFH01, ZNWLL, MACM01, LM01, SKM02, CKM02, CVZTZ02, TVBS02]. Παρόλο όμως την πληθώρα των υπαρχόντων σχημάτων κωδικοποίησης, είναι ανοικτό ακόμα το πρόβλημα της επιλογής ενός κατάλληλου σχήματος που να μπορεί με ικανοποιητικό τρόπο να υποστηρίξει την λειτουργικότητα που χρειάζεται από τις διαδικτυακές πύλες που χρησιμοποιούνται από τις διάφορες επιμέρους κοινότητες χρηστών (*Community Web Portals*). Συγκεκριμένα, υπάρχει μια πληθώρα επιμέρους θεμάτων τα οποία καθιστούν ή όχι μια κωδικοποίηση πιο βέλτιστη σε σχέση με κάποια ήδη υπάρχουσα, όπως είναι :

- Τα σχήματα των διαδικτυακών πυλών, των οποίων εξάγουμε τα δεδομένα με την μορφή *RDF*, ποικίλουν ως προς την μορφή τους, από απλά δέντρα ιεραρχιών υποκλάσεων έως και πολύπλοκα *DAGs* [MACP02]. Εδώ θα πρέπει να παρατηρήσουμε πως η διάταξη των υποκλάσεων μεταξύ τους με την αναπαράσταση που τους προσφέρει η χρήση του *RDF* (σε αντίθεση τις μηχανές αναζήτησης που χρησιμοποιούν *XML*)καθίσταται λιγότερο σημαντική. Συνεπώς, χρειαζόμαστε σχήματα κωδικοποίησης για δέντρα, τα οποία στη συνέχεια να

μπορούν εύκολα να επεκταθούν και για την περίπτωση των *DAGs*, με το μικρότερο δυνατό επιπλέον κόστος αποθήκευσης και άσκησης των επερωτήσεων.

- Η όποια ενέργεια άσκησης επερώτησης και πλοήγησης σε ένα σχήμα μιας διαδικτυακής πύλης, βασίζεται σε λειτουργίες ανάκτησης μεγάλης κλίμακας, χρησιμοποιώντας πολύπλοκες συνθήκες φιλτραρίσματος, οι οποίες βασίζονται στην διερεύνηση σχέσεων υπαλληλίας (*subsumption*). Αυτή είναι μια ιδιότητα, η οποία διακρίνει απόλυτα τις διαδικτυακές πύλες από άλλες εφαρμογές όπως είναι η δρομολόγηση δικτύων, ο αντικειμενοστραφής προγραμματισμός, και τα συστήματα αναπαράστασης της γνώσης, τα οποία σε κάθε περίπτωση χειρίζονται 2 κόμβους /κλάσεις την φορά. Σε όλα αυτά θα πρέπει να προσθέσουμε πως το επιδιωκόμενο σχήμα κωδικοποίησης που θέλουμε, θα πρέπει να μπορεί να παράγει επιμέρους κωδικούς στις κλάσεις, οι οποίοι να μπορούν να επεξεργαστούν στη συνέχεια από ένα σύστημα διαχείρισης βάσεων δεδομένων, χρησιμοποιώντας συγκεκριμένες επιπλέον δομές διευθυνσιοδότησης (*index structures*), όπως είναι για παράδειγμα τα B^+*-trees* κλπ.

1.4 Οργάνωση και Συνεισφορά της Εργασίας

Μέσα στα πλαίσια της εργασίας αυτής θα ασχοληθούμε ιδιαίτερα με την μελέτη ενός βέλτιστου συμβιβασμού ανάμεσα στις απαιτήσεις χώρου και του χρόνου εκτέλεσης των επερωτήσεων, χρησιμοποιώντας διάφορα σχήματα κωδικοποίησης εξίσου τόσο για την περίπτωση ιεραρχιών με την μορφή δέντρων, όσο και για *DAGs*. Η έρευνα μάς θα δρομολογηθεί πάνω στην επιδίωξη για ικανοποιητική ανάπτυξη επερωτήσεων βασιζόμενων σε κωδικούς, με την χρήση της τρέχουσας, υφιστάμενης τεχνολογίας σε συστήματα διαχείρισης βάσεων δεδομένων, όπως είναι οι *SQL* μηχανές.

Αναλυτικά η συνεισφορά μας μπορεί να προσδιοριστεί στα εξής ακόλουθα σημεία με την σειρά αναφοράς κι ανάλυσής τους:

- Στο κεφάλαιο 2 θα παρουσιάσουμε πως μπορούμε να μοντελοποιήσουμε με την χρήση *RDF/S* σχημάτων, καταλόγους διαδικτυακών πυλών, όπως είναι του *Open Directory Portal* (*ODP*) ο κατάλογος. Επιπλέον, το ενδιαφέρον επικεντρώνεται στην παρουσίαση ενός συνόλου στατιστικών στοιχείων γύρω από τα ακριβή προσδιοριστικά μεγέθη κι εν γένει την υφιστάμενη μορφολογία των ημιπλεγμάτων από κλάσεις που συνθέτουν τον *ODP* κατάλογο. Στην συνέχεια της εργασίας μας θα χρησιμοποιηθεί ο *ODP* κατάλογος σαν πεδίο δοκιμών (*testbed*) για την πειραματική αξιολόγηση των υπαρχόντων σχημάτων κωδικοποίησης που θα μελετήσουμε.

- Στο κεφάλαιο 3 θα προχωρήσουμε σε μια συγκριτική ανάλυση των τριών βασικών οικογενειών σχημάτων κωδικοποίησης του δυαδικού διανύσματος (*bit-vector*), του προθέματος (*prefix*), και του διαστήματος (*interval*), για δεδομένα που μπορούν να αναπαρασταθούν τόσο με την μορφή δέντρου όσο και με την μορφή γράφου. Θα επικεντρωθούμε κυρίως γύρω από την έκφραση του βασικού πυρήνα της παρεχόμενης λειτουργικότητας επερωτήσεων (π.χ. απόγονοι /πρόγονοι /φύλλα, συνεχόμενοι, ομοεπίπεδοι, πιο κοντινός κοινός πρόγονος – *nca*) που είναι απαραίτητοι για την ανάπτυξη των μοντέρνων διεπαφών των διαδικτυακών πυλών.

- Στο κεφάλαιο 4 θα αναφέρουμε την συμπεριφορά απόδοσης δυο απόλυτα αντιπροσωπευτικών σχημάτων κωδικοποίησης, αναφορικά *α)* το σχήμα κωδικοποίησης του *Dewey*, που βασίζεται στην κατηγορία των σχημάτων προθέματος (*prefix-based*) [OCLC], και *β)* το εκτεταμένο σχήμα της απόδοσης των κωδικών μετά ην διάσχιση (*post-order*), που έχει προταθεί από τους *Agrawal, Borgida* και *Jagadish* [ABJ89], και το οποίο ανήκει στην κατηγορία των σχημάτων με την χρήση διαστημάτων (*interval-based*). Η επιχειρούμενη σύγκριση της απόδοσης των δυο αυτών επιμέρους αντιπροσωπευτικών σχημάτων κωδικοποίησης γίνεται ως προς τις απαιτήσεις και των δυο σε αποθηκευτικό χώρο αλλά και σε χρόνο εκτέλεσης πάνω σε ένα συγκεκριμένο οντοκεντρικό- σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων (*ORDBMS*),

όπως είναι κι η *PostgreSQL*. Μέσα στο πλαίσιο αυτό δίνουμε ιδιαίτερη έμφαση στην ικανοποιητική μετάφραση των διαφορετικών τύπων επερωτήσεων πάνω σε δέντρα κλάσεων (τα οποία χαρακτηρίζονται από την ύπαρξη σχέσεων *isA* με μονή πληθικότητα) σε επερωτήσεις *SQL*,

- Στο κεφάλαιο 5 θα προχωρήσουμε σε μία αντίστοιχη σύγκριση της συμπεριφορά απόδοσης των σχημάτων κωδικοποίησης, *Dewey* (*prefix-based*) και *Agrawal, Borgida* και *Jagadish* για την περίπτωση των *DAGs* (δηλαδή με πολλαπλές *isA* συνδέσεις), με στόχο την εκτίμηση του επιπλέον κόστους διαχείρισής τους. Η παραγωγή των επιπρόσθετων *isA* συνδέσεων γίνεται πάνω στα δέντρα κλάσεων του κεφαλαίου 4 με ισοκατανομή τους, με βάση τη ιεραρχική τοποθέτησή τους στην διαμορφούμενη ιεραρχία.

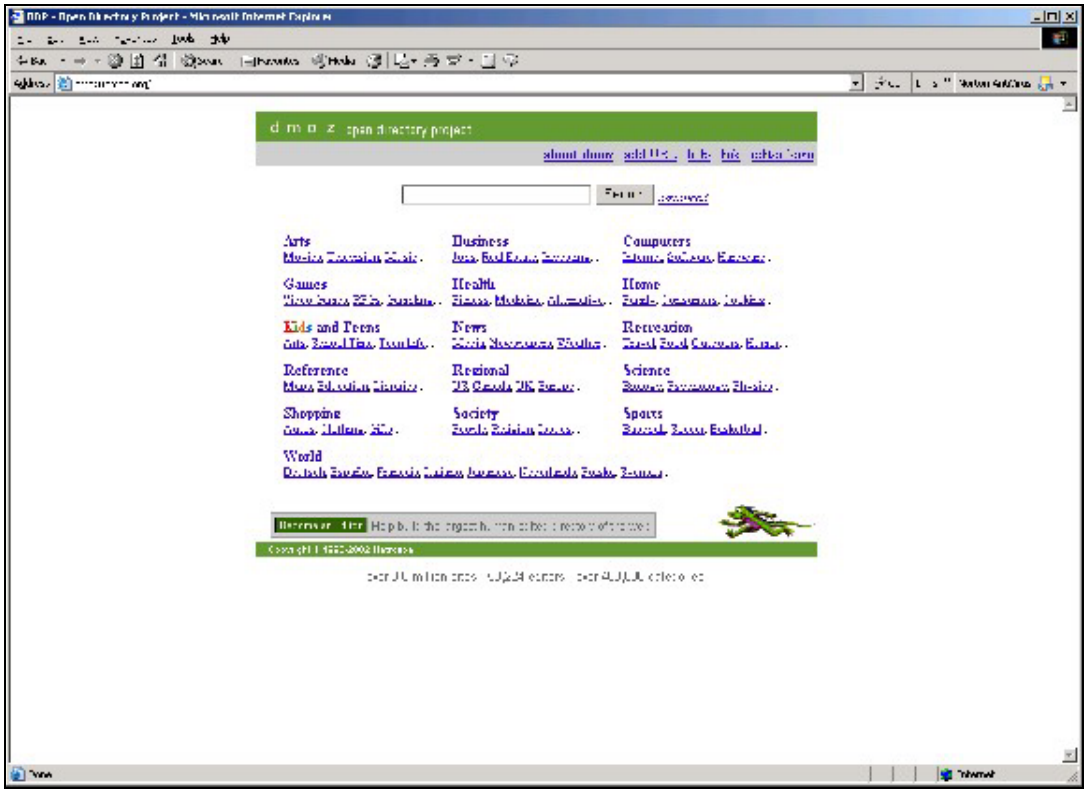
- Ολοκληρώνοντας, στο κεφάλαιο 6 προχωρούμε σε μια γενική ανασκόπηση και αναφέρουμε μια σειρά από τελικά συμπεράσματα. Επίσης παρουσιάζουμε μελλοντικές επεκτάσεις του συστήματος και θέματα και προοπτικές του όλου θέματος που θα πρέπει να μελετηθούν περαιτέρω.

Κεφάλαιο 2

Πύλες Διαδικτύου: Μοντελοποίηση σε RDF/S και Στατιστική Ανάλυση

Ως γνωστόν οι διαδικτυακές πύλες (portals) οργανώνουν σημασιολογικά επιμέρους διακριτές πληροφοριακές πόροι ποικίλου είδους αλλά και κυρίως και περιεχομένου, αποσκοπώντας ενίοτε και σε διαφορετικής πληθικότητας και σύνθεσης ακροατήριο (π.χ. επαγγελματικά, επιχειρηματικά, εμπορικά). Κάθε μία πύλη επιμέρους προσδιορίζεται από έναν κατάλογο που εμπερικλείει όλη την αναγκαία αυτή περιγραφική πληροφορία των πληροφοριακών πόρων που προσφέρει. Όπως μπορούμε να φανταστούμε η δυνατότητα πρόσβασης στους παρεχόμενους πληροφοριακούς πόρους, εξαρτάται άμεσα από την πολυπλοκότητα των παρεχόμενων σημασιολογικών περιγραφών (εναλλακτικά θα μπορούσαν να είναι είτε η χρήση θησαυρών, είτε η χρήση ταξινομήσεων, είτε και η χρήση περισσότερο σύνθετων οντολογιών).

Στην σύνηθες πραγματικότητα, όπως αυτή διαμορφώνεται από τις υφιστάμενες υπάρχουσες διαδικτυακές πύλες (*Web Portals*) (π.χ. *Netscape Dmoz (ChefMoz), MusicBrain, CNET, XMLTree* κλπ), οι παρεχόμενοι πληροφοριακοί πόροι ταξινομούνται κάτω από πραγματικά "μεγάλες" ιεραρχίες, ως προς το πλήθος των θεματικών κατηγοριών ή των μεμονωμένων θεμάτων, τα οποία με εξαιρετική ευκολία μπορούν να αναπαρασταθούν και εν συνεχεία να ανταλλαχθούν μέσω της χρήσης του προτύπου RDF/S. (βλ. Σχήμα 1 παράδειγμα *ODP* καταλόγου).



Σχήμα 1: Κατάλογος Open Directory (Dmoz)

2.1 Το πρότυπο RDF-RDF/S

2.1.1 Εισαγωγή – Χαρακτηριστικά του RDF

Στην ενότητα αυτή θα επιχειρήσουμε να παρουσιάσουμε μια αναλυτική περιγραφή του *RDF* (*Resource Description Framework*) - Πλαισίου Περιγραφής Πόρων. Το *RDF* έχει προταθεί από το *World Wide Web Consortium* (W3C) [LS99] όπως θα δούμε και το *RDF Schema Specification* (*RDF/S*), η οποία είναι μια γλώσσα περιγραφής σχημάτων περιγραφής πληροφοριακών πόρων, που ορίζουν τη σημασιολογία των περιεχομένων των μεταδεδομένων για διάφορες εφαρμογές, που εξ ορισμού περιέχουν από κοινού κλάσεις και ιδιότητες, και σχετίζεται με το RDF όπως θα περιγράψουμε με πάσα λεπτομέρεια στην συνέχεια [BG00].

Η χρήση του *RDF* σαν επιμέρους βέλτιστο μηχανισμό περιγραφής των πληροφοριακών πόρων, έγκειται επιμέρους στα εξής χαρακτηριστικά γνωρίσματα της:

1. *ανεξαρτησίας* του πεδίου εφαρμογής σε κάθε περίπτωση και η παροχή της δυνατότητας σε οποιονδήποτε να ορίζει κλάσεις ή ιδιότητες και να της επαναχρησιμοποιεί με βάση την επιβεβλημένη σημασιολογία.

2. *ευκολίας ανταλλαγής και μεταφοράς* των *RDF* περιγραφών τόσο από ανθρώπους αλλά κι από μηχανές διαχείρισης.

3. στην *δυνατότητα* της αναπαράστασης της γνώσης μέσω της *περιγραφής ιδιοτήτων /κλάσεων*, σε ένα πεδίο εφαρμογής μέσω μηχανισμών που υποστηρίζουν κοινές συμβάσεις για την σημασιολογία, τη σύνταξη και την δομή των μεταδεδομένων.

4. Μάλιστα μπορεί να αξιοποιηθεί και για μια σημασιολογικά πλούσια περιγραφή πληροφοριακών πόρων με την δημιουργία μεταδεδομένων μεταξύ υπαρχόντων και επαναχρησιμοποιούμενων ή κατ' επέκταση σχημάτων.

Εν γένει ο πυρήνας του *RDF* είναι ένα μοντέλο που αποτελείται από κατευθυνόμενους γράφους με ετικέτες (*labels*), που μπορούν να χρησιμοποιηθούν για την περιγραφή των σχέσεων μεταξύ τόσο των πόρων, όσο και των γνωρισμάτων των πόρων. Τα λεξιλόγια αυτά επιμέρους αποτελούνται από ιεραρχίες κλάσεων που στηρίζονται, καθώς κι από τύπους ιδιοτήτων που τις συσχετίζουν, αλλά και μπορούν να επεκταθούν μέσω των σχέσεων εξειδίκευσης (*isA*) τόσο για κλάσεις όσο και ιδιότητες.

2.1.2 RDF Μοντέλο Δεδομένων

Το μοντέλο δεδομένων του *RDF* είναι ένα πολύ γενικό και απλό μοντέλο για την περιγραφή των πληροφοριακών πόρων. Στους πόρους αποδίδονται ιδιότητες, οι οποίες αντιστοιχούν είτε σε *σχέσεις* μεταξύ πόρων είτε σε *γνωρίσματα* του πόρου που αποδίδονται. Συγκεκριμένα, το μοντέλο δεδομένων του *RDF* αποτελείται από τρεις

βασικούς τύπους δεδομένων: *Πόρους* (*Resources*), *Ιδιότητες* (*Properties*) και *Δηλώσεις* (*Statements*) τους οποίους και θα παρουσιάσουμε εν συνεχεία :

- ***Πόρος***, ονομάζεται οτιδήποτε που μπορεί να περιγραφεί με *RDF* εκφράσεις. Αναφορικά, μπορεί να είναι οτιδήποτε από τα παρακάτω:
    - μια σελίδα του διαδικτύου,
    - έγγραφα ή σελίδες με την μορφή *HTML, XML*
    - ένα τμήμα τέτοιων εγγράφων ή αποσπασματικά κάποιο στοιχείο μιας σελίδας μας
    - με τον χαρακτηρισμό πόροι μπορούμε κι ονομάζουμε και τα αντικείμενα τα οποία δεν είναι προσπελάσιμα μέσω του δια-δικτύου, όπως είναι για παράδειγμα ένα άτομο.

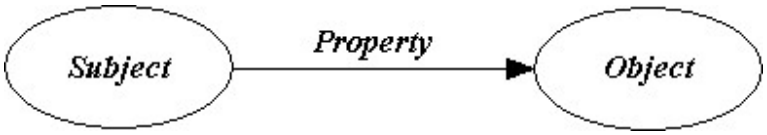
  Εν γένει πόρος μπορεί να θεωρηθεί οτιδήποτε στο οποίο αποδίδεται ένα μοναδικό *URI* (*Uniform Resource Identifier- Ομοιόμορφο Αναγνωριστικό Πόρων*) [LFIM98] στο οποίο μπορεί να προστεθεί και μια άγκυρα (anchor ids) – τα καλούμενα *qualified URIs*, τα οποία είναι *URIs* που μπορεί να περιέχουν ένα αναγνωριστικό τμήματος (fragment identifier), και το οποίο διαχωρίζεται από το υπόλοιπο *URI* με το χαρακτήρα '#'.

  Εάν θεωρήσουμε για παράδειγμα το URI "*http://dmoz.org*" αποδίδεται στην σελίδα με την αντίστοιχη διεύθυνση της τιμής του *URI*, ενώ το *URI* "*http://www.ics.forth.gr/RDF/Dublin_Core.rdf#Title*" αντιστοιχεί την ιδιότητα *Title* που ορίζεται στο αρχείο "*http://www.../Dublin_Core.rdf.*".

- ***Ιδιότητα,*** είναι ένα γνώρισμα που χρησιμοποιείται για να περιγράψει είτε ένα πόρο είτε μια σχέση μεταξύ πόρων. Μια ιδιότητα έχει καθορισμένη σημασιολογία και πιθανόν έχει κάποιο πεδίο ορισμού, πεδίο τιμών καθώς και συσχετίσεις με άλλες ιδιότητες. Κι οι ιδιότητες με την σειρά τους είναι επιμέρους πόροι με απόλυτα διακριτά *URIs* και άρα μπορούν να περιγράφουν με αντίστοιχες των κλάσεων *RDF* δηλώσεις. Αποτελούν αυτόνομες οντότητες και δεν χαρακτηρίζονται από την κλάση στην οποία ορίζονται και προσδιορίζουν, όπως

συμβαίνει στα διάφορα υπάρχοντα επικρατούντα οντοκεντρικά μοντέλα. Στις ιδιότητες έχει πολύ μεγάλη σημασία η κατεύθυνση της ιδιότητας από την άποψη ότι δεικνυοδοτεί το αντικείμενο της *δήλωσης* που ανήκει. Ένα σύνολο ιδιοτήτων που αποδίδονται σε ένα πόρο ονομάζεται *περιγραφή* του πόρου αυτού.

- **Δήλωση,** με την σειρά της ονομάζεται ο συνδυασμός της τριάδας *πόρος - ιδιότητα - τιμή ιδιότητας*. Συγκεκριμένα, τα τρία αυτά μέρη κάθε μίας δήλωσης ονομάζονται θέμα (*subject*), κατηγόρημα (*predicate*) και αντικείμενο (*object*) αντίστοιχα με την σειρά αναφοράς τους (βλ. Σχήμα 2 παράδειγμα δήλωσης ενός γράφου).



Σχήμα 2: RDF Γράφος

Το αντικείμενο μίας δήλωσης μπορεί να είναι είτε πόρος – ο οποίος με την σειρά του μπορεί να έχει κι άλλες ιδιότητες - , είτε κι ένα *Literal*, δηλαδή μια συγκεκριμένη κλάση που χρησιμοποιείται για να περιγράψει τιμές όπως π.χ. αριθμοί και ημερομηνίες μέσω λεκτικών σταθερών (*literals*), όπως είναι απλές συμβολοσειρές οι τιμές άλλων βασικών τύπων δεδομένων *XML* [BPS98]. Οτιδήποτε που μπορεί να αναπαρασταθεί με ένα *literal* μπορεί και να αναπαρασταθεί με ένα *URI*., εάν κι η χρήση του *literal* είναι πιο βολική και δημιουργική συνάμα. Ένα literal μπορεί να είναι το *αντικείμενο* μιας δήλωσης *RDF*, αλλά ποτέ το *υποκείμενό* της.

Το μοντέλο του *RDF* μπορεί να συντελέσει τόσο στην δημιουργία, περιγραφή αλλά και διαμόρφωση ενός μοντέλου μεταδεδομένων με σημασιολογία κατανοητή από τις μηχανές. Μάλιστα για την επιθυμητή αναπαράσταση του *RDF* υπάρχουν τρεις διαφορετικοί τρόποι προσέγγισης και περιγραφής του:

1. Με βάση μια οντοκεντρική προσέγγιση προχωρούμε στη χρήση κατευθυνόμενων γράφων με ετικέτες, που αποτελούν την απεικόνιση του μοντέλου δεδομένων με τρόπο ανεξάρτητο από το χρησιμοποιούμενο συντακτικό.

2. Με βάση μια σχεσιακή περιγραφή, μπορούμε να περιγράψουμε το RDF μέσω ενός μοντέλου δηλώσεων-των προαναφερθέντων σχηματιζόμενων *τριάδων* που επίσης εκφράζει την σημασιολογία των περιγραφών.

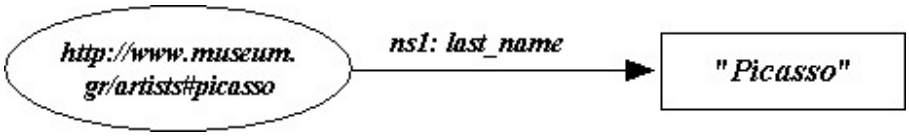
3. Τέλος, με βάση μια σειριοποίηση των μεταδεδομένων σε αρχεία, και μάλιστα στην συντακτική αναπαράσταση του RDF, που με την σειρά του βασίζεται στο συντακτικό που διέπει την XML όπως περιγράφεται στο RDF M&S [LS99]. Η αναπαράσταση με την μορφή αυτή είναι βέλτιστη υπό την άποψη ότι είναι κατανοητή κι από τον άνθρωπο, αλλά κι άμεσα επεξεργάσιμη από διάφορα υπολογιστικά προγράμματα να τα διαχειριστούν περαιτέρω.

Εκ του σημείου αυτού και ύστερα θα προχωρήσουμε, μόνο με την χρήση της αναπαράστασης των RDF περιγραφών σε μορφή κατευθυνόμενων γράφων με ετικέτες σε μια ανάλυση του RDF. Κι αυτό γιατί θεωρούμε πως με αυτήν την μορφή η περιγραφή του RDF απλοποιείται στο μέγιστο βαθμό και παρουσιάζει με πάσα ευκρίνεια τα χαρακτηριστικά του μοντέλου δεδομένων της RDF, που θέλουμε άλλωστε να αναδείξουμε μέσω ενός βέλτιστου σχήματος κωδικοποίησης.

2.1.3.1 RDF Γράφοι

Οι RDF γράφοι είναι κατευθυνόμενοι γράφοι με ετικέτες, όπου όπως προείπαμε οι μεν κόμβοι αντιστοιχούν στους πόρους ή σε *Literal* και οι δε ακμές αντιστοιχούν στις ιδιότητες. Διαφοροποίηση υπάρχει και ανάμεσα στην γραφική απεικόνιση των κόμβων που αντιστοιχούν σε πόρους να απεικονίζονται με ελλείψεις, ενώ οι κόμβοι που αντιστοιχούν σε *literal* να απεικονίζονται με παραλληλόγραμμα. Δεδομένου ότι το πεδίο ορισμού μιας ιδιότητας είναι πάντα ένας πόρος , ενώ η τιμή της είναι είτε ένας πόρος είτε μια τιμή βασικού τύπου. Άρα μια ακμή στον γράφο ξεκινά πάντα από ένα ελλειψοειδές

κόμβο και καταλήγει σε ένα ελλειψοειδή ή παραλληλόγραμμο κόμβο, όπως βλέπουμε και στο σχήμα 3 ένα απλό παράδειγμα γράφου.

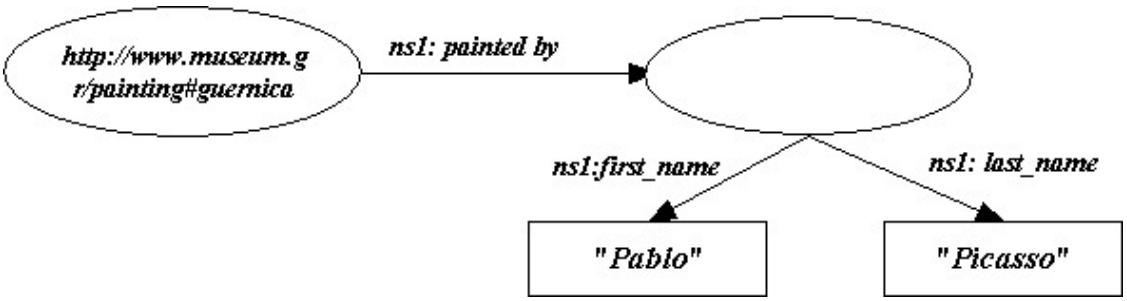


Σχήμα 3: Παράδειγμα RDF γράφου (με τιμή ιδιότητα ένα Literal)

Ο γράφος του σχήματος 3, μεταφράζοντάς τον σε φυσική γλώσσα, μας αναπαριστά την φράση :Το επώνυμο (*ns1:last_name*) του ***http://www.museum.gr/artists#picasso*** είναι "*Picasso*". Στο σημείο αυτό θα πρέπει να προσθέσουμε πως το πρόθεμα ns1 που χρησιμοποιήσαμε μπροστά από την ονομασία της ιδιότητας *last_name* είναι μια έκφραση συντομογραφίας του πλήρους URI που θα μπορούσε να χρησιμοποιηθεί, δηλαδή *http://www.culture.gr/schema.rdf#last_name*. Η δυνατότητα αυτή συντομογραφίας προέρχεται από την αντίστοιχη δυνατότητα ονοματοδοσίας της XML [BHL99], η οποία καθορίζει τον ορισμό προθεμάτων που υποδεικνύουν το έγγραφο στο οποίο έχει οριστεί το αντίστοιχο στοιχείο.

Τα αναγνωριστικά τόσο των κόμβων όσο και των βελών είναι τα *URIs*. Δεδομένου ότι ένας πόρος μπορεί να εμφανίζεται σε πολλές περιγραφές, η μοναδικότητα των *URIs* κάνει εφικτό τον συνδυασμό μεταδεδομένων που αναφέρονται στον ίδιο πόρο και βρίσκονται σε διαφορετικά αρχεία ή σε διαφορετικές περιγραφές.

Ας προχωρήσουμε σε ένα πιο δύσκολο παράδειγμα γράφου, που είναι αυτό του παρακάτω σχήματος 4. Σε αυτό το σχήμα περιγράφουμε το άτομο με όνομα "Pablo" και επίθετο "Picasso" πρέπει να εισάγουμε ένα καινούριο πόρο (όχι *literal*) ώστε να μπορούμε να του αποδώσουμε ιδιότητες. Το αναγνωριστικό του πόρου αυτού είναι άγνωστο και γι' αυτό χρησιμοποιείται ένα άδειο οβάλ σχήμα για την αναπαράστασή του. Σε επίπεδο εφαρμογής θα πρέπει να αποδοθεί κάποιο μοναδικό αναγνωριστικό σε όλους τους ανώνυμους πόρους ώστε να μπορούν να διαφοροποιηθούν απολύτως μεταξύ τους.



Σχήμα 4: Σύνθετο Παράδειγμα RDF γράφου (με τιμή ιδιότητα έναν πόρο)

2.1.3.2 RDF Συλλογές

Οι τιμές μιας ιδιότητας συχνά χρειάζεται να είναι μια συλλογή από πόρους ή από *literals* (αλφαριθμητικά). Οι *RDF* συλλογές (*RDF Containers*) στο *RDF* μοντέλο δεδομένων είναι μια ειδική κατηγορία πόρων οι οποίοι περιέχουν συλλογές από πόρους ή *literals*. Υπάρχουν 3 είδη *RDF* συλλογών:

1. ***Bag,*** *ονομασία με την οποία* καλείται ένα επιμέρους σύνολο από πόρους ή literals. Χρησιμοποιείται για να δηλώσει ότι μία ιδιότητα έχει πολλαπλές τιμές και ότι η σειρά των τιμών αυτών δεν έχει σημασία.. Τα πολυσύνολα μπορούν να έχουν περιέχουν πολλές φορές τα ίδια μέλη.
2. ***Sequence***, με την οποία καλείται μια ορισμένη λίστα από πόρους ή literals. Χρησιμοποιείται για να δηλώσει ότι μία ιδιότητα έχει πολλαπλές τιμές και ότι η σειρά των τιμών αυτών είναι σημαντική. Οι ακολουθίες μπορούν να έχουν περιέχουν πολλές φορές τα ίδια μέλη.
3. ***Alternative,*** που είναι μια συλλογή από πόρους ή *literals*. Τα μέλη του συνόλου αντιπροσωπεύουν εναλλακτικές τιμές για την τιμή μιας ιδιότητας. Εφαρμογές οι οποίες διαχειρίζονται παραλλαγές 'γνωρίζουν' ότι μπορούν να επιλέξουν οποιαδήποτε από τα μέλη της λίστας σαν τιμή της ιδιότητας. Το πρώτο μέλος μίας παραλλαγής αποτελεί την προκαθορισμένη τιμή, αλλά βέβαια μπορεί να επιλεχτεί οποιαδήποτε άλλο μέλος της παραλλαγής σαν τιμή της ιδιότητας.

2.1.4 Γλώσσα Περιγραφής RDF Σχημάτων (RDF/S)

Η *RDF/S* (*RDF Schema Specification Language*) αποτελεί μια γλώσσα ορισμού σχημάτων, σχήματα στα οποία περιλαμβάνονται επιμέρους κλάσεις και ιδιότητες, κατορθώνει να ικανοποιεί και να συμπληρώνει την εξαρχής αδυναμία του προτύπου του *RDF* να παρέχει επιμέρους μηχανισμούς δήλωσης κι εν συνεχεία περιγραφής των ιδιοτήτων και των κλάσεων που χρησιμοποιούνται στις *RDF* περιγραφές. Η επέκταση του προτύπου αυτού είναι αναγκαία δεδομένου ότι τ απλό μοντέλο δεδομένων του *RDF* είναι ένα από μοντέλο περιγραφής των σχέσεων μεταξύ των πόρων.

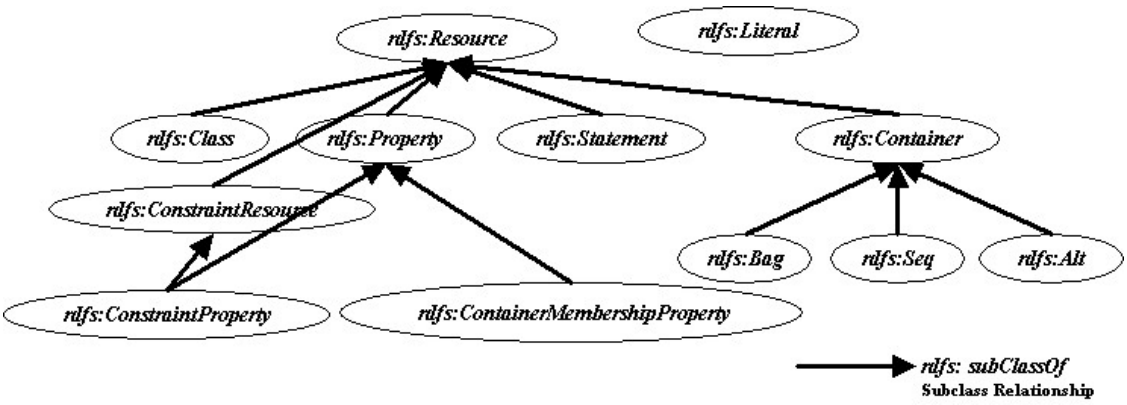
. Η *RDF/S* με την σειρά της παρέχει ένα σύνολο από βασικούς τύπους - κλάσεις και ιδιότητες -, που χρησιμοποιούνται για την δημιουργία των RDF σχημάτων. Τέτοια χαρακτηριστικά παραδείγματα είναι ότι :

1. Η *RDF/S* ορίζει ιδιότητες που εκφράζουν περιορισμούς για το πεδίο ορισμού και εξίσου για το πεδίο τιμών των ιδιοτήτων.
2. Επίσης η *RDF/S* παρέχει πόρους (αντιστοίχως κι ιδιότητες), οι οποίες με την σειρά τους δηλώνουν σχέσεις μεταξύ κλάσεων ή μεταξύ ιδιοτήτων.

Τα *RDF* σχήματα ορίζονται χρησιμοποιώντας τους τύπους που παρέχει η *RDF/S* και βασίζονται στο RDF μοντέλο δεδομένων. Άρα τα σχήματα είναι κι αυτά επιμέρους RDF μεταδεδομένα. Το γεγονός αυτό παρέχει ευελιξία στις εφαρμογές δεδομένου ότι δεν χρειάζεται να γνωρίζουν εκ των προτέρων την σημασιολογία των σχημάτων στα οποία στηρίζονται τα μεταδεδομένα που αναλύουν αλλά μπορούν να την εξάγουν. Τα RDF σχήματα παρέχουν δυνατότητα ερμηνείας των μεταδεδομένων και έμμεσα επηρεάζουν την δομή των μεταδεδομένων λόγω των περιορισμών που υπάρχουν στο πεδίο τιμών (*rdfs:domain*) και ορισμού (*rdfs:range*) των ιδιοτήτων.

2.1.4.1 Βασικές RDF/S Κλάσεις - Ιδιότητες

Στη συνέχεια θα προχωρήσουμε στην ανάλυση των βασικών κλάσεων που σε κάθε περίπτωση είτε ορίζονται είτε περιγράφονται στο RDF σχήμα. Οι κλάσεις συσχετίζονται με σχέσεις υποσυνόλου /υπερσυνόλου κι άρα δημιουργούν μια μορφής ιεραρχία απεικονίζεται στο σχήμα 5 με κορυφή της την κλάση *rdfs:Resource*



Σχήμα 5: Ιεραρχία RDF/S κλάσεων

- ***rdfs:Resource*****:** Μέλη της κλάσης αυτής είναι οτιδήποτε μπορεί να περιγραφτεί με RDF δηλώσεις. Μπορούμε σε αυτήν να αντιστοιχήσουμε οτιδήποτε προαναφέραμε ανήκει στο σύνολο *Πόροι* (που προαναφέραμε στο 2.3 *RDF* Μοντέλο Δεδομένων).
- ***rdfs:Class*****:** Η σημασιολογία της κλάσης αυτής έγκειται άμεσα με τις γενικές έννοιες *Τύπος*, *Κατηγορία* ή *Κλάση* στις διάφορες οντοκεντρικές γλώσσες. Αναλυτικά, θα πρέπει ο κάθε πόρος που αντιπροσωπεύει μια επιμέρους RDF κλάση να δηλώνεται σαν μέλος της δηλαδή συνοδεύεται απαρεγκλίτως από την ιδιότητα *rdf:type* με τιμή *rdfs:Class*.
- ***rdf:Property*****:** Αναπαριστάνει το υποσύνολο των πόρων που είναι ιδιότητες.
- ***rdfs:Literal:*** Αντιπροσωπεύει το σύνολο *Literal*, στο οποίο ανήκουν ατομικές τιμές όπως αλφαριθμητικά και ακέραιοι.
- ***rdfs:ConstraintResource:*** Η κλάση αυτή όπως φαίνεται και στο σχήμα είναι υποκλάση της ***rdfs:Resource***. Μέλη της είναι το υποσύνολο των κλάσεων και ιδιοτήτων αυτών που χρησιμοποιούνται για να εκφράζουν περιορισμούς και με αυτόν τον τρόπο να ελέγχεται η ορθότητα μεταξύ των εξεταζόμενων RDF σχημάτων. Βέβαια ελλοχεύει ο κίνδυνος. Περιορισμοί που δεν έχουν οριστεί στην *RDF/S*, δεν δύνανται να επεξεργαστούν από RDF επεξεργαστή δεν είναι σε θέση να αποφανθεί για την συνέπεια τους ή ασυμβατότητά τους.
- ***rdfs:ConstraintProperty:*** Αποτελεί υποκλάση της ***rdfs:ConstraintResource*** και της ***rdf:Property*** και μέλη της είναι όλες οι ιδιότητες που δηλώνουν κάποιο

περιορισμό. Οι ιδιότητες ***rdfs:range*** και ***rdfs:domain*** που θα περιγραφούν παρακάτω και περιορίζουν το πεδίο τιμών και το πεδίο ορισμού μιας ιδιότητας ανήκουν στη κλάση αυτή.

- ***rdfs:Container*****:** Η κλάση αυτή αντιπροσωπεύει το σύνολο των *συλλογών πόρων* (containers). Υποκλάσεις της είναι οι κλάσεις ***rdf:Bag, rdf:Seq*** και ***rdf:Alt***.
- ***rdf:Bag rdf:Seq*** και ***rdf:Alt*****:** Αναπαριστάνουν τα σύνολα *Bag, Sequences* και *Alternative* κι είναι υποκλάσεις της rdfs:Container.
- ***rdfs:ContainerMembershipProperty***: Περιέχει τις ιδιότητες rdf:_1, rdf:_2…, οι οποίες δηλώνουν τα μέλη μιας συλλογής. Οι ιδιότητες είναι άπειρες. Είναι υποκλάση της rdf:Property.
- ***rdf:Statement*****:** Αναπαριστάνει το σύνολο των υποστασιοποιημένων δηλώσεων. Η υποστασιοποίηση αποτελεί μία από τις κύριες ιδιότητες του RDF/S σχήματος, καθώς χρησιμοποιείται στην περιγραφή των ίδιων των μεταδεδομένων. Κι αυτό για να περιγράψουμε μια δήλωση θα πρέπει να δημιουργήσουμε την υποστασιοποιημένη δήλωσή της. Η υποστασιοποιημένη δήλωση είναι ένας νέος πόρος με τέσσερις ιδιότητες ***rdf:subject*** (αντίστοιχο θέμα υποστασιοποίησης της δήλωσης), ***rdf:predicate*** (ομοίως το θέμα της δήλωσης), ***rdf:object*** (ομοίως το θέμα της δήλωσης)και ***rdf:type*** (για κάθε πόρο που αντιστοιχεί σε υποστασιοποιημένη δήλωση η κλάση r***df: Statement***). Ένα παράδειγμα υποστασιοποιημένη δήλωσης είναι "Ο Δημήτρης λέει ότι η σελίδα http://www.csd.uoc.gr/~tsigos έχει τίτλο προσωπική σελίδα".

2.1.4.2 Βασικές RDF/S Ιδιότητες

Παρακάτω αναλύονται οι πλέον βασικές ιδιότητες που ορίζονται κι εν συνεχεία χρησιμοποιούνται στο *RDF* σχήμα.

- ***rdf:type*****:** Δηλώνει ότι ένας πόρος είναι μέλος μιας κλάσης. Στο *RDF* δεν υπάρχει η έννοια της αυστηρής κληρονομικότητας των ιδιοτήτων και οι κλάσεις δεν επιβάλλουν καμιά δομή στα μέλη τους κι ένας πόρος μπορεί να είναι μέλος πολλαπλών κλάσεων. Οι κλάσεις που ορίζονται στην *RDF/S* άλλα και οι κλάσεις

που ορίζονται στα *RDF* σχήματα έχουν την ιδιότητα rdf:type με τιμή **rdfs:Class**. Αντίστοιχα, η ιδιότητα rdf:type αποδίδεται τόσο στις ιδιότητες που ορίζονται στην *RDF/S* όσο και στις απλές ιδιότητες με τιμή ***rdf:Property***. Το πεδίο ορισμού της ιδιότητας rdf:type είναι η κλάση rdfs:Resource.

- ***rdfs:subClassOf:*** Δηλώνει την σχέση υποσυνόλου-υπερσυνόλου μεταξύ επιμέρους κλάσεων κι είναι μεταβατική ιδιότητα.. Θα πρέπει ο πόρος στον οποίο εφαρμόζεται η ιδιότητα ***rdfs:subClassOf*** όσο και η τιμή της ιδιότητας πρέπει να είναι κλάσεις. Μια κλάση μπορεί να είναι υποκλάση πολλών κλάσεων. Δεν μπορεί όμως να δηλωθεί σαν υποκλάση του εαυτού της ή κάποιας από τις υποκλάσεις της, κι αυτό γιατί δεν επιτρέπεται η δημιουργία κύκλων στην ιεραρχία των υποκλάσεων.
- ***rdfs:range /rdfs:domain*:** Από κοινού μέλη της κλάσης ***rdfs:ConstraintProperty*** και χρησιμοποιούνται για να ορίσει τις κλάσεις οι οποίες αποτελούν το πεδίο τιμών ή στις οποίες μπορεί να εφαρμοστεί μια ιδιότητα. Μια ιδιότητα μπορεί να έχει μηδέν, μία ή περισσότερες rdfs:domain ιδιότητες.
- **rdf:predicate, rdf:subject, rdf:object:** Αντιπροσωπεύουν τις ιδιότητες κατηγόρημα (predicate), θέμα (subject) και αντικείμενο (object) και χρησιμοποιούνται για να σχηματίσουν την υποστασιοποιημένη δήλωση.

2.1.5 Μηχανισμός Ονοματοδοσίας XML

Ο βασικός μηχανισμός που χρησιμοποιεί το *RDF* για τον προσδιορισμό των *RDF* σχημάτων και για την δημιουργία *RDF* μεταδεδομένων είναι ο μηχανισμός ονοματοδοσίας *XML* [BHL99]. Θεωρούμε ως ένα χώρο ονοματοδοσίας μια συλλογή από τύπους στοιχείων και /ή γνωρίσματα, όπου σε κάθε ένα αποδίδεται ένα μοναδικό *URI*, όπως και ένα *RDF* σχήμα ορίζεται επίσης σε ένα χώρο ονοματοδοσίας με το *URI* του χώρου αυτού. Για παράδειγμα στο *RDF* σχήμα αποδίδεται το *URI http://www.w3.org/2000/01/rdf-schema#.* Οι πόροι που ορίζονται σε ένα σχήμα έχουν μοναδικά ονόματα που προκύπτουν συνδυάζοντας το *URI* του σχήματος κι εν συνεχεία με τα τοπικά ονόματα των πόρων. Είναι προφανές ότι το *RDF* αγνοεί πιο έξυπνους μηχανισμούς ονοματοδοσίας σε βάσεις πληροφοριών όπως αυτοί που στηρίζονται στην

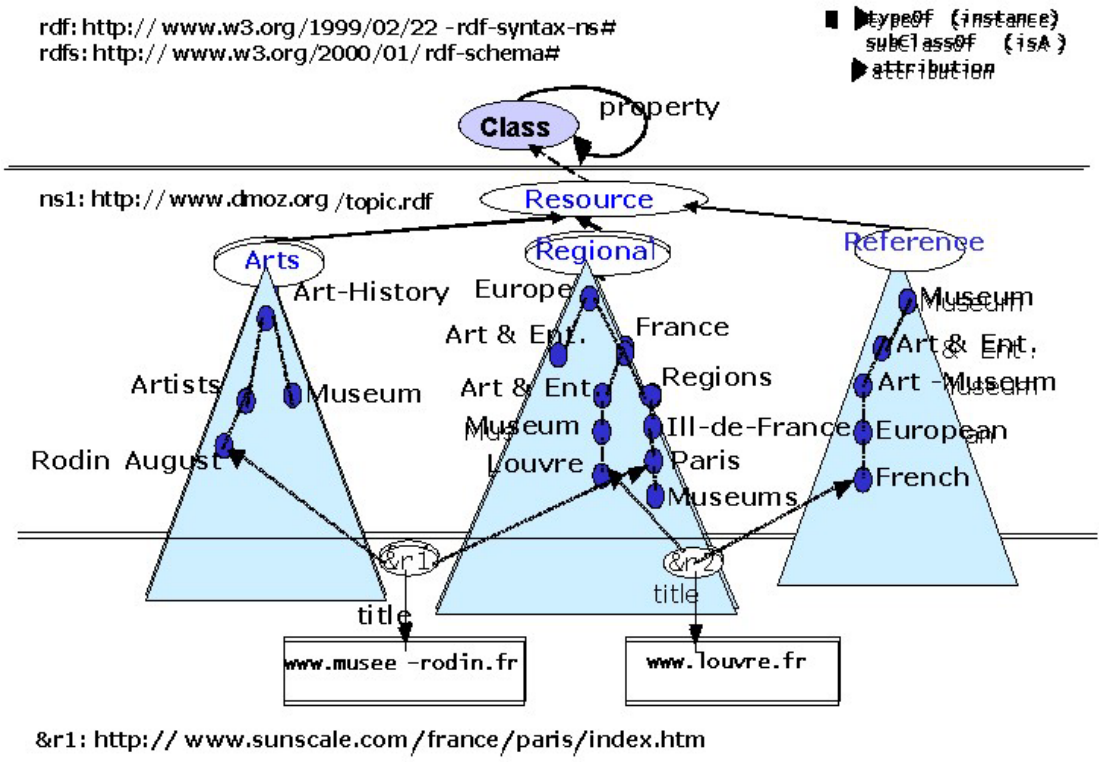
έννοια των πλαισίων συμφραζομένων [TC97].

Κατά την δημιουργία RDF μεταδεδομένων προκειμένου να χρησιμοποιήσουμε τις ιδιότητες ή τις κλάσεις που ορίζονται σε κάποιο σχήμα, δημιουργούμε μια δήλωση με την οποία σχετίζεται ένας κωδικός /πρόθεμα (π.χ., ***rdfs***) με το *URI* ενός σχήματος (π.χ., *http://www.w3.org/2000/01/rdf-schema#*). Οι κλάσεις και οι ιδιότητες του σχήματος αυτού τότε αναφέρονται μέσα στα RDF μεταδεδομένα συνδυάζοντας το πρόθεμα με το τοπικό όνομα της ιδιότητας (π.χ., ***rdfs:Class***).

2.2 Παράδειγμα Πύλης Διαδικτύου – ODP κατάλογος

Στο παρακάτω σχήμα 6 αναπαρίσταται ένα μέρος του χρησιμοποιούμενου από το *RDFS* σχήμα του *Netscape Open Directory* (ή *Dmoz* που αποτελεί ακρώνυμο του Directory Mozilla) *Portal* (*ODP*) όπως φαίνεται και στο παραπάνω σχήμα 1, το οποίο αναγνωρίζεται μοναδικά από το αναγνωριστικό (*namespace*) ***ns1*** (π.χ. *www.dmoz.org/topics.rdfs*)[2]: οι κόμβοι στην εν λόγω αναπαράσταση του σχήματος αυτού δηλώνουν διακριτά ονόματα κλάσεων / θεμάτων (π.χ. *Museum*) και οι χωρίς κενά ακμές δηλώνουν σχέσεις υπαλληλίας (*subsumption*) μεταξύ κόμβων (π.χ. *ArtMuseum* $\prec$ *Museum*). Στο σημείο αυτό να παρατηρήσουμε πως οι ριζικοί κόμβοι των θεματικών ιεραρχιών (π.χ. *Arts, Regional, Reference*) εξειδικεύουν την βασική RDF/S κλάση *Resource*. Συνεπώς, όλες αυτές οι ιεραρχίες μεμονωμένα αποτελούν αντίστοιχα δέντρα υποκλάσεων, τα οποία παίζουν τον ρόλο των "*όψεων*" (*facets*), που είναι ορθογώνια σύνολα πληροφοριακών κατηγοριών, οι οποίες μπορούν να συνδυαστούν μεταξύ τους ώστε να περιγράψουν και να ανακτήσουν διαδικτυακές πόροι.

[2] Για λόγους απλότητας στην παρουσίαση του εν λόγω παραδείγματος παραλείπουμε μια σειρά από πολλά διαχειριστικά μεταδεδομένα (όπως π.χ. τίτλοι, μιμητικοί τύποι, ημερομηνίες τροποποίησης) δικτυακών πηγών που αναπαρίστανται στο ODP μέσω ενός ομοιάζοντος OCLC Dublin-Core σχήματος [WMD95]



Σχήμα 6 : Παράδειγμα της ιεραρχίας του RDF καταλόγου της πύλης του Open Directory

Χρησιμοποιώντας ταξινόμηση με την χρήση "όψεων", ένας πόρος μπορεί να περιγραφεί (ταξινομηθεί κατά μία έννοια) χρησιμοποιώντας ένα ή περισσότερα θέματα από κάθε όψη. Για να γίνει αυτό εμφανές ας πάρουμε ένα χαρακτηριστικό παράδειγμα από το πάνω σχήμα , ο διαδικτυακός τόπος του μουσείου του Rodin στο Παρίσι, ταξινομείται κάτω από και τα δυο *Reference/Museum/Arts&Entertainment/ArtMuseum/European/France* και *Regional/Europe/France/Regiona/Ill-de-France/Paris/Museums* – οι διακεκομμένες ακμές χρησιμοποιούνται για την δήλωση επιμέρους σχέσεων instantiation. Μπορούμε να παρατηρήσουμε πως τα ονόματα των θεμάτων συντάσσονται από διαφορετικούς περιγραφικούς όρους (π.χ. *Museum, France*). Κι αυτό βασίζεται στο γεγονός ότι οι σχεδιαστές του σχήματος του ODP μερικώς επαναλαμβάνουν αυτούς τους όρους σε διάφορες θεματικές ιεραρχίες, στοχεύοντας να δηλώσουν όλους τους έγκυρους συνδυασμούς όρων (από διαφορετικές "*όψεις*"), και οι οποίοι μπορούν να χρησιμοποιηθούν κατά την διάρκεια της περιγραφής των πηγών. Συγκεκριμένα, στο προαναφερθέν χρησιμοποιούμενο παράδειγμά μας πολιτιστικοί και γεωγραφικοί όροι

(π.χ. οι Museum, France) εμφανίζονται από κοινού στις *Reference* και *Regional* ιεραρχίες. Το πλήρες μονοπάτι από την ρίζα μέχρι την εκάστοτε εξεταζόμενο θεματική κλάση χρησιμοποιείται σαν πρόθεμα για να διαφοροποιήσουμε τα ονόματα. Για λόγους απλότητας, εκ του σημείου αυτού κι εν συνεχεία θα παραλείπουμε τα αναγνωριστικά του σχήματος, όπως επίσης και τα μονοπάτια από τον ριζικό κόμβο των θεματικών κατηγοριών του ODP σχήματος που χρησιμοποιούνται σαν πρόθεμα.

Στον πίνακα 1 παρουσιάζονται πλήρως όλες οι στατιστικές πληροφορίες που εξάγονται από τις 16 ιεραρχίες του *ODP* σχήματος (έκδοση 16-01-2001). Στις ιεραρχίες αυτές περιλαμβάνονται από κοινού 253214 επιμέρους διακριτά θέματα (*topics*), βάση των οποίων μπορούν 1688037 επιμέρους διαδικτυακούς πόρους (*Resources*) να οριστούν και να ταξινομηθούν. Στο σημείο αυτό να αναφέρουμε πως με τον ορισμό *fan-in* εννοούμε τον βαθμό fan-in ανά κλάση στο δέντρο που σχηματίζεται, δηλαδή με άλλα λόγια το πλήθος των άμεσων υποκλάσεων της εκάστοτε εξεταζόμενης κλάσης. Ο συνολικός αριθμός των διακριτών όρων, που χρησιμοποιούνται από το σύνολο όλων των θεμάτων, είναι 80795, εκ των οποίων 14355 από αυτούς (ποσοστό 17,77%) επαναλαμβάνεται σε περισσότερα από ένα ονόματα θεμάτων (μάλιστα στην κατηγορία αυτών των όρων επαναλαμβάνονται όροι κυρίως από τις ιεραρχίες *Arts*, *Regional*, *Business* και *Shopping*). Επιπρόσθετα, κάτω από αυτά τα θέματα, ορίζεται ένας αριθμός από 1715225 πηγές, ενώ 118925 (ποσοστό 6,93%) από αυτούς ορίζεται πολλαπλώς, δηλαδή κάτω από περισσότερο από ένα θέμα και μόνο. Συνεπώς, εξαιτίας της μερικής επανάληψης των όρων, οι θεματικές ιεραρχίες του *ODP* είναι σχετικά βαθιές με τον μέσο όρο βάθους να είναι 7.83 και με την μέγιστη τιμή βάθους να είναι 13.

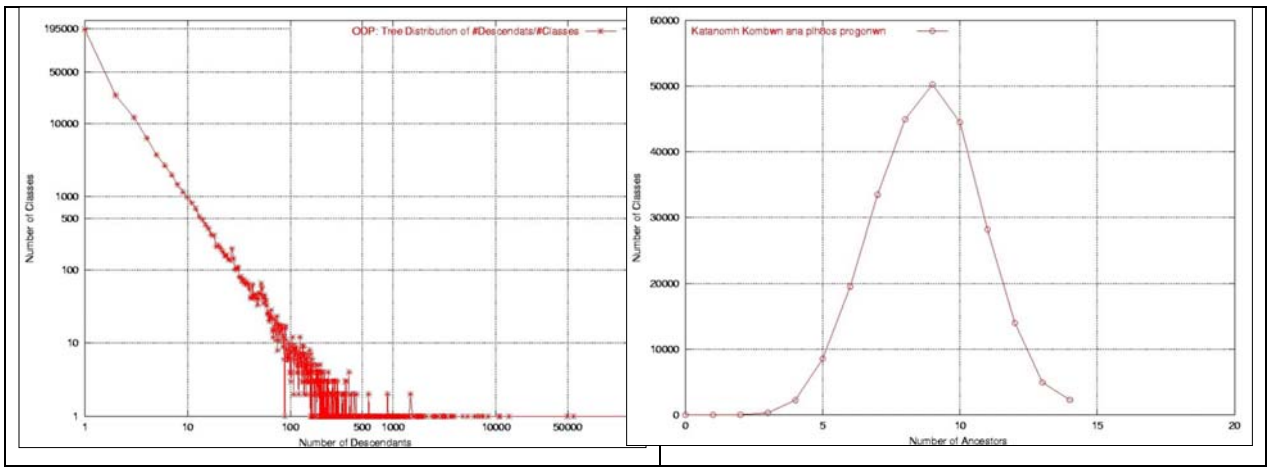
Επίσης, στον πίνακα 1 παρατηρούμε ότι ο μέσος όρος του αριθμού των εξαγόμενων ακμών (*fan-out*) όλων των θεματικών κατηγοριών είναι 0.9999 και η μέγιστη τιμή είναι 314. Ένα επίσης αξιοπρόσεχτο σημείο που θα πρέπει να δώσουμε την προσοχή μας σε αυτόν τον πίνακα είναι ότι με σχετική λεπτομέρεια αναφέρονται το βάθος των κόμβων με την μέγιστη τιμή εξαγόμενων ακμών (*fan-out*) για κάθε μία ιεραρχία αλλά και την τιμή αυτή αντίστοιχα.

Hierarchy	Max. Depth	Avg. Depth	Max. Fan-in at Depth	Avg. Fan-in	#Topics	#Terms	#Resources
netscape.rdf	7	5.75	24 / 1	0.9948	389	203	27188
News.rdf	7	5.05	51 / 4	1.0027	721	411	47735
Kat.rdf	7	4.84	46 / 4	1.0026	761	646	7730
home.rdf	8	5.43	53 / 4	1.0011	1722	1353	26688
shopping.rdf	9	5.67	61 / 2	1.0005	3349	2357	88821
health.rdf	9	5.3	52 / 8,5	1.0006	3202	1728	45519
games.rdf	10	6.74	125 / 3	1.0004	4857	3710	36181
computers.rdf	10	6.4	147 / 3	1.0003	6010	3259	91597
recreation.rdf	11	6.8	85 / 3	1.0002	7269	3243	93929
business.rdf	11	6.44	52 / 4,5	1.0002	6833	3630	161877
reference.rdf	13	8.73	154 / 3	1.0003	6483	3759	75105
sports.rdf	9	7.14	178 / 6	1.0001	10625	5927	66280
science.rdf	10	8.35	314 / 6	1.0002	8667	6812	65939
society.rdf	12	7.9	157 / 7	1.0001	16250	8678	161433
Arts.rdf	11	7.04	267 / 4	1.0000	25314	16840	214795
Regional.rdf	13	8.27	254 / 7	0.9999	150762	32594	587152
Total	13	7.83	314	0.9999	253215	80795	1715225

Πίνακας 1: Στατιστικά των θεματικών ιεραρχιών του ODP καταλόγου

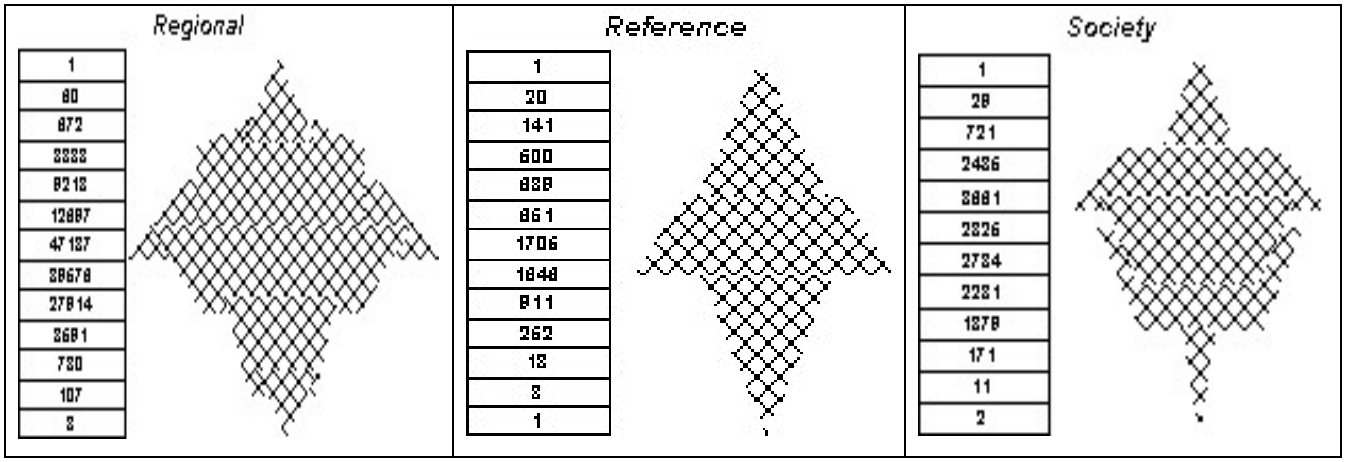
Στο σχήμα 8 παρουσιάζεται με γραφική αναπαράσταση η κατανομή των κλάσεων ανά τιμή βάθους για τις ιεραρχίες *Regional*, *Reference* και *Society*. Μπορούμε μέσω αυτών των γραφημάτων να παρατηρήσουμε πως σ' αυτές τις ιεραρχίες, τα δέντρα των σχηματιζόμενων υποκλάσεων δεν είναι πλήρη και το μεγαλύτερο ποσοστό των κλάσεων εμφανίζεται στο πάνω μισό μέρος των δέντρων των ιεραρχιών. Αυτό γίνεται άμεσα ορατό από το σχήμα 7α) όπου περιγράφεται η κατανομή των κόμβων του ODP καταλόγου σε σχέση με το πλήθος των προγόνων τους. Συγκεκριμένα, διαφαίνεται εύκολα ότι το μεγαλύτερο μέρος των κόμβων συγκεντρώνεται κυρίως και πολύ περισσότερο κοντά στην μέση της ιεραρχίας (τιμές βάθους μεταξύ του 6 και 11). Επίσης από το σχήμα 7b) παρατηρείται ότι ο μεγάλος αριθμός κόμβων της ιεραρχίας είναι "φύλλα"-κόμβοι δηλαδή χωρίς απογόνους (190231, δηλαδή ποσοστό 75,16 %), ενώ έως

10 το πολύ απογόνους ο αριθμός των κόμβων είναι 54250, δηλαδή ποσοστό 21,42%.Οι υπόλοιποι κόμβοι σχεδόν μεμονωμένα ή σε μικρή πληθικότητα φτάνουν μέχρι και την τιμή των 253214 απογόνων, που αντιστοιχεί στον ριζικό κόμβο της ιεραρχίας του ODP.



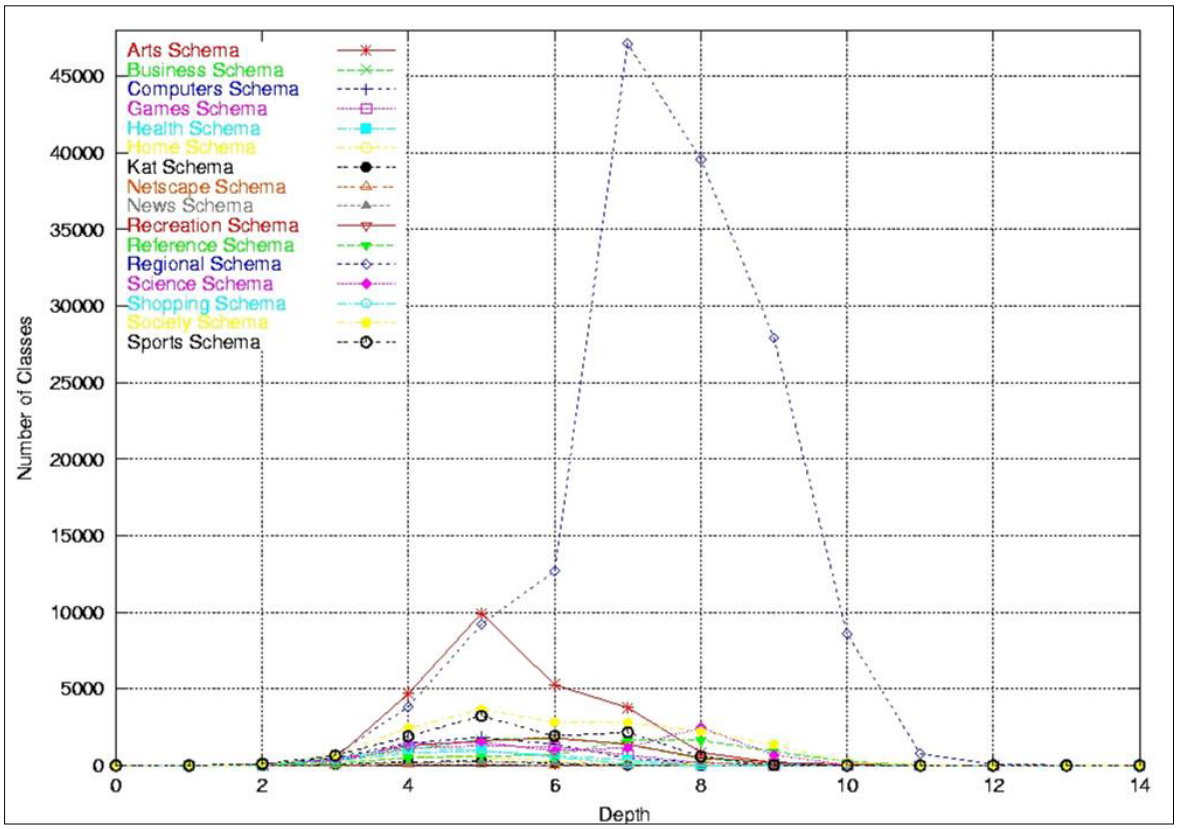
Σχήμα 7 : a) Κατανομή Κλάσεων/ Πλήθος Απογόνων, b) Κατανομή Κλάσεων/ Πλήθος Προγόνων

Επιπρόσθετα, στην πλειονότητά τους οι ιεραρχίες του ODP καταλόγου παρουσιάζουν το μέγιστο αριθμό των εξαγόμενων ακμών /υποκλάσεων (*fan-out*) στην μέση και στο πάνω του μισού του βάθους των ιεραρχιών. Αυτό μας επιτρέπει να παρατηρήσουμε πως στα σχήματα των διαδικτυακών πυλών, τα δέντρα των υποκλάσεων με σχετικά μικρό βάθος (ή αντίστοιχα μικρό *fan-out*) αναμένονται να μπορούν να ανανεωθούν με την προσθήκη καινούργιων "παιδιών" ή "αδελφών" κόμβων /κλάσεων. Συγκεκριμένα, είναι σαφώς πιο συχνό το φαινόμενο των αυξητικών ενημερώσεων ενδιάμεσων κλάσεων της ιεραρχίας από ότι "φύλλων"-κλάσεων. Δηλαδή, με άλλα λόγια τα λιγότερα αναπτυγμένα υποδέντρα θεμάτων είναι αντικείμενα οιοσδήποτε επιμέρους τροποποιήσεων σε αντίθεση με την εξέλιξη των δέντρων που έχουν παρατηρηθεί στα *XML* δεδομένα.



Σχήμα 8 : Κατανομή θεμάτων (topics) κατά βάθος για τι ιεραρχίες α) Regional, b) Reference και c) Society

Παραθέτουμε επίσης στο σχήμα 9 αναλυτικά στοιχεία με την ακριβή κατανομή ανά επίπεδο πλήθους κλάσεων για όλες τις ιεραρχίες του ODP καταλόγου**.** Σε βάθος ίσο με 0 θεωρούμε ότι υπάρχει η κλάση *Resource,* κάτω από την οποία βρίσκονται σε ιδεατό καθαρά ιεραρχικό επίπεδο στην ιεραρχία κλάσεων του όλου σχήματος του ODP καταλόγου οι μεμονωμένες κλάσεις που είναι οι επιμέρους ριζικές κλάσεις των σχημάτων, όπως είναι οι *Arts, Business, Computers* κλπ, που εμφανίζονται στον πίνακα στην στήλη των κόμβων με βάθος ίσο με 1.



Σχήμα 9: Αναλυτική Κατανομή των κλάσεων / επίπεδο βάθους

2.3 Αναγκαιότητα Βασικών Επερωτήσεων

2.3.1 Επισκόπηση Προβλήματος

Μέσω των υπαρχόντων κι ευρέως χρησιμοποιούμενων διαδικτυακών πυλών σήμερα, οι χρήστες μπορούν εναλλακτικά είτε να "πλοηγηθούν" μέσα στις υπάρχουσες θεματικές ιεραρχίες με στόχο να εντοπίσουν τις ακριβείς πόροι των ενδιαφερόντων τους, είτε να διαμορφώσουν σε μορφή κειμένου επερώτηση στα ονόματα των θεμάτων και στα *URIs* των περιγραφόμενων πηγών (είτε τις αλφαριθμητικές τιμές άλλων γνωρισμάτων όπως είναι οι τίτλοι, οι περιγραφές κλπ). Αναλυτικά, στην μεν πρώτη περίπτωση οι χρήστες πρέπει να οδηγηθούν δια μέσου αρχικά του ριζικού κόμβου της κάθε ιεραρχίας μέχρι τα "φύλλα" κόμβους, με σκοπό να προσεγγίσουν τις πόροι που ερίζουν τον ενδιαφέρον τους,

στηριζόμενοι στο γεγονός πως οι περισσότερες από τις πόροι έχουν οριστεί κάτω από τα θέματα των "φύλλων" κλάσεων. Στη δε δεύτερη περίπτωση, οι χρήστες αναγκάζονται χειροκίνητα να επεξεργαστούν τα επιστρεφόμενα θέματα και τις πόροι URIs από την εκτέλεση του επερώτησης που είχαμε θέσει. Στην βασική λειτουργία αυτή υπάρχουν σήμερα πολλές παραλλαγές διεπαφών χρήσης που επιτρέπουν προωθημένη πλοήγηση και εξίσου επερωτήσεις που σε κάθε περίπτωση επιχειρούν να την απλοποιήσουν, επιτρέποντας ομαλή πλοήγηση /επεξεργασία από κοινού στα σχήματα των διαδικτυακών πυλών και στις περιγραφές των πηγών.

Σε αυτήν την κατεύθυνση και με απώτερο γνώμονα να αποσκοπούμε στην διαμόρφωση κι λειτουργική υποστήριξη τέτοιων ενισχυμένων ως προς την λειτουργική τους απόδοση οι διαδικτυακές πύλες, χρειάζεται να αξιολογήσουμε και να βελτιστοποιήσουμε ένα πλήθος από βασικές επερωτήσεις στις ιεραρχίες των υποκλάσεων.

2.3.2. Μεταβατικότητα στην Αναπαράσταση Γνώσης

Η ανάγκη για ικανοποιητική πρόσβαση και διαχείριση μεγάλου όγκου σχέσεων οδηγεί στην ανάγκη για αποκλειστική αποθήκευση του μεταβατικού κλεισίματος (*transitive closure*) με τον πλέον συμπιεσμένο τρόπο αλλά και στο μέγιστο βαθμό σε τοπικό επίπεδο, για την εύρωστη διαχείριση των βάσεων δεδομένων και των συστημάτων κι που επιτρέπει την αυξητική ανανέωση της βασικής σχέσης.

Η χρήση των μεταβατικών συσχετίσεων στην άσκηση επερωτήσεων σε βάσεις δεδομένων είναι όντως πολύ μεγάλης έμπρακτης σημασίας κι ιδίως σε ότι αφορά τις βάσεις δεδομένων ως προς την επιμέρους διαχείριση και χρήση ιεραρχιών υποκλάσεων (*isA*) στα συστήματα αναπαράστασης γνώσης. Με την αυξανόμενη χρήση μεγάλης πλέον κλίμακας συστημάτων αναπαράστασης γνώσης ωθούμαστε κι επικεντρωνόμαστε στις δυαδικές μεταβατικές σχέσεις.

Τα σημασιολογικά δίκτυα καθώς και τα συστήματα πλαισίου είναι μεταξύ των πιο διάσημων τεχνικών αναπαράστασης γνώσης. Βασική τους αρχή η οργάνωση των περιγραφόμενων εννοιών σε ιεραρχίες υποκλάσεων (*subclass hierarchies*), επίσης γνωστών κι ως *isA* ιεραρχιών. Προσδιοριστικό των ιεραρχιών αυτών είναι η ύπαρξη της κληρονομικότητας (όπου κληρονομικότητα θεωρούμε την μεταβατική διάσχιση τέτοιων

ιεραρχιών). Με τον υπολογισμό της σχέσης υπαλληλίας (*subsumption*), μεταξύ μιας νέας ιδέας και των πρότερα γνωστών ιδεών προκύπτει το θεμελιώδες συμπέρασμα από τους λογικούς, περιγραφικούς αυτούς όρους, εάν δυο προαναφερθέντες γνωστές έννοιες βρίσκονται σε μια τέτοια σχέση "υπαλληλίας". Άρα η σχέση αυτή έχει προϋπολογισθεί και αποθηκευτεί προσωρινά σαν ιεραρχία, και η οποία εν συνεχεία μπορεί να διαχειριστεί ικανοποιητικά. Ένα χαρακτηριστικό γνώρισμα των συστημάτων αυτών είναι ότι οι καινούργιες έννοιες μπορούν να οριστούν ανά πάσα στιγμή, κι άρα η ιεραρχία μπορεί κατ' αυτόν τον τρόπο να αυξάνεται δυναμικά.

Ο χώρος αποθήκευσης τέτοιων εννοιών σε μια τέτοια γνωσιολογική βάση μπορεί να αυξηθεί αρκετά ώστε να πρέπει να διαχειριστεί με τον προαναφερθέντα τρόπο. Συγκεκριμένα, όλες οι ερωτήσεις σε ότι αφορούν το μεταβατικό κλείσιμο της σχέσης *isA*, πρέπει να ασκηθούν με άλλη τεχνική πέραν της σειριακής - τετριμμένης διερεύνησης δεικτών διευθυνσιοδότησης στην δομή αποθήκευσής τους, με σκοπό την ελάττωση για παράδειγμα του ρυθμού κίνησης, της πρόσβασης του δίσκου κλπ.

2.3.3 Προσεγγίσεις στη διαχείριση των Μεταβατικών Σχέσεων

Οι δυαδικές σχέσεις μπορούν να αναπαρασταθούν σαν κατευθυνόμενοι γράφοι, οι οποίοι τουλάχιστον στην περίπτωση των παραπάνω συστημάτων αναπαράστασης γνώσης είναι άκυκλοι. Έχουμε ήδη αναφέρει ότι εξίσου και στις δυο περιπτώσεις μεγάλων βάσεων δεδομένων καθώς και βάσεων αναπαράστασης γνώσης, είναι επιθυμητή στο μέγιστο δυνατό βαθμό η αποφυγή του υπολογισμού του μεταβατικού κλεισίματος της (*transitive closure*) κατά την στιγμή της εκτέλεσής του. Ωστόσο, η εκάστοτε τεχνική αποθήκευσης των υλοποιημένων μεταβατικών κλεισιμάτων οφείλει να ικανοποιεί πολλαπλές κι αλληλοσυγκρουόμενες απαιτήσεις, όπως αναφορικά είναι οι κάτωθι απαιτήσεις :

- Η αναζήτηση πρέπει να είναι αποδοτική και αρκετά γρήγορη, τουλάχιστον στην μέση περίπτωση
- Η αποθήκευση δεν μπορεί να αναλίσκεται άσκοπα, και κυρίως στην κύρια μνήμη. Γι' αυτό το λόγο και οι όποιες δημιουργούμενες, ενδιάμεσες runtime δομές

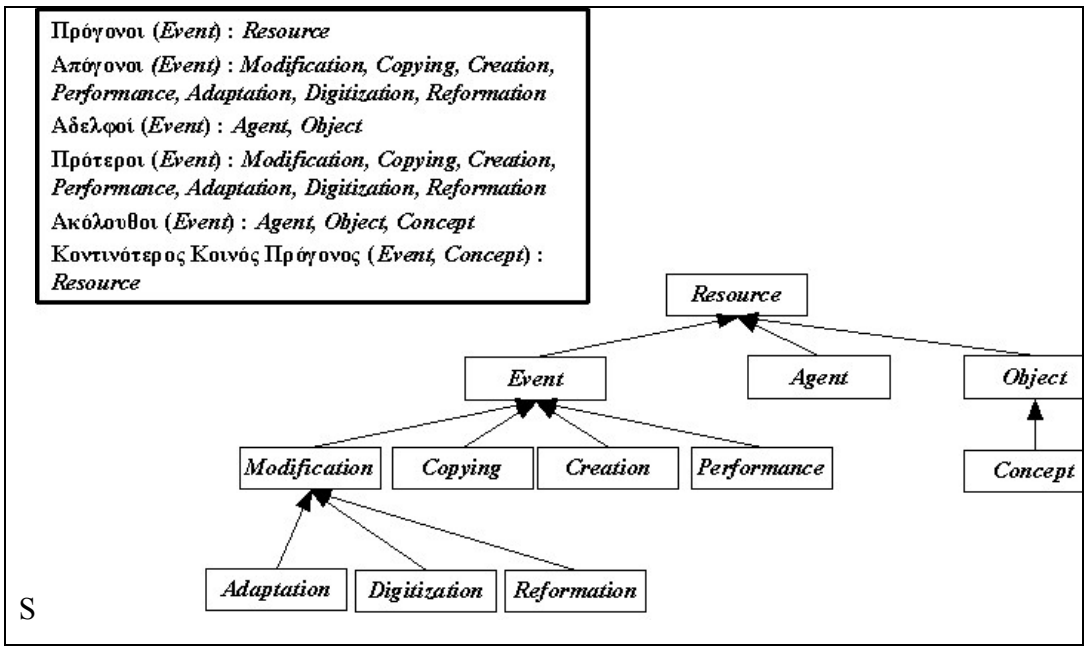
δεδομένων οι οποίες αποθηκεύουν εν μέρει ή πλήρως την υπάρχουσα μεταβατικό κλείσιμο θα πρέπει να είναι ικανοποιητικώς μικρά κι αποδεκτά.

- Στην περίπτωση μεγάλων σχέσεων , η πληροφορία μπορεί να βρίσκεται σε μια δευτερεύουσα μνήμη, και συνεπώς θα πρέπει ο ρυθμός κίνησης I/O μια την μεταφορά από τη δευτερεύουσα στην κύρια μνήμη κι αντιστρόφως να ελαχιστοποιηθεί στο μέγιστο δυνατό βαθμό.
- Όλες οι ενημερώσεις (τουλάχιστον οι προσθέσεις) στη βασική σχέση θα πρέπει να μην είναι μη συχνές, έτσι ώστε το καταγραφέν επαυξανόμενο κόστος της πρόσθεσης των καινούργιων κόμβων και των μεταξύ τους συσχετίσεων θα πρέπει να είναι πολύ μικρότερο από το αντίστοιχο κόστος του συνολικού επαναϋπολογισμού του μεταβατικού κλεισίματος.

Τα προσδοκώμενα αποτελέσματα στα δεδομένα καθιστούν ανεπίτρεπτες τις προφανείς προσεγγίσεις για διατήρηση του μεταβατικού κλεισίματος, όπως για παράδειγμα είναι οι διασυνδεδεμένες λίστες ή πίνακες των απογόνων , ή 2-διάστατοι Boolean πίνακες, καθώς η πρόσθεση των όλων μεταβατικά προκυπτόμενων σχέσεων μπορεί να αυξήσει το πλήθος των ακμών του γράφου από $O(n)$ σε $O(n^2)$. Συνεπώς, είναι αναγκαία μια πιο συμπιεσμένη και τοπική αναπαράσταση.

2.3.4 Ορισμοί Σχέσεων Διασύνδεσης Ιεραρχίας

Στο σημείο αυτό θα δείξουμε ένα χαρακτηριστικό, ρεαλιστικό παράδειγμα μιας *RDF/S* ιεραρχίας, της *MetaNet* και όλες τις δυνατές συσχετίσεις μιας κλάσης της, για παράδειγμα της *Event*, που μπορούμε να πάρουμε με τις βασικές επερωτήσεις, που θα δούμε εν συνεχεία (βλ. Σχήμα 10). Σε κάθε περίπτωση θα μπορεί, ξεκινώντας ένας χρήστης από ένα συγκεκριμένο θέμα στο σχήμα, θα μπορεί εύκολα να προσεγγίσει τους άμεσους "πατρικούς" και "παιδικούς" του κόμβους και ακόμα τους "φύλλα" κόμβους στην κάτω του πλευρά (όπου οι περισσότερες διαδικτυακοί πόροι είναι ορισμένοι), να ανακαλύψει εν συνεχεία τα συγγενικά – ομοεπίπεδα θέματα (όπου σχετικοί διαδικτυακοί πόροι μπορούν να βρεθούν) ή ακόμα να συνεχίσουν την πλοήγηση από το κοντινότερο κοινό απόγονο δυο συγκεκριμένων κλάσεων στην ιεραρχία.



Σχήμα 10 : Παράδειγμα Βασικών Σχέσεων Διασύνδεσης στην RDF/S ιεραρχία MetaNet

Μία δυαδική σχέση, η οποία αποτελείται από ένα πεδίο «αφετηρία» κι ένα πεδίο «προορισμός» κατάλληλα προσδιορισμένων στο ίδιο πεδίο επιμέρους τιμών, αντιστοιχεί σε έναν γράφο, όπου κάθε κόμβος αντιστοιχεί σε μία διακριτή τιμή των πεδίων της «αφετηρίας» και του «προορισμού» αντιστοίχως, καθώς και ένα κατευθυνόμενο τόξο για κάθε πλειάδα της σχέσης.

Αναλυτικά εάν θεωρήσουμε ότι υπάρχει μια ιεραρχία ταξινόμησης $H = \langle T, <: \rangle$, η οποία αποτελείται από μια σειρά τύπων T και μία αντανακλαστική, μεταβατική, αντιδιαμετρική σχέση υπαλληλίας (*subtype*) <:. Άρα εάν υφίσταται η σχέση A <: B, μπορούμε να θεωρήσουμε ότι η μεν κλάση A είναι υποκλάση της κλάσης B και ότι η κλάση B είναι υπερκλάση της A αντιστοίχως.

Ομοίως με την παραπάνω σχέση και επεκτείνοντάς την μπορούμε να ορίσουμε μια επίσης αντανακλαστική, μεταβατική, αντιδιαμετρική σχέση υπαλληλίας (subtype) $<:_d$, όπου:

$$<:_d \equiv \{\langle x \in T, \mathrm{y} \in T \rangle, \mid x <: \mathrm{y} \wedge (\nexists\, z \in T \mid x \neq z \wedge y \neq z \wedge x <: z <: y)\}$$

Τώρα με βάση τον ορισμό αυτής της επιμέρους σχέσης μπορούμε να καθορίσουμε εν συνεχεία τις εξής συσχετίσεις για τον κάθε κόμβο-κλάση της ιεραρχίας σε σχέση με την θέση του στην ιεραρχία και την ιδιαίτερη διασύνδεσή του με τους υπόλοιπους κόμβους :

- *Roots* $(T) \equiv \{x \in T \mid \nexists\, y \in T : x <: y \}$

Ως ριζικούς κόμβους μίας ιεραρχίας θεωρούμε τους κόμβους αυτούς μιας ιεραρχίας, οι οποίοι δεν έχουν με την σειρά τους υπερκλάσεις και συνεπώς δεν υπάρχουν κατευθυνόμενα τόξα διασύνδεσης (*directed arcs*) στον γράφο, όπως αυτός προκύπτει από την διάσχιση της εν λόγω ιεραρχίας, προς αυτούς.

- *Descendants* $(x) \equiv \{x \in T \mid\ y <: x \}$

Απόγονοι ενός κόμβου θεωρούμε το σύνολο των κόμβων οι οποίοι με την σειρά τους είναι προσβάσιμοι ξεκινώντας από τον εν λόγω κόμβο και διασχίζοντας τα κατευθυνόμενα τόξα διασύνδεσης του γράφου. Οι κόμβοι αυτοί αποτελούν υποκλάσεις της κλάσης *x*.

- *Children (x) – Direct Descendants* $\equiv \{x \in T \mid y <:_{\mathrm{d}} x \}$

Άμεσοι απόγονοι ενός κόμβου θεωρούμε το υποσύνολο των απογόνων του κόμβου αυτού και οι οποίοι έχουν την επιπρόσθετη ιδιαιτερότητα ότι είναι προσβάσιμοι με την σειρά τους μέσω ενός άμεσου κατευθυνόμενου τόξου του γράφου.

- *Ancestors (x)* $\equiv \{x \in T \mid x <: y \}$

Εν συνεχεία ως πρόγονους ενός κόμβου θεωρούμε το σύνολο των κόμβων του γράφου από τους οποίους μπορούμε μέσω της διάσχισης κατευθυνόμενων τόξων διασύνδεσης του γράφου να καταλήξουμε στον εν λόγω κόμβο. Οι κόμβοι αυτοί είναι υπερκλάσεις της κλάσης *x*.

- *Parents (x) – Direct Ancestors* $\equiv \{x \in T \mid x <:_{\mathrm{d}} y\}$

Άμεσοι πρόγονοι ενός κόμβου θεωρούμε το υποσύνολο των προγόνων του κόμβου αυτού, από τους οποίους είναι προσβάσιμος ο εν λόγω κόμβος με την σειρά τους, μέσω ενός ακριβώς και μόνο άμεσου κατευθυνόμενου τόξου του γράφου.

- *Leaves (x)* $\equiv \{x \in T \mid \nexists\, y \in T : y <: x\}$

Ως «φύλλα» κόμβους θεωρούμε το σύνολο των κόμβων αυτών της ιεραρχίας, οι οποίοι με την σειρά τους δεν έχουν άλλους επιμέρους κόμβους-κλάσεις της ιεραρχίας ως υποκλάσεις τους, δηλαδή δεν υπάρχουν κατευθυνόμενα τόξα διασύνδεσης στον γράφο από αυτούς παρά μόνο προς αυτούς.

- *Brothers (x)* $\equiv \{x \in T \mid \exists\, z \mid (x <:_{\mathrm{d}} z) \wedge (y <:_{\mathrm{d}} z) \wedge (y \neq x)\}$

«Αδελφούς» κόμβους ενός κόμβου θεωρούμε το σύνολο όλων αυτών των κόμβων οι οποίοι έχουν κοινό άμεσο πατρικό κόμβο, όπως έχουμε ορίσει πρότερα την έννοια του άμεσα πατρικού κόμβου και δεν είναι ο εν λόγω αυτός κόμβος.

- *Precedings(x)* $\equiv \{x \in T \mid \exists\, \mathrm{y} \in T \mid ((y <:_{\mathrm{d}} x) \wedge (y \neq x)\}$

«Πρότερους» κόμβους ενός κόμβου σε μία ιεραρχία θεωρούμε το σύνολο των κόμβων οι οποίοι προηγούνται κατά την *pre-order* διάσχιση της ιεραρχίας αυτής, και συνεπώς φέρουν ετικέτες που σε κάθε μορφή κωδικοποίησης υπολείπονται αριθμητικά της αντίστοιχης ετικέτας του κόμβου αυτού.

- *Followings(x)* $\equiv \{x \in T \mid \exists\, \mathrm{y} \in T \mid ((x <:_{\mathrm{d}} y) \wedge (y \neq x)\}$

«Ακόλουθους» κόμβους ενός κόμβου σε μία ιεραρχία θεωρούμε το σύνολο των κόμβων οι οποίοι έπονται του αυτού κόμβου κατά την *pre-order* διάσχιση της ιεραρχίας

αυτής, και συνεπώς φέρουν ετικέτες που σε κάθε μορφή κωδικοποίησης υπερέχουν αριθμητικά της αντίστοιχης ετικέτας του κόμβου αυτού.

Επιπρόσθετα στο σύνολο των ορισμών αυτών να αναφέρουμε και το cardinality δηλαδή τον αριθμό του εκάστοτε εξεταζόμενου συνόλου των στοιχείων. Μάλιστα με βάση τον ορισμό αυτό μπορούμε να διαφοροποιήσουμε τις εκάστοτε εξεταζόμενες ιεραρχίες σε *μονής κληρονομικότητας* ιεραρχίες, οι οποίες έχουν τον περιορισμό ότι ο αριθμό των άμεσων πατρικών κόμβων τους είναι ίσος με 1, cardinality(Parents(x)) ≤ 1, και σε *πολλαπλής κληρονομικότητας*, όπου κατά αντιστοιχία έχουν αριθμό άμεσων πατρικών κόμβων μεγαλύτερο του 1, *cardinality(Parents(x))* ≥ 1.

Κεφάλαιο 3

Οικογένειες Σχημάτων Κωδικοποίησης Δέντρων και Γράφων

Στο κεφάλαιο αυτό θα μελετήσουμε μια σειρά από διάφορα σχήματα κωδικοποίησης, που χρησιμοποιούνται τόσο για δέντρα όσο και για γράφους, και οι οποίοι έχουν προταθεί στα πλαίσια διαφόρων εφαρμογών, όπως είναι η δρομολόγηση δικτύων [GP03], ο οντοκεντρικός προγραμματισμός [D82, DS87, W88, KBL89, C93, KVH97], τα συστήματα αναπαράστασης γνώσης [ABJ89, E97] και προσφάτως οι διαδικτυακές μηχανές αναζήτησης δεδομένων με την χρήση της XML (*Web search engines*) [CSFH01, ZNWLL, MACM01, LM01, SKM02, CKM02, CVZTZ02, TVBS02].

Στη συνέχεια θα μελετήσουμε διαφορετικές οικογένειες κωδικοποίησης και παραλλαγών τους με βάση τα παρακάτω σημαντικά χαρακτηριστικά τους :

- Η δομή των κωδικοποιημένων ιεραρχικών δεδομένων (π.χ. δέντρα, DAGs, ημιπλέγματα αναζήτησης).
- Οι υποστηριζόμενες λειτουργίες, δηλαδή η ικανότητα αναζήτηση προγόνων, απογόνων, φύλλων, ιεραρχικά πρότερων, επόμενων και ομοεπίπεδων κόμβων, όπως επίσης και τους κοντινότερους κοινούς πρόγονους /απόγονους
- Η πολυπλοκότητα των αλγορίθμων που χρησιμοποιούνται για την παραγωγή των κωδικών
- Το μέγεθος των παραγόμενων κωδικών, το οποίο προσδιορίζεται είτε από το μέγιστο μέγεθος των κωδικών εάν το επιλεγέν σχήμα κωδικοποίησης διαμορφώνει σταθερού μεγέθους αναπαράσταση (*fixed size*), είτε από το μέσο μέγεθος στη περίπτωση μεταβλητού μεγέθους αναπαράστασης (*variable size*).

- Ο χρόνος αξιολόγησης των διάφορων επερωτήσεων πάνω στους κωδικούς, χρησιμοποιώντας κατάλληλες δομές δεδομένων για την σχετική αποθήκευση κι χρήση τους.
- Οι συνέπειες στο πλήθος των αναγκαίων επαναϋπολογισμών ετικετών που συνεπάγονται ενημερώσεις στις προαναφερθείσες δομές δεδομένων.

Εν συνεχεία, θα σας παρουσιάσουμε με πάσα λεπτομέρεια τρεις βασικές κατηγορίες κωδικοποίησης και αντιπροσωπευτικά τους παραδείγματα, που είναι οι αλγόριθμοι κωδικοποίησης με αναπαράσταση των κωδικών σε διανύσματα δηφίων (*Bit-Vector*), οι αλγόριθμοι κωδικοποίησης με χρήση προθέματος για την παραγωγή των κωδικών (*Prefix-based*), και τέλος οι αλγόριθμοι κωδικοποίησης με χρήση διαστημάτων /πεδίων (*Interval-based*).

3.1 Αλγόριθμοι με αναπαράσταση κωδικών σε διάνυσμα δηφίων (Bit-Vector)

Στην κατηγορία αυτή σχημάτων κωδικοποίησης βασική αρχή είναι ότι ο κάθε κόμβος αναπαρίσταται μέσω μιας ετικέτας με την μορφή ενός δυαδικού διανύσματος, Στην διανυσματική αυτή αναπαράσταση η ύπαρξη ενός δηφίου "1" σε κάποια συγκεκριμένη θέση υποδηλώνει μοναδικά τον κάθε κόμβο σε ένα πλέγμα *D*, καθώς και κάθε κόμβος κληρονομεί όλες τις θέσεις των δηφίων αυτές των άμεσων προγόνων του – πατρικών (απογόνων) του σε μία *top-down* (ή *bottom-up* αντίστοιχα) διαδικασία κωδικοποίησης.

Το πλέον αντιπροσωπευτικό σχήμα διανυσματικής κωδικοποίησης είναι, όπως θα δούμε κι αναλυτικά εν συνεχεία, το σχήμα του *Wirth* [W88]. Σε αυτό, ο κωδικός ενός κόμβου *u* του γράφου *D*, έστω *l(u)* συντίθεται από το σύνολο των bits $\{b_1,b_{2,\ldots,}b_{n,}\}$, όπου *n* = μέγεθος του υπό εξέταση ημιπλέγματος και τα επιμέρους *bits* b_i προσδιορίζονται από το δυαδικό σύνολο αναπαράστασης {0,1}. Συγκεκριμένα, το b_i ισούται με την τιμή 1 εάν ο *i*-οστος κόμβος είναι είτε ο *u* είτε ένας πρόγονος (εναλλακτικά απόγονος) έστω *v* του *u* ανάλογα με την διαδικασία κωδικοποίησης. Σε κάθε άλλη περίπτωση έχουμε ότι $b_i = 0$.

Συνεπώς, εκ του σημείου αυτού μπορεί να διαπιστωθεί με την χρήση των δυαδικών τελεστών OR (|) και AND (&) στις ετικέτες, εάν ένας κόμβος *ν* είναι πρόγονος ή εναλλακτικά απόγονος ενός κόμβου *u* σε έναν γράφο *D*. Αναφορικά, ισχύει ότι $u < \nu$ εάν και μόνο εάν $l(\nu) \,\&\, l(u) = l(u)$ (αντίστοιχα $l(\nu) \mid l(u) = l(u)$). Ομοίως μπορούν να υπολογιστούν οι "κοντινότεροι κοινοί πρόγονοι /απόγονοι" (*Least Upper Bound - LUB/ Greatest Lower Bound - GLB*), χρησιμοποιώντας αντίστοιχα τις πράξεις του δυαδικού OR (|) και AND (&) στους κωδικούς. Το σχήμα αυτό υποστηρίζει τους ελέγχους ύπαρξης προγόνων /απογόνων και των υπόλοιπων λειτουργιών πλέγματος (*lattice*) σε σταθερό χρόνο (και ίσο με τον χρόνο σύγκρισης δυο επιμέρους δυαδικών διανυσμάτων), καθώς και η κατασκευή των κωδικών των κόμβων σε γραμμικό χρόνο. Εδώ θα πρέπει να αναφερθεί ότι όλοι οι κωδικοί έχουν σταθερό μέγεθος έστω *n* bits, και ο αναγκαίος χώρος για την αποθήκευση των κωδικών ενός πλέγματος *D* είναι ακριβώς n^2,όπου *n* είναι το πλήθος των κόμβων στον *D*.

Όπως θα δούμε υπάρχουν και νέες, πιο συμπιεσμένες τροποποιήσεις των αλγορίθμων δυαδικών διανυσμάτων κωδικοποίησης, που αποσκοπούν σε κάθε περίπτωση στην χρήση νέων θέσεων δηφίων μόνο όταν είναι αυτό αναγκαίο, κάτι που γίνεται σαφές όταν απαιτείται η διαφοροποίηση μεταξύ κόμβων με κοινούς απογόνους. Το σχήμα που προτείνει ο *Near Hierarchical Encoding* (*NHE*) [KVH97] το ολικό μέγεθος των παραγόμενων δυαδικών διανυσμάτων είναι $2*n*logn$ για την περίπτωση που έχουμε ισορροπημένα δέντρα και αντιστοίχως *logn* όταν το ποσοστό της υπάρχουσας πολλαπλής κληρονομικότητας είναι σχετικά χαμηλό. Όμως η επιχειρούμενη αυτή συμπίεση έχει σαν αποτέλεσμα την μη υποστήριξη όλων των επιθυμητών λειτουργιών που θα θέλαμε διαχείρισης των κωδικών. Το σχήμα του *Caseau* [C93] υποστηρίζει μόνο τον έλεγχο ύπαρξης απογόνων /προγόνων, ενώ από την άλλη πλευρά ο *NHE* υποστηρίζει τις προαναφερθείσες λειτουργίες πλέγματος (*GLB, LUB*) και την κωδικοποίηση των αυθαίρετα μερικών διατεταγμένων συνόλων, από ότι των απλών πλεγμάτων στον *Caseau*. Το σχήμα του *Ait-Kaci* [KBLN89] υποστηρίζει όλες τις προαναφερθείσες λειτουργίες, αλλά δημιουργεί κωδικούς μεγέθους της τάξεως O(*logn*) και O(*n*) στην καλύτερη και στην χειρότερη περίπτωση αντίστοιχα.

Το κύριο μειονέκτημα αυτής της οικογένειας σχημάτων κωδικοποίησης έγκειται στο ότι η εκτέλεση των επερωτήσεων για τη εύρεση των "προγόνων", "απογόνων",

"αδελφών" κόμβων απαιτεί την πρόσβαση κι η σύγκριση με όλους τους παραγόμενους κωδικούς και δεν αρκεί η μονή διερεύνηση μέσω του κωδικού μιας κλάσης εάν είναι ή δεν είναι υποκλάση μιας άλλης κοκ. Άρα ο χρόνος εκτέλεσής τους είναι της τάξης *O(n)*, γεγονός που δεικνύει με το πλέον κατηγορηματικό τρόπο πως μπορούν να αποτιμηθούν μόνο μέσω σειριακών σαρώσεων σε μία βάση δεδομένων, όπου θα είναι αποθηκευμένα, και δεν υπάρχει δομή δεδομένων που να είναι ικανή να αντιμετωπίσει αυτές τις επερωτήσεις έχοντας απόδοση *O(logn)*, που συνεπάγεται τον ορισμό εξ αρχής και χρήσης κατόπιν B - δέντρων στην δομή αποθήκευσης με τους παραγόμενους κωδικούς. Επιπρόσθετα, το σταθερό μέγεθος των παραγόμενων κωδικών σε κάθε περίπτωση εξαρτάται εντόνως από το μέγεθος αλλά και από την μοναδική μορφολογία των εισαγόμενων σχημάτων, καθιστώντας την κωδικοποίηση του δυαδικού διανύσματος μη απόλυτα ικανή να εφαρμοστεί στην υλοποίηση μιας δυναμικής βάσης δεδομένων που ποσώς χαρακτηρίζεται από την παρουσία συχνών αυξητικών ενημερώσεων.

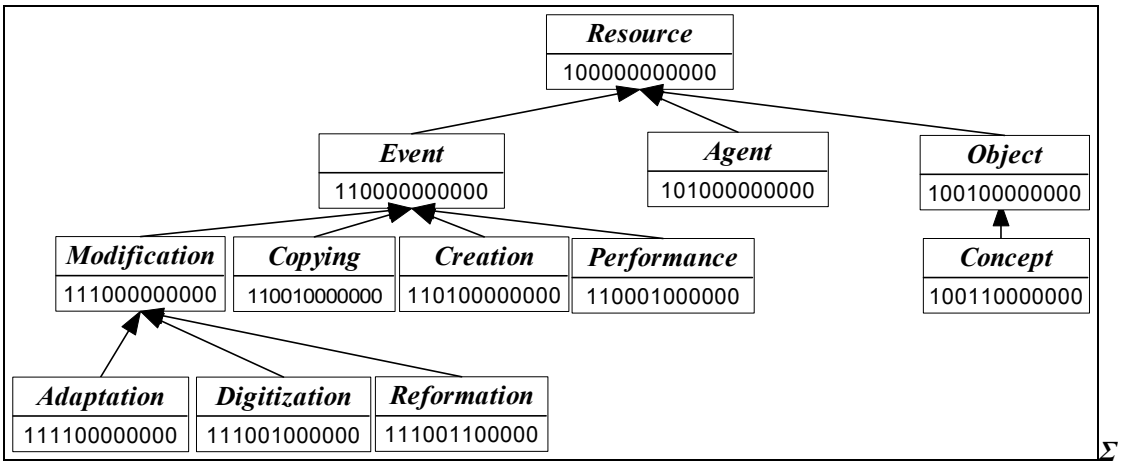
3.1.1 Κλασσικός Δυαδικός Πίνακας - Binary Matrix (Αλγόριθμος Wirth)

Η βασική ιδέα που προτάσσει ο αλγόριθμος του *Wirth* [W88] έγκειται στην κωδικοποίηση της ύπαρξης της σχέσης υπαλληλίας μεταξύ δυο κλάσεων μέσω της αναπαράστασής της και καταγραφής της σε ένα δυαδικό πίνακα. Παρατηρούμε ότι η βασική λογική του προέρχεται από την υιοθέτηση της αρχής κωδικοποίησης της προσέγγισης του απλού μεταβατικού κλεισίματος, ενσωματώνοντας σε κάθε κόμβο μαζί με την διακριτή, δυαδική διανυσματική αναπαράστασή του και το σύνολο όλων των «*κατώτερων*» ιεραρχικά ορίων του.

Συγκεκριμένα, θεωρούμε ότι ο αλγόριθμος βασίζεται σε μια συνάρτηση αναπαράστασης $\gamma : T \rightarrow [1 .. N]$, η οποία εκτελεί μία "1 προς 1" αντιστοίχηση τύπων και δεικτών αντίστοιχα. Το τελικό αποτέλεσμα της διαδικασίας αυτής είναι τελικά ότι κατασκευάζουμε ένα $N \times N$ δυαδικό πίνακα M_{BM} έτσι ώστε:

$$M_{BM} = [\gamma(\chi), \gamma(y)] \equiv \begin{cases} 1 & ,\text{εάν } x <: y, \\ 0 & ,\text{αλλιώς} \end{cases}$$

Στην συνέχεια παραθέτουμε ένα χαρακτηριστικό παράδειγμα της κωδικοποίησης που προσφέρει ο αλγόριθμος του *Wirth* πάνω στην αρχικά υπό εξέταση ιεραρχία, όπως φαίνεται στο σχήμα 11 .



Σχήμα 11: Κλασσικός Δυαδικός Πίνακας – Αλγόριθμος Wirth

Με βάση τους παραπάνω παραγόμενους κωδικούς του αλγορίθμου του *Wirth*, η εν λόγω κωδικοποίηση πραγματοποιείται μέσω μιας τετριμμένης και καθ' όλα εύκολης διαδικασίας, η οποία σίγουρα προσφέρει την δυνατότητα επιπλέον αυξητικής ενημέρωσης με νέους κωδικούς για τους νέους κόμβους, που πλαισιώνουν τους ήδη υπάρχοντες κωδικούς και κόμβους. Κυρίως η απόλυτα διακριτή απλότητα της έγκειται στο ότι βασίζεται στην απλότητα που συνεπάγεται από την αναπαράσταση και κατόπιν την χρήση του δυαδικού πίνακα.

Ωστόσο, όμως διακρίνεται και από την ανάγκη της για αρκετό χώρο αποθήκευσης για τους κωδικούς της. Συγκεκριμένα, η κωδικοποίηση απαιτεί τετραγωνικού βαθμού επιμέρους χώρο αποθήκευσης για το σύνολο των αποδοθέντων κωδικών για όλους τους κόμβους. Γι αυτό κι έχει διαπιστωθεί πολλάκις ότι σε διάφορες μεγάλες εφαρμογές, που οδηγούνται στην κατασκευή τέτοιων δυαδικών πινάκων που προσεγγίζουν την τάξη μεγέθους του ενός *Megabyte*.

Επιπρόσθετα, η κωδικοποίηση διακρίνεται για την εύκολη ικανότητα παροχής ελέγχου ύπαρξης σχέσης κλάσης – υποκλάσης μεταξύ των κόμβων της ιεραρχίας. Αναλυτικά, κάθε ένας τέτοιος έλεγχος συντίθεται από 3 επιμέρους υποδιαδικασίες, που είναι με την σειρά αναφοράς : 1) απλή πρόσβαση ενός πίνακα 2) μια αλλαγή θέσης

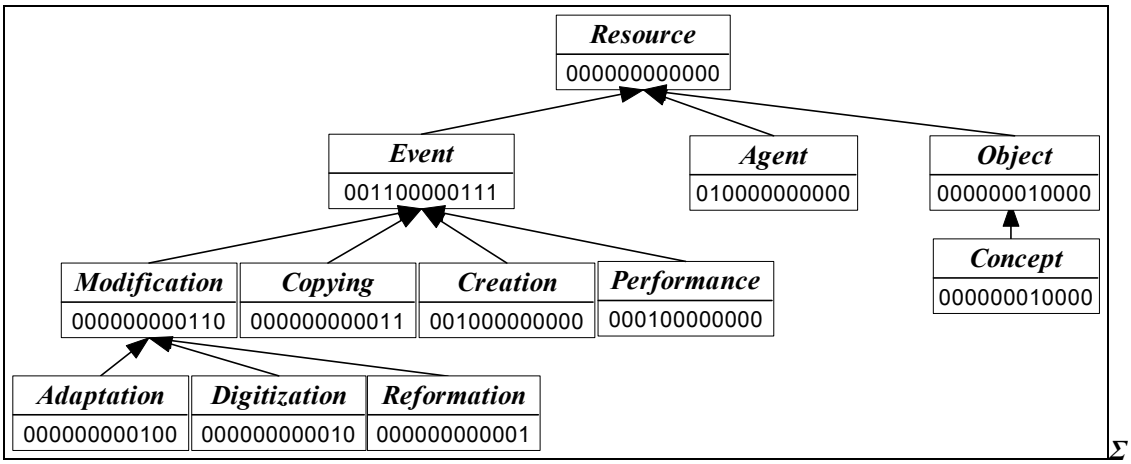
δηφίου (bit) και τέλος 3) μια σύγκριση. Όμως οι έλεγχοι αυτοί είναι καθ' όλα αργοί και εκτελούνται σε χρόνο απόλυτα ανάλογο με την επιμέρους απόσταση που διαχωρίζει τους εκάστοτε εξεταζόμενους κόμβους της ιεραρχίας. Υπάρχει ακόμα η δυνατότητα για την αναπαράσταση ύπαρξης πολλαπλής κληρονομικότητας, η οποία σαφώς κι έχει ένα ακόμα επιπρόσθετο κόστος εκτέλεσης και περάτωσης του ελέγχου ύπαρξης σχέσεως κλάσης - υποκλάσης.

Η προσέγγιση της κωδικοποίησης με βάση τον αλγόριθμο του *Wirth* ήταν μια πρώτιστη προσπάθεια που χρησιμοποιήθηκε εν αρχή ώστε να δώσει εν γένει την βασική πλατφόρμα, που είναι η βασική χρήση του δυαδικού πίνακα, σε καινούργιες προτάσεις – κωδικοποιήσεις που θα θεωρηθούν ως συμπιεσμένες εκδόσεις αναπαράστασης και χρήσης του. Βασικός γνώμονας υιοθέτησης της εκάστοτε νέας συμπιεσμένης έκδοσης του δυαδικού πίνακα είναι ότι θα εγγυάται εξ αρχής την βέλτιστη απόδοση της συμπίεσης σε ότι αφορά τον αναγκαίο χρόνο τυχαίας πρόσβασης στους επιμέρους κόμβους.

3.1.2 Αλγόριθμος Hasse – (Κωδικοποίηση Απλού μεταβατικού κλεισίματος)

Η κωδικοποίηση αυτή που προσδιορίζει ο αλγόριθμος του *Hasse* [E97] στηρίζεται με τον πλέον απλό τρόπο πάνω στις πολύ βασικές αρχές του απλού αυτοπαθούς και μεταβατικού κλεισίματος, μέσω της δυαδικής αναπαράστασης των προσφερόμενων κωδικών. Η απόδοση των κωδικών στους κόμβους-κλάσεις στηρίζεται πάνω στην κατάλληλη επιμέρους διάταξή τους, η οποία έγκειται στην διαμορφωθείσα σχέση «*αμέσως μεγαλύτερο από*». Με βάση την σχέση αυτή μπορούμε να διαμορφώσουμε έναν τετραγωνικό πίνακα μεγέθους $n \times n$, όπου n θεωρούμε το πλήθος των κόμβων της ιεραρχίας στον οποίο σε κάθε σημείο της γραμμής του υπάρχει τιμή ίση με 1 μόνο εάν σε αυτές τις στήλες υπάρχουν κόμβοι-κλάσεις που ακολουθούν την ιεραρχική σχέση του «*αμέσως μικρότερο από*» τον εν λόγω κόμβο που είναι στην αρχή της γραμμής αυτής. Στην αντίθετη περίπτωση περιέχει 0 σε αυτές τις θέσεις που προείπαμε. Ακολουθεί ένα χαρακτηριστικό παράδειγμα της κωδικοποίησης αυτής (βλ. Σχήμα 12), όπου διαφαίνεται γλαφυρά η σχέση αυτή μεταξύ των κόμβων. Άρα κάθε

στήλη αυτού του πίνακα αποτελεί για τον κόμβο αυτό στην αρχή της μία χαρακτηριστική δυαδική διανυσματική αναπαράσταση του όλου συνόλου των αυστηρώς άμεσων μικρότερων ορίων του.



Σχήμα 12: Αλγόριθμος Hasse – Κωδικοποίηση Απλού Μεταβατικού Κλεισίματος

Η εν λόγω κωδικοποίηση δεδομένου ότι βασίζεται στη θεμελιώδη αρχή του αλγορίθμου του *Hasse* καθίσταται πολύ πιο απλή, γεγονός που επιτρέπει πολύ χαμηλής πολυπλοκότητα και χαμηλή δυσκολία στις επιμέρους ενέργειες διαχείρισης των κόμβων - κλάσεων. Όμως, όπως προφανώς περιμένουμε, οι παραγόμενοι κωδικοί σε κάθε περίπτωση έχουν μέγεθος το οποίο άμεσα εξαρτάται από το πλήθος όλων των κόμβων της κάθε φοράς υπό κωδικοποίησης ιεραρχίας, χωρίς να επιχειρείται η όποια δυνατή συμπίεση. Συνεπώς, εκ τη απουσίας της όποιας επιχειρούμενης συμπίεσης, υπάρχει ένα υπερβολικό κόστος τόσο στην απόδοση –διαδικασία κωδικοποίησης - και αποθήκευσης των κωδικών σε μια δομή, όσο μετέπειτα και στη εν συνόλω διαχείριση κι επεξεργασία τους.

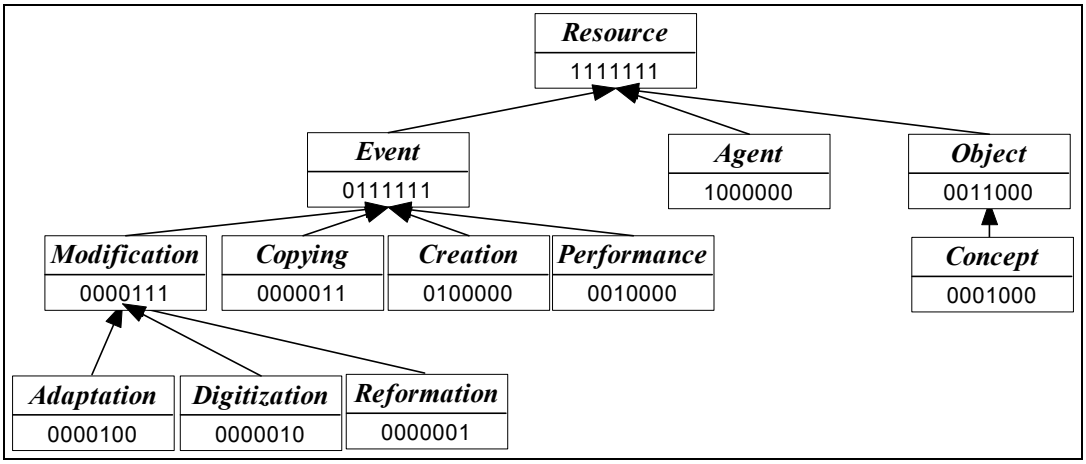
Κάθε κόμβος έχει κωδικό, ο οποίος προκύπτει από την διενέργεια της πράξης του δυαδικής διάζευξης (*binary OR*) μεταξύ των κωδικών των άμεσων απογόνων του – "παιδιών" του. Επίσης υποστηρίζει, εν γένει, την λειτουργία του *GLB*, χωρίς όμως να εγγυάται πάντοτε την απόδοση ενός υπάρχοντος κωδικού που σαφώς να αντιστοιχεί σε κάποιο κόμβο της εν λόγω εξεταζόμενης ιεραρχίας. Αντιθέτως, η αντίστροφη συνάρτηση επιστρέφει πάντοτε τους δυο αυτούς κόμβους που εισάγονται στην αρχική λειτουργία του *GLB,* δηλαδή εύρεσης των μέγιστων κοινών κατώτερων ορίων του αποτελέσματος της *GLB* λειτουργίας.

Οποιοδήποτε πλήθος ενεργειών αυξητικής ενημέρωσης της κωδικοποιημένης ιεραρχίας, είτε μέσω της εισαγωγής, είτε μέσω της διαγραφής κόμβων έχει ακριβώς ανάλογη επιρροή στην αναδιοργάνωση της ιεραρχίας και συγκεκριμένα στην αναγκαία επανακωδικοποίηση όλων των κόμβων εκείνων, που βρίσκονται «*κάτω*» ιεραρχικά του προστιθέμενου /αφαιρούμενου κόμβου κάθε φορά..

3.1.3 Κωδικοποίηση Αλγορίθμου Ait-Kaci

Ο αλγόριθμος κωδικοποίησης του *Ait-Kaci* [AKBLN89], είναι μία συμπιεσμένη έκδοση του κλασσικού μεταβατικού κλεισίματος, εισάγοντας και χρησιμοποιώντας καινούργιες θέσεις δηφίων (*bit*) εάν κι όπου αυτό κρίνεται αναγκαίο για την ελάχιστη, πλην όμως επαρκή διαφοροποίηση των προκυπτόμενων κωδικών. Η παραγόμενη κωδικοποίηση είναι αρκετά συμπαγής συγκρινόμενη άμεσα με την κλασσική κωδικοποίηση του μεταβατικού κλεισίματος, υπό την έννοια ότι προσθέτονται νέες θέσεις δηφίων (*bit*) όταν χρειάζεται να διαφοροποιηθούν κόμβοι της ιεραρχίας που έχουν κοινούς απογόνους μεταξύ τους. Αυτό το βασικό χαρακτηριστικό τους γνώρισμα έχει σαν συνεπακόλουθο αποτέλεσμα μια κωδικοποίηση η οποία εκτελείται σε σταθερό χρόνο και επιτρέπει πολύ ικανοποιητικούς *lattice* υπολογισμούς, όπως χαρακτηριστικά καλούνται οι υπολογισμοί *GLB, LUB* και της διερεύνησης υπαλληλίας (*subsumption*), οι οποίοι εκτελούνται χρησιμοποιώντας τις πράξεις της δυαδικής σύζευξης (*binary AND*), δυαδικής διάζευξης (*binary OR*) και της σύγκρισης αντίστοιχα.

Η διαδικασία της κωδικοποίησης με βάση τον αλγόριθμο του *Ait-Kaci* είναι καθ' όλα *bottom-up*, κι έτσι κωδικοποιούνται πρώτα οι κόμβοι – «φύλλα» στα κατώτερα επίπεδα της ιεραρχίας. Στο παρακάτω σχήμα 13 ακολουθεί ένα χαρακτηριστικό παράδειγμα της κωδικοποίησης αυτής για την ιεραρχία.



Σχήμα 13: Αλγόριθμος Κωδικοποίησης Ait-Kaci

Ένα άλλο χαρακτηριστικό γνώρισμα της μεθοδολογίας αυτής είναι η στατική φύση της κωδικοποίησης, η οποία παρουσιάζει ένα πολύ μεγάλο, επιπρόσθετο χρονικό κόστος και άρα καθυστέρηση στην όλη περάτωση της κωδικοποίησης ιδιαζόντως για μεγάλου πληθυσμού κόμβων ιεραρχίες. Συνεπώς, η κωδικοποίηση αυτή δεν διακρίνεται έπ' ουδενί για την δυνατότητά της για εν συνεχεία αυξητικές ενημερώσεις πάνω στην υπάρχουσα ιεραρχία και άρα συνεπώς αποτυγχάνει εκ του προοιμίου να μπορεί να αναπαραστήσει επαρκώς εφαρμογές, οι οποίες αναπτύσσονται καθ' όλη την διάρκεια του χρόνου από την κορυφή προς τα κάτω της ιεραρχίας (*top-down*).

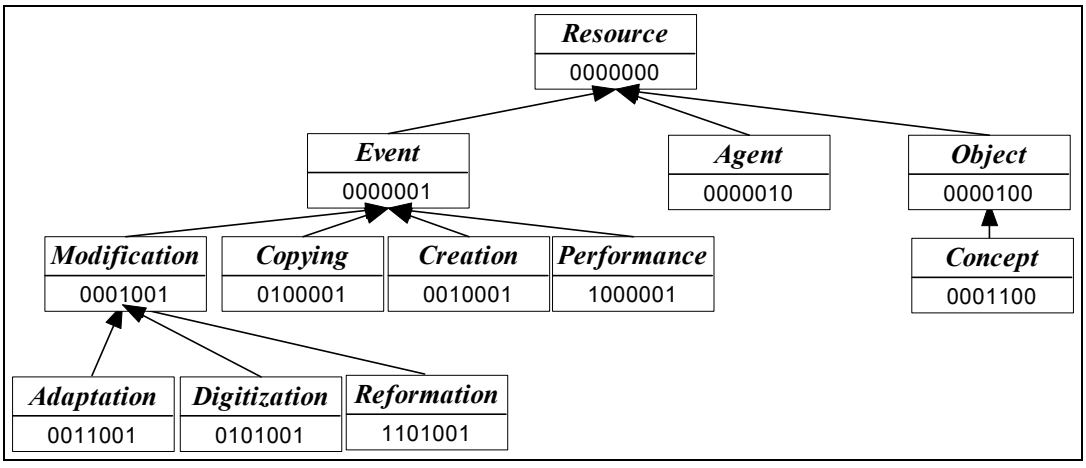
Έχουμε εν συνεχεία να αναφέρουμε πως η εν λόγω μεθοδολογία παρουσιάζει χαρακτηριστικά απόδοσης μεγέθους *O(logn)* στην καλύτερη περίπτωση και *O(n)* στην χειρότερη περίπτωση, όπου *n* θεωρούμε το πλήθος των κόμβων στην ιεραρχία που εξετάζουμε κάθε φορά.

3.1.4 Κωδικοποίηση Αλγορίθμου Caseau

Η κωδικοποίηση που προτείνεται μέσω του αλγορίθμου του *Caseau* [Cas93] είναι ένα κλασσικό παράδειγμα top-down κωδικοποίησης, η οποία εφαρμόζεται σε ένα κατάλληλο διαμορφωθέν πλέγμα (*lattice*), όπως αυτό έχει προκύψει από τον μετασχηματισμό της ιεραρχίας. Η απαίτηση αυτή για μετασχηματισμό της ιεραρχίας σε

lattice σίγουρα έχει σαν επιβάρυνση την προσθήκη νέων επιπρόσθετων κόμβων μέσα στην ιεραρχία, μπορεί και εκθετικού αριθμού απαιτώντας αντιστοίχως και εκθετικό χρόνο εκτέλεσης. Ανήκει κι αυτή στις κωδικοποιήσεις που εκτελούνται σε σταθερό χρόνο (*compile-time*) και άρα έχει στατική φύση, κάτι που στην οποιαδήποτε προσπάθεια ενημέρωσης της ιεραρχίας, μέσω της εισαγωγής /διαγραφής ενός κόμβου να οδηγήσει σε ολικό επανακωδικοποίηση την ιεραρχία με προφανές αντίστοιχο κόστος κι επιβάρυνση. Η απόδοση κωδικών σαν διαδικασία αποτελείται από το στάδιο του εντοπισμού των «πρωτευόντων» κόμβων της ιεραρχίας – όπως ορίζονται οι κόμβοι αυτοί με τους μοναδικούς άμεσους πατρικούς κόμβους – και εν συνεχεία στην ανάθεση μίας θέσης δηφίου (*bit*) σε κάθε ένα από αυτούς ξεχωριστά. Η επιλογή της θέσης του δηφίου (*bit*) για κάθε κόμβο σχετίζεται με την θέση που έχει ο κάθε κόμβος στην ιεραρχία σε άμεση σχέση με τους προγόνους του. Ενδέχεται να εμφανιστεί περίπτωση όπου να υπάρξει ανάγκη για τροποποίηση σε επιλεγμένες θέσεις δηφίων (*bits*) καθώς θα έχουν εμφανιστεί συμπτώσεις με την προσθήκη ενός καινούργιου κόμβου.

Στη συνέχεια ακολουθεί ένα χαρακτηριστικό παράδειγμα εφαρμογής της κωδικοποίησης του *Caseau* στο σχήμα 14, όπου εμφανίζεται το προαναφερθέν φαινόμενο αυτό.



Σχήμα 14: Κωδικοποίηση Αλγορίθμου Caseau

Το πλέον όμως ενδεικτικό χαρακτηριστικό γνώρισμα της κωδικοποίησης του *Caseau* είναι ότι επιτελεί μια συμπιεσμένη δυαδική κωδικοποίηση – πολύ μεγαλύτερου βαθμού από ότι της συμπίεσης του αλγορίθμου του *Ait-Kaci* – με βέβαια το

συνεπακόλουθο κόστος ότι δεν υποστηρίζονται οι βασικές *lattice* πράξεις του *GLB* και *LUB*, παρά μόνο η λειτουργία της υπαλληλίας (*subsumption*). Άρα δεδομένου της μη δυνατότητας υποστήριξης των λειτουργιών αυτών, δεν υφίσταται το όποιο πρόβλημα για απόλυτη μοναδικότητα των αποδοθέντων για τον κάθε κόμβο θέσης και τιμής δηφίου (*bit*), αλλά αντιθέτως οι θέσεις των δηφίων (*bits*) επαναχρησιμοποιούνται σε κόμβους, όταν δεν φαίνεται να σχετίζονται με άλλες κληρονομούμενες θέσεις. Σαφώς και η συμπίεση που υιοθετείται έχει σαν αποτέλεσμα την παραγωγή πιο μικρών κωδικών, οι οποίοι χρειάζονται λιγότερο αναγκαίο χώρο αποθήκευσης, ο οποίος είναι άμεσα εξαρτώμενος από το πλήθος των κόμβων, καθώς και από την μορφολογία της διασύνδεσής τους στην ιεραρχία.

Επίσης στην περίπτωση που υπάρχει πολλαπλή κληρονομικότητα στην ιεραρχία μας και στο εν λόγω κόμβο, τότε αυτό δεν σηματοδοτεί ότι ο κόμβος αυτός κληρονομεί τα διακριτά και απολύτως χαρακτηριστικά δηφία θέσης των προγόνων του, παρά μόνο του πρωτεύοντος πατρικού του κόμβου, όπως και προείπαμε σχετικά. Συνεπώς, εκ του χαρακτηριστικού αυτού προκύπτει ότι η εν συνόλω κωδικοποίηση, η οποία παράγεται για κάθε μία υπό εξέταση ιεραρχία είναι πολύ πιο συμπαγής από ότι θα μπορούσε να παραχθεί με την εφαρμογή του αλγορίθμου του *Ait-Kaci* αντιστοίχως.

3.1.5 Near-Optimal Hierarchical Encoding (NHE)

Στο σημείο αυτό θα παρουσιάσουμε μία εναλλακτική κωδικοποίηση η οποία είναι αντίστοιχη στην προηγηθείσα κωδικοποίηση που προτείνει ο *Caseau*, καθώς παρομοίως ακολουθεί την από πάνω προς τα κάτω (*top-down*), καθώς και την κατά βάθος επιμέρους διαδικασία διάσχισης της ιεραρχίας. Η ουσιαστική όμως διαφοροποίηση της καινούργιας αυτής εναλλακτικής κωδικοποίησης έγκειται στο ότι δεν προαπαιτεί η εισαγόμενη ιεραρχία να έχει την δομή πίνακα, αλλά μπορεί να δεχτεί το οποιοδήποτε μερικώς διατεταγμένο σύνολο (*partially ordered set*).

Με αυτόν τον τρόπο αποφεύγουμε τον κάθε περιορισμό που φέρουν οι δομές των πινάκων και εν συνεχεία αποφεύγουμε την εκθετική συμπεριφορά της διαδικασίας συμπλήρωσης των πινάκων. Κι αυτό γίνεται εξαλείφοντας τον κάθε περιορισμό με την

συσχέτιση κάθε κόμβου της ιεραρχίας με ένα διακριτό «*γονίδιο*» (*gene*), το οποίο αντιστοιχεί σε ένα απολύτως διακριτό δηφίο (*bit*), σε άμεση αντίθεση με την μεθοδολογία του *Caseau* που εισηγείται τον συσχετισμό ενός *gene* μόνο με τους κόμβους αυτούς που έχουν έναν μοναδικό άμεσο πατρικό κόμβο. Άρα είναι σαφώς βέλτιστη η διαφοροποίηση αυτή της επιλογή χρήσης ενός *gene* στους κόμβους αυτούς που όντως χρειάζονται ώστε να διαφοροποιηθούν μεταξύ τους. Μάλιστα η δυαδική αναπαράσταση του κωδικού του κάθε κόμβου διαμορφώνεται από την επιμέρους ένωση όλων των προσδιοριστικών αναπαραστάσεων μέσω των *genes* των προγόνων του, συμπεριλαμβανομένου και του δικού του διακριτού *gene*.

Ένα χαρακτηριστικό παράδειγμα της *NHE* κωδικοποίησης φαίνεται στο παράδειγμα του παρακάτω σχήματος (βλ. Σχήμα 15).

Ο *NHE* αλγόριθμος κωδικοποίησης με βάση την δυνατότητά του να διαχειρίζεται και να κωδικοποιεί στην συνέχεια μερικώς διατεταγμένα σύνολα (*partially ordered sets*) από ότι πλήρως διατεταγμένους πίνακες κατορθώνει και παράγει πολύ γρηγορότερα (αποφεύγει τον αναγκαίο εκθετικό χρόνο επεξεργασίας της όποιας περίπτωσης πλέγματος) και σαφώς πολύ μικρότερα σύνολα αναπαράστασης. Συγκεκριμένα, τα παραγόμενα σύνολα είναι κατά 50% μικρότερα από ότι αυτά που προκύπτουν από τον κλασικό *Caseau* [C93]] που προείδαμε. Με αυτόν τον τρόπο και οι πράξεις διερεύνησης ύπαρξης σχέσεως υπαλληλίας μεταξύ δυο κόμβων της εν λόγω ιεραρχίας καθίσταται πιο εφικτή. Προφανώς όμως οι παραγόμενοι κωδικοί ίσως να μη είναι οι πλέον συμπιεσμένοι δυνατοί συγκρινόμενοι με του *Caseau*, αλλά είναι αρκετά συμπιεσμένοι.

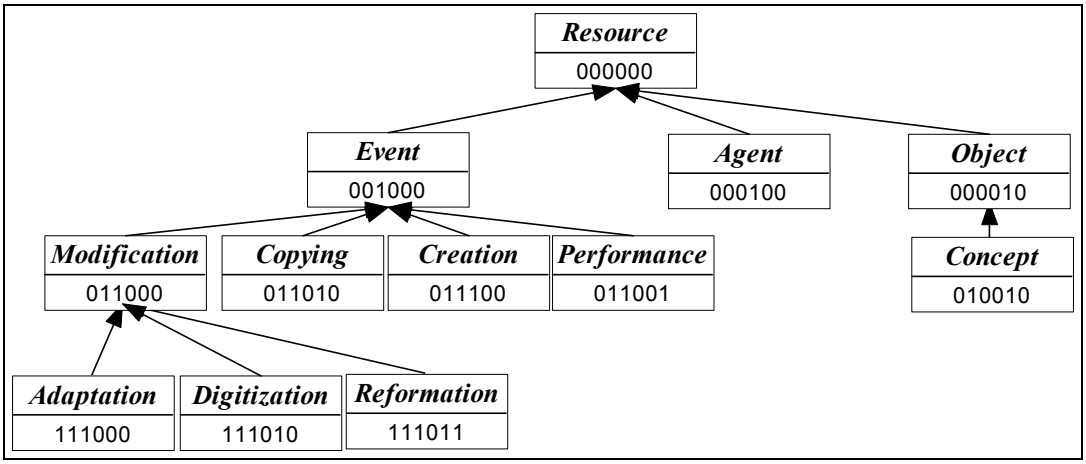
Επίσης ο *NHE* αλγόριθμος στηρίζεται στην εξισορρόπηση του ύψους της ιεραρχίας, κάτι που επιτυγχάνεται με την χρήση μιας διαδικασίας χαρακτηρισμού του γράφου (*graph coloring*) με απώτερο σκοπό την εύρεση μιας σχεδόν πάντοτε βέλτιστης λύσης. Η διαδικασία αυτή χρησιμοποιείται στην περίπτωση ιεραρχιών όπου υπάρχουν ή εισάγονται νέοι κόμβοι, οι οποίοι αναγκάζονται εκ της φύσης τους να διαμοιραστούν το ίδιο gene, αναθέτοντάς τους σε κάθε περίπτωση διαφορετικό *gene*. Βασικό γνώρισμα της ακολουθούμενης αυτής τεχνικής είναι η αντιμετώπιση αυτού του προβλήματος της κατά τέτοιο τρόπο ώστε να διατηρηθεί η ισορροπία του γράφου, επιτυγχάνοντας την μείωση του μέγιστου πλήθους των άμεσων απογόνων – "παιδιών" –του κόμβου αυτού και εκ παραλλήλου μειώνοντας έτσι και το σύνολο των αναγκαίων, διακριτών *genes*.

Αποσκοπώντας να προσδιορίσουμε την σωστή ιεραρχική *top-down* κωδικοποίηση, θα πρέπει σε κάθε περίπτωση να ισχύει η κάτωθι ισοδυναμία από και προς τις δυο κατευθύνσεις εξίσου:

$$x < y \Leftrightarrow \gamma(x) \subseteq \gamma(y),$$

όπου *γ(x)* αντιστοιχεί στην συνάρτηση αυτή η οποία αντιστοιχεί τον κάθε κόμβο στην αναπαράστασή του στο σύνολο. Εάν έχουμε έναν τύπο με ένα μόνο πατρικό κόμβο, τότε αυτός παίρνει ένα *gene* και αναδρομικά κάθε κλάση όλα τα επιμέρους *genes* των ιεραρχικά υπερκλάσεών του, διασχίζοντας από τα αριστερά προς τα δεξιά τη συνθήκης που ισχύει. Η οποιαδήποτε προσπάθεια προς την αντίθετη κατεύθυνση θα είναι αρκετά δύσκολο να εφαρμοστεί κι ιδιαιτέρως στην περίπτωση του μη πλέγματος. Συνεπώς, επιβάλλεται όλοι οι επιμέρους κόμβοι-κλάσεις που έχουν πολλαπλές άμεσες υπερκλάσεις, να έχουν κι αρκετό πλήθος διαφορετικών *genes* ώστε να μην παραβιάζεται η παραπάνω συνθήκη.

Για την διαμόρφωση μιας σωστής ιεραρχικής κωδικοποίησης με τον *NHE* αλγόριθμο δεν είναι απαραίτητο εκ του προοιμίου όλα τα χρησιμοποιούμενα *genes* να είναι απολύτως διακριτά μεταξύ τους. Μπορούν για παράδειγμα δυο επιμέρους *genes* να είναι τα ίδια, εάν όλα τα λοιπά χρησιμοποιούμενα *genes* εγγυώνται συνολικά διαφορετικές κωδικοποιήσεις. Για παράδειγμα, εάν θεωρήσουμε την περίπτωση μιας ιεραρχίας με δυο επιμέρους αλυσίδες, τότε προφανώς τα *genes* της μιας αλυσίδας μπορούν αν είναι τα ίδια με τα *genes* και της άλλης αλυσίδας. Ο μόνος περιορισμός που υπάρχει είναι ο πιο ψηλά στην ιεραρχία κόμβος της κάθε αλυσίδας αυτής να είναι απαραιτήτως διαφορετικός, ώστε να διασφαλιστεί έτσι σωστή κωδικοποίηση



Σχήμα 15: Near Optimal Hierarchical Encoding (NHE)

Εν γένει ο *NHE* αλγόριθμος χαρακτηρίζεται από βέλτιστη συμπεριφορά κι απόδοση στην κωδικοποίηση ιεραρχιών με μονές συνδέσεις, και σχεδόν βέλτιστη για ιεραρχίες με πολλαπλές συνδέσεις. Παρά το γεγονός ότι χαρακτηρίζεται από τη παραγωγή αρκετά συμπαγών κωδικών, η οποιαδήποτε αύξηση του μεγέθους της ιεραρχίας, συνιστά αύξηση του μέσου μεγέθους κωδικού, το οποίο εν συνεχεία συναρτά αναλόγως αύξηση και για τον χρόνο εκτέλεσης, αλλά και του μεγέθους του χώρου που λαμβάνει η εντολή. Δεδομένου ότι η όλη εξέλιξη των επιμέρους βημάτων του αλγορίθμου είναι γνωστά μόνο κατά την διάρκεια της σύνδεσης όταν έχουμε πλήρως όλη την ιεραρχία μας, κι όχι εκ των προτέρων, καθιστούν τον αλγόριθμό μας σταθερού χρόνου απόδοσης κατά την διάρκεια εκτέλεσης αλλά όχι σταθερό κατά την διάρκεια της μεταγλώττισης.

Συγκεντρωτικά, στη συνέχεια παραθέτουμε έναν πίνακα (Πίνακας 2) με την πολυπλοκότητα που παρουσιάζουν όλοι τα προαναφερθέντα σχήματα της οικογένειας αυτής κωδικοποίησης. Αναλυτικά, θα έχουμε :

Σχήμα Κωδικοποίησης	**Βέλτιστη Τάξη Μεγέθους Πολυπλοκότητας**	**Χείριστη Τάξη Μεγέθους Πολυπλοκότητας**
Wirth	n^2	n^2
Hasse	n^2	n^2
Caseau	*logn*	*n*
Ait-Kaci	*logn*	*n*
Near-Hierarchical–Encoding (NHE)	*logn* (μικρό ποσοστό πολλαπλής κληρονομικότητας)	*2*n*logn* (ισορροπημένα δέντρα)

Πίνακας 2: Πολυπλοκότητα Μεγεθών Δημιουργίας Ετικετών Σχημάτων Κωδικοποίησης Bit-Vector

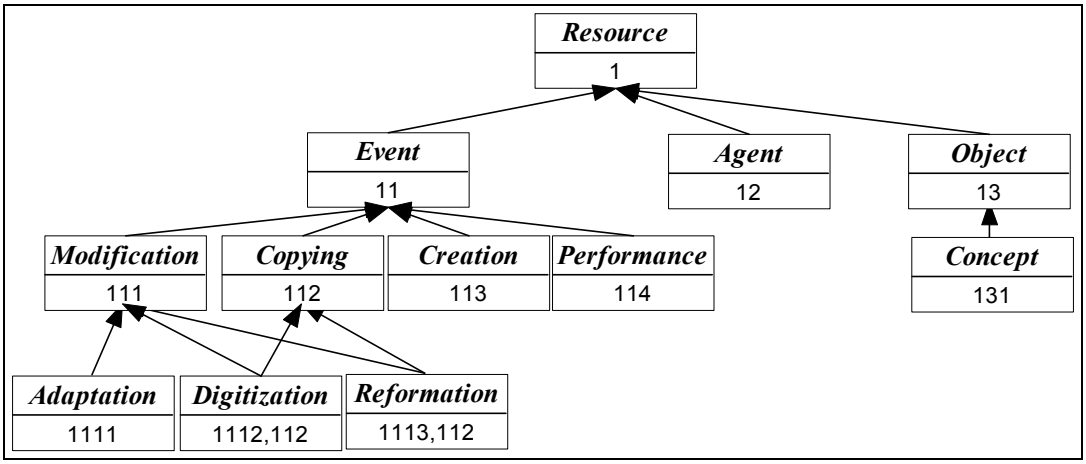
3.2 Αλγόριθμοι κωδικοποίησης με χρήση προθέματος (Prefix-based)

Η κατηγορία αυτή κωδικοποίησης με την χρήση προθέματος (*prefix-based*), έχει σαν βασική ιδέα την επιμέρους κωδικοποίηση του "άμεσου" πατρικού κόμβου κάποιου κόμβου σε ένα δέντρο *T*, με έναν *prefix* κωδικό, ο οποίος προκύπτει με βάση την διάσχιση του δέντρου *T κατά βάθος* (*pre-order* – από αριστερά προς τα δεξιά). Για αυτό τον λόγο μπορούν όλοι οι ετικέτες του εκάστοτε δέντρου *T* να υπολογισθούν σε γραμμικό χρόνο ως προ τον αριθμό των κόμβων του δέντρου.

Ο πλέον απλός αλλά και αντιπροσωπευτικός *prefix-based* αλγόριθμος κωδικοποίησης είναι ο αλγόριθμος του Dewey, πάνω στον οποίο βασίζεται και το γνωστό Σύστημα Δεκαδικής Κωδικοποίησης του Dewey (*Dewey Decimal Coding System -DDC*), το οποίο ευρέως χρησιμοποιείται από τους βιβλιοθηκάριους [OCLC]. Βασικό του χαρακτηριστικό είναι ότι ο κωδικός του κάθε κόμβου *u* του δέντρο *T* είναι της γενικής μορφής $l(v)l(u)$, όπου $l(v)$ είναι ο κωδικός του άμεσου πατρικού του κόμβου

v και ο κωδικός $l(u) \in \{0, ..,9\}$[3]. Ένας κόμβος *v* είναι πρόγονος του κόμβου *u* στο δέντρο *T* (δηλαδή $v < u$) εάν και μόνο εάν $l(v) \in prefixes(l(u))$. Κι αυτό σαν επιμέρους λειτουργία απαιτεί πρακτικά σταθερό χρόνο εκτέλεσης, ανάλογο με τον χρόνο που χρειάζεται για την σύγκριση επιμέρους δυο αλφαριθμητικών (strings). Ομοίως ισχύουν όλα τα προαναφερθέντα και στην περίπτωση διερεύνησης του «κοντινότερου κοινού προγόνου»(Nearest Common Ancestor -*nca*) δυο αντιστοίχων δεντρικών κόμβων.

Στη συνέχεια στο σχήμα 16 παραθέτεται ένα χαρακτηριστικό παράδειγμα του αλγορίθμου κωδικοποίησης του Dewey πάνω στην ιεραρχία *MetaNet*.



Σχήμα 16: Κωδικοποίηση Αλγορίθμου Dewey

Μια πολύ ενδιαφέρουσα ιδιότητα που σχετίζεται με την χρήση σχημάτων κωδικοποίησης βασισμένων σε πρόθεμα είναι ότι οι παραγόμενοι κωδικοί είναι λεξικογραφικά διατεταγμένοι. Για παράδειγμα οι κωδικοί των κόμβων ενός υποδέντρου είναι μεγαλύτεροι (μικρότεροι) από τους αντίστοιχους κωδικούς των δεξιών (αριστερών) "αδελφών" υποδέντρων. Σημασιολογικά θα μπορούσαμε αυτό να το αναπαραστήσουμε με την εξής διάταξη : $prev(l(v)) < l(u) < l(v) < next(l(u))$, όπου *next*("19") = "2" και αντίστοιχα *prev*("12") = "11". Στην περίπτωση αυτή συνίσταται η δημιουργία και χρήση δομών με δείκτες βασισμένων πάνω στο πεδίο τιμών του κλειδιού (B^+-*trees*), των οποίων η διάταξη μπορεί σαφώς να επιταχύνει την αποτίμηση των επιζητούμενων προς

[3] Η ιδέα αυτή είναι κοινή με αυτήν που χρησιμοποείται στον *ODP* κατάλογο για να αναγνωρίσουμε τις κλάσεις από την ρίζα της κάθε ιεραρχίας, με άμεσα αντιληπτές από τους χρήστες ετικέτες, με την χρήση ενός λεξιλογίου από διακριτές λέξεις /όρους

δημιουργία επερωτήσεων (που είναι οι επερωτήσεις αναζήτησης προγόνων, απογόνων, "φύλλων"- κόμβων, ομοεπίπεδων κόμβωνκ.λ.π.).

Είδος Επερώτησης	**Dewey**
descendants(u) $\{u \mid u \prec v\}$	$l(u) < next(l(u)) \wedge l(u) > l(v)$
ancestors(u) $\{u \mid u \succ v\}$	*prefixes(l(u))*
leaves(u) $\{u \mid u \prec v \wedge \neg\exists u' : (u' \prec v \wedge u' \prec u)\}$	$l(u) < next(l(v)) \wedge l(u) > l(v) \wedge \neg\exists u' : (l(u') > l(v) \wedge l(u') < next(l(v)) \wedge l(u') < next(l(u)) \wedge l(u') > l(u))$
precedings(u) $\{u \mid u \ll v\}$	$l(u) <= prev(l(v)) \wedge l(u) > prefix(l(v))$
followings(u) $\{u \mid u \gg v\}$	$next(l(u)) >= l(v) \wedge l(u) < next(mprefix(l(v))$
siblings(u) $\{u \mid u \leftrightarrow v\}$	$mprefix(l(u) = mprefix(l(u)$
nca(v,w) $\{u \mid u \succ v \wedge \neg\exists u' : (u' \succ v \wedge u' \succ w \wedge u' \prec u)\}$	$l(u) \in prefixes(l(v)) \cap prefixes(l(w)) \wedge mlength(l(u))$

Πίνακας 3: Συνθήκες Επερωτήσεων Σχημάτων κωδικοποίησης Dewey

Στον πίνακα 3 κάθε μία από τις βασικές επερωτήσεις της κωδικοποίησης αυτής και ιδίως οι συνθήκες τους, εκφράζονται σε δηλωτική έκφραση στην δεύτερη στήλη του. Θα πρέπει στο σημείο αυτό να αναφερθεί ότι ανάμεσα στις δυνατές *SQL* ερωτήσεις που επιτρέπει ο αλγόριθμος της *Dewey* κωδικοποίησης, οι ερωτήσεις που αναφέρονται στην εύρεση των άμεσων προγόνων (πατέρων), των άμεσων απογόνων (παιδιών), και των "αδελφών" κόμβων στηρίζονται αποκλειστικά στην χρήση συναρτήσεων συνταιριασμού αλφαριθμητικών. Αναλυτικά, στην μεν περίπτωση της αναζήτησης του άμεσου πατρικού κόμβου ενός κόμβου στο υπό διερεύνηση δέντρο *T*, αυτός προκύπτει άμεσα από το μεγαλύτερο πρόθεμα που προκύπτει από τον κωδικό του κόμβου αυτού (χρήση συνάρτησης *mprefix*). Ωστόσο, εδώ θα πρέπει να αναφερθεί πως σε όλες τις υπάρχουσες

SQL μηχανές αναζήτησης, οι οποίες ακολουθούν το *SQL standard* (*SQL-99*) τέτοιου είδους ερωτήσεις που συναρτώνται αποκλειστικά σε συναρτήσεις οιαδήποτε επεξεργασίας των αλφαριθμητικών κωδικών, δεν μπορούν να ευεργετηθούν από την ύπαρξη δεικτών πάνω στους αλφαριθμητικούς αυτούς κωδικούς.

Ανάμεσα στα διακριτά χαρακτηριστικά των *prefix-based* αλγορίθμων κωδικοποίησης είναι ότι έχουν μεταβλητό μέγεθος (σε δηφία) αναπαράστασης, όπου σε κάθε περίπτωση το μέγιστο μέγεθος του χρησιμοποιούμενου κωδικού εξαρτάται από το μέγιστο βάθος (έστω d), αλλά και από το πλήθος των εξερχόμενων ακμών από τον κόμβο (έστω Δ) του υπό κωδικοποίηση δέντρου μέσω της σχέσης $d*\log\Delta$. Το σχήμα αυτό έχει χρησιμοποιηθεί σε πολλές εφαρμογές που χρησιμοποιούν την αναπαράσταση των δεδομένων με την μορφή *XML* δέντρων, όπως είναι οι περιπτώσεις [CSFHS01][TVBSSZ02]. Η πιο απλή περίπτωση μορφολογίας δέντρου, είναι αυτή, όπου ο αποδοθέν κωδικός του κάθε κόμβου, ανεξάρτητα επιπέδου, είναι ακριβώς ένα byte – σαφώς και υπάρχει στην περίπτωση αυτή χρησιμοποίηση περισσοτέρων από το πλήθος των πραγματικά αναγκαίων προς επιμέρους διαφοροποίηση πλήθους δηφίων -. Στην συγκεκριμένη αυτή περίπτωση, οι χρησιμοποιούμενοι κωδικοί προέρχονται από το σύνολο των δεκαδικών αριθμών αναπαράστασης {0,..,9}.

Εν συνεχεία, στις περιπτώσεις ύπαρξης επιμέρους κόμβων με αριθμό εξερχόμενων κόμβων μεγαλύτερο του 10 (όπως συνήθως είναι κι η περίπτωση των χρησιμοποιούμενων σχημάτων αναπαράστασης των διαδικτυακών πυλών - *portals*), είναι αναγκαία η χρήση σαφώς μεγαλύτερων αλφαβήτων για την αναγκαία κωδικοποίηση των κόμβων. Ένα βέλτιστο τέτοιο χαρακτηριστικό παράδειγμα χρησιμοποιούμενου αλφαβήτου αναμφισβήτητα είναι το *Unicode Character Set* (*UTF-8*). Στην περίπτωση αυτή του *UTF-8* απαιτείται ένας μεταβλητός αριθμός από *bytes* για την κωδικοποίηση ακεραίων κωδικών : οι μεν *ASCII* χαρακτήρες μπορούν να κωδικοποιηθούν με την χρήση ενός και μόνο byte (συγκεκριμένα από τον 0x00 μέχρι τον 0x7f κωδικό), ενώ οι λοιποί χαρακτήρες μπορούν να κωδικοποιηθούν με την σειρά τους μέσω μιας πολλαπλής σειράς από bytes (μόνος περιορισμός του πλήθους αυτού είναι ανάμεσα σε 0x80 και σε 0xfd). Στη δεύτερη αυτή περίπτωση το μεν πρώτο byte θα χρησιμοποιείται για να μας δείξει το μέγεθος του υπό διαμόρφωση κωδικού – προθέματος κι το οποίο μπορεί να φτάσει έως και 3 bytes μέγιστο μέγεθος. Στην συγκεκριμένη όμως αυτή περίπτωση όμως της πύλης

του ODP και με την συγκεκριμένη μορφολογία των σχημάτων του, όπου ο μέσος όρος του εμφανιζόμενου πλήθους εξερχόμενων ακμών ανά κλάση είναι αρκετά μικρός (0,9999), καθώς και σε σχέση με τον μέγιστο πλήθος εξερχόμενων ακμών (314), ο αριθμός των κωδικών των κόμβων που χρειάζονται ένα byte για το αναπαράσταση του κωδικού τους είναι αρκετά μεγάλος (αντιπροσωπεύονται ικανοποιητικώς μέχρι ένα ποσοστό από τους ASCII χαρακτήρες).

Σαν επιμέρους μεθοδολογία η κωδικοποίηση βασιζόμενη στην χρήση προθέματος είναι εξίσου ακολουθούμενη και σε περιπτώσεις επιμέρους εφαρμογών που σχετίζονται με την δεντρική αναπαράσταση XML δεδομένων [K93, CSFH01, TVBS02]. Επίσης, μια πληθώρα εναλλακτικών προτάσεων έχουν μελετηθεί σε σχέση με τις κωδικοποιήσεις προθέματος, που εναλλακτικά είτε επιδιώκουν περισσότερο συμπιεσμένους κωδικούς, οι οποίοι με την σειρά τους ελαχιστοποιούν είτε και το μέγιστο μέγεθος του κωδικού-προθέματος αναπαράστασης (δεδομένου ότι αναφερόμαστε για αναπαράσταση σταθερού μεγέθους κωδικών), καθώς και το μέσο μέγεθος του κωδικού (για αναπαράσταση μεταβλητού μεγέθους κωδικών) [GP03], [SKM02].

Εν γένει οι αλγόριθμοι κωδικοποίησης με πρόθεμα παρουσιάζουν το εξής κύριο θετικό χαρακτηριστικό γνώρισμα, ότι διακρίνονται για την ιδιαίτερη δυναμικότητα που παρουσιάζουν στην περίπτωση αυξητικών ενημερώσεων (*incremental updates*). Κι αυτό οφείλεται στο ότι η βασική ιδέα της κωδικοποίησης αυτής είναι ότι δεν επιδιώκει, εφόσον δεν είναι ολωσδιόλου σημαντική, η ταξινόμηση μεταξύ των απογόνων στις εκάστοτε εξεταζόμενες υπο-ιεραρχίες, καθώς δύναται να προσθέτονται καινούργια "παιδιά" ή "αδελφοί" κόμβοι στα δεξιά των υπαρχόντων κόμβων κάθε φορά, χωρίς να χρειάζεται να τους επανακωδικοποιήσεις εξ αρχής. Στο σημείο αυτό να αναφέρουμε πως το μεγαλύτερο μέρος των οφελών που απορρέουν από την χρήση κωδικοποιήσεων με πρόθεμα σε ότι αφορά τους τομείς της επιχειρούμενης βέλτιστης απόδοσης των ενημερώσεων και της συμπίεσης των κωδικών έγκειται σαφώς στο μεταβλητό μέγεθος των κωδικών.

Από την άλλη πλευρά όμως η κατηγορία αυτή των κωδικοποιήσεων με μεταβλητό μέγεθος ετικετών έχει ένα ιδιαίτερα μεγάλο κόστος σε ότι αφορά την εκτέλεση των ερωτήσεων που βασίζονται στην χρήση συναρτήσεων διαχείρισης αλφαριθμητικών - και εξαιρετικά μεγάλο κόστος στην περίπτωση συμπιεσμένων

εκδόσεων κωδικοποιήσεων προθέματος (τέτοιο χαρακτηριστικό παράδειγμα είναι οι ερωτήσεις για τους "αδελφούς" κόμβους). Κι αυτό έχει σαν επακόλουθο αποτέλεσμα την ελάττωση στο ελάχιστο των όποιων δυνατοτήτων που υπάρχουν για βελτιστοποίηση των υπαρχόντων *SQL* μηχανών αναζήτησης που στηρίζονται σε τέτοιας μορφής κωδικοποιήσεις. Συγκεκριμένα, οι βελτιστοποιητές αυτών των βάσεων δεν μπορούν να εκτιμήσουν το κόστος εκτέλεσης των συναρτήσεων που έχουν οριστεί και χρησιμοποιηθεί εν συνεχεία. Επιπρόσθετα, οι αλγόριθμοι αυτοί παρουσιάζουν το μεγάλο μειονέκτημα ότι η χρήση τους συνίσταται κυρίως για την περίπτωση δέντρων, αλλά σαφώς κι επεκτείνεται και για την περίπτωση των *DAGs*, όπου δεδομένου της διαχείρισης ιεραρχιών με πολλαπλή κληρονομικότητα, που ευρέως εμφανίζονται σε ιεραρχίες διαδικτυακών πυλών, συνεπάγεται επιπλέον κόστος αποθήκευσης. Κι αυτό γιατί στην περίπτωση των *DAGs* με την χρήση κωδικοποιήσεων με πρόθεμα (*prefix-based*) παράγονται εν τέλει εκτεταμένου μεγέθους κωδικοί-προθέματα.

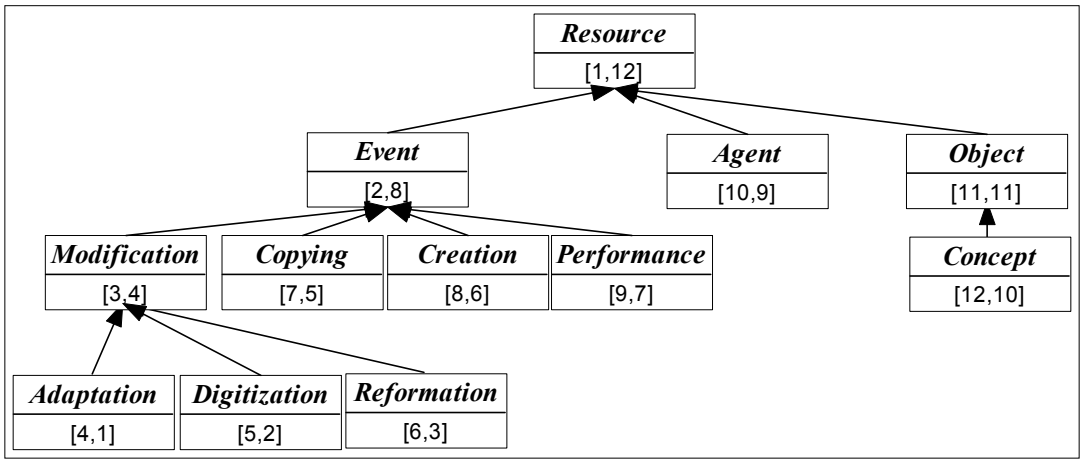
3.3. Αλγόριθμοι κωδικοποίησης με χρήση διαστημάτων /πεδίων (Interval-based).

Η κατηγορία αυτή κωδικοποίησης για την αναπαράσταση των σχέσεων προγόνων /απογόνων σε ένα δέντρο *T*, στηρίζεται στην χρήση κατάλληλων διαστημάτων της γενικής μορφής [*start, end*]. Συγκεκριμένα, με την παραγωγή των διαστημάτων αυτών κι μέσω της χρήσης των άκρων τους, διερευνάται η συσχέτιση των διαστημάτων με το διάστημα αυτό που προσδιορίζει μοναδικά τον εκάστοτε εξεταζόμενο κόμβο. Οι επιμέρους διαφοροποιήσεις που διακρίνουν τώρα τους αλγορίθμους που ανήκουν στην κατηγορία αυτή είναι ως προς την αποδίδουσα σημασία, την διαδικασία με την οποία παράγεται και την έννοια που αποδίδεται στο κάθε άκρο του διαστήματος και τι χαρακτηριστικά γνωρίσματα προσφέρει επακόλουθα στο ακολουθούμενο σχήμα που εισάγει. Ευθύς αμέσως, θα παραθέσουμε τέσσερις διαφορετικούς αλγορίθμους που χρησιμοποιούν με παραλλαγές την διαμόρφωση των διακριτών αναγνωριστικών διαστημάτων τους.

3.3.1 Dietz Αλγόριθμος

Στον κλασσικό αλγόριθμο του *Dietz* , σε κάθε κόμβο, έστω *u* αποδίδεται ένα πεδίο, του οποίου τα άκρα είναι οι *pre-order* και *post-order* αριθμοί που προκύπτουν για τον κόμβο από την *pre-order* και *post-order* διάσχιση αντιστοίχως του δέντρου *T*. Συνεπώς, το πεδίο-κωδικός είναι της γενικής μορφής [*pre(u), post(u)*]. Η παραγωγή των κωδικών αυτών ως γνωστών που προκύπτουν από την *pre-order* και *post-order* διάσχιση δέντρων γίνεται σε χρόνο γραμμικό ως προς το μέγεθος του δέντρου *T*.

Στη συνέχεια ακολουθεί η σχηματική αναπαράσταση της διαμόρφωσης των κωδικών με βάση τον αλγόριθμο *Dietz* για την ιεραρχία *MetaNet* (βλ. σχήμα 17).



Σχήμα 17: Κωδικοποίηση Αλγορίθμου Dietz

Με βάση αυτήν την σημασιολογία που έχουν τα άκρα του πεδίου κάθε κόμβου, ένας προγονικός κόμβος *v* εμφανίζεται πριν (μετά) από έναν απόγονο κόμβο *u* στην pre-(post) order διάσχιση του δέντρου *T* εάν κι εφόσον ο *u* είναι ένας πρόγονος του *v* και ισχύουν ότι *pre(u) < pre(v)* και *post(u) > post(v)*. Ομοίως προκύπτουν και οι λοιπές συνθήκες που χρησιμοποιούνται για την αναζήτηση των "προγόνων" / "απογόνων" / "φύλλων"- κόμβων, όπως και των *precedings /followings*.(βλ. Πίνακα 4). Η αναζήτηση μέσω επερώτησης των της διερεύνησης της σχέσης απογόνου - προγόνου μεταξύ δυο κόμβων μπορεί να πραγματοποιηθεί σε σταθερό χρόνο, ενώ η αποθήκευση των κωδικών των κόμβων ενός δέντρου *T* γίνεται σε χρόνο της τάξης *O(n)*, και το μέγεθος των παραγόμενων κωδικών σε δηφία είναι ακριβώς *2*logn* [T84].

Είδος Επερώτησης	**Dietz /Zhang et al**
descendants(u) $\{u \mid u \prec v\}$	*Post(u) < post(v)* $\wedge$ *pre(v) < pre(u)*
ancestors(u) $\{u \mid u \succ v\}$	*Post(v) > post(u)* $\wedge$ *pre(u) > pre(v)*
leaves(u) $\{u \mid u \prec v$ $\wedge \neg \exists u' :$ $(u' \prec v$ $\wedge u' \prec u)\}$	*Post(u) < post(v)* $\wedge$ *pre(v) < pre(u)* $\wedge$ *pre(u) =* *depth(u) +post(u)*
precedings(u) $\{u \mid u \ll v\}$	*Post(u) < post(v)* $\wedge$ *pre(u) < pre(v)*
followings(u) $\{u \mid u \gg v\}$	*Post(u) > post(v)* $\wedge$ *pre(u) > pre(v)*
siblings(u) $\{u \mid u \leftrightarrow v\}$	*Pre(u) =* *pre(father(v))*
nca(v,w) $\{u \mid u \succ v$ $\wedge \neg \exists u':$ $(u' \succ v$ $\wedge u' \succ w$ $\wedge u' \prec u)\}$	*Post(u) > post(w)* $\wedge$ *pre(u) > post(v)* $\wedge \neg \exists u':$ *(post(u') > post(w)* $\wedge$*pre(u') > pre(v)* $\wedge$ *pre(u') > pre(u)* $\wedge$ *post(u') <post(u))*

Πίνακας 4: Συνθήκες Επερωτήσεων Σχημάτων κωδικοποίησης Dietz

Συγκεκριμένα, με βάση την μορφολογία των χρησιμοποιούμενων κωδικών για έναν κόμβο *u* στο δέντρο *T (*βλέπε πίνακα 4), έχουμε ότι

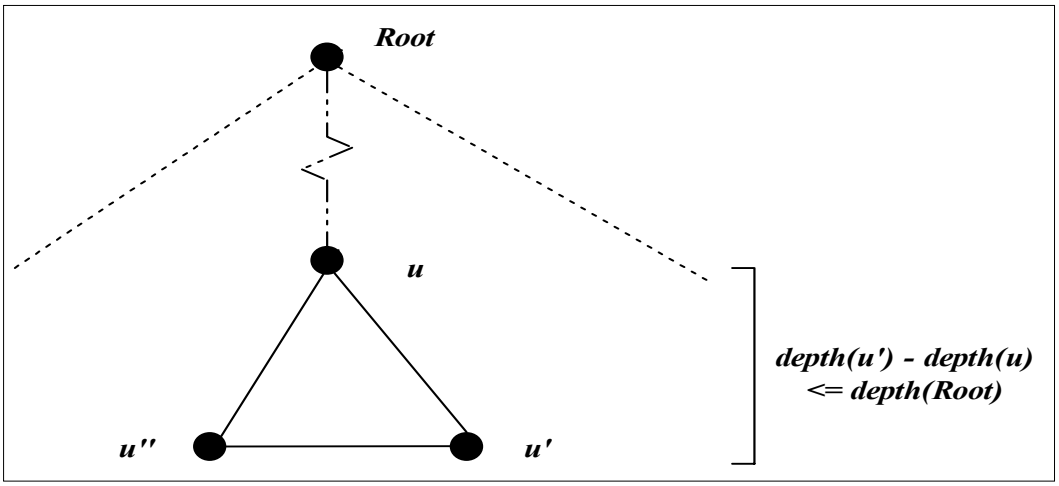
$$pre(u) - post(u) + size(u) = depth(u) \quad (1)$$

,όπου θεωρούμε ως *size(u)* το μέγεθος του υποδέντρου που ορίζεται στον κόμβο *u* και *level(u)* την τιμή του επιπέδου που βρίσκεται ο *u* στην ιεραρχία. Για παράδειγμα εάν εξετάσουμε τον κόμβο *Resource* στο παραπάνω παράδειγμα με *pre(Resource)* = 1, *post(Resource)* = 12 και *size(u)* = 11 διαπιστώνουμε πως όντως ο *Resource* βρίσκεται στο επίπεδο *depth(Resource)* = 0. Εάν τώρα θεωρήσουμε ότι ο κόμβος *u* είναι "φύλλο" – κόμβος τότε η παραπάνω συνθήκη σύνδεσης των άκρων του πεδίου – κωδικού τους, δεδομένου ότι *size(u)*=0 μετατρέπεται σε

$$pre(u)\text{-}post(u)=depth(u)<\text{μέγιστο_βάθος_δέντρου}. \quad (2)$$

Συνεχίζοντας πάνω στην ίδια λογική, θεωρούμε ένα "φύλλο" – το πιο αριστερό όπως προσδιορίζεται από την μέγιστη *preorder* ταξινόμησή του- έστω *u'* του *u* κόμβου. Με βάση τον πίνακα 4 και τα προαναφερθέντα μπορούμε εύκολα να καταλήξουμε στην εξής συσχέτιση μεταξύ των *u'* και *u*.

$$post(u') = post(u') + ((depth(u')- depth(u)) \quad (3)$$
$$\text{όπου } (depth(u')- depth(u)) \leq depth(Root)$$



Σχήμα 18: Συσχέτιση minimum post τιμής (u'') και μέγιστης pre τιμής (u') δεδομένης ταξινόμησης από το u

Παρατηρούμε δηλαδή πως με την *postorder* διάσχιση του εξεταζόμενου δέντρου, χρειάζεται να διατρέξουμε όλους τους προγόνους του *u'* οι οποίοι με την σειριακά κατατάσσονται στο πεδίο *(level(u')- level(u))* μέχρι την προσέγγιση του κόμβου *u*. Με βάση τις ισότητες (2) και (3) μπορούμε να καταλήξουμε σε ένα μέγιστο όριο για το *pre(u')* και κατ' επέκταση για όλους τους κόμβους του υποδέντρου, δηλαδή :

$$pre(u') \leq post(u)+depth(Root) \quad (4)$$

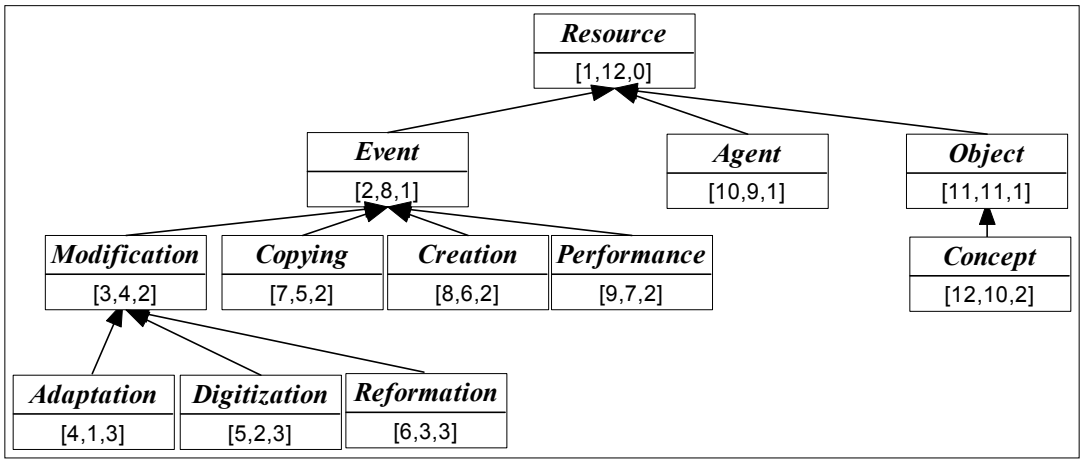
Αντιστοίχως, μπορούμε να προσδιορίσουμε καi το πιο αριστερό ως προς την σειρά του ταξινόμησης απόγονο κόμβο του *u* έστω *u''*. Αναλυτικά, παρατηρούμε πως η *postorder* θέση ταξινόμησής του, όπως προσδιορίζεται από το *post(u'')* είναι η ελάχιστη σε όλο το υπέδεντρο. Κια πάλι με βάση τις ισότητες (2) και (3) καταλήγουμε σε ένα όριο για το *post(u'')* και συνεπαγόμενος για όλους τους άλλους κόμβους του υπόδεντρου.

$$post(u'') \geq pre(u) + depth(Root) \quad (5)$$

3.3.2 Zhang Αλγόριθμος

Ένα χαρακτηριστικό παράδειγμα εφαρμογής μιας παραλλαγής της βασικής ιδέας του κλασικού αλγορίθμου του *Dietz* είναι το σχήμα κωδικοποίησης του αλγορίθμου του *Zhang* [D82, DS87], για τα δεδομένα αναπαριστάμενα με την μορφή ενός XML δέντρου, με την επιπλέον προσθήκη της πληροφορίας του βάθους του κάθε κόμβο στην ιεραρχία. Αναλυτικά, σε κάθε κόμβο *u* αποδίδεται ένα πεδίο της γενικής μορφής [*pre(u), post(u), depth(u)*] με άκρα τους *pre-order* και *post-order* αριθμούς που προκύπτουν για τον κόμβο από την *pre-order* και *post-order* διάσχιση αντιστοίχως του δέντρου του *T*, σε χρόνο γραμμικό ως προς το μέγεθος του δέντρου *T*. Επιπρόσθετα, υπάρχει στο πεδίο αυτό και ο κωδικός του βάθους του εν λόγω κόμβου στην ιεραρχία, *depth(u)*.

Στη συνέχεια ακολουθεί η σχηματική αναπαράσταση της διαμόρφωσης των κωδικών με βάση τον αλγόριθμο *Zhang* για την ιεραρχία *MetaNet* (σχήμα 19).



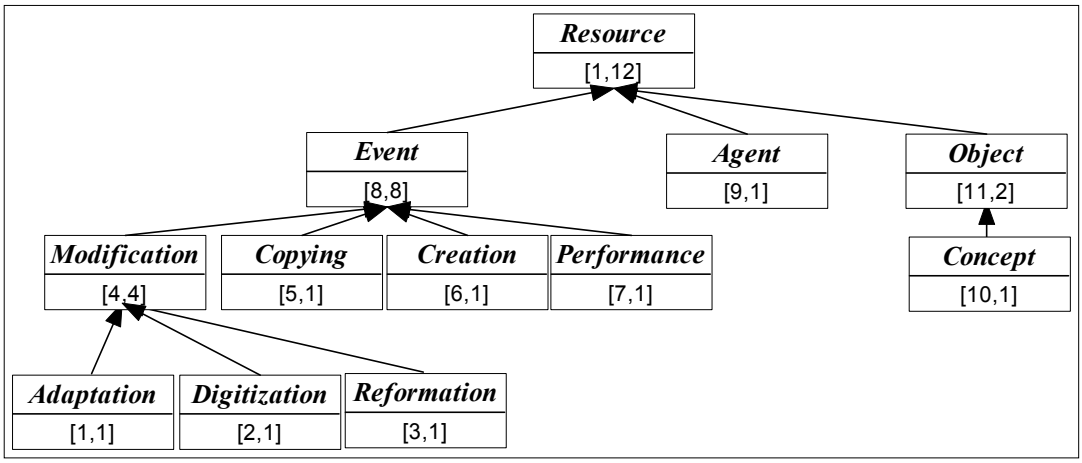
Σχήμα 19 : Κωδικοποίηση Αλγορίθμου Zhang

Στην παραλλαγή αυτή του αλγορίθμου του *Zhang* ισχύουν όλες οι συνθήκες επερωτήσεων που ίσχυαν και για τον αλγόριθμο *Dietz* (βλ. Πίνακα 4) με την προσθήκη του πεδίου *depth()*. Η εν λόγω αυτή προσθήκη στοχεύει στον υπολογισμό των άμεσων πατρικών (“πατέρων”), απογόνων (“παιδιών”) και “φύλλων” κόμβων. Εάν και το πρόβλημα της πραγμάτωσης των ερωτήσεων αυτών πραγματώνεται με την προσθήκη αυτή, δεν είναι ικανό το σχήμα αυτό να υπολογίσει ικανοποιητικά του ομοεπίπεδους κόμβους (*siblings nodes*), όπου και σαφώς χρειάζεται η επιπρόσθετη κωδικοποίηση του πατρικού του κάθε κόμβου του δέντρου. Άρα, η νέα αυτή προσθήκη καθιστά την σημασία και την χρησιμότητα της μεν πρώτης προσθήκης, της πληροφορίας του βάθους, πλέον περιττή και ολωσδιόλου μη αναγκαία (*redundant*).

3.3.3 Αλγόριθμος Li & Moon

Το σχήμα κωδικοποίησης που προτείνουν οι *Li* και *Moon* [LM01], για κάθε κόμβο έστω *u*, αποδίδεται ένα πεδίο –κωδικός , όπου τον μεν αριστερό του άκρο είναι ο *pre-order* αριθμός που προκύπτει από την *pre-order* διάσχιση του δέντρου T, έστω *pre(u)*, και από τον αριθμό των απογόνων του u ή εναλλακτικά το μέγεθος του υποδέντρου που έχει σαν ριζικό κόμβο τον *u*, έστω *size(u)*. Άρα το πεδίο-κωδικός είναι της γενικής μορφής [*pre(u), size(u)*]. Η χρήση του αλγορίθμου έγκειται στην αναπαράσταση των δεδομένων στην μορφή *XML* δέντρου.

Στο σχήμα αναπαριστούμε την κωδικοποίηση που προφέρει ο αλγόριθμος *Li* και *Moon* στην *RDF/S* ιεραρχία *MetaNet* (Σχήμα 20).



Σχήμα 20: Κωδικοποίηση Αλγορίθμου Li & Moon

Είδος Επερώτησης	**Li & Moon**
descendants(u) *{u|u ≺v}*	*pre(v) < pre(u)* *∧ pre(u) +size(u) <= pre(v) + size(v)*
ancestors(u) *{u|u ≻ v}*	*pre(u) > pre(v)* *∧ pre(v) +size(v) >= pre(u) +size(u)*
leaves(u) *{u| u ≺v* *∧ ¬∃ u' :* *(u' ≺ v* *∧ u' ≺ u)}*	*pre(v) < pre(u) ∧size(u) = 1* *∧pre(u) < pre(v) +size(v)*
precedings(u) *{u|u ≪v}*	*pre(u)+size(u) <= pre(v)*
followings(u) *{u|u ≫v}*	*pre(u) >= pre(v)+size(v)*
siblings(u) *{u|u ↔v}*	*pre(u) > pre(father(v))*
nca(v,w) *{u|u ≻ v* *∧ ¬∃ u':* *(u' ≻ v* *∧ u' ≻ w* *∧u' ≺u)}*	*pre(u) > pre(v)* *∧ pre(w) +size(w) >= pre(u) +size(u* *∧ ¬∃u': (pre(u') > pre(v)* *∧pre(w)+size(w) >= pre(u')+size(u'* *∧size(u') < size(u))*

Πίνακας 5: Συνθήκες Επερωτήσεων Σχημάτων κωδικοποίησης Li & Moon

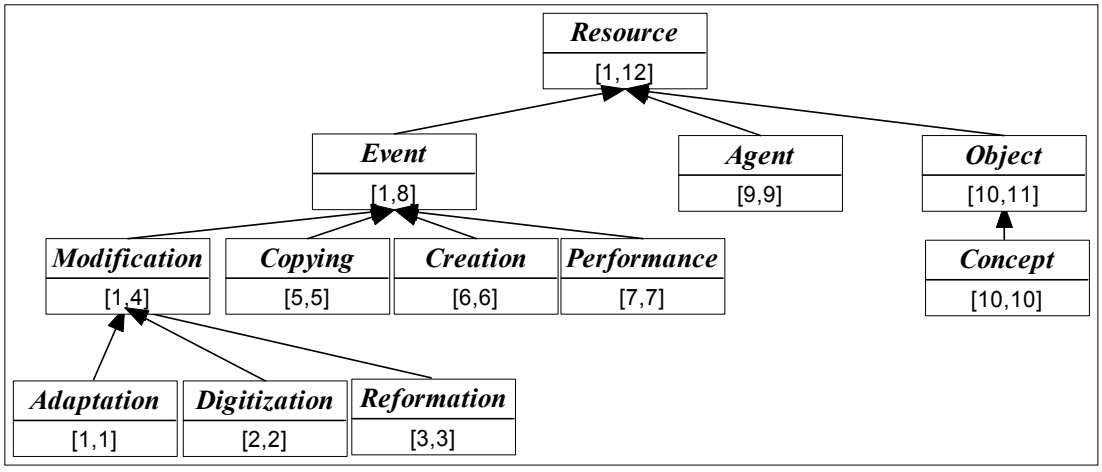
Παρατηρούμε με βάση την μορφολογία του χρησιμοποιούμενου κωδικού στο σχήμα 20 του αλγορίθμου των *Li & Moon* ότι ακολουθείται κι εδώ η βασική, κοινή αρχή του αλγορίθμου *Dietz / Zhang* που προείδαμε με την διαφοροποίηση ότι αντί του *post* πεδίου εισάγουμε την έννοια του *size()*, δηλαδή του μεγέθους του υποδέντρου του εκάστοτε κόμβου. Συνεπώς, ο συνδυασμός των *pre(u)+index(u)* αλλά και του *pre(u)* μας δίνουν μια δυνατότητα συσχέτισης του κόμβου *u* (βλ. Πίνακα 5 και τις σχετικές ερωτήσεις αναζήτησης προγόνων /απογόνων).

3.3.4 Αλγόριθμος Agrawal, Borgida & Jagadish

Στον κλασσικό αλγόριθμο των *Agrawal, Borgida & Jagadish* [ABJ89] – Agrawal et al - στον κάθε κόμβο *u* της ιεραρχίας αποδίδεται ένα πεδίο-κωδικός της μορφής [*index(u), post(u)*], όπου το *post(u)* είναι το απόλυτα διακριτό άνω όριο, ο *postorder index* αριθμός, και *index(u)* ο μικρότερος σε απόλυτη τιμή *postorder index* αριθμός μεταξύ των απογόνων του κόμβου *u*. Το σχήμα αυτό κωδικοποίησης μπορεί κάλλιστα να επεκταθεί στην κωδικοποίηση τόσο δέντρων όσο και πλεγμάτων όπως θα δούμε στην συνέχεια.

3.3.4.1 Περίπτωση Δέντρου

Στο σχήμα 21 αναπαριστούμε την κωδικοποίηση που προφέρει ο αλγόριθμος των *Agrawal, Borgida & Jagadish* στην RDF/S ιεραρχία *MetaNet,* η οποία παρουσιάζει μορφολογία συνδέσεων μονής πολυπλοκότητας (δέντρο).



Σχήμα 21: Κωδικοποίηση Αλγορίθμου Agrawal, Borgida, Jagadish–Περίπτωση Δέντρου

Στον πίνακα 6 εν συνεχεία θα δείξουμε όλες τις δυνατές συσχετίσεις, με την κάτωθι μορφή φορμαλισμού, ενός κόμβου *u* με τους υπόλοιπους κόμβους της υφιστάμενης υπό εξέταση ιεραρχίας.

Είδος Επερώτησης	**Agrawal et al**
descendants(u) $\{u \mid u \prec v\}$	$index(v) <= post(u)$ $\wedge\ post(u) < post(v)$
ancestors(u) $\{u \mid u \succ v\}$	$index(u) <= post(v)$ $\wedge\ post(v) < post(u)$
leaves(u) $\{u \mid u \prec v$ $\wedge\ \neg\exists\ u' :$ $(u' \prec v$ $\wedge\ u' \prec u)\}$	$index(v) <= post(u)$ $\wedge\ post(u) < post(v)$ $\wedge\ post(u) = index(u)$
precedings(u) $\{u \mid u \ll v\}$	$post(u) < index(v)$
followings(u) $\{u \mid u \gg v\}$	$index(u) > post(v)$
siblings(u) $\{u \mid u \leftrightarrow v\}$	$post(father(u)) = post(father(v))$
nca(v,w) $\{u \mid u \succ v$ $\wedge\ \neg\exists\ u' :$ $(u' \succ v$ $\wedge\ u' \succ w$ $\wedge u' \prec u)\}$	$index(u) <= index(v)$ $\wedge\ post(u) > post(w)$ $\wedge\ \neg\exists u' : (index(u') <= index(v)$ $\wedge\ post(u') > post(w)$ $\wedge\ index(u') <= post(u)$ $\wedge\ post(u') < post(u))$

Πίνακας 6: Συνθήκες Επερωτήσεων Σχήματος κωδικοποίησης Agrawal et al

Η πιο σημαντική ιδιότητα που μας δίνει η μορφολογία της αναπαράστασης του πεδίου-κωδικού του *Agrawal et al* είναι ο ακριβής συσχετισμός που μας δίνουν τα άκρα του σε σχέση με όλους τους λοιπούς κόμβους (βλ. Πίνακα 6). Αναλυτικά, δεδομένου του κόμβου *u* η σχέση

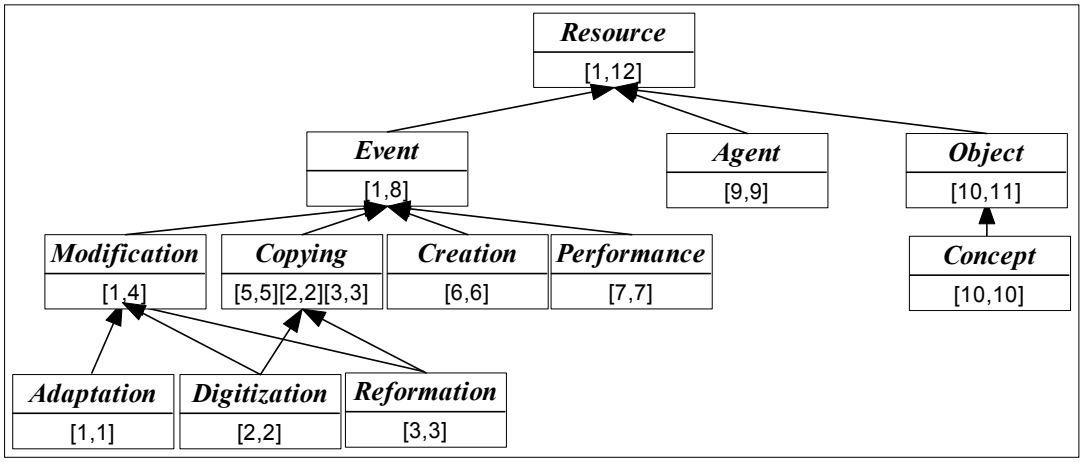
$$post(u) - index(u) = size(u)+1$$

μας υποδεικνύει ότι το πλήθος του υποδέντρου, *size(u),* που ορίζεται από τον κόμβο *u* ισούται ακριβώς από την διαφορά των δυο άκρων του πεδίου κωδικού του μείον 1. Προφανώς, εάν $post(u) = index(u)$ τότε ομιλούμε περί "φύλλων" - κόμβων καθώς το

μέγεθος του υποδέντρου στην περίπτωση αυτή είναι μηδενικό. Απόλυτο διακριτό προσδιοριστικό γνώρισμα είναι το πεδίο *post*(*u)* και ουσιαστικά η ταξινόμηση των κόμβων γίνεται ως προς αυτό. Η χρήση του πεδίου *index(u)* έγκειται στην ολοκλήρωση της ιεραρχικής διασύνδεση έκαστου κόμβου και την διαμόρφωση συσχετίσεων μεταξύ των κόμβων. Για παράδειγμα εάν θεωρήσουμε έναν άλλο κόμβο *v* τότε μπορούμε να συγκρίνοντας αν τα όρια του περιλαμβάνουν τον άλλο, *u,* εάν αντιστοίχως ο πρώτος είναι πρόγονος του άλλου (*index(v) <=post(u)*και *post(u) <post(v))* κι αντίστροφα. Εάν τα διαστήματα δεν περιλαμβάνει το ένα το άλλο τότε μιλάμε για *sibling* κόμβους, δηλαδή "αδελφούς"-κόμβους ή "παιδιά"-κόμβους των "αδελφών"- κόμβων του πατρικού του εκάστοτε εξεταζόμενου κόμβου, δηλαδή (*index(v) <= post(v) < index(u) <= post(u)) ή (index(v) <= post(v) <= index(u) <= post(u))*.

3.3.4.2 Περίπτωση Πλέγματος

Όπως προείπαμε, το σχήμα της κωδικοποίησης αυτό μπορεί κάλλιστα να επεκταθεί και για την περίπτωση μιας ιεραρχίας που έχει την μορφολογία πλέγματος, όπως εικονίζεται και στο σχήμα 22. Χαρακτηριστικά, το συγκεκριμένο παράδειγμα προέκυψε με την προσθήκη πολλαπλών συνδέσεων στην αρχική ιεραρχία *Metanet* που μέχρι τώρα είχαμε μελετήσει μέχρι και το σχήμα 21 που έχει μορφολογία τύπου δέντρου.



Σχήμα 22: Κωδικοποίηση Αλγορίθμου Agrawal, Borgida, Jagadish–Περίπτωση Πλέγματος

Όμως, η περίπτωση διαχείρισης του πλέγματος διαφέρει εν μέρει ως προς τον συνολικό τρόπο απόδοσης των κωδικών. Αναλυτικά, περιλαμβάνει την προσθήκη επιπρόσθετων βημάτων στην ήδη υπάρχουσα διαδικασία διαμόρφωσης πεδίων –ετικετών της περίπτωσης του δέντρου. Συνεπώς, η συνολική αυτή διαδικασία για την κωδικοποίηση ενός πλέγματος εξελίσσεται με την εξής συγκεκριμένη σειρά βημάτων:

1. Αρχικά, διαμορφώνουμε ένα βέλτιστο *spanning tree* που να καλύπτει επαρκώς το εν λόγω πλέγμα, που περιγράφει την εκάστοτε εισαγόμενη ιεραρχία όρων – κλάσεων. Εν συνεχεία θα επεξηγήσουμε περαιτέρω τα κριτήρια που καθορίζουν τα κριτήρια καθώς και την διαδικασία καθορισμού της βέλτιστης απόδοσης του αλγορίθμου κατασκευής του *spanning-tree*.
2. Εν συνεχεία γίνεται η ανάθεση των κατάλληλων πεδίων - κωδικών στους κόμβους – κλάσεις που περιλαμβάνονται στο προαναφερθέν *spanning tree* που καταλήξαμε στο βήμα 1. Η ανάθεση αυτή περιλαμβάνει την απόδοση ενός μοναδικού άνω ορίου, του *postorder index* αριθμού, έστω *post*, και ενός κάτω ορίου, που είναι ο χαμηλότερος *postorder index* αριθμός, έστω *index*, μεταξύ των απογόνων του εκάστοτε εξεταζόμενου κόμβου.
3. Κατόπιν, με την αντίστροφη τοπολογική σειρά εκτέλεσης που ακολουθούσαμε μέχρι τώρα στη σειρά απόδοσης των κωδικών - *top-down* -, δηλαδή με την *bottom-up* προσθέτουμε σε κάθε κόμβο πέραν του "πρωτεύοντος" διαστήματος που του αποδόθηκε με τα προαναφερθέντα βήματα 1 και 2 και τα λοιπά διαστήματα, που προέρχονται από την διασύνδεσή του με τους λοιπούς κόμβους-κλάσεις που δεν καλύπτονται με βάση το επιλεχθέν *spanning tree*. Με αυτόν τον τρόπο επιτυγχάνεται η αναπαράσταση των *non-tree-arcs* σε αντίστοιχα πεδία-ετικέτες κι κατά συνέπεια επιτυγχάνεται μια εκάστοτε υλοποίηση των μεταβατικών κλεισιμάτων για όλα τα υπάρχοντα μονοπάτια που εκ νέου διαμορφώνονται από τις *non-spanning tree* ακμές, κι οι οποίες δεν καλύπτονται από το υπάρχον *spanning tree*.

    - Μάλιστα σε ότι αφορά τα διαστήματα αυτά, εάν υφίσταται περίπτωση όπου ένα διάστημα έπεται άμεσα σε σχέση με ένα άλλο, θα μπορούσαν να

συνενωθούν. Επιλέγουμε όμως και δεν προχωρούμε σε μια συμπίεση τους σε ένα κοινό διάστημα. Κι αυτό γιατί με αυτόν τον τρόπο διατηρούμε την διακριτότητα των διαστημάτων αυτών, καθώς έτσι διασφαλίζεται η διασύνδεση του κάθε κόμβου με τους κόμβους της υπόλοιπης ιεραρχίας. Αντιθέτως εάν υπάρχει περίπτωση πλήρους (η διακριτότητα αποκλείει την όποια περίπτωση μερικής επικάλυψης) επικάλυψης ενός τέτοιου διαστήματος από το πρωτεύων διάστημα από την διάσχιση του *spanning tree*, σαφώς και δύναται η απορρόφηση των διαστημάτων αυτών χωρίς να υπάρχει ο κίνδυνος για την όποια απώλεια επιμέρους διασυνδετικής πληροφόρησης, που παρέχεται μέσω των μεταβατικών κλεισιμάτων.

3.3.4.2.1 Βέλτιστη Απόδοση υιοθετηθέντος spanning tree (tree-cover)

Ως γνωστόν οι κόμβοι – κλάσεις μιας ιεραρχίας μπορούν να καλυφθούν από πολλά επιμέρους *spanning trees*. Ωστόσο επιδίωξη μας θα πρέπει να είναι η εύρεση ενός βέλτιστου *spanning tree*. Ένα αξιοσημείωτο κριτήριο βέλτιστης απόδοσης ενός αλγορίθμου διαμόρφωσης του spanning tree έναντι άλλου έγκειται στην ελάττωση του πλήθος των διαστημάτων (*intervals*) που απαιτούνται για την όλη αναπαράσταση της υπό εξέταση ιεραρχίας και συνάμα στην προσπάθεια για διατήρηση της όλης πληροφορίας πρόσβασης και διασύνδεσης μεταξύ όλων των κόμβων μεταξύ τους. Αναλυτικά, η διασύνδεση κάθε κόμβου με τον κάθε ένα από τους υπόλοιπους κόμβους της ιεραρχίας αναπαρίσταται με την μορφή ενός έκαστου κατάλληλου «δεντρικού διαστήματος» (*tree-interval*), τα οποία εντάσσονται εν σειρά στο διαμορφωθέν *spanning tree*. Αντιθέτως οι λοιπές διασυνδέσεις οι οποίες χαρακτηρίζονται ως *non-tree-arcs* δεν ανήκουν στο επιλεχθέν *spanning tree*.

Αναφορικά με την βέλτιστη απόδοση αυτού του περιεχομένου, έχουμε να παρατηρήσουμε ότι κάθε διάστημα που αποδίδεται σε κάθε κόμβο προκαλεί επιπλέον μία μονάδα επιπλέον αποθήκευσης στο όλο υπάρχων *compressed closure* που προσδιορίζει την ιεραρχία. Επίσης, κάθε προσθήκη ενός διαστήματος έχει συνέπεια και επιβάρυνση στο κόστος χρόνου που χρειάζεται για την διαδικασία αναζήτησης του εν λόγω κόμβου.

Εάν θεωρήσουμε ότι κάθε διάστημα σχετίζεται αποκλειστικά με έναν κόμβο – κλάση, τότε κάθε ένα αποτελεί κι ένα επιπρόσθετη μονάδα βάρους για τον εν λόγω κόμβο. Στόχος συνεπώς της συγκεκριμένης προσπάθειας αυτής για βελτιστοποίηση έγκειται στην εύρεση μιας επαρκούς κάλυψης του δέντρου που σχηματίζεται από την εξεταζόμενη κάθε φορά ιεραρχία, ώστε να αποσκοπούμε έτσι στην ελαχιστοποίηση του συνολικού βάρους που προσδιορίζει από κοινού όλους τους κόμβους –κλάσεις. Προφανώς κι η ελαχιστοποίηση αυτή του συνολικού βάρους των κόμβων μιας ιεραρχίας σχετίζεται άμεσα με την επιδιωκόμενη επίσης ελαχιστοποίηση της επιζητούμενης καταχώρησής τους.

Εάν υφίσταται περίπτωση όπου ένας κόμβος αποτελείται τουλάχιστον από δυο διαστήματα, όπου το μεν ένα εμπεριέχει το δε άλλο, τότε και προφανώς μπορούμε να προχωρήσουμε σε διαγραφή της καταχώρησης του δεύτερου αυτού υποδιαστήματος. Για παράδειγμα εάν ένας κόμβος σχετίζεται με τα διαστήματα $[i_1, j_1]$, $[i_2, j_2]$, και έχουμε την εξής ακόλουθη σχέση μεταξύ των ορίων τους $i_1 \leq i_2 \leq j_2 \leq j_1$, τότε μόνο το πρώτο από τα δυο αυτά διαστήματα χρειάζεται να ληφθεί εν συνεχεία υπόψη και το άλλο παραλείπεται.

Αντιστοίχως μπορούμε να υποθέσουμε ότι έχουμε έναν κόμβο ο οποίος σχετίζεται με δυο τουλάχιστον υποδιαστήματα $[i_1, j_1]$, $[i_2, j_2]$, με $i_2 = i_1 + 1$. Μια πρώτιστη σκέψη θα ήταν να συνενώσουμε τα δυο υποδιαστήματα, σε ένα διάστημα $[i_1, j_2]$. Όμως αυτή η συνένωση γειτονικών διαστημάτων δημιουργεί εν συνεχεία μερικές φορές πολλαπλά προβλήματα, καθώς είναι τα αποτελέσματα της συνένωσης αυτής είναι άμεσα εξαρτώμενη από την εκάστοτε σειρά των διαστημάτων. Κάθε επιμέρους διάστημα πχ. $[i_1, i_2]$ αναπαριστά ένα υπόδεντρο της ιεραρχίας με εκάστοτε ριζικό κόμβο τον μεν κόμβο i_2 και τον δε κόμβο i_1 ως τον μικρότερο *postorder* αριθμό ανάμεσα στους κόμβους – παιδιά του εν λόγω υποδέντρου. Άρα εάν ένα υποδιάστημα εμπερικλείει ένα άλλο υποδιάστημα, τότε υπάρχει ένα συγκεκριμένο μονοπάτι διασύνδεσης μεταξύ του ριζικού κόμβου του πρώτου υποδιαστήματος με τον αντίστοιχο ριζικό κόμβο του δεύτερου υποδιαστήματος το οποίο έχει «απορροφηθεί» από τα όρια του πρώτου.

Όμως η διατήρηση των διαστημάτων αυτών που προέρχονται από τα *non-tree-arcs* και η μη «απορρόφηση» τους από άλλα γειτονικά, εκτενέστερα υποδιαστήματα είναι ζωτικής σημασίας καθώς με αυτόν τον τρόπο διαφυλάττεται η διασύνδεση τους με τους επιπλέον, μη - πρωτεύοντες πατρικούς αλλά μεταβατικά και από τους λοιπούς κόμβους

και συνεπώς όλα τα μη *non-tree* μονοπάτια που αρχικά δεν περιλαμβάνονται στην επιλογή του βέλτιστου, όπως αναφέραμε προηγουμένως, *spanning - tree*. Συγκεκριμένα, κάθε διάστημα είναι μοναδικό υπό την έννοια ότι εμπεριέχει αποκλειστικά την εμπεριεχόμενη πληροφορία για την διασύνδεση από τον τρέχων κόμβο προς τους κόμβους αυτούς, ακολουθώντας τα «βέλη» του δέντρου (*tree-arcs*) εξ αυτού, ή αντιστοίχως *non-tree-arcs*, δηλαδή διασυνδέσεις που δεν εμπεριέχονται στο υιοθετηθέν βέλτιστο *spanning tree*.

Στην προσπάθεια για την περαιτέρω βελτιστοποίηση του εκάστοτε δημιουργηθέντος συμπιεσμένου καθ' όλα μεταβατικού κλεισίματος (*compressed closure*) είναι σαφώς κι ως προς το κριτήριο της αποθήκευσής του. Αναλυτικά, προχωρώντας στον υπολογισμό ενός μεταβατικού κλεισίματος, ένα σημαντικό πρόβλημα είναι η αποφυγή περιττών (*redundant*) υπολογισμών. Συγκεκριμένα, ένας κατευθυνόμενος γράφος μπορεί να περιέχει ζευγάρια από επιμέρους κόμβους, οι οποίοι να συνδέονται μεταξύ τους μέσω πολλαπλών επιμέρους διακριτών μονοπατιών. Ενδέχεται ακόμη να υπάρχει η περίπτωση όπου να υπάρχουν κόμβοι οι οποίοι να έχουν το ίδιο σύνολο από «απογόνους» κόμβους. Αυτή η περίπτωση είναι μια από τις πιο σημαντικές περιπτώσεις «περιττών» υπολογισμών, καθώς τότε οι κόμβοι αυτοί ανήκουν σε επιμέρους δυνατά συστατικά σύνολα, στα οποία έχουν από κοινού το χαρακτηριστικό γνώρισμα του ακριβώς ιδίου συνόλου «απογόνων» κόμβων.

Η χρήση της αναπαράστασης διαστημάτων με ακέραιος αριθμούς όπως προτείνεται από τον κλασσικό αλγόριθμο του *Agrawal*, κατορθώνει και αντιμετωπίζει την δημιουργούμενη περιττότητα από τα πολλαπλά μονοπάτια μεταξύ δυο κόμβων. Κι αυτό επιτυγχάνεται μέσω της χρήσης αντί απλών συνόλων απογόνων, δέντρων απογόνων. Κι αυτό γιατί ένα δέντρο απογόνων περιέχει τους ίδιους κόμβους με το σύνολο απογόνων, αλλά επιπλέον εμπεριέχει και τα μονοπάτια που συνδέουν τους κόμβους αυτούς. Τα πολλαπλά μονοπάτια εντοπίζονται και οι κόμβοι αυτοί που υπάρχουν ήδη στο δέντρο απογόνων δεν ξαναπροσθέτονται. Με αυτόν τον τρόπο σαφώς υπάρχει το επιπλέον κόστος στον αναγκαίο χώρο και χρόνο για την διαμόρφωση του εν λόγω δέντρου απογόνων.

3.3.4.2.2 Αλγόριθμος διαμόρφωσης βέλτιστου spanning tree

Ο κάτωθι αλγόριθμος αποτελεί μία προσέγγιση η οποία επιχειρεί την βέλτιστη δυνατή κάλυψης ενός spanning tree ενός εισαχθέντος κατευθυνόμενου άκυκλου γράφου (*Directed Acyclic Graph -DAG*):

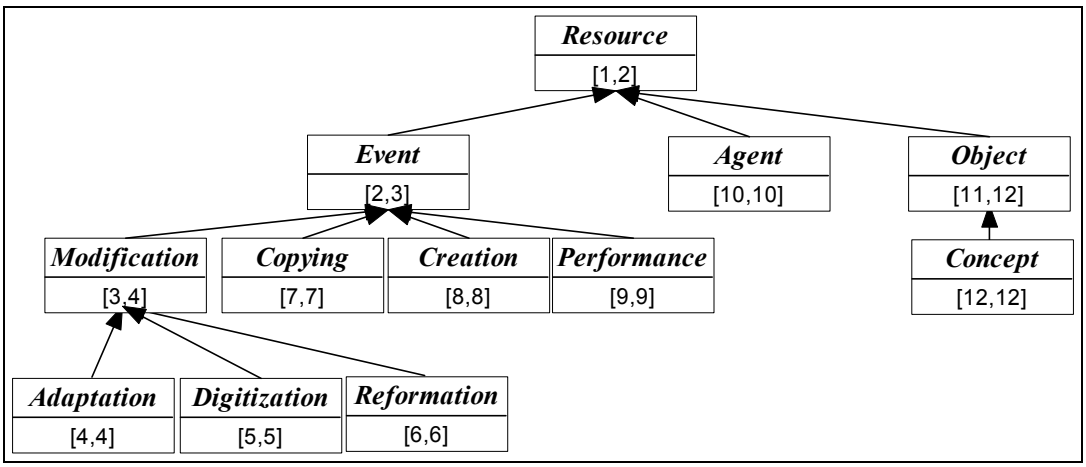
1. Αρχικά, προχωρούμε σε μια κατάλληλη τοπολογική ταξινόμηση του κατευθυνόμενο άκυκλου γράφου μας *G*.
2. Υποθέτουμε πως εάν υφίστανται κόμβοι του γράφου μας χωρίς προγόνους, θεωρούμε ότι συνδέονται σε έναν ιδεατό ριζικό κόμβο στο επίπεδο 0.
3. Για κάθε κόμβο του τοπολογικά ταξινομημένου μας γράφου, εν συνεχεία συγκρίνουμε ανά δυο κι κατόπιν αναδρομικά όλους τους άμεσους προγόνους του ως προς το πλήθος των επιμέρους προγόνων τους – ο προσδιορισμός των προγόνων γίνεται με βάση το πλήθος των κατευθυνόμενων tree arcs από τον εν λόγω εξεταζόμενο κόμβο. Συγκεκριμένα έχουμε έναν βρόγχο όπου εξετάζουμε τους κόμβους – άμεσους προγόνους του εκάστοτε εξεταζόμενου κόμβου, και τους ταξινομούμε σε απόλυτη φθίνουσα σειρά με βάση το πλήθος των προγόνων τους.
4. Αναδρομικά, ο επιλεχθέν κόμβος από τους κόμβους – προγόνους του αναδρομικά κληρονομεί και τους κωδικούς – πεδία των προγόνων του πατρικού του, καθώς και του ιδίου (έστω *i)*, δηλαδή $\{i\} \cup$ *πρόγονοι(i)*.

Ο παραπάνω αλγόριθμος διαμόρφωσης βέλτιστου spanning-tree εξασφαλίζει ότι :

1. Για κάθε κόμβο η επιλογή του βέλτιστου άμεσου πατρικού του κόμβου σηματοδοτεί άμεσα ότι αυτός κι οι άμεσοί του πρόγονοι, οι οποίοι δεν είναι ταυτοχρόνως και άμεσοι πρόγονοι και των υπόλοιπων άμεσων προγόνων του εκάστοτε κόμβου, θα κληρονομήσουν τα δεντρικά διαστήματα (*tree intervals*) που σχετίζονται με τον εν λόγω κόμβο.
2. Εάν ένα διάστημα $[i_1, i_2]$ απορροφάται από ένα άλλο διάστημα $[j_1, j_2]$, τότε υπάρχει ένα κάποιο μονοπάτι από το άκρο i_1 στο άκρο j_2, περικλείοντας αποκλειστικά μόνο *tree – arcs*.

3.3.5 Αλγόριθμος Schubert

Τέλος, μια συναφής παραλλαγή του αλγορίθμου κωδικοποίησης του *Dietz*, που προείδαμε, είναι το προτεινόμενο σχήμα από τον *Schubert* [SPT87], το οποίο διαφοροποιείται ως προς το ότι χρησιμοποιεί τις αντίστροφες συνθήκες των ίδιων των επερωτήσεων του *Agrawal et al* (βλ. Πίνακα 6). Αναλυτικά, ο κωδικός του κάθε κόμβου *u* είναι ένα πεδίο του τύπου [*pre(u), index(u)*], όπου η καινοτομία έγκειται στο πεδίο *index(u),* το οποίο είναι ο υψηλότερος σε τιμή *postorder* αριθμός μεταξύ των απογόνων του *u* κόμβου (βλ. Σχήμα 23).



Σχήμα 23 : Κωδικοποίηση Αλγορίθμου Schubert

3.3.6 Σύγκριση Κωδικοποιήσεων με Διαστήματα

Με βάση όλα τα προαναφερθέντα μπορούμε να προχωρήσουμε σε μια συγκριτική αναφορά των επιμέρους αυτών σχημάτων κωδικοποίησης που ανήκουν εν γένει στην κατηγορία των κωδικοποιήσεων με την χρήση κατάλληλων διαστημάτων. Αναμφισβήτητα, το εκτεταμένο *postorder* σχήμα των *Agrawal, Borgida* και *Jagadish* κρίνεται ως το βέλτιστο όλων των προαναφερθέντων, για τους εξής λόγους :

1. Εξ αρχής, απαιτεί αισθητά πιο μικρά πλήθη από index κωδικούς (κι αντιστοίχως κόστη ενημέρωσης), καθώς χρειαζόμαστε μόνο την ύπαρξη ενός και μόνο *B-tree*

στη τιμή του *post* πεδίου των κωδικών που παράγει. Αντιθέτως, οι κωδικοί του *Dietz* αλγορίθμου [D82, DS87], απαιτούν τον ορισμό δεικτών *B-trees* και στα δυο πεδία *pre(u)* και *post(u)*. Μάλιστα στην περίπτωση της παραλλαγής του *Zhang* αλγορίθμου [ZNWLL] χρειάζεται πέρα των δεικτών στα πεδία *pre(u)* και *post(u*), επιπλέον κι έναν δείκτη στο πεδίο *depth(u)*.

2. Κρίνεται επίσης βέλτιστο στην αποτίμηση των επερωτήσεων από τις υπάρχουσες, δεδομένες SQL μηχανές, οι οποίες χρησιμοποιούν τους δείκτες των B-trees, καθώς οι βασικές συνθήκες για τις δομημένες συσχετίσεις μεταξύ των κόμβων είναι πολύ πιο απλές. Αντιθέτως, οι κόμβοι στο σχήμα που προτείνουν οι *Li* και *Moon* περιλαμβάνουν την χρήση αριθμητικών λειτουργιών σε όλες τις επερωτήσεις.
3. Τέλος, με μοναδικό τρόπο παρουσιάζει μια μεγάλη κι ενδιαφέρουσα δυνατότητα συμπίεσης των παραγόμενων κωδικών από τους εισαγόμενους γράφους, κατ' αρχήν απορροφώντας τα υπαγόμενα διαστήματα είτε πλήρως εάν συμπίπτουν τα άκρα μεταξύ τους, είτε εν μέρει εάν εμπεριέχονται στο ενδιάμεσο που ορίζουν τα άκρα κάποιου άλλου. Επίσης, υπάρχει κι η περίπτωση συμπίεσης, η οποία στηρίζεται στην συνένωση συνεχόμενων πεδίων, που να προέρχονται από την προσθήκη *non-spanning tree* ακμών.

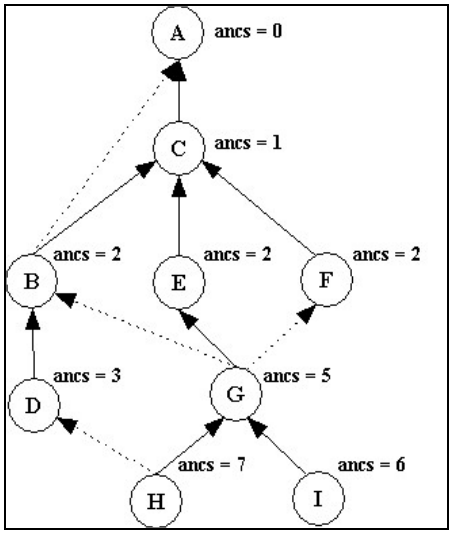
3.4 Σύγκριση Σχημάτων Κωδικοποίησης για DAGs

Για παράδειγμα ας θεωρήσουμε το γράφο *DAG D*, ο οποίος απεικονίζεται στο σχήμα 24. Στον απεικόνιση αυτή του γράφου *D* οι κλάσεις αναπαρίστανται με τους κόμβους και αντιστοίχως οι ακμές του με τις *isA* διασυνδέσεις που πρέπει να ορίζονται μεταξύ τους. Η άμεση σύνδεση μέσω non-tree ακμής μεταξύ των κόμβων *B* και *A* χαρακτηρίζεται ως πλεονάζουσα και περιττή (*redundant*), αλλά έχει διαπιστωθεί πολλάκις ότι είναι πολύ συχνά τέτοια φαινόμενα πλεοναζόντων (*redundant*) κόμβων στα *RDF/S* σχήματα που βρίσκονται ευρέως στο διαδίκτυο [18].

Εν συνεχεία, και στο ίδιο σχήμα παραθέτουμε έναν αναλυτικό πίνακα με τους κωδικούς που προκύπτουν εάν εφαρμόσουμε και για τις τρεις επιμέρους διακριτές

κωδικοποιήσεις με διαστήματα. Στο σημείο αυτό θα πρέπει να αναφερθεί ότι η κωδικοποίηση των *Li & Moon* που προείδαμε για την περίπτωση δέντρου μπορεί να επεκταθεί και για την περίπτωση πλέγματος, υιοθετώντας σε γενικά πλαίσια την σχετική προαναφερθείσα στον αλγόριθμο του *Agrawal* διαδικασία διαμόρφωσης του *spanning-tree* αλλά και της *bottom-up* διάδοσης των πεδίων –ετικετών των *non-tree* ακμών. Σαφώς, και η διάδοση και η ενδεχόμενη συνένωση μεταξύ των πεδίων –ετικετών - μόνο μέσω της διερεύνησης σχέσεως υπαλληλίας - διαφέρει εδώ καθώς υπάρχει διαφορετική σημασιολογία στα άκρα των πεδίων-ετικετών. Αναφορικά, η διερεύνηση ύπαρξης υπαλληλίας μεταξύ δυο πεδίων έστω Α και Β γίνεται εάν το αριστερό άκρο του Β (που έχει προκύψει από την *pre-order* διάσχιση του επιλεχθέντος *spanning-tree*) προηγείται αριθμητικά του αντίστοιχου άκρου του Α, και βέβαια εάν το δεξιό άκρο-πλήθος του Β είναι μεγαλύτερο του Α.

Η ίδια η βασική ιδέα που χρησιμοποιήσαμε εξ αρχής στην περίπτωση του *Agrawal et al*, μπορεί να χρησιμοποιηθεί πέρα της απλής τροποποιήσεως που κάναμε στην περίπτωση του σχήματος των *Li & Moon*, που δεν είναι παρά μια άλλη περίπτωση σχήματος κωδικοποίησης με διάνυσμα (*interval-based*) και στην περίπτωση ενός σχήματος κωδικοποίησης με την χρήση προθέματος όπως είναι το σχήμα του *Dewey*. Συγκεκριμένα, ακολουθείται η ίδια διαδικασία επιλογής του *spanning-tree* όπως περιγράψαμε και πιο πάνω με την διαφοροποίηση όμως ότι η διάδοσης των αλφαριθμητικών ετικετών, των επιπρόσθετων στο *spanning-tree*, *non-tree* ακμών γίνεται πλέον με *top-down* τοπολογική σειρά διάδοσης. Διαφοροποίηση υπάρχει επίσης και σε όποια ενδεχόμενη συνένωση μεταξύ των ετικετών - μόνο και πάλι μέσω της διερεύνησης σχέσεως υπαλληλίας – π.χ. ο Β κόμβος υπάγεται στον Α εάν κι εφόσον εάν η πρωτεύουσα αλφαριθμητική ετικέτα του Α έχει μεταδοθεί αυτούσια σαν δευτερεύουσα – δηλαδή δεν έχει παραχθεί μέσω του *spanning-tree* – ετικέτα του Β κόμβου ή η ετικέτα του Α είναι πρόθεμα της όποιας ετικέτας του Β. Συνεπώς, η μόνη δυνατή συνένωση είναι η ύπαρξη κοινών δευτερευόντων ετικετών μέσω των *non-tree* ακμών.



Σχήμα 24: Παράδειγμα Γράφου DAG προς κωδικοποίηση

Node	Agrawal et al			Li and Moon			Dewey		
	Spanning Tree	Graph Propagation	Compression	Spanning Tree	Graph Propagation	Compression	Spanning Tree	Graph Propagation	Compression
A	[1,9]	[1,2][3,3] [3,5][3,5]	[1,9]	[1,9]	[2,2][7,1] [6,3][6,3]	[1,9]	'1'		
B	[1,2]	[3,3][3,5]	[1,5]	[2,2]	[7,1][6,3]	[2,2][6,3]	'111'	'1'	'111'
C	[3,8]	[3,3][3,5] [3,5]	[3,8]	[4,8]	[7,1][6,3] [6,3]	[4,8][6,3]	'11'		
D	[1,1]	[3,3]		[3,1]	[7,1]		'1111'	'1'	'1111'
E	[3,6]			[5,4]			'112'		
F	[7,7]	[3,5]		[9,1]	[6,3]		'113'		
G	[3,5]			[6,3]			'1121'	'111' '1' '113'	'1121' '111''113'
H	[3,3]			[7,1]			'11211'	'111' '1' '1' '113' '1111'	'11211' '111''113' '1111'
I	[4,4]			[8,1]			'11212'	'111' '1' '113'	'11212' '111''113'

Σχήμα 25: Συμπίεση Κωδικών Γράφων : a) Agrawal et Al b) Li & Moon c) Dewey

Με βάση το παράδειγμα του παραπάνω αυτού γράφου (βλ. Σχήμα 24), μπορούμε ευθύς να διαπιστώσουμε πως οι επερωτήσεις για την αναζήτηση των *precedings /followings* , που είχαμε δει με την μορφή φορμαλισμού στους πίνακες 1, 2, 3, και 4 μαζί με τις λοιπές βασικές επερωτήσεις για δέντρο, δεν υφίστανται πλέον ούτε σαν έννοια στην περίπτωση της δομής ενός οιοδήποτε γράφου. Με βάση το προτεινόμενο από τους *Agrawal, Borgida* και *Jagadish* σχήμα κωδικοποίησης [ABJ89], καταρχήν επιλέγουμε ένα βέλτιστο *spanning-tree*, έστω *T,* όπου η βελτιστότητά του έγκειται με βάση τα

κριτήρια που προείδαμε, δηλαδή επιλέγεται ο πρόγονος *n'* κάθε κόμβου *n* να ανήκει στο *spanning-tree*, μέσω καθαρών ακμών, εάν και μόνο εάν ο *n'* έχει τον μέγιστο αριθμό προγόνων κοκ. Για παράδειγμα μπορούμε να δούμε στο παραπάνω σχήμα 24 ότι η ακμή από τον κόμβο *B* στον *C* ανήκει στο *spanning-tree*, ενώ η ακμή από τον κόμβο *B* στον *A* δεν ανήκει. Για αυτό και τους μεν πρώτους τους απεικονίζουμε με σταθερές συνεχόμενες γραμμές, ενώ τους δε δεύτερους με διακεκομμένες γραμμές. Μόνο μη πλεονάζοντες ακμές ανήκουν στο βέλτιστο *spanning-tree*. Με βάση όλα αυτά μπορούμε από τους κωδικούς του σχήματος 25 να διαπιστώσουμε πως για κάθε μία *non-spanning tree* ακμή (πχ από τον κόμβο *H* στον κόμβο *D*) το πεδίο-κωδικός του κόμβου "*αφετηρία*" (πχ [3, 3] μεταδίδεται στον κόμβο "*προορισμό*" (πχ *D*) και αναδρομικά εν συνεχεία σε όλους τους λοιπούς του προγόνους (πχ οι κόμβοι *B, C, A*).

Κατά την διάρκεια της μετάδοσης των πεδίων – κωδικών αυτών με κατεύθυνση *προς τα επάνω*, τα διαστήματα αυτά των απογόνων κόμβων ενδέχεται να απορροφηθούν από αυτά των προγόνων τους (πχ το διάστημα [3,3], που είναι πρωτεύων κωδικός του κόμβου *H* απορροφιέται από τα διαστήματα των κόμβων C και όπως φαίνεται και στον παραπάνω σχήμα 25). Συνεπώς, όλα τα διαστήματα μπορούν να απορροφηθούν από τον κωδικό ενός κόμβου (είτε αυτός προέρχεται από το βασικό *spanning-tree*, είτε από τα *propagations* που γίνονται διαμέσου των *non-spanning tree* ακμών) και σε κάθε περίπτωση αποτελεί τον κοντινότερο κοινό τους πρόγονο (*nca*). Ένα άλλο είδος δυνατής απορρόφησης είναι η συνένωση τυχόν συνεχόμενων διαστημάτων σε ένα κοινό (πχ τα διαστήματα [1,2] και [3,3] μπορούν να συνενωθούν στο [1,3]). Προφανώς , κι αυτή η συνένωση των διαστημάτων εξαρτάται αποκλειστικά στην σειρά με την οποία είναι ταξινομημένες οι ακμές που ανήκουν στο επιλεχθέν spanning tree [ABJ89]. Επίσης η συνένωση αυτή εγγυάται πάντα και για όλες τις δυνατές περιπτώσεις εξεταζόμενων γράφων τον κανόνα συνυπολογισμού των διαστημάτων μεταξύ τους, που να περικλείει την σχέση κόμβου – προγόνου του (πχ κατόπιν της συνένωσης ο κόμβος *B* είναι πρόγονος του κόμβου *D* αλλά και του *H*). Όμως το σημαντικό της μειονέκτημα είναι ότι επηρεάζει σημαντικά την πάνω από όλα επιθυμητή διακριτοποίηση των κόμβων μεταξύ τους με βάση τον *postorder* αριθμό τους (βλέπε στο κεφάλαιο 5 της αξιολόγησης των *DAGs*) κι άρα γεννάται θέμα εφαρμογής της, κάτι που μπορεί να αξιολογηθεί σε σχέση με την εμβέλεια της συμπίεσης συνολικά των κωδικών του σχήματός μας σε σχέση με

αυτό το είδος της συνένωσης. Πάντως, παρατηρούμε πως με την χρήση και των ειδών αυτών συμπίεσης επιτυγχάνουμε στην καταγραφείσα κωδικοποίηση του σχήματός μας την χρήση δυο και μόνο επιπρόσθετων διαστημάτων (συγκεκριμένα για τους κόμβους *D* και *E*) για το σύνολο των 4 *non-spanning tree* ακμών του γράφου μας.

Η ίδια συμπίεση μπορεί να εφαρμοστεί όπως εύκολα μπορούμε να συμπεράνουμε επίσης και για τα άλλα σχήματα κωδικοποίησης που βασίζονται στην χρήση διαστημάτων, όπως είναι στο παραπάνω παράδειγμα η περίπτωση του σχήματος με διαστήματα των *Li* και *Moon* [LM01]. Όμως, η αποτελεσματικότητα του ρυθμού συμπίεσης σε αυτήν την περίπτωση είναι αισθητά μειωμένη, καθώς δεν μπορεί να υφίσταται συχνά η περίπτωση υπαλληλίας μεταξύ των διαστημάτων (με βάση και τις σχετικές συνθήκες ελέγχου υπαλληλίας του πίνακα 5). Χαρακτηριστικό παράδειγμα της απορρόφησης αυτής στο σχήμα κωδικοποίησης των *Li* και *Moon* είναι το ότι το διάστημα [6,3] απορροφάται από το διάστημα [7,1].

Το σχήμα του *Dewey*, που βασίζεται στην χρήση προθεμάτων [OCLC] μπορεί εξίσου εύκολα να επεκταθεί και να διαχειριστεί την περίπτωση των επιπλέον κωδικών από την περίπτωση των *DAGs*. Ομοίως χρησιμοποιούμε την ίδια διαδικασία διαμόρφωσης κι επιλογής του spanning-tree, με την διαφορά όμως τώρα ότι η μεταδιδόμενοι κωδικοί μεταβαίνουν *προς τα κάτω* (*downwards*), δηλαδή από τον κόμβο "*προορισμός*" των *non-spanning tree* ακμών (πχ ο κόμβος *A*) στον κόμβο " *αφετηρία*" (πχ κόμβος *B*) και αναδρομικά στους απογόνους του (πχ κόμβοι *D, G, H* και *I*). Στην περίπτωση αυτή του σχήματος του *Dewey* η μόνη δυνατή συμπίεση είναι η απορρόφηση ενός κωδικού όταν ήδη εμφανίζεται σαν πρόθεμα ενός άλλου υπάρχοντος. Για παράδειγμα ο κωδικός '1' απορροφάται από τους κωδικούς '111', '11111' κλπ. Συνεπώς, όπως παρουσιάζεται εύγλωττα και στον πίνακα με τους κωδικούς του εν λόγω γράφου κατόπιν της κωδικοποίησης του *Dewey* απαιτούνται 6 συνολικά επιπρόσθετοι κωδικοί (που είναι αντιστοίχως για τους κόμβους *G*, *H* και *I*).

3.5 Συνολικά Συμπεράσματα

Συμπερασματικά, τα σχήματα κωδικοποίησης με την χρήση διανυσμάτων με δηφία (*bit-vector based*) δεν μπορούν να υποστηρίζουν με τον πλέον κατηγορηματικό τρόπο όλες τις ζητούμενες από εμάς βασικές επερωτήσεις, οι οποίες υλοποιούνται σε *SQL* μορφή. Από την άλλη μεριά τα σχήματα κωδικοποίησης με την χρήση προθεμάτων (*prefix -based*) παρουσιάζουν με μια τρομερή ευκολία την ικανότητα να παρέχουν τις κατάλληλες αυτές εκφράσεις για την υλοποίηση των επερωτήσεων για τους *προγόνους /απογόνους*, καθώς βασίζονται σε επιμέρους τελεστές σύγκρισης αλφαριθμητικών και επίσης επιτρέπουν τις απλές αυξητικές ενημερώσεις.

Τέλος, από την ευρύτερη κατηγορία των σχημάτων κωδικοποίησης με την χρήση διαστημάτων παρατηρήσαμε πως το προτεινόμενο εκτεταμένο σχήμα των *Agrawal, Borgida* και *Jagadish* (το οποίο θα το αναφέρουμε κι έως *PInterval*) εμφανίζει πληθώρα πλεονεκτημάτων ως προς την επίδοσή του, όπως είναι η συμπίεση των κωδικών στην περίπτωση που εξετάζουμε ιεραρχίες *DAGs*, καθώς και ικανοποιητική αποτίμηση κόστους των βασικών επιζητούμενων επερωτήσεων από τις υπάρχουσες συνήθεις *SQL* μηχανές. Στην επόμενη ενότητα θα δούμε μια αναλυτική πειραματική αξιολόγηση της εν γένει συμπεριφοράς των σχημάτων *PInterval* και του σχήματος του αλγόριθμου με *Unicode* προθέματα κωδικοποίησης (το οποίο θα αναφέρεται ως *UPrefix*) με κριτήρια τον χώρο αποθήκευσης και τον χρόνο εκτέλεσης των επερωτήσεων.

Κεφάλαιο 4

Αξιολόγηση Σχημάτων Κωδικοποίησης - με πρόθεμα και με - διάστημα - Περίπτωση των Δέντρων

Στα πλαίσια του κεφαλαίου αυτού θα προχωρήσουμε σε διάφορες συγκρίσεις κι εν συνέχεια αξιολογήσεις γύρω από τα χαρακτηριστικά γνωρίσματα και τις ιδιότητες που παρουσιάζουν δυο αντιπροσωπευτικά δείγματα των δυο αυτών βασικών κατηγοριών κωδικοποιήσεων με χρήση προθέματος (*prefix-based*) και με διάστημα (*interval-based*). Αναφορικά, οι δυο αυτοί αντιπροσωπευτικοί αλγόριθμοι είναι ο μεν αλγόριθμος κωδικοποίησης, βασισμένος σε post-order αρίθμηση των *Agrawal, Borgida* και *Jagadish- PInterval* - (*Interval-based*) [ABJ89] και του *Unicode Dewey* (*Prefix-based*) [OCLC] αντίστοιχα.

Για την πραγματοποίηση όλων αυτών των πειραμάτων χρησιμοποιήσαμε σαν πεδίο δοκιμής τα *RDF* σχήματα του *ODP* καταλόγου (έκδοση 16-01-2001), που προαναφέραμε στην ενότητα του κεφαλαίου 2 και χαρακτηρίζονται από την ύπαρξη ιεραρχιών με μονές σχέσεις *isA* μεταξύ των κλάσεων. Εξίσου πάντως και για την περίπτωση των κατευθυνόμενων άκυκλων γράφων (*DAGs)* – στην περίπτωση αυτή των *DAGs* στις προϋπάρχουσες ιεραρχίες του *ODP* καταλόγου προσθέτονται με συνθετικό τρόπο παραγόμενες πολλαπλές σχέσεις *isA* κατά αντιστοιχία. Την περίπτωση αυτή του αντικειμένου αξιολόγησης των *DAGs* θα την αναπτύξουμε εν συνεχεία στο επόμενο κεφάλαιο 5 σαν ξεχωριστή ενότητα.

Η διεξαγωγή των όλων των πειραμάτων έγινε σε μηχάνημα τύπου *Sun-Blade*-1000, με έναν *Ultra SPARC-III 750MHz* επεξεργαστή και με 512 *MB* κύριας μνήμης, χρησιμοποιώντας την πλατφόρμα της *PostgreSQL* (έκδοση 7.2.1), η οποία χαρακτηρίζεται από την δυνατότητά της αποθήκευσης των δεδομένων της με την μορφή της *Unicode* αναπαράστασης. Στη συνέχεια για τις ανάγκες φόρτωσης των δεδομένων σε μια βάση δεδομένων στην πλατφόρμα της *PostgreSQL*, και κατόπιν

άσκησης επερωτήσεων πάνω σε αυτήν, ορίσαμε και χρησιμοποιήσαμε επιμέρους 1000 ενταμιευτές (*buffers*), έκαστος μεγέθους 8 *Kbytes*. Η λογική της επιλογής αυτής του καθορισμού του πλήθους των ενταμιευτών ως βέλτιστης θα αναπτυχθεί στη συνέχεια του κεφαλαίου. Φορτώσαμε τις 16 ιεραρχίες κλάσεων του *ODP* καταλόγου, με συνολικό αριθμό 253215 επιμέρους θέματα (*topics*), τα οποία αναπαρίστανται υπό τη μορφή *RDF* αρχείου περίπου 51 *Mbytes* χώρο. Επίσης, να αναφέρουμε ότι στους παραγόμενους κωδικούς έχουν κατασκευαστεί κατάλληλα δείκτες (B^+-*tree*), εφόσον πρότερα οι κωδικοί έχουν ταξινομηθεί κατά αύξουσα σειρά ως προς τις επιμέρους τιμές τους.

Συγκεκριμένα, στην ενότητα αυτή θα ασχοληθούμε με την σχεσιακή αναπαράσταση των δυο προαναφερθέντων αλγόριθμων κωδικοποίησης και εν συνεχεία εξίσου και με την αξιολόγηση των ζητούμενων επερωτήσεων (βλ Σχήμα 10) χρησιμοποιώντας την πλατφόρμα της *PostgreSQL* πάνω στην αναπαράσταση αυτή. Συγκεκριμένα, ένα σημείο στο οποίο θα επικεντρώσουμε το ενδιαφέρον μας θα είναι η βέλτιστη δυνατή μετάφραση των διαφορετικών τύπων των επερωτήσεων που επιζητούμε στη μορφή *SQL* επερωτήσεων για την πλέον απλή περίπτωση σχημάτων διαδικτυακών πυλών, που είναι προφανώς τα δέντρα υποκλάσεων.

4.1 Χρονική Διάρκεια Κατασκευής Κωδικών

Ένα πρώτιστο κριτήριο σύγκρισης των δυο σχημάτων κωδικοποίησης είναι ο χρόνος που απαιτείται για την δημιουργία των αντίστοιχων κωδικών για τις κλάσεις των εξεταζόμενων *RDF* σχημάτων. Ο χρόνος αυτός ισούται με τον αναγκαίο χρόνο κατασκευής κωδικών για όλες τις κλάσεις του κάθε εξεταζόμενου προς κωδικοποίηση *RDF* σχήματος. Στον Πίνακα 1 παραθέτουμε τους ληφθέντες χρόνους (μετρημένους σε *seconds*) που χρειαστήκαμε για την κωδικοποίηση των ιεραρχιών του *ODP* καταλόγου τόσο με τον αλγόριθμο του *Agrawal* (βλ. Σχήμα 26) όσο και του *Unicode Dewey* (βλ. Σχήμα 27). Η μέτρηση των χρόνων έγινε σε ένα μηχάνημα τύπου *Sun-Blade*-1000, με έναν *Ultra SPARC-IV 400MHz* επεξεργαστή και με 1152 *MB* κύριας μνήμης.

Algorithm ***Agrawal***()
Parameters:
(1) Input Intermediate Structure
Procedure Description:
(1) Αρχικοποίηση Στοίβας *Stack*
(2) *Create_Range_Codel*(Input Intermediate Structure)
(3) *Propagate_LIC* (*Random Class*, Input Intermediate Structure)

όπου

Algorithm ***Create_Range_Codel*()**
Parameters:
(1) *Current Class*
(2) Input Intermediate Structure
Procedure Description:
(1) *Count* =0
(2)*Loop* ∀ Class ∈ Input Intermediate Structure
{
(1) Εισαγωγή *Current Class* στην στοίβα *Stack*
(2) If (*Stack* δεν είναι κενή)
{
(2.1) Εξάγουμε το Πρώτο Στοιχείο της *Stack* → *elem*
(2.2) If (*elem* έχει «σημειωθεί»)
{
(2.1)*Count* = *Count* + 1
(2.2) *Post* = *Count*
}
else if (*elem* έχει «σημειωθεί»)
{
(2.3) Σημειώνεται το *elem*
(2.4) Εισάγουμε το *elem* στην *Stack*
(2.5) Εισάγω κάθε θυγατρικό κόμβο του *elem* στην *Stack*
}
(3) Αριθμός Απογόνων *elem* → *Index*
(4) Αποθήκευση *Current Class* πεδίου-κωδικού [*Index, Post*]
}

και

Algorithm ***Propagate_LIC*** **()**
Parameters:
(1) Input Intermediate Structure
Procedure Description:
Loop (∀ *Class* εκτός *Root Class*) | *Class* ∈ Input Intermediate Structure
{
(1)Αντικατάσταση του *Index* του Πατρικού του *Current Class* με το *Index* του *Current Class*
(2) Propagate_LIC (*Father_Current Class*, Input Intermediate Structure)
}

Σχήμα 26 : Αλγόριθμος του Agrawal et al – σε μορφή ψευδοκώδικα

Algorithm ***UDewey*** **()**
Parameters:
(1) Input Intermediate Structure[1]
Procedure Description:

Root node → Code = First Unicode Character

Loop∀ Class except Root Class
{
if ((∃ Father class of Current Class) && (Father Class has stored code[2]))
{
Create_Code (Current Class, Input Intermediate Structure[1])
}
else if ((∃ Father Class of Current Class) && (Father Class doesn't have stored code[2]))
{
(1) *UDewey*(Father Class, Input Intermediate Structure[1])
(2) if ((∃ Father class of Current Class) && (Father Class has stored code[2]))
{
Create Code (Current Class, Input Intermediate Structure[1])

```
                    }
            }
}
```

όπου

Algorithm ***Create_Code*** **()**
Parameters:
(1) Current Class,
(2) Input Intermediate Structure[1]
Procedure Description:

(1) Use of Father Class's Code (label)
(2) Number of Brother Nodes of Current Class that have been encoded → *num_brothers*
(3) Correspondence of *num_brothers* to a Unicode Character → *UC(num_brothers)*
(4) Concatenation label + *UC(num_brothers)*

Σχήμα 27 : Αλγόριθμος του Unicode Dewey – σε μορφή ψευδοκώδικα

Η καταγραφή του χρόνου έγινε με συναρτήσεις που μετρούσαν τον αναγκαίο χρόνο μηχανής για την ολοκλήρωση της εκτέλεσης της κάθε κωδικοποίησης.

ODP RDF schemas	**Agrawal et al**	**Unicode Dewey**
netscape_schema	0,470	0,276
news_schema	0,665	0,327
kat_schema	0,721	0,383
home_schema	1,340	0,666
health_schema	2,525	1,373
shopping_schema	2,481	1,360
games_schema	3,247	1,674
computers_schema	4,446	2,418
reference_schema	5,634	2,556
business_schema	4,853	2,181
recreation_schema	5,034	2,299
science_schema	7,339	3,318
sports_schema	7,448	3,971
society_schema	13,122	6,992
arts_schema	17,587	9,715
regional_schema	128,768	52,889
Σύνολο	**205,680**	**92,398**

Πίνακας 7 : Χρόνος Κατασκευής Κωδικών (σε seconds)

Με βάση τα αποτελέσματα του παραπάνω πίνακα 7, παρατηρούμε έναν δεδομένο επιπλέον κόστος στην κωδικοποίηση που προσφέρει ο αλγόριθμος του *Agrawal et al* έναντι του αντίστοιχου αλγορίθμου του *Unicode Dewey*. Κι αυτό

σαφώς κι οφείλεται στην διαδικασία παραγωγής των κωδικών στην μία και στην άλλη περίπτωση, καθώς στην περίπτωση του αλγόριθμου του *Agrawal*, πέραν του κόστους της παραγωγής του μοναδικού κι απόλυτα διακριτού δεξιού άκρου του προσδοθέντος κωδικού του – *postnum* κωδικού – υπάρχει και το επίσης αναγκαίο κόστος παραγωγής και του αριστερού άκρου/ κωδικού του πεδίου αυτού – *index* κωδικού -, το οποίο όπως είδαμε στην περιγραφή του κλασσικού αλγορίθμου του *Agrawal*, προέρχεται αναδρομικά από την αντίστοιχη υπάρχουσα μικρότερη τιμή των απόγόνων του εκάστοτε εξεταζόμενου κόμβου. Προφανώς, κι η αναδρομική αυτή ενημέρωση έχει μια σημαντική επιβάρυνση στην όλη διαδικασία παραγωγής κωδικών, και η οποία συναρτάται άμεσα από το βάθος και το πλήθος των υπό εξέταση *RDF* ιεραρχιών, κάτι που σαφώς και διαφαίνεται και από τους παραπάνω καταγραφέντες χρόνους. Μάλιστα, από τις καταγραφείσες τιμές παρατηρείται ότι ο επιπρόσθετος αυτός χρόνος παραγωγής και απόδοσης των *index* κωδικών σε όλες τις κλάσεις - κόμβους είναι περίπου ίδιας τάξης μεγέθους με τον αντίστοιχο χρόνο παραγωγής των *postnum* κωδικών.

Σε αντίθεση η παραγωγή των κωδικών με βάση τον αλγόριθμο του *Unicode Dewey*, στηρίζεται στην αναδρομή μόνο κατά ένα βήμα για την λήψη του κωδικού του άμεσου πατρικού του κόμβου και με βάση αυτόν την προσθήκη ενός νέου αλφαριθμητικού για τη σύνθεση και την δημιουργία του κωδικού του. Η όλη λειτουργία σαφώς κι είναι μικρότερη σε χρονική διάρκεια από την αντίστοιχη του *Agrawal et al*.

4.2 Σχεσιακή Αναπαράσταση και Μέγεθος Αποθήκευσης

Η *RDF/S* ιεραρχία κλάσεων του καταλόγου του *ODP* – κατά αναλογία και κάθε άλλου αντίστοιχου ή παρόμοιου καταλόγου – μπορεί να αναπαρασταθεί ικανοποιητικά από ένα και μόνο πίνακα. Στον πίνακα αυτό μπορούμε να συμπεριλάβουμε το όνομα της κλάσης, το οποίο και παίζει ρόλο αναγνωριστικού, καθώς και το σύνολο των ονομάτων των άμεσων υποκλάσεών της. Δεδομένου όμως ότι τα ονόματα που προσδίδονται να αναπαριστούν τις κλάσεις είναι εξαιρετικά μεγάλα, κι αυτό καθώς χρησιμοποιούνται σαν προθέματα όλα τα επιμέρους ονόματα όλων των κλάσεων που μεσολαβούν μεταξύ της εκάστοτε κλάσης και του ριζικού κόμβου της ιεραρχίας.

Συνεπώς, χρειάζεται να υιοθετήσουμε ένα καινούργιο, κανονικοποιημένο σχήμα αναπαράστασης βάσεων δεδομένων. Αναλυτικά αυτό το σχήμα θα είναι της εξής γενικής μορφής :

Class (*Id : int4, Name : varchar(256)*)

SubClass (*Id : int4, Father_Id : int4*)

Αναλύοντας το επιμέρους αυτό σχήμα αναπαράστασης, θα πρέπει καταρχήν να προσδιορίσουμε ότι σαν *id* θεωρούμε ένα απόλυτα διακριτό αναγνωριστικό κάθε κλάσης, το οποίο παράγεται δυναμικά κατά την διαδικασία προσπέλασης των κλάσεων από το *RDF* σχήμα. Σαν αναγνωριστικά μπορούμε να χρησιμοποιήσουμε τους κωδικούς ή μέρος αυτών (όπως π.χ. είναι το πεδίο *post*) που παράγονται από τους αλγορίθμους του *Agrawal et al* και του *Unicode Dewey* κι επομένως με τον τρόπο αυτό να αποφύγουμε την ανάγκη χρήσης αυξητικών αναγνωριστικών.

Είναι προφανές, ότι θα πρέπει όμως για κάθε έναν επιμέρους αλγόριθμο να προχωρήσουμε σε κάποιες αναγκαίες κι καθ' όλα επιτακτικές τροποποιήσεις του παραπάνω πίνακα *SubClass* της σχεσιακής αναπαράστασης.

Αναλυτικά, θα έχουμε τους εξής δυο εναλλακτικούς πίνακες για κάθε ένα από τους δυο αλγορίθμους κωδικοποίησης :

1) Για τον μεν *Agrawal* αλγόριθμο θα έχουμε τις εξής τροποποιήσεις στον πίνακα της γενικής μορφής

SubClass_Agrawal(index : int4, post : int4, father_post :int4)

Στον πίνακα αυτό τα πεδία *index* και *post* αποτελούν τα δύο άκρα που ορίζουν το πεδίο – κωδικό που προκύπτει από την κωδικοποίηση αυτή που προτείνει ο *Agrawal*, με απόλυτα διακριτό αναγνωριστικό αριθμό το πεδίο *post*. Το πεδίο *father_post* αντιστοιχεί στην τιμή του πεδίου *post*, που με την σειρά του μοναδικά προσδιορίζει την άμεση, πατρική του κλάση (στην περίπτωση των δέντρων που εξετάζουμε είναι κι ο μοναδικός πατρικός).

2) Για τον δε *Unicode Dewey* αλγόριθμο θα έχουμε την εξής κάτωθι τροποποίηση του πίνακα της προηγηθείσας γενικής μορφής :

SubClass_Unicode_Dewey(label : varchar(15), father_label : varchar(15))

Αντίστοιχα, στον πίνακα αυτό το πεδίο *label* από μόνο του αποτελεί τον απόλυτα διακριτό, αναγνωριστικό κωδικό της κάθε κλάσης που προκύπτει με τον αλγόριθμο κωδικοποίησης του *Unicode Dewey*, ενώ το *father_label* αντιστοιχεί στην τιμή του πεδίου *label*, που με την σειρά του μοναδικά προσδιορίζει την άμεση πατρική του κλάση.

Στο σημείο αυτό θα πρέπει να κάνουμε μια σειρά από παρατηρήσεις οι οποίες σχετίζονται με την σημασιολογία που απορρέει από την επιμέρους τροποποίηση του αρχικού σχήματος αναπαράστασης που υιοθετήσαμε και για τους δυο συγκεκριμένους, αυτούς πίνακες. Σε κάθε περίπτωση το μέγεθος του τύπου του αλφαριθμητικού του πεδίου *label* στον πίνακα της κωδικοποίησης του *Unicode Dewey* καθορίζεται άμεσα από το μέγιστο βάθος της ιεραρχίας των κλάσεων του *ODP* σχήματος (βλ. Πίνακα 1) συν ένα, δεδομένης της ιδεατής τοποθέτηση της κλάσης *Resource* πάνω από τους ριζικούς κόμβους της ιεραρχίας του κάθε επιμέρους σχήματος, δηλαδή υπό την μορφή μιας ιεραρχίας από την συνένωση των ιεραρχιών ολόκληρου του *ODP* καταλόγου. Από την άλλη πλευρά, οι τύποι των πεδίων *index* και *post*, στα οποία βασίζεται ο αλγόριθμος του *Agrawal*, σχετίζονται άμεσα από το συνολικό πλήθος των κλάσεων του *ODP* καταλόγου.

Και στους δυο πίνακες το πεδίο *father*, με την μορφή είτε του *father_post για* τον *Agrawal*, είτε του *father_label* για τον *Unicode Dewey* αλγόριθμο, είναι απαραίτητο για την ανάκτηση σε *RDF/S* του *ODP* σχήματος, καθώς επίσης και για να υποστηριχτούν οι επιθυμητές επερωτήσεις στους "πατρικούς" /"παιδικούς" /"αδελφικούς" κόμβους, που είναι αποθηκευμένοι στη βάση. Η επιλογή της προσθήκης του πεδίου αυτού εξασφαλίζει βέλτιστες αποδόσεις στις προηγούμενες ερωτήσεις. Συγκρινόμενη με την χρήση της συνάρτησης *prefix* για την παραγωγή των προγόνων μιας κλάσης με βάση τους κωδικούς που προκύπτουν με τη κωδικοποίηση του *Unicode Dewey*, καθώς επίσης και με την προσθήκη επιπρόσθετης πληροφορίας στους κωδικούς των κόμβων και στις δυο κωδικοποιήσεις που είναι του βάθους στον *Agrawal*.

4.3 Δείκτες (Indices)

Η χρήση δεικτών (*indices*) είναι ένας από τους πλέον ενδεδειγμένους τρόπους για να ενισχύσουμε την απόδοση της λειτουργίας της βάσης δεδομένων. Ένας δείκτης επιτρέπει τον εξυπηρετητή (*server*) της βάσης να βρίσκει και να ανακτά συγκεκριμένες πλειάδες αισθητά πιο γρήγορα από ότι θα μπορούσε χωρίς την χρήση του δείκτη. Ωστόσο η χρήση δεικτών επίσης επιφέρει σημαντική επιβάρυνση στην απόδοση της λειτουργίας του συστήματος της βάσης, και θα πρέπει γι αυτό να χρησιμοποιούνται με λογική. Κι αυτό γιατί εν γένει εάν η όλη πληροφορία είναι στην κυρίως μνήμη οι όλοι οι υπολογισμοί, που απαιτούνται για την εκτέλεση επερωτήσεων, είναι αρκετά γρήγοροι, ενώ όταν είναι πολύ μεγαλύτερη κι αναγκάζεται να αποθηκευτεί και σε επιπλέον δευτερεύουσα μνήμη, το κόστος ανεβαίνει καθώς οι I/O λειτουργίες κυριαρχούν αισθητά στο επιπρόσθετο αυτό κόστος.

Συνήθως, το σύστημα χωρίς την χρήση δεικτών ψάχνει σειριακά - γραμμή προς γραμμή - για να βρει όλες τις εγγραφές αυτές που ταιριάζουν. Όμως η μέθοδος αυτή αναζήτησης καθίσταται ιδιαίτερα χρονοβόρα κι αναποτελεσματική εάν λάβεις υπόψη σου την πολύ συνήθη περίπτωση, όπου η επερώτησή σου ψάχνει για ένα περιορισμένο πλήθος πλειάδων σε σχέση με το όλο πλήθος των αποθηκευμένων πλειάδων. Αντιθέτως, η δημιουργία κι η χρήση των δεικτών μπορεί να προσφέρει τελικά ένα πιο ικανοποιητικό τρόπο εντοπισμού των επιζητούμενων αποθηκευμένων πλειάδων στη βάση.

Επιπρόσθετα, η χρήση των δεικτών εν γένει μπορεί κι ευνοεί αισθητά πιθανές εκτελέσεις των λειτουργιών *Updates* και *Deletes* μαζί με τις διενεργούμενες συνθήκες αναζήτησης. Μάλιστα η συμβολή τους είναι ιδιαίτερα καθοριστική στην περίπτωση διαχειριστικών εντολών, όπου χρειάζεται το πολύ η ύπαρξη ενός δείκτη, καθώς και στις περιπτώσεις συνδυαστικών μεθόδων (*join methods*) καθώς τότε η επερώτηση εμπεριέχει πεδίο στο οποίο έχει οριστεί ένας δείκτης τότε μπορεί αξιοσημείωτα να επιταχυνθεί. Όταν ένας δείκτης έχει δημιουργηθεί, θα πρέπει να έχει προηγηθεί ο συγχρονισμός του με τον πίνακα. Η ενέργεια αυτή σαφώς και προσθέτει επιβάρυνση στις πράξεις διαχείρισης δεδομένων. Άρα, η χρήση των δεικτών θα πρέπει να γίνεται με φειδώ και να αποφεύγεται όταν κρίνεται μη αναγκαία η χρήση τους αντιστοίχως.

Εν γένει, η χρήση δείκτη ενός δείκτη σε ένα συγκεκριμένο πεδίο-γνώρισμα ενδείκνυται όταν θέλουμε μια αναπαράσταση που δεν θα εξαρτάται από τα εκάστοτε δεδομένα, αλλά θα μπορεί να διατηρεί την εξισορρόπηση των υπαρχόντων μονοπατιών με σχετικά μικρό κόστος. Βέβαια θα πρέπει να παρατηρήσουμε πως ο αναγκαίος αυτός σχεδιασμός είναι σχετικά πολύπλοκος. Το παρεχόμενο κόστος αναζήτησης, μεγέθους *log(N)* είναι πολύ καλύτερο από τον όποιο ένα *hash* πίνακα.

Αναφορικά, και στους 2 επιμέρους πίνακες που δημιουργήσαμε για την σχεσιακή αναπαράσταση της κωδικοποίησης του αλγορίθμου του *Agrawal et al* όσο και του *Unicode Dewey,* να αναφέρουμε ότι δημιουργούμε επιμέρους ευρετήρια (*indexes*) στα «πρωτεύοντα» πεδία τους *post* και *label* αντίστοιχα. Στη περίπτωση μάλιστα του αλγορίθμου του *Agrawal* ο σχηματιζόμενος δείκτης λειτουργεί πάνω σε *επιλογές πεδίων* (*range selections*) για ένα πεδίο π.χ. στο *post* στις εγγραφές του υπό εξέταση πίνακα *SubClass*.

Μιλώντας συγκεκριμένα για την περίπτωση του χρησιμοποιούμενου συστήματος της *PostgreSQL*, μπορούν να χρησιμοποιηθούν διάφορα είδη-κατηγορίες δεικτών, όπως είναι το *B-tree* (υλοποίηση του *Lehman - Yao* υψηλής συνταύτισης *B-trees*), το *R-tree* (*Gutman's quadratic split algorithm*) και το *Hash* (*Litwin's linear hashing*) [IPD]. Η χρήση του κάθε είδους δείκτη έγκειται στο ότι προσαρμόζεται σε ένα συγκεκριμένο είδος επερώτησης εξαρτώμενο από τον αλγόριθμο που χρησιμοποιεί.

Στην προκειμένη περίπτωση, προσαρμοζόμενοι στο σύστημα της *PostgreSQL*, χρησιμοποιούμε τον προεπιλεγμένο τύπο του B^+*-tree*, ο οποίος άλλωστε ταιριάζει και στις περισσότερες των περιπτώσεων των επιζητούμενων επερωτήσεων. Συγκεκριμένα, ο βελτιστοποιητής των επερωτήσεων (*query optimizer*) της *PostgreSQL* οδηγείται στην χρήση ενός δείκτη B^+*-tree*, οιαδήποτε φορά υπάρχει ένα δεικνυοδοτούμενο πεδίο του πίνακα, το οποίο εμπλέκεται σε μια σύγκριση η οποία περιλαμβάνει έναν από τους ακόλουθους τελεστές : <, <=, =, >=, >.

4.3.1 Δείκτης τύπου B^+-tree

Ο δείκτης B^+*-tree* αποτελεί ένα πολυεπίπεδο δείκτη, στον οποίο κάθε κόμβος περιέχει μέχρι *n-1* τιμές κλειδιών αναζήτησης, έστω *K1, K2,…, Kn-1* , και ομοίως *n* δείκτες *P1 , P2, .., Pn*. Εύλογα, όπως παρατηρούμε και στο παρακάτω σχήμα 28, όλες

οι τιμές των κλειδιών αναζήτησης του κάθε κόμβου, κρατούνται σε μια απόλυτη ταξινομημένη σειρά μεταξύ τους.

P_1	K_1	P_2	....	P_{n-1}	K_{n-1}	P_n

Σχήμα 28: Διάταξη τυπικού κόμβου του B+-tree

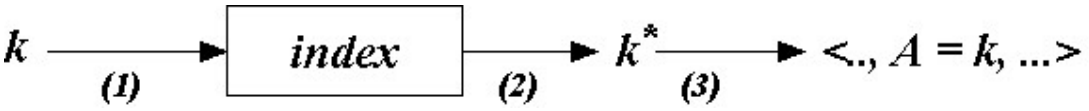
Για τον κάθε κόμβο "φύλλο", κάθε P_i (όπου $i = 1,..,n-1$) δείχνει εναλλακτικά είτε σε μία εγγραφή αρχείου με ένα κλειδί αναζήτησης με τιμή K_i , είτε εναλλακτικά σε ένα «κάδο» από επιμέρους δείκτες σε εγγραφές (*pointers*) με την συγκεκριμένη αυτή τιμή στο κλειδί αναζήτησης. Η δομή αυτού του κάδου χρησιμοποιείται στην περίπτωση όπου το κλειδί αναζήτησης δεν είναι το πρώτιστο κλειδί (*primary*) και το αρχείο δεν είναι ταξινομημένο με την ταξινόμηση του κλειδιού αναζήτησης. Ο δείκτης *Pn,* ο οποίος είναι δείκτης σε έναν κόμβο - "φύλλο", χρησιμοποιείται για συνδέσει τους κόμβους "φύλλα" σε γραμμική σειρά (σειρά ταξινόμησης ως προς το κλειδί αναζήτησης). Αυτό επιτρέπει ικανοποιητική, ακολουθιακή επεξεργασία του αρχείου. Σε καμία περίπτωση όμως το πεδίο τιμών σε κάθε φύλλο δεν αλληλο-επικαλύπτεται με των υπολοίπων.

Οι κόμβοι- "μη φύλλα" διαμορφώνουν έναν πολύ-επίπεδο δείκτη στους κόμβους "φύλλα". Ένας "μη-φύλλο" κόμβος μπορεί να κρατήσει μέχρι το πολύ και n δείκτες και το λιγότερο από $\lceil n/2 \rceil$ δείκτες. Το πλήθος των δεικτών ενός κόμβου ονομάζεται *fan-out*. Δεδομένου ενός κόμβου με m δείκτες, ο δείκτης *Pi* (όπου i = *2,..,n-1*) δείχνει στο υπό-δέντρο αυτό που περιέχει τις τιμές αυτές των κλειδιών αναζήτησης $\geq Ki-1$ και $< Ki$. Ο δείκτης P_m δείχνει στο υπό-δέντρο που περιέχει τις τιμές των κλειδιών αναζήτησης $\geq Km-1$

Δεδομένου ότι οι εγγραφές είναι σχετικά μεγαλύτερες από τους δείκτες, ο μέγιστος αριθμός των εγγραφών που μπορούν να αποθηκευτούν σε έναν "φύλλο" κόμβο είναι μικρότερος από τον μέγιστο αριθμό των δεικτών σε έναν μη - "φύλλο" κόμβο. Σε κάθε περίπτωση όμως οι κόμβοι φύλλα χρειάζεται να είναι τουλάχιστον μέχρι το μισό πλήρης.

Η λειτουργία του B^+*-tree* δείκτη ειδικεύεται στο να ενισχύει την αναζήτηση πάνω σε ένα συγκεκριμένο γνώρισμα (ή συνδυασμό γνωρισμάτων αντίστοιχα) σε όλες τις εγγραφές. Αναλυτικά, ο κάθε δείκτης λειτουργεί ως εξής (βλ. Σχήμα 29) :

1) Αρχικά, εισάγουμε το κλειδί αναζήτησης (*search key*) έστω ***k***, βάση του οποίου θα ερωτηθεί ο δείκτης πάνω στο πεδίο αυτό για να βρει τη εγγραφή με τη σχετική διεύθυνση.
2) Ο δείκτης στην ερώτηση αυτή που του ασκείται ανταποκρίνεται με μία σχετιζόμενη καταχώρηση του δείκτη (*index entry*), έστω k^*. Η εν λόγω καταχώρηση περιέχει αρκετή πληροφορία ώστε να προσπελαστεί εν συνεχεία η επιζητούμενη εγγραφή.
3) Τέλος διαβάζεται η πραγματική εγγραφή χρησιμοποιώντας την σχετική πληροφορία που παρέχει η καταχώρηση k^*. Από την εγγραφή αυτή υπάρχει σαφώς μια συγκεκριμένη τιμή για το συγκεκριμένο αυτό γνώρισμα, όπου έχουμε ορίσει τον δείκτη.

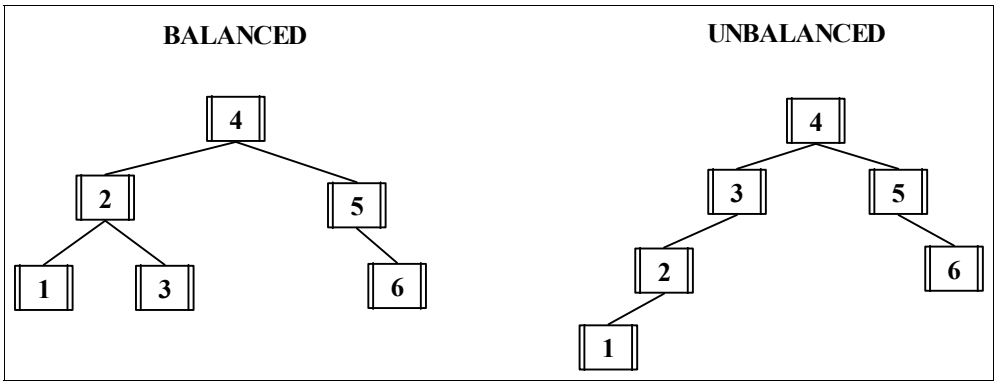


Σχήμα 29 : Λειτουργία ενός B+-tree δείκτη

4.3.2 Ταξινόμηση Κλειδιών

Βασική επιδίωξή μας είναι οι εγγραφές που εισάγονται στον πίνακα της αναπαράστασής μας, να είναι αριθμητικά ταξινομημένες πάνω στο πεδίο, στο οποίο είναι και ορισμένος ο δείκτης (εξ αυτού του λόγου χαρακτηρίζεται και ως ***clustered*** όπως φαίνεται και στο σχήμα 31). Αυτή η εξ αρχής μας επιλογή στοχεύει στην δημιουργία, στο βέλτιστο βαθμό, ισορροπημένων δέντρων. Κι αυτό λόγω της χαρακτηριστικής ιδιότητας των ισορροπημένων δέντρων, ότι το μονοπάτι του ριζικού κόμβου προς κάθε «φύλλο» - κόμβο είναι σταθερή και του ίδιου μεγέθους. Επιπρόσθετα, κάθε ισορροπημένο δέντρο έχει το πολύ $log_{order}(n)$ επίπεδα, όπου n το πλήθος των κόμβων - κλειδιών, κάτι που είναι ιδιαίτερα βέλτιστο για την περίπτωση των δεικτών και ιδίως για το B^+-*tree* όπου το *order* = 3. Με αυτόν τον τρόπο το κόστος αναζήτησης για τον κάθε κόμβο είναι κοινό και δεν διαφοροποιείται ανάλογα με την τοπολογική θέση του μέσα στο δέντρο και επίσης είναι σχετικά μικρό για τη δε περίπτωση του B^+-*tree*. Ένα χαρακτηριστικό παράδειγμα ακολουθεί στο σχήμα 30, όπου το κόστος αναζήτησης π.χ. του κόμβου με κλειδί 1 είναι στην μεν περίπτωση του ισορροπημένου δέντρου 2 προσβάσεις ξεκινώντας από τον ριζικό κόμβο, όπως και για κάθε άλλο κόμβο, ενώ στην περίπτωση του μη-ισορροπημένου είναι 3 προσβάσεις. Με αυτόν τον τρόπο βελτιστοποιούμε τα δημιουργούμενα B^+-*tree* και

τον χρόνο πρόσβασης σε όλους τους κόμβους τους. Άρα στην περίπτωση του οριζόμενου αυτού δείκτη πάνω σε ένα πεδίο-γνώρισμα βελτιστοποιείται επιπρόσθετα η απόδοση χρήσης του ως προς τον χρόνο εκτέλεσης επερωτήσεων που εμπεριέχουν συγκρίσεις και συσχετισμούς σε σχέση με το εν λόγω πεδίο-γνώρισμα.



Σχήμα 30 : Παράδειγμα α) ισορροπημένου δέντρου β) μη-ισορροπημένου δέντρου

Τα συνήθη πεδία τιμών, όπως είναι ως γνωστόν οι ακέραιοι αριθμοί, οι πραγματικοί και οι αλφαριθμητικοί χαρακτήρες έχουν εξίσου τυπική ταξινόμησή τους. Για τους μεν ακεραίους και πραγματικούς αριθμούς υπάρχει η αριθμητική ταξινόμηση των τιμών τους. Για τους δε αλφαριθμητικούς χαρακτήρες υπάρχει η λεξικογραφική ή όπως κι επίσης χαρακτηρίζεται ως ταξινόμηση λεξικού. Εάν θεωρήσουμε ότι έχουμε δυο αλφαριθμητικά $X_1\ X_2\ ...X_k$ και $Y_1\ Y_2\ ...Y_k$, όπου X_i, Y_i είναι αντιστοίχως οι χαρακτήρες αναπαράστασής τους, τότε μεταξύ τους ισχύει $X_1\ X_2\ ...X_k < Y_1\ Y_2\ ...Y_k$ αν και μόνο αν :

1. :$k < m$ και $X_1\ X_2\ ...X_k = Y_1\ Y_2\ ...Y_k$ ή
2. Για κάποιο $i \leq min(k\ ,\ m)$, έχουμε ότι $X_1 = Y_1$, $X_2 = Y_2$, ..., $X_{i-1} = Y_{i-1}$, και
   Ο δυαδικός κωδικός του χαρακτήρα X_i είναι αριθμητικά μικρότερος από ότι τον δυαδικό κωδικό του χαρακτήρα Y_i.

Στην κωδικοποίηση ASCII όπως και σε κάθε άλλη ευρέως χρησιμοποιούμενη κωδικοποίηση χαρακτήρων , η ταξινόμηση των γραμμάτων της ίδιας περίπτωσης είναι η αλφαριθμητική, και η ταξινόμηση των κωδικών των ψηφίων είναι η αριθμητική κωδικοποίηση των ψηφίων. Για παράδειγμα, ισχύει *'an' < 'and'* με βάση τον προαναφερθέν κανόνα (1) και *'banana' < 'bandana'* με βάση τον κανόνα (2).

4.3.3 Πρόσβαση Ταξινομημένων / Μη –ταξινομημένων Αρχείων από Πλειάδες

Με την διατήρηση ενός αρχείου από πλειάδες ταξινομημένο ως προς τις τιμές των κλειδιών τους κατά αύξουσα ή φθίνουσα σειρά, μπορούμε να αξιοποιήσουμε την γνωστή σειρά της ταξινόμησης για να βρούμε άμεσα μια συγκεκριμένη πλειάδα με βάση την τιμή του κλειδιού της. Προφανώς, κι αυτή στρατηγική είναι εν γένει καλύτερη από την σειριακή διερεύνηση της κάθε εγγραφής σε κάθε επιμέρους σελίδα που περιέχουν αποθηκευμένες τις εν λόγω εγγραφές. Συγκεκριμένα, εκτός από μία σελίδα, την οποία θα πρέπει να διερευνήσουμε πλήρως, χρειάζεται μόνο να κοιτάξουμε σε μία εγγραφή σε κάθε σελίδα.

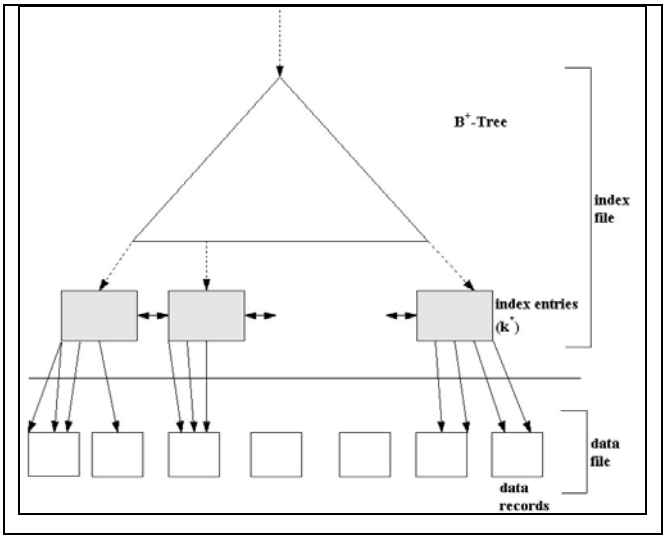
Αναλυτικά, το απαιτούμενο κόστος αναζήτησης μιας πλειάδας στην περίπτωση του ταξινομημένου δείκτη *clustered index*) ισούται με :

$$Total_Cost = depth\ of\ tree + 2 * cost_of_scan \quad (1)$$

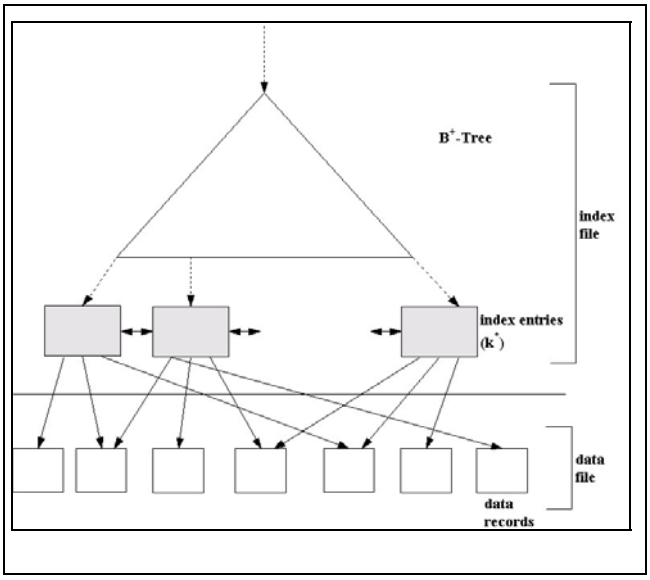
Γενικά, το κόστος για μια *επιλογή πεδίου* αυξάνεται κατακόρυφα εάν ο δείκτης ορίζεται πάνω σε ένα πεδίο, το οποίο δεν είναι ταξινομημένο (*unclustered* – βλ. σχήμα 32). Στην περίπτωση αυτή, η εγγύτητα των καταχωρήσεων του δείκτη δεν συνεπάγεται παρομοίως και εγγύτητα σε όλο το πλήθος των εγγραφών του πίνακα. Σε έναν πίνακα μπορούν να οριστούν το πολύ ένας *clustered* δείκτη και ένας οποιοσδήποτε αριθμός από μη-ταξινομημένους δείκτες.

Κατά αντιστοιχία το κόστος στην περίπτωση του ταξινομημένου δείκτη (*clustered index*) ισούται με :

$$Total_Cost = depth\ of\ tree + cost_of_scan + cost_of_fetching_outputting_selected_rows \quad (2)$$



Σχήμα 31 : Clustered Index



Σχήμα 32 : Unclustered Index

4.4 Χώρος Και Χρόνος Αποθήκευσης

Στην συνέχεια παραθέτουμε το σχήμα 33, όπου απεικονίζεται το μέγεθος των πινάκων, που απορρέουν από τα σχήματα αναπαράστασης *SubClass_Agrawal* και *SubClass_Unicode_Dewey* αντίστοιχα, καθώς και το μέγεθος του εκάστοτε κατασκευαζόμενου δείκτη που έχουμε ορίσει στα πεδία *post* στον μεν *SubClass_Agrawal* και του *label* στο δε *SubClass_Unicode_Dewey*. Επιπλέον, στο σχήμα 34 απεικονίζεται ο αναγκαίος χρόνος φόρτωσης των 16 ιεραρχιών των

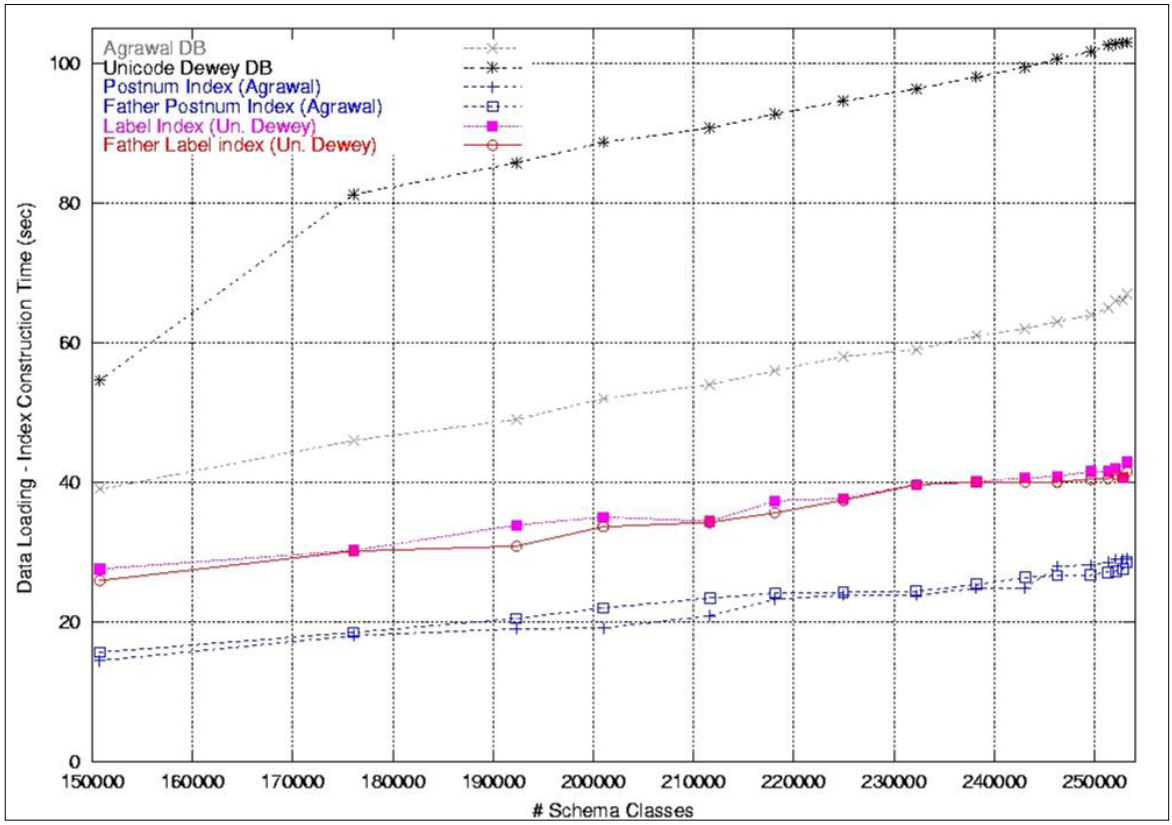
σχημάτων του *ODP* καταλόγου κατά φθίνουσα σειρά του μεγέθους τους (μετρήσιμες ως προς τον αριθμό κλάσεών τους).

Συγκεκριμένα, το μέγεθος του *SubClass_Unicode_Dewey* είναι 16376 Kbytes και το μέγεθος του *SubClass_Agrawal* είναι 12902 Kbytes, όταν και κι από κοινού περιέχουν 253215 πλειάδες (που ισοδυναμούν με το πλήθος των κλάσεων) σε 1613 και 2073 σελίδες δίσκου (*disk pages*) αντίστοιχα. Με βάση όλα αυτά τα πειράματα μπορούμε αβίαστα να συμπεράνουμε πως για να αποθηκεύσουμε την πλήρη περιγραφή μιας κλάσης χρειαζόμαστε συγκεκριμένα 52,17 bytes στον *Agrawal* και 66,22 bytes στον *Unicode Dewey*. Με βάση το σχήμα αναπαράστασης του *Agrawal et al, PInterval*, χρειάζονται 12 bytes και ομοίως για το σχήμα *UPrefix* του *Unicode Dewey* χρειαζόμαστε 30 *bytes*. Το παρατηρούμενο επιπλέον κόστος ανά πλειάδα, που εμφανίζεται και στους δυο προαναφερθέντες σχεσιακούς πίνακες αναπαράστασης (40 *bytes*), οφείλεται στο *αναγνωριστικό της πλειάδας* (*tuple id*), το οποίο με την σειρά του αποτελείται από την επιμέρους σύνθεση των στοιχείων (*block number, tuple index within block*), τα οποία παράγονται από το σύστημα της *PostgreSQL*, ώστε να αναγνωρίζεται με απόλυτα μοναδικό τρόπο η φυσική θέση κάθε πλειάδας μέσα στον πίνακα που ανήκει.

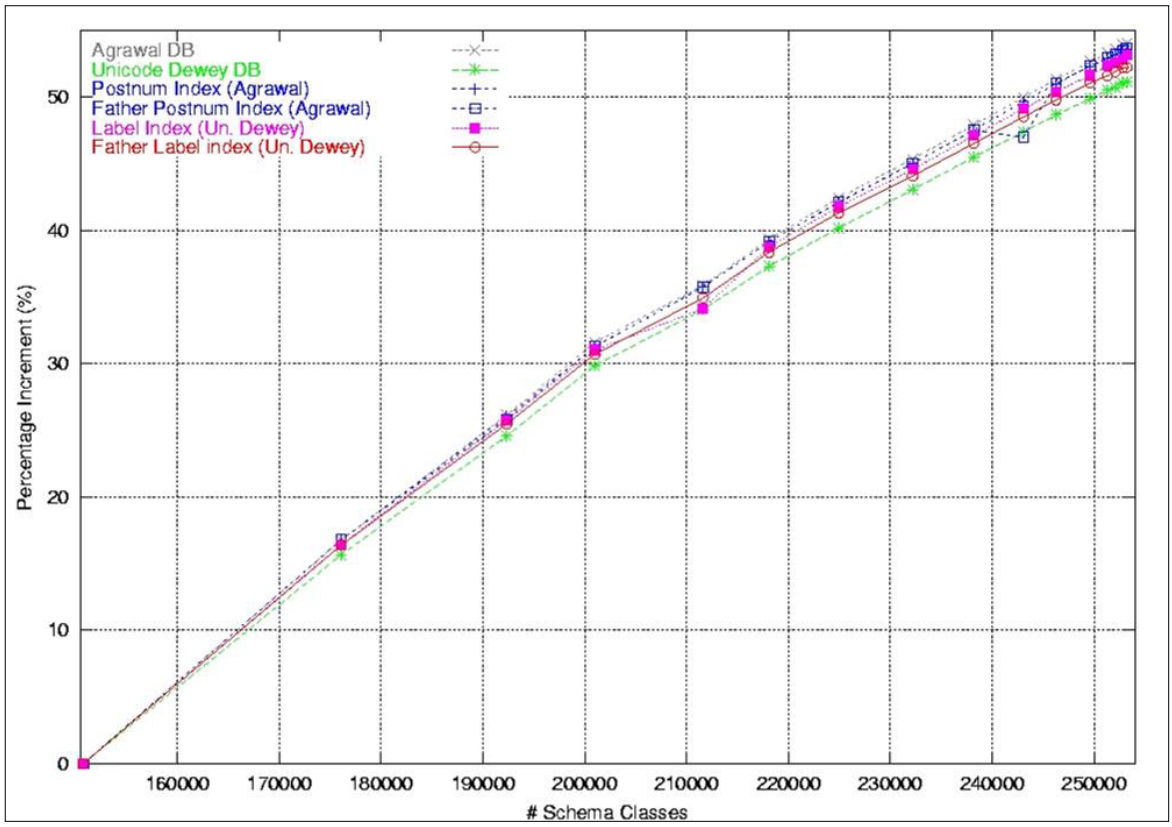
Επιπρόσθετα, η *PostgreSQL* προβάλλει σαν αποθηκευτική απαίτηση για τους αλφαριθμητικούς τύπους τα 4 *bytes* επιπρόσθετα του πραγματικού μεγέθους του αλφαριθμητικού. Για αυτούς τους λόγους είναι που χρειαζόμαστε για την περίπτωση αποθήκευσης του γνωρίσματος *label* του πίνακα *SubClass_Unicode_Dewey* τον μέσο όρο bytes που είναι 13.11 bytes, προσεγγίζει το μέγιστο μέγεθος του τύπου, που είναι 15 *bytes* αντίστοιχα. Στο σημείο αυτό θα μπορούσε επίσης να δοθεί ιδιαίτερη έμφαση στο ότι μόνο το 0.133% επί του συνολικού πλήθους των κωδικοποιημένων κλάσεων στην σχεσιακή αναπαράσταση του *SubClass_Unicode_Dewey* χρειάζονται κωδικούς με Unicode χαρακτήρες, οι οποίοι να ξεπερνούν τα 2 bytes (συγκεκριμένα μόλις 2 κλάσεις έχουν πλήθος εισερχόμενων ακμών – *fan-in* μεγαλύτερο από 256, καθώς και οι 336 άμεσα σχετιζόμενες υποκλάσεις τους.

Ο πίνακας *SubClass_Agrawal* είναι 21.2% μεγαλύτερος σε μέγεθος αποθήκευσης από ότι ο *SubClass_Unicode_Dewey*, ενώ όσο το μέγεθος του σχηματιζόμενου δείκτη στο γνώρισμα *label* του *SubClass_Unicode_Dewey* (1001 σελίδες δίσκου) είναι 29.8% μεγαλύτερο από ότι το μέγεθος του αντίστοιχου σημαίνοντος γνωρίσματος *post* στον πίνακα *Subclass_Agrawal* (697 σελίδες δίσκου). Από την άλλη πλευρά, ο χρόνος φόρτωσης στην βάση των δεδομένων των πινάκων

και από κοινού ο χρόνος κατασκευής των δεικτών στον *SubClass_Unicode_Dewey* είναι 34.75% και 42.21% μεγαλύτερος από τους αντίστοιχους χρόνους του *Subclass_Agrawal*. Επίσης έχουν παρατηρηθεί ελαφρώς πιο μικρές τιμές αποθηκευτικού χώρου και χρόνου κατασκευής για τους δείκτες στο γνώρισμα *father_post* και στο γνώρισμα *father_label* των δυο πινάκων σε σχέση με τα προαναφερθέντα πρωτεύουσας σημασίας γνωρίσματά τους *post* και *label* αντίστοιχα. Κι αυτό οφείλεται στα ότι τα πατρικά αυτά γνωρίσματα, σε κάθε περίπτωση, απευθύνονται πολύ σε μικρότερα διαστήματα διακριτών τιμών (για *post* και *label*), καθώς υπάρχουν πολλές φορές κοινοί πατρικοί κόμβοι. Μπορούμε λοιπόν να διαπιστώσουμε το σημαντικό επιπλέον κόστος που παρατηρείται τόσο στην αποθήκευση αλλά πολύ περισσότερο στην χρήση δεικτών σε αλφαριθμητικά πεδία-γνωρίσματα σε ένα σύστημα διαχείρισης βάσεων δεδομένων, όπως είναι η *PostgreSQL*.



Σχήμα 33 : Μέγεθος Βάσης Αποθήκευσης και Δεικτών Agrawal et al και Unicode Dewey



Σχήμα 34 : Χρόνος Κατασκευής των δέντρων των ιεραρχιών του ODP καταλόγου

4.5 SQL Ερωτήσεις

Στο σημείο αυτό θα προχωρήσουμε σε μα αναλυτική περιγραφή όλων των επιμέρους επερωτήσεων έτσι ώστε προκειμένου να διασφαλιστεί στο μέγιστο έγκυρο βαθμό, η επιδιωκόμενη λειτουργικότητα όπως αυτή καθορίζεται από μια σειρά υπηρεσιών - λειτουργιών που θα πρέπει να ικανοποιούν οι διάφορες διαδικτυακές πύλες, μέσω μιας σειράς από δεδομένες μηχανές *SQL* (χαρακτηριστικό παράδειγμα στην περίπτωση μας είναι η *PostgreSQL*). Μάλιστα θα μελετηθεί αυτή η επιδιωκόμενη λειτουργικότητα από κοινού μέσω τόσο του σχήματος αναπαράστασης με πρόθεμα (*prefix*), όσο και μέσω του σχήματος αναπαράστασης με τη χρήση διαστήματος (*interval*), το οποίο σε κάθε περίπτωση θα μπορεί να μεταφραστεί σε απλή *SQL* έκφραση.

4.5.1 Βασικές Επερωτήσεις

Ας θεωρήσουμε ότι έχουμε έναν κόμβο *u* ο οποίος με την μεν διαδικασία κωδικοποίησης του αλγορίθμου του *Agrawal et al* (σχήμα *PInterval*) αποκτά έστω την ετικέτα [*i, p*] και με την κωδικοποίηση του αλγορίθμου του *Dewey* (σχήμα *UPrefix*) έστω ότι αποκτά την αλφαριθμητική ετικέτα '*l*'. Αναφορικά, με τις παραγόμενες αυτές ετικέτες θα έχουμε τις εξής επιμέρους βασικές επερωτήσεις υλοποιημένες υπό την βέλτιστη *SQL* μορφή :

Α. Απόγονοι – (Descendants)

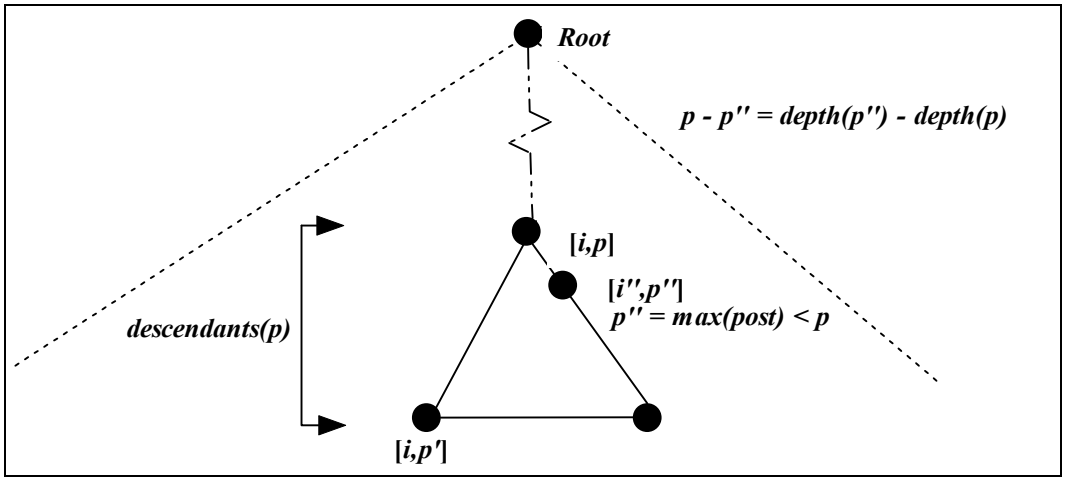
1. **Agrawal**

```
select u.post
from PInterval u
where i <= u.post and u.post < p
```

2. **Unicode Dewey**

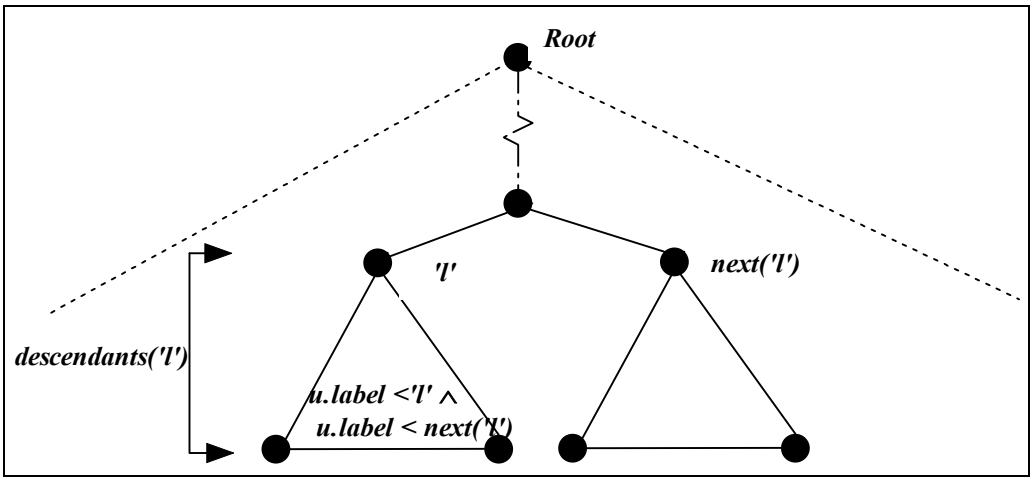
```
select u.label
from UPrefix u
where 'l' < u.label and u.label <next('l')
```

Η μεν περίπτωση των απογόνων του *Agrawal* απορρέει κατευθείαν από τον φορμαλισμό του πίνακα 6 του κλασσικού *Agrawal*. Στο σχήμα 35 ακολουθεί μια γραφική απεικόνιση της διασύνδεσης του πιο αριστερού φύλλου - κόμβου με πεδίο [*i, p'*] του ορισμένου υποδέντρου από τον κόμβο με πεδίο-κωδικό [*i, p*], δηλαδή έχει το μικρότερο *index* πεδίο, κι όπου ο πιο δεξιός –"φύλλο" κόμβος έχει το μέγιστο *post* πεδίο που είναι μικρότερο του post πεδίου *p* του κόμβου *u.*



Σχήμα 35 : Ερώτηση Απογόνων στο PInterval

Αντιθέτως, στο σχήμα του *UDewey* αναζητούμε τους απογόνους ενός κόμβου με κωδικό το αλφαριθμητικό *'l'*, οι οποίοι είναι μεγαλύτεροι αλφαριθμητικά από αυτόν και μικρότεροι από τον κωδικό του επόμενου αλφαριθμητικά αδελφού του κόμβου, αποφεύγοντας έτσι τους μεγαλύτερους κόμβους των υποδέντρων των αδελφών κόμβων (βλ. Σχήμα 36).



Σχήμα 36 : Ερώτηση Απογόνων στο UDewey

Β. Πρόγονοι – (Ancestors)

1. Agrawal Encoding

select u.post
from PInterval u
where u.index <= *p* and *p* <= u.post

2. Unicode Dewey Encoding

select *prefix('l')*

Και στη ερώτηση αυτή ο μεν *Agrawal* και πάλι απορρέει κατευθείαν από σχετικό τον φορμαλισμό του πίνακα 6, ενώ στον *UDewey* αναζητούμε τους προγόνους ενός κόμβου με κωδικό του γνωρίσματος *label* το αλφαριθμητικό *'l'*, οι οποίοι είναι προφανώς προθέματα της ετικέτας του. Για τον λόγο αυτό δημιουργήσαμε μια *SQL* συν *prefix()* που επιστρέφει το σύνολο των προθεμάτων και συνεπώς των αντιστοιχιζόμενων σε αυτά προγόνων του εκάστοτε εξεταζόμενου κόμβου.

C. Φύλλα – (Leaves)

1. Agrawal Encoding

select u.post
from PInterval u
where *i* <= u.post and u.post < *p* and u.post = u.index

2. Unicode Dewey Encoding

1η Μορφή

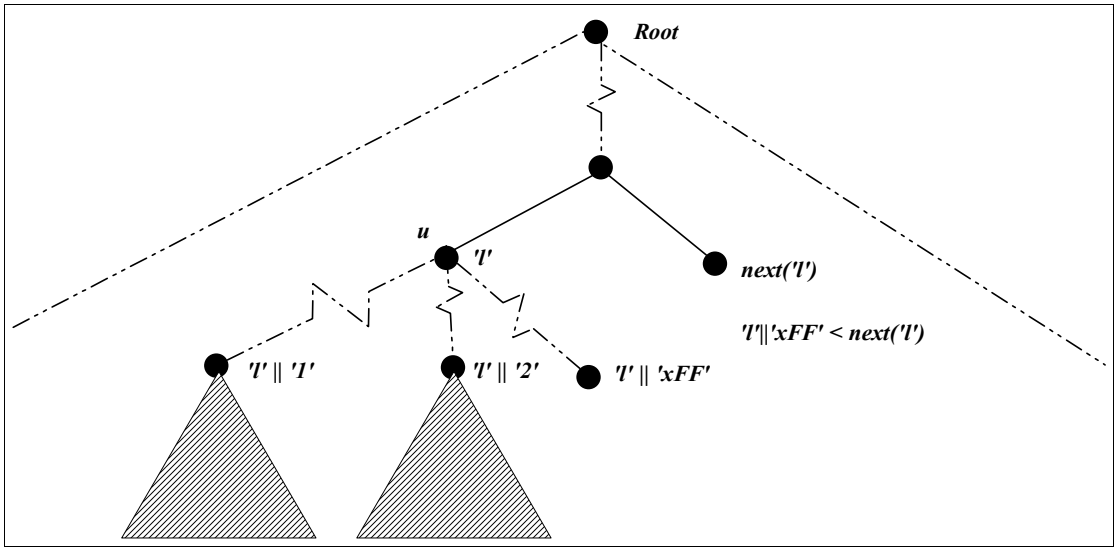
select *u.label*
from *UPrefix u*
where '*l*' <= *u.label* and *u.label* < *next('l')* and
NOT EXISTS (select *
from *UPrefix z*
where (*u.label* <= *z.label*) and (*z.label* < *next(u.label)*)

ή εναλλακτικά

2η Μορφή

```
select u.label
from UPrefix u
where 'l' <= u.label and u.label < next('l') and
NOT EXISTS ( select *
                from UPrefix z
                where (u.label <= z.label) and (z.label < u.label ||'xFF')
```

Η ερώτηση των φύλλων κόμβων για την περίπτωση του *Agrawal* αποτελεί επέκταση της προαναφερθείσας ερώτησης των απογόνων για την ίδια κωδικοποίηση με την προσθήκη της συνθήκης ισότητας *post* και *index* κωδικών, που χαρακτηρίζει τους φύλλα-κόμβους. Από την άλλη πλευρά ο *Unicode Dewey* είναι μια σύνθετη περίπτωση δυο φωλιασμένων ερωτήσεων. Συγκεκριμένα, η εσωτερική ερώτηση αποτελεί την προαναφερθείσα ερώτηση του *Unicode Dewey* αναζήτησης των απογόνων ενός κόμβου, με την χρήση της ενσωματωμένης συνάρτησης *next()*, η οποία χρησιμοποιείται από την φωλιασμένη υπο-ερώτηση πάνω σε όλα τα αλφαριθμητικά κωδικούς που επιστρέφονται από την εξωτερική επιμέρους ερώτηση. Αυτή είναι η πρώτιστη προσέγγιση (1η Μορφή) στην απεικόνιση της εν λόγω επερώτησης. Υπάρχει όμως κι η εναλλακτική πρόταση απεικόνισης της επερώτησης που είναι όπως εικονίζεται παραπάνω στην 2η Μορφή με το να παραλείπουμε από τις δυο επιμέρους υπο-ερωτήσεις την χρήση της συνάρτησης *next()* πάνω στο γνώρισμα *label* και να προτιμούμε άντ' αυτού την χρήση πράξεων με αλφαριθμητικά ως βέλτιστη από την άποψη πλήθους συγκρίσεων. Ο τελεστής '||' συνενώνει τον Unicode αλφαριθμητικό χαρακτήρα '*xFF'*, ο οποίος αποτελεί την αναπαράσταση ενός *byte* με όλο άσσους δηλαδή '*11111111'*, με το γνώρισμα *label*. Το αποτέλεσμα της συνένωσης του *label* με τον κωδικό '*xFF'* αποτελεί το ιδεατό πιο δεξιό παιδί του κόμβου με *label = 'l'* και του οποίου ο άμεσα επόμενος δεξιός κόμβος έχει κωδικό τον *next(label)*. Η συνένωση αυτών των δυο έχει σαν αποτέλεσμα το μέγιστο αλφαριθμητικό που μπορεί να παραχθεί και παραμένει μικρότερο της τιμής που επιστρέφει η κλήση της συνάρτησης με όρισμα το *label* , δηλαδή *next(label)* (βλ. Σχήμα 37).



Σχήμα 37 : Ερώτηση "Φύλλων"- Κόμβων στο Unicode Dewey (βέλτιστη μορφή)

Μάλιστα, η χρήση της συνένωσης αυτής με τον ειδικό αυτό χαρακτήρα μας επιτρέπει την ενεργοποίηση της λειτουργίας του δείκτη πάνω στο γνώρισμα *label* για την βέλτιστη αξιολόγηση της εσωτερικής φωλιασμένης ερώτησης. Η μετάφραση της εν λόγω ερώτησης επιχειρήθηκε και με πλήθος άλλων τρόπων (όπως είναι η χρήση του πατρικού κωδικού *father_label* στη φωλιασμένη ερώτηση) όμως η πρότερη λύση της συνένωσης με τον ειδικό χαρακτήρα '*xFF'* αξιολογήθηκε με την βέλτιστη απόδοση.

D. Precedings

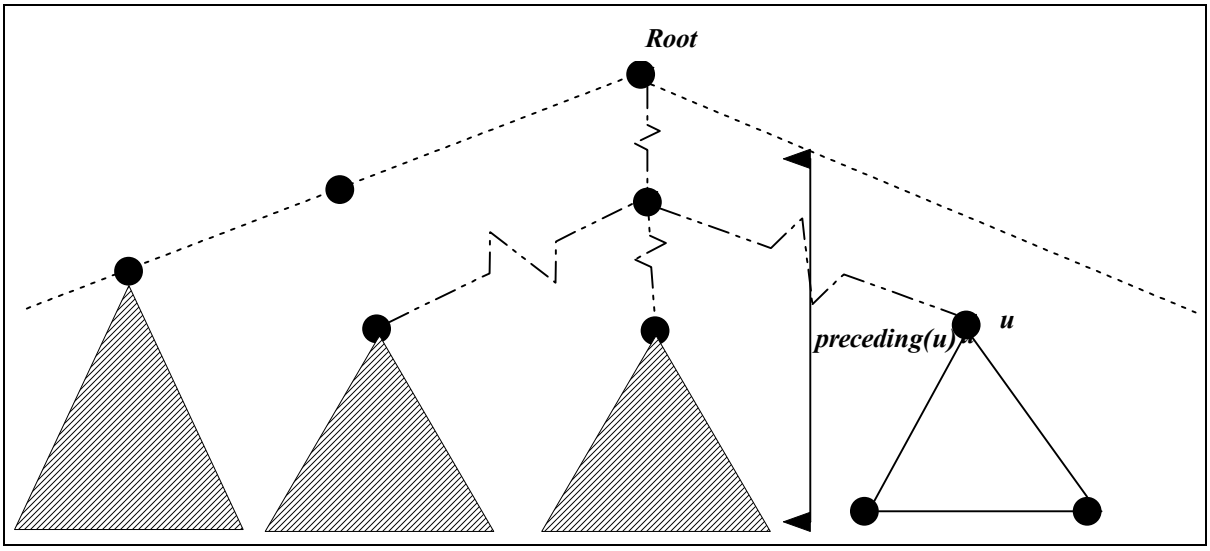
1. Agrawal Encoding

select u.post
from PInterval u
where u.post <= *p*

2. Unicode Dewey Encoding

select u.label
from UPrefix u
where u.label <= *previous*('*l*') and u.label > '*l*'

Οι καλούμενοι «πρότεροι» της κωδικοποίησης του *Agrawal* είναι οι κόμβοι που έχουν μικρότερο πεδίο *post*, από τον τρέχων κόμβο που χαρακτηρίζεται από το *post* πεδίο με τιμή *"p"*(βλ. Σχήμα 38). Η λογική αυτή επεκτείνεται και στην περίπτωση του *Unicode Dewey* όπου αναζητάμε όλους τους αλφαριθμητικούς αυτούς κωδικούς οι οποίοι είναι μικρότεροι ή και ίσοι του πατρικού κόμβου του κόμβου με κωδικό *"l"*, αλλά μεγαλύτεροι του ίδιου του *"l"* (βλ. Σχήμα 38).



Σχήμα 38 : Ερώτηση Precedings

E. Followings

1. Agrawal Encoding

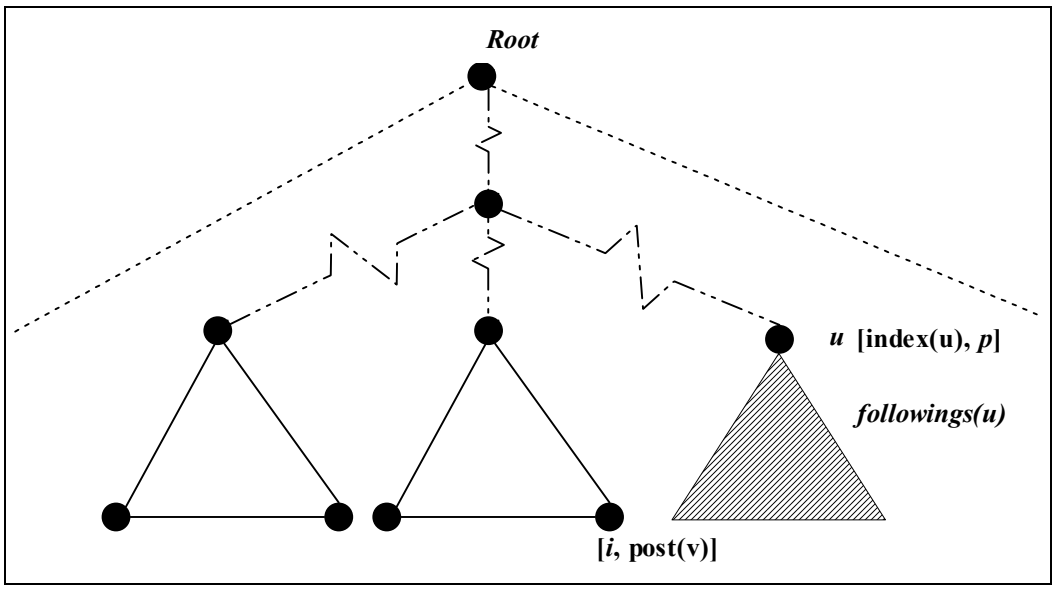
select u.post
from PInterval u
where u.post >= *p* and u.index > *i*

2. Unicode Dewey Encoding

select u.label
from UPrefix u
where u.label >= *next*(*'l'*) and u.label < *next_brother_of_father('l')*[*]

Σε αντίθεση με τους *πρότερους* (*precedings*) οι καλούμενοι «επόμενοι» της κωδικοποίησης του *Agrawal* είναι οι κόμβοι με μεγαλύτερο πεδίο *post,* που δεν είναι "*πρόγονοι*" του τρέχοντος κόμβου με το *post* πεδίο "*p"*. Στο σημείο αυτό θα πρέπει να αναφέρουμε πως η ερώτηση αυτή στηρίζεται αρχικά στην απλή συνθήκη *u.index > p*, η οποία μας δηλώνει ότι οι «ακόλουθοι» κόμβοι έχουν πάντα μεγαλύτερους *postorder* και *index* αριθμητικές τιμές. Συνεπώς, η ερώτηση αυτή στηρίζεται στις τιμές του πεδίου *index* , στο οποίο στην αρχική μας σχεσιακή αναπαράσταση δεν έχουμε δημιουργήσει δείκτη (έχουμε δημιουργήσει μόνο δείκτη στο πεδίο *post*). Εάν και θα μπορούσαμε να δημιουργούσαμε τον συμπληρωματικό αυτό δείκτη στο πεδίο *index* δεν το κάνουμε καθώς θέλουμε να διατηρήσουμε το μέγεθος του παραγόμενου αυτού σχήματος (*PInterval*) σε σύγκριση με το αντίστοιχο μέγεθος του σχήματος του *UPrefix*(το οποίο επίσης έχει ορισμένο έναν δείκτη). Για αυτό κι υιοθετούμε την παραπάνω τροποποίηση της μορφής της *SQL* ερώτησης, η οποία μπορεί να βελτιστοποιηθεί ως προς την απόδοσή της με την χρήση ενός B^+*-tree*, ορισμένου στο πεδίο *post*.

Τώρα στην περίπτωση του διατύπωσης της επερώτηση στο σχήμα του *Unicode Dewey* υπάρχει αντιστοιχία με τον πίνακα (3), όπου αναζητάμε όλους τους αλφαριθμητικούς αυτούς κωδικούς οι οποίοι είναι μεγαλύτεροι ή και ίσοι του επόμενου αδελφού του άμεσα πατρικού κόμβου του κόμβου με κωδικό *"l"*, αλλά μικρότεροι του ίδιου του *"l"* (βλ. Σχήμα 39).



Σχήμα 39 : Ερώτηση Followings

F. Ομοεπίπεδοι – (Siblings)

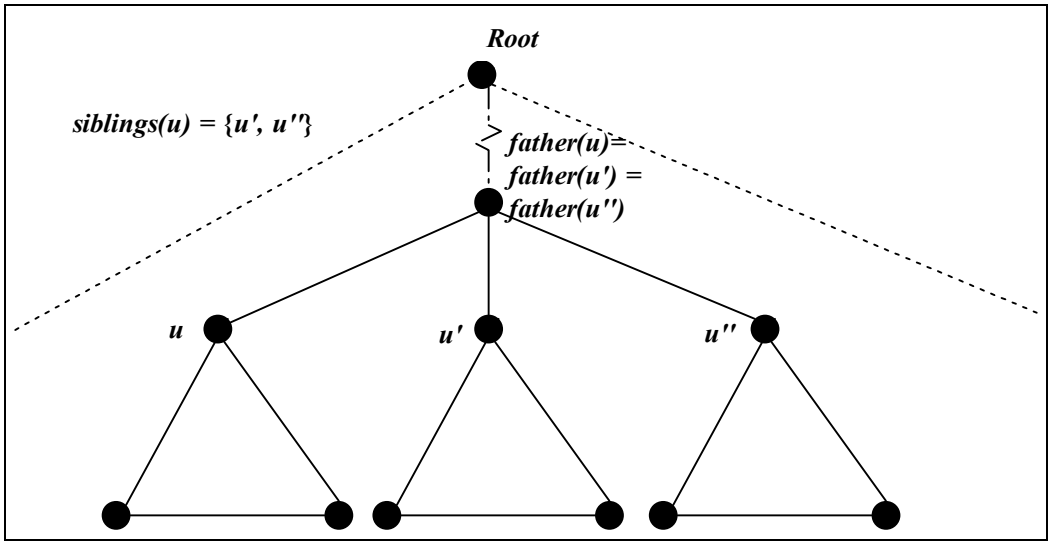
1. Agrawal Encoding

select *u.post*
from *PInterval u, PInterval v*
where *u.post* != *p* and u.*father_post = v.father_post* and *u.post= p*

2. Unicode Dewey Encoding

select *u.label*
from *UPrefix u, UPrefix v*
where (u.label != 'l') and (v.father_label = u.father_label) and (*v.label = 'l'*)

Η υλοποίηση σε *SQL* της επερώτησης για τους *siblings* έστω του κόμβου *l* και των δυο επιμέρους σχημάτων κωδικοποίησης είναι ακριβώς ίδια. Βρίσκουμε όλους τους κόμβους με κωδικό διαφορετικό του *'l'* οι οποίοι έχουν κοινό άμεσο πατέρα με τον *'l'* (βλ. Σχήμα 40).



Σχήμα 40 : Ερώτηση Siblings

G. Κοντινότερος κοινός Πρόγονος – (*nca)*

1. Agrawal Encoding

```
select u.post
from PInterval u
where u.index <= min{i, i'} and max{p, p'} < u.post and
NOT EXISTS (    select *
                from  PInterval w
                where w.index <= min{i, i'} and max{p, p'}< w.post and
                u.index <= w.post and  w.post < u.post)
```

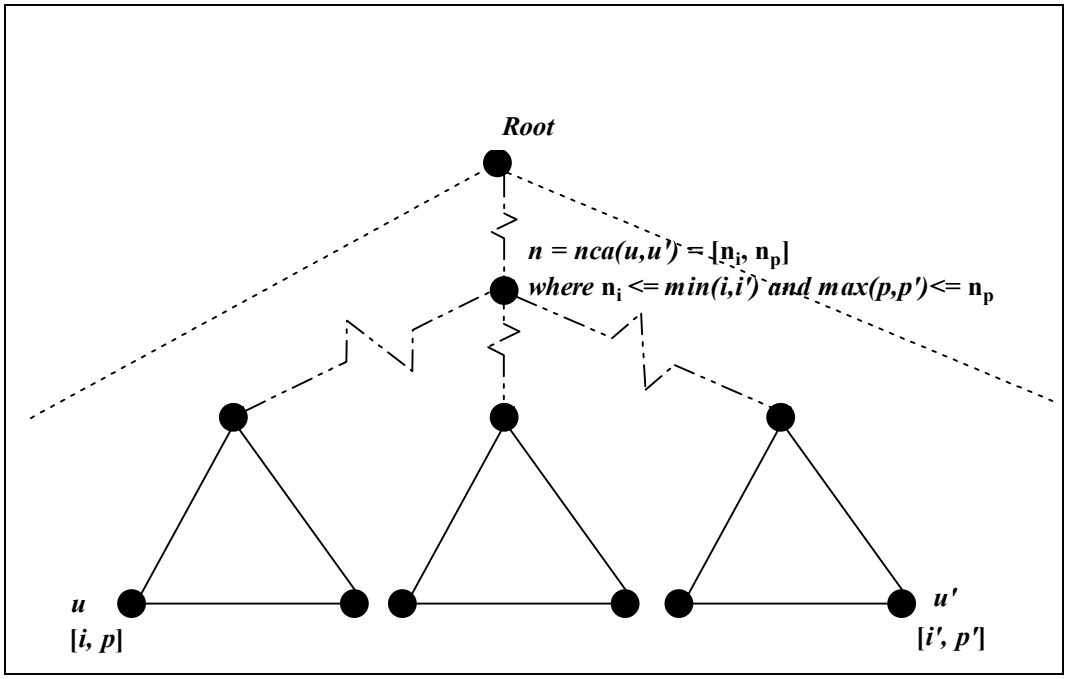
2. Unicode Dewey Encoding

select *nca('l1','l2')*

```
Αλγόριθμος nca ()
Παράμετροι :
(1)  Κωδικός l1
(2)  Κωδικός l2
Περιγραφή Αλγορίθμου :
∀ Χαρακτήρα Αναπαράστασης Κωδικού l1 → l1[Count]
if (l1[Count] = l2[Count])
{
     (1)  result[Count] = l1[Count] – Αποθήκευση Τρέχοντος Κοινού Προθέματος;
     (2)  Count = Count + 1
}
return result[Count]
```

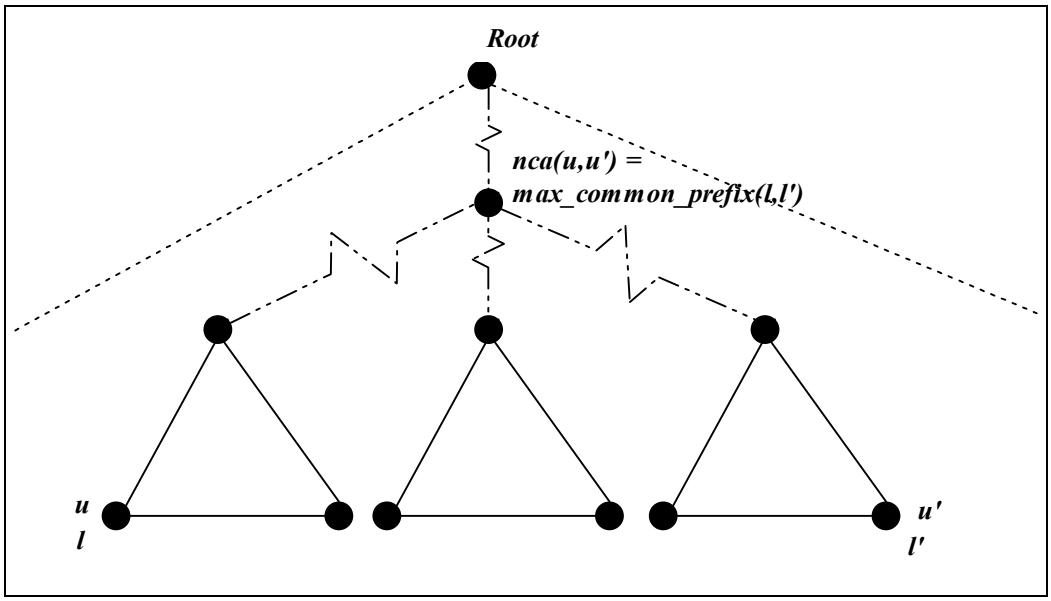
Σχήμα 41 : Αλγόριθμος Ενσωματωμένης Συνάρτησης nca στον Unicode Dewey

Η ερώτηση της αναζήτησης της ύπαρξης του μικρότερου –και ιεραρχικά πλησιέστερου κοινού προγόνου δυο επιμέρους κόμβων έστω [*i, p*] [*i', p'*]στο σχήμα του *PInterval* προχωρά στην αναζήτηση του κόμβου αυτού που το πεδίο του θα περιλαμβάνει την ελάχιστη τιμή από τα πεδία των *index* των δυο κόμβων καθώς και την μέγιστη αντιστοίχως των πεδίων *post* των δυο κόμβων κι δεν υπάρχει άλλος αντίστοιχος τέτοιος κόμβος, ο οποίος να ικανοποιεί τις παραπάνω συνθήκες και να έχει μικρότερο διάστημα από αυτού (βλ. Σχήμα 42).



Σχήμα 42 : Ερώτηση του ελάχιστου κοινού προγόνου δυο κόμβων στο PInterval

Αντιστοίχως, η ερώτηση στο σχήμα του *UPrefix* πραγματώνεται με την ενσωματωμένη συνάρτηση *nca* (βλ. Σχήμα 41), η οποία αναζητά κι επιστρέφει το μέγιστο κοινό – εάν υπάρχει βέβαια- αλφαριθμητικό πρόθεμα των δυο εξεταζόμενων κόμβων, το οποίο εάν υφίσταται αντιστοιχεί σε ένα υπάρχων κόμβο της ιεραρχίας (βλ. Σχήμα 43).



Σχήμα 43 : Ερώτηση του ελάχιστου κοινού προγόνου δυο κόμβων στο Uprefix

4.6 Πειραματική Αξιολόγηση Επερωτήσεων

Αρχικά, για την μέγιστη διασφάλιση της εγκυρότητας των πειραματικών αποτελεσμάτων, η επιλογή των εν λόγω χρησιμοποιούμενων επερωτήσεων ήταν η πλέον ενδεικτική, καθώς οι εκτελέσεις τους πραγματοποιήθηκαν ένα επαρκές, προδιαγεγραμμένο σαφώς πλήθος (10). Αρχικά, η μεν πρώτη εκτέλεση σε κάθε μία εκτέλεση της κάθε επερώτησης πραγματοποιήθηκαν με απώτερο και μοναδικό σκοπό την ενεργοποίηση, το καλούμενο «*ζέσταμα*» των καταχωρητών της βάσης, χωρίς βέβαια να ληφθούν και να καταγραφούν τα αποτελέσματά της. Εν συνεχεία, οι λοιπές 9 συνεχόμενες εκτελέσεις των πειραμάτων ήταν αυτές οι οποίες από κοινού μας δίνουν τον πλέον αντικειμενικό μέσο όρο του χρόνου εκτέλεσης έκαστης επερώτησης, που θα παρουσιάσουμε αναλυτικά στη συνέχεια. Ο υπολογισμός του χρόνου εκτέλεσης σε κάθε μια ερώτηση πραγματοποιείται με την χρήση της παρεχόμενης εκ του συστήματος λειτουργίας του *PostgreSQL Explain Facility*,η οποία εκτελεί μια συνολική χρονική εκτίμηση του κόστους εκτέλεσης όλων των επιμέρους βημάτων του όλου πλάνου που διαμορφώνει ο βελτιστοποιητής του συστήματος για την εκτέλεση της επερώτησης [IPD].

Παρατηρώντας το σύνολο των εκφράσεων αλλά και των μεταφράσεών τους σε SQL επερωτήσεις όπως προείδαμε, παρατηρούμε ότι το μεγαλύτερο τους μπορεί άμεσα να μεταφραστεί με το βέλτιστο τρόπο με την μορφή *SQL*, χρησιμοποιώντας το σχεσιακό σχήμα στο οποίο καταλήξαμε προηγουμένως. Τώρα από το σύνολο των επερωτήσεων αυτών που σχετίζονται με την αναπαράσταση του *Subclass_Agrawal* υπάρχει ένα υποσύνολο το οποίο χρησιμοποιεί συναρτήσεις καθορισμένες από τον χρήστη (*user-defined*), οι οποίες είναι εκ του προοιμίου αποθηκευμένες *SQL* διαδικασίες. Χαρακτηριστικά παραδείγματα των ερωτήσεων αυτών είναι πχ η επερώτηση των "*προγόνων*" (συγκεκριμένα με την συνάρτηση *prefix()*) και η επερώτηση του "άμεσου κοινού προγόνου" – *nca* (με τις συναρτήσεις *prefix()* και *character_length()* η οποία τελευταία είναι ενσωματωμένη συνάρτηση του συστήματος της *PostgreSQL* για την διαχείριση των αλφαριθμητικών και η οποία επιστρέφει το πλήθος των χαρακτήρων αναπαράστασης ενός αλφαριθμητικού). Επιπλέον άλλες επερωτήσεις, στις οποίες χρησιμοποιούμε τέτοιου είδους

ενσωματωμένες συναρτήσεις, είναι επίσης για την υλοποίηση της διερεύνησης της ύπαρξης σχέσεως υπαλληλίας (*subsumption*) μεταξύ δυο κλάσεων μιας ιεραρχίας, εξετάζοντας τους κωδικούς τους και στα δυο επιμέρους σχήματα κωδικοποίησης.

Για να είμαστε πιο αναλυτικοί κι ακριβείς, ο μόνος λειτουργικός περιορισμός που παρουσιάζουν οι *SQL* ερωτήσεις στην περίπτωση του αλγορίθμου του *Unicode Dewey – UPrefix* - , οφείλεται στην παρουσία αυτών των ορισμένων από τον χρήστη και ενσωματωμένων συναρτήσεων (*next(), previous(),* και *max_prefix()*) στις συνθήκες εν γένει όλων των επερωτήσεων που αφορούν το πεδίο-γνώρισμα *label*. Όμως, η ύπαρξη αυτών των συναρτήσεων έχει σαν αποτέλεσμα η αποτίμηση αυτών των επερωτήσεων από την *SQL* μηχανή να γίνεται χωρίς να λαμβάνεται σε καμία περίπτωση η ύπαρξη του δείκτη που έχει οριστεί στο γνώρισμα *label* με απώτερο στόχο την βελτιστοποίηση της απόδοσής τους. Για να υπερκεραστεί στο μέγιστο δυνατό βαθμό κι άμεσα το εν λόγω πρόβλημα, οποτεδήποτε είναι δυνατόν οι εν λόγω ενσωματωμένες συναρτήσεις αποτιμούνται πρότερα από την καθ' αυτού *SQL* επερώτηση. Για παράδειγμα, ας θεωρήσουμε την επερώτηση αναζήτησης των απογόνων της ριζικής κλάσης *Resource*, που εμπεριέχει την συνθήκη *label* > '1' ∧ *label* < *next*('1'). Δεδομένου ότι η συνάρτηση *next()* εφαρμόζεται άμεσα στον εισαγόμενο κόμβο της επερώτησης (στο συγκεκριμένο παράδειγμα το '1' ως κωδικός της κλάσης *Resource*), μπορούμε να προχωρήσουμε σε αντικατάσταση της συνθήκης αυτής ως *label* > '1' ∧ *label* < '2', καθώς *next*('1') = '2' και θεωρώντας ότι η συνάρτηση *next()* έχει προ-υπολογισθεί. Όμως η επανεγγραφή αυτή των συνθηκών αυτών δεν πάντα δυνατή, όπως εύγλωττα παρατηρούμε στην επερώτηση των "*φύλλων*", όπου η συνάρτηση *next()* χρησιμοποιείται στη εσωτερική υπο-επερώτηση (*subquery*) πάνω στους κωδικούς, που επιστρέφονται από το εξωτερικό μπλοκ.

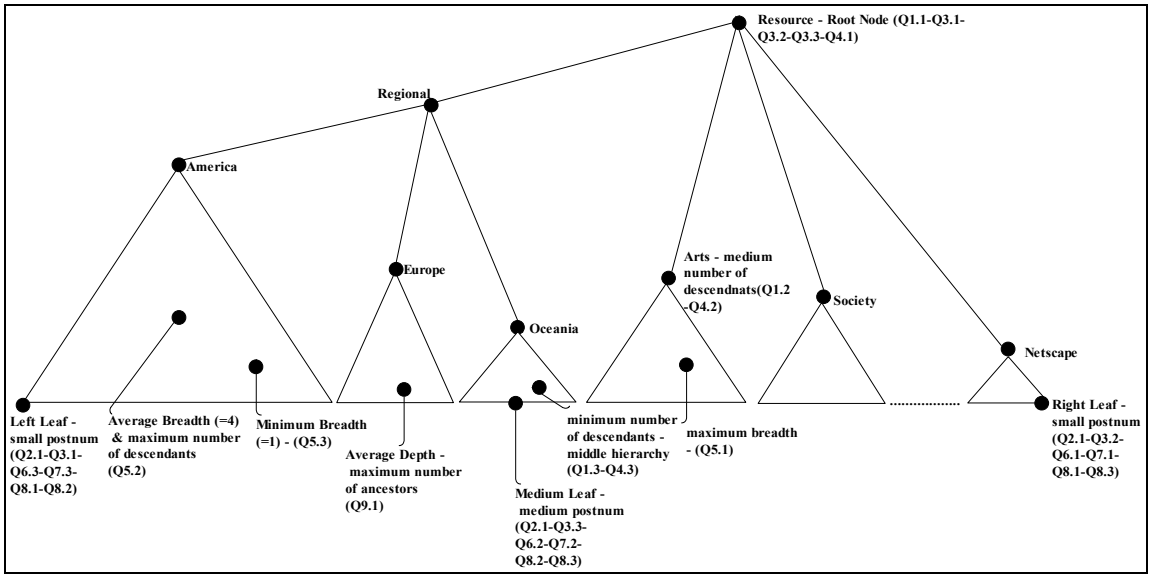
Στην συνέχεια ακολουθούν στον Πίνακα 8 οι καταγραφέντες χρόνοι επερώτησης, το πλήθος των επιστρεφόμενων πλειάδων καθώς και το ποσοστό επιλεκτικότητάς τους επί του συνολικού πλήθους των πλειάδων

Επερωτήσεις	Περιπτώσεις Επερωτήσεων	Agrawal et al	Unicode Dewey	Return tuples	Selectivity of Returned tuples(%)
Q1:Descendants					
	Q1.1 Root node	2,869392*	3,296126	253214	99,99960508
	Q1.2 Node with medium number of descendants	0.429730	0.57702	25314	9,99703809
	Q1.3 Node with a minimum number of descendants at the middle of hierarchy	0.00027	0.00028	1	0,000394921
Q2 : Ancestors					
	Q2.1 Left leaf with a small postnum	1,18904	0.00163	5	0,001974607
	Q2.2 Right leaf with a big postnum	0.00027	0.00167	4	0,001579685
	Q2.3 Medium leaf with a medium postnum	1,547990	0.00178	9	0,003554292
Q3:Subsumption Test					
	Q3.1 Root-Node – Left leaf	0.00018	0.00017	0	0
	Q3.2 Root-Node – Right leaf	0.00017	0.00017	0	0
	Q3.3 Root Node - Medium leaf	0.00018	0.00017	0	0
Q4 : Leaves					
	Q4.1 Root Node	2,957039	25,542670	190231	75,12627609
	Q4.2 Node with a medium number of descendants	0,493230	16,519630	21419	8,45881958
	Q4.3 Node with a minimum number of descendants at the middle of hierarchy	0,000270	0,000550	1	0,000394921
Q5 : Brothers					
	Q5.1 Node with the maximum breadth (=314)	0,008630	0,009610	313	0,123610371
	Q5.2 Node with the AVG breadth (=4) and the maximum number of descendants	0,000540	0,000560	3	0,001184764
	Q5.3 Node with the minimum breadth (=1)	0,000470	0,000490	1	0,000394921
Q6:Precceedings					
	Q6.1 Big Postnum (Leaf node)	2,629188	2,874457	253211	99,99842031
	Q6.2 Medium Postnum (Leaf node)	2,276290	2,345290	126623	50,00612128
	Q6.3 Small Postnum (Leaf node)	0,000240	0,000500	0	0
Q7 : Followings					
	Q7.1 Big Postnum (Leaf node)	0,000240	0,000240	0	0
	Q7.2 Medium	2,485790	2,566292	126592	49,99387872

	Postnum (Leaf node)				
	Q7.3 Small Postnum (Leaf node)	2,9721	3,378250	253209	99,99763047
Q8 : NCA					
	Q8.1 Left leaf –Right leaf	3,227420	0,000530	1	0,000394921
	Q8.2 Left leaf - Medium leaf	0,000410	0,000570	1	0,000394921
	Q8.3 Medium leaf – Right leaf	0,000438	0,000530	1	0,000394921

Πίνακας 8 : Χρόνοι Εκτέλεσης των βασικών Επερωτήσεων στο δέντρο Υποκλάσεων του ODP καταλόγου

Η επιλογή των κόμβων που χρησιμοποιήθηκαν σε όλες τις επιμέρους ερωτήσεις κι υποπεριπτώσεις τους είναι ενδεικτική λαμβάνοντας υπόψη την θέση τους μέσα στην μορφολογία και την είδος της διασύνδεσής τους με τους υπόλοιπους κόμβους. Αναλυτικά, η επιλογή όλων των κόμβων εικονίζεται ανάγλυφα στο σχήμα 44 και επεξηγούμε εν συνεχεία όλες τις επερωτήσεις.



Σχήμα 44 : Κατανομή των Κόμβων Επερωτήσεων στην ιεραρχία του ODP κατάλογου

Στην επερώτηση **Q1** παρατηρούμε ότι και οι τρεις διαφορετικές περιπτώσεις που εξετάζονται αντιστοιχούν στην επιλογή ενός διαφορετικού κόμβου για τον οποίο θέλουμε να υπολογίσουμε τους απογόνους του : (a) στην περίπτωση Q1.1 του ριζικού κόμβου όλης της ιεραρχίας (*Resource*) (b) στην περίπτωση Q1.2 ενός κόμβου με έναν μεσαίο κατά προσέγγιση αριθμό απογόνων (της κλάσης *Arts*) και (c) στην περίπτωση Q1.3 ενός κόμβου με τον ελάχιστο δυνατό αριθμό απογόνων. Στην περίπτωση Q1.2 και Q1.3 το αναζητούμενο πεδίο-κωδικός ή ο αλφαριθμητικός κωδικός (*label*) αντίστοιχα σε κάθε περίπτωση εμφανίζεται στην μέση των *post* και

label διαστημάτων τιμών. Ο βελτιστοποιητής της PostgreSQL και για τις δυο κωδικοποιήσεις επιλέγει *σειριακή σάρωση* (*sequential scan*) για την μεν πρώτη εξεταζόμενη περίπτωση (Q1.1), ενώ χρησιμοποιεί *σάρωση με δείκτες* (*index scan*) για τις δε δυο άλλες περιπτώσεις (Q1.2 & Q1.3). Κι αυτό είναι λογικό εάν λάβουμε υπόψη μας ότι η επερώτηση του διαστήματος βασίζεται αποκλειστικά στο γνώρισμα *post* (δηλαδή $i <= post < p$) ή εναλλακτικά του *label* (δηλαδή $l < label < l'$), τότε η σάρωση με δείκτες κρίνεται η βέλτιστη από άποψη χρονικού κόστους εκτέλεσης. Συγκεκριμένα, ο βελτιστοποιητής της *PostgreSQL* χρησιμοποιεί τον ορισμένο δείκτη για να αποκτήσει πρόσβαση στην πλειάδα, η οποία ικανοποιεί την συνθήκη του μικρότερου ορίου του πεδίου – κωδικού, και δεδομένου ότι τα κλειδιά των εξεταζόμενων δεικτών είναι εκ του προοιμίου ταξινομημένοι, από το ορισμένο από εμάς B^+*-tree*, κατά αύξουσα σειρά, και σταμάτα την σειριακή σάρωση των πλειάδων όταν το άνω άκρο έχει προσεγγιστεί.

Στην επερώτηση **Q2** έχουμε τις εξής τρεις περιπτώσεις : (a) περίπτωση Q2.1 του πιο αριστερού και συνάμα πιο μικρού σε απόλυτη τιμή "φύλλου" - κόμβου της ιεραρχίας (b) περίπτωση Q2.2 ενός "φύλλου" - κόμβου σχετικά στην μέση της ιεραρχίας και (c) περίπτωση Q2.2 ενός "φύλλου" – κόμβου, ο οποίος κατά αντιστοιχία θα είναι ο πιο δεξιός στην υπό εξέταση ιεραρχία υποκλάσεων του ODP καταλόγου. Παρατηρώντας τον πίνακα 8 διαπιστώνουμε από τους χρόνους εκτέλεσης πως ο χρόνος ανταπόκρισης είναι σημαντικά καλύτερος στην περίπτωση του σχήματος του *UPrefix* αλγορίθμου κωδικοποίησης στις δυο πρώτες περιπτώσεις. Ο βελτιστοποιητής της *PostgreSQL* στη κατηγορία ερώτησης αυτής και για το σχήμα του *PInterval* – δεδομένου ότι στο σχήμα του *UPrefix* χρησιμοποιείται ενσωματωμένη συνάρτηση και άρα δεν υφίσταται θέμα χρήσης του δείκτη – επιλέγει για την μεν πρώτη περίπτωση Q2.1 σειριακή σάρωση και σάρωση με δείκτες στις δυο άλλες περιπτώσεις Q2.2 και Q2.3 αντίστοιχα. Στην κατηγορία αυτή τώρα της επερώτησης διαστήματος, ο βελτιστοποιητής βασίζεται σε διαφορετικά γνωρίσματα, που είναι τα *post* και *index* αντιστοίχως, υπό την συνθήκη $index <= p \wedge p < post$. Επίσης, όλες οι επιμέρους τιμές, οι οποίες επιστρέφονται από την διεξαγωγή της σάρωσης των δεικτών στο γνώρισμα *post*, θα πρέπει οπωσδήποτε να σαρωθούν κι έτσι να εξεταστούν κι ως προς την πρώτη συνθήκη, που είναι ως προς το γνώρισμα *index*. Τέλος, παρατηρούμε πως ο βελτιστοποιητής πραγματοποιεί μια καθ' όλα λανθασμένη προσέγγιση γύρω από την σύζευξη που υπάρχει στην εν λόγω χρησιμοποιούμενη συνθήκη και μάλιστα στην περίπτωση Q2.2 το οποίο

σημασιολογικά περιλαμβάνει το μισό διάστημα, επιλέγει να πραγματοποιήσει σάρωση με δείκτες, το οποίο και προφανώς αποδεικνύεται να είναι πιο χρονοβόρο από ότι θα επιδείκνυε η χρήση σειριακής σάρωσης σε ολόκληρο το υπάρχων διάστημα.

Προχωρώντας τώρα στην κατηγορία επερώτησης **Q3**, έχουμε να παρατηρήσουμε εξ αρχής πως και οι τρεις εξεταζόμενες περιπτώσεις εμπίπτουν στους ίδιους ακριβώς κόμβους της επερώτησης **Q1**, δεδομένου ότι και το πρώτο μέρος της σχετικής επερώτησης και στις δυο περιπτώσεις των σχημάτων κωδικοποίησης είναι κοινό. Συνεπώς, εκ της θεώρησης αυτής ο βελτιστοποιητής επιλέγει λογικά το ίδιο πλάνο εκτέλεσης των επερωτήσεων και για τις τρεις περιπτώσεις και για το πρώτο αυτό μέρος της επερώτησης. Τώρα, στην περίπτωση του σχήματος *PInterval* το επιπλέον κομμάτι της επερώτησης που περιλαμβάνει την καθ' αυτού συνθήκη για τον εάν είναι "*φύλλο*" (*index = post*), έχει εκ των καταγραφέντων νούμερων σχετικά πολύ μικρή επιπρόσθετη αύξηση του συνολικού χρόνου εκτέλεσης της επερώτησης **Q1**. Κι αυτό είναι απολύτως λογικό εάν λάβουμε υπόψη ότι το πλήθος των σελίδων δίσκου που υφίστανται πρόσβαση στην περίπτωση αυτή είναι κοινή με της **Q1**. Από την άλλη πλευρά, σε σχέση με το σχήμα *UPrefix* η εν λόγω επερώτηση υφίσταται μια σημαντική χρονική επιβάρυνση, κι αυτό εξαιτίας της χρήσης της βεβαρημένης από άποψη κόστους φωλιασμένης επερώτησης. Κι αυτό εξηγείται δεδομένου ότι και πάλι παρατηρούμε την χρήση σάρωσης με δείκτες και στις τρεις εξεταζόμενες περιπτώσεις για την φωλιασμένη αυτή επιμέρους επερώτηση, εάν και θα περιμέναμε τουλάχιστον στην πρώτη περίπτωση την χρήση σειριακής σάρωσης.

Εν συνεχεία, έχουμε τις ερωτήσεις **Q3** και **Q4**, οι οποίες αναζητούν τους σειριακά πρότερους (***precedings***) και ακόλουθους (***followings***) με βάση τον κωδικό τους κόμβους. Προφανώς, και οι τρεις εξεταζόμενες επιμέρους περιπτώσεις είναι κοινές και για τις δυο κατηγορίες επερωτήσεων. Αναλυτικά, οι τρεις εξεταζόμενες περιπτώσεις αφορούν (a) τον "φύλλο" - κόμβο με το μικρότερο από άποψη τιμής κωδικό, (b) έναν μεσαίο από άποψη τιμής "φύλλο" - κόμβο και (c) τον "φύλλο" - κόμβο με την μεγαλύτερη από άποψη τιμής κωδικό. Δεδομένου της φύσης τους σε κάθε μεμονωμένη περίπτωση παρουσιάζουν αντίθετη επιλεκτικότητα μεταξύ τους, Δηλαδή, για παράδειγμα στην μεν πρώτη περίπτωση με τον μικρότερο κωδικό η επερώτηση για τους ***precedings*** Q3.1 παρουσιάζει την μέγιστη επιλεκτικότητα (*selectivity*), ενώ η αντίστοιχη εξεταζόμενη περίπτωση για τους ***followings*** Q4.1 μας δίνει μηδαμινή επιλεκτικότητα. Δεν θα πρέπει να ξεχάσουμε να αναφέρουμε σε αυτό

το σημείο πως οι χρόνοι εκτέλεσης των επερωτήσεων αυτών των δυο κατηγοριών, που επιδεικνύουν μηδενική επιλεκτικότητα, μπορούν να χρησιμοποιηθούν και περαιτέρω ώστε να μας δώσουν μια ένδειξη γύρω από το αναγκαίο κόστος αναζήτησης των δεικτών που ορίζεται πάνω στα γνωρίσματα *post* και *label* αντίστοιχα.

Η ερώτηση **Q6** η οποία ασχολείται με την αναζήτηση των ομο-επίπεδων (*siblings*) κι ενίοτε και "αδερφών" κόμβων στις περιπτώσεις του (a) κόμβου με το μεγαλύτερο πλήθος εισαγόμενων ακμών (*fan-in*), (b) ενός κόμβου με το μέσο πλήθος εισαγόμενων ακμών και (c) του κόμβου με το μικρότερο πλήθος ακμών στις ιεραρχίες υποκλάσεων του *ODP* καταλόγου. Το πλάνο εκτέλεσης της ερώτησης αυτής περικλείει ένα φωλιασμένο βρόχο που συνδέει δυο σαρώσεις με δείκτες, όπου η μεν πρώτη βρίσκει τον πατέρα ενός κόμβου και η δε δεύτερη βρίσκει τους άμεσους ομοεπίπεδους κόμβους, που προκύπτουν με τη χρήση των κατηγορημάτων ισότητας στα γνωρίσματα *post* (ή *label* αντιστοίχως) και του πατρικού γνωρίσματος και στα δυο σχήματα.

Τέλος, ολοκληρώνοντας την σειρά αυτή ερωτήσεων έχουμε την επερώτηση **Q7** στην οποία εξετάζουμε την ύπαρξη αλλά και συγκεκριμένα ποιος είναι ο πιο κοντινός σε επίπεδο απόστασης μέσα στην ιεραρχία κοινός "πρόγονος" κόμβος μεταξύ των δυο υπό εξέταση, εισαγόμενων κόμβων. Οι τρεις εξεταζόμενες περιπτώσεις διαμορφώνονται και πάλι με τον συνδυασμό των "φύλλων" – κόμβων που έχουν προεπιλεχτεί στην επερώτηση **Q2** είναι (a) πιο αριστερού "φύλλου" - κόμβου / πιο δεξιού "φύλλου" – κόμβου, (b) πιο αριστερού "φύλλου" - κόμβου / μεσαίου φύλλου" – κόμβου και (c) μεσαίου φύλλου" – κόμβου / πιο δεξιού "φύλλου" – κόμβου. Αναλυτικά, στην μεν περίπτωση του *UPrefix* έχουμε κατασκευάσει και χρησιμοποιήσει μια ενσωματωμένη συνάρτηση, ενώ στο *PInterval* έχουμε μια φωλιασμένη επερώτηση η οποία συντελείται με την σαρώσεις με δείκτες και για τα δυο επιμέρους μπλοκ (μέσω κι έξω) και για τις τρεις περιπτώσεις. Όπως είχαμε προαναφέρει και στην επερώτηση των "απογόνων" και στις τρεις περιπτώσεις τα πλάνα εκτέλεσης περιλαμβάνουν επιμέρους σαρώσεις με δείκτες, εάν και θα έπρεπε στην μεν πρώτη περίπτωση να ήταν σάρωση σειριακή. Ομοίως και εδώ, δεδομένου ότι η επερώτηση *nca* στηρίζεται συνδυαστικά στην χρήση της επερώτηση των "προγόνων" παρουσιάζει αντίστοιχο πλάνο εκτέλεσης. Πάντως σε κάθε περίπτωση ο χρόνος σε αυτή την περίπτωση είναι λογικά πολύ μεγαλύτερος σε σχέση με τις άλλες

δυο, οι οποίες άλλωστε είναι και συγκρίσιμες μεταξύ τους ως προς τον χρόνο εκτέλεσής τους.

4.7 Καθορισμός Πλήθους Ενταμιευτών (Buffers)

Όπως προαναφέραμε για την διεξαγωγή αυτού του πειράματος θεωρήσαμε ως ενδεικτική αλλά και βέλτιστη επιλογή ως προς την χρονική απόδοση των ζητούμενων επερωτήσεων που θα δούμε με πλήρη λεπτομέρεια στην συνέχεια, το πλήθος των 1000 δηλωμένων ενταμιευτών. Παράγοντας που συνέβαλλε στην λήψη της απόφασης σίγουρα είναι κι εκ του σχήματος 33 ότι οι δημιουργούμενοι δείκτες στα γνωρίσματα *post* του σχήματος *PInterval* (με μέγεθος 1001 *disk pages)*; και *label* του σχήματος *UPrefix* (με μέγεθος 697 *disk pages)* μπορούν εξ ολοκλήρου να αποθηκευτούν στην κύρια μνήμη του υπολογιστή και δεν χρειάζεται η χρήση δευτερεύουσας μνήμης με το επιπρόσθετο κόστος των απαραίτητων I/O λειτουργιών. Ωστόσο η διαδικασία επιλογής της τιμής συγκεκριμένης αυτής τιμής ενταμιευτών έγινε με την χρονική απόδοση που είχαν κρίνοντας από κοινού κι εν συνόλω 3 καθ' όλα ενδεικτικές κατηγορίες επερωτήσεων και για τους δυο αλγορίθμους κωδικοποίησης και τις οποίες θα αναλύσουμε με μεγαλύτερη λεπτομέρεια στη συνέχεια. Η καλούμενη ενδεικτικότητα ως μέτρο σύγκρισης που φέρονται να έχουν οι συγκεκριμένες επερωτήσεις αυτές έγκειται σαφώς στο μεγάλο πλήθος των επιστρεφόμενων πλειάδων (*selectivity*) σε σχέση με το μέγεθος του συνόλου των κλάσεων της ιεραρχίας του *ODP* καταλόγου. Κι αυτό γιατί μας δίνει μια τάξη μεγέθους του αναγκαίου χρόνου εκτέλεσης και για τις λοιπές ερωτήσεις, καθώς είναι σε άμεση συνάφεια και με τις λοιπές επιζητούμενες προς αξιολόγηση επερωτήσεις.

Αναλυτικά, η διερεύνηση αυτή έγινε με τις επερωτήσεις :

1. για τους απογόνους του ριζικού κόμβου όλου του *ODP* καταλόγου (*descendants*),
2. για τις κλάσεις που προηγούνται μιας κλάσης, φύλλου-κόμβου, με ένα πολύ μεγάλο *postnum* (*post*) κωδικό και ο οποίος προηγείται αριθμητικά σχεδόν όλων των άλλων κλάσεων του *ODP* καταλόγου (*precedings*),

3. και τέλος για τις κλάσεις που έπονται μια κλάσης, φύλλου-κόμβου επίσης, με πολύ μικρό *postnum* κωδικό και υπολείπεται αριθμητικά σχεδόν όλων των άλλων κλάσεων του *ODP* καταλόγου (*followings*) αντίστοιχα.

Εν συνεχεία, παραθέτουμε με λεπτομέρεια στους πίνακες 9, 10 και 11 αποτελέσματα των χρόνων εκτέλεσης των 3 αυτών επερωτήσεων για τις περιπτώσεις των 1000, 2000, 5000, 10000 και 15000 *buffers* αντίστοιχα, στην προσπάθειά μας να εντοπίσουμε την βέλτιστη επιλογή ιδανικού πλήθους ενταμιευτών για όλο το σύνολο των ζητούμενων προς αξιολόγηση επερωτήσεων.

Descendants		
# buffers	**Agrawal et al**	**Unicode Dewey**
1000	**3.04804**	**3.33152**
2000	**2.90955**	**3.24515**
5000	**2.87826**	**3.12903**
10000	**3.02739**	**3.18627**
15000	**3.21259**	**3.21930**

Πίνακας 9 : Επερώτηση Απογόνων – Descendants

Precedings		
# buffers	**Agrawal et al**	**Unicode Dewey**
1000	**2.68888**	**2.82670**
2000	**2.67182**	**2.84896**
5000	**2.64843**	**3.11631**
10000	**2.75320**	**2.81306**
15000	**2.66717**	**2.80366**

Πίνακας 10 : Επερώτηση Precedings

Followings		
# buffers	**Agrawal et al**	**Unicode Dewey**
1000	**2.73596**	**3.03263**
2000	**2.74088**	**3.02413**
5000	**2.61069**	**3.06901**
10000	**2.71678**	**3.01905**

15000	**2.72422**	**3.01579**

Πίνακας 11 : Επερώτηση Followings

Με βάση τους παραπάνω χρόνους και για τις τρεις αυτές ενδεικτικές ερωτήσεις, (βλ. Πίνακες 9, 10, 11) παρατηρούμε ότι η καλύτερη επιλογή πλήθους ενταμιευτών και για τις δυο επιμέρους κωδικοποιήσεις από κοινού κρινόμενοι, είναι των 1000, εάν και σε μεμονωμένες περιπτώσεις κάποιας κωδικοποίησης, καθώς αυξάνουμε το πλήθος βελτιώνεται ο χρόνος εκτέλεσης μιας συγκεκριμένης επερώτησης. Όμως η τελική επιλογή γίνεται λαμβάνοντας υπόψη και τις τρεις αυτές καθ' όλα ενδεικτικές, βασικές επερωτήσεις και για τις δυο κωδικοποιήσεις.

4.8 Συμπεράσματα

Με βάση την διεξαχθείσα πειραματική αξιολόγηση για την περίπτωση των δέντρων-ιεραρχιών του *ODP* καταλόγου, μπορούμε να καταλήξουμε σε μια σειρά από πολύ ενδιαφέροντα συμπεράσματα. Αρχικά, τα σχήματα αυτά κωδικοποίησης για τις ογκώδεις ιεραρχίες κλάσεων υπαλληλίας παρουσιάζουν μία πράγματι εξαιρετική απόδοση αξιολογούμενα ως προς τον χρόνο εκτέλεσης των βασικών αυτών επερωτήσεων (3 έως 4 τάξεις μεγέθους), συγκρινόμενα σε σχέση με τους υπολογισμούς όλων των μεταβατικών κλεισιμάτων γι αυτά τα σχήματα.[KACP02]. Επιπρόσθετα, το κέρδος αυτό συνοδεύεται κι από όχι σημαντική αύξηση στις απαιτήσεις αποθήκευσης για την περίπτωση των ιεραρχιών με μορφολογία δέντρου, ειδικά για την περίπτωση ετικετών με την μορφή πεδίων – διαστημάτων. Συγκεκριμένα, το σχήμα του *UPrefix* υπερέχει του *PInterval* κατά 21,2% ως προς τον χώρο αποθήκευσης των δεδομένων του. Τώρα ο αντίστοιχος αναγκαίος χώρος αποθήκευσης για τον δημιουργούμενο δείκτη πάνω στο αναγνωριστικό πεδίο-γνώρισμα *label* του *UPrefix* είναι 29,8% μεγαλύτερο έναντι του πεδίου *post* του *PInteral*. Τώρα ως προς τον απαιτούμενο χρόνο αποθήκευσης των δεδομένων ο *UPrefix* υπερέχει του *PInterval* κατά 34,75%, ενώ για τη δημιουργία των δεικτών πάνω στα αναγνωριστικά αυτά πεδία και για τα δυο σχήματα αντίστοιχα ο *UPrefix* υπερέχει του *PInterval* κατά 32,21%.

Οι αποδόσεις των επερωτήσεων και για τα δυο σχήματα είναι συγκρίσιμες (αναφορικά υπάρχει μια μικρή υπεροχή απόδοσης του *Agrawal et al* σχήματος έναντι

του *Unicode Dewey*). Κι αυτό οφείλεται στον αναγκαίο αποθηκευτικό χώρο που απαιτείται και για τα δυο σχήματα, για το μεν *PInterval* 697 disk pages, και για το δε *UPrefix* 1001 disk pages, κρίνοντας ως προς τις ίδιες επερωτήσεις κάθε φορά με κοινό αριθμό επιλεκτικότητας επιστρεφόμενων πλειάδων.

Κεφάλαιο 5

Αξιολόγηση Σχημάτων Κωδικοποίησης - με πρόθεμα και με διάστημα - Περίπτωση των DAGs

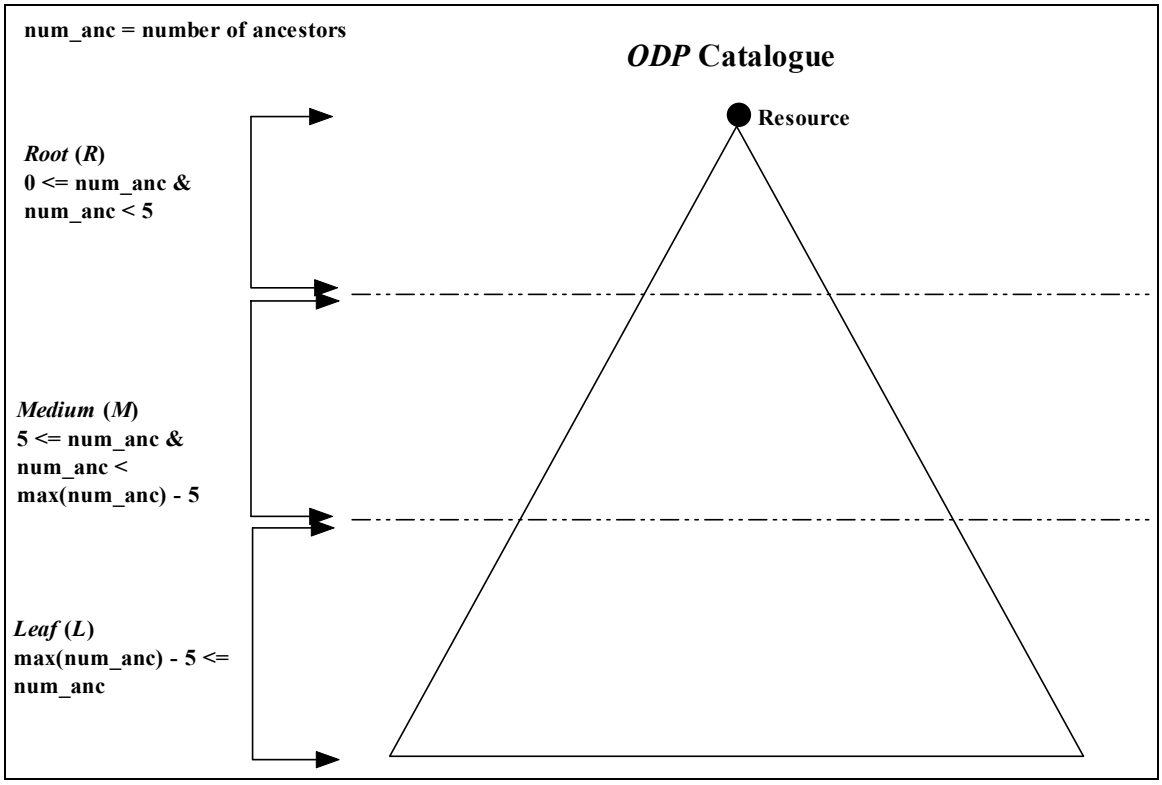
Στο κεφάλαιο αυτό θα προχωρήσουμε στην πειραματική αξιολόγηση των δυο αντιπροσωπευτικών σχημάτων κωδικοποιήσεων που είδαμε στο κεφάλαιο 4, δηλαδή των *Agrawal, Borgida* και *Jagadish* (*Agrawal et al*) - *PInterval* και του *Unicode Dewey – Uprefix,* αντιστοίχως τώρα για την περίπτωση ιεραρχιών που έχουν μορφολογία γράφου (*DAG)*. Συγκεκριμένα, για την πραγματοποίηση όλων αυτών των πειραμάτων χρησιμοποιήσαμε και πάλι σαν πεδίο δοκιμής αρχικά τα *RDF* σχήματα του *ODP* καταλόγου –έκδοση 16-01-2001, τα οποία προαναφέραμε στην προηγούμενη ενότητα. Όμως, εξετάσαμε τον κατάλογο του *ODP* για την περίπτωση των κατευθυνόμενων άκυκλων γράφων *DAGs*, τους οποίους δημιουργήσαμε από τις ιεραρχίες του *ODP* καταλόγου, προσθέτοντας με συνθετικό τρόπο πολλαπλές σχέσεις *isA* στους ήδη υπάρχοντες κόμβους κι ιεραρχίες.

Θα ασχοληθούμε με την σχεσιακή αναπαράσταση των σχημάτων *PInterval* και *UPrefix* πλέον για την περίπτωση ενός *DAG* υποκλάσεων και εν συνεχεία πρωτίστως την αξιολόγησή τους σε σχέση με το επιπλέον κόστος αποθήκευσής τους. Κατόπιν αυτού του βήματος και πάλι σε πλήρη αντιστοιχία θα προχωρήσουμε με την αξιολόγηση των ζητούμενων επερωτήσεων, χρησιμοποιώντας την πλατφόρμα της *PostgreSQL*, στην βέλτιστη δυνατή μετάφραση των διαφορετικών τύπων των επερωτήσεων που επιζητούμε στη μορφή *SQL* επερωτήσεων για την περίπτωση σχημάτων από *DAGs* υποκλάσεων. Σε πλήρη αντιστοιχία με την προηγηθείσα περίπτωση των δέντρων οι ζητούμενες επερωτήσεις είναι οι ερωτήσεις για τον έλεγχο υπαλληλίας (*subsumption*), για τους απογόνους, τους προγόνους, τους "φύλλα" - κόμβους, τους ομο-επίπεδους κόμβους και της αναζήτησης των πιο "κοντινών" κοινών προγόνων. Δεν συμπεριλαμβάνονται σε αυτές οι ερωτήσεις για τους σειριακά

πρότερους (***precedings***) και ακόλουθους (***followings***) με βάση τον κωδικό κόμβους, καθώς δεν μπορούν να οριστούν στην περίπτωση των *DAGs*.

5.1 Διαδικασία Κατασκευής DAGs

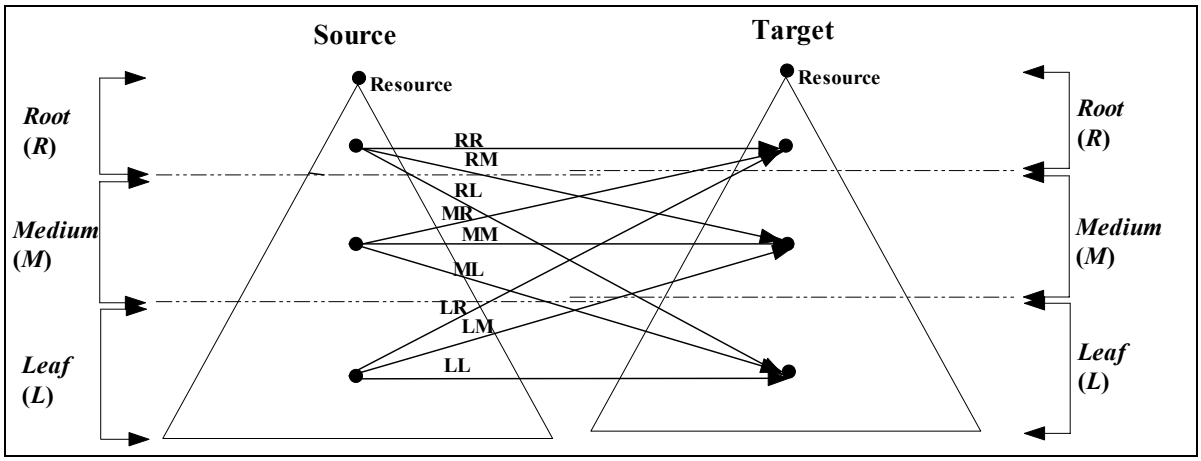
Επί των αρχικών ιεραρχιών του *ODP* καταλόγου που έχουμε, θεωρούμε ότι οι κλάσεις μπορούν να ταξινομηθούν σε τρεις διαφορετικές κατηγορίες ανάλογα με το πλήθος των προγόνων τους (*num_anc*) μέσα στην ιεραρχία – δέντρο, το οποίο έχει μηδενική τιμή ξεκινώντας από τον ριζικό κόμβο και αυξάνει καθώς κατευθυνόμαστε στο κάτω μέρος της ιεραρχίας. Συγκεκριμένα, κατηγοριοποιούνται ως : (a) κοντά στο ριζικό κόμβο (*Root*) και τους δηλώνουμε με το *R* όλους τους κόμβους εκείνους οι έχουν πλήθος προγόνων ίσο μέχρι και 5 στην ιεραρχία (***0*** $\leq$***num_anc*** $\leq$ ***5***), (b) κοντά στους κόμβους – "φύλλα", που τους σημειώνουμε με *L* και περιλαμβάνονται όλοι οι κόμβοι με πλήθος προγόνων μεγαλύτερο της μέγιστης τιμής πλήθους προγόνων, αναφερόμενοι για όλες τις κλάσεις του *ODP* καταλόγου, μειούμενης κατά 5 – (***num_anc*** $\geq$***max(num_anc) -5*****),** όπως βλέπουμε στον Πίνακα 2., και τέλος όλους τους λοιπούς κόμβους που βρίσκονται ενδιάμεσα στους δυο προηγούμενους – ενδιάμεσους -, που τους δηλώνουμε με το *M* (βλ. Σχήμα 45) και θεωρούμε όλους τους κόμβους με πλήθος προγόνων μεγαλύτερου του 5 και μικρότερο από την μέγιστη τιμή του πλήθους των προγόνων, αναφερόμενοι από κοινού στον *ODP* καταλόγου, μειούμενης κατά 5 (***5*** $<$ ***num_anc*** $\leq$***max(num_anc)-5***). Η κατηγοριοποίηση των κλάσεων με βάση το βάθος τους βασίστηκε στο σχήμα 3. , και η επιλογή των πεδίων τιμών της κατηγοριοποίησης αυτής στοχεύει στην καλύτερη σύγκριση της διαδικασίας αλλά και του πλήθους μετάδοσης των ετικετών (*bottom-up* για τον *Agrawal et al* και *top-down* για τον *Unicode Dewey*), καθώς και την απόδοση των συμπιέσεων. Μάλιστα οι παρατηρήσεις αυτές δεν είναι μόνο ποσοτικές αλλά και ποιοτικές καθώς εν συνεχεία θα δούμε ότι υπάρχει και χαρακτηρισμός (κατηγοριοποίηση) των μεταδόσεων.



Σχήμα 45 : Προτεινόμενη Κατανομή Κόμβων ODP Καταλόγου ως προς το πλήθος των προγόνων τους

Με βάση την ταξινόμηση αυτή των κόμβων –κλάσεων μπορούμε να διαμορφώσουμε το είδος αλλά πιο συγκεκριμένα το πλήθος ανά είδος των επιπρόσθετων *non-spanning tree* ακμών που προσθέτουμε για να συνθέσουμε το *DAG* μας. Αναλυτικά, η επιλογή των κόμβων "αφετηρία" κι "προορισμός" της εκάστοτε προστιθέμενης *non-spanning tree* ακμής γίνεται με την χρήση μιας γραμμικώς συγκλίνουσας, ψευδοτυχαίας γεννήτριας, η οποία ακολουθεί την ομοιόμορφη κατανομή κι έχει την ιδιότητα ότι ισοκατανέμει το πλήθος των διακριτών τιμών του εισαγόμενου πεδίου τιμών με μια μοναδική τιμή πιθανότητας μεταξύ του 0 και του 1 αντίστοιχα. Η επιλογή της χρήση της γεννήτριας αυτής είναι επιβεβλημένη καθώς επιδιώκουμε την τυχαιότητα στην επιλογή και κατόπιν εξέταση των κόμβων, και δεν επιλέγουμε την *a priori* δεδομένη σειρά των κόμβων, με την οποία προκύπτουν από την χρησιμοποιούμενη εσωτερική δομή που είναι αποθηκευμένος ο επί του παρόντος υφιστάμενος γράφος. Επίσης, με βάση την ακολουθούμενη διαδικασία οι κόμβοι ισοκατανέμονται στις εννέα κατηγορίες, οι οποίες διαμορφώνονται από όλους τους δυνατούς συνδυασμούς των τριών προαναφερθέντων δυνατών χαρακτηρισμών εξίσου για τους κόμβους "αφετηρία" κι "προορισμός" για την κάθε προστιθέμενη ακμή του γράφου. Άρα, το σύνολο των

κατηγοριών αυτών κατάταξης είναι : RR, RM, RL, MR, MM, ML, LR, LM, και LR, ως προς την σειρά εξέτασής τους. Στο σχήμα 46 έχουμε να παρατηρήσουμε πως είναι πιθανό οι κόμβοι *Source* και *Target* της κάθε προστιθέμενης ακμής να είναι στην ίδια υποιεραρχία (πχ *Regional, Science* κλπ) ή και σε διαφορετικές.



Σχήμα 46 : Χαρακτηρισμός Προστιθέμενων non-spanning-tree ακμών

Επίσης να αναφέρουμε πως στην διαμόρφωση του DAG με την σύνθεση της δομής που θέλουμε, προσδιορίσαμε κι μια άλλη παράμετρο που είναι ο μέγιστος αριθμός των εξαγόμενων ακμών από έναν κόμβο (*maximum fan-out*) ή αλλιώς ο μέγιστος αριθμός άμεσων πατρικών κόμβων κι ο οποίος επιλέχτηκε να είναι στην τιμή 2. Η απόφαση αυτή λήφθηκε με γνώμονα μια ευρύτατη στατιστική μελέτη [MACP02], στην οποία αναφέρονται μια σειρά από κι άλλα μορφολογικά μετρικά (*topology metrics*) μιας εκτενής συλλογής από υπάρχουσες κι ευρέως χρησιμοποιούμενες *RDF/S* ιεραρχίες, στις οποίες σαφώς παρατηρείται ως ένα αντικειμενικά ανώτατο όριο πλήθους άμεσων πατρικών κόμβων μη-ριζικών κόμβων η τιμή 2.

5.2 Χρονική Διάρκεια Κατασκευής Κωδικών

Όπως και στην μελέτη της περίπτωσης διαχείρισης του δέντρου, έτσι κι εδώ στην περίπτωση του συνθετικού μας γράφου προχωρούμε σε μια σύγκριση των δυο σχημάτων κωδικοποίησης επί του αναγκαίου χρόνου δημιουργίας των αντίστοιχων κωδικών για τις κλάσεις των RDF σχημάτων που προκύπτουν με την προσθήκη ανά βήμα ενός συγκεκριμένου ποσοστού κάθε φορά non-tree ακμών (5%) στις ιεραρχίες του ODP καταλόγου. Ο χρόνος αυτός ισούται με τον αναγκαίο χρόνο παραγωγής

κωδικών για όλες τις κλάσεις του εξεταζόμενου προς κωδικοποίηση *RDF* σχήματος. Προφανώς, θα πρέπει να παρατηρήσουμε πως ανά βήμα σύνθεσης το υπάρχων RDF σχήμα που θέλουμε να κωδικοποιήσουμε τροποποιείται. Στον Πίνακα 1 παραθέτουμε τους ληφθέντες χρόνους (μετρημένους σε seconds) που χρειαστήκαμε για την κωδικοποίηση των DAG ιεραρχιών του ODP καταλόγου τόσο με τον αλγόριθμο του Agrawal et al όσο και του Unicode Dewey. Η μέτρηση των χρόνων έγινε, όπως και στην περίπτωση του δέντρου, στο ίδιο μηχάνημα τύπου Sun-Blade-1000, που έχει έναν Ultra SPARC-IV 400MHz επεξεργαστή και 1152 MB κύριας μνήμης, συγκεκριμένα με τη χρήση των συναρτήσεων μέτρησης του αναγκαίου χρόνου ολοκλήρωσης της κάθε κωδικοποίησης.

Ποσοστό προσθήκης *non - spanning tree* ακμών στον ODP Κατάλογο (%)	***Agrawal et al***	***Unicode Dewey***
0	**205,680**	**92,398**
5	**221,893**	**277,331**
10	**236,318**	**176,668**
15	**228,875**	**166,556**
20	**235,469**	**175,795**
25	**232,321**	**171,367**
30	**235,976**	**180,622**
35	**248,877**	**174,638**
40	**293,896**	**182,255**
45	**414,581**	**179,821**
50	**636,851**	**184,742**
55	**1183,047**	**190,358**
60	**2405,315**	**189,521**
65	**4861,191**	**191,603**
70	**9904,638**	**201,692**
75	**18613,378**	**202,534**
80	**35932,359**	**206,261**
85	**66559,042**	**197,002**
90	**123515,198**	**206,294**
95	**218488,906**	**208,115**
100	**386490,106**	**210,254**
Σύνολο	**871143,917**	***3965,827***

Πίνακας 12 : Χρόνος Κατασκευής Κωδικών (σε seconds)

Συγκρίνοντας τον αναγκαίο χρόνο κωδικοποίησης του *ODP* καταλόγου για την περίπτωση του δέντρου που είδαμε στο κεφάλαιο 4 και εδώ για την περίπτωση του γράφου παρατηρούμε ότι ο χρόνος κωδικοποίησης του τρέχοντος συνθετικού γράφου είναι πολλαπλάσιος του αντίστοιχου χρόνου στην περίπτωση του δέντρου. Κι αυτό είναι λογικό κι οφείλεται στον αυξημένο, αναγκαίο χρόνο επαναϋπολογισμού του βέλτιστου *spanning-tree* του υπάρχοντος γράφου, που αυξάνει αντίστοιχα, όπως παρατηρούμε και από τους παραπάνω χρόνους εκτέλεσης, με την πολυπλοκότητα του υφιστάμενου γράφου, σε κάθε επιμέρους βήμα προσθήκης 5% επί του αρχικού συνόλου των υπαρχόντων *non-spanning tree* ακμών του. Αντιθέτως, στην περίπτωση του δέντρου, δεν υπάρχει λόγος κλήσης της διαδικασίας διαμόρφωσης του βέλτιστου *spanning-tree*, λόγω της υφιστάμενης δενδρικής μορφολογίας.

5.3 Σχεσιακή Αναπαράσταση και Μέγεθος Αποθήκευσης

Για την περίπτωση των *DAGs* χρειαζόμαστε για κάθε σχήμα κωδικοποίησης έναν επιπλέον πίνακα από αυτόν που έχουμε αναφέρει στην περίπτωση του δέντρου. Συγκεκριμένα, στην αναπαράσταση αυτή χρησιμοποιούμε σαν πρώτο πίνακα τους προαναφερθέντες *UPrefix* και *PInterval* για τα δυο επιμέρους σχήματα κωδικοποίησης. Η μόνη τροποποίηση όμως η οποία προκύπτει αβίαστα στην περίπτωση αυτή των *DAGs*, είναι ότι οι πρώτοι αυτοί πίνακες πλέον δεν θα περιέχουν μόνο τις πλειάδες που προκύπτουν από τις ακμές του δέντρου που είχαμε μέχρι στιγμής, αλλά κάθε είδους ακμές (δηλαδή θα περιλαμβάνει ουσιαστικά και τις ακμές του γράφου που είτε προέρχονται από το επιλεχθέν *spanning-tree* ή ακμές που δεν προέρχονται από το *spanning-tree – non-spanning-tree* ακμές). Η λογική της επιλογής της επέκτασης αυτής του πρωτεύοντας αυτού πίνακα έγκειται στο ότι η επερώτηση εύρεσης των *ομοεπίπεδων* (*siblings*) – όπως ομοίως και των επερωτήσεων των άμεσων πατρικών και παιδιών ενός κόμβου - μπορεί πλέον εύκολα να πραγματοποιηθεί με βάση τους πρωτεύοντες πίνακες χρησιμοποιώντας το πεδίο-γνώρισμα του πατρικού κωδικού (*father_post* και *father_label*), όπως άλλωστε και στην περίπτωση των δέντρων. Συνεπώς, εκ της επιλογής αυτής προχωρούμε σε μια προέκταση των κλειδιών και των δυο πινάκων με στόχο να συμπεριλάβουμε σε κάθε περίπτωση το πατρικό πεδίο-γνώρισμα (*father_label* ή *father_post* εναλλακτικά για το κάθε σχήμα), και άρα θα έχουμε το κάτωθι ακόλουθο σχήμα αναπαράστασης :

PInterval(*post* : int4, *index* : int4, *father_post* : int4)
UPrefix (*label* : varchar(15), *father_label* : varchar(15)

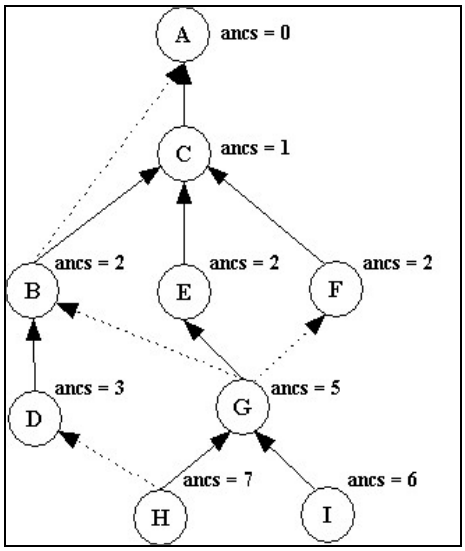
Εν συνεχεία, απαιτείται η εισαγωγή ενός επιπρόσθετου, δευτερεύοντος πίνακα και για τα δυο σχήματα που έστω καλούνται *DPInterval* και *DUPrefix* αντιστοίχως – θεωρούμε ότι προσθέτουμε το πρόθεμα *D* στην ονομασία των δυο προϋπαρχόντων πινάκων εκ του γεγονότος ότι οι πίνακες αυτοί σχετίζονται με *DAGs*. Η διαμόρφωση της δομής αυτών των πινάκων εξαρτάται αρκετά από την σημασιολογία που υπάρχει στους πρωτεύοντες πίνακες. Δηλαδή στην περίπτωση του *DPInterval* θα έχουμε και πάλι την δομή (*post*, *index*, *father_post*), όπου πλέον η κάθε μεμονωμένη πλειάδα του θα δεικνύει ότι ο κόμβο με το καθ' όλα αναγνωριστικό πεδίο [*index, post*] μεταδίδει τον κωδικό του προς τους επάνω κόμβους με τους οποίους ενώνεται άμεσα, κι οι οποίοι αναγνωρίζονται με την τιμή τους στο post πεδίο και το οποίο αποτυπώνεται στο γνώρισμα *ancestor*. Ομοίως στον δευτερεύων πίνακα του σχήματος κωδικοποίησης του *UPrefix* θα έχουμε παρόμοια δομή με του πρωτεύοντος, δηλαδή (*label, father_label*), όπου η κάθε πλειάδα του θα δεικνύει ότι ο κόμβος που ταυτοποιείται μοναδικά με το αλφαριθμητικό κωδικό *father_label,* μεταδίδει προς τα κάτω τον κωδικό του, συγκεκριμένα στον κόμβο που με την σειρά του αυτοπροσδιορίζεται με τον κωδικό *label*.

Στο σημείο αυτό θα πρέπει να σημειώσουμε πως ο ορισμός κλειδιών σε αυτούς τους δευτερεύοντες πίνακες δεν είναι σε καμία περίπτωση υποχρεωτικός, καθώς εκ της χρήσης τους στις επερωτήσεις όπως θα δούμε εν συνεχεία δεν αντιμετωπίζονται μοναδικά κι ανεξάρτητα από τον πρωτεύοντα πίνακα του σχήματος κωδικοποίησης που ανήκουν. Συγκεκριμένα, έχουμε ορίσει δείκτες πάνω στα πεδία - γνωρίσματα *post, father_post, label* και *father_label* αντιστοίχως (βλ. υπογραμμισμένα πεδία στις ακόλουθες δηλώσεις). Παρακάτω, παραθέτουμε τον σαφή ορισμό των δευτερευόντων πινάκων :

DPInterval(*post* : int4, *index* : int4, *ancestor* : int4)
DUPrefix (*label* : varchar(15), *ancestor* : varchar(15)

Στη συνέχεια θα παραθέσουμε ένα παράδειγμα διαμόρφωσης και πλήρωσης των πινάκων αυτών για τον γράφο του σχήματος 47 κι ενδεικτικά για το σχήμα

DPInterval (αντιστοίχως ισχύουν και για το σχήμα του *DUPrefix*). Αναλυτικά, παραθέτουμε στο σχήμα 47 πως διαμορφώνονται οι πίνακες *DPInterval* χωρίς τη συμπίεση και με συμπίεση.



Σχήμα 47: Παράδειγμα Γράφου DAG προς κωδικοποίηση

***DPInterval* (with compression)**

post	*index*	*ancestor*
3	3	1
5	3	7
5	3	2
5	3	8

***DPInterval* (no compression)**

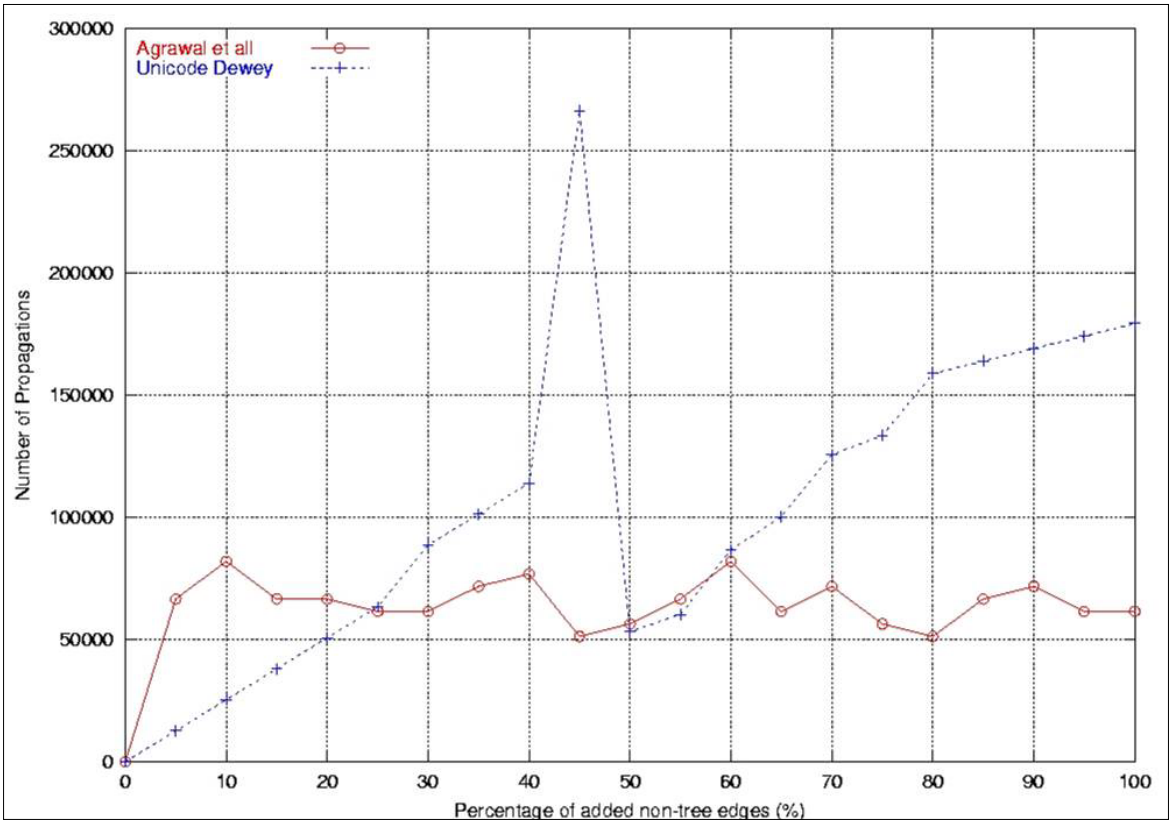
post	*index*	*ancestor*	
2	1	9	A
3	3	9	
5	3	9	
3	3	2	B
5	3	2	
3	3	8	C
5	3	8	
3	3	1	D
5	3	7	F

Σχήμα 48 : Παράδειγμα σχηματισμού των πινάκων DPInteval α) με συμπίεση και β) χωρίς συμπίεση για τον γράφο του σχήματος 47

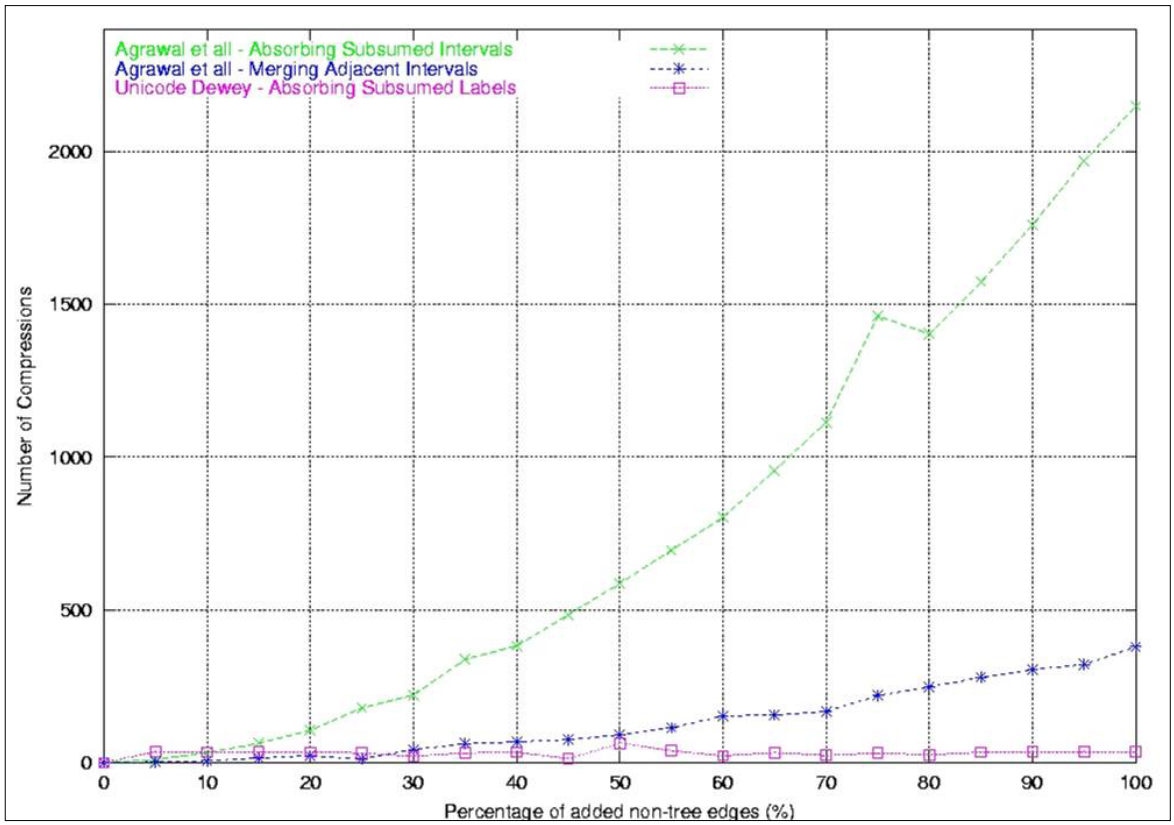
5.4 Χώρος και Χρόνος Αποθήκευσης

Εξ αρχής εκ της θεώρησης των δευτερευόντων πινάκων που πλαισιώνουν τους ορισθέντες πίνακες που είχαμε ορίσει στην περίπτωση των δέντρων, είναι προφανές ότι και για τα δυο σχήματα κωδικοποίησης θα έχουμε ένα επιπλέον κόστος αποθήκευσης των *DAGs*. Δεδομένου, ότι και στους δυο πρωτεύοντες πίνακες των δυο

σχημάτων κωδικοποίησης *PInterval* και *UPrefix* περιέχονται όλες οι ακμές του εξεταζόμενου γράφου μας – *DAG* – (η από κοινού αυτή αποθήκευση αποσκοπεί στην επανακατασκευή του γράφου μας σε μορφή *RDF/S*), το επιπλέον κόστος αποθήκευσης των σχημάτων ισούται με το μέγεθος των πινάκων *DPInterval* και *DUPrefix*. Άρα, συνεπάγεται πως για τον υπολογισμό του επιπλέον αυτού κόστους χρειάζεται να υπολογίσουμε το πλήθος των κωδικών που μεταδίδονται (*propagated*) σε κάθε σχήμα ξεχωριστά με διαφορετική κατεύθυνση μετάδοσης και δομής αποθήκευσης στους δευτερεύοντες αυτούς πίνακες. Η μετάδοση αυτή των κωδικών (προς τα επάνω ή προς τα κάτω) εξαρτάται από την θέση μέσα στην ιεραρχία του υπάρχοντος γράφου των δυο άκρων της κάθε κατευθυνόμενης *non-spanning tree* ακμής, που προστίθεται στο *DAG*, δηλαδή από τον κόμβο "αφετηρία" αλλά κι από τον κόμβο "τερματισμός". Μάλιστα για να είμαστε πιο συγκεκριμένοι το πλήθος αυτών των μεταδόσεων εξαρτάται ακριβώς από το πλήθος των απογόνων (προγόνων εναλλακτικά) των κόμβων "αφετηρίας" ("τερματισμού" εναλλακτικά) των *non-spanning tree* ακμών του *DAG* για την περίπτωση των σχημάτων των *DUPrefix* και *DPInterval* αντιστοίχως. Στην συγκεκριμένη περίπτωση το χρησιμοποιούμενο *DAG* μας προς αξιολόγηση είναι οι *ODP* ιεραρχίες (βλέπε Πίνακα 1), στις οποίες έχουν προστεθεί με καθαρά συνθετικό τρόπο παραγωγής εκ νέου πολλαπλές *isA* συνδέσεις.



Σχήμα 49: Πλήθος Propagations ανά βήμα προσθήκης (5%) non-spanning tree ακμών

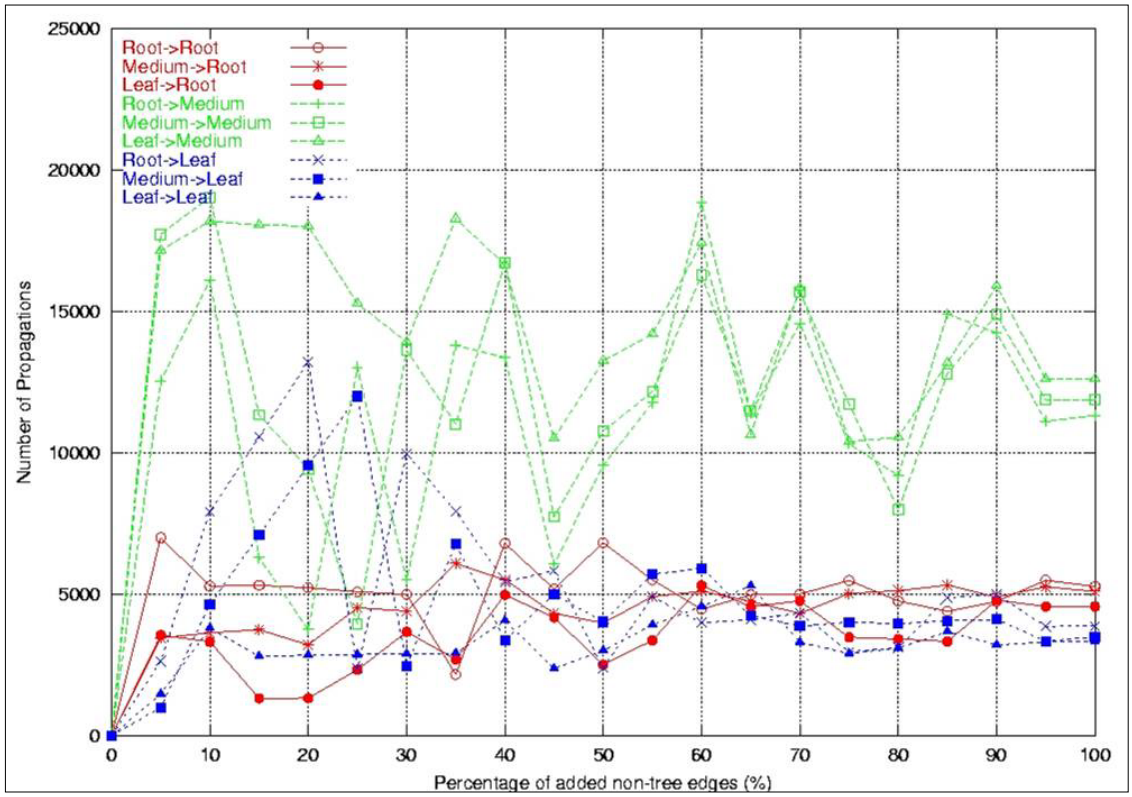


Σχήμα 50 : Πλήθος Συμπιέσεων ανά βήμα προσθήκης (5%) non-spanning tree ακμών

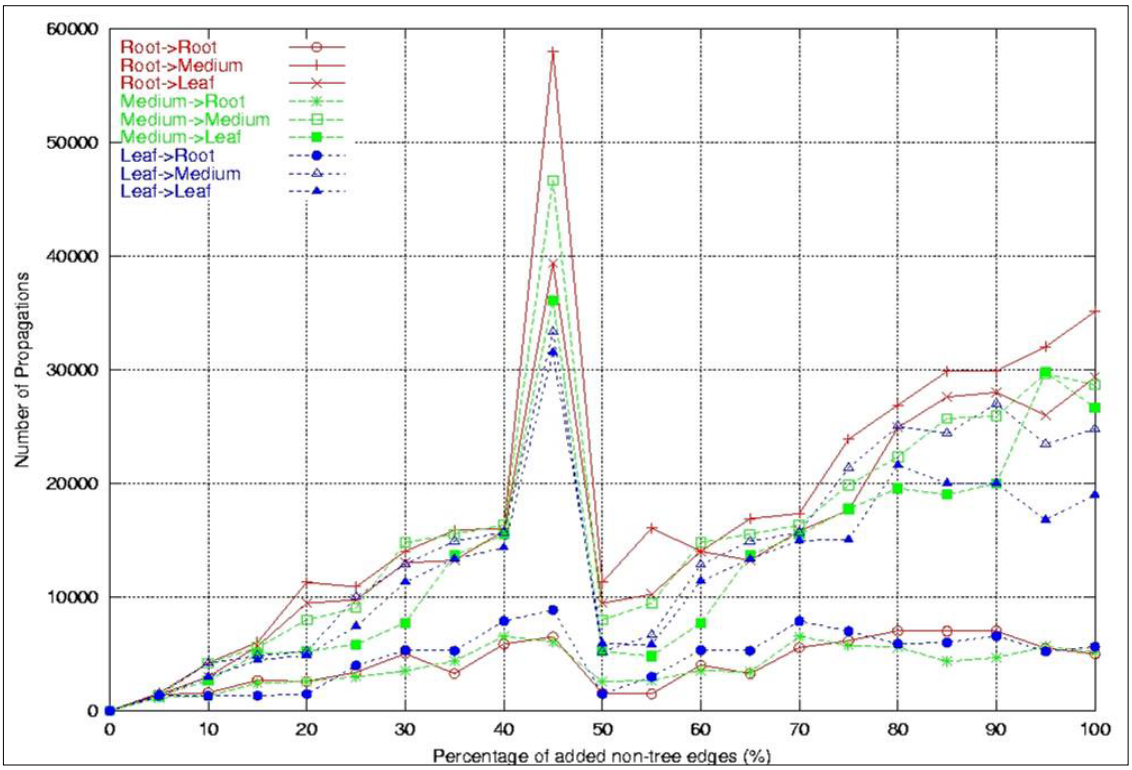
Το συνολικό πλήθος των μεταδόσεων των κωδικών και για τα δυο περιγραφόμενα σχήματα κωδικοποίησης εικονίζεται στο σχήμα 49, κι είναι όπως παρατηρούμε σε άμεση συσχέτιση με το ποσοστό των επιπρόσθετων *non-spanning tree* ακμών που προσθέτουμε στο εκάστοτε υπάρχων γράφο. Η διεξαγωγή του όλου πειράματος, έγινε με σταδιακό, αυξητικό τρόπο, δηλαδή προσθέσαμε σε κάθε ένα βήμα εκτέλεσης ένα 5%, επί του αρχικού πλήθους ακμών του δέντρου, *non-tree* ακμές στον γράφο μας μέχρι να προσεγγίσουμε το ποσοστό των 100% των επιπρόσθετων ακμών, που ισοδυναμεί δηλαδή με τον διπλασιασμό του πλήθους των ακμών του δέντρου του *ODP* καταλόγου αρχικά. Για τον γράφο που δημιουργείται με την πρόσθεση 5% επιπλέον ακμών, επί του πλήθους των ακμών του της ιεραρχίας-δέντρου του *ODP* καταλόγου, εκτελούμε και τους δυο αλγορίθμους κωδικοποίησης. Είναι σαφές κι εκ των τιμών του σχήματος 49 ότι το υφιστάμενο *spanning-tree*, που υπολογίζουμε και για τους δυο αλγορίθμους από κοινού, διαφοροποιείται από βήμα σε βήμα λόγω της προσθήκης των *non-tree* ακμών και της διαμόρφωσης ίσως

βέλτιστων καινούργιων μονοπατιών που θα πρέπει να συμπεριληφθούν και να αναδιαμορφώσουν έτσι το εκάστοτε επιλεχθέν μας *spanning-tree*.

Εκ του σχήματος 50 η πρώτιστη και βασική παρατήρηση που προκύπτει είναι ότι το πλήθος των συμπιέσεων των κωδικών στο σχήμα του *PInterval* είναι ανάλογο του πλήθους των *non-tree* ακμών που προσθέτουμε στο γράφο μας, ασχέτως από την τοπολογική θέση τους μέσα στο γράφο, σε αντίθεση με την αντίστοιχη συμπεριφορά του σχήματος του *UPrefix*. Το συμπέρασμα αυτό προκύπτει κι εκ του σχήματος 49 όπου το πλήθος των μεταδιδόμενων κωδικών στο μεν σχήμα του *PInterval* σταθεροποιείται στο πεδίο τιμών μεταξύ των 50000 και 80000 μεταδόσεων. ενώ στο δε σχήμα του *UPrefix* το πλήθος των μεταδιδόμενων κωδικών του παρατηρείται ότι εξαρτάται άμεσα από τον πραγματικό αριθμό των "απογόνων" κόμβων της κλάσης "αφετηρίας" της κάθε προστιθέμενης ακμής στον γράφο. Επιπρόσθετα, παρατηρούμε πλέον μια σαφέστατη απόκλιση του πλήθους των μεταδόσεων για τα δυο σχήματα που συνεχώς όλο και μεγαλώνει. εφόσον έχει προστεθεί ένα σημαντικό πλήθος από ακμές στον υπάρχων μας γράφο, ποσοστό που ανέρχεται στο 65% επί του αρχικού πλήθους ακμών



Σχήμα 51 : Κατανομή Πλήθους Propagations σχήματος DPInteval ως προς τις κατηγορίες των non-spanning tree ακμών ανά βήμα προσθήκης (5%)



Σχήμα 52 : Κατανομή Πλήθους Propagations σχήματος DUPrefix ως προς τις κατηγορίες των non-spanning tree ακμών ανά βήμα προσθήκης (5%)

Στο σχήμα 51 και 52 βλέπουμε την ακριβή κατανομή των μεταδόσεων (*propagations*) των ετικετών για τις 9 κατηγορίες *isA* ακμών που προαναφέραμε, όταν προσθέσουμε περίπου ίδιο αριθμό *non-spanning tree* ακμές για κάθε μία κατηγορία και για τα δυο σχήματα κωδικοποίησης. Όπως άλλωστε φαίνεται και στο σχήμα 49, αξιοσημείωτη είναι η ξαφνική αύξηση του πλήθους των μεταδόσεων για όλες τις κατηγορίες στην τιμή 45% αύξησης του πλήθους των συνολικών ακμών στο σχήμα *DUPrefix*, η οποία οφείλεται στην συγκεκριμένη μορφολογία του *ODP* καταλόγου. Συγκεκριμένα, διαπιστώθηκε ότι στο βήμα αυτό ότι το μεγαλύτερο ποσοστό των προστιθέμενων ακμών ανήκαν στην υπο-ιεραρχία *Regional-America* στην οποία παρουσιάζεται η μεγαλύτερη τιμή βάθους επιπέδου σε όλον τον *ODP* κατάλογο και το σύνολο των κόμβων της είναι 134721 ποσοστό 89,38% επί του συνόλου των κόμβων της υπο-ιεραρχίας *Regional* και 53,20% επί του συνόλου του *ODP* καταλόγου. Εν συνόλω, πάντως διαφαίνεται ότι το πλήθος των μεταδιδόμενων κωδικών εξαρτάται άμεσα από τον πραγματικό αριθμό των "απογόνων" κόμβων της κλάσης "αφετηρίας" της κάθε προστιθέμενης ακμής στον γράφο, και οι οποίοι από κοινού αυξάνονται συνεχώς με τα επιμέρους βήματα προσθήκης των *non-spanning*

tree ακμών. Από την άλλη πλευρά το πλήθος των μεταδιδόμενων ετικετών στο σχήμα *DPInterval* (βλ. Σχήμα 50) παρά τις όποιες μικρές αυξομειώσεις του ανά βήμα προσθήκης των *non-spanning tree* ακμών, παραμένει εν τέλει σταθερό κινούμενο σε ένα πεδίο τιμών πλήθους, διαφορετικά για κάθε κατηγορία ακμών. Αναλυτικά το πλήθος των μεταδόσεων για το σχήμα *DPInterval* διαμορφώνεται ως εξής : για την κατηγορία RR ακμών είναι [2000,7000], για την RM είναι [3000,18000], για την RL είναι [2000,11000], για την MR είναι [3000,6000], για την MM είναι [3000,20000], για τη ML είναι [1000,12000], για τη LR είναι [1000,6000], για τη LM είναι [10000,19000]και τέλος για την LL είναι [2000,5000]. Συνεπώς παρατηρούμε, ότι το μεγαλύτερο πλήθος των μεταδόσεων προέρχεται κυρίως από τις *non-spanning tree* ακμές με κόμβους – "προορισμούς" που τοπολογικά είναι ενδιάμεσοι (*M*) στην ιεραρχία και εν συνεχεία με κόμβους "προορισμούς" κοντά στα φύλλα (*L*) και έπονται με διαφορά οι κόμβοι κοντά στην ρίζα (*R*). Κι αυτό εξαρτάται από την διαφορά πληθικότητας στους απογόνους, που σαφώς αντιστοιχεί απόλυτα και στο πλήθος των μεταδόσεων (*bottom-up*), που μόλις προαναφέραμε για αυτές τις κατηγορίες.

Αντιστοίχως, τώρα για το σχήμα του *DUPrefix* (βλ. Σχήμα 51) η κατηγοριοποίηση των μεταδόσεων με βάση τις κατηγορίες ακμών έχει ως εξής : για την κατηγορία RR ακμών είναι [1000,7000], για την RM είναι [1000,35000], για την RL είναι [1000,30000], για την MR είναι [1000,7000] για την MM είναι [1000,47000], για τη ML είναι [1000,37000], για τη LR είναι [1000,8000], για τη LM είναι [1000,34000] και τέλος για την LL είναι [1000,32000]. Σαφώς, και με βάση τα πεδία αυτά τιμών των μεταδόσεων ανά κατηγορία ακμών παρατηρούμε ότι το μεγαλύτερο πλήθος των μεταδόσεων προέρχεται κυρίως από τις *non-spanning tree* ακμές με κόμβους – "αφετηρίες" (λαμβάνοντας υπόψη την κατεύθυνση των μεταδόσεων στο σχήμα του *DUPrefix*) που είναι R (συγκεκριμένα οι RM και RL) κι εν συνεχεία ακολουθούν οι ακμές με κόμβους – "αφετηρίες" τύπου M (συγκεκριμένα οι MM και ML). Με σχετική διαφορά έπονται οι μεταδόσεις με κόμβους – "αφετηρίες" L, η οποία όμως μετά το 60-65% αύξησης του πλήθους των ακμών η απόκλιση τους από τις κατηγορίες με κόμβους – "αφετηρίες" R και M αυξάνεται σταδιακά.

Τώρα, επί του συνόλου των δυνατών τρόπων συμπιέσεων θα αγνοήσουμε την όποια δυνατή συνένωση πεδίων – κωδικών με αριθμητικά συνεχόμενα τα άκρα τους, στοχεύοντας με αυτόν τον τρόπο να διατηρήσουμε τους επιμέρους *post-order*

αριθμούς τους ως μοναδικά κι απόλυτα διακριτά αναγνωριστικά των κλάσεών τους, αναφερόμενοι δηλαδή σε επιπρόσθετες 2492 πλειάδες περίπου για τον πίνακα *PInterval* (ποσοστό γύρω στο 4% επί του συνολικού πλήθους πλειάδων). Άρα, από τις δυνατές απορροφήσεις διατηρούμε τις απορροφήσεις αυτές που εμπεριέχουν τον έλεγχο της ύπαρξης υπαλληλίας μεταξύ των διαφόρων πεδίων. Μάλιστα, εάν στο σύνολο αυτών των συνενώσεων θα παραλείπαμε τις πλήρεις συμπιέσεις κι αφήναμε μόνο τις συμπιέσεις αυτές πεδίων με ίσα άκρα μεταξύ τους, τότε θα είχαμε σε πρώτη φάση αύξηση του μέσου όρου κωδικών ανά κόμβο-κλάση. Εν συνεχεία όλη η πληροφορία που θα προσδιορίζει κάθε έναν κόμβο θα εμπεριέχεται στους κωδικούς του κι άρα αναιρείται εκ των πραγμάτων η όποια ανάγκη για πολύπλοκες επερωτήσεις. Αντιθέτως, θα χρειάζονται απλές επερωτήσεις, οι οποίες θα μπορούν να μεταφραστούν επίσης και πολύ εύκολα σε *SQL* μορφή κι οι οποίες να μας δίνουν τους προσδιοριστικούς κωδικούς του κάθε κόμβου. Με βάση όλα αυτά τα προσδιοριστικά γνωρίσματα θα συνθέσουμε το *DAG* αυτό πάνω στο οποίο εν συνεχεία θα χρησιμοποιήσουμε για να αξιολογήσουμε την απόδοση των ζητούμενων επερωτήσεων και για τα δυο σχήματα κωδικοποίησης.

Αναφορικά, επιφέροντας την προσθήκη 253214 επιπλέον ακμών (διαμορφώνοντας έτσι ένα ποσοστό αύξησης γύρω στο 100%) για το σχήμα του μεν *DUPrefix* θα περιέχει 179270 πλειάδες (με 35 συμπιέσεις λόγω απορρόφησης αλφαριθμητικών ετικετών στο 100%), ενώ για το δε σχήμα του *DPInterval* θα περιέχει 61445 πλειάδες, αναφερόμενοι στην περίπτωση που εφαρμόζουμε συμπίεση των πεδίων-ετικετών (περιλαμβάνουμε τις συμπιέσεις μέσω απορρόφησης πεδίων – διαστημάτων, των οποίων το συνολικό τους πλήθος στο 100 % είναι 2185 και στις οποίες συμπεριλαμβάνονται και τις συμπιέσεις λόγω συνένωσης γειτονικών μεταξύ τους ετικετών-πεδίων - των οποίων το πλήθος μετρήθηκε να είναι 379). Εάν τώρα παραλείψουμε την συμπίεση με την συνένωση γειτονικών πεδίων – διαστημάτων και διατηρήσουμε μόνο την συμπίεση λόγω απορρόφησης των πεδίων –διαστημάτων το σχήμα του *DPInterval* θα περιέχει 63937 πλειάδες. Στην περίπτωση που δεν υιοθετήσουμε τη συμπίεση των πεδίων-ετικετών για το σχήμα του *DPInterval* θα έχουμε αντιστοίχως.191812 πλειάδες. Εδώ θα πρέπει να αναφέρουμε πως στην περίπτωση που δεν έχουμε συμπίεση πεδίων –ετικετών, ενδέχεται η ύπαρξη πολλαπλής εμφάνισης ενός ίδιου πεδίου - ετικέτας. Σε αυτήν την περίπτωση θα πρέπει να διαγράψουμε τις διπλοεγγραφές, χωρίς το γεγονός αυτό να συγκαταλέγεται σαν μια επιμέρους συμπίεση ετικετών – πεδίων. Μάλιστα το πλήθος συνολικά των

διαγραφών των διπλοεγγραφών αυτών στην πειραματική μας αξιολόγηση ήταν 2738). Συμπερασματικά, συμπεραίνουμε πως το κόστος στο χώρο αποθήκευσης για το σχήμα του *DPInterval* είναι καλύτερο συγκριτικά από του *DUPrefix* στη περίπτωση που έχουμε συμπίεση ετικετών, καθώς δεδομένου ότι στην περίπτωση του γράφου υπάρχει κοινός σχεδόν διπλασιασμός του μεγέθους του πρωτεύοντος πίνακα, αλλά όμως υπάρχει πολύ μικρότερη επιβάρυνση στη δημιουργία του δευτερεύοντος, καθώς το μέγεθος του *DUPrefix* είναι 2,86 φορές μεγαλύτερο του *DPInterval*. Σε ότι αφορά το σχήμα του *DPInterval* το πλήθος των πλειάδων χωρίς συμπίεση είναι 3,01 φορές μεγαλύτερο από την περίπτωση της ύπαρξης της δυνατότητας συμπίεσης. Συνεπώς, η συμπίεση σαν επιλογή έχει σημαντική μείωση στο συνολικό αποθηκευτικό χώρο του επιλεχθέντος σχήματος αναπαράστασης μας. Επιπρόσθετα, η απόδοση του *Agrawal* με βάση αυτό το κριτήριο με συμπίεση κρίνεται καλύτερη από ότι του *Unicode Dewey*.

5.5 SQL Ερωτήσεις

Θεωρούμε στον πίνακα 13 και για τα δυο εξεταζόμενα σχήματα κωδικοποίησης, μια καθ' όλα δηλωτική (φορμαλιστική) διατύπωση των 5 πλέον βασικών επερωτήσεων που μας ενδιαφέρει να θέσουμε στο υπό εξέταση γράφο μας, χρησιμοποιώντας συμβατή ορολογία με αυτή που χρησιμοποιήσαμε και στην έκφραση των αντίστοιχων επερωτήσεων στην περίπτωση του δέντρου. Θα δείξουμε στη συνέχεια πως αυτός ο φορμαλισμός των βασικών επερωτήσεων έχει γενική ισχύ αναφορικά και για τα δυο σχήματα κωδικοποίησης, είτε επιλέγουμε τη δυνατότητα συμπίεσης των ετικετών είτε όχι. Προφανώς, εν συνεχεία θα υπάρχουν τροποποιήσεις στην μετάφραση του φορμαλισμού των επερωτήσεων των σχημάτων αυτών σε *SQL* μορφή ανάλογα με την επιλογή ή μη της δυνατότητας αυτής συμπίεσης ετικετών.

Επεξηγώντας τους συμβολισμούς του πίνακα 13, να αναφέρουμε πως με τον όρο *Propdown(u)* εννοούμε το σύνολο των απογόνων κόμβων του *u*, όπως καταγράφονται στον πίνακα *DUPrefix*, στους οποίους μεταδίδεται ο αλφαριθμητικός κωδικός του *u*. Ορίζουμε με τον όρο *Propansc(v)* εννοούμε το σύνολο των προγόνων κόμβων του v, έστω *u* στον πίνακα *DUPrefix*, στους οποίους μεταδίδεται ο κωδικός του *u* και όπου ισχύει $v \in$ *Propdown(u)*. Με τον όρο *Propup(u)* στο σχήμα *PInterval* καλούμε το σύνολο των προγονικών κόμβων του, στους οποίους το πεδίο-ετικέτα του

u μεταδίδεται σαν επιπρόσθετος κωδικός και αντιστοίχως *Propdesc(u)* είναι το σύνολο των απογόνων, έστω *u* του κόμβου *v*, για τον οποίο ισχύει u $\in$ *Propup(u)*. Ο έλεγχος σχέσης υπαλληλίας μεταξύ δυο κόμβων του γράφου έστω *u* και *v* κρίνεται ορθός στον πίνακα *DUPrefix* (και αντιστοίχως στον *DPInterval*) εάν και μόνο εάν η συνθήκη *subsumption(u,v)*, η οποία και φαίνεται στον πίνακα 3 (και πίνακα 6 για τον *DPInterval* αντιστοίχως) που αναφέρεται για την περίπτωση των δέντρων κριθεί ορθή ή εναλλακτικά βρεθεί ότι ο κόμβος *u* $\in$ *Propansc(u) (v* $\in$ *Propup(u)* αντιστοίχως για τον πίνακα *DUPrefix*).

Επερώτηση	**Σχήμα *PInterval***	**Σχήμα *DUPrefix***
Ddescendants (v)	*descendants(v)* $\cup \bigcup_{w \in \Pr opdesc(v)}$ *descendants(w)*	*descendants(v)* $\cup$ *Propdown(v)*
Dancestors(v)	*ancestors(v)* $\cup$ *Propup(v)*	*ancestors(v)* $\cup \bigcup_{w \in \Pr opup(v)}$ *ancestors(w)*
Dleaves(u)	*{u | u* $\in$ *Ddescendants(v)* $\wedge$ *Propdesc(u) = 0* $\nexists$ *u' | u'* $\in$ *Ddescendants(u)* $\wedge$ *Propdown(u') = 0}*	{u | u $\in$ *Ddescendants(v)* $\wedge$ *Propdown(u) = 0* $\nexists$ u' | u' $\in$ *Ddescendants(u)* $\wedge$ *Propdown(u') = 0*}
Dsiblings(u)	*siblings(u)*	*siblings(u)*
Dnca(u,v)	*{u | u* $\in$ *Dancestors(v)* $\wedge$ *u* $\in$ *Dancestors(w)* $\wedge$ $\nexists$ *u' | u'* $\in$ *Dancestors(v)* $\wedge$ *u'* $\in$ *Dancestors(w)* $\wedge$ *u* $\in$ *Dancestors(u')}*	*{u | u* $\in$ *Dancestors(v)* $\wedge$ *u* $\in$ *Dancestors(w)* $\wedge$ $\nexists$ *u' | u'* $\in$ *Dancestors(v)* $\wedge$ *u'* $\in$ *Dancestors(w)* $\wedge$ *u* $\in$ *Dancestors(u')}*

Πίνακας 13 : Φορμαλισμός Βασικών Επερωτήσεων για DAGs στα σχήματα κωδικοποίησης a) PInterval και b) UPrefix

Στην συνέχεια θα προχωρήσουμε στη μετάφραση σε μορφή *SQL,* των δηλωτικών εκφράσεων των επερωτήσεων που μόλις είδαμε στον πίνακα 13 και για τα δυο σχήματα κωδικοποίησης, με και χωρίς τη δυνατότητα συμπίεσης ετικετών. Πάντως, εξ αρχής ένα γενικό συμπέρασμα που προκύπτει από την διαδικασία αυτή υλοποίησης των επερωτήσεων είναι ότι η συμπίεση των κωδικών οδηγεί σε πιο σύνθετες εκφράσεις επερωτήσεων, καθώς τα μονοπάτια που ενώνουν δυο κόμβους ενός *DAG*, είναι σχεδόν σίγουρο ότι θα συμπεριλαμβάνουν και *non-spanning tree* ακμές, που σαφώς και δεν μπορούν να υλοποιηθούν και να αποτυπωθούν πλήρως στους πίνακες *DUPrefix* και *DPInterval* αντιστοίχως.

5.5.1 Βασικές Επερωτήσεις

Ας θεωρήσουμε ότι έχουμε έναν κόμβο *v* στο σύνθετο *DAG* μας, ο οποίος με την μεν διαδικασία κωδικοποίησης του αλγορίθμου του *Agrawal et al* (σχήμα *PInterval*) αποκτά έστω την ετικέτα – πεδίο [*i, p*] και με την κωδικοποίηση του αλγορίθμου του Unicode *Dewey* (σχήμα *UPrefix*) έστω ότι αποκτά την αλφαριθμητική ετικέτα '*l*'. Αναφορικά, με τις παραγόμενες αυτές ετικέτες και με τον φορμαλισμό των επιμέρους βασικών επερωτήσεων (βλ. Πίνακας 13) μπορούμε να προχωρήσουμε στην υλοποίησή τους υπό την βέλτιστη, κάτωθι *SQL* μορφή. Παρατηρούμε, πως ο εν γένει ο σχηματισμός των επερωτήσεων και για τα δυο σχήματα κωδικοποίησης βασίζεται στον συνδυασμό του αποτελέσματος της αντίστοιχης επερώτησης που είχαμε χρησιμοποιήσει στο κεφάλαιο 4 για την περίπτωση του δέντρου, αλλά και την εκ νέου διαμόρφωση επιμέρους υποεπερώτησης, η οποία απευθύνεται για τα μονοπάτια του συνθετικά σχηματισμένου γράφου μας, που περιλαμβάνουν τουλάχιστον μία *non-spanning tree* ακμή. Επίσης, η υλοποίηση των βασικών επερωτήσεων αυτών, θα γίνει από κοινού για κάθε μία περίπτωση υιοθετώντας παράλληλα την δυνατότητα συμπίεσης των ετικετών –εφόσον είναι δυνατό – αλλά και εναλλακτικά χωρίς συμπίεση.

Αναφορικά, η σύνδεση των δυο υποεπερωτήσεων - και για τα δυο σχήματα – μέσω της χρήσης του τελεστή *UNION ALL* για την περίπτωση που υφίσταται η συμπίεση των ετικετών. Η επιλογή της χρήσης του *UNION ALL* έναντι του *UNION,* είναι εκ του γεγονότος ότι το *UNION* διακρίνεται από την λειτουργία να εξαλείφει τις όποιες, υπάρχουσες πολλαπλές κοινές εγγραφές, που μπορούν να προκύψουν εξίσου από το *spanning-tree,* αλλά κι από τον υπόλοιπο γράφο (***duplicate elimination***). Το σοβαρό μειονέκτημα της χρήσης του *UNION* έναντι του *UNION ALL,* είναι ότι επιβαρύνει την απόδοση του συνολικού χρόνου εκτέλεσης της ζητούμενης επερώτησης. Συγκεκριμένα, λαμβάνοντας υπόψη τον γενικό σχεδιασμό της σχεσιακής μας αναπαράστασης, σε κάθε περίπτωση οι επιμέρους επερωτήσεις (που απευθύνονται και στους δυο πίνακες αναπαράστασης για το κάθε σχήμα αντίστοιχα) αποτελούν επιμέρους συνδέσεις βασικών υποεπερωτήσεων, οι οποίες σε κάθε περίπτωση επιστρέφουν μη κοινές πλειάδες κι άρα δεν υφίσταται θέμα ανάγκης απαλοιφής ομοίων πλειάδων με την χρήση του *UNION ALL*. Στην περίπτωση όμως

τη μη συμπίεσης δεν υπάρχει η διασφάλιση της διακριτότητας των επιστρεφόμενων πλειάδων, οπότε τότε ενδείκνυται η χρήση του *UNION*.

Αναλυτικά, θα έχουμε για κάθε επιμέρους βασική επερώτηση και για τα δυο σχήματα με αλλά και χωρίς τη συμπίεση των ετικετών:

Α. Απόγονοι – (Descendants)

Η επερώτηση *Ddescendants(v)* όπως βλέπουμε και στον πίνακα 13 μέσω του φορμαλισμού διατύπωσής της χρησιμοποιεί στο $1^{ο}$ μέρος της την πρώτη έκφραση επερώτησης των απογόνων, που χρησιμοποιήσαμε άλλωστε και στην περίπτωση του δέντρου – *descendants(v)*. Επιπρόσθετα και για τα δυο εξεταζόμενα σχήματα κωδικοποίησης υπολογίζει και τους απογόνους αυτούς που αποκτά μέσω της μετάδοσης (*propagation*) των ετικετών του κόμβου *v*, μέσω των φορμαλισμένων υποεπερωτήσεων *Propdown(v)* και *Propdesc(v)* αντιστοίχως.

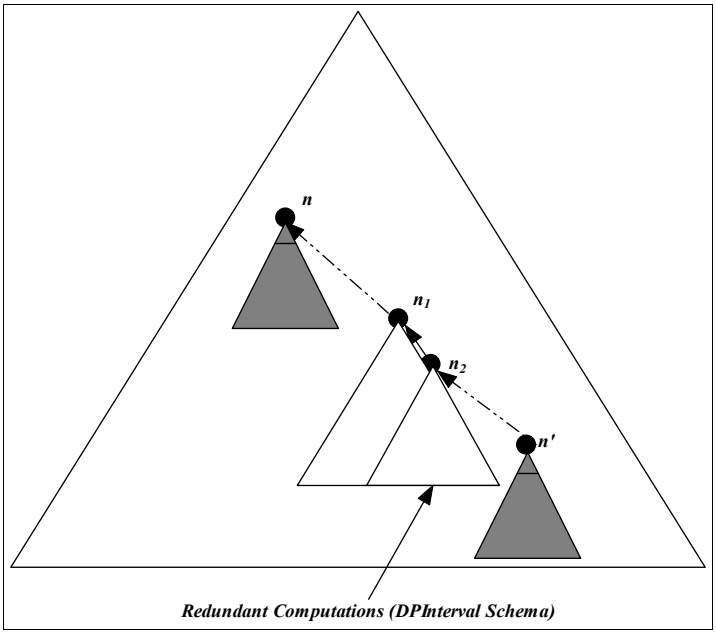
Στην περίπτωση όμως, όπως θα δούμε με λεπτομέρεια και στη συνέχεια, που υιοθετήσουμε την μη χρήση

1. Agrawal Encoding

Η έκφραση *Propdesc(v)* αναφέρεται στις μεταδόσεις των ετικετών από τις *non-spanning tree* ακμές για το σχήμα *DPInterval*. Για την περίπτωση που έχουμε συμπιέσεις ετικετών-πεδίων η συνάρτηση *Propdesc(v)* μεταφράζεται σε *SQL* με το κομμάτι της επερώτησης που ακολουθεί μετά το *UNION ALL*. Αναλυτικά, διατρέχει τον πίνακα *DPInterval* που εμπεριέχει τις πλειάδες αυτές με πεδία-ετικέτες που έχουν μεταδοθεί (*propagated*), λόγω της προσθήκης μίας τουλάχιστον *non-tree* ακμής στον κόμβο *v*, ξεκινώντας από αυτόν (κάτι που ελέγχεται με την συνθήκη v.ancestor = *p*). Για το σύνολο των κόμβων αυτών *v*, από τους οποίους μεταδίδεται το πρωτεύων πεδίο του κόμβου *u*, εν συνεχεία αναζητούμε επίσης τους απογόνους του. Στην περίπτωση που εφαρμόζουμε την συμπίεση πεδίων – ετικετών, οι συγκρίσεις υπαλληλίας μεταξύ των πεδίων – ετικετών επιτυγχάνουν την μη πλήρη καταγραφή όλων των επιμέρους κόμβων που συναποτελούν όλα τα μεταβατικά κλεισίματα που δημιουργούνται στον γράφο.

```
select u.post
from PInterval u
where i <= u.post and u.post < p)
UNION ALL
(select u'.post
 from PInterval u', DPInterval v
where v.ancestor = p and u'.post <= v.post and u'.post >= v.index)
```

Στην περίπτωση που δεν υιοθετούμε την δυνατότητα συμπίεσης πεδίων –ετικετών η συνάρτηση *Propdesc(v)* θα έπρεπε με μια πρώτη σκέψη να τροποποιείται και συγκεκριμένα να απλοποιείται. Συγκεκριμένα, η μόνη πλέον δυνατή ‘‘συμπίεση’’ είναι αυτή της ύπαρξης απολύτως ομοίων ετικετών, οι οποίες από κοινού προσφέρουν την ίδια πληροφορία, κι άρα μπορούμε να τις διαγράψουμε πλην μίας βέβαια. Συνεπώς, η σύνδεση των δυο επιμέρους επερωτήσεων χρειάζεται να γίνει με το UNION για την διαγραφή διπλοεγγραφών που προκύπτουν από spanning-tree και λοιπού γράφου. Κι αυτό γιατί στην περίπτωση αυτή θα είναι πλήρως υλοποιημένα τα μεταβατικά κλεισίματα, μέσω των οποίων θα αναπαρασταθούν τα μονοπάτια σύνδεσης μεταξύ όλων των κόμβων μεταξύ τους. Και πάλι όμως χρειάζεται να αναζητήσουμε τους απογόνους των κόμβων οι οποίοι συνδέονται με τον τρέχων κόμβο (πατρικοί του), έκαστος μέσω μιας *non-tree* ακμής. Συνεπώς, η συνθήκη διερεύνησης αυτών των απογόνων στο δεύτερο μέρος της συνολικής επερώτησης δεν τροποποιείται. Ενδέχεται με την εκτέλεση της όμως να επιτελεί άσκοπους υπολογισμούς (***redundant computations***), που δεν ανταποκρίνονται στο πλήθος των επιστρεφόμενων πλειάδων και έτσι να ανεβάζει το συνολικό χρόνο εκτέλεσης της επερώτησης. (βλ. Σχήμα 53).



Σχήμα 53 : Παράδειγμα εκτέλεσης redundant computations με την προσθήκη της non-tree ακμής n_2 n_1 (περίπτωση μη ύπαρξη της δυνατότητας συμπίεσης των πεδίων –ετικετών)

Άρα η επερώτηση στην περίπτωση μη συμπίεσης των πεδίων ετικετών γράφεται ως εξής :

```
select u.post
from PInterval u
where i <= u.post and u.post < p)
UNION
(select u'.post
 from PInterval u', DPInterval v
where v.ancestor = p and u'.post <= v.post and u'.post >= v.index)
```

Εάν χρησιμοποιήσουμε το παράδειγμα του γράφου του σχήματος 24. Εν συνεχεία παραθέτουμε τους πίνακες *PInterval* και *DPInterval* με τη χρήση συμπίεσης ετικετών και μη, που δημιουργούνται (Σχήμα 54).

PInterval

post	index	father_post
9	1	-
2	1	8
8	3	9
1	1	2
6	3	8
7	7	8
5	3	6
3	3	5
4	4	5

***DPInterval* (with compression)**

post	index	ancestor
3	3	1
5	3	7
5	3	2
5	3	8

***DPInterval* (no compression)**

post	index	ancestor
2	1	9
3	3	9
5	3	9
3	3	2
5	3	2
3	3	8
5	3	8
3	3	1
5	3	7

Σχήμα 54 : Πίνακες α) PInterval (μόνο spanning-tree), β) DPInterval (με συμπίεση ετικετών) και γ) DPInterval (χωρίς συμπίεση ετικετών) για το γράφο του σχήματος 24.

Έστω ότι θέλουμε να υπολογίσουμε τους απογόνους του κόμβου B. Παρατηρούμε ότι όλα τα μονοπάτια διασύνδεσης μέσω των ακμών και μόνο του *spanning-tree* βρίσκονται στον πρωτεύων πίνακα, τον *PInterval*. Άρα οι απόγονοι του κόμβου B δεν είναι παρά αυτοί των οποίων τα πεδία με βάση την παραπάνω συνθήκη ($1^{ο}$ μέρος της) εμπεριέχονται σαν διαστήματα στο πεδίο-ετικέτα του, που αντιπροσωπεύουν τον κόμβο D. Με βάση το $2^{ο}$ μέρος προκύπτει, με βάση τον *DPInterval,* ότι ο κόμβος G έχει μέσω *non-spanning tree* ακμής τον B πρόγονο, κι άρα αναδρομικά συνυπολογίζονται μέσω του G και οι δικοί του απόγονοι κ.ο.κ.(συνθήκη $2^{ου}$ μέρους επερώτησης). Στην περίπτωση χρήσης της συμπίεσης πεδίων-ετικετών παρατηρούμε ότι για τα μονοπάτια που έχουν τουλάχιστον μία *non-spanning tree* ακμή μπορεί να μην είναι πλήρως υλοποιημένα – καθώς μπορεί να παραλείπονται επιμέρους ακμές τους οι οποίες συμπιέζονται. Στο παράδειγμα μας (βλ. Σχήμα 24), ο κόμβος H είναι απόγονος του B τόσο μέσω του μονοπατιού που ορίζουν οι ακμές H→G (*tree* ακμή) + G→B (*non-tree* ακμή), όσο και μέσω του μονοπατιού που ορίζουν οι ακμές H→D (*non - tree* ακμή) + D→B (*tree* ακμή). Παρατηρούμε πως με την συμπίεση η μετάδοση της ετικέτας από τον H στον D απορροφάται (η μορφή της πλειάδας είναι [5 3 1] και δεν περιλαμβάνεται στον πίνακα *DPInterval*), ενώ χωρίς την συμπίεση υπολογίζεται δυο φορές. Συνεπώς, για την σωστή απόδοση της επερώτησης χωρίς συμπίεση (σωστού πλήθους πλειάδων), επιβάλλεται η χρήση του *UNION*.

2. Unicode Dewey Encoding

Εάν προχωρήσουμε στην υιοθέτηση της συμπίεσης των ετικετών, η εν λόγω *SQL* επερώτηση περιλαμβάνει επίσης και μια «φωλιασμένη» επερώτηση στον πίνακα *UPrefix*, ώστε να εντοπίσει τους απογόνους που προκύπτουν από τις ακμές *DAG*. Χρησιμοποιούμε κι εδώ αντί της κλήσης της συνάρτησης *next()*, την καλύτερη από άποψη πλήθους συγκρίσεων αλφαριθμητική πράξη της συνένωσης του *'l'* με τον Unicode αλφαριθμητικό χαρακτήρα '*xFF'*,που αποτελεί τον ιδεατό (*virtual*), πιο δεξιό παιδί κι συνάμα το επιθυμητό, ανώτατο όριο του κόμβου με *label = 'l',*του οποίου ο άμεσα επόμενος δεξιός κόμβος έχει κωδικό τον *next(label)* και τον οποίο μπορούμε να χρησιμοποιήσουμε άντ' αυτού. Επίσης εν συνεχεία, στο $2^{ο}$ μέρος, αναζητούμε όλους τους απογόνους του κόμβου με *label = 'l'*, οι οποίοι αντιστοιχούν στις πλειάδες του *DUPrefix*, οι οποίες στο πεδίο *ancestor*, τους έχουν ως επιμέρους πατρικούς κόμβους. Πρόκειται δηλαδή για όλους τους κόμβους οι οποίοι μέσω μιας προσθήκης μιας *non-tree* ακμής έχουν συμπεριληφθεί σε όλα τα καινούργια μονοπάτια που τους καθιστούν απογόνους του κόμβου "αφετηρία" της ακμής, και στους οποίους έχει μεταδοθεί η αλφαριθμητική ετικέτα του κόμβου "προορισμού" της.

```
(select u.label
from UPrefix u
where u.label > 'l' and u.label < 'l'||'xFF')
UNION ALL
(select v.label
from DUPrefix v, (select u.label as t from UPrefix u where ((u.label >= 'l') and
(u.label <'l'||'xFF'))) u'
where v.ancestor = u'.t)
```

Εναλλακτική μορφή απόδοσης της παραπάνω επερώτησης θα μπορούσε να είναι κι κάτωθι, η οποία όπως βλέπουμε διαφοροποιείται ως προς το δεύτερο μέρος της και συγκεκριμένα στην φωλιασμένη επερώτηση πάνω στον πρωτεύοντα πίνακα *UPrefix*. Θα επιλέξουμε την πρώτη για λόγους απόδοσης που θα αναφέρουμε στη συνέχεια στο υποενότητα της Πειραματικής Αξιολόγησης.

```
(select u.label
from UPrefix u
```

```
where u.label > 'l' and u.label < 'l'||'xFF')
UNION ALL
(select v.label
from DUPrefix v,
where v.ancestor in (     select X.label
                          from UPrefix X
                          where ((X.label >='l') and (X.att0 <'1'||'xFF')))
```

Στην περίπτωση που είχαμε παραλείψει την συμπίεση ετικετών, η προαναφερθείσα συνάρτηση *Propdown(v)*, που αντιστοιχεί στο δεύτερο μέρος της παραπάνω *SQL* μετάφρασης της εν λόγω, ζητούμενης επερώτησης, θα μπορούσε να αντικατασταθεί με την ακόλουθη επιμέρους πιο απλή *SQL* επερώτηση, όπου *'l'* θεωρούμε την αλφαριθμητική ετικέτα του κόμβου *v*.

```
(select u.label
from UPrefix u
where u.label > 'l' and u.label < 'l'||'xFF')
UNION
(select v.label
from DUPrefix v
where v.ancestor = 'l'
)
```

Στην μορφή της αυτή η επερώτηση χαρακτηρίζεται από την μικρή πολυπλοκότητα της υπο-επερώτησης *Propdown(v)*. Δεν είναι όμως σίγουρος ο μικρός χρόνος εκτέλεσής της, κι αυτό λόγω της αναγκαίας χρήσης του *UNION*, το οποίο χαρακτηρίζεται από την επιβάρυνση διαγραφής των όποιων πολλαπλών κοινών εγγραφών.

Β. Πρόγονοι – (Ancestors)

Ομοίως, η επερώτηση για τους απογόνους, η συνάρτηση *Dancestors(v)* (βλ. πίνακα 13) μέσω της φορμαλιστικής διατύπωσής της, χρησιμοποιεί κατ' αρχήν την έκφραση επερώτησης των προγόνων που χρησιμοποιήσαμε και στην περίπτωση του δέντρου – συνάρτηση *ancestors(v)*. Επιπρόσθετα, αντιστοίχως και για τα δυο

επιμέρους σχήματα κωδικοποίησης, ακολουθεί το 2ο μέρος της επερώτησης που υπολογίζει τους προγόνους αυτούς που αποκτά με την προσθήκη *non-spanning tree* ακμών, δηλαδή μέσω των εγγραφών που προκύπτουν για παράδειγμα από την μετάδοση (*propagation*) των ετικετών του κόμβου *v*, όπως αναπαρίσταται μέσω των φορμαλισμού απόδοσης των επερωτήσεων *Propdown(v)* και *Propdesc(v)* αντιστοίχως των σχημάτων *DPInterval* και *DUPrefix*.

1. Agrawal Encoding

Η έκφραση *Propup(v)* αναφέρεται στις μεταδόσεις των ετικετών από τις *non-spanning tree* ακμές για το σχήμα *DPInterval* κι αντιστοιχεί στο 2ο μέρος της συνολικής επερώτησης μετά το *UNION ALL*. Αναλυτικά, αναζητά στον *DPInterval* όλους τους προγόνους του κόμβου *u*– κόμβοι στους οποίους έχει μεταδοθεί το πεδίο –ετικέτα [*i,p*] – πρόγονοι του κόμβου *u* μέσω των ακμών του γράφου, κι οι οποίοι προκύπτουν από την προσθήκη των *non-tree* ακμών στην διαμόρφωση του γράφου.

Ας θεωρήσουμε το παράδειγμα του γράφου που χρησιμοποιήσαμε και για την περίπτωση προηγουμένως των απογόνων, έστω ότι αναζητούμε τους προγόνους του κόμβου H. Με βάση το *spanning-tree* και το 1ο μέρος της παραπάνω επερώτησης βρίσκουμε τους κόμβους G, E, C και A. Κατόπιν, με βάση το 2ο μέρος προκύπτει από τον *DPInterval,* ότι ο κόμβος H έχει μέσω *non-spanning tree* ακμής τον D πρόγονό του, κι άρα αναδρομικά συνυπολογίζονται οι πρόγονοι του D (βλ. Σχήμα 53). Με χρήση της συμπίεσης θα επιστραφεί μόνο ο κόμβος B και ο F. Στη αντίθετη περίπτωση, άνευ της συμπίεσης, θα υπολογιστούν πέραν των κόμβων B και F επίσης και οι C και A, οι οποίοι όμως έχουν ήδη προϋπολογιστεί από το 1ος μέρος της συνολικής επερώτησης. Για αυτό τον λόγω κι η μετάφραση της επερώτησης χωρίς τη συμπίεση περιλαμβάνει το *UNION*. Άρα η συμπίεση βλέπουμε ότι παραλείπει την *non-spanning tree* ακμή B→A και άρα είναι μη πλήρως υλοποιήσιμα όλα τα υπάρχοντα μονοπάτια που χρησιμοποιούν την εν λόγω ακμή. Όμως η συνθήκη της υποεπερώτησης στο 2ο μέρος, με την σύνθεση των πινάκων *PInterval* και *DPInterval* κατορθώνει, βέβαια με το κόστος της αυξημένης πολυπλοκότητάς της, να αναπληρώσει το παραλειπόμενο αυτό μονοπάτι. Το πεδίο-κωδικός του κόμβου B προφανώς περιέχεται στα όρια του κόμβου C και αναδρομικά στον κόμβο A κι άρα η

πληροφορία της *non-tree* ακμής B→A υπάρχει μέσω του μονοπατιού του *spanning-tree* B→C→A.

Εφαρμόζοντας την δυνατότητα συμπίεσης πεδίων – ετικετών η εν λόγω *SQL* επερώτηση εκφράζεται ως εξής :

```
select u.post
from PInterval u
where (u.index <= p and u.post > p)
UNION ALL
(select u.post
from PInterval u, DPInterval v , (select *
                                  from PInterval u''
                                  where u''.index <= p and u''.post >= p) u'
where u.index <= v.ancestor and u.post >= v.ancestor and u'.post = v.post)
)
```

Εναλλακτικά, τώρα στην περίπτωση που δεν υιοθετούμε την συμπίεση των πεδίων – ετικετών, η επερώτηση τροποποιείται (απλοποιείται) ως προς το 2ο μέρος της ως εξής :

```
select u.post
from PInterval u
where (u.index <= p and u.post > p)
UNION
(select u.post
from DPInterval v, (      select *
                          from PInterval u''
                          where u''.index <= p and u''.post >= p
                    ) u'
where u'.post = v.ancestor)
)
.
```

2. Unicode Dewey Encoding

Στην περίπτωση του σχήματος του *DUPrefix* η επερώτηση *Dancestors(v)* μεταφράζεται με την μορφή μιας ενσωματωμένης συνάρτησης, η οποία με την σειρά της χρησιμοποιεί κατάλληλα μια ενδιάμεση *SQL* επερώτηση, για τον υπολογισμό της ζητούμενης έκφρασης *Propansc(v)*, την οποία παρουσιάζουμε στη συνέχεια :

select *father_label* D*UPrefix* where *label* = '*l*'

Εν συνεχεία, παραθέτουμε στο σχήμα 55 σε μορφή ψευδοκώδικα την ενσωματωμένη αυτή συνάρτηση η οποία υπολογίζει την έκφραση *Propansc(v)*. Θεωρούμε και χρησιμοποιούμε την συνάρτηση *Prefixes* η οποία μας δίνει το σύνολο των προθεμάτων του αλφαριθμητικού *'l'*:

```
Algorithm Dancestors()
Parameters:
(3)  Label 'l'
Procedure Description:
     (1)  List Labels = {'l'}
     (2)  List Prefs = {}
     (3)  Labels += (select label
                     from DUPrefix
                     where father_label = 'l')
     (4)  while (Labels.hasNext) Prefs += Prefixes(Labels.Next)
     (5)  Return Prefs
```

Σχήμα 55 : Αλγόριθμος Dancestors για το σχήμα Unicode Dewey – με τη μορφή ψευδοκώδικα

C. Φύλλα – (Leaves)

Η επερώτηση για τα "φύλλα"- κόμβους – *Dleaves(v)* έχει την ίδια λογική, ως προς το 1^{ο} μέρος της, με την αντίστοιχη επερώτηση για την περίπτωση του δέντρου και υπάρχει επίσης άμεση αντιστοιχία της εν τέλει *SQL* υλοποίησης και για τα δυο σχήματα κωδικοποίησης, τα *DUPrefix* και *DPInterval* αντίστοιχα. Αναλυτικά, για την διαμόρφωση της εν λόγω επερώτησης χρησιμοποιείται η μετάφραση σε *SQL* της επερώτησης για τους απογόνους - *Ddescendants(v)* που προαναφέραμε για την περίπτωση του γράφου.

1. Agrawal Encoding

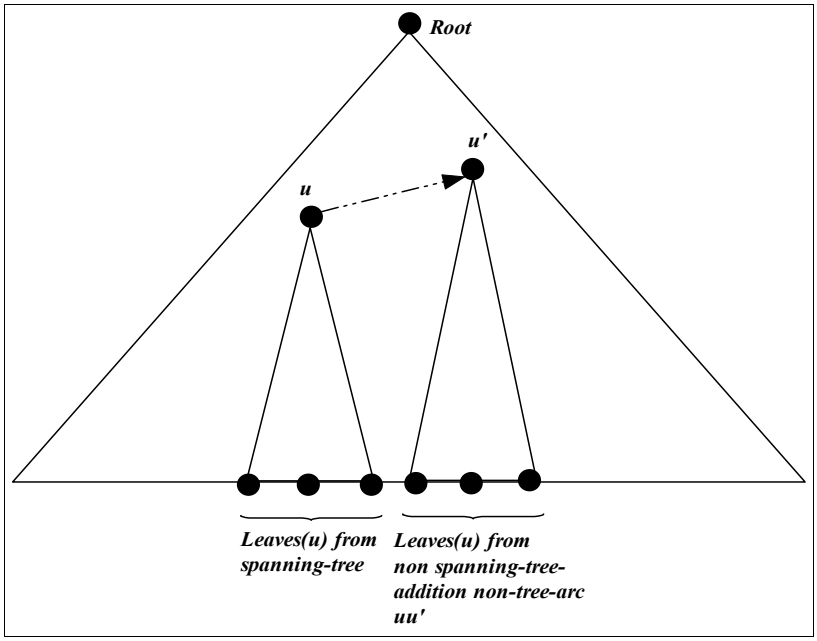
Στην περίπτωση που έχουμε συμπίεση των πεδίων-ετικετών, η επερώτηση μεταφράζεται σε μορφή *SQL* ως ακολούθως:

```
(select u.post
from PInterval u
where u.post < p and u.post >= i and u.post = u.index and
NOT EXISTS      (select *
                 from DPInterval v
                 where v.ancestor = u.post
                 )
)
UNION ALL
```

```
(select u.post
from PInterval u, DPInterval u''
where u''.father = p and u.post >= u''.index and u.post <= u''.post
and NOT EXISTS (select *
                from DPInterval w
                where w.ancestor =u.post)
)
```

Αναλυτικά, το πρώτο μέρος (προ του *UNION ALL*), όπως και στην περίπτωση του δέντρου, διερευνά όλους τους απογόνους του κόμβου *u*, οι οποίοι είναι "φύλλα"- κόμβοι, με την επιπλέον συνθήκη *post = index*. Το δεύτερο μέρος αναζητά τα "φύλλα" - κόμβους των απογόνων κόμβων του κόμβου επίσης *u*, που προκύπτουν από τη προσθήκη των *non-spanning tree* ακμών στον τρέχων γράφο μας. Υπάρχει προφανώς σαφή αντιστοιχία με την επερώτηση των απογόνων ενός κόμβου, οι οποίοι προκύπτουν εν γένει από τη προσθήκη *non-tree* ακμών, μαζί με την επιπρόσθετη συνθήκη ότι οι κόμβοι αυτοί είναι "φύλλα" – η διερεύνηση αυτή είναι με τον έλεγχο ότι δεν υπάρχει άλλη εγγραφή στον *DPInterval* (***NOT EXISTS***), που να έχει το αναγνωριστικό αυτών των κόμβων (*post*) ως τιμή του πατρικού πεδίου (*ancestor*) σε κάποια εγγραφή άλλου κόμβου (βλ. Σχήμα 56).

Όπως και στην περίπτωση της επερώτησης των απογόνων, έτσι κι εδώ στην περίπτωση της μη χρήσης της δυνατότητας συμπίεσης των πεδίων – ετικετών χρειάζεται και πάλι να αναζητήσουμε –εάν υπάρχουν βέβαια - τους "φύλλα"-κόμβους, οι οποίοι προκύπτουν από κάθε τον τρέχων κόμβο - πρόγονο, που προκύπτει μέσω της προσθήκης μιας *non-tree* ακμής. Συνεπώς, το δεύτερο μέρος της συνολικής επερώτησης τροποποιείται με βάση την λογική τροποποίησης κι επίσης ενδέχεται να επιτελεί άσκοπους υπολογισμούς (***redundant computations***), που δεν ανταποκρίνονται στο πλήθος των επιστρεφόμενων πλειάδων και στον χρόνο εκτέλεσης τους.



Σχήμα 56 : Παράδειγμα "Φύλλων"- κόμβων του κόμβου u, που έστω συνδέεται με μια non-tree ακμή με τον κόμβο u'

Η επερώτηση στης περίπτωση μη συμπίεσης των πεδίων –ετικετών γράφεται κατά αντιστοιχία με την περίπτωση των απογόνων ως εξής :

```
(select u.post
from PInterval u
where u.post < p and u.post >= i and u.post = u.index and
NOT EXISTS      (select *
                 from DPInterval v
                 where v.ancestor = u.post
                 )
)
UNION
(select u.post
from PInterval u, DPInterval u''
where u''.father = p and u.post >= u''.index and u.post <= u''.post
and NOT EXISTS (select *
                 from DPInterval w
                 where w.ancestor =u.post)
)
```

Στο παράδειγμα του γράφου αναζητώντας τα "φύλλα" - κόμβους του κόμβου B παίρνουμε από το *spanning-tree* τον κόμβο D ($1^{ο}$ μέρος όλης επερώτησης), ενώ από τον υπόλοιπο γράφο παίρνουμε τους κόμβους H και I (απόγονοι κόμβοι μέσω *spanning – tree* του κόμβου G που συνδέεται με *non-tree* ακμή με τον B) – συνθήκη $2^{ου}$ μέρους προ του τελευταίου ***NOT EXISTS***. Όμως παρατηρούμε πως υπάρχει μια *non-tree* ακμή η HD που καθιστά πλέον του εκ του *spanning-tree* "φύλλου" – κόμβου D του B, μη "φύλλο"-κόμβο. Το συμπέρασμα αυτό προκύπτει από την συνθήκη αυτή του $2^{ου}$ ***NOT EXISTS*** (το $2^{ο}$ μέρος της όλης επερώτησης σχετίζεται με τα μονοπάτια πλην του *spanning-tree* του γράφου). Συνεπώς, τελικά οι ζητούμενοι "φύλλα"- κόμβοι είναι οι H και I.

2. Unicode Dewey Encoding

Ακολουθώντας την αντίστοιχη λογική με αυτήν της μόλις αναλυθείσας επερώτησης για το σχήμα του *DPInterval* και χρησιμοποιώντας την επερώτηση του σχήματος για την αναζήτηση των απογόνων του τρέχοντος κόμβου με ετικέτα *'l'*, χρησιμοποιούμε κι εδώ αντί της κλήσης της συνάρτησης *next()*, την αλφαριθμητική πράξη της συνένωσης του *'l'* με τον *Unicode* αλφαριθμητικό χαρακτήρα *'xFF'*, που όπως αναλύσαμε προηγουμένως διακρίνεται από την βέλτιστη απόδοση της και συνεπώς στο πλήθος των διενεργούμενων συγκρίσεων

```
(select u.label
from UPrefix u
where u.label > 'l' and u.label < 'l'||'xFF' and
NOT EXISTS      (select *
                 from UPrefix v
                 where v.label > u.label and v.label < u.label||'xFF')
and NOT EXISTS (select *
                from DUPrefix w
                where w.ancestor = u.label))
UNION ALL
(select w.label
```

```
from DUPrefix w, (select p.label as t from UPrefix p where p.label >= 'l' and
p.label < 'l'||'xFF') v
where w. ancestor = v.t and
NOT EXISTS (    select *
                from UPrefix s
                where s.label > w.label and s.label < w.label||'xFF')
and NOT EXISTS (select *
                from DUPrefix u''
                where u''. ancestor = w.label)
)
```

Τώρα μη χρησιμοποιώντας την δυνατότητα συμπίεσης των ετικετών η παραπάνω επερώτηση τροποποιείται (απλοποιείται), όπως και στην περίπτωση της επερώτησης για τους απογόνους του *DUPrefix*, όπου είχαμε παρατηρήσει ότι η συνάρτηση *Propdown* απλοποιείται καθώς ελέγχει εγγραφές ετικετών κόμβων, που έχουν ως πατρικό τους κόμβο τον κόμβο αυτό με την ετικέτα '*l*', δηλαδή το σύνολο όλων των απογόνων κόμβων – καθώς δεν υπάρχει η δυνατότητα συμπίεσης ώστε να παραλείπονται /απορροφούνται ενδεχόμενα κάποιοι από αυτούς - του κόμβου αυτού με ετικέτα '*l*', στους οποίους και μεταδίδεται η ετικέτα '*l*'. Θα πρέπει το σύνολο αυτών των κόμβων επιπρόσθετα να μην έχουν με τη σειρά τους άλλους απογόνους (βασική συνθήκη ελέγχου ότι είναι "φύλλα" - κόμβοι). Συνεπώς, με βάση όλα αυτά τα συμπεράσματα ο φορμαλισμός της εν λόγω επερώτησης μεταφράζεται ως εξής :

```
(select u.label
from UPrefix u
where u.label > 'l' and u.label < 'l'||'xFF' and
NOT EXISTS      (select *
                from UPrefix v
                where v.label > u.label and v.label < u.label||'xFF')
and NOT EXISTS (select *
                from DUPrefix w
                where w.ancestor = u.label))
UNION
```

```
(select w.label
from DUPrefix w,
where w. ancestor = 'l' and
NOT EXISTS (    select *
                from UPrefix s
                where s.label > w.label and s.label < w.label||'xFF')
and NOT EXISTS (select *
from DUPrefix u''
where u''. ancestor = w.label)
)
```

F. Ομοεπίπεδοι – (Siblings)

Η επερώτηση για τους ομο-επίπεδους κόμβους (*siblings*) – *Dsiblings(v)* (βλ. Πίνακα 13) έχει και για τα δυο σχήματα κωδικοποίησης την ίδια έκφραση όπως και στην περίπτωση των δέντρων. Συγκεκριμένα, η συνάρτηση *siblings* του πίνακα 13 μεταφραζόμενη σε *SQL* μορφή, αναζητά όλους τους κόμβους εκείνους, με κοινό πατρικό κόμβο μεταξύ τους αλλά και διαφορετική τιμή του αναγνωριστικού πεδίου *post* του κόμβου *u*. Η επερώτηση, δεδομένου ότι είναι κοινή –σαν λογική - και για τα δυο σχήματα κωδικοποίησης, δεν διαφοροποιείται τόσο στην περίπτωση ύπαρξης της δυνατότητας συμπίεσης των ετικετών όσο και στην αντίθετη περίπτωση. Κι αυτό είναι λογικό, καθώς η συνθήκη εμπλέκει μόνο τον πρωτεύων, βασικό πίνακα και τα πεδία του *PInterval* (*UPrefix* αντιστοίχως), κι άρα η επιλογή της χρήσης της δυνατότητας συμπίεσης ή μη των ετικετών, όταν αυτό είναι δυνατό, δεν έχει κατά συνέπεια καμία μεταβολή στο συνολικό χρονικό κόστος εκτέλεσης της επερώτησης.

1. Agrawal Encoding

```
select u.post
from PInterval u, PInterval v
where u.post != p and u.father_post = v.father_post and v.post= p
```

2. Unicode Dewey Encoding

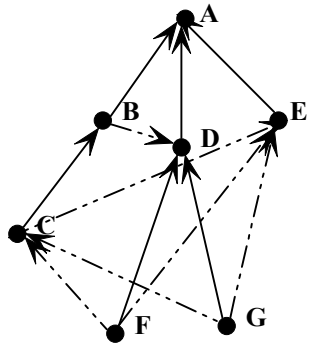
```
select u.label
from UPrefix u, UPrefix v
where (u.label != 'l') and (v.father_label = u.father_label) and (v.label = 'l')
```

Με βάση το παράδειγμα του γράφου παρατηρούμε πως ομοεπίπεδοι κόμβο του B είναι εκ του *spanning-tree* οι κόμβοι E και F (κοινός πατέρας ο C). Παρατηρούμε ότι δεν εξετάζουμε τους ομοεπίπεδους κόμβος του B μέσω non-tree ακμών, που είναι ο C (κοινός πατέρας ο A), καθώς ο C είναι μέσω του spanning-tree άμεσος πατρικός του κόμβος.

G. Κοντινότερος κοινός Πρόγονος – (*nca)*

1. Agrawal Encoding

Στην εξεταζόμενη περίπτωση του *DAG* είναι πιθανή η ύπαρξη πέρα του ενός *nca* που προκύπτει από το υιοθετηθέν *spanning-tree* (μελετηθείσα περίπτωση δέντρου) η ύπαρξη και πλήθους άλλων. Παραθέτουμε ένα παράδειγμα γράφου(βλ. Σχήμα 57), ο οποίος παρατηρούμε έχει για τους κόμβους *F* και *G* φαινόμενο ύπαρξης πολλών *nca* (αναλυτικά είναι ο κόμβος *D* μέσω του *spanning-tree* και οι κόμβοι *C* και *E*, μέσω των *non-spanning tree* ακμών) ομοίως και για τα δυο σχήματα κωδικοποίησης που εξετάζουμε.



Node	Agrawal et al	Unicode Dewey
A	[1,7]	'1'
B	[1,2]	'11','12'
C	[1,1],[3,3],[4,4]	'111','13'
D	[3,5],[1,2]	'12'
E	[6,6],[3,3],[4,4],[1,1]	'13'
F	[3,3]	'121','111','13'
G	[4,4]	'122','111','13'

nca(*F, G*) = { *D, C, E* }

Σχήμα 57 : Παράδειγμα γράφου με την ύπαρξη πολλαπλών nca's

Η διαμόρφωση σε *SQL* μορφή του φορμαλισμού της επερώτησης *nca* για το σχήμα κωδικοποίησης *DPInterval* χρησιμοποιεί «φωλιασμένες» επερωτήσεις, όπως και στην περίπτωση της διατύπωσης της επερώτησης *Dancestors(v)*. Το 1^{ο}

μέρος της επερώτησης αναζητά (όπως και στην περίπτωση του δέντρου) πάντοτε, με βάση το διαμορφωθέν δέντρο (*spanning-tree*), τον "κοντινότερο", κοινό απόγονο (*nca)* – που είναι ο κόμβος *D* στο παράδειγμά μας (βλ. σχήμα 53).

Τώρα στο δεύτερο μέρος έχουμε δυο επιμέρους φωλιασμένες επερωτήσεις, όπου στη μεν πρώτη αναζητούμε τους προγόνους του κόμβου *u*,που έχουν προκύψει από την εκ νέου διασύνδεση που μας παρέχουν οι προστιθέμενες στον γράφο μας *non-tree* ακμές (ομοίως με το 2^o μέρος της επερώτησης των προγόνων). Επιπρόσθετο, επιδιωκόμενο χαρακτηριστικό του συνόλου των επιστρεφόμενων κόμβων της πρώτης φωλιασμένης αυτής επερώτησης είναι ότι θα πρέπει να περιλαμβάνουν την ελάχιστη τιμή εκ των πεδίων *index* των δυο εξεταζόμενων κόμβων. Καλούνται δηλαδή να έχουν την ελάχιστη δυνατή απόσταση (που συνεπάγεται το ελάχιστο πλήθος ενδιάμεσων κόμβων) στο μονοπάτι διασύνδεσης με τον κόμβο εκ των δυο που έχει την ελάχιστη τιμή του πεδίου της *index*. Κατά πλήρη αντιστοιχία η δεύτερη φωλιασμένη επερώτηση ομοίως αναζητά τους προγόνους του κόμβου *u* από τον γράφο, οι οποίοι συνεπικουρικά θα πρέπει να περιέχουν στο πεδίο-ετικέτα τους την μέγιστη τιμή εκ των πεδίων *post* των δυο εξεταζόμενων κόμβων. Δηλαδή, θα χαρακτηρίζονται από την ελάχιστη δυνατή απόσταση (ελάχιστο πλήθος ενδιάμεσων κόμβων) στο μονοπάτι διασύνδεσης με τον κόμβο εκ των δυο εξεταζόμενων, που έχει την μέγιστη τιμή του πεδίου της *post.* Τώρα, τα αποτελέσματα των δυο αυτών φωλιασμένων επερωτήσεων (σύνολο από πεδία *index* και *post*) συνταιριάζονται και μας προσδίδουν πεδία-ετικέτες, που αντιστοιχούν με το εκάστοτε εφαρμοζόμενο σχήμα κωδικοποίησης σε κάποιους όντως υπάρχοντες κόμβους.

```
(select u.post
from PInterval u
where u.att1 <= min(i, i') and u.att0 > max(p, p') and
NOT EXISTS      (select *
                 from PInterval w
                 where w.index <= min(i, i') and max(p, p') < w.post and
                 u.index <= w.post and w.post < u.post))
UNION ALL
(select r.post
from        (select w.post, w.index, w.father_post
```

1η φωλιασμένη επερώτηση

from (select u'.post AS t from PInterval u' where u'.index <= min(*i*, *i'*) and u'.post > min(*i*, *i'*)) z, DPInterval v, PInterval w where v.post = z.t and w.index <= v.ancestor and w.post >= v.ancestor)

2η φωλιασμένη επερώτηση

r,
(select w.post, w.index, w.father_post from (select s.post AS t from PInterval s where s.index <= max(*p, p'*) and s.post > max(*p, p'*)) z, DPInterval v, PInterval w where v.post = z.t and w.index <= v. ancestor and w.post >= v. ancestor) s
where r.father_post = s. father_post)

Στη περίπτωση που δεν υφίσταται η δυνατότητα συμπίεσης των πεδίων – ετικετών η παραπάνω επερώτηση απλοποιείται κατά την ίδια λογική που απλοποιήσαμε και την επερώτηση των προγόνων για την περίπτωση του γράφου. Συγκεκριμένα, στο 2ο μέρος και σε κάθε μία από τις δυο φωλιασμένες επερωτήσεις απλοποιούνται οι επιμέρους τους συνθήκες σύγκρισης ως φαίνεται ευθύς αμέσως στη πλήρη, τροποποιημένη επερώτηση αναζήτησης *nca*.

```
(select u.post
from PInterval u
where u.att1 <= min(i, i') and u.att0 > max(p, p') and
NOT EXISTS      (select *
                 from PInterval w
                 where w.index <= min(i, i') and max(p, p') < w.post and
                 u.index <= w.post and w.post < u.post))
UNION
(select r.post
from        (select w.post, w.index, w.father_post
            from (select u'.post AS t from PInterval u' where u'.index <= min(i, i')
            and u'.post > min(i, i')) z, DPInterval v, PInterval w
            where z.t = v.ancestor) r,
            (select w.post, w.index, w.father_post from (select s.post AS t from
            PInterval s where s.index <= max(p, p') and s.post > max(p, p')) z,
            DPInterval v, PInterval w where z.t = v.ancestor) s
where r.father_post = s. father_post)
```

1η φωλιασμένη επερώτηση

2η φωλιασμένη επερώτηση

2. Unicode Dewey Encoding

Τώρα, η διαμόρφωση σε SQL μορφή του φορμαλισμού της επερώτησης *nca* για το σχήμα κωδικοποίησης *DPUPrefix* (βλ. Πίνακα 13) είναι εν μέρει σύνθετη καθώς εμπεριέχει ενσωματωμένες *SQL* επερωτήσεις σε κώδικα κι άρα μεταφράζεται σε μια ενσωματωμένη συνάρτηση. Με την σειρά της αυτή στηρίζεται στην χρήση των συναρτήσεων διαχείρισης αλφαριθμητικών *Prefixes* – η οποία παράγει το όλο το πλήθος των δυνατών προθεμάτων της αλφαριθμητικής ετικέτας του εκάστοτε εξεταζόμενου κόμβου - και της *character_length()*. Αναλυτικά, η συνάρτηση είναι η ακόλουθη (βλ. Σχήμα 58):

Algorithm ***NCA()***
Parameters:
(1) Label *'l1'*
(2) Label *'l2'*
Procedure Description:
(1) List *Labels1* = {*'l1'*}
(2) List *Labels2* = {*'l1'*}
(3) List *Prefs* = {}
(4) ∀ Χαρακτήρα Αναπαράστασης Κωδικού *l1* → *l1*[*Count*]
(5) if (*l1*[*Count*] = *l2*[*Count*])
    a. result[*Count*] = *l1*[*Count*] – Αποθήκευση Τρέχοντος Κοινού Προθέματος;
    b. *Count* = *Count* + 1
(6) *Prefs += result[Count]*
(7) *Labels* = {}, *Labels1* = {}, *Labels2* = {}
(8) *Labels1* += (select *label*
    from DUPrefix
    where *father_label* = *'l1'*)
(9) *Labels2* += (select *label*
    from DUPrefix
    where *father_label* = *'l2'*)
(10) While (*Labels1.hasNext & Labels2.hasNext*)
    a. If *Prefixes(Labels1.Next)* ∈ *Prefixes*(*Labels2*) *Labels* += *Prefixes*(*Labels1.Next*)
(11) *Prefs* +=*MAX(character_length(Labels))*
(12) Return *Prefs*

Σχήμα 58 : Αλγόριθμος NCA για το σχήμα Unicode Dewey – με την μορφή ψευδοκώδικα

5.6 Πειραματική Αξιολόγηση

Όπως και στην περίπτωση του δέντρου, για τη διασφάλιση της εγκυρότητας των πειραματικών αποτελεσμάτων, ακολουθήθηκε η ίδια μεθοδολογία αλλά και τρόπος εκτέλεσης για την έγκυρη αξιολόγηση των εν λόγω χρησιμοποιούμενων επερωτήσεων. Επακριβώς, πραγματοποιήθηκαν και πάλι 10 εκτελέσεις για την κάθε μία επερώτηση, όπου η μεν πρώτη εκτέλεση της κάθε επερώτησης πραγματοποιήθηκε με μόνο στόχο την ενεργοποίηση, «*ζέσταμα*» των καταχωρητών της βάσης και χωρίς βέβαια να ληφθούν υπόψη τα αποτελέσματά της. Οι λοιπές 9 συνεχόμενες εκτελέσεις των πειραμάτων ήταν αυτές οι οποίες μας δίνουν τον μέσο όρο του χρόνου εκτέλεσης της έκαστης επερώτησης,. Ο υπολογισμός του χρόνου εκτέλεσης σε κάθε μια ερώτηση πραγματοποιείται με την χρήση της παρεχόμενης εκ του συστήματος λειτουργίας του *PostgreSQL Explain Facility*, η οποία εκτελεί μια συνολική χρονική εκτίμηση του κόστους εκτέλεσης όλων των επιμέρους βημάτων του όλου πλάνου που διαμορφώνει ο βελτιστοποιητής του συστήματος για την εκτέλεση της επερώτησης [IPD].

Στο σημείο αυτό θα πρέπει να σημειωθεί πως σαν πεδίο δοκιμής χρησιμοποιήθηκαν οι τροποποιημένες *RDF* ιεραρχίες του *ODP* καταλόγου, από τις οποίες έχουμε δημιουργήσει από ένα συγκεντρωτικό, συνθετικό *DAG*. Η επιλογή των συγκεκριμένων κόμβων που χρησιμοποιήθηκαν σε όλες τις επιμέρους ερωτήσεις κι υποπεριπτώσεις τους είναι απολύτως κοινή με την προαναφερθείσα περίπτωση του δέντρου, ώστε να υπάρχει άμεση συγκρισιμότητα των χρονικών αποτελεσμάτων του. Αναλυτικά, η επιλογή όλων των κόμβων εικονίζεται ανάγλυφα στο σχήμα 36 και για τους οποίους θα επεξηγήσουμε εν συνεχεία όλες τις επερωτήσεις. Αναλυτικά, ακολουθούν στον Πίνακα 14 οι καταγραφέντες χρόνοι επερώτησης, το πλήθος των επιστρεφόμενων πλειάδων καθώς και το ποσοστό επιλεκτικότητάς τους επί του συνολικού πλήθους των πλειάδων στην περίπτωση που έχουμε συμπίεση πεδίων - ετικετών.

Επερωτήσεις	Περιπτώσεις Επερωτήσεων	Agrawal	Unicode Dewey	Returned tuples	Selectivity of Returned tuples (%)
Q1 : Descendants					
	Q1.1 Root node	11,929880	67,121239	253214	50
	Q1.2 Node with a medium number of descendants	0,617279	21,786647	25581	5,051
	Q1.3 Node with minimum number of descendants at the middle of hierarchy	0,000263	20,440890	2	0,0004
Q2 : Direct Descendants					
	Q2.1 Root node	0,000200	0,000560	17	0,0034
	Q2.2 Node with a medium number of descendants	0,000210	0,002520	109	0,0218
	Q2.3 Node with the minimum number of descendants at the middle of hierarchy	0,000200	0,000200	1	0,0002
Q3 : Ancestors					
	Q3.1 Leftmost leaf – smaller postnum	2,632303	0,005140	23	0,00454
	Q3.2 Rightmost leaf – bigger postnum	3,095479	0,004720	14	0,00276
	Q3.3 medium leaf - medium postnum	3,790538	0,006190	167	0,033
Q4 : Direct Ancestors					
	Q4.1 Leftmost leaf – smaller postnum	0,000320	0,000210	2	0,0004
	Q4.2 Rightmost leaf – bigger postnum	0,000330	0,000210	2	0,0004
	Q4.3 medium leaf - medium postnum	0,000300	0,000200	2	0,0004
Q5 : Subsumption Test					
	Q5.1 Root Node - Leftmost leaf	0,000190	0,001360	0	0
	Q5.2 Root Node – Rightmost leaf	0,000190	0,001370	0	0
	Q5.3 Root Node - medium leaf	0,000200	0,001340	0	0
Q6 : Leaves					

	Q6.1 Root Node	14,506418	95,582223	109132	21,55
	Q6.2 Node with a medium number of descendants	1,578370	48,192655	11201	1,6439
	Q6.3 Node with the minimum number of descendants at the middle of hierarchy	0,000087	47,297535	1	0,00101
Q7 : Brothers					
	Q7.1 Node with maximum breadth (=314)	0,002591	0,003011	317	0,0626
	Q7.2 A node with AVG breadth (=4) and with the maximum number of descendants	0,002442	0,002946	4	0,0008
	Q7.3 Node with the minimum breadth (=1)	0,000330	0,000320	2	0,0004
Q8 : NCA					
	Q8.1 Leftmost leaf –Rightmost leaf	4,870160	0,000580	1	0,0002
	Q8.2 Leftmost leaf – medium leaf	3,702870	0,000550	1	0,0002
	Q8.3 Medium leaf – Rightmost leaf	4,691551	0,000570	1	0,0002

Πίνακας 14 : Χρόνοι Εκτέλεσης (σε seconds) των βασικών Επερωτήσεων στο DAG Υποκλάσεων του ODP καταλόγου (με συμπίεση των ετικετών)

Αναλυτικά, για την επερώτηση **Q1 (*Απόγονοι*)** παρατηρούμε ότι ο βελτιστοποιητής της *PostgreSQL* και ομοίως και για τις δυο κωδικοποιήσεις επιλέγει για την μεν πρώτη εξεταζόμενη περίπτωση (Q1.1) ένωση *1)* της *σειριακής σάρωσης* (*sequential scan*) για τον πίνακα *PInterval* που περιλαμβάνει τις ετικέτες που προκύπτουν από το *spanning-tree* (ομοίως ο πίνακας *UPrefix* για τον *Unicode Dewey*) και *2)* ένα φωλιασμένο βρόχο (*nested loop*) από δυο *σαρώσεις* μία *με δείκτες* (*index scan*) στον πίνακα *DPInterval* – περιλαμβάνει τις ετικέτες που δημιουργούνται και διαδίδονται μέσω των *non-spanning tree* ακμές - και η άλλη με *σειριακή σάρωση* στον *PInterval* (στους πίνακες *DUPrefix* και *UPrefix* για το σχήμα του *Unicode Dewey* αντιστοίχως). Για τις δε δυο άλλες περιπτώσεις (Q1.2 & Q1.3) το πλάνο εκτέλεσης διαφοροποιείται στο ότι περιλαμβάνει ένωση πλέον *1)* της *σάρωσης με δείκτες* (*index scan*) για τον πίνακα *PInterval* (*UPrefix* για τον *Unicode Dewey*) και ομοίως *2) nested loop* από δυο *σαρώσεις* από κοινού *με δείκτες* (*index scan*) στον πίνακα *DPInterval* και *PInterval* (*DUPrefix* και *UPrefix* για τον *Unicode Dewey* αντιστοίχως). Κι αυτό είναι λογικό, εάν λάβουμε υπόψη ότι στην μεν $1^{η}$ περίπτωση το

πλήθος των επιστρεφόμενων πλειάδων σε σχέση με το σύνολό τους στον πίνακα είναι πολύ μεγαλύτερο από τις δε δυο άλλες, όπως ακριβώς άλλωστε παρατηρήσαμε και στην περίπτωση του δέντρου. Το κομμάτι της επερώτησης που διερευνά τον πίνακα *DPInterval* (με την χρήση του *PInterval*) λογικά εξαιτίας του σχετικά μικρού πλήθους σε κάθε περίπτωση των επιστρεφόμενων πλειάδων (πέρα της $1^{ης}$) οδηγεί τον βελτιστοποιητή στην χρησιμοποίηση των δεικτών για τους πίνακες *DPInterval*, όπου βρίσκουμε ποιοι κόμβοι έχουν - όχι μέσω του *spanning-tree*- "*πατρικό*" κόμβο τον εκάστοτε εξεταζόμενο κόμβο, και για τους οποίους μεμονωμένα επαναλαμβάνουμε την αναζήτηση των απογόνων τους στον *PInterval* (ομοίως όπως και στο $1^{ο}$ μέρος της αναζήτησης στον πίνακα *PInterval* - *DUPrefix* και *UPrefix* αντιστοίχως). Εναλλακτικά, η χρήση της δεύτερης μετάφρασης σε *SQL* θα είχε σαν αποτέλεσμα την διαμόρφωση ενός διαφορετικού πλάνου εκτέλεσης, το οποίο θα περιλάμβανε ένα *merge-join* σε μια σειριακή σάρωση στον δευτερεύοντα πίνακα *DUPrefix* – του οποίου το αποτέλεσμά του μετά ταξινομείται – και σε μια σάρωση με δείκτες στον δείκτη στο γνώρισμα *label* του πρωτεύοντος πίνακα *UPrefix*. Το κόστος εκτέλεσης του πλάνου αυτού της υποεπερώτησης, εάν κι είναι βέλτιστο για πολύ μικρό σύνολο επιστρεφόμενων πλειάδων έναντι της πρώτης περίπτωσης, καθίσταται ιδιαίτερα επιβεβαρημένο για λίγο πιο μεγάλο σύνολο επιστρεφόμενων πλειάδων (μετρήθηκε ο χρόνος για την περίπτωση Q1.2 και Q1.3 και είχαμε 277,27408 sec και 1,62654 sec αντιστοίχως). Συνεπώς, με βάση αυτές τις μετρήσεις συμπεραίνουμε πως η πρώτη μετάφραση της υποεπερώτησης αυτής που απευθείνεται για τις *non-tree* ακμές είναι εν συνόλω βέλτιστη.

Για την επερώτηση **Q2 (Άμεσοι *Απόγονοι*)** παρατηρούμε ότι ο βελτιστοποιητής της *PostgreSQL* και ομοίως και για τις δυο κωδικοποιήσεις επιλέγει για τις τρεις εξεταζόμενες περιπτώσεις (Q1.1, Q2.1 και Q2.3) *σάρωση με δείκτες* (*index scan*) για τον πίνακα *PInterval* (*UPrefix* αντιστοίχως) απευθύνεται η επερώτηση αναζήτησης των άμεσων απογόνων που έχουν ως άμεσο πατρικό τους κόμβο τον πατέρα κόμβο του εξεταζόμενου κόμβου. Σε αυτό σημαίνοντα ρόλο παίζει το γεγονός ότι έχουμε ορίσει δείκτη πάνω στο πεδίο με την ετικέτα του πατρικού κόμβου και ο οποίος χρησιμοποιείται ένεκα του μικρού πλήθους επιστρεφόμενων πλειάδων (17, 109 και 1 αντίστοιχα για τις τρεις υποπεριπτώσεις με την σειρά αναφοράς τους).

Στην επερώτηση **Q3 (*Πρόγονοι*)** έχουμε και πάλι τις τρεις περιπτώσεις : (a) περίπτωση Q2.1 του πιο αριστερού και συνάμα πιο μικρού σε απόλυτη τιμή "φύλλου"

- κόμβου της ιεραρχίας (b) περίπτωση Q2.2 ενός "φύλλου" - κόμβου σχετικά στην μέση της ιεραρχίας και (c) περίπτωση Q2.2 ενός "φύλλου" – κόμβου, ο οποίος κατά αντιστοιχία θα είναι ο πιο δεξιός στην υπό εξέταση ιεραρχία υποκλάσεων του *ODP* καταλόγου. Παρατηρώντας τον πίνακα 5 διαπιστώνουμε από τους χρόνους εκτέλεσης πως ο χρόνος ανταπόκρισης είναι σημαντικά καλύτερος στην περίπτωση του σχήματος του *UPrefix* αλγορίθμου κωδικοποίησης για τις δυο πρώτες περιπτώσεις. Ο βελτιστοποιητής της *PostgreSQL* στη κατηγορία ερώτησης αυτής δεδομένου ότι στο σχήμα του *UPrefix* χρησιμοποιείται ενσωματωμένη συνάρτηση. Τώρα για το σχήμα *PInterval* επιλέγει για την μεν πρώτη περίπτωση Q2.1 το εξής πλάνο εκτέλεσης: ένωση με απαλοιφή κοινών αποτελεσμάτων *1) σειριακή σάρωση* για τις ετικέτες από τον πίνακα *PInterval* και *σάρωση με δείκτες* για τις δυο άλλες περιπτώσεις Q2.2 και Q2.3 αντίστοιχα και *2)* ένα φωλιασμένο βρόχο (*nested loop*) . Τέλος, παρατηρούμε πως ο βελτιστοποιητής πραγματοποιεί μια καθ' όλα λανθασμένη προσέγγιση γύρω από την σύζευξη που υπάρχει στην εν λόγω χρησιμοποιούμενη συνθήκη και μάλιστα στην περίπτωση Q2.2 το οποίο σημασιολογικά περιλαμβάνει το μισό διάστημα, επιλέγει να πραγματοποιήσει *σάρωση* με *δείκτες*, το οποίο και προφανώς αποδεικνύεται να είναι σαφώς πιο χρονοβόρο από ότι θα επιδείκνυε η χρήση *σειριακής σάρωσης* σε ολόκληρο το υπάρχων διάστημα.

Στην επερώτηση **Q4 (Άμεσοι Πρόγονοι)** και για τις τρεις εξεταζόμενες περιπτώσεις έχουμε κοινό πλάνο εκτέλεσης για το κάθε σχήμα κωδικοποίησης. Συγκεκριμένα, στον *Agrawal et al* εκτελείται ένας φωλιασμένος βρόχος (*nested loop*) από δυο επιμέρους *σαρώσεις με δείκτες* (*index scan*) από κοινού στον *PInterval*, ενώ στον *Unicode Dewey* πραγματοποιείται μια σάρωση με δείκτη στον *UPrefix* αντιστοίχως. Η απλότητα του πλάνου εκτέλεσης βασίζεται στο έχουμε ορίσει δείκτη πάνω στο πεδίο του εξεταζόμενου κόμβου για τον οποίο αναζητούμε τις ετικέτες των πατρικών του κόμβων.

Προχωρώντας τώρα στην κατηγορία επερώτησης **Q6 (Φύλλα)**, έχουμε να παρατηρήσουμε εξ αρχής πως και οι τρεις εξεταζόμενες περιπτώσεις εμπίπτουν στους ίδιους ακριβώς κόμβους της επερώτησης **Q1** και μάλιστα έχουν κοινό πλάνο εκτέλεσης, δεδομένου ότι και το πρώτο μέρος της σχετικής επερώτησης και στις δυο περιπτώσεις των σχημάτων κωδικοποίησης είναι κοινό. Συνεπώς, εκ της θεώρησης αυτής ο βελτιστοποιητής επιλέγει λογικά το ίδιο πλάνο εκτέλεσης των επερωτήσεων και για τις τρεις περιπτώσεις και για το πρώτο αυτό μέρος της επερώτησης. Τώρα, στην περίπτωση του σχήματος *PInterval* το επιπλέον μέρος της επερώτησης που

περιλαμβάνει την καθ' αυτού συνθήκη για την διερεύνηση εάν είναι "*φύλλο*" (*index = post*), έχει εκ των καταγραφέντων νούμερων σχετικά πολύ μικρή επιπρόσθετη αύξηση του συνολικού χρόνου εκτέλεσης της επερώτησης **Q1**. Κι αυτό είναι απολύτως λογικό εάν λάβουμε υπόψη ότι το πλήθος των σελίδων δίσκου που υφίστανται πρόσβαση στην περίπτωση αυτή είναι κοινή με της **Q1**. Από την άλλη πλευρά, σε σχέση με το σχήμα *UPrefix* η εν λόγω επερώτηση υφίσταται μια σημαντική χρονική επιβάρυνση, κι αυτό εξαιτίας της χρήσης της βεβαρημένης από άποψη κόστους φωλιασμένης επερώτησης. Κι αυτό εξηγείται δεδομένου ότι και πάλι παρατηρούμε την χρήση σάρωσης με δείκτες και στις τρεις εξεταζόμενες περιπτώσεις για την φωλιασμένη αυτή επιμέρους επερώτηση, εάν και θα περιμέναμε τουλάχιστον στην πρώτη περίπτωση την χρήση σειριακής σάρωσης.

Η ερώτηση **Q7** η οποία ασχολείται με την αναζήτηση των ομο-επίπεδων (*siblings*) κι ενίοτε και "αδερφών" κόμβων στις περιπτώσεις του (a) κόμβου με το μεγαλύτερο πλήθος εισαγόμενων ακμών (*fan-in*), (b) ενός κόμβου με το μέσο πλήθος εισαγόμενων ακμών και (c) του κόμβου με το μικρότερο πλήθος ακμών στις ιεραρχίες υποκλάσεων του *ODP* καταλόγου. Ομοίως με την περίπτωση του δέντρου το πλάνο εκτέλεσης της ερώτησης αυτής περικλείει ένα φωλιασμένο βρόχο (*nested loop*) που συνδέει δυο *σαρώσεις με δείκτες (index scan)*, όπου η μεν πρώτη βρίσκει τον πατέρα ενός κόμβου και η δε δεύτερη βρίσκει τους άμεσους ομο-επίπεδους κόμβους, που προκύπτουν με τη χρήση των κατηγορημάτων ισότητας στα γνωρίσματα *post* (ή *label* αντιστοίχως) και του πατρικού γνωρίσματος και στα δυο σχήματα.

Ολοκληρώνοντας, έχουμε την επερώτηση **Q8** η οποία αναζητά τον πιο κοντινό - σε επίπεδο απόστασης μέσα στην ιεραρχία - κοινό "*προγονικό*" κόμβο μεταξύ των εκάστοτε δυο εισαγόμενων κόμβων. Οι τρεις εξεταζόμενες περιπτώσεις διαμορφώνονται και πάλι με τον συνδυασμό των "φύλλων" – κόμβων που έχουν προεπιλεχτεί στην επερώτηση **Q2** είναι (a) πιο αριστερού "φύλλου" - κόμβου / πιο δεξιού "φύλλου" – κόμβου, (b) πιο αριστερού "φύλλου" - κόμβου / μεσαίου φύλλου" – κόμβου και (c) μεσαίου φύλλου" – κόμβου / πιο δεξιού "φύλλου" – κόμβου. Αναλυτικά, στην μεν περίπτωση του *UPrefix* έχουμε κατασκευάσει και χρησιμοποιήσει μια ενσωματωμένη συνάρτηση και άρα δεν υφίσταται δημιουργία πλάνου εκτέλεσης με *σειριακής σάρωση* ή *σάρωσης με δείκτες*. Τώρα στην περίπτωση του *PInterval* (του *UPrefix* για τον *Unicode Dewey* αντιστοίχως) έχουμε μια ένωση με μοναδικά κοινά αποτελέσματα *1)* της *σάρωσης* με *δείκτες* (*index scan*) για τον πίνακα *PInterval* (*UPrefix*) στην αναζήτηση του πιο "κοντινού" κοινού προγόνου, και

συνεπακολούθως για την διερεύνηση ότι όντως αποτελεί ο πλέον κοντινός – αναφερόμαστε στο μέρος ***NOT EXISTS*** και *2)* ένα *merge join* από δυο φωλιασμένους βρόχους (*nested loop*) που περιλαμβάνουν ο μεν πρώτος *σειριακή σάρωση* και *σαρώσεις με δείκτες* (*index scan*) στον πίνακα *DPInterval* – περιλαμβάνει τις ετικέτες που δημιουργούνται και διαδίδονται μέσω των *non-spanning tree* ακμές - και η άλλη με *σειριακή σάρωση* στον *PInterval* (*DUPrefix* και *UPrefix* για τον *Unicode Dewey* αντιστοίχως). Όπως είχαμε προαναφέρει και στην επερώτηση των "απογόνων" και στις τρεις περιπτώσεις τα πλάνα εκτέλεσης περιλαμβάνουν επιμέρους σαρώσεις με δείκτες, εάν και θα έπρεπε στην μεν πρώτη περίπτωση να ήταν σάρωση σειριακή. Ομοίως και εδώ, δεδομένου ότι η επερώτηση *nca* στηρίζεται συνδυαστικά στην χρήση της επερώτηση των "προγόνων" παρουσιάζει αντίστοιχο πλάνο εκτέλεσης. Πάντως σε κάθε περίπτωση ο χρόνος σε αυτή την περίπτωση είναι λογικά πολύ μεγαλύτερος σε σχέση με τις άλλες δυο, οι οποίες άλλωστε είναι και συγκρίσιμες μεταξύ τους ως προς τον χρόνο εκτέλεσής τους.

Κατά αντιστοιχία, παραθέτουμε ακολούθως τους χρόνους εκτέλεσης των ίδιων βασικών επερωτήσεων στην περίπτωση που έχουμε συμπίεση στα πεδία – ετικέτες για το σχήμα του *DPInterval*. Εκ της θεωρίας είναι προφανές, το οποίο άλλωστε έμπρακτα αποδεικνύεται κι από τους χρόνους εκτέλεσης (για την περίπτωση μόνο του *DPInterval*) πως η μη συμπίεση των ετικετών απλοποιεί εν γένει την μετάφραση σε *SQL* κάποιων επερωτήσεων και για τα δυο σχήματα. Χαρακτηριστικά, μπορούμε με βάση τα αποτελέσματα των πινάκων 14 και 15 να παρατηρήσουμε την απλοποίηση εν γένει της έκφρασης σε *SQL* μορφή, για το σχήμα του *DPInterval* των επερωτήσεων αναζήτησης των προγόνων και του "κοντινότερου" –κοινού προγόνου (*nca*). Σε κάθε περίπτωση η απλοποίηση μετάφρασης άπτεται μεταξύ συναφών επερωτήσεων (π.χ. η επερώτηση των 'φύλλων" – κόμβων είναι υποπερίπτωση της επερώτησης αναζήτησης των απογόνων, περιλαμβάνει δηλαδή την συνθήκη δηλαδή των απογόνων περιορισμένη με μία επιπλέον συνθήκη περιορισμού που είναι ότι οι κόμβοι αυτοί δεν έχουν άλλους απογόνους, είναι κατά μία έννοια "τελικοί" τόσο μέσω των ακμών του *spanning-tree* όσο και του *non-spanning tree*).

Η πλέον βασική παρατήρηση που έχουμε να κάνουμε εκ της συγκρίσεως αυτής είναι παρά την εν γένει απλοποίηση της μετάφρασης σε SQL των επερωτήσεων, οι χρόνοι εκτέλεσής τους δεν μειώνονται όπως θα αναμενόταν. Κι αυτό εξαιτίας της σύνδεσης με τον τελεστή *UNION*, του οποίου το κόστος χρήσης είναι πολύ μεγάλο (σαφέστατα πιο μεγάλο από ότι του *UNION ALL*). Μάλιστα

παρατηρούμε πως η υιοθέτηση ή μη της συμπίεσης των ετικετών δεν επιφέρει καμία μεταβολή στους χρόνους των επερωτήσεων. Επιπρόσθετα, μία ακόμα αξιοσημείωτη παρατήρηση είναι ότι η απλοποίηση αυτή χωρίς τη συμπίεση των ετικετών, παρατηρείται στις ερωτήσεις αυτές που χαρακτηρίζονται από εκείνη την "*φορά*" (εναλλακτικά κατεύθυνση) αναζήτησης που συμπίπτει με την φορά μετάδοσης των ετικετών με την προσθήκη των *non-spanning tree* ακμών στο γράφο, δηλαδή για παράδειγμα *bottom-up* για το *DPInterval*.

Επερωτήσεις	Περιπτώσεις Επερωτήσεων	Agrawal	Returned tuples	Selectivity of Returned tuples (%)
Q1 : Descendants				
	Q1.1 Root node	11,825500	253214	50
	Q1.2 Node with a medium number of descendants	0,595728	25581	5,051
	Q1.3 Node with minimum number of descendants at the middle of hierarchy	0,000490	2	0,0004
Q2 : Direct Descendants				
	Q2.1 Root node	0,000200	17	0,0034
	Q2.2 Node with a medium number of descendants	0,000210	109	0,0218
	Q2.3 Node with the minimum number of descendants at the middle of hierarchy	0,000200	1	0,0002
Q3 : Ancestors				
	Q3.1 Leftmost leaf – smaller postnum	2,427604	23	0,033
	Q3.2 Rightmost leaf – bigger postnum	2,854762	14	0,00454
	Q3.3 medium leaf - medium postnum	3,49577	167	0,00276
Q4 : Direct Ancestors				
	Q4.1 Leftmost leaf – smaller postnum	0,000330	2	0,0004
	Q4.2 Rightmost leaf – bigger postnum	0,000320	2	0,0004
	Q4.3 medium leaf - medium postnum	0,000330	2	0,0004
Q5 : Subsumption Test				
	Q5.1 Root Node - Leftmost leaf	0,000200	0	0
	Q5.2 Root Node – Rightmost leaf	0,000190	0	0
	Q5.3 Root Node - medium leaf	0,000200	0	0
Q6 : Leaves				

	Q6.1 Root Node	14,488376	109132	21,55
	Q6.2 Node with a medium number of descendants	1,560662	11201	1,6439
	Q6.3 Node with the minimum number of descendants at the middle of hierarchy	0,000188	1	0,00101
Q7 : Brothers				
	Q7.1 Node with maximum breadth (=314)	0,002449	317	0,0626
	Q7.2 A node with AVG breadth (=4) and with the maximum number of descendants	0,002375	4	0,0008
	Q7.3 Node with the minimum breadth (=1)	0,000330	2	0,0004
Q8 : NCA				
	Q8.1 Leftmost leaf –Rightmost leaf	4,680228	1	0,0002
	Q8.2 Leftmost leaf – medium leaf	3,417603	1	0,0002
	Q8.3 Medium leaf – Rightmost leaf	4,184992	1	0,0002

Πίνακας 15 : Χρόνοι Εκτέλεσης (σε seconds)των βασικών Επερωτήσεων στο DAG υποκλάσεων του ODP καταλόγου (χωρίς συμπίεση των ετικετών)

Εξ αρχής, και συνολικά επί όλων των αποτελεσμάτων θα πρέπει να αναφέρουμε πως εξαιτίας της προσθήκης των *non-spanning tree* ακμών το μέγεθος των πινάκων *UPrefix* και *PInterval* πρακτικά διπλασιάζεται έναντι του μεγέθους τους στην προμελετηθείσα περίπτωση του δέντρου, ενώ η καταγραφόμενη επιλεκτικότητα των επερωτήσεων ως προς τις επιστρεφόμενες πλειάδες αντιστρόφως μειώνεται με τον ίδιο ρυθμό. Συνεπώς, με βάση τα αποτελέσματα των μετρήσεων αυτών μπορούμε να οδηγηθούμε σε μια σειρά από ασφαλή, συγκριτικά αποτελέσματα. Η πλέον βασική παρατήρηση είναι ότι οι επερωτήσεις στο σχήμα του *DUPrefix* είναι μέχρι και 3335 φορές πιο βεβαρημένες σε κόστος εκτέλεσης από ότι τις αντίστοιχες στο *DPInterval* για την περίπτωση της συμπίεσης ετικετών (περίπτωση Q1.3 και Q6.3). Κι αυτό οφείλεται ότι τα δυο είδη επερωτήσεων αυτών βασίζονται στην ίδια λογική και στη δε περίπτωση του *DUPrefix* χαρακτηρίζεται από την χρήση ενσωματωμένων συναρτήσεων διαχείρισης, που παρουσιάζουν εξ αρχής ένα μεγάλο κόστος εκτέλεσης, σχεδόν ανεξάρτητα από τον εκάστοτε εξεταζόμενο κόμβο (περιπτώσεις Q1.2 - Q1.3 και Q6.2 - Q6.3).

Επιπρόσθετα, παρατηρείται ότι οι χρόνοι εκτέλεσης των επερωτήσεων για τους απογόνους για τον *DPInterval* υπερτερούν του *DUPrefix* από 7 μέχρι και 27088 φορές (περίπτωση Q1.3 – πίνακας 14) και για τα "φύλλα"- κόμβους από 7 μέχρι και 51159 φορές (περίπτωση Q6.3 - πίνακας 14). Συγκεκριμένα, η επερώτηση για τα "φύλλα"- κόμβους στο *DUPrefix* επιβαρύνεται λόγω της χρήσης δυο επιμέρους φωλιασμένων επιμέρους επερωτήσεων – όπου η μεν μία πραγματοποιεί τον έλεγχο *Propdown(u)* =0, με βάση τον σχετικό φορμαλισμό της εν λόγω επερώτησης στον πίνακα 13, και η δε άλλη αναφέρεται στο *spanning tree* (αντιστοιχία με την μελετηθείσα περίπτωση του δέντρου). Κι αυτό έχει σαν συνέπεια μεγάλη επιβάρυνση στο κόστος αφού η εκτέλεση της πολύπλοκης αυτής επερώτησης γίνεται πάντα ανεξάρτητα από το πλήθος των επιστρεφόμενων πλειάδων. Από την άλλη πλευρά, ο χρόνος εκτέλεσης των επερωτήσεων για τους προγόνους και το *nca* στο *DUPrefix* σε σχέση με τον *DPInterval* είναι πρακτικά σε σταθερό χρόνο, κάτι που εξηγείται εάν λάβουμε υπόψη μας ότι η δημιουργία αυτών των επερωτήσεων γίνεται μέσω ενσωματωμένων συναρτήσεων.

Στην περίπτωση μη συμπίεσης των ετικετών, συγκρίνοντας ενδεικτικά για το σχήμα του *DPInterval,* παρατηρούμε μια πολύ ελάχιστη ελάττωση του χρόνου εκτέλεσης των επερωτήσεων αυτών. Κι αυτό γιατί όντως απλοποιήθηκαν ως προς την μετάφραση τους σε λιγότερο πολύπλοκες έναντι των αντίστοιχων επερωτήσεων μεγαλύτερης πολυπλοκότητας στην περίπτωση της συμπίεσης και εν μέρει μεγαλύτερων χρόνων εκτέλεσης. Εν γένει όμως, έχουμε να παρατηρήσουμε ότι τα αποτελέσματα των χρόνων είναι περίπου όμοια κάτι που βασίζεται στο γεγονός ότι η χρήση του *UNION* με το (*duplicate elimination*) επιβαρύνει με ένα κόστος εκτέλεσης που ισοσκελίζει την απλοποίηση της μετάφρασης σε *SQL* κάποιων επερωτήσεων. Άρα συμπεραίνουμε πως η διαδικασία εισαγωγής της συμπίεσης εάν κι επιφέρει κάποια μείωση στο αναγκαίο χώρο αποθήκευσης του εκάστοτε σχήματος κωδικοποίησης γράφων, χρησιμοποιεί πιο πολύπλοκες επερωτήσεις με το ίδιο περίπου χρονικό κόστος εκτέλεσης των πλέον βασικών επερωτήσεων

5.7 Συμπεράσματα

Ένα πρώτιστο συμπέρασμα που προκύπτει ως αποτέλεσμα από την υιοθέτηση της συμπίεσης των ετικετών είναι η αύξηση της πολυπλοκότητας έκφρασης των επερωτήσεων και της μετάφρασής τους σε *SQL* μορφή. Κι αυτό είναι λογικό εάν λάβουμε υπόψη ότι τα μονοπάτια που συνδέουν δυο επιμέρους κόμβους του *DAG* που μπορούν να περιλαμβάνουν κι επιμέρους *non-spanning tree* ακμές δεν μπορούν να αναπαρασταθούν πλήρως στους πίνακες *DUPrefix* και *DPInterval*, αλλά υπάρχουν πολλές *non-spanning tree* ακμές τους που παραλείπονται, λόγω της συμπίεσης. Εναλλακτικά, η μη υιοθέτηση της συμπίεσης οδηγεί σε πλήρη καταγραφή όλων των ακμών και στους δυο πίνακες της σχεσιακής αναπαράστασης του κάθε σχήματος, γεγονός που επακόλουθα αυξάνει τον επιζητούμενο χώρο αποθήκευσης. Ενδεικτικά, στην περίπτωση του *DPInterval* έχουμε πάνω από 3 φορές μεγαλύτερο το μέγεθος του στην περίπτωση της μη συμπίεσης από αυτή με συμπίεση.

Η σχεσιακή αναπαράσταση των εν λόγω σχημάτων κωδικοποίησης για την περίπτωση των *DAG* ιεραρχιών αποφέρει σαφώς ένα επιπρόσθετο κόστος στο αναγκαίο χώρο αποθήκευσής του – μάλιστα προσεγγίζει σχεδόν διπλάσια τιμή. Αυτό οφείλεται στο φαινόμενο της μετάδοσης των ετικετών, που προκαλείται από την προσθήκη των *non-spanning tree* ακμών στο *DAG*. Μάλιστα έχει παρατηρηθεί ότι εξίσου και τα δυο σχήματα επηρεάζονται κατά το ίδιο σημαντικό βαθμό από την διαδικασία προσθήκης *non-spanning tree* ακμών, κάτι που συμβαίνει στις περισσότερες περιπτώσεις στην πράξη όντως (δηλαδή μικρό ποσοστό προστιθέμενων ακμών). Σε οποιαδήποτε άλλη περίπτωση και συγκεκριμένα καθώς το πλήθος των προστιθέμενων ακμών αυξάνεται σημαντικά, η παρατηρούμενη συμπεριφορά αποκλίνει αρκετά μεταξύ τους με το μέγεθος του *DUPrefix* να αυξάνεται συνεχώς, ενώ του *DPInterval* να παραμένει σχεδόν σταθερό . Αντιθέτως, στην δε περίπτωση του δέντρου τα δυο σχήματα πάλι έχουν μια διαφορά στον αναγκαίο χώρο αποθήκευσης, με το *UPrefix* να υπερέχει του *PInterval* ,αλλά με μια σταθερή διαφορά εξ αρχής, η οποία παρά τις όποιες διακυμάνσεις ανά βήμα παραμένει εν γένει σταθερή. Επίσης το μέγεθος τους δεν καθορίζεται από το πλήθος των πλειάδων που είναι ίσο με το πλήθος των κλάσεων, αλλά κυρίως από το μέγεθος αναπαράστασης

της κάθε πλειάδας για το κάθε σχήμα. Αναλυτικά, το σχήμα του *DPInterval* (με συμπίεση) είναι 2,24 φορές μεγαλύτερο από ότι την περίπτωση του *PInterval* , και το σχήμα του *DPInterval* (χωρίς συμπίεση) είναι 2,76 φορές μεγαλύτερο από ότι την περίπτωση του *Pinterval* επίσης. Για το σχήμα του *DUPrefix* (με συμπίεση) έχουμε να αναφέρουμε πως είναι 2,71 φορές μεγαλύτερο έναντι του *UPrefix*.

Παρατηρούμε επίσης πως ο χρόνος εκτέλεσης των επερωτήσεων στα δέντρα σε σχέση με τα *DAGs* – με συμπίεση- είναι πολύ μεγαλύτερος. Κι αυτό είναι λογικό, εφόσον στην περίπτωση του σχήματος αναπαράστασής των *DAGs* διαχειρίζονται δυο επιμέρους πίνακες με πολλαπλάσιο μέγεθος, ενώ αντίστοιχα κι οι επερωτήσεις τους καθίστανται πιο πολύπλοκες με σημαντική επιβάρυνση στον συνολικό χρόνος εκτέλεσής τους. Εν γένει οι χρόνοι εκτέλεσης των επερωτήσεων στην περίπτωση μελέτης των *DAGs* είναι από 1,43 μέχρι και 11464 φορές για τον *DPInterval* (με χρήση συμπίεσης) και από 2,83 μέχρι και 84459 φορές (με χρήση συμπίεσης) για το σχήμα του *DUPrefix* μεγαλύτεροι έναντι της περίπτωσης των δέντρων. Αναλυτικά, ο χρόνος εκτέλεσης της επερώτησης των απογόνων για τα *DAGs* είναι από 3,34 μέχρι και 4,15 φορές πιο μεγάλος σε σχέση με την περίπτωση του δέντρου για το *DPInterval* και από 24 φορές μέχρι και 75728 για το *DUPrefix*. Μάλιστα η διαφορά επίδοσης χρόνων μεταξύ των δυο σχημάτων είναι μέχρι και 5 τάξεις μεγέθους μεγαλύτεροι του *DUPrefix* έναντι του *DPInterval*. Στην περίπτωση των προγόνων ο χρόνος είναι από 2,21 μέχρι και 11464,73 φορές μεγαλύτερος για το *DPInterval* και 5 φορές μεγαλύτερος για το σχήμα του *DUPrefix*. Για τα φύλλα είναι από 3,2 μέχρι και 7 φορές πιο μεγαλύτερος για το *DPInterval* και από 2,91 μέχρι και 85995,5 φορές πιο μεγάλος για το *DUPrefix* αντιστοίχως. Τέλος, ο χρόνος εκτέλεσης της επερώτησης για *nca* είναι από 1,5 μέχρι και 9,45 φορές μεγαλύτερος στην περίπτωση του *DAG* έναντι του δέντρου . Στην περίπτωση των μελετούμενων *DAGs* και μάλιστα για το σχήμα κωδικοποίησης του προθέματος (*DUPrefix*) μερικές ερωτήσεις όπως αναφέραμε αναλυτικά και πιο πάνω, είναι μέχρι και 543649 φορές πιο δαπανηρές σε χρόνο εκτέλεσης από τις αντίστοιχες τους για το σχήμα *DPInterval*. Εναλλακτικά, στην περίπτωση μη υιοθέτησης της συμπίεσης οι χρόνοι εκτέλεσης για το σχήμα *DPInterval* κυμαίνονται από 1,39 έως και 10573 φορές μεγαλύτεροι από την περίπτωση των αντίστοιχων επερωτήσεων σε δέντρο.

Το πιο σημαντικό όμως εδώ είναι να αναφέρουμε πως όταν υιοθετούμε την συμπίεση των ετικετών, τότε η υλοποίηση σε μορφή *SQL* των επερωτήσεων, που επιζητούμε, οδηγεί σε αποτελέσματα υλοποιήσεων ιδιαίτερα πολύπλοκα με επίδραση

αύξησης στο χρόνο εκτέλεσης των βασικών επερωτήσεων. Το φαινόμενο αυτό αντισταθμίζεται από την περίπτωση εκείνη που δεν χρησιμοποιούμε την δυνατότητα της συμπίεσης και χρησιμοποιούμε, για την σύνδεση των επιμέρους υποεπερωτήσεων, το *UNION* με σημαντική επιβάρυνση στο κόστος εκτέλεσης, καλύπτοντας έτσι το ενδεχόμενο να παραχθούν οι ίδιοι κωδικοί από τις ακμές του επιλεχθέντος *spanning-tree* (δέντρο) αλλά κι από τις *non- spanning-tree* ακμές του σχηματιζόμενου γράφου. Συνεπώς, καταλήγουμε στο ότι όπως αποδείψαμε για το σχήμα του *DPInterval* η χρήση ή μη συμπίεσης μας οδηγεί σε παρόμοιους χρόνους εκτέλεσης των βασικών επερωτήσεων.

ΚΕΦΑΛΑΙΟ 6

ΕΠΙΛΟΓΟΣ

Μέσα στα πλαίσια της μεταπτυχιακής μου εργασίας προχώρησα σε μια επιμέρους μελέτη των πλέον βασικών κατηγοριών σχημάτων κωδικοποίησης που απευθύνονται για τις *RDF* ιεραρχίες κλάσεων /ιδιοτήτων κι προχωρήσαμε σε μία συγκριτική αξιολόγηση των επιδόσεών τους σε σχέση με την αναπαράσταση και εν συνεχεία αποθήκευση των *RDF* μεταδεδομένων σε οντοκεντρικές – σχεσιακές βάσεις δεδομένων (αναφερόμενος στη χρήση της *PostgreSQL* στην διαδικασία αξιολόγησης των σχημάτων κωδικοποίησης).

Μάλιστα, προχωρήσαμε στην επιλογή των δυο πλέον αντιπροσωπευτικών σχημάτων κωδικοποίησης των βασικών κατηγοριών, δηλαδή των *Agrawal et al* για τα σχήματα κωδικοποίησης με χρήση *διαστήματος* και *Unicode Dewey* για τα σχήματα κωδικοποίησης με χρήση *προθέματος*. Εν συνεχεία τους συγκρίναμε με κριτήρια τον αναγκαίο χώρο και χρόνο αποθήκευσής τους για την γενική ακολουθούμενη – με σχετική αντιστοιχία μεταξύ τους - σχεσιακή αναπαράσταση που διαμορφώσαμε για αυτούς, και ιδίως ως προς τον χρόνο εκτέλεσης ενός συνόλου βασικών επερωτήσεων γραμμένων σε βέλτιστη μορφή, στο μέγιστο δυνατό βαθμό σε *SQL* μορφή, που απαιτείται για την αναζήτηση και διαχείριση, βέλτιστη σε σχέση με το αποτέλεσμα και τον χρόνο, με βάση την τοπολογία των κόμβων στις ιεραρχίες (που περιλαμβάνει τον ορισμό και χρήση δεικτών πάνω στα πεδία-κωδικούς).

Επιπρόσθετα, η συγκριτική αυτή αξιολόγηση των διάφορων αυτών αλγορίθμων κωδικοποίησης επεκτάθηκε πέρα των *RDF* ιεραρχιών του *ODP* καταλόγου με την μονή κληρονομικότητα (μορφολογία δέντρου), και σε ιεραρχίες με πολλαπλή κληρονομικότητα – ύπαρξη πολλαπλών *isA* συνδέσεων (μορφολογία γράφου - *DAG*). Η παραγωγή των *RDF* αυτών ιεραρχιών με πολλαπλή κληρονομικότητα (συνθετικών *DAG*) βασίστηκε στις αρχικές ιεραρχίες μονής κληρονομικότητας που χρησιμοποιήθηκαν για την αξιολόγηση στην περίπτωση των δέντρων (οι 16 *RDF* ιεραρχίες του *ODP* καταλόγου) με την προσθήκη *non-spanning tree* ακμών. Η επιμέρους διαδικασία προσθήκης των *non-tree* αυτών ακμών έγινε με

βάση μια ψευδο-πιθανοκρατική κατανομή, σε ότι αφορά την επιλογή των κόμβων "*αφετηρίας*" και "*προορισμού*" των ακμών αυτών κι ισοκατανέμοντάς τους με κριτήριο το βάθος τους στην ιεραρχία (αναφορικά σε κοντά στον ριζικό κόμβο, κοντά στην μέση της ιεραρχίας και κοντά στα "φύλλα" – κόμβους). Μία επιπλέον προοπτική η οποία μελετήθηκε σε σχέση με την διαφοροποίηση ως προς τον χρόνο εκτέλεσης των βασικών επερωτήσεων για την περίπτωση αυτή του γράφου με την χρήση της δυνατότητας συμπίεσης των ετικετών αλλά και χωρίς.

6.1 Μελλοντική Εργασία

Σίγουρα, το θέμα της βελτιστοποίησης επερωτήσεων στο σύνολο των υπαρχόντων συστημάτων βάσεων δεδομένων είναι ένα ευρύ θέμα που άπτεται συνεχών βελτιωτικών προτάσεων λαμβάνοντας υπόψη την αναγκαία εξισορρόπηση πολλών, ίσως κι αντιφατικών κριτηρίων μεταξύ τους. Συγκεκριμενοποιούμε τα κριτήρια μας με βάση το ευρύτερο περιβάλλον εργασίας μας, που είναι ο *Σημασιολογικός Ιστός* (*Semantic Web*), κι ο οποίος χαρακτηρίζεται από την ανάγκη για γρήγορη κι αποδοτική δημιουργία, ανταλλαγή κι επεξεργασία πληροφορίας. Συνεπώς, για να επιτευχθεί η αναγκαία αυτή λειτουργικότητα είναι απαραίτητη η βελτιστοποίηση των επερωτήσεων τόσο από την άποψη του χρόνου εκτέλεσης αλλά και του απαραίτητου χώρου αποθήκευσης των κωδικοποιημένων δεδομένων σε αντίθεση με την ανεπεξέργαστη μορφή τους.

Ένα πρώτιστο έναυσμα μιας μελλοντικής εργασίας θα μπορούσε καταρχήν να είναι η εκ νέου μελέτη ενός άλλου υπάρχοντος σχήματος κωδικοποίησης ή στον σχεδιασμό ενός υβριδικού συνδυασμού των δυο αντιπροσωπευτικών αυτών σχημάτων κωδικοποίησης, που προμελετήσαμε αναλυτικά, ενστερνίζοντας στο μέγιστο βαθμό τα θετικά χαρακτηριστικά γνωρίσματα του καθενός. Μάλιστα η μελέτη αυτή αξιολόγησης ενός άλλου σχήματος κωδικοποίησης θα μπορούσε να διευρυνθεί κι υπό το πρίσμα πρωτίστως άλλων παραμέτρων ή και συνεπικουρικά κι άλλων, δεδομένου ότι στην περίπτωση της πειραματικής μας αξιολόγησης τέθηκαν από κοινού ο χρόνος εκτέλεσης των επερωτήσεων αλλά εξίσου κι ο χώρος αποθήκευσης των κωδικοποιημένων σχημάτων.

Μια άλλη πολύ σημαντική συνισταμένη που θα μπορούσε να διερευνηθεί θα ήταν η χρήση ενός διαφορετικούς φιλοσοφίας και δυνατοτήτων συστήματος

διαχείρισης βάσεων δεδομένων (*DBMS*) από την οντοκεντρική τύπου *PostgreSQL* που χρησιμοποιήσαμε στα πλαίσια της πειραματικής μας αξιολόγησής μας. Κι αυτό γιατί μέσω της πειραματικής μας αξιολόγησης παρατηρήσαμε πως για την αποθήκευση κατά μέσο όρο μιας πλειάδας και για τα δυο σχήματα, απαιτείται ένας εξαιρετικά επιπλέον μεγάλος αποθηκευτικός χώρος πέρα του αναγκαίου για την αποθήκευση της σχεσιακής αναπαράστασης και που οφείλεται στην ανάγκη του συστήματος για απόλυτο προσδιορισμό ταυτότητας της κάθε πλειάδας στο φυσικό χώρο αποθήκευσής της. Συνεπώς, θα πρέπει να διερευνηθεί εάν υπάρχουν άλλα συστήματα που θα μπορούν να έχουν μια πιο βέλτιστη συμπεριφορά και χαρακτηριστικά επιμέρους απόδοσης λειτουργίας.

Επίσης, το θέμα της βέλτιστης απόδοσης των επερωτήσεων στην περίπτωση ενός γράφου *DAG* θα μπορούσε να διερευνηθεί και περαιτέρω, καθώς είναι ένα πολύ σύνθετο θέμα. Θα μπορούσαν καταρχήν να χρησιμοποιηθούν ως πεδία δοκιμής κι άλλες υπάρχουσες και ευρέως χρησιμοποιούμενες ιεραρχίες πολλαπλών συνδέσεων *isA* ή να δημιουργηθούν από αντίστοιχες μόνων συνδέσεων *isA* (όπως ενεργήσαμε κι εμείς στην πειραματική μας αξιολόγηση) πέρα των ιεραρχιών που προέρχονται από πύλες διαδικτυακών κοινοτήτων (*ODP* κατάλογος) επεκτείνοντας έτσι την έρευνα και σε άλλες εφαρμογές του Σημασιολογικού Ιστού. Μάλιστα στην περίπτωση διαμόρφωσης συνθετικών γράφων, θα μπορούσε να υιοθετηθεί και μια καθ' όλα διαφορετική φιλοσοφία διασύνδεσης των κόμβων μεταξύ τους, λαμβάνοντας ότι στην παρούσα εργασία μας ήταν κάποιοι παράμετροι που σχετίζονται με την επιδιωκόμενη μορφολογία και τοπολογική τοποθέτηση των κόμβων (όπως το βάθος του κόμβου στην ιεραρχία, ο αριθμός εισερχόμενων /εξερχόμενων ακμών- (*fan-in / fan-out*), ο μέγιστος αριθμός άμεσων πατρικών κόμβων ανά κόμβο, το πλήθος των προγόνων κάθε κόμβου κλπ) και κατά συνέπεια ολόκληρου του γράφου. Επιπρόσθετα, στα κριτήρια αυτά θα μπορούσαν επίσης να συμπεριληφθούν και το πλήθος των απογόνων του κάθε κόμβου. Σίγουρα όμως, θα μπορούσαν τα επιλεχθέντα κριτήρια αυτά να επεκταθούν και πέρα από την στυγνή τοπολογική θέση των κόμβων μέσα στις ιεραρχίες και να αποκτήσουν επιπρόσθετα και σημασιολογικό περιεχόμενο. Αναλυτικά, θα μπορούσαν να υπάρχουν σαφή όρια γύρω από την εκ νέου προσθήκη μιας *non-tree* ακμής που θα θέλουμε να συνδέει δυο συγκεκριμένους κόμβους και που να προσδίδει μια ειδική σημασιολογία μέσα στην ιεραρχία που βρίσκεται κάθε φορά.

Τέλος, μία άλλη πτυχή που δεν έχουμε θίξει σε βάθος σ' αυτήν την εργασία και χρίζει σίγουρα της προσοχής μας είναι η δυνατότητα διαχείρισης ενδεχόμενων αυξητικών ενημερώσεων των *RDF (*ομοίως και *RDF/S*) ιεραρχιών. Γνωρίζουμε ότι εν γένει το *RDF* δεν παρέχει την δυνατότητα ενημέρωσης των μεταδεδομένων καθώς η *XML* σύνταξη με την οποία αυτά αναπαριστάνονται δεν υποστηρίζει την ενημέρωση. Ωστόσο η επιλογή της κατάλληλης κωδικοποίησης των ιεραρχιών με βάση την θέση των κόμβων μέσα τους, μπορεί να μας επιτρέψει την προσθήκη και κωδικοποίηση των καινούργιων κόμβων, αποφεύγοντας κυρίως τον επαναϋπολογισμό όλων των αποδοθέντων κωδικών ή ενός μεγάλου μέρους αυτών, γεγονός που θα έχει ένα σαφέστατα επιπρόσθετο κόστος επιβάρυνσης στην όλη διαδικασία των επιθυμητών αυξητικών ενημερώσεων. Επίσης, θα πρέπει αυτή η δυνατότητα λειτουργίας να παρέχεται με την ελάχιστη δυνατή πολυπλοκότητα και να μην είναι ιδιαίτερα χρονοβόρα σε αναλογία με το μέγεθος και την μορφολογία των ιεραρχιών αλλά και το πλήθος των επιδιωκόμενων αυξητικών ενημερώσεων.

Βιβλιογραφία

[ABJ89] Rakesh Agrawal, Alexander Borgida, and H. V.Jagadish. *Efficient management of transitive relationships in large data and knowledge bases*. In Proc. of the 1989 SIGMOD Inter. Conf. On Manag. Of Data, pages 253-262, Portland, Oregon, 1989

[AKCPT01] S. Alexaki, G. Karvounarakis, V. Christophides, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: *Managing Voluminous RDF Description Bases*. In 2nd Inter. Workshop on the Semantic Web, pages 1-13, Hong Kong, 2001.

[BG00] D. Brickley and R.V. Guha. *Resource Description Framework (RDF) Schema Specification 1.0*, W3C Candidate Recommendation, 2000.

[BHL99] Tim Bray, Dave Hollander, and Andrew Layman. *Namespaces in XML.*W3C Recommendation, Jan 1999. Available at http://www.w3.org/TR/REC-xml-names.

[BLHL01] Tim Berners-Lee, James Hendler, Ora Lassila. *The Semantic Web*. Scientific American. May 2001. http://www.sciam.com/2001/0501issue/0501berbers=lee.html

[BPS98] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. *Extensible Markup Language (XML) 1.0.* W3C Recommendation, 10 February 1998. Available at http://www.w3.org/TR/REC-xml.

[C93] Yves Caseau. *Efficient handling of multiple inheritance hierarchies*. In OOPSLA'93, Washington, 1993.

[CVZTZ02] S.-Y. Chien, Z. Vagena, D. Zhang, V. J. Tsotras, and C. Zaniolo. *Efficient structural joins on indexed xml documents*. In Proc. of the Inter. Conf. On Very Large Data Bases (VLDB'02), Hong Kong, China, 2002.

[CKM02] E. Cohen, H. Kaplan, and T. Milo. *Labeling dymanic xml trees*. In Proc. of the Twenty-first Symposium on Principles of Database Systems (PODS'02), Wisconsin, USA, 2002. ACM.

[CSFH01] Brian Cooper, Neal Sample, Michael J. Franklin, G'isli R. Hjaltason, and Moshe Shadmon. *A fast index for semistructured data*. In Proc. of the Inter. Conf. On Very Large Data Bases (VLDB'01), 2001.

[D82] Paul F. Dietz. *Maintaining order in a linked list.* In Proc. of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC'82), pages 122-127, San Francisco, California, USA, 1982.

[Delphi99] The Delphi Group Releases *Model for Characterizing Portal Markets*. San Diego, CA - Delphi's International Knowledge Management Summit March 29, 1999. Available at http://wwd.delphigroup.com/pressreleases/1999-PR/03291999IKMSPortalModel.html

[DS87] Paul F. Dietz and Daniel Dominic Sleator. *Two algorithms for maintaining order in a list.* In Proc. of the Sixteen Annual ACM Symposium on Theory of Computing (STOC'87), pages 365-372, New York, NY, USA, 1987.

[E97] Gerard Ellis *Sparse term encoding for dynamic taxonomies*. In Proc. Of The Fourth Intern. Conference on Conceptual Structures, Sydney, Australia, 1997.

[FA99] C. Finkelstein and P. Aiken. *Building Corporate Portals with XML*. McGaw-Hill, 1999.

[GB97] R.V. Guha and Tim Bray. *Meta Content Framework using XML.* June 1997. Available at http://www.w3.org/TR/NOTE-MCF-XML/

[GP03] Cyril Gavoille and David Peleg. *Compact and localized distributed data structures*. Journal of Distributed Computing, Special Issue for the Twenty Years of Distributed Computing Research, 2003.

[IPD] *Interactive PostgreSQL Documentation*. Availiable at http://www.postgresql.org/idocs/.

[K93] W. Eliot Kimber. HyTime and SGML: Understanding the HyTime HyQ Query Language, August 1993. Available via anonymous ftp at ftp.ifi.uio.no/pub/SGML/HyTime.

[KACP02] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: *A Declarative Query Language for RDF*. In Proc. of the Eleventh Inter. World Wide Web Conf. (WWW'02), pages 592-603, Honolulu, Hawaii, USA, 2002.

[KBL89] Hassan Ait-Kaci, Robert Boyer, Patrick Lincoln, and Roger Nasr. *Efficient implementation of lattice operations*. ACM Trans. on Progr. Languages and Systems, 11(1):115-146, 1989.

[KMRT96] Tim Krauskopf, Jim Miller, Paul Resnick, and Win Treese. *PICS Label Distribution Label Syntax and Communication Protocols.* W3C Recommendation, October 1996. Available at http://www.w3.org/TR/REC-PICS-labels-961031

[KVH97] Andreas Krall, Jan Vitek, and Nigel Horspool. *Near optimal hierarchical encoding of types*. In 11th European Conf. on Object Oriented Programming (ECOOP'97), pages 128-145, Finland, 1997.

[LFIM98] Tim Berners-Lee, R. Fielding, U.C. Irvine, and L. Masinter.*Uniform resource identifiers (uri): Generic syntax.* RFC 2396, August 1998. Available at http://www.ietf.org/rfc/rfc2396.txt.

[LLD96] C. Lagoze, C. A. Lynch, and R. Daniel. *The Warwick Framework: A Container Architecture for Aggregating Sets of Metadata*. June 2 1996. Available at http://cs-tr.cs.cornell.edu:80/Dienst/UI/2.0/Describe/ncstrl.cornell/TR96-1593

[LM01] Quanzhong Li and Bongki Moon. *Indexing and querying XML data for regular path expressions*. In Proc. of 27th Inter. Conf. on Very Large Data Bases(VLDB'02), pages 361-370, Roma, Italy, 2001.

[LS99] O. Lassila and R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification,* W3C Recommendation, 1999.

[MACM01] Amelie Marian, Serge Abiteboul, Gregory Cobena, and Laurent Migne. *Change-centric management of versions in an xml warehouse*. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass editors, *Proc. Of 27th International Conference on Very Large Data Bases (VLDB'02)* Roma, Italy, September 2001. Morgan Kaufmann

[MACP02] A. Maganaraki, S. Alexaki, V. Christophides, and D. Plexousakis. *Benchmarking rdf schemas for the semantic web.* In Proc. of the First Inter. Semantic Web Conf. (ISWC'02), pages 132-147, Italy, 2002.

[OCLC] Online Computer Library Center. *Dewey decimal classification*. Availiable at www.oclc.org/dewey/.

[S99] Avi Saha. *Application Framework for e-business: Portals.* IBM Software Strategy, November 1999. Available at http:// ww4.ibm.com/ software / developer / library / portals /index.html

[SKM02] R. Shabo, H. Kaplan, T. Milo. *A comparison of labeling schemes for ancestor queries*. In Proc of the thirteen Annual Symposium on Discrete Algorithms

(SODA'02), San Francisco, California, USA, 2002. ACM/SIAM.

[SPT87] L. K. Schubert, M. A. Papalaskaris, and J. Taugher. *Accelerating deductive inference: Special methods for taxonomies, colours and times*. In N. Cercone and G. McCalla, editors, The Knowledge Frontier, pages 187-220. Springer-Verlag, New York, 1987.

[STC02] N. Spyratos, Y. Tzitzikas, and V. Christophides. *On personalizing the catalogs of web portals*. In Proc. of the Special Track on the Semantic Web at the 15th Inter. FLAIRS'02 Conf., Florida, USA, 2002.

[TC97] M. Theodorakis and P. Constantopoulos. *On Context-based Naming in Information Bases*. Proc. 2nd IFCIS International Conference on Cooperative Information Systems (CoopIS'97), Charleston, South Carolina, USA, June 24-27, 1997 pp.140-149. Available at http://www.ics.forth.gr/~etheodor/ijcis97/html/paper.html

[TVBS02] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang. *Storing and querying ordered xml using a relational database system*. In Proc. of the SIGMOD Inter. Conf. On Manag. Of Data, Wisconsin, USA, 2002.

[T84] A.K. Tsakalidis. *Maintaining order in a generalized linked list*. Acta Informatica, 21:101-112, 1984.

[T99] Daniel Tkach. *Knowledge Portals*. IBM 1999. Available at http://www-4.ibm.com/software/data/km/advances/kportals.html

[W88] N. Wirth. Type extensions. ACM Trans. on Progr. Languages and Systems, 10(2):204-214, 1988.

[W99] Sarah Roberts-Witt. *Making Sense of Portal Pandemonium.* 1999. Available at http://department.stthomas.edu/msejourn/V10-1/art-Roberts-Witt.htm

[WMD95] S. Weibel, J. Miller, and R. Daniel. Dublin Core. In OCLC/NCS *A metadata workshop report*, 1995.

[ZNWLL] Chun Zhang, Jeffrey F. Naughton, David J. DeWitt, Qiong Luo, and Guy M. Lohman. *On supporting containment queries in relational database management systems*. In Proc. of the SIGMOD Inter. Conf. On Manag. Of Data, Santa Barbara, California, USA, 2001.

Παράρτημα

Πλάνα Εκτέλεσης Βασικών Επερωτήσεων – Περίπτωση Δέντρων και DAGs

1. Περιγραφή Παραμέτρων Υπολογισμού κόστους κατά την εκτέλεση σειριακής σάρωσης (sequential scan) και σάρωσης με δείκτες (index scan)

Ας θεωρήσουμε το παράδειγμα του παρακάτω πλάνου εκτέλεσης μιας επερώτησης. Με βάση αυτό το παράδειγμα προχωρούμε σε μία ενδεικτική επεξήγηση των βασικών παραμέτρων που διαμορφώνουν το πλάνο εκτέλεσης.

(1) (2) (3) (4)

Index Scan using index1 on tenk1 (cost=0.00..181.09 rows=49 width=148)

Αναλυτικά, θα έχουμε να πούμε για κάθε μία από τις υποσημειώσεις :

(1) Εκτίμηση κόστους χρόνου εκκίνησης (*start-up cost*) –Πρόκειται για τον χρόνο που καταναλώνεται προτού μια εξωτερική σάρωση μπορεί να ξεκινήσει (για παράδειγμα ο χρόνος που χρειάζεται για να γίνει η ταξινόμηση σε έναν κόμβο SORT)

(2) Εκτίμηση συνολικού κόστους (εάν όλες οι πλειάδες έχουν ανακληθεί, οι οποίες μπορεί και να μην έχουν --- μια επερώτηση με ένα LIMIT θα σταματήσει να στερείται να αποδίδει στο συνολικό κότος εκτέλεσης)

(3) Εκτίμηση του πλήθους σειρών που επιστρέφονται από το πλάνο εκτέλεσης του κόμβου αυτού

(4) Εκτίμηση μέσου όρου πλάτους (σε bytes) των πλειάδων που επιστρέφονται από το πλάνο εκτέλεσης του κόμβου αυτού

Παρατηρήσεις :

1. Τα κόστη εκτέλεσης μετριούνται με επιμέρους μονάδες πρόσβασης στις σελίδες δίσκου (***disk page fetches***). Αναφορικά, για το παράδειγμα που προείδαμε πλάνου εκτέλεσης υπολογίζεται το κόστος σε 181 ***page reads***, όπου κάθε ένα επιμέρους ***page read*** έχει κόστος 1.0 + 49 * ***cpu_tuple_cost*** (όπου 49 θεωρούμε το πλήθος των πλειάδων στην προκείμενη περίπτωση του παραδείγματος και εκ των προτέρων θεωρούμε ότι το ***cpu_tuple_cost*** = 0.01- η παράμετρος αυτή by default και μπορεί και καθορίζεται /τροποποιείται με την εντολή set cpu_tuple_cost).

2. Οι επιστρεφόμενες σειρές δεν ανταποκρίνονται ακριβώς στον αριθμό των σειρών που επεξεργάζονται / σαρώνονται από την επερώτηση --- συνήθως είναι λιγότερες, απεικονίζοντας στη εκτιμώμενη επιλεκτικότητα της κάθε WHERE-περιορισμών συνθήκης, τα οποία δεν ανταποκρίνονται στον κόμβο αυτό. Ιδεατά, οι σειρές που βρίσκονται σε

υψηλό επίπεδο που εκτιμώνται θα προσεγγίζουν τον αριθμό των σειρών που όντως επιστρέφονται, ανανεώνονται είτε διαγράφονται από την επερώτηση.

3. Τέλος, να παρατηρήσουμε πως θα εξετάσουμε τα πλάνα εκτέλεσης των επερωτήσεων και για τα δυο σχήματα κωδικοποίησης στην περίπτωση που υιοθετήσουμε την δυνατότητα συμπίεσης μέσω της απορρόφησης των ετικετών στην περίπτωση των DAGs.

2. Πλάνα Εκτέλεσης Επερωτήσεων για την περίπτωση Δέντρων

2.1 Απόγονοι

1. Απόγονοι (Ριζικός κόμβος)

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where ((1 <= Y.post) and (Y.post < 253215))
```

b) Execution Plan:

NOTICE: QUERY PLAN:

Seq Scan on class y (cost=0.00..5411.18 rows=253161 width=4) (actual time=0.04..2669.65 rows=253215 loops=1)
Total runtime: 2869.39 msec

NOTICE: QUERY PLAN:

Seq Scan on class y (cost=0.00..5411.18 rows=253161 width=4) (actual time=0.04..2669.65 rows=253215 loops=1)
Total runtime: 2869.39 msec

c) Execution Time = 2.869392

d) Result tuples = 253214

B. Unicode Dewey

1.Απόγονοι (Ριζικός κόμβος)

a) SQL query

```
select Y.label
from UPrefix Y
where '1' < Y.label and Y.label < '2')
```

b) Execution Plan:

NOTICE: QUERY PLAN:

Seq Scan on class y (cost=0.00..5870.95 rows=253146 width=13) (actual time=0.12..3093.68 rows=253215 loops=1)
Total runtime: 3293.93 msec

NOTICE: QUERY PLAN:

Seq Scan on class y (cost=0.00..5870.95 rows=253146 width=13) (actual time=0.12..3093.68 rows=253215 loops=1)
Total runtime: 3293.93 msec

c) Execution Time = 3.296126

d) Result tuples = 253214

2. Απόγονοι (ένας κόμβος με μέσο όρο πλήθους απογόνων (= 25314 - Arts))

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where ((150765 <= Y.post) and (Y.post < 176077))
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3662.16 rows=1266 width=4) (actual time=0.10..405.61 rows=25314 loops=1)
Total runtime: 429.73 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3662.16 rows=1266 width=4) (actual time=0.10..405.61 rows=25314 loops=1)
Total runtime: 429.73 msec

c) Execution Time = 0.429730

d) Result tuples = 25315

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix Y
where ((12' <= Y.label) and (Y.label <= '13'))
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3905.63 rows=1266 width=32) (actual time=0.11..550.11 rows=25314 loops=1)
Total runtime: 577.02 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3905.63 rows=1266 width=32) (actual time=0.11..550.11 rows=25314 loops=1)
Total runtime: 577.02 msec

c) Execution Time = 0.57702

d) Result tuples = 25314

3. Απόγονοι (ένας κόμβος με τον ελάχιστο αριθμό απογόνων (= 1))

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where ((126251 <= Y.post) and (Y.post < 126252))
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3662.16 rows=1266 width=4) (actual time=0.10..0.14 rows=1 loops=1)
Total runtime: 0.27 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3662.16 rows=1266 width=4) (actual time=0.10..0.14 rows=1 loops=1)
Total runtime: 0.27 msec

c) Execution Time = 0.00027

d) Result tuples = 1

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix Y
where ('12f226af1d54' <= Y.label) and (Y.label < '12f226af1d55'))
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3905.63 rows=1266 width=32) (actual time=0.10..0.16 rows=1 loops=1)
Total runtime: 0.28 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3905.63 rows=1266 width=32) (actual time=0.10..0.16 rows=1 loops=1)
Total runtime: 0.28 msec

c) Execution Time = 0.00028

d) Result tuples = 1

2.2 Πρόγονοι

1. Αριστερός "Φύλλο" – κόμβος – με την μικρότερη τιμή του postnum (id = 1)

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where ((Y.index <= 1) and (1 < Y.post))
```

b) Execution Plan

NOTICE: QUERY PLAN:

Seq Scan on class y (cost=0.00..5411.18 rows=25 width=4) (actual time=1189.28..1189.30 rows=5 loops=1)
Total runtime: 1189.04 msec

NOTICE: QUERY PLAN:

Seq Scan on class y (cost=0.00..5411.18 rows=25 width=4) (actual time=1189.28..1189.30 rows=5 loops=1)
Total runtime: 1189.04 msec

c) Execution Time = 1.18904

d) Result tuples = 5

B. Unicode Dewey

a) SQL query

select *prefix*('121f23')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.63..1.63 rows=1 loops=1)
Total runtime: 1.63 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.63..1.63 rows=1 loops=1)
Total runtime: 1.63 msec

c) Execution Time = 0.00163

d) Result tuples = 5

2. Δεξιό "Φύλλο" κόμβος – με την μεγαλύτερη τιμή postnum (id = 253211)

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where ((Y.index <= 253211) and (253211 < Y.post))
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.41 rows=25 width=4) (actual time=0.09..0.13 rows=2 loops=1)
Total runtime: 0.27 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.41 rows=25 width=4) (actual time=0.09..0.13 rows=2 loops=1)
Total runtime: 0.27 msec

c) Execution Time = 0.00027

d) Result tuples = 4

B. Unicode Dewey

a) SQL query

select *prefix*('1G221')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.67..1.67 rows=1 loops=1)
Total runtime: 1.67 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.67..1.67 rows=1 loops=1)
Total runtime: 1.67 msec

c) Execution Time = 0.00167

d) Result tuples = 4

3. Μεσαίος "Φύλλο" – κόμβος (id = 126623)

A. Agrawal

a) SQL query

select Y.post
from PInterval Y
where ((Y.index <= 126623) and (126623 < Y.post))

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3266.01 rows=63207 width=4) (actual time=0.43..1547.76 rows=8 loops=1)
Total runtime: 1547.99 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3266.01 rows=63207 width=4) (actual time=0.43..1547.76 rows=8 loops=1)
Total runtime: 1547.99 msec

c) Execution Time = 1.54799

d) Result tuples = 9

B. Unicode Dewey

a) SQL query

select *prefix*('12f226a91c')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.78..1.78 rows=1 loops=1)
Total runtime: 1.63 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.78..1.78 rows=1 loops=1)
Total runtime: 1.78 msec

c) Execution Time = 0.00178

d) Result tuples = 9

2.3 Έλεγχος Υπαλληλίας

1. Ριζικός κόμβος – Αριστερός "Φύλλο" – κόμβος με την μικρότερη τιμή postnum (id = 1)

A. Agrawal

a) SQL query

select *class_subclass*(1,253214,1)

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.13..0.13 rows=1 loops=1)
Total runtime: 0.18 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.13..0.13 rows=1 loops=1)
Total runtime: 0.18 msec

c) Execution Time = 0.00018

d) Result tuples = 0

B. Unicode Dewey

a) SQL query

select *test_subclass*('1','121f23')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.12..0.12 rows=1 loops=1)
Total runtime: 0.17 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.12..0.12 rows=1 loops=1)
Total runtime: 0.17 msec

c) Execution Time = 0.00017

d) Result tuples = 0

2. Ριζικός κόμβος – δεξιό "Φύλλο" κόμβος – με την μεγαλύτερη τιμή postnum (id = 253211)

A. Agrawal

a) SQL query

select *class_subclass*(1,253214,253211)

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.12..0.12 rows=1 loops=1)
Total runtime: 0.17 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.12..0.12 rows=1 loops=1)
Total runtime: 0.17 msec

c) Execution Time = 0.00017

d) Result tuples = 0

B. Unicode Dewey

a) SQL query

select *test_subclass*('1','1G221')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.12..0.12 rows=1 loops=1)
Total runtime: 0.17 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.12..0.12 rows=1 loops=1)
Total runtime: 0.17 msec

c) Execution Time = 0.00017

d) Result tuples = 0

3. Ριζικός κόμβος – μεσαίος “Φύλλο” -κόμβος (id = 126623)

A. Agrawal

a) SQL query

select *class_subclass*(1,253214,126623)

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.13..0.13 rows=1 loops=1)
Total runtime: 0.18 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.13..0.13 rows=1 loops=1)
Total runtime: 0.18 msec

c) Execution Time = 0.00018

d) Result tuples = 0

B. Unicode Dewey

a) SQL query

select *test_subclass*('1','12f226a91c')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.12..0.12 rows=1 loops=1)
Total runtime: 0.17 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.12..0.12 rows=1 loops=1)
Total runtime: 0.17 msec

c) Execution Time = 0.00017

d) Result tuples = 0

2.3 "Φύλλα" - Κόμβοι

1. "Φύλλα" (Ριζικός κόμβος = 253215)

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where ((1 <= Y.post) and (Y.post < 253215) and (Y.index = Y.post))
```

b) Execution Plan:

NOTICE: QUERY PLAN:

Seq Scan on class y (cost=0.00..6044.21 rows=2 width=4) (actual time=0.05..2805.97 rows=190231 loops=1)
Total runtime: 2957.039 msec

NOTICE: QUERY PLAN:

Seq Scan on class y (cost=0.00..6044.21 rows=2 width=4) (actual time=0.05..2805.97 rows=190231 loops=1)
Total runtime: 2957.039 msec

c) Execution Time = 2.957039

d) Result tuples = 190231

B. Unicode Dewey

1. "Φύλλα" (Ριζικός κόμβος = 190231)

a) SQL query

```
select Y.label
from UPrefix Y
where ('12' <= Y. label) and (Y. label < '2') and NOT EXISTS
                                                    (select *
                                                    from UPrefix Z
                                                    where (Y.label <= Z.label) and (Z.label < (Y.label || 'xFF')))
```

b) Execution Plan:

NOTICE: QUERY PLAN:

Seq Scan on class y (cost=0.00..13955.10 rows=126573 width=13) (actual time=25540.04..25540.04 rows=0 loops=1)
SubPlan
-> Index Scan using index1 on class z (cost=0.00..40.42 rows=1266 width=25) (actual time=0.07..0.07 rows=1 loops=253215)
Total runtime: 25542.67 msec

NOTICE: QUERY PLAN:

Seq Scan on class y (cost=0.00..13955.10 rows=126573 width=13) (actual time=25540.04..25540.04 rows=0 loops=1)
SubPlan
-> Index Scan using index1 on class z (cost=0.00..40.42 rows=1266 width=25) (actual time=0.07..0.07 rows=1 loops=253215)
Total runtime: 25542.67 msec

c) Execution Time = 25.54267

d) Result tuples = 190231

2. "Φύλλα" (ένας κόμβος με έναν μέσο αριθμό από απογόνους (= 25314))

Ο μέσος αριθμός απογόνων είναι 4.4 και άρα προσεγγιστικά παίρνω την τιμή του 5 και βρίσκω έναν κόμβο στην μέση περίπου με έναν τέτοιο αριθμό απογόνων

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where ((150765 <= Y.post) and (Y.post < 176077) and (Y.index = Y.post))
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Index Scan using index1 on class y  (cost=0.00..762.24 rows=1 width=4) (actual time=0.35..472.52 rows=21419 loops=1)
Total runtime: 493.23 msec

NOTICE:  QUERY PLAN:

Index Scan using index1 on class y  (cost=0.00..762.24 rows=1 width=4) (actual time=0.35..472.52 rows=21419 loops=1)
Total runtime: 493.23 msec
```

c) Execution Time = 0.493230

d) Result tuples = 21419

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix Y
where ('1221' <= Y.label) and (Y.label < '13') and NOT EXISTS
                                                    (select *
                                                    from UPrefix Z
                                                    where Y.label <= Z.label and Z.label < (Y.label ||'xFF'))
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Index Scan using index1 on class y  (cost=0.00..9588.74 rows=79804 width=13) (actual time=16516.13..16516.13 rows=0 loops=1)
  SubPlan
    ->  Index Scan using index1 on class z  (cost=0.00..40.42 rows=1266 width=25) (actual time=0.07..0.07 rows=1 loops=158390)
Total runtime: 16519.63 msec

NOTICE:  QUERY PLAN:
```

Index Scan using index1 on class y (cost=0.00..9588.74 rows=79804 width=13) (actual time=16516.13..16516.13 rows=0 loops=1)
SubPlan
-> Index Scan using index1 on class z (cost=0.00..40.42 rows=1266 width=25) (actual time=0.07..0.07 rows=1 loops=158390)
Total runtime: 16519.63 msec

c) Execution Time = 16.519630

d) Result tuples = 21419

3. "Φύλλα" (ένας κόμβος με ελάχιστο αριθμό από απογόνους (= 1))

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where ((126251 <= Y.post) and (Y.post < 126252) and (Y.index = Y.post))
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.02 rows=1 width=4) (actual time=0.10..0.13 rows=1 loops=1)
Total runtime: 0.27 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.02 rows=1 width=4) (actual time=0.10..0.13 rows=1 loops=1)
Total runtime: 0.27 msec

c) Execution Time = 0.00027

d) Result tuples = 1

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix Y
where ('12f226af1d54' <= Y.label) and (Y.label < '12f226af1d55') and NOT EXISTS
                                                                    (select *
                                                                    from UPrefix Z
                                                                    where (Y.label <= Z.label) and
                                                                    (Z.label < (Y.label ||'xFF'))
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.05 rows=1 width=13) (actual time=0.31..0.31 rows=0 loops=1)
SubPlan
-> Index Scan using index1 on class z (cost=0.00..40.42 rows=1266 width=25) (actual time=0.08..0.08 rows=1 loops=2)
Total runtime: 0.55 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.05 rows=1 width=13) (actual time=0.31..0.31 rows=0 loops=1)
SubPlan
-> Index Scan using index1 on class z (cost=0.00..40.42 rows=1266 width=25) (actual time=0.08..0.08 rows=1 loops=2)
Total runtime: 0.55 msec

c) Execution Time = 0.00055

d) Result tuples = 1

2.5 Ομοεπίπεδοι (Siblings)

A. Agrawal

1. Ένας κόμβος με το μέγιστο εύρος (=314)

a) SQL query

```
select Y.post
from PInterval X, PInterval Y
where (Y.post != 206712) and (X.father_post = Y.father_post) and (X.post = 206712))
```

b) Execution Plan

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.60 rows=28 width=12) (actual time=0.18..8.07 rows=313 loops=1)
-> Index Scan using index1 on class x (cost=0.00..3.01 rows=1 width=4) (actual time=0.08..0.09 rows=1 loops=1)
-> Index Scan using index2 on class y (cost=0.00..3.33 rows=20 width=8) (actual time=0.08..6.33 rows=313 loops=1)
Total runtime: 8.63 msec

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.60 rows=28 width=12) (actual time=0.18..8.07 rows=313 loops=1)
-> Index Scan using index1 on class x (cost=0.00..3.01 rows=1 width=4) (actual time=0.08..0.09 rows=1 loops=1)
-> Index Scan using index2 on class y (cost=0.00..3.33 rows=20 width=8) (actual time=0.08..6.33 rows=313 loops=1)
Total runtime: 8.63 msec

c) Execution Time = 0.00863

d) Result tuples = 313

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix X, UPrefix Y
where (Y.label != '1522c143416d') and (X.father_label = Y.father_label) and (X.label ='1522c143416d')
```

b) Execution Plan

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.95 rows=29 width=37) (actual time=0.20..9.03 rows=313 loops=1)
-> Index Scan using index1 on class x (cost=0.00..3.01 rows=1 width=12) (actual time=0.09..0.10 rows=1 loops=1)
-> Index Scan using index2 on class y (cost=0.00..3.46 rows=21 width=25) (actual time=0.09..7.07 rows=313 loops=1)
Total runtime: 9.61 msec

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.95 rows=29 width=37) (actual time=0.20..9.03 rows=313 loops=1)
-> Index Scan using index1 on class x (cost=0.00..3.01 rows=1 width=12) (actual time=0.09..0.10 rows=1 loops=1)
-> Index Scan using index2 on class y (cost=0.00..3.46 rows=21 width=25) (actual time=0.09..7.07 rows=313 loops=1)
Total runtime: 9.61 msec

c) Execution Time = 0.00961

d) Result tuples = 313

2. Ένας κόμβος με το μέσο όρο εύρους (breadth) και με το μέγιστο αριθμό απογόνων

A. Agrawal

a) SQL query

```
select Y.post
```

```
from PInterval X, PInterval Y
where (Y.post != 133209) and (X.father_post = Y.father_post) and (X.post = 133209)
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Nested Loop  (cost=0.00..6.60 rows=28 width=12) (actual time=0.21..0.28 rows=3 loops=1)
  ->  Index Scan using index1 on class x  (cost=0.00..3.01 rows=1 width=4) (actual time=0.09..0.10 rows=1
loops=1)
  ->  Index Scan using index2 on class y  (cost=0.00..3.33 rows=20 width=8) (actual time=0.10..0.15 rows=3
loops=1)
Total runtime: 0.54 msec

NOTICE:  QUERY PLAN:

Nested Loop  (cost=0.00..6.60 rows=28 width=12) (actual time=0.21..0.28 rows=3 loops=1)
  ->  Index Scan using index1 on class x  (cost=0.00..3.01 rows=1 width=4) (actual time=0.09..0.10 rows=1
loops=1)
  ->  Index Scan using index2 on class y  (cost=0.00..3.33 rows=20 width=8) (actual time=0.10..0.15 rows=3
loops=1)
Total runtime: 0.54 msec
```

c) Execution Time = 0.00054

d) Result tuples = 3

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix X, UPrefix Y
where (Y.label != '12f2cb181db2') and (X.father_label = Y. father_label) and (X.label = '12f2cb181db2')
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Nested Loop  (cost=0.00..6.95 rows=29 width=37) (actual time=0.20..0.31 rows=3 loops=1)
  ->  Index Scan using index1 on class x  (cost=0.00..3.01 rows=1 width=12) (actual time=0.10..0.11 rows=1
loops=1)
  ->  Index Scan using index2 on class y  (cost=0.00..3.46 rows=21 width=25) (actual time=0.09..0.17
rows=3 loops=1)
Total runtime: 0.56 msec

NOTICE:  QUERY PLAN:

Nested Loop  (cost=0.00..6.95 rows=29 width=37) (actual time=0.20..0.31 rows=3 loops=1)
```

-> Index Scan using index1 on class x (cost=0.00..3.01 rows=1 width=12) (actual time=0.10..0.11 rows=1 loops=1)
-> Index Scan using index2 on class y (cost=0.00..3.46 rows=21 width=25) (actual time=0.09..0.17 rows=3 loops=1)
Total runtime: 0.56 msec

c) Execution Time = 0.00056

d) Result tuples = 3

3. Ένας κόμβος με το ελάχιστο εύρος (=1)

A. Agrawal

Βρίσκεται στην μέση της ιεραρχίας και έχει το ελάχιστο εύρος (=1)

a) SQL query

```
select Y.post
from PInterval X, PInterval Y
where (Y.post != 126609) and (X.father_post = Y. father_post) and (X.post = 126609)
```

b) Execution Plan

Nested Loop (cost=0.00..6.60 rows=28 width=12) (actual time=0.20..0.22 rows=1 loops=1)
-> Index Scan using index1 on class x (cost=0.00..3.01 rows=1 width=4) (actual time=0.09..0.10 rows=1 loops=1)
-> Index Scan using index2 on class y (cost=0.00..3.33 rows=20 width=8) (actual time=0.09..0.10 rows=1 loops=1)
Total runtime: 0.47 msec

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.60 rows=28 width=12) (actual time=0.20..0.22 rows=1 loops=1)
-> Index Scan using index1 on class x (cost=0.00..3.01 rows=1 width=4) (actual time=0.09..0.10 rows=1 loops=1)
-> Index Scan using index2 on class y (cost=0.00..3.33 rows=20 width=8) (actual time=0.09..0.10 rows=1 loops=1)
Total runtime: 0.47 msec

c) Execution Time = 0.00047

d) Result tuples = 1

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix X, UPrefix Y
```

where (Y.label !='12f226aa1') and (X.father_label = Y.father_label) and (X.label = '12f226aa1')

b) Execution Plan

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.95 rows=29 width=37) (actual time=0.20..0.24 rows=1 loops=1)
-> Index Scan using index1 on class x (cost=0.00..3.01 rows=1 width=12) (actual time=0.10..0.11 rows=1 loops=1)
-> Index Scan using index2 on class y (cost=0.00..3.46 rows=21 width=25) (actual time=0.09..0.12 rows=1 loops=1)
Total runtime: 0.49 msec

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.95 rows=29 width=37) (actual time=0.20..0.24 rows=1 loops=1)
-> Index Scan using index1 on class x (cost=0.00..3.01 rows=1 width=12) (actual time=0.10..0.11 rows=1 loops=1)
-> Index Scan using index2 on class y (cost=0.00..3.46 rows=21 width=25) (actual time=0.09..0.12 rows=1 loops=1)
Total runtime: 0.49 msec

c) Execution Time = 0.00049

d) Result tuples = 1

2.6 Πρότεροι (Precceedings)

1. Το μεγαλύτερο postnum (253211)

A. Agrawal

a) SQL query

select Y.post
from PInterval Y
where (Y.post <= 253211)

b) Execution Plan

NOTICE: QUERY PLAN:

Seq Scan on class y (cost=0.00..4778.15 rows=253187 width=4) (actual time=0.12..2417.95 rows=253211 loops=1)
Total runtime: 2629.188 msec

NOTICE: QUERY PLAN:

Seq Scan on class y (cost=0.00..4778.15 rows=253187 width=4) (actual time=0.12..2417.95 rows=253211 loops=1)

Total runtime: 2629.188 msec

c) Execution Time = 2.629188

d) Result tuples = 253211

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix Y
where Y.label <= '1G220' and Y.label > '1G22'
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.02 rows=1 width=13) (actual time=0.08..1653.24 rows=253211 loops=1)
Total runtime: 2874.457 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.02 rows=1 width=13) (actual time=0.08..1653.24 rows=253211 loops=1)
Total runtime: 2874.457 msec

c) Execution Time = 2.874457

d) Result tuples = 253211

2. Το μικρότερο postnum (1)

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where (Y.post < 1)
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.34 rows=25 width=4) (actual time=0.07..0.10 rows=2 loops=1)
Total runtime: 0.24 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.34 rows=25 width=4) (actual time=0.07..0.10 rows=2 loops=1)
Total runtime: 0.24 msec

c) Execution Time = 0.00024

d) Result tuples = 0

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix Y
where Y.label <= '121f21' and Y.label > '121f2'
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3905.63 rows=1266 width=32) (actual time=0.10..0.35 rows=14 loops=1)
Total runtime: 0.50 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3905.63 rows=1266 width=32) (actual time=0.10..0.35 rows=14 loops=1)
Total runtime: 0.50 msec

c) Execution Time = 0.00050

d) Result tuples = 0

3. Το μεσαίο postnum (126623)

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where (Y.post <= 126263)
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..2929.81 rows=129467 width=4) (actual time=0.38..2154.26 rows=126623 loops=1)
Total runtime: 2276.29 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..2929.81 rows=129467 width=4) (actual time=0.38..2154.26 rows=126623 loops=1)
Total runtime: 2276.29 msec

c) Execution Time = 2.27629

d) Result tuples = 126623

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix Y
where Y.label <= '12f226a91b' and Y.label > '12f226a91'
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3905.63 rows=1266 width=32) (actual time=0.11..1.33 rows=126623 loops=1)
Total runtime: 2345.29 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3905.63 rows=1266 width=32) (actual time=0.11..1.33 rows=126623 loops=1)
Total runtime: 2345.29 msec

c) Execution Time = 2.34529

d) Result tuples = 126623

2.7 Επόμενοι (Followings)

1. Μεγαλύτερο postnum

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
```

where Y.post > 253211 and Y.index > 235211

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.41 rows=2 width=4) (actual time=0.10..0.10 rows=0 loops=1)
Total runtime: 0.24 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.41 rows=2 width=4) (actual time=0.10..0.10 rows=0 loops=1)
Total runtime: 0.24 msec

c) Execution Time = 0.00024

d) Result tuples = 0

B. Unicode Dewey

a) SQL query

select Y.label
from UPrefix Y
where Y.label >= '1G222' and Y.label < '1G23'

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.02 rows=1 width=13) (actual time=0.09..0.10 rows=1 loops=1)
Total runtime: 0.24 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class y (cost=0.00..3.02 rows=1 width=13) (actual time=0.09..0.10 rows=1 loops=1)
Total runtime: 0.24 msec

c) Execution Time = 0.00024

d) Result tuples = 0

2. Μικρότερο postnum

a) SQL query

```
select Y.post
from PInterval Y
where Y.post > 1 and Y.index > 1
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Seq Scan on class y  (cost=0.00..5411.18 rows=253162 width=4) (actual time=0.13..2775.72 rows=253209 loops=1)
Total runtime: 2972.10 msec

NOTICE:  QUERY PLAN:

Seq Scan on class y  (cost=0.00..5411.18 rows=253162 width=4) (actual time=0.13..2775.72 rows=253209 loops=1)
Total runtime: 2972.10 msec
```

c) Execution Time = 2.9210

d) Result tuples = 253209

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix Y
where Y.label >= '121f23' and Y.label < '121f3'
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Index Scan using index1 on class y  (cost=0.00..3774.04 rows=54643 width=13) (actual time=0.40..3255.88 rows=253209 loops=1)
Total runtime: 3378.25 msec

NOTICE:  QUERY PLAN:

Index Scan using index1 on class y  (cost=0.00..3774.04 rows54463=3 width=13) (actual time=0.40..3255.88 rows=253209 loops=1)
Total runtime: 3378.25 msec
```

c) Execution Time = 3.378250

d) Result tuples = 253209

3. Μεσαίο postnum

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where Y.post > 126623 and Y.index > 126623
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Index Scan using index1 on class y  (cost=0.00..3110.92 rows=60460 width=4) (actual time=0.45..2362.72
rows=126582 loops=1)
Total runtime: 2487.696 msec

NOTICE:  QUERY PLAN:

Index Scan using index1 on class y  (cost=0.00..3110.92 rows=60460 width=4) (actual time=0.45..2362.72
rows=126582 loops=1)
Total runtime: 2487.696 msec
```

c) Execution Time = 2.487696

d) Result tuples = 126582

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix Y
where Y.label >= '12f226a91d' and Y.label < '12f226a92'");
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Index Scan using index1 on class y  (cost=0.00..1543.02 rows=35201 width=13) (actual time=0.63..1752.15
rows=126582 loops=1)
Total runtime: 2566.292 msec

NOTICE:  QUERY PLAN:

Index Scan using index1 on class y  (cost=0.00..1543.02 rows=35201 width=13) (actual time=0.63..1752.15
rows=126582 loops=1)
Total runtime: 2566.292 msec
```

c) Execution Time = 2.566292

d) Result tuples = 126582

2.8 Lower Upper Bound

1. Αριστερός "Φύλλο" κόμβος – κόμβος με την μικρότερη τιμή postnum – Δεξιός "Φύλλο" κόμβος – κόμβος με την μεγαλύτερη τιμή postnum

A. Agrawal

a) SQL query

```
select Z.post
from PInterval Z
where (Z.index <= 1) and (253211 <Z.post) and NOT EXISTS
                                                (select *
                                                from PInterval W
                                                where (W.index <= 1) and (253211 < W.post) and
                                                (Z.index <= W.post) and (W.post < Z.post))
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Index Scan using index1 on class z (cost=0.00..5277096.18 rows=6 width=4) (actual time=3227.08..3227.09 rows=1 loops=1)
  SubPlan
    -> Index Scan using index1 on class w (cost=0.00..40.59 rows=1 width=12) (actual time=1650.70..1650.70 rows=0 loops=1)
Total runtime: 3227.42 msec

NOTICE:  QUERY PLAN:

Index Scan using index1 on class z (cost=0.00..5277096.18 rows=6 width=4) (actual time=3227.08..3227.09 rows=1 loops=1)
  SubPlan
    -> Index Scan using index1 on class w (cost=0.00..40.59 rows=1 width=12) (actual time=1650.70..1650.70 rows=0 loops=1)
Total runtime: 3227.42 msec
```

c) Execution Time = 3.227420

d) Result tuples = 1

B. Unicode Dewey

a) SQL query

```
select nca('121f23','1G221')
```

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.48..0.48 rows=1 loops=1)
Total runtime: 0.53 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.48..0.48 rows=1 loops=1)
Total runtime: 0.53 msec

c) Execution Time = 0.00053

d) Result tuples = 1

2. **Αριστερός "Φύλλο" κόμβος – κόμβος με την μικρότερη τιμή postnum – Μεσαίος "Φύλλο" κόμβος**

A. Agrawal

a) SQL query

```
select Z.post
from PInterval Z
where (Z.index <= 1) and (126623 <Z.post) and NOT EXISTS
                                                (select *
                                                from PInterval W
                                                where (W.index <= 1) and (126623 < W.post) and
                                                (Z.index <= W.post) and (W.post < Z.post))
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class z (cost=0.00..1031.12 rows=1 width=4) (actual time=0.16..0.17 rows=1 loops=1)
SubPlan
-> Index Scan using index1 on class w (cost=0.00..40.59 rows=1 width=12) (actual time=0.06..0.06 rows=0 loops=1)
Total runtime: 0.41 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class z (cost=0.00..1031.12 rows=1 width=4) (actual time=0.16..0.17 rows=1 loops=1)
SubPlan
-> Index Scan using index1 on class w (cost=0.00..40.59 rows=1 width=12) (actual time=0.06..0.06 rows=0 loops=1)
Total runtime: 0.41 msec

c) Execution Time = 0.00041

d) Result tuples = 1

B. Unicode Dewey

a) SQL query

select *nca*('121f23','12f226a91c')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.53..0.53 rows=1 loops=1)
Total runtime: 0.57 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.53..0.53 rows=1 loops=1)
Total runtime: 0.57 msec

c) Execution Time = 0.00057

d) Result tuples = 1

3. Μεσαίος "Φύλλο" κόμβος - Δεξιός "Φύλλο" κόμβος – κόμβος με την μεγαλύτερη τιμή postnum

A. Agrawal

a) SQL query

```
select Z.post
from PInterval Z
where (Z.index <= 126623) and (253211 < Z.post) and NOT EXISTS
                                                    (select *
                                                    from PInterval W
                                                    where (W.index <= 126623) and (253211 < W.post) and
                                                    (Z.index <= W.post) and (W.post < Z.post))
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on class z (cost=0.00..5.07 rows=6 width=4) (actual time=0.17..0.18 rows=1 loops=1)
SubPlan
-> Index Scan using index1 on class w (cost=0.00..40.59 rows=616 width=12) (actual time=0.06..0.06 rows=0 loops=1)

Total runtime: 0.438 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on class z (cost=0.00..5.07 rows=6 width=4) (actual time=0.17..0.18 rows=1 loops=1)
SubPlan
-> Index Scan using index1 on class w (cost=0.00..40.59 rows=616 width=12) (actual time=0.06..0.06 rows=0 loops=1)
Total runtime: 0.438 msec

c) Execution Time = 0.000438

d) Result tuples = 1

B. Unicode Dewey

a) SQL query

select *nca*('12f226a91c','1G221')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.48..0.49 rows=1 loops=1)
Total runtime: 0.53 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.48..0.49 rows=1 loops=1)
Total runtime: 0.53 msec

c) Execution Time = 0.00053

d) Result tuples = 1

3. Πλάνα Εκτέλεσης Επερωτήσεων για την περίπτωση DAGs

3.1 Απόγονοι

1. Απόγονοι (Ριζικός κόμβος)

A. Agrawal

a) SQL query

```
(select Y.post
from PInterval Y
where 1 <= Y.post and Y.post < 253215)
UNION ALL
(select X.post
from PInterval X, DPInterval O
where O.ancestor = 253215 and X.post <= O.post and X.post >= O.index)
```

b) Execution Plan:

```
NOTICE:  QUERY PLAN:

-> Append  (cost=0.00..29826.85 rows=860348 width=12) (actual time=0.07..4830.34 rows=253229 loops=1)
        -> Subquery Scan *SELECT* 1  (cost=0.00..5411.18 rows=253161 width=4) (actual time=0.07..4498.90 rows=253214 loops=1)
                -> Seq Scan on ba y  (cost=0.00..5411.18 rows=253161 width=4) (actual time=0.06..3090.36 rows=253214 loops=1)
        -> Subquery Scan *SELECT* 2  (cost=0.00..24415.67 rows=607187 width=12) (actual time=0.29..2.51 rows=18 loops=1)
                -> Nested Loop  (cost=0.00..24415.67 rows=607187 width=12) (actual time=0.28..2.38 rows=18 loops=1)
                        -> Index Scan using index4 on oa o  (cost=0.00..3.29 rows=22 width=8) (actual time=0.14..0.54 rows=18 loops=1)
                        -> Index Scan using index1 on ba x  (cost=0.00..709.15 rows=28135 width=4) (actual time=0.07..0.08 rows=1 loops=18)
Total runtime: 11929.880 msec

NOTICE:  QUERY PLAN:

-> Append  (cost=0.00..29826.85 rows=860348 width=12) (actual time=0.07..4830.34 rows=253229 loops=1)
        -> Subquery Scan *SELECT* 1  (cost=0.00..5411.18 rows=253161 width=4) (actual time=0.07..4498.90 rows=253211 loops=1)
                -> Seq Scan on ba y  (cost=0.00..5411.18 rows=253161 width=4) (actual time=0.06..3090.36 rows=253211 loops=1)
        -> Subquery Scan *SELECT* 2  (cost=0.00..24415.67 rows=607187 width=12) (actual time=0.29..2.51 rows=18 loops=1)
```

-> Nested Loop (cost=0.00..24415.67 rows=607187 width=12) (actual time=0.28..2.38 rows=18 loops=1)
-> Index Scan using index4 on oa o (cost=0.00..3.29 rows=22 width=8) (actual time=0.14..0.54 rows=18 loops=1)
-> Index Scan using index1 on ba x (cost=0.00..709.15 rows=28135 width=4) (actual time=0.07..0.08 rows=1 loops=18)
Total runtime: 11929.880 msec

c) Execution Time = 11.929880

d) Result tuples = 253204

B. Unicode Dewey

1. Απόγονοι (Ριζικός κόμβος)

a) SQL query

```
(select Y.label
from UPrefix Y
where Y.label > '1' and Y.label < '1'||'xFF')
UNION ALL
(select W.label
from DUPrefix W, (      select X.label as T
                        from UPrefix X
                        where ((X.label >= '1') and (X.label <'1'||'xFF'))) Z
where W.ancestor = Z.T)
```

b) Execution Plan:

NOTICE: QUERY PLAN:

-> Append (cost=0.00..64394.79 rows=999322 width=33) (actual time=0.17..47041.13 rows=849230 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..5858.16 rows=28135 width=32) (actual time=0.17..5536.41 rows=253214 loops=1)
-> Seq Scan on bd y (cost=0.00..5858.16 rows=28135 width=32) (actual time=0.15..3929.69 rows=253214 loops=1)
-> Subquery Scan *SELECT* 2 (cost=38099.20..58536.63 rows=971188 width=33) (actual time=20688.04..40269.22 rows=596019 loops=1)
-> Merge Join (cost=38099.20..58536.63 rows=971188 width=33) (actual time=20688.03..36575.32 rows=596019 loops=1)
-> Index Scan using index3 on od x (cost=0.00..7031.64 rows=253165 width=12) (actual time=0.54..5336.74 rows=253159 loops=1)
-> Sort (cost=38099.20..38099.20 rows=253215 width=21) (actual time=20621.31..23426.17 rows=654716 loops=1)
-> Seq Scan on od w (cost=0.00..4533.15 rows=253215 width=21) (actual time=0.03..2779.28 rows=253215 loops=1)
Total runtime: 67121.239 msec

NOTICE: QUERY PLAN:

-> Append (cost=0.00..64394.79 rows=999322 width=33) (actual time=0.17..47041.13 rows=849230 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..5858.16 rows=28135 width=32) (actual time=0.17..5536.41 rows=253214 loops=1)
-> Seq Scan on bd y (cost=0.00..5858.16 rows=28135 width=32) (actual time=0.15..3929.69 rows=253214 loops=1)
-> Subquery Scan *SELECT* 2 (cost=38099.20..58536.63 rows=971188 width=33) (actual time=20688.04..40269.22 rows=596019 loops=1)
-> Merge Join (cost=38099.20..58536.63 rows=971188 width=33) (actual time=20688.03..36575.32 rows=596019 loops=1)
-> Index Scan using index3 on od x (cost=0.00..7031.64 rows=253165 width=12) (actual time=0.54..5336.74 rows=253159 loops=1)
-> Sort (cost=38099.20..38099.20 rows=253215 width=21) (actual time=20621.31..23426.17 rows=654716 loops=1)
-> Seq Scan on od w (cost=0.00..4533.15 rows=253215 width=21) (actual time=0.03..2779.28 rows=253215 loops=1)
Total runtime: 67121.239 msec

c) Execution Time = 67.121239

d) Result tuples = 253214

2. Απόγονοι (ένας κόμβος με μέσο αριθμό απογόνων (=25314))

A. Agrawal

a) SQL query

```
(select Y.post
from PInterval Y
where 150765 <= Y.post and Y.post < 176077)
UNION ALL
(select X.post
from PInterval X, DPInterval O
where O.ancestor = 176077 and X.post <= O.post and X.post >= O.index)
```

b) Execution Plan

NOTICE: QUERY PLAN:

-> Append (cost=0.00..24230.60 rows=607727 width=12) (actual time=0.44..322.86 rows=25581 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..347.65 rows=13791 width=4) (actual time=0.44..303.87 rows=25314 loops=1)
-> Index Scan using index1 on ba y (cost=0.00..347.65 rows=13791 width=4) (actual time=0.43..232.98 rows=25314 loops=1)
-> Subquery Scan *SELECT* 2 (cost=0.00..23882.95 rows=593937 width=12) (actual time=0.86..15.69 rows=267 loops=1)

-> Nested Loop (cost=0.00..23882.95 rows=593937 width=12) (actual time=0.86..10.47 rows=267 loops=1)
-> Index Scan using index4 on oa (cost=0.00..3.28 rows=21 width=8) (actual time=0.32..0.82 rows=267 loops=1)
-> Index Scan using index1 on ba x (cost=0.00..709.15 rows=28135 width=4) (actual time=0.32..0.32 rows=1 loops=1)
Total runtime: 617.279 msec

NOTICE: QUERY PLAN:

-> Append (cost=0.00..24230.60 rows=607727 width=12) (actual time=0.44..322.86 rows=25581 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..347.65 rows=13791 width=4) (actual time=0.44..303.87 rows=25314 loops=1)
-> Index Scan using index1 on ba y (cost=0.00..347.65 rows=13791 width=4) (actual time=0.43..232.98 rows=25314 loops=1)
-> Subquery Scan *SELECT* 2 (cost=0.00..23882.95 rows=593937 width=12) (actual time=0.86..15.69 rows=267 loops=1)
-> Nested Loop (cost=0.00..23882.95 rows=593937 width=12) (actual time=0.86..10.47 rows=267 loops=1)
-> Index Scan using index4 on oa (cost=0.00..3.28 rows=21 width=8) (actual time=0.32..0.82 rows=267 loops=1)
-> Index Scan using index1 on ba x (cost=0.00..709.15 rows=28135 width=4) (actual time=0.32..0.32 rows=1 loops=1)
Total runtime: 617.279 msec

c) Execution Time = 0.617279

d) Result tuples = 25581

B. Unicode Dewey

a) SQL query

```
(select Y.label f
from UPrefix Y
where Y.label > '122' and Y.label < '122'||'xFF')
UNION ALL
(select W.label
from DUPrefix W, (      select X.label as T
                        from UPrefix X
                        where ((X.label >= '122') and (X.label <'122'||'xFF'))) Z
where W.ancestor = Z.T)
```

b) Execution Plan

NOTICE: QUERY PLAN:

-> Append (cost=0.00..49420.66 rows=264951 width=33) (actual time=233.65..22416.32 rows=25581 loops=1)

-> Subquery Scan *SELECT* 1 (cost=0.00..5858.16 rows=28135 width=32) (actual time=233.64..1922.80 rows=25314 loops=1)
-> Seq Scan on bd y (cost=0.00..5858.16 rows=28135 width=32) (actual time=233.62..1921.19 rows=25314 loops=1)
-> Subquery Scan *SELECT* 2 (cost=38099.20..43562.49 rows=236817 width=33) (actual time=20493.14..20493.14 rows=267 loops=1)
-> Merge Join (cost=38099.20..43562.49 rows=236817 width=33) (actual time=20493.13..20493.13 rows=267 loops=1)
-> Sort (cost=38099.20..38099.20 rows=253215 width=21) (actual time=20492.60..20492.60 rows=1 loops=1)
-> Seq Scan on od w (cost=0.00..4533.15 rows=253215 width=21) (actual time=0.10..2771.84 rows=253215 loops=1)
-> Index Scan using index3 on od x (cost=0.00..1715.72 rows=61732 width=12) (actual time=0.52..0.52 rows=267 loops=1)
Total runtime: 21786.647 msec

NOTICE: QUERY PLAN:

-> Append (cost=0.00..49420.66 rows=264951 width=33) (actual time=233.65..22416.32 rows=25581 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..5858.16 rows=28135 width=32) (actual time=233.64..1922.80 rows=25314 loops=1)
-> Seq Scan on bd y (cost=0.00..5858.16 rows=28135 width=32) (actual time=233.62..1921.19 rows=25314 loops=1)
-> Subquery Scan *SELECT* 2 (cost=38099.20..43562.49 rows=236817 width=33) (actual time=20493.14..20493.14 rows=267 loops=1)
-> Merge Join (cost=38099.20..43562.49 rows=236817 width=33) (actual time=20493.13..20493.13 rows=267 loops=1)
-> Sort (cost=38099.20..38099.20 rows=253215 width=21) (actual time=20492.60..20492.60 rows=1 loops=1)
-> Seq Scan on od w (cost=0.00..4533.15 rows=253215 width=21) (actual time=0.10..2771.84 rows=253215 loops=1)
-> Index Scan using index3 on od x (cost=0.00..1715.72 rows=61732 width=12) (actual time=0.52..0.52 rows=267 loops=1)
Total runtime: 21786.647 msec

c) Execution Time = 21,786647

d) Result tuples = 25581

3. Απόγονοι (ένας κόμβος με τον ελάχιστο αριθμό απογόνων (= 1))

A. Agrawal

a) SQL query

```
(select Y.post
from PInterval Y
where 126251 <= Y.post and Y.post <= 126252)
UNION ALL
```

```
(select X.post
from PInterval X, DPInterval O
where O.ancestor = 126252 and X.post <= O.post and X.post >= O.index)
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

-> Append  (cost=0.00..24375.63 rows=606117 width=12) (actual time=0.11..0.19 rows=2 loops=1)
        -> Subquery Scan *SELECT* 1  (cost=0.00..3.02 rows=1 width=4) (actual time=0.10..0.12 rows=1
loops=1)
            -> Index Scan using index1 on ba y  (cost=0.00..3.02 rows=1 width=4) (actual time=0.09..0.11
rows=1 loops=1)
        -> Subquery Scan *SELECT* 2  (cost=0.00..24372.62 rows=606116 width=12) (actual
time=0.06..0.06 rows=0 loops=1)
            -> Nested Loop  (cost=0.00..24372.62 rows=606116 width=12) (actual time=0.06..0.08 rows=1
loops=1)
                -> Index Scan using index4 on oa  (cost=0.00..3.29 rows=22 width=8) (actual
time=0.06..0.08 rows=1 loops=1)
                -> Index Scan using index1 on ba x  (cost=0.00..709.15 rows=28135 width=4) (actual
time=0.07..0.08 rows=1 loops=1)
Total runtime: 0.263 msec

NOTICE:  QUERY PLAN:

-> Append  (cost=0.00..24375.63 rows=606117 width=12) (actual time=0.11..0.19 rows=2 loops=1)
        -> Subquery Scan *SELECT* 1  (cost=0.00..3.02 rows=1 width=4) (actual time=0.10..0.12 rows=1
loops=1)
            -> Index Scan using index1 on ba y  (cost=0.00..3.02 rows=1 width=4) (actual time=0.09..0.11
rows=1 loops=1)
        -> Subquery Scan *SELECT* 2  (cost=0.00..24372.62 rows=606116 width=12) (actual
time=0.06..0.06 rows=0 loops=1)
            -> Nested Loop  (cost=0.00..24372.62 rows=606116 width=12) (actual time=0.06..0.08 rows=1
loops=1)
                -> Index Scan using index4 on oa  (cost=0.00..3.29 rows=22 width=8) (actual
time=0.06..0.08 rows=1 loops=1)
                -> Index Scan using index1 on ba x  (cost=0.00..709.15 rows=28135 width=4) (actual
time=0.07..0.08 rows=1 loops=1)
Total runtime: 0.263 msec
```

c) Execution Time = 0.000263

d) Result tuples = 2

B. Unicode Dewey

a) SQL query

```
(select Y.label
from UPrefix Y
```

```
where Y.label > '12f226af1d54' and Y.label < '12f226af1d54'||'xFF')
UNION ALL
(select W.label
from DUPrefix W, (      select X.label as T
                        from UPrefix X
                        where ((X.label >= '12f226af1d54') and (X.label <'12f226af1d54'||'xFF'))) Z
where W.ancestor = Z.T)
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

-> Append (cost=0.00..44097.96 rows=231013 width=33) (actual time=20500.55..20500.55 rows=2
loops=1)
        -> Subquery Scan *SELECT* 1 (cost=0.00..1072.86 rows=38179 width=13) (actual
time=0.51..0.51 rows=1 loops=1)
            -> Index Scan using index1 on bd y (cost=0.00..1072.86 rows=38179 width=13) (actual
time=0.50..0.50 rows=1 loops=1)
        -> Subquery Scan *SELECT* 2 (cost=38099.20..43025.09 rows=192834 width=33) (actual
time=20500.03..20500.03 rows=1 loops=1)
            -> Merge Join (cost=38099.20..43025.09 rows=192834 width=33) (actual
time=20500.02..20500.02 rows=1 loops=1)
                -> Sort (cost=38099.20..38099.20 rows=253215 width=21) (actual
time=20499.58..20499.58 rows=1 loops=1)
                    -> Seq Scan on od w (cost=0.00..4533.15 rows=253215 width=21) (actual
time=0.13..2775.57 rows=253215 loops=1)
                -> Index Scan using index3 on od x (cost=0.00..1727.10 rows=62131 width=12) (actual
time=0.42..0.42 rows=2 loops=1)
Total runtime: 20440.89 msec

NOTICE:  QUERY PLAN:

-> Append (cost=0.00..44097.96 rows=231013 width=33) (actual time=20500.55..20500.55 rows=2
loops=1)
        -> Subquery Scan *SELECT* 1 (cost=0.00..1072.86 rows=38179 width=13) (actual
time=0.51..0.51 rows=1 loops=1)
            -> Index Scan using index1 on bd y (cost=0.00..1072.86 rows=38179 width=13) (actual
time=0.50..0.50 rows=1 loops=1)
        -> Subquery Scan *SELECT* 2 (cost=38099.20..43025.09 rows=192834 width=33) (actual
time=20500.03..20500.03 rows=1 loops=1)
            -> Merge Join (cost=38099.20..43025.09 rows=192834 width=33) (actual
time=20500.02..20500.02 rows=1 loops=1)
                -> Sort (cost=38099.20..38099.20 rows=253215 width=21) (actual
time=20499.58..20499.58 rows=1 loops=1)
                    -> Seq Scan on od w (cost=0.00..4533.15 rows=253215 width=21) (actual
time=0.13..2775.57 rows=253215 loops=1)
                -> Index Scan using index3 on od x (cost=0.00..1727.10 rows=62131 width=12) (actual
time=0.42..0.42 rows=2 loops=1)
Total runtime: 20440.89 msec
```

c) Execution Time = 20.440890

d) Result tuples = 2

3.2 Άμεσοι Απόγονοι

1. Μεγάλο postnum

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where (Y.father_post = 253215)
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index2 on ba y (cost=0.00..3.24 rows=18 width=4) (actual time=0.18..0.18 rows=17 loops=1)
Total runtime: 0.20 msec

NOTICE: QUERY PLAN:

Index Scan using index2 on ba y (cost=0.00..3.24 rows=18 width=4) (actual time=0.18..0.18 rows=17 loops=1)
Total runtime: 0.20 msec

c) Execution Time = 0.00020

d) Result tuples = 17

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix Y
where Y.father_label
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index2 on bd y (cost=0.00..3.27 rows=20 width=13) (actual time=0.10..0.39 rows=17 loops=1)
Total runtime: 0.56msec

NOTICE: QUERY PLAN:

Index Scan using index2 on bd y (cost=0.00..3.27 rows=20 width=13) (actual time=0.10..0.39 rows=17 loops=1)
Total runtime: 0.56msec

c) Execution Time = 0.00056

d) Result tuples = 17

2. Μεσαίο Postnum

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where (Y.father_post = 176077)
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index2 on ba y (cost=0.00..3.24 rows=18 width=4) (actual time=0.09..1.07 rows=109 loops=1)
Total runtime: 0.21 msec

NOTICE: QUERY PLAN:

Index Scan using index2 on ba y (cost=0.00..3.24 rows=18 width=4) (actual time=0.09..1.07 rows=109 loops=1)
Total runtime: 0.21 msec

c) Execution Time = 0.000210

d) Result tuples = 109

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix Y
where Y.father_label = '12'
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index2 on bd y (cost=0.00..3.27 rows=20 width=13) (actual time=0.11..2.28 rows=109 loops=1)
Total runtime: 2.52 msec

NOTICE: QUERY PLAN:

Index Scan using index2 on bd y (cost=0.00..3.27 rows=20 width=13) (actual time=0.11..2.28 rows=109 loops=1)
Total runtime: 2.52 msec

c) Execution Time = 0.00252

d) Result tuples = 109

3. Μικρό Postnum

A. Agrawal

a) SQL query

```
select Y.post
from PInterval Y
where (Y.father_post = 126252)
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index2 on ba y (cost=0.00..3.24 rows=18 width=4) (actual time=0.07..0.07 rows=1 loops=1)
Total runtime: 0.20 msec

NOTICE: QUERY PLAN:

Index Scan using index2 on ba y (cost=0.00..3.24 rows=18 width=4) (actual time=0.07..0.07 rows=1 loops=1)
Total runtime: 0.20 msec

c) Execution Time = 0.00020

d) Result tuples = 1

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix Y
where Y.label = '12f226af1d54'
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index2 on bd y (cost=0.00..3.27 rows=20 width=13) (actual time=0.08..0.08 rows=1 loops=1)
Total runtime: 0.20 msec

NOTICE: QUERY PLAN:

Index Scan using index2 on bd y (cost=0.00..3.27 rows=20 width=13) (actual time=0.08..0.08 rows=1 loops=1)
Total runtime: 0.20 msec

c) Execution Time = 0.00020

d) Result tuples = 1

3.3 Πρόγονοι

1. Αριστερός "Φύλλο" κόμβος με το μικρότερο postnum (= 1)

A. Agrawal

a) SQL query

```
(select Y.post
from PInterval Y
where (Y.index <= 2 and Y.post > 2))
UNION ALL
(select X.post
from PInterval X, DPInterval Y, (select * as T
                                 from PInterval U
                                 where U.index <= 2 and U.post >= 2 and U.post = Y.post) Z
where X.index <= Y.ancestor and X.post >= Y.ancestor and Z.T = Y.post)
```

b) Execution Plan

NOTICE: QUERY PLAN:

-> Append (cost=0.00..305234287.51 rows=2698283367 width=12) (actual time=17701.14..17701.14 rows=23 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..5411.18 rows=25 width=4) (actual time=1188.20..1188.20 rows=5 loops=1)
-> Seq Scan on ba y (cost=0.00..5411.18 rows=25 width=4) (actual time=1188.19..1188.19 rows=253215 loops=1)
-> Subquery Scan *SELECT* 2 (cost=0.00..305228876.33 rows=2698283341 width=12) (actual time=16512.93..16512.93 rows=18 loops=1)
-> Nested Loop (cost=0.00..305228876.33 rows=2698283341 width=12) (actual time=16512.91..16512.91 rows=18 loops=1)

-> Seq Scan on oa y (cost=0.00..582124.64 rows=95906 width=4) (actual time=16512.91..16512.91 rows= loops=1)
SubPlan
-> Index Scan using index1 on ba u (cost=0.00..3.02 rows=1 width=12) (actual time=0.07..0.07 rows=1 loops=191812)
-> Index Scan using index1 on ba x (cost=0.00..1910.45 rows=84404 width=8) (actual time=0.07..0.09 rows=1 loops=1)
Total runtime: 2632.303 msec

NOTICE: QUERY PLAN:

-> Append (cost=0.00..305234287.51 rows=2698283367 width=12) (actual time=17701.14..17701.14 rows=23 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..5411.18 rows=25 width=4) (actual time=1188.20..1188.20 rows=5 loops=1)
-> Seq Scan on ba y (cost=0.00..5411.18 rows=25 width=4) (actual time=1188.19..1188.19 rows=253215 loops=1)
-> Subquery Scan *SELECT* 2 (cost=0.00..305228876.33 rows=2698283341 width=12) (actual time=16512.93..16512.93 rows=18 loops=1)
-> Nested Loop (cost=0.00..305228876.33 rows=2698283341 width=12) (actual time=16512.91..16512.91 rows=18 loops=1)
-> Seq Scan on oa y (cost=0.00..582124.64 rows=95906 width=4) (actual time=16512.91..16512.91 rows= loops=1)
SubPlan
-> Index Scan using index1 on ba u (cost=0.00..3.02 rows=1 width=12) (actual time=0.07..0.07 rows=1 loops=191812)
-> Index Scan using index1 on ba x (cost=0.00..1910.45 rows=84404 width=8) (actual time=0.07..0.09 rows=1 loops=1)
Total runtime: 2632.303 msec

c) Execution Time = 2.632303

d) Result tuples = 23

B. Unicode Dewey

a) SQL query

select *ud_ancestors*('121f23')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=4.74..4.74 rows=1 loops=1)
Total runtime: 5.14 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=4.74..4.74 rows=1 loops=1)

Total runtime: 5.14 msec

c) Execution Time = 0.00514

d) Result tuples = 23

2. Δεξιός "Φύλλο" κόμβος με το μεγαλύτερο postnum (id = 253211)

A. Agrawal

a) SQL query

```
(select Y.post
from PInterval Y
where (Y.index <= 253211 and Y.post > 253211))
UNION ALL
(select X.post
from PInterval X, DPInterval Y, (select * as T
                                 from PInterval U
                                 where U.index <= 253211 and U.post >= 253211 and U.post = Y.post) Z
where X.index <= Y.ancestor and X.post >= Y.ancestor and Z.T = Y.post)
```

b) Execution Plan

NOTICE: QUERY PLAN:

-> Append (cost=0.00..305232187.90 rows=2698346541 width=12) (actual time=12334.58..12334.58 rows=14 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..3311.58 rows=63200 width=4) (actual time=1720.17..1720.17 rows=4 loops=1)
-> Index Scan using index1 on ba y (cost=0.00..3311.58 rows=63200 width=4) (actual time=1720.16..1720.16 rows=1 loops=1)
-> Subquery Scan *SELECT* 2 (cost=0.00..305228876.33 rows=2698283341 width=12) (actual time=10614.39..10614.39 rows=10 loops=1)
-> Nested Loop (cost=0.00..305228876.33 rows=2698283341 width=12) (actual time=10614.38..10614.38 rows=10 loops=1)
-> Seq Scan on oa y (cost=0.00..582124.64 rows=95906 width=4) (actual time=10614.37..10614.37 rows=10 loops=1)
SubPlan
-> Index Scan using index1 on ba u (cost=0.00..3.02 rows=1 width=12) (actual time=0.04..0.04 rows=1 loops=191812)
-> Index Scan using index1 on ba x (cost=0.00..1910.45 rows=84404 width=8)
Total runtime: 3095.479 msec

NOTICE: QUERY PLAN:

-> Append (cost=0.00..305232187.90 rows=2698346541 width=12) (actual time=12334.58..12334.58 rows=14 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..3311.58 rows=63200 width=4) (actual time=1720.17..1720.17 rows=4 loops=1)

-> Index Scan using index1 on ba y (cost=0.00..3311.58 rows=63200 width=4) (actual time=1720.16..1720.16 rows=1 loops=1)
-> Subquery Scan *SELECT* 2 (cost=0.00..305228876.33 rows=2698283341 width=12) (actual time=10614.39..10614.39 rows=10 loops=1)
-> Nested Loop (cost=0.00..305228876.33 rows=2698283341 width=12) (actual time=10614.38..10614.38 rows=10 loops=1)
-> Seq Scan on oa y (cost=0.00..582124.64 rows=95906 width=4) (actual time=10614.37..10614.37 rows=10 loops=1)
SubPlan
-> Index Scan using index1 on ba u (cost=0.00..3.02 rows=1 width=12) (actual time=0.04..0.04 rows=1 loops=191812)
-> Index Scan using index1 on ba x (cost=0.00..1910.45 rows=84404 width=8)
Total runtime: 3095.479 msec

c) Execution Time = 3.095479

d) Result tuples = 14

B. Unicode Dewey

a) SQL query

select *ud_ancestors*('1G261')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=4.83..4.83 rows=1 loops=1)
Total runtime: 4.72 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=4.83..4.83 rows=1 loops=1)
Total runtime: 4.72 msec

c) Execution Time = 0.00472

d) Result tuples = 14

3. Μεσαίος “Φύλλο” κόμβος με το (= 126623)

A. Agrawal

a) SQL query

```
(select Y.post
from PInterval Y
where (Y.index <= 126623 and Y.post > 126623))
UNION ALL
```

```
(select X.post
from PInterval X, DPInterval Y, (select * as T
                                 from PInterval U
                                 where U.index <= 126623 and U.post >= 126623 and U.post = Y.post) Z
where X.index <= Y.ancestor and X.post >= Y.ancestor and Z.T = Y.post)
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

-> Append (cost=0.00..305233707.08 rows=2698329634 width=12) (actual time=15934.36..15934.36
rows=167 loops=1)
        -> Subquery Scan *SELECT* 1 (cost=0.00..4830.75 rows=46293 width=4) (actual
time=2326.14..2326.14 rows=9 loops=1)
            -> Index Scan using index1 on ba y (cost=0.00..4830.75 rows=46293 width=4) (actual
time=2326.13..2326.13 rows=1 loops=1)
        -> Subquery Scan *SELECT* 2 (cost=0.00..305228876.33 rows=2698283341 width=12) (actual
time=13608.20..13608.20 rows=158 loops=1)
            -> Nested Loop (cost=0.00..305228876.33 rows=2698283341 width=12) (actual
time=13608.19..13608.19 rows=158 loops=1)
                -> Seq Scan on oa y (cost=0.00..582124.64 rows=95906 width=4) (actual
time=13608.18..13608.18 rows=158 loops=1)
                    SubPlan
                     -> Index Scan using index1 on ba u (cost=0.00..3.02 rows=1 width=12) (actual
time=0.06..0.06 rows=1 loops=191812)
                -> Index Scan using index1 on ba x (cost=0.00..1910.45 rows=84404 width=8) (actual
time=0.06..0.07 rows=1 loops=1)
Total runtime: 3790.538 msec

NOTICE:  QUERY PLAN:

-> Append (cost=0.00..305233707.08 rows=2698329634 width=12) (actual time=15934.36..15934.36
rows=167 loops=1)
        -> Subquery Scan *SELECT* 1 (cost=0.00..4830.75 rows=46293 width=4) (actual
time=2326.14..2326.14 rows=9 loops=1)
            -> Index Scan using index1 on ba y (cost=0.00..4830.75 rows=46293 width=4) (actual
time=2326.13..2326.13 rows=1 loops=1)
        -> Subquery Scan *SELECT* 2 (cost=0.00..305228876.33 rows=2698283341 width=12) (actual
time=13608.20..13608.20 rows=158 loops=1)
            -> Nested Loop (cost=0.00..305228876.33 rows=2698283341 width=12) (actual
time=13608.19..13608.19 rows=158 loops=1)
                -> Seq Scan on oa y (cost=0.00..582124.64 rows=95906 width=4) (actual
time=13608.18..13608.18 rows=158 loops=1)
                    SubPlan
                     -> Index Scan using index1 on ba u (cost=0.00..3.02 rows=1 width=12) (actual
time=0.06..0.06 rows=1 loops=191812)
                -> Index Scan using index1 on ba x (cost=0.00..1910.45 rows=84404 width=8) (actual
time=0.06..0.07 rows=1 loops=1)
Total runtime: 3790.538 msec
```

c) Execution Time = 3.790538

d) Result tuples = 167

B. Unicode Dewey

a) SQL query

select *ud_ancestors*('12f226a91c')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=6.21..6.22 rows=1 loops=1)
Total runtime: 6.19 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=6.21..6.22 rows=1 loops=1)
Total runtime: 6.19 msec

c) Execution Time = 0.00619

d) Result tuples = 167

3.4 Άμεσοι Πρόγονοι (Πατέρες)

1. Μεγάλο Postnum

A. Agrawal

a) SQL query

select Y.father_post
from PInterval Y
where (Y.post = 253215)

b) Execution Plan

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.04 rows=1 width=8) (actual time=0.08..0.08 rows=0 loops=1)
-> Index Scan using index1 on ba y (cost=0.00..3.01 rows=1 width=4) (actual time=0.16..1.69 rows=2 loops=1)
-> Index Scan using index1 on ba x (cost=0.00..3.01 rows=1 width=4) (actual time=0.09..0.10 rows=1 loops=1)
Total runtime: 0.33 msec

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.04 rows=1 width=8) (actual time=0.08..0.08 rows=0 loops=1)
-> Index Scan using index1 on ba y (cost=0.00..3.01 rows=1 width=4) (actual time=0.16..1.68 rows=2 loops=1)
-> Index Scan using index1 on ba x (cost=0.00..3.01 rows=1 width=4) (actual time=0.09..0.10 rows=1 loops=1)
Total runtime: 0.33 msec

c) Execution Time = 0.00033

d) Result tuples = 0

B. Unicode Dewey

a) SQL query

```
select Y.father_label
from UPrefix Y
where Y.label = '12f226a91c'
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on bd y (cost=0.00..3.02 rows=1 width=12) (actual time=0.12..0.12 rows=0 loops=1)
Total runtime: 0.21 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on bd y (cost=0.00..3.02 rows=1 width=12) (actual time=0.12..0.12 rows=0 loops=1)
Total runtime: 0.21 msec

c) Execution Time = 0.00021

d) Result tuples = 0

2. Μεσαίο Postnum

A. Agrawal

a) SQL query

```
select Y.father_post
from PInterval Y
where Y.post = 176077
```

b) Execution Plan

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.04 rows=1 width=8) (actual time=0.08..0.08 rows=0 loops=1)
-> Index Scan using index1 on ba y (cost=0.00..3.01 rows=1 width=4) (actual time=0.16..2.37 rows=2 loops=1)
-> Index Scan using index1 on ba x (cost=0.00..3.01 rows=1 width=4) (actual time=0.09..0.11 rows=1 loops=1)
Total runtime: 0.30 msec

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.04 rows=1 width=8) (actual time=0.08..0.08 rows=0 loops=1)
-> Index Scan using index1 on ba y (cost=0.00..3.01 rows=1 width=4) (actual time=0.16..2.37 rows=2 loops=1)
-> Index Scan using index1 on ba x (cost=0.00..3.01 rows=1 width=4) (actual time=0.09..0.11 rows=1 loops=1)
Total runtime: 0.30 msec

c) Execution Time = 0.00030

d) Result tuples = 2

B. Unicode Dewey

a) SQL query

```
select Y.father_label
from UPrefix Y
where Y.label = '121f23'
```

b) Execution Plan

NOTICE: QUERY PLAN:

Index Scan using index1 on bd y (cost=0.00..3.02 rows=1 width=12) (actual time=0.07..0.07 rows=2 loops=1)
Total runtime: 0.20 msec

NOTICE: QUERY PLAN:

Index Scan using index1 on bd y (cost=0.00..3.02 rows=1 width=12) (actual time=0.07..0.07 rows=2 loops=1)
Total runtime: 0.20 msec

c) Execution Time = 0.00020

d) Result tuples = 2

3. Μικρό Postnum

A. Agrawal

a) SQL query

```
select Y.father_post
from PInterval Y
where Y.post = 126252
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Nested Loop  (cost=0.00..6.04 rows=1 width=8) (actual time=0.07..0.07 rows=0 loops=1)
  ->  Index Scan using index1 on ba y  (cost=0.00..3.01 rows=1 width=4) (actual time=0.07..0.23 rows=2 loops=1)
  ->  Index Scan using index1 on ba x  (cost=0.00..3.01 rows=1 width=4) (actual time=0.07..0.09 rows=1 loops=1)
Total runtime: 0.32 msec

NOTICE:  QUERY PLAN:

Nested Loop  (cost=0.00..6.04 rows=1 width=8) (actual time=0.07..0.07 rows=0 loops=1)
  ->  Index Scan using index1 on ba y  (cost=0.00..3.01 rows=1 width=4) (actual time=0.07..0.23 rows=2 loops=1)
  ->  Index Scan using index1 on ba x  (cost=0.00..3.01 rows=1 width=4) (actual time=0.07..0.09 rows=1 loops=1)
Total runtime: 0.32 msec
```

c) Execution Time = 0.00032

d) Result tuples = 2

B. Unicode Dewey

a) SQL query

```
select Y.father_label
from UPrefix Y
where Y.label = '1G261'
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Index Scan using index1 on bd y  (cost=0.00..3.02 rows=1 width=12) (actual time=0.08..0.08 rows=2 loops=1)
Total runtime: 0.20 msec

NOTICE:  QUERY PLAN:
```

Index Scan using index1 on bd y (cost=0.00..3.02 rows=1 width=12) (actual time=0.08..0.08 rows=2 loops=1)
Total runtime: 0.20 msec

c) Execution Time = 0.00020

d) Result tuples = 2

3.5 Έλεγχος Υπαλληλίας (Subsumption Check)

1. Ριζικός κόμβος – Αριστερός "Φύλλο" κόμβος με το μικρότερο postnum (= 1)

A. Agrawal

a) SQL query

select *class_subclass*(1,253214,1)

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.14..0.14 rows=1 loops=1)
Total runtime: 0.19 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.14..0.14 rows=1 loops=1)
Total runtime: 0.19 msec

c) Execution Time = 0.00019

d) Result tuples = 0

B. Unicode Dewey

a) SQL query

select *test_subclass*('1','121f23')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.33..1.33 rows=1 loops=1)
Total runtime: 1.36 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.33..1.33 rows=1 loops=1)

Total runtime: 1.36 msec

c) Execution Time = 0.00136

d) Result tuples = 0

2. Ριζικός κόμβος – Δεξιός "Φύλλο" κόμβος με το μεγαλύτερο postnum (= 253211)

A. Agrawal

a) SQL query

select *class_subclass*(1,253214,253211)

b) Execution Plan

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.14..0.14 rows=1 loops=1)
Total runtime: 0.19 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.14..0.14 rows=1 loops=1)
Total runtime: 0.19 msec

c) Execution Time = 0.00019

d) Result tuples = 0

B. Unicode Dewey

a) SQL query

select *test_subclass*('1','1G221')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.34..1.34 rows=1 loops=1)
Total runtime: 1.37 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.34..1.34 rows=1 loops=1)
Total runtime: 1.37 msec

c) Execution Time = 0.00137

d) Result tuples = 0

3. Ριζικός κόμβος – Μεσαίος "Φύλλο" κόμβος (= 126623)

A. Agrawal

a) SQL query

select *class_subclass*(1,253214,126623)

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.14..0.14 rows=1 loops=1)
Total runtime: 0.20 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.14..0.14 rows=1 loops=1)
Total runtime: 0.20 msec

c) Execution Time = 0.00020

d) Result tuples = 0

B. Unicode Dewey

a) SQL query

select *test_subclass*('1','12f226a91c')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.32..1.32 rows=1 loops=1)
Total runtime: 1.34 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.32..1.32 rows=1 loops=1)
Total runtime: 1.34 msec

c) Execution Time = 0.00134

d) Result tuples = 0

3.6 "Φύλλα"

1. "Φύλλα" (Ριζικός κόμβος)

A. Agrawal

a) SQL query

```
(select Y.post
from PInterval Y
where Y.post < 253215 and Y.post >= 1 and
Y.post = Y.index and NOT EXISTS (    select *
                                     from DPInterval X
                                     where X.ancestor = Y.post))
UNION ALL
(select Y.post
from PInterval Y, DPInterval W
where W.ancestor = 253215 and Y.post >= W.index and
Y.post <= W.post and NOT EXISTS (    select *
                                     from DPInterval S
                                     where S.ancestor = Y.post))
```

b) Execution Plan:

NOTICE: QUERY PLAN:

-> Append (cost=0.00..132830.58 rows=292313 width=12) (actual time=1.18..12638.79 rows=109132 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..23457.16 rows=1 width=4) (actual time=1.17..12434.97 rows=109132 loops=1)
-> Index Scan using index1 on ba y (cost=0.00..23457.16 rows=1 width=4) (actual time=1.16..11741.56 rows=109132 loops=1)
SubPlan
-> Index Scan using index4 on oa x (cost=0.00..3.29 rows=21 width=12) (actual time=0.05..0.05 rows=0 loops=126604)
-> Subquery Scan *SELECT* 2 (cost=0.00..109373.41 rows=292312 width=12) (actual time=0.06..0.06 rows=0 loops=1)
-> Nested Loop (cost=0.00..109373.41 rows=292312 width=12) (actual time=0.06..0.06 rows=0 loops=1)
-> Index Scan using index4 on oa w (cost=0.00..3.28 rows=21 width=8) (actual time=0.05..0.05 rows=0 loops=1)
-> Index Scan using index1 on ba y (cost=0.00..5052.37 rows=14067 width=4)
SubPlan
-> Index Scan using index4 on oa s (cost=0.00..3.29 rows=21 width=12)
Total runtime: 14506.418 msec

NOTICE: QUERY PLAN:

-> Append (cost=0.00..132830.58 rows=292313 width=12) (actual time=1.18..12638.79 rows=109132 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..23457.16 rows=1 width=4) (actual time=1.17..12434.97 rows=109132 loops=1)

```
        ->  Index Scan using index1 on ba y  (cost=0.00..23457.16 rows=1 width=4) (actual time=1.16..11741.56 rows=109132 loops=1)
              SubPlan
               ->  Index Scan using index4 on oa x  (cost=0.00..3.29 rows=21 width=12) (actual time=0.05..0.05 rows=0 loops=126604)
      ->  Subquery Scan *SELECT* 2  (cost=0.00..109373.41 rows=292312 width=12) (actual time=0.06..0.06 rows=0 loops=1)
            ->  Nested Loop  (cost=0.00..109373.41 rows=292312 width=12) (actual time=0.06..0.06 rows=0 loops=1)
              ->  Index Scan using index4 on oa w  (cost=0.00..3.28 rows=21 width=8) (actual time=0.05..0.05 rows=0 loops=1)
              ->  Index Scan using index1 on ba y  (cost=0.00..5052.37 rows=14067 width=4)
                  SubPlan
                   ->  Index Scan using index4 on oa s  (cost=0.00..3.29 rows=21 width=12)
Total runtime: 14506.418 msec
```

c) Execution Time = 14.506418

d) Result tuples = 109132

B. Unicode Dewey

a) SQL query

```
(select Y.label
from UPrefix Y
where Y.label > '1' and Y.label < '1'||'xFF'
and NOT EXISTS (      select *
                      from UPrefix X
                      where X.label > Y.label and X.label < Y.label||'xFF')
and NOT EXISTS (      select *
                      from DUPrefix Z
                      where Z.ancestor = Y.label))
UNION ALL
(select W.label
from DUPrefixl W, (   select P.label as T
                      from UPrefix P where P.label >= '1' and P.label < '1'||'xFF') V
                      where W.father_label = V.T and
                      NOT EXISTS ( select *
                                   from UPrefix S
                                   where S.label > W.label and S.label < W.label||'xFF') and
                                   NOT EXISTS ( select *
                                                from DUPrefix B
                                                where B.ancestor = W.label)
)
```

b) Execution Plan:

NOTICE: QUERY PLAN:

-> Append (cost=0.00..357202.04 rows=130275 width=34) (actual time=1.56..102478.89 rows=354183 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..171974.43 rows=63290 width=13) (actual time=1.56..40105.15 rows=185382 loops=1)
-> Seq Scan on bd y (cost=0.00..171974.43 rows=63290 width=13) (actual time=1.55..38651.99 rows=185382 loops=1)
SubPlan
-> Index Scan using index1 on bd x (cost=0.00..40.42 rows=1266 width=25) (actual time=0.07..0.07 rows=0 loops=253211)
-> Index Scan using index4 on od z (cost=0.00..107.58 rows=172 width=21) (actual time=0.06..0.06 rows=0 loops=185382)
-> Subquery Scan *SELECT* 2 (cost=176493.58..185227.61 rows=66985 width=34) (actual time=51698.07..61751.78 rows=168801 loops=1)
-> Merge Join (cost=176493.58..185227.61 rows=66985 width=34) (actual time=51698.05..60705.46 rows=168801 loops=1)
-> Index Scan using index1 on bd p (cost=0.00..7105.57 rows=253160 width=13) (actual time=0.27..4960.13 rows=253193 loops=1)
-> Sort (cost=176493.58..176493.58 rows=63304 width=21) (actual time=51674.23..52560.62 rows=204053 loops=1)
-> Seq Scan on od w (cost=0.00..170652.04 rows=63304 width=21) (actual time=1.00..38110.04 rows=197097 loops=1)
SubPlan
-> Index Scan using index1 on bd s (cost=0.00..40.42 rows=1266 width=25) (actual time=0.07..0.07 rows=0 loops=253215)
-> Index Scan using index4 on od b (cost=0.00..107.58 rows=172 width=21) (actual time=0.06..0.06 rows=0 loops=197097)
Total runtime: 95582.223 msec

NOTICE: QUERY PLAN:

-> Append (cost=0.00..357202.04 rows=130275 width=34) (actual time=1.56..102478.89 rows=354183 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..171974.43 rows=63290 width=13) (actual time=1.56..40105.15 rows=185382 loops=1)
-> Seq Scan on bd y (cost=0.00..171974.43 rows=63290 width=13) (actual time=1.55..38651.99 rows=185382 loops=1)
SubPlan
-> Index Scan using index1 on bd x (cost=0.00..40.42 rows=1266 width=25) (actual time=0.07..0.07 rows=0 loops=253211)
-> Index Scan using index4 on od z (cost=0.00..107.58 rows=172 width=21) (actual time=0.06..0.06 rows=0 loops=185382)
-> Subquery Scan *SELECT* 2 (cost=176493.58..185227.61 rows=66985 width=34) (actual time=51698.07..61751.78 rows=168801 loops=1)
-> Merge Join (cost=176493.58..185227.61 rows=66985 width=34) (actual time=51698.05..60705.46 rows=168801 loops=1)
-> Index Scan using index1 on bd p (cost=0.00..7105.57 rows=253160 width=13) (actual time=0.27..4960.13 rows=253193 loops=1)
-> Sort (cost=176493.58..176493.58 rows=63304 width=21) (actual time=51674.23..52560.62 rows=204053 loops=1)
-> Seq Scan on od w (cost=0.00..170652.04 rows=63304 width=21) (actual time=1.00..38110.04 rows=197097 loops=1)

SubPlan
-> Index Scan using index1 on bd s (cost=0.00..40.42 rows=1266 width=25) (actual time=0.07..0.07 rows=0 loops=253215)
-> Index Scan using index4 on od b (cost=0.00..107.58 rows=172 width=21) (actual time=0.06..0.06 rows=0 loops=197097)
Total runtime: 95582.223 msec

c) Execution Time = 95.582223

d) Result tuples = 109132

2. "Φύλλα" (ένας κόμβος με μεσαίο πλήθος απογόνων (= 25314))

A. Agrawal

a) SQL query

```
(select Y.post
from PInterval Y
where Y.post < 176077 and Y.post >= 150765 and Y.post = Y.index and
NOT EXISTS ( select *
             from DPInterval X
             where X.ancestor = Y.post))
UNION ALL
(select Y.post
from PInterval Y, DPInterval W
where W.ancestor = 176077 and Y.post >= W.index and Y.post <= W.post and
NOT EXISTS ( select *
             from DPInterval S
             where S.ancestor = Y.post))
```

b) Execution Plan

NOTICE: QUERY PLAN:

-> Append (cost=0.00..111598.81 rows=292313 width=12) (actual time=0.22..1171.61 rows=11201 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..2225.39 rows=1 width=4) (actual time=0.22..1151.76 rows=11201 loops=1)
-> Index Scan using index1 on ba y (cost=0.00..2225.39 rows=1 width=4) (actual time=0.21..1083.45 rows=11201 loops=1)
SubPlan
-> Index Scan using index4 on oa x (cost=0.00..3.29 rows=21 width=12) (actual time=0.05..0.05 rows=0 loops=12657)
-> Subquery Scan *SELECT* 2 (cost=0.00..109373.41 rows=292312 width=12) (actual time=0.06..0.06 rows=0 loops=1)
-> Nested Loop (cost=0.00..109373.41 rows=292312 width=12) (actual time=0.06..0.06 rows=0 loops=1)
-> Index Scan using index4 on oa w (cost=0.00..3.28 rows=21 width=8) (actual time=0.05..0.05 rows=0 loops=1)

```
                -> Index Scan using index1 on ba y  (cost=0.00..5052.37 rows=14067 width=4)
                   SubPlan
                     -> Index Scan using index4 on oa s  (cost=0.00..3.29 rows=21 width=12)
Total runtime: 1578.370 msec

NOTICE:  QUERY PLAN:

-> Append  (cost=0.00..111598.81 rows=292313 width=12) (actual time=0.22..1171.61 rows=11201 loops=1)
        -> Subquery Scan *SELECT* 1  (cost=0.00..2225.39 rows=1 width=4) (actual time=0.22..1151.76 rows=11201 loops=1)
            -> Index Scan using index1 on ba y  (cost=0.00..2225.39 rows=1 width=4) (actual time=0.21..1083.45 rows=11201 loops=1)
                SubPlan
                 -> Index Scan using index4 on oa x  (cost=0.00..3.29 rows=21 width=12) (actual time=0.05..0.05 rows=0 loops=12657)
        -> Subquery Scan *SELECT* 2  (cost=0.00..109373.41 rows=292312 width=12) (actual time=0.06..0.06 rows=0 loops=1)
            -> Nested Loop  (cost=0.00..109373.41 rows=292312 width=12) (actual time=0.06..0.06 rows=0 loops=1)
                -> Index Scan using index4 on oa w  (cost=0.00..3.28 rows=21 width=8) (actual time=0.05..0.05 rows=0 loops=1)
                -> Index Scan using index1 on ba y  (cost=0.00..5052.37 rows=14067 width=4)
                   SubPlan
                     -> Index Scan using index4 on oa s  (cost=0.00..3.29 rows=21 width=12)
Total runtime: 1578.370 msec
```

c) Execution Time = 1.578370

d) Result tuples = 11201

B. Unicode Dewey

a) SQL query

```
(select Y.label
from UPrefix Y
where Y.label > '122' and Y.label < '122'||'xFF' and
NOT EXISTS ( select *
             from UPrefix X
             where X.label > Y.label and X.label < Y.label||'xFF') and
NOT EXISTS ( select *
             from DUPrefix Z
             where Z.ancestor = Y.label)
)
UNION ALL
(select W.label
from DUPrefix W, (      select P.label as T
                        from UPrefix P
                        where P.label >= '122' and P.label < '122'||'xFF') V
```

```
where W.ancestor = V.T and
NOT EXISTS ( select *
                    from UPrefix S
                    where S.label > W.label and S.label < W.label||'xFF') and
NOT EXISTS ( select *
                from DUPrefix B
                where B.ancestor = W.label))
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

-> Append  (cost=0.00..199931.66 rows=16683 width=34) (actual time=0.45..52572.77 rows=11201 loops=1)
        -> Subquery Scan *SELECT* 1  (cost=0.00..22179.85 rows=8105 width=13) (actual time=0.44..43.17 rows=11201 loops=1)
            -> Index Scan using index1 on bd y  (cost=0.00..22179.85 rows=8105 width=13) (actual time=0.43..41.64 rows=11201 loops=1)
                SubPlan
                 -> Index Scan using index1 on bd x  (cost=0.00..40.42 rows=1266 width=25) (actual time=0.06..0.06 rows=1 loops=267)
                 -> Index Scan using index4 on od z  (cost=0.00..107.58 rows=172 width=21) (actual time=0.05..0.05 rows=1 loops=224)
        -> Subquery Scan *SELECT* 2  (cost=176493.58..177751.81 rows=8578 width=34) (actual time=52529.16..52529.16 rows=224 loops=1)
            -> Merge Join  (cost=176493.58..177751.81 rows=8578 width=34) (actual time=52529.16..52529.16 rows=224 loops=1)
                -> Sort  (cost=176493.58..176493.58 rows=63304 width=21) (actual time=51827.64..52147.85 rows=96034 loops=1)
                    -> Seq Scan on od w  (cost=0.00..170652.04 rows=63304 width=21) (actual time=1.14..38158.14 rows=197097 loops=1)
                        SubPlan
                         -> Index Scan using index1 on bd s  (cost=0.00..40.42 rows=1266 width=25) (actual time=0.07..0.07 rows=1 loops=253215)
                         -> Index Scan using index4 on od b  (cost=0.00..107.58 rows=172 width=21) (actual time=0.06..0.06 rows=1 loops=197097)
                -> Index Scan using index1 on bd p  (cost=0.00..911.70 rows=32419 width=13) (actual time=0.41..5.62 rows=268 loops=1)
Total runtime: 48192.655 msec

NOTICE:  QUERY PLAN:

-> Append  (cost=0.00..199931.66 rows=16683 width=34) (actual time=0.45..52572.77 rows=11201 loops=1)
        -> Subquery Scan *SELECT* 1  (cost=0.00..22179.85 rows=8105 width=13) (actual time=0.44..43.17 rows=11201 loops=1)
            -> Index Scan using index1 on bd y  (cost=0.00..22179.85 rows=8105 width=13) (actual time=0.43..41.64 rows=11201 loops=1)
                SubPlan
                 -> Index Scan using index1 on bd x  (cost=0.00..40.42 rows=1266 width=25) (actual time=0.06..0.06 rows=1 loops=267)
```

-> Index Scan using index4 on od z (cost=0.00..107.58 rows=172 width=21) (actual time=0.05..0.05 rows=1 loops=224)
-> Subquery Scan *SELECT* 2 (cost=176493.58..177751.81 rows=8578 width=34) (actual time=52529.16..52529.16 rows=224 loops=1)
-> Merge Join (cost=176493.58..177751.81 rows=8578 width=34) (actual time=52529.16..52529.16 rows=224 loops=1)
-> Sort (cost=176493.58..176493.58 rows=63304 width=21) (actual time=51827.64..52147.85 rows=96034 loops=1)
-> Seq Scan on od w (cost=0.00..170652.04 rows=63304 width=21) (actual time=1.14..38158.14 rows=197097 loops=1)
SubPlan
-> Index Scan using index1 on bd s (cost=0.00..40.42 rows=1266 width=25) (actual time=0.07..0.07 rows=1 loops=253215)
-> Index Scan using index4 on od b (cost=0.00..107.58 rows=172 width=21) (actual time=0.06..0.06 rows=1 loops=197097)
-> Index Scan using index1 on bd p (cost=0.00..911.70 rows=32419 width=13) (actual time=0.41..5.62 rows=268 loops=1)
Total runtime: 48192.655 msec

c) Execution Time = 48.192655

d) Result tuples = 11201

3. "Φύλλα" (ένας κόμβος με τον ελάχιστο αριθμό απογόνων (= 1))

A. Agrawal

a) SQL query

```
(select Y.post
from PInterval Y
where Y.post < 126252 and Y.post >= 126251 and Y.post = Y.index and
NOT EXISTS ( select *
                from DPInterval X
                where X.ancestor = Y.post)
)
UNION ALL
(select Y.post
from PInterval Y, DPInterval W
where W.ancestor = 126252 and Y.post >= W.index and Y.post <= W.post and
NOT EXISTS ( select *
                from DPInterval S
                where S.ancestor = Y.post)
)
```

b) Execution Plan

NOTICE: QUERY PLAN:

-> Append (cost=0.00..109376.59 rows=292313 width=12) (actual time=0.18..0.26 rows=1 loops=1)

```
        -> Subquery Scan *SELECT* 1  (cost=0.00..3.17 rows=1 width=4) (actual time=0.18..0.20 rows=1
loops=1)
            -> Index Scan using index1 on ba y  (cost=0.00..3.17 rows=1 width=4) (actual time=0.17..0.18
rows=1 loops=1)
                SubPlan
                 -> Index Scan using index4 on oa x  (cost=0.00..3.29 rows=21 width=12) (actual
time=0.05..0.05 rows=0 loops=1)
        -> Subquery Scan *SELECT* 2  (cost=0.00..109373.41 rows=292312 width=12) (actual
time=0.06..0.06 rows=0 loops=1)
            -> Nested Loop  (cost=0.00..109373.41 rows=292312 width=12) (actual time=0.05..0.05
rows=0 loops=1)
                -> Index Scan using index4 on oa w  (cost=0.00..3.28 rows=21 width=8) (actual
time=0.05..0.05 rows=0 loops=1)
                -> Index Scan using index1 on ba y  (cost=0.00..5052.37 rows=14067 width=4)
                    SubPlan
                     -> Index Scan using index4 on oa s  (cost=0.00..3.29 rows=21 width=12)
Total runtime: 0.87 msec

NOTICE:  QUERY PLAN:

-> Append  (cost=0.00..109376.59 rows=292313 width=12) (actual time=0.18..0.26 rows=1 loops=1)
        -> Subquery Scan *SELECT* 1  (cost=0.00..3.17 rows=1 width=4) (actual time=0.18..0.20 rows=1
loops=1)
            -> Index Scan using index1 on ba y  (cost=0.00..3.17 rows=1 width=4) (actual time=0.17..0.18
rows=1 loops=1)
                SubPlan
                 -> Index Scan using index4 on oa x  (cost=0.00..3.29 rows=21 width=12) (actual
time=0.05..0.05 rows=0 loops=1)
        -> Subquery Scan *SELECT* 2  (cost=0.00..109373.41 rows=292312 width=12) (actual
time=0.06..0.06 rows=0 loops=1)
            -> Nested Loop  (cost=0.00..109373.41 rows=292312 width=12) (actual time=0.05..0.05
rows=0 loops=1)
                -> Index Scan using index4 on oa w  (cost=0.00..3.28 rows=21 width=8) (actual
time=0.05..0.05 rows=0 loops=1)
                -> Index Scan using index1 on ba y  (cost=0.00..5052.37 rows=14067 width=4)
                    SubPlan
                     -> Index Scan using index4 on oa s  (cost=0.00..3.29 rows=21 width=12)
Total runtime: 0.87 msec
```

c) Execution Time = 0.00087

d) Result tuples = 1

B. Unicode Dewey

a) SQL query

```
(select Y.label
from UPrefix Y
where Y.label > '12f226af1d54' and Y.label < '12f226af1d54' ||'xFF' and
```

```
NOT EXISTS ( select *
                from UPrefix X
                where X.label > Y.label and X.label < Y.label||'xFF') and
NOT EXISTS ( select *
                from DUPrefix Z
                where Z.ancestor = Y.label)
)
UNION ALL
(select W.label
from DUPrefix W, (      select P.label as T
                        from UPrefix P
                        where P.label >= '12f226af1d54' and P.label < '12f226af1d54'||'xFF') V
where W.ancestor = V.T and
NOT EXISTS ( select *
                        from UPrefix S
                        where S.label > W.label and S.label < W.label||'xFF') and
NOT EXISTS ( select *
                from DUPrefix B
                where B.ancestor = W.label))
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

-> Append (cost=0.00..204066.13 rows=19647 width=34) (actual time=51621.30..51621.30 rows=1
loops=1)
        -> Subquery Scan *SELECT* 1 (cost=0.00..26119.72 rows=9545 width=13) (actual
time=0.44..0.44 rows=1 loops=1)
            -> Index Scan using index1 on bd y (cost=0.00..26119.72 rows=9545 width=13) (actual
time=0.43..0.43 rows=1 loops=1)
                SubPlan
                  -> Index Scan using index1 on bd x (cost=0.00..40.42 rows=1266 width=25)
                  -> Index Scan using index4 on od z (cost=0.00..107.58 rows=172 width=21)
        -> Subquery Scan *SELECT* 2 (cost=176493.58..177946.42 rows=10102 width=34) (actual
time=51620.85..51620.85 rows=1 loops=1)
            -> Merge Join (cost=176493.58..177946.42 rows=10102 width=34) (actual
time=51620.84..51620.84 rows=1 loops=1)
                -> Sort (cost=176493.58..176493.58 rows=63304 width=21) (actual
time=51620.46..51620.46 rows=1 loops=1)
                    -> Seq Scan on od w (cost=0.00..170652.04 rows=63304 width=21) (actual
time=1.44..38109.11 rows=197097 loops=1)
                        SubPlan
                          -> Index Scan using index1 on bd s (cost=0.00..40.42 rows=1266 width=25)
(actual time=0.07..0.07 rows=1 loops=253215)
                          -> Index Scan using index4 on od b (cost=0.00..107.58 rows=172 width=21)
(actual time=0.06..0.06 rows=0 loops=197097)
                -> Index Scan using index1 on bd p (cost=0.00..1072.86 rows=38179 width=13) (actual
time=0.37..0.37 rows=1 loops=1)
Total runtime: 47297.535 msec

NOTICE:  QUERY PLAN:
```

-> Append (cost=0.00..204066.13 rows=19647 width=34) (actual time=51621.30..51621.30 rows=1 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..26119.72 rows=9545 width=13) (actual time=0.44..0.44 rows=1 loops=1)
-> Index Scan using index1 on bd y (cost=0.00..26119.72 rows=9545 width=13) (actual time=0.43..0.43 rows=1 loops=1)
SubPlan
-> Index Scan using index1 on bd x (cost=0.00..40.42 rows=1266 width=25)
-> Index Scan using index4 on od z (cost=0.00..107.58 rows=172 width=21)
-> Subquery Scan *SELECT* 2 (cost=176493.58..177946.42 rows=10102 width=34) (actual time=51620.85..51620.85 rows=1 loops=1)
-> Merge Join (cost=176493.58..177946.42 rows=10102 width=34) (actual time=51620.84..51620.84 rows=1 loops=1)
-> Sort (cost=176493.58..176493.58 rows=63304 width=21) (actual time=51620.46..51620.46 rows=1 loops=1)
-> Seq Scan on od w (cost=0.00..170652.04 rows=63304 width=21) (actual time=1.44..38109.11 rows=197097 loops=1)
SubPlan
-> Index Scan using index1 on bd s (cost=0.00..40.42 rows=1266 width=25) (actual time=0.07..0.07 rows=1loops=253215)
-> Index Scan using index4 on od b (cost=0.00..107.58 rows=172 width=21) (actual time=0.06..0.06 rows=0 loops=197097)
-> Index Scan using index1 on bd p (cost=0.00..1072.86 rows=38179 width=13) (actual time=0.37..0.37 rows=1 loops=1)
Total runtime: 47297.535 msec

c) Execution Time = 47.297535

d) Result tuples = 1

3.7 Ομοεπίπεδοι (Siblings)

A. Agrawal

1. Ο κόμβος με το μέγιστο εύρος (=314)

a) SQL query

```
select Y.post
from PInterval X, PInterval Y
where X.post = 206712 and X.father_post = Y. father_post
```

b) Execution Plan

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..12.35 rows=24 width=12) (actual time=3.39..19.31 rows=317 loops=1)
-> Index Scan using index1 on ba x (cost=0.00..3.24 rows=1 width=4) (actual time=0.09..0.21 rows=1 loops=1)

-> Index Scan using index2 on ba y (cost=0.00..4.12 rows=17 width=8) (actual time=0.08..7.56 rows=317 loops=1)
Total runtime: 2.591 msec

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..12.35 rows=24 width=12) (actual time=3.39..19.31 rows=317 loops=1)
-> Index Scan using index1 on ba x (cost=0.00..3.24 rows=1 width=4) (actual time=0.09..0.21 rows=1 loops=1)
-> Index Scan using index2 on ba y (cost=0.00..4.12 rows=17 width=8) (actual time=0.08..7.56 rows=317 loops=1)
Total runtime: 2.591 msec

c) Execution Time = 0.002591

d) Result tuples = 317

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix X, UPrefix Y
where Y.label != '1522c143416d' and X.father_label = Y.father_label and X.label = '1522c143416d'
```

b) Execution Plan

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..8.22 rows=34 width=37) (actual time=9.11..29.11 rows=317 loops=1)
-> Index Scan using index1 on bd x (cost=0.00..4.47 rows=1 width=12) (actual time=0.18..0.34 rows=1 loops=1)
-> Index Scan using index2 on bd y (cost=0.00..3.44 rows=21 width=25) (actual time=0.09..11.02 rows=317 loops=1)
Total runtime: 3.011 msec

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..8.22 rows=34 width=37) (actual time=9.11..29.11 rows=317 loops=1)
-> Index Scan using index1 on bd x (cost=0.00..4.47 rows=1 width=12) (actual time=0.18..0.34 rows=1 loops=1)
-> Index Scan using index2 on bd y (cost=0.00..3.44 rows=21 width=25) (actual time=0.09..11.02 rows=317 loops=1)
Total runtime: 3.011 msec

c) Execution Time = 0.003011

d) Result tuples = 317

2. Ένας κόμβος με τον μέσο όρο εύρους και με τον μέγιστο αριθμό απογόνων

A. Agrawal

a) SQL query

```
select Y.post
from PInterval X, PInterval Y
where X.post = 133209 and X.father_post = Y. father_post
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Nested Loop  (cost=0.00..6.51 rows=24 width=12) (actual time=0.29..0.78 rows=4 loops=1)
  ->  Index Scan using index1 on ba x  (cost=0.00..3.03 rows=1 width=4) (actual time=0.09..0.13 rows=1
loops=1)
  ->  Index Scan using index2 on ba y  (cost=0.00..3.27 rows=18 width=8) (actual time=0.10..0.15 rows=4
loops=1)
Total runtime: 2.442 msec

NOTICE:  QUERY PLAN:

Nested Loop  (cost=0.00..6.51 rows=24 width=12) (actual time=0.29..0.78 rows=4 loops=1)
  ->  Index Scan using index1 on ba x  (cost=0.00..3.03 rows=1 width=4) (actual time=0.09..0.13 rows=1
loops=1)
  ->  Index Scan using index2 on ba y  (cost=0.00..3.27 rows=18 width=8) (actual time=0.10..0.15 rows=4
loops=1)
Total runtime: 2.442 msec
```

c) Execution Time = 0.002442

d) Result tuples = 4

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix X, UPrefix Y
where Y.label != '12f2cb181db2' and X.father_label = Y. father_label and X.label = '12f2cb181db2'
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Nested Loop  (cost=0.29..5.55 rows=34 width=37) (actual time=0.34..2.32 rows=4 loops=1)
  ->  Index Scan using index1 on bd x  (cost=0.00..4.47 rows=1 width=12) (actual time=0.11..0.21 rows=1
loops=1)
  ->  Index Scan using index2 on bd y  (cost=0.00..3.44 rows=21 width=25) (actual time=0.09..1.32 rows=4
loops=1)
```

Total runtime: 2.946 msec

NOTICE: QUERY PLAN:

Nested Loop (cost=0.29..5.55 rows=34 width=37) (actual time=0.34..2.32 rows=4 loops=1)
-> Index Scan using index1 on bd x (cost=0.00..4.47 rows=1 width=12) (actual time=0.11..0.21 rows=1 loops=1)
-> Index Scan using index2 on bd y (cost=0.00..3.44 rows=21 width=25) (actual time=0.09..1.32 rows=4 loops=1)
Total runtime: 2.946 msec

c) Execution Time = 0.002946

d) Result tuples = 4

3. Ένας κόμβος με το ελάχσιτο εύρος (=1)

A. Agrawal

a) SQL query

```
select Y.post
from PInterval X, PInterval Y
where X.post = 126609 and X.father_post = Y.father_post
```

b) Execution Plan

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.46 rows=24 width=12) (actual time=0.18..0.43 rows=2 loops=1)
-> Index Scan using index1 on ba x (cost=0.00..3.01 rows=1 width=4) (actual time=0.09..0.10 rows=1 loops=1)
-> Index Scan using index2 on ba y (cost=0.00..3.23 rows=17 width=8) (actual time=0.07..0.13 rows=2 loops=1)
Total runtime: 0.33 msec

NOTICE: QUERY PLAN:

Nested Loop (cost=0.00..6.46 rows=24 width=12) (actual time=0.18..0.43 rows=2 loops=1)
-> Index Scan using index1 on ba x (cost=0.00..3.01 rows=1 width=4) (actual time=0.09..0.10 rows=1 loops=1)
-> Index Scan using index2 on ba y (cost=0.00..3.23 rows=17 width=8) (actual time=0.07..0.13 rows=2 loops=1)
Total runtime: 0.33 msec

c) Execution Time = 0.000330

d) Result tuples = 2

B. Unicode Dewey

a) SQL query

```
select Y.label
from UPrefix X, UPrefix Y
where Y.label !='12f226aa1' and X.father_label = Y.father_label and X.label = '12f226aa1'
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Nested Loop  (cost=0.00..7.33 rows=34 width=37) (actual time=0.18..0.24 rows=2 loops=1)
  ->  Index Scan using index1 on bd x  (cost=0.00..3.02 rows=1 width=12) (actual time=0.11..0.13 rows=1
loops=1)
  ->  Index Scan using index2 on bd y  (cost=0.00..3.44 rows=21 width=25) (actual time=0.09..0.18 rows=2
loops=1)
Total runtime: 0.32 msec

NOTICE:  QUERY PLAN:

Nested Loop  (cost=0.00..7.33 rows=34 width=37) (actual time=0.18..0.24 rows=2 loops=1)
  ->  Index Scan using index1 on bd x  (cost=0.00..3.02 rows=1 width=12) (actual time=0.11..0.13 rows=1
loops=1)
  ->  Index Scan using index2 on bd y  (cost=0.00..3.44 rows=21 width=25) (actual time=0.09..0.18 rows=2
loops=1)
Total runtime: 0.32 msec
```

c) Execution Time = 0.000320

d) Result tuples = 2

3.8 Lower Upper Bound (NCA)

1. Αριστερός “Φύλλο” κόμβος με το μικρότερο postnum (= 1) – Δεξιός “Φύλλο” κόμβος με το μεγαλύτερο postnum (= 253211)

A. Agrawal

a) SQL query

```
(select Y.post
from PInterval Y
where Y.index <= 1 and Y.post > 253211 and
NOT EXISTS ( select *
                from PInterval W where W.index <= 1 and 253211 < W.post and
                Y.index <= W.post and W.post < Y.post)
)
UNION ALL
(select R.post
from
```

```
        (select W.post, W.index, W.father_post
        from (   select S.post AS T
                 from PInterval S
                 where S.index <= 1 and S.post > 1) Z, DPInterval V, PInterval W
        where V.post = Z.T and W.index <= V.ancestor and W.post >= V.ancestor) R,

        (select W.post, W.index, W.father_post
        from (   select S.post as T
                 from PInterval S
                 where S.index <= 253211and S.post > 253211) Z, DPInterval V, PInterval W
        where V.post = Z.T and W.index <=V.ancestor and W.post >= V.ancestor) S
where R. father_post = S. father_post)
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Append (cost=0.00..1604786058773.36 rows=128296069833794 width=48) (actual time=3965.59..3965.59
rows=0 loops=1)
 ->  Subquery Scan *SELECT* 1  (cost=0.00..19291.60 rows=15937 width=4) (actual
time=1595.16..1595.16 rows=0 loops=1)
     ->  Index Scan using index1 on ba y  (cost=0.00..19291.60 rows=15937 width=4) (actual
time=1595.15..1595.15 rows=0 loops=1)
         SubPlan
           ->  Index Scan using index1 on ba w  (cost=0.00..40.59 rows=319 width=12)
 ->  Subquery Scan *SELECT* 2  (cost=1079251857.37..1604786039481.76 rows=128296069817856
width=48) (actual time=2370.41..2370.41 rows=0 loops=1)
     ->  Merge Join  (cost=1079251857.37..1604786039481.76 rows=128296069817856 width=48) (actual
time=2370.40..2370.40 rows=0 loops=1)
         ->  Sort  (cost=445661544.88..445661544.88 rows=1016818949 width=24)  (actual
time=2370.40..2370.40 rows=0 loops=1)
             ->  Nested Loop  (cost=0.00..114812512.63 rows=1016818949 width=24)  (actual
time=2370.14..2370.14 rows=0 loops=1)
                 ->  Merge Join  (cost=0.00..9672.91 rows=36141 width=12) (actual time=2370.13..2370.13
rows=0 loops=1)
                     ->  Index Scan using index1 on ba s  (cost=0.00..4760.07 rows=47710 width=4) (actual
time=2370.12..2370.12 rows=0 loops=1)
                     ->  Index Scan using index3 on oa v  (cost=0.00..3862.28 rows=191812 width=8)
                 ->  Index Scan using index1 on ba w  (cost=0.00..1910.45 rows=84404 width=12)
         ->  Sort  (cost=633590312.49..633590312.49 rows=1349141523 width=24)
             ->  Nested Loop  (cost=0.00..152331639.70 rows=1349141523 width=24)
                 ->  Merge Join  (cost=0.00..8280.50 rows=47953 width=12)
                     ->  Index Scan using index1 on ba s  (cost=0.00..3181.03 rows=63303 width=4)
                     ->  Index Scan using index3 on oa v  (cost=0.00..3862.28 rows=191812 width=8)
                 ->  Index Scan using index1 on ba w  (cost=0.00..1910.45 rows=84404 width=12)
Total runtime: 4870.16 msec
```

c) Execution Time = 4.870160

```
NOTICE:  QUERY PLAN:
```

Append (cost=0.00..1604786058773.36 rows=128296069833794 width=48) (actual time=3965.59..3965.59 rows=0 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..19291.60 rows=15937 width=4) (actual time=1595.16..1595.16 rows=0 loops=1)
-> Index Scan using index1 on ba y (cost=0.00..19291.60 rows=15937 width=4) (actual time=1595.15..1595.15 rows=0 loops=1)
SubPlan
-> Index Scan using index1 on ba w (cost=0.00..40.59 rows=319 width=12)
-> Subquery Scan *SELECT* 2 (cost=1079251857.37..1604786039481.76 rows=128296069817856 width=48) (actual time=2370.41..2370.41 rows=0 loops=1)
-> Merge Join (cost=1079251857.37..1604786039481.76 rows=128296069817856 width=48) (actual time=2370.40..2370.40 rows=0 loops=1)
-> Sort (cost=445661544.88..445661544.88 rows=1016818949 width=24) (actual time=2370.40..2370.40 rows=0 loops=1)
-> Nested Loop (cost=0.00..114812512.63 rows=1016818949 width=24) (actual time=2370.14..2370.14 rows=0 loops=1)
-> Merge Join (cost=0.00..9672.91 rows=36141 width=12) (actual time=2370.13..2370.13 rows=0 loops=1)
-> Index Scan using index1 on ba s (cost=0.00..4760.07 rows=47710 width=4) (actual time=2370.12..2370.12 rows=0 loops=1)
-> Index Scan using index3 on oa v (cost=0.00..3862.28 rows=191812 width=8)
-> Index Scan using index1 on ba w (cost=0.00..1910.45 rows=84404 width=12)
-> Sort (cost=633590312.49..633590312.49 rows=1349141523 width=24)
-> Nested Loop (cost=0.00..152331639.70 rows=1349141523 width=24)
-> Merge Join (cost=0.00..8280.50 rows=47953 width=12)
-> Index Scan using index1 on ba s (cost=0.00..3181.03 rows=63303 width=4)
-> Index Scan using index3 on oa v (cost=0.00..3862.28 rows=191812 width=8)
-> Index Scan using index1 on ba w (cost=0.00..1910.45 rows=84404 width=12)
Total runtime: 4870.16 msec

c) Execution Time = 4.870160

d) Result tuples = 1

B. Unicode Dewey

a) SQL query

select *nca*('121f23','1G221')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.49..0.49 rows=1 loops=1)
Total runtime: 0.58 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.49..0.49 rows=1 loops=1)

Total runtime: 0.58 msec

c) Execution Time = 0.00058

d) Result tuples = 1

2. Αριστερός "Φύλλο" κόμβος με το μικρότερο postnum (= 1) – Μεσαίος "Φύλλο" κόμβος (= 126623)

A. Agrawal

a) SQL query

```
(select Y.post
from PInterval Y
where Y.index <= 1 and Y.post > 126623 and
NOT EXISTS ( select *
                from PInterval W where W.index <= 1 and 126623 < W.post and
                Y.index <= W.post and W.post < Y.post)
)
UNION ALL
(select R.post
from
        (select W.post, W.index, W.father_post
        from (   select S.post AS T
                from PInterval S
                where S.index <= 1 and S.post > 1) Z, DPInterval V, PInterval W
        where V.post = Z.T and W.index <= V.ancestor and W.post >= V.ancestor) R,

        (select W.post, W.index, W.father_post
        from (   select S.post as T
                from PInterval S
                where S.index <= 126623 and S.post > 126623) Z, DPInterval V, PInterval W
        where V.post = Z.T and W.index <=V.ancestor and W.post >= V.ancestor) S
where R. father_post = S. father_post)
```

b) Execution Plan

NOTICE: QUERY PLAN:

Append (cost=0.00..1604786058773.36 rows=128296069833794 width=48) (actual time=3965.59..3965.59 rows=0 loops=1)
-> Subquery Scan *SELECT* 1 (cost=0.00..19291.60 rows=15937 width=4) (actual time=1595.16..1595.16 rows=0 loops=1)
-> Index Scan using index1 on ba y (cost=0.00..19291.60 rows=15937 width=4) (actual time=1595.15..1595.15 rows=0 loops=1)
SubPlan
-> Index Scan using index1 on ba w (cost=0.00..40.59 rows=319 width=12)
-> Subquery Scan *SELECT* 2 (cost=1079251857.37..1604786039481.76 rows=128296069817856 width=48) (actual time=2370.41..2370.41 rows=0 loops=1)

```
    -> Merge Join  (cost=1079251857.37..1604786039481.76 rows=128296069817856 width=48) (actual
time=2370.40..2370.40 rows=0 loops=1)
        -> Sort  (cost=445661544.88..445661544.88 rows=1016818949 width=24) (actual
time=2370.40..2370.40 rows=0 loops=1)
            -> Nested Loop  (cost=0.00..114812512.63 rows=1016818949 width=24) (actual
time=2370.14..2370.14 rows=0 loops=1)
                -> Merge Join  (cost=0.00..9672.91 rows=36141 width=12) (actual time=2370.13..2370.13
rows=0 loops=1)
                    -> Index Scan using index1 on ba s  (cost=0.00..4760.07 rows=47710 width=4) (actual
time=2370.12..2370.12 rows=0 loops=1)
                    -> Index Scan using index3 on oa v  (cost=0.00..3862.28 rows=191812 width=8)
                -> Index Scan using index1 on ba w  (cost=0.00..1910.45 rows=84404 width=12)
        -> Sort  (cost=633590312.49..633590312.49 rows=1349141523 width=24)
            -> Nested Loop  (cost=0.00..152331639.70 rows=1349141523 width=24)
                -> Merge Join  (cost=0.00..8280.50 rows=47953 width=12)
                    -> Index Scan using index1 on ba s  (cost=0.00..3181.03 rows=63303 width=4)
                    -> Index Scan using index3 on oa v  (cost=0.00..3862.28 rows=191812 width=8)
                -> Index Scan using index1 on ba w  (cost=0.00..1910.45 rows=84404 width=12)
Total runtime: 3702.879 msec

NOTICE:  QUERY PLAN:

Append  (cost=0.00..1604786058773.36 rows=128296069833794 width=48) (actual time=3965.59..3965.59
rows=0 loops=1)
 -> Subquery Scan *SELECT* 1  (cost=0.00..19291.60 rows=15937 width=4) (actual
time=1595.16..1595.16 rows=0 loops=1)
    -> Index Scan using index1 on ba y  (cost=0.00..19291.60 rows=15937 width=4) (actual
time=1595.15..1595.15 rows=0 loops=1)
        SubPlan
          -> Index Scan using index1 on ba w  (cost=0.00..40.59 rows=319 width=12)
 -> Subquery Scan *SELECT* 2  (cost=1079251857.37..1604786039481.76 rows=128296069817856
width=48) (actual time=2370.41..2370.41 rows=0 loops=1)
    -> Merge Join  (cost=1079251857.37..1604786039481.76 rows=128296069817856 width=48) (actual
time=2370.40..2370.40 rows=0 loops=1)
        -> Sort  (cost=445661544.88..445661544.88 rows=1016818949 width=24) (actual
time=2370.40..2370.40 rows=0 loops=1)
            -> Nested Loop  (cost=0.00..114812512.63 rows=1016818949 width=24) (actual
time=2370.14..2370.14 rows=0 loops=1)
                -> Merge Join  (cost=0.00..9672.91 rows=36141 width=12) (actual time=2370.13..2370.13
rows=0 loops=1)
                    -> Index Scan using index1 on ba s  (cost=0.00..4760.07 rows=47710 width=4) (actual
time=2370.12..2370.12 rows=0 loops=1)
                    -> Index Scan using index3 on oa v  (cost=0.00..3862.28 rows=191812 width=8)
                -> Index Scan using index1 on ba w  (cost=0.00..1910.45 rows=84404 width=12)
        -> Sort  (cost=633590312.49..633590312.49 rows=1349141523 width=24)
            -> Nested Loop  (cost=0.00..152331639.70 rows=1349141523 width=24)
                -> Merge Join  (cost=0.00..8280.50 rows=47953 width=12)
                    -> Index Scan using index1 on ba s  (cost=0.00..3181.03 rows=63303 width=4)
                    -> Index Scan using index3 on oa v  (cost=0.00..3862.28 rows=191812 width=8)
                -> Index Scan using index1 on ba w  (cost=0.00..1910.45 rows=84404 width=12)
Total runtime: 3702.879 msec
```

c) Execution Time = 3.702879

d) Result tuples = 1

B. Unicode Dewey

a) SQL query

select *nca*('121f23','12f226a91c')

b) Execution Plan

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.50..0.51 rows=1 loops=1)
Total runtime: 0.55 msec

NOTICE: QUERY PLAN:

Result (cost=0.00..0.01 rows=1 width=0) (actual time=0.50..0.51 rows=1 loops=1)
Total runtime: 0.55 msec

c) Execution Time = 0.00055

d) Result tuples = 1

3. Μεσαίος "Φύλλο" κόμβος (= 126623) - Δεξιός "Φύλλο" κόμβος με το μεγαλύτερο postnum (= 253211)

A. Agrawal

a) SQL query

```
(select Y.post
from PInterval Y
where Y.index <= 126623 and Y.post > 253211 and
NOT EXISTS ( select *
                    from PInterval W
                    where W.index <= 126623 and 253211 < W.post and
                    Y.index <= W.post and W.post < Y.post)
)
UNION ALL
(select R.post
from
          (select W.post, W.index, W.father_post
          from (   select S.post AS T
                    from PInterval S
                    where S.index <= 126623 and S.post > 126623) Z, DPInterval V, PInterval W
          where V.post = Z.T and W.index <= V.ancestor and W.post >= V.ancestor) R,
```

```
        (select W.post, W.index, W.father_post
        from (   select S.post as T
                 from PInterval S
                 where S.index <= 253211 and S.post > 253211) Z, DPInterval V, PInterval W
        where V.post = Z.T and W.index <=V.ancestor and W.post >= V.ancestor) S
where R. father_post = S. father_post)
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Append  (cost=0.00..1604786058773.36 rows=128296069833794 width=48) (actual time=3965.59..3965.59
rows=0 loops=1)
 ->  Subquery Scan *SELECT* 1  (cost=0.00..19291.60 rows=15937 width=4) (actual
time=1595.16..1595.16 rows=0 loops=1)
     ->  Index Scan using index1 on ba y  (cost=0.00..19291.60 rows=15937 width=4) (actual
time=1595.15..1595.15 rows=0 loops=1)
          SubPlan
           ->  Index Scan using index1 on ba w  (cost=0.00..40.59 rows=319 width=12)
 ->  Subquery Scan *SELECT* 2  (cost=1079251857.37..1604786039481.76 rows=128296069817856
width=48) (actual time=2370.41..2370.41 rows=0 loops=1)
     ->  Merge Join  (cost=1079251857.37..1604786039481.76 rows=128296069817856 width=48) (actual
time=2370.40..2370.40 rows=0 loops=1)
          ->  Sort  (cost=445661544.88..445661544.88 rows=1016818949 width=24)  (actual
time=2370.40..2370.40 rows=0 loops=1)
               ->  Nested Loop  (cost=0.00..114812512.63 rows=1016818949 width=24)  (actual
time=2370.14..2370.14 rows=0 loops=1)
                    ->  Merge Join  (cost=0.00..9672.91 rows=36141 width=12) (actual time=2370.13..2370.13
rows=0 loops=1)
                         ->  Index Scan using index1 on ba s  (cost=0.00..4760.07 rows=47710 width=4) (actual
time=2370.12..2370.12 rows=0 loops=1)
                         ->  Index Scan using index3 on oa v  (cost=0.00..3862.28 rows=191812 width=8)
                    ->  Index Scan using index1 on ba w  (cost=0.00..1910.45 rows=84404 width=12)
          ->  Sort  (cost=633590312.49..633590312.49 rows=1349141523 width=24)
               ->  Nested Loop  (cost=0.00..152331639.70 rows=1349141523 width=24)
                    ->  Merge Join  (cost=0.00..8280.50 rows=47953 width=12)
                         ->  Index Scan using index1 on ba s  (cost=0.00..3181.03 rows=63303 width=4)
                         ->  Index Scan using index3 on oa v  (cost=0.00..3862.28 rows=191812 width=8)
                    ->  Index Scan using index1 on ba w  (cost=0.00..1910.45 rows=84404 width=12)
Total runtime: 4691.551 msec

NOTICE:  QUERY PLAN:

Append  (cost=0.00..1604786058773.36 rows=128296069833794 width=48) (actual time=3965.59..3965.59
rows=0 loops=1)
 ->  Subquery Scan *SELECT* 1  (cost=0.00..19291.60 rows=15937 width=4) (actual
time=1595.16..1595.16 rows=0 loops=1)
     ->  Index Scan using index1 on ba y  (cost=0.00..19291.60 rows=15937 width=4) (actual
time=1595.15..1595.15 rows=0 loops=1)
          SubPlan
```

```
          -> Index Scan using index1 on ba w  (cost=0.00..40.59 rows=319 width=12)
 -> Subquery Scan *SELECT* 2  (cost=1079251857.37..1604786039481.76 rows=128296069817856 width=48) (actual time=2370.41..2370.41 rows=0 loops=1)
      -> Merge Join  (cost=1079251857.37..1604786039481.76 rows=128296069817856 width=48) (actual time=2370.40..2370.40 rows=0 loops=1)
           -> Sort  (cost=445661544.88..445661544.88 rows=1016818949 width=24) (actual time=2370.40..2370.40 rows=0 loops=1)
                -> Nested Loop  (cost=0.00..114812512.63 rows=1016818949 width=24) (actual time=2370.14..2370.14 rows=0 loops=1)
                     -> Merge Join  (cost=0.00..9672.91 rows=36141 width=12) (actual time=2370.13..2370.13 rows=0 loops=1)
                          -> Index Scan using index1 on ba s  (cost=0.00..4760.07 rows=47710 width=4) (actual time=2370.12..2370.12 rows=0 loops=1)
                          -> Index Scan using index3 on oa v  (cost=0.00..3862.28 rows=191812 width=8)
                     -> Index Scan using index1 on ba w  (cost=0.00..1910.45 rows=84404 width=12)
           -> Sort  (cost=633590312.49..633590312.49 rows=1349141523 width=24)
                -> Nested Loop  (cost=0.00..152331639.70 rows=1349141523 width=24)
                     -> Merge Join  (cost=0.00..8280.50 rows=47953 width=12)
                          -> Index Scan using index1 on ba s  (cost=0.00..3181.03 rows=63303 width=4)
                          -> Index Scan using index3 on oa v  (cost=0.00..3862.28 rows=191812 width=8)
                     -> Index Scan using index1 on ba w  (cost=0.00..1910.45 rows=84404 width=12)
Total runtime: 4691.551 msec
```

c) Execution Time = 4.691551

d) Result tuples = 1

B. Unicode Dewey

a) SQL query

```
select nca('12f226a91c','1G221')
```

b) Execution Plan

```
NOTICE:  QUERY PLAN:

Result  (cost=0.00..0.01 rows=1 width=0) (actual time=0.47..0.48 rows=1 loops=1)
Total runtime: 0.57 msec

NOTICE:  QUERY PLAN:

Result  (cost=0.00..0.01 rows=1 width=0) (actual time=0.47..0.48 rows=1 loops=1)
Total runtime: 0.57 msec
```

c) Execution Time = 0.00057

d) Returned Tuples = 1