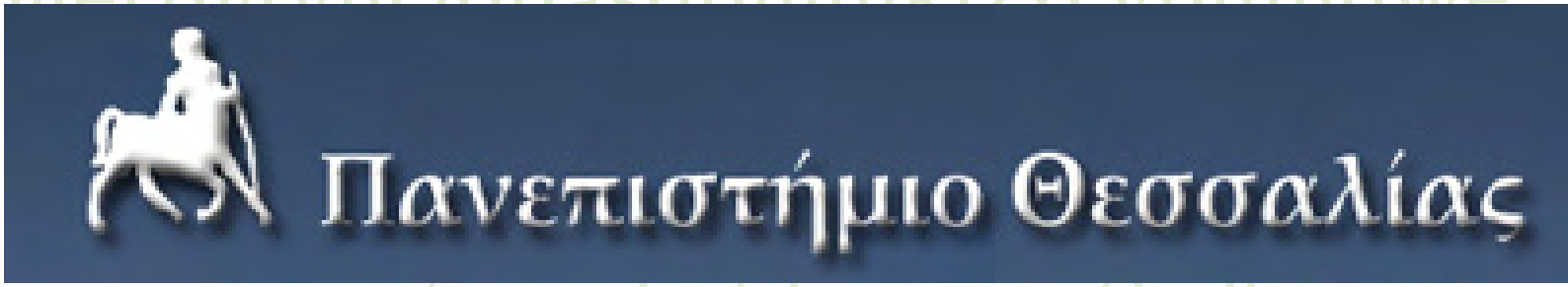


Reducing the space and time requirements of LZ-index using the XBW transformation

Διπλωματική Εργασία για την απόκτηση του Διπλώματος του Μηχανικού Ηλεκτρονικών Υπολογιστών Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας

Μιχελής Παναγιώτης

Μάρτιος 2009
Βόλος

Επιβλέπων Καθηγητής: Μποζάνης Παναγιώτης
Συμβουλευτική Επιτροπή: Κατσαρός Δημήτριος

Στους Γονείς Μου

Περιεχόμενα

Εισαγωγή

0.1 Full-Text Search Problem

Ένα από τα πλέον κλασικά προβλήματα στην επιστήμη των υπολογιστών είναι η αναζήτηση και ανάκτηση κειμένου, γνωστό και ως text searching/retrieval.

Στο πρόβλημα αυτό υπάρχουν τρία πολύ συνηθισμένα είδη ερωτήσεων (queries):

1) Πληροφόρηση για τον αν υπάρχει η φράση (phrase ή pattern) P μέσα στο κείμενό μας T.
2) Πληροφόρηση για τον αριθμό των στιγμιότυπων της φράσης P μέσα στο κείμενο.
3) Πληροφόρηση για τα σημεία μέσα στο κείμενο T στα οποία ξεκινάει η φράση P.

Τα τρία αυτά είδη ερωτήσεων αναφέρονται μαζί και ως full-text search problem.

Στη σημερινή εποχή ο όγκος των πληροφοριών είναι τεράστιος. Ενδεικτικά μπορούμε να αναφέρουμε ορισμένα είδη βάσεων δεδομένων οι οποίες είναι τεράστιες σε μέγεθος. Τέτοιες μπορεί να είναι βάσεις δεδομένων DNA, πρωτεϊνών, MIDI pitch sequences, κώδικας προγραμμάτων κτλ. Επίσης, το full-text search βρίσκει πολύ σημαντική εφαρμογή στην αναζήτηση στο διαδίκτυο.

Για να επιλύσουμε αυτό το πρόβλημα πρέπει να περάσουμε όλο το κείμενο σε μια δομή ευρετηρίου μέσω της οποίας θα κάνουμε τις παραπάνω τρεις ερωτήσεις. Όλες οι δομές που έχουν παρουσιαστεί από την επιστημονική κοινότητα δίνουν καλύτερα αποτελέσματα από τη σειριακή αναζήτηση του κειμένου όσο αφορά τον χρόνο. Πολλές όμως από αυτές τις δομές καταλάμβαναν πολύ περισσότερο χώρο από το ίδιο το κείμενο. Λύση στο πρόβλημα αυτό ήρθαν να δώσουν δομές όπως ο LZ-Index οι οποίες καταλάμβαναν πολύ λιγότερο χώρο από το κείμενο και ταυτόχρονα ήταν πιο γρήγορες από υπάρχουσες δομές.

0.2 Σκοπός της διπλωματικής εργασίας

Η παρούσα διπλωματική εργασία έχει σαν σκοπό να χρησιμοποιήσει τον μετασχηματισμό xbw για labeled trees με τον οποίο θα αντικαταστήσουμε κάποιες δομές του LZ-Index με σκοπό να γίνει πιο γρήγορος και να απαιτεί λιγότερο χώρο.

0.3 Οργάνωση της διπλωματικής εργασίας

Στο κεφάλαιο 1 θα παρουσιάσουμε αναλυτικότερα το πρόβλημα full-text search, θα δούμε γιατί είναι τόσο σημαντικό και τέλος θα αναφερθούμε σε ορισμένους από τους υπάρχοντες αλγορίθμους που επιλύουν το πρόβλημα αυτό. Στο κεφάλαιο 2 θα περιγράψουμε αναλυτικά τον αλγόριθμο LZ-Index και στο κεφάλαιο 3 τον μετασχηματισμό XBW. Στο κεφάλαιο 4 θα δούμε πως ο αλγόριθμος LZ-Index συνδυάζεται με τον μετασχηματισμό XBW.

Κεφάλαιο 1
Full-Text Search Problem

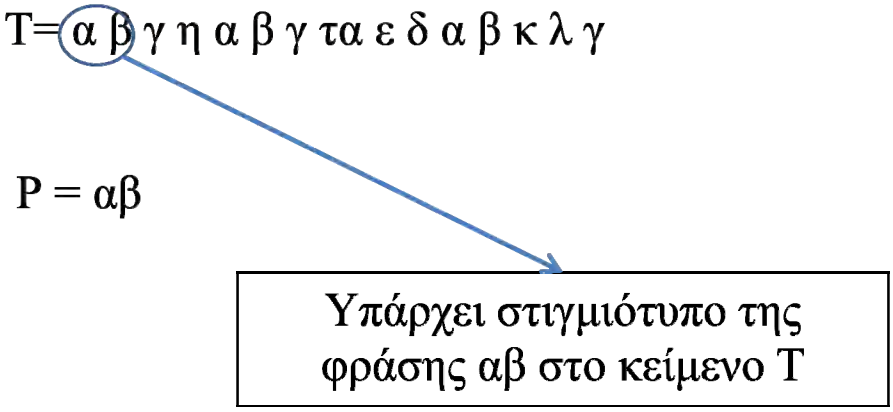
1.1 Παρουσίαση του Full-text Search problem

Στο συγκεκριμένο πρόβλημα έχουμε ένα κείμενο Τ[1...κ] με χαρακτήρες από ένα αλφάβητο Σ και δοσμένης μιας φράσης Ρ κάνουμε ερωτήσεις σχετικά με την ύπαρξη, τις επαναλήψεις και την θέση της φράσης Ρ.

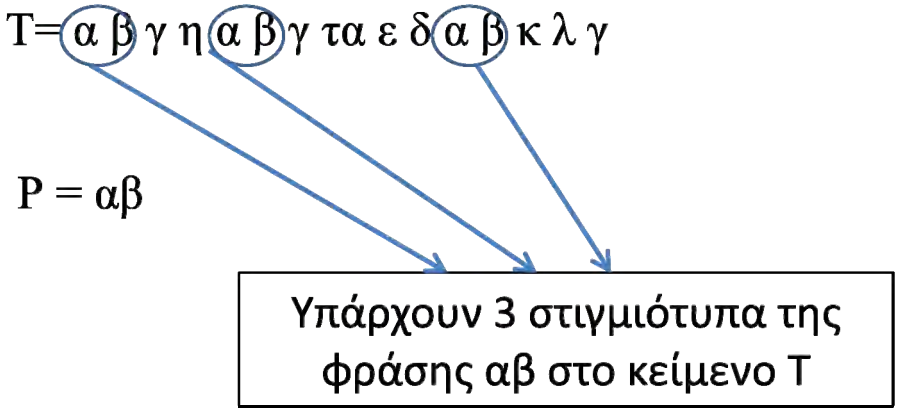
1.2 Ερωτήσεις στις οποίες απαντάει μια δομή Full-text Search

Υπάρχουν τριών ειδών ερωτήσεις για μια δοσμένη φράση Ρ[1...q] οι οποίες αναφέρονται παρακάτω και παρουσιάζονται καλύτερα μέσω παραδειγμάτων:

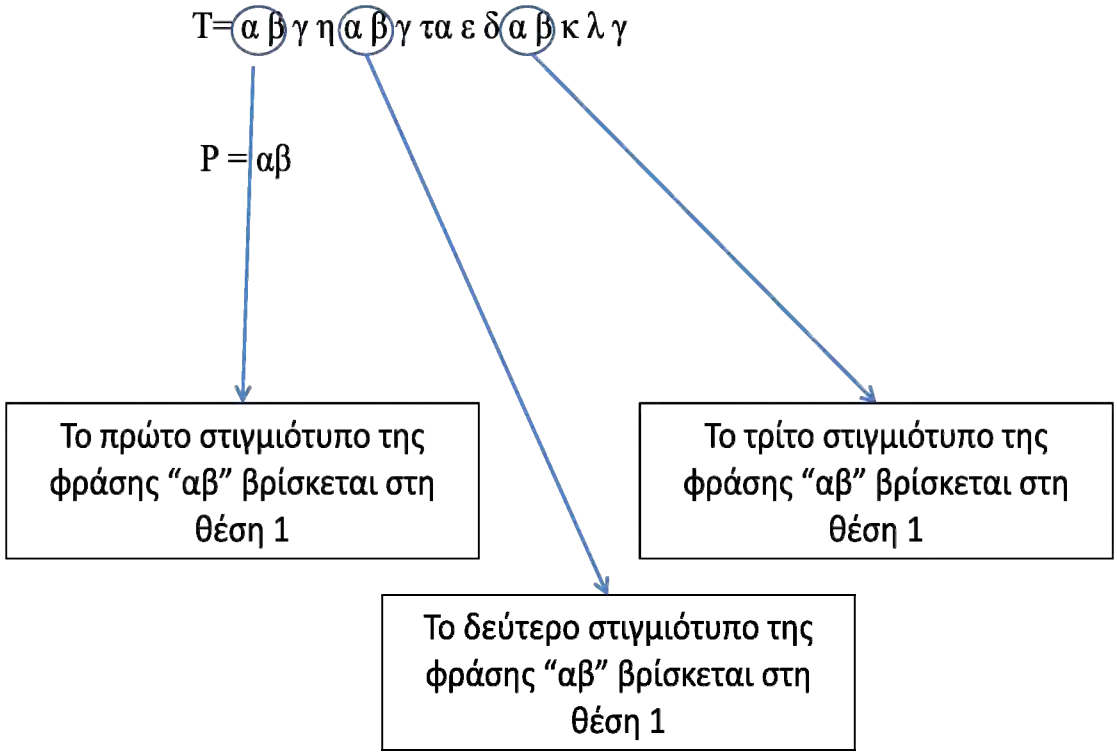
1) Πληροφόρηση για τον αν υπάρχει η φράση Ρ μέσα στο κείμενό μας Τ.



2) Πληροφόρηση για τον αριθμό των στιγμιότυπων της φράσης P μέσα στο κείμενο.



3) Πληροφόρηση για τα σημεία μέσα στο κείμενο T στα οποία ξεκινάει η φράση P.



1.3 Δομή του Full-text Search

Για να απαντηθούν όμως οι παραπάνω ερωτήσεις χρειάζεται να κατασκευάσουμε μια δομή η οποία θα περιέχει το κείμενο. Φυσικά, μια απλοϊκή δομή θα μπορούσαμε να πούμε ότι είναι το ίδιο το κείμενο καθώς οι ερωτήσεις δύνανται να απαντηθούν ψάχνοντας σειριακά σε όλο το κείμενο για να βρούμε τα στιγμιότυπα της φράσης που μας ενδιαφέρει. Αυτός ο τρόπος είναι ο πλέον χρονοβόρος και θα μας οδηγούσε σε πολύ μεγάλες καθυστερήσεις καθώς θα αυξανόταν το μέγεθος του κειμένου. Για το λόγο αυτό είναι απαραίτητο να χρησιμοποιήσουμε ειδικές δομές οι οποίες θα μας εξοικονομούν χρόνο.

Οι ειδικές αυτές δομές ονομάζονται index και με απλοϊκό τρόπο μπορούμε να πούμε ότι αποτελούν το ευρετήριο για κάθε χαρακτήρα του κειμένου. Παρόλο όμως που μας δίνουν ικανοποιητικά αποτελέσματα όσο αφορά τον χρόνο, απαιτούν συνήθως μεγάλο αποθηκευτικό χώρο και σε συνδυασμό με την ύπαρξη του αρχικού κειμένου οι απαιτήσεις για αποθηκευτικό χώρο γίνονται πολύ μεγάλες. Ενδεικτικά να αναφέρουμε ότι οι απαιτήσεις μιας τέτοιας δομής ήταν από τέσσερις μέχρι και είκοσι φορές το μέγεθος του αρχικού κειμένου. Αυτό είχε σαν αποτέλεσμα πολλές φορές να χωράει στην κύρια μνήμη το αρχικό κείμενο αλλά όχι το index το οποίο ήταν απαραίτητο για την επεξεργασία. Επομένως η αποθήκευση του index έπρεπε να γίνει υποχρεωτικά στη δευτερεύουσα μνήμη κάτι που είχε σαν αποτέλεσμα μεγάλες καθυστερήσεις στην επεξεργασία.

Ταυτόχρονα οι περισσότεροι επιστήμονες που έφτιαχναν εφαρμογές που χρησιμοποιούσαν κάποιου είδους index απαιτούσαν αυτό να αποθηκεύεται στην κυρίως μνήμη ώστε ο απαιτούμενος χρόνος επεξεργασίας να μειωνόταν όσο γινόταν περισσότερο. Αυτό αυτόματα δημιούργησε την ανάγκη να φτιαχτούν δομές index οι οποίες εκτός από ικανοποιητικούς χρόνους θα είχαν και μικρές απαιτήσεις σε αποθηκευτικό χώρο.

Η απαίτηση ήταν γρήγορη πρόσβαση στο κείμενο χρησιμοποιώντας όσο γίνεται λιγότερο χώρο.

1.4 Compressed full-text self-indices

Η σημερινή τάση στον χώρο η οποία καλύπτει την παραπάνω απαίτηση είναι οι ονομαζόμενες δομές *compressed full-text self-indexes*. Οι δομές αυτού του τύπου γίνονται καλύτερα κατανοητές αν τις χωρίσουμε στα βασικά τους χαρακτηριστικά. Το self-index αναφέρεται στο χαρακτηριστικό εκείνο της δομής το οποίο επιτρέπει να κάνουμε αναζήτηση και ανάκτηση μια φράσης χωρίς να χρειάζεται να αποθηκεύσουμε το αρχικό κείμενο. Ο χαρακτηρισμός compressed index αναφέρεται στο ότι οι απαιτήσεις τις δομής σε αποθηκευτικό χώρο είναι είτε συγκρίσιμες είτε κατά πολύ μικρότερες σε σχέση με το μέγεθος του αρχικού κειμένου. Τελικά, μια δομή *compressed full-text self-index* αντικαθιστά το αρχικό κείμενο με μια λιγότερο απαιτητική σε χώρο μορφή, ενώ παράλληλα προσφέρει τη δυνατότητα προσπέλασης του κειμένου και όλα αυτά χωρίς να υστερεί στον χρόνο που απαιτείται.

1.4.1 Που χρησιμοποιούνται

Αν θέλαμε να δώσουμε μια γρήγορη και γενική απάντηση θα μπορούσαμε να πούμε παντού. Μια πιο σωστή προσέγγιση θα ήταν να πούμε ότι χρησιμοποιούνται οπουδήποτε υπάρχει η απαίτηση για οποιουδήποτε είδους αναζήτηση. Ακόμα και σε ένα αρχείο ήχου η αναζήτηση μιας συγκεκριμένης μελωδίας γίνεται με αυτές τις δομές. Αξίζει όμως να δούμε μερικές εφαρμογές στις οποίες η αναζήτηση γίνεται σε πολύ μεγάλα αρχεία και ο χρόνος αναζήτησης είναι επίσης πολύ σημαντικός.

1.4.2 PubMed

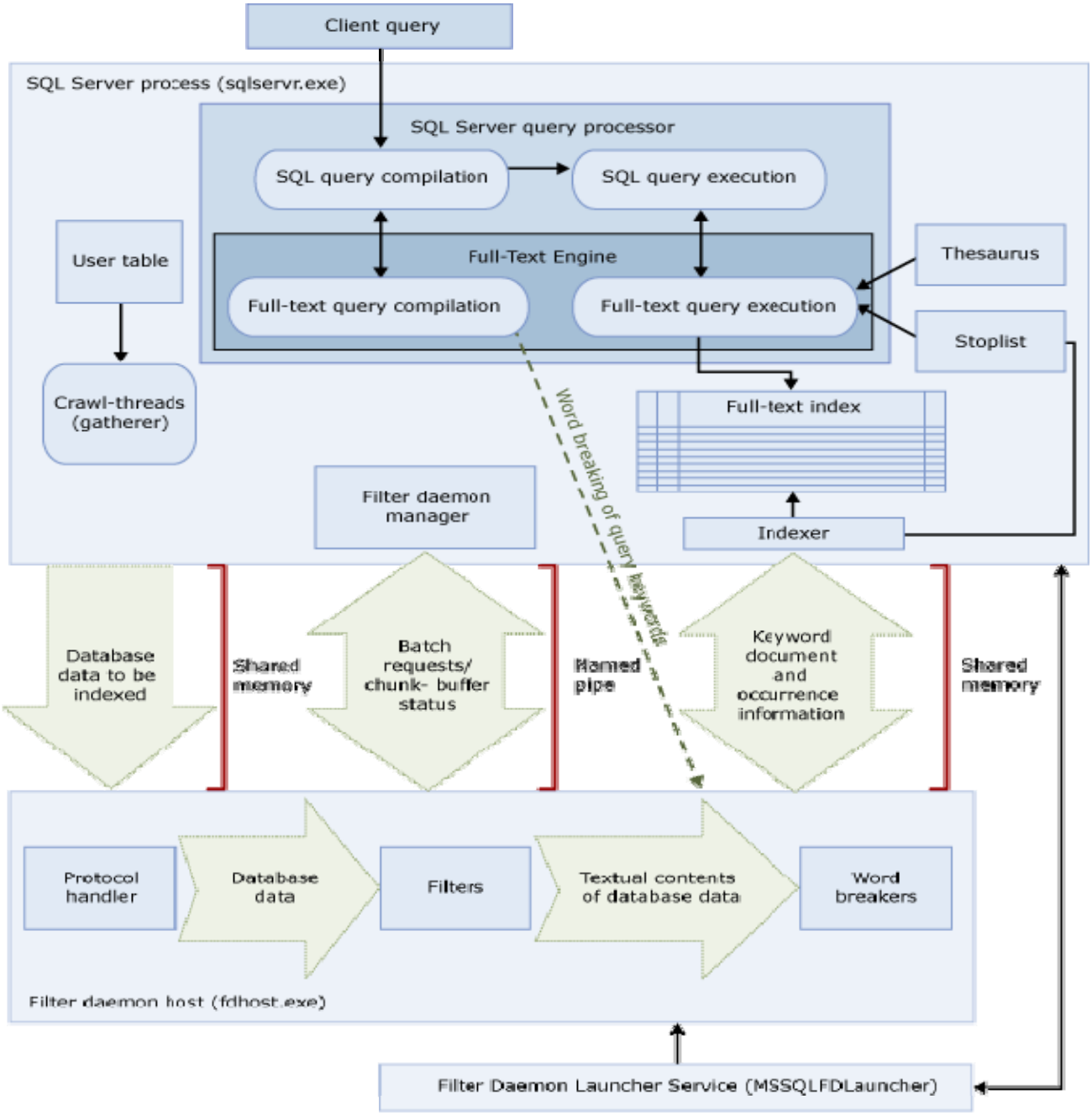
Η PubMed είναι μια μηχανή αναζήτησης για την βάση δεδομένων MEDLINE η οποία περιέχει παραπομπές αλλά και πλήρη άρθρα σε θέματα πάνω στις επιστήμες υγείας και πάνω σε βιοϊατρικά θέματα. Ενδεικτικά για το μέγεθος της αναφέρουμε ότι στις 22 Ιανουαρίου 2009 υπήρχαν περίπου 18.600.000 παραπομπές οι οποίες ξεκινούσαν χρονολογικά από το 1865.

1.4.3 GenBank

Η GenBank είναι μια βάση δεδομένων από γενετικές ακολουθίες. Περιέχει όλες τις γνωστές έως σήμερα ακολουθίες DNA. Στην τελευταία της έκδοση (Release 170.0 February 2009) αποτελείται συνολικά από 1426 αρχεία τα οποία περιέχουν όλες τις ακολουθίες DNA. Το μέγεθος της δομής index που χρησιμοποιεί ανέρχεται σε 65 αρχεία. Το συνολικό μέγεθος της βάσης είναι περίπου 417 GB. Στον παρακάτω πίνακα βλέπουμε την αύξηση του μεγέθους της από τον Απρίλιο του 2002 μέχρι σήμερα.

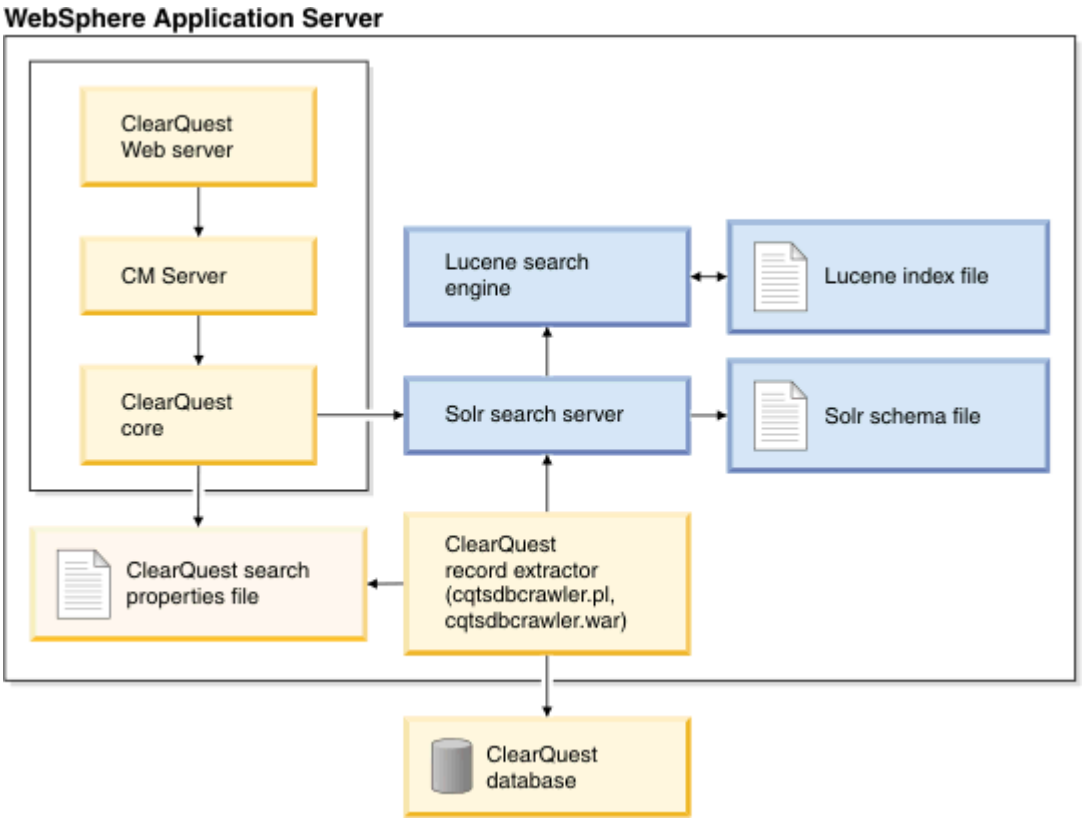
Release	Date	Base Pairs	Entries
129	Apr 2002	692266338	172768
130	Jun 2002	3267608441	397502
131	Aug 2002	3848375582	427771
132	Oct 2002	3892435593	434224
133	Dec 2002	6702372564	597042
134	Feb 2003	6705740844	597345
135	Apr 2003	6897080355	596818
136	Jun 2003	6992663962	607155
137	Aug 2003	7144761762	593801
138	Oct 2003	8662242833	1683437
139	Dec 2003	14523454868	2547094
140	Feb 2004	22804145885	3188754
141	Apr 2004	24758556215	4112532
142	Jun 2004	25592758366	4353890
143	Aug 2004	28128611847	4427773
144	Oct 2004	30871590379	5285276
145	Dec 2004	35009256228	5410415
146	Feb 2005	38076300084	6111782
147	Apr 2005	39523208346	6685746
148	Jun 2005	46767232565	8711924
149	Aug 2005	53346605784	10276161
150	Oct 2005	56162807647	11169448
151	Dec 2005	59638900034	12088491
152	Feb 2006	63183065091	12465546
153	Apr 2006	67488612571	13573144
154	Jun 2006	78858635822	17733973
155	Aug 2006	80369977826	17960667
156	Oct 2006	81127502509	18500772
157	Dec 2006	81611376856	18540918
158	Feb 2007	86043478524	19421576
159	Apr 2007	93022691867	23446831
160	Jun 2007	97102606459	23718400
161	Aug 2007	101964323738	25384475
162	Oct 2007	102003045298	25354041
163	Dec 2007	106505691578	26177471
164	Feb 2008	108635736141	27439206
165	Apr 2008	110500961400	26931049
166	Jun 2008	113639291344	39163548
167	Aug 2008	118593509342	40214247
168	Oct 2008	136085973423	46108952
169	Dec 2008	141374971004	48394838
170	Feb 2009	143797800446	49036947

Παρακάτω βλέπουμε το σχέδιο μιας βάσης δεδομένων

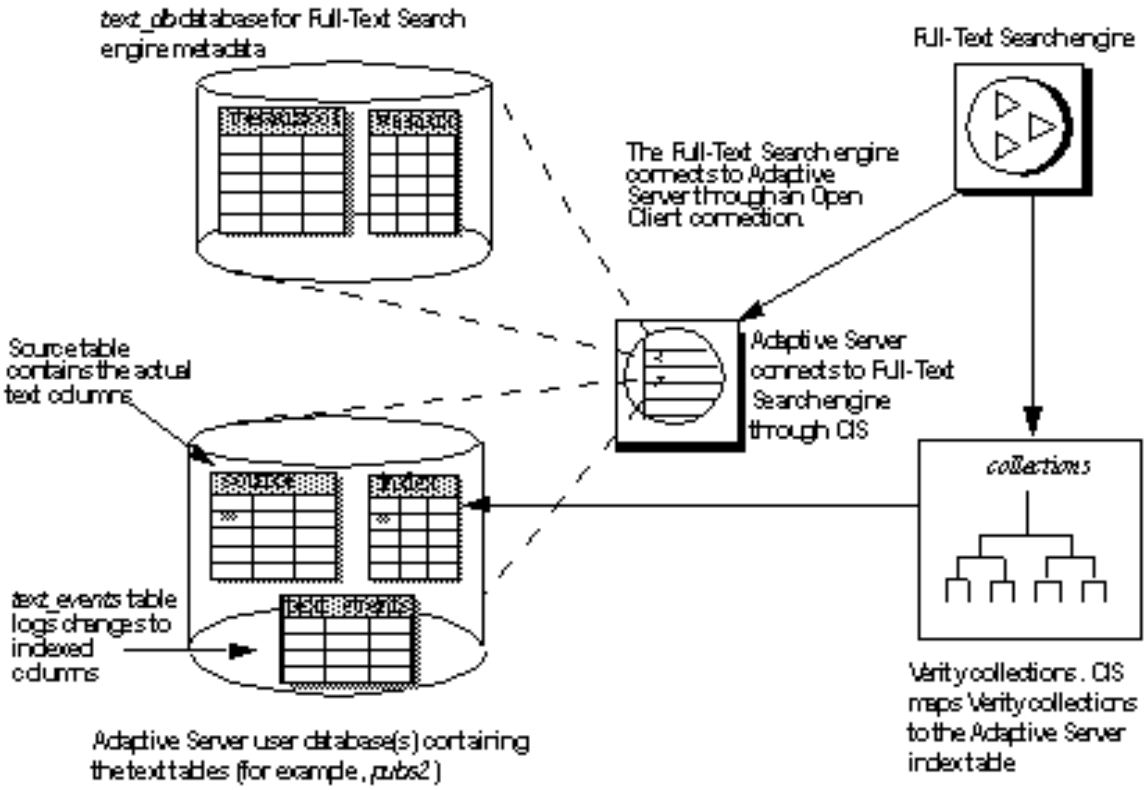


Όπως μπορούμε να παρατηρήσουμε οι περισσότερες λειτουργίες της βάσης περνάνε από έναν Full-text index.

Παρακάτω βλέπουμε μια υλοποίηση μιας υπηρεσίας με τον IBM WebSphere Application Server.



Επίσης, βλέπουμε ξεκάθαρα ότι μια βασική λειτουργία μέσα στον server είναι η αναζήτηση.



Εδώ βλέπουμε τον σχεδιασμό μιας μηχανής αναζήτησης η οποία βασίζεται μόνο σε δομή Full-text.

Θα μπορούσαμε να αναφέρουμε αμέτρητα ακόμα παραδείγματα που έχουν σαν βασική τους λειτουργία την αναζήτηση.

1.5 Γιατί είναι σημαντικές αυτές οι δομές;

Όπως είδαμε ειδικά στην περίπτωση των βιολογικών βάσεων δεδομένων το μέγεθος είναι τεράστιο. Επομένως, η εξοικονόμηση χώρου είναι πολύ σημαντική. Εξίσου σημαντική είναι και η ταχύτητα αφού αν αναζητούσαμε κάτι σειριακά σε τέτοια μεγέθη μπορεί να χρειαζόταν να περιμένουμε ακόμα και χρόνια για να πάρουμε απάντηση.

Κεφάλαιο 2
LZ-INDEX

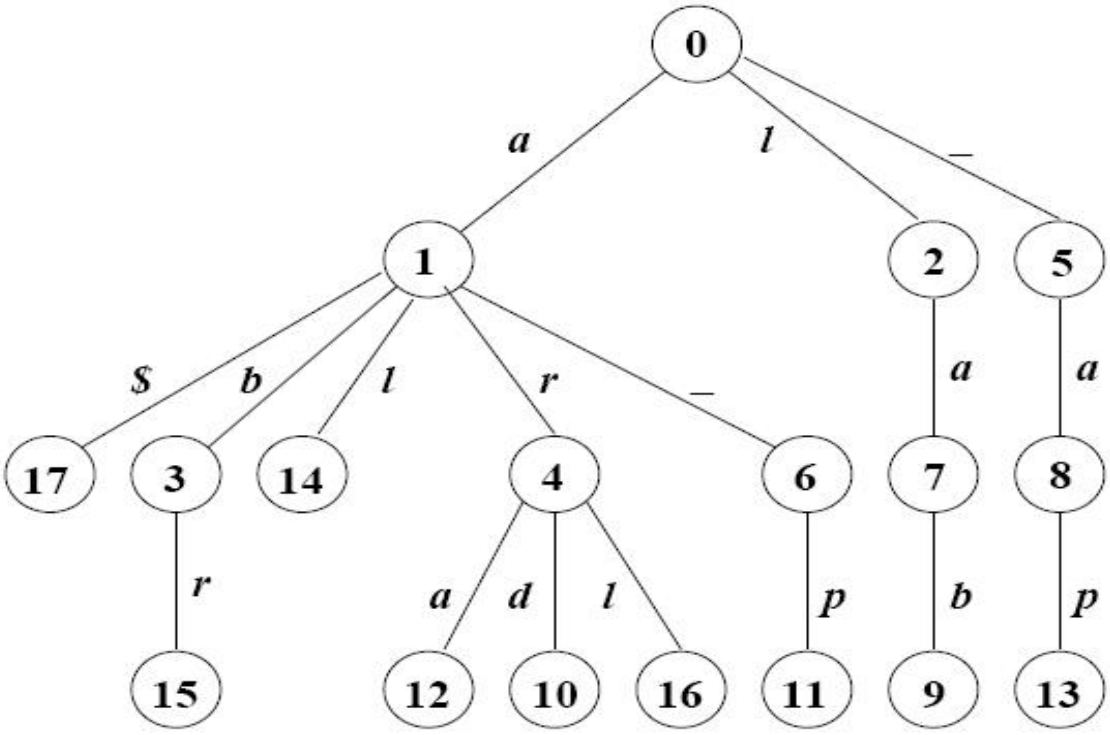
2.1 Περιγραφή LZ78

Ο αλγόριθμος LZ-Index βασίζεται στον αλγόριθμο συμπίεσης LZ78. Ο LZ-Index χρησιμοποιεί το δέντρο που παράγει ο LZ78 κατά την συμπίεση ενός κειμένου, όπως ακόμα και μερικές άλλες δομές του ίδιου αλγορίθμου. Ένα από τα χαρακτηριστικά του LZ-Index είναι ότι μπορούμε να αναπαράγουμε το κείμενο από το δέντρο που δημιουργεί ο αλγόριθμος.

Ο LZ78 δημιουργεί το δέντρο ως εξής:

> *Στην αρχή της συμπίεσης το λεξικό περιέχει ένα block b_0 μήκους 0.*
>
> *Κάθε βήμα της συμπίεσης γίνεται ως εξής: Αν έχουμε ένα πρόθεμα $T_{1..j}$ το οποίο έχει ήδη συμπιεστεί και βρίσκεται σε μια ακολουθία block $Z=b_1b_2….b_r$ τα οποία βρίσκονται ήδη στο λεξικό, τότε ψάχνουμε για το μεγαλύτερο πρόθεμα του υπόλοιπου κειμένου $T_{j+1…u}$ το οποίο είναι ένα block στο λεξικό. Όταν το βρούμε δημιουργούμε ένα νέο block b_{r+1}.*

Η ακολουθία “alabar_a_la_alabarda_para_apalabrarla$” μας δίνει το παρακάτω δέντρο μετά από συμπίεση με τον LZ78.



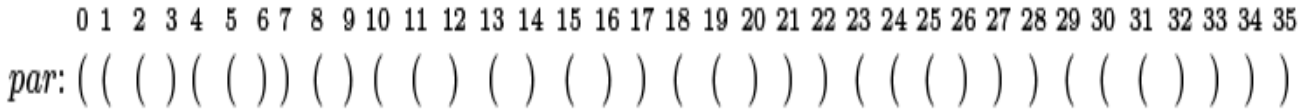
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	*l*	*ab*	*ar*	*_*	*a_*	*la*	*_a*	*lab*	*ard*	*a_p*	*ara*	*_ap*	*al*	*abr*	*arl*	*a$*

2.2 Η δομή του LZ-Index

Balanced parentheses representation

Πριν περιγράψουμε τη δομή του LZ-Index θα πρέπει να περιγράψουμε έναν τρόπο αναπαράστασης ενός δένδρου το οποίο θα χρησιμοποιήσουμε στην υλοποίηση του LZ-Index. Η αναπαράσταση αυτή ονομάζεται balanced parentheses representation και έχει δημιουργηθεί από τους Munro και Raman. Για να επιτύχουμε την αναπαράσταση αυτή διατρέχουμε το δέντρο με preorder σειρά και ανοίγουμε μια παράνθεση όταν επισκεπτόμαστε πρώτη φορά έναν κόμβο. Ακολούθως, όταν επισκεπτόμαστε έναν κόμβο για τελευταία φορά κλείνουμε την παρένθεση. Με αυτόν τον τρόπο κάθε κόμβος αναπαρίσταται από ένα ζευγάρι παρενθέσεων. Αναγνωρίζουμε έναν κόμβο x με το άνοιγμα της παρένθεσης που αντιστοιχεί σε αυτόν. Το υποδένδρο του x περιλαμβάνει όλους τους κόμβους των οποίων οι παρανθέσεις βρίσκονται ανάμεσα στις παρενθέσεις του x.

Παρακάτω βλέπουμε την balanced parentheses representation της φράσης “alabar_a_la_alabarda_para_apalabrarla$” για την οποία έχουμε ήδη δημιουργήσει το δέντρο LZ78.

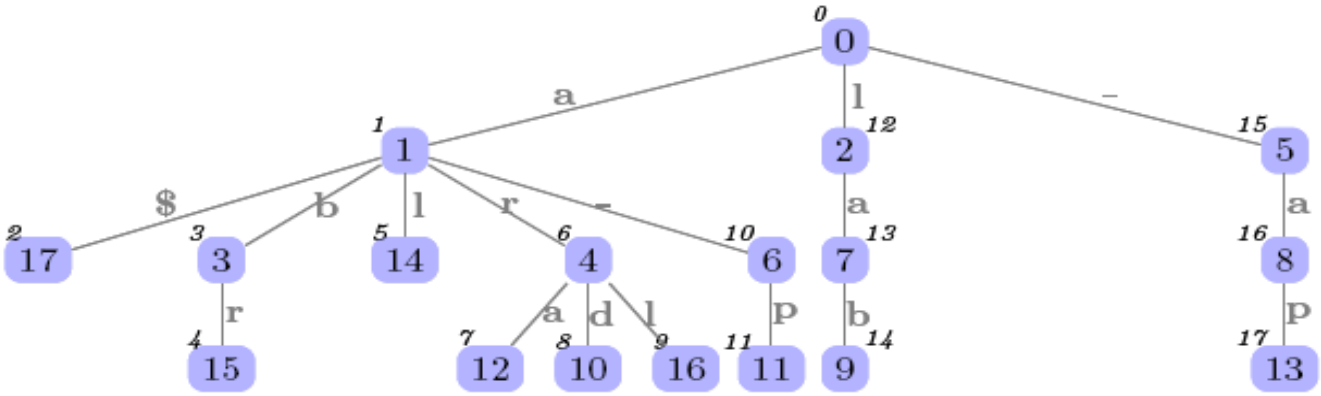


Ο LZ-Index αποτελείται από τα παρακάτω κομμάτια:

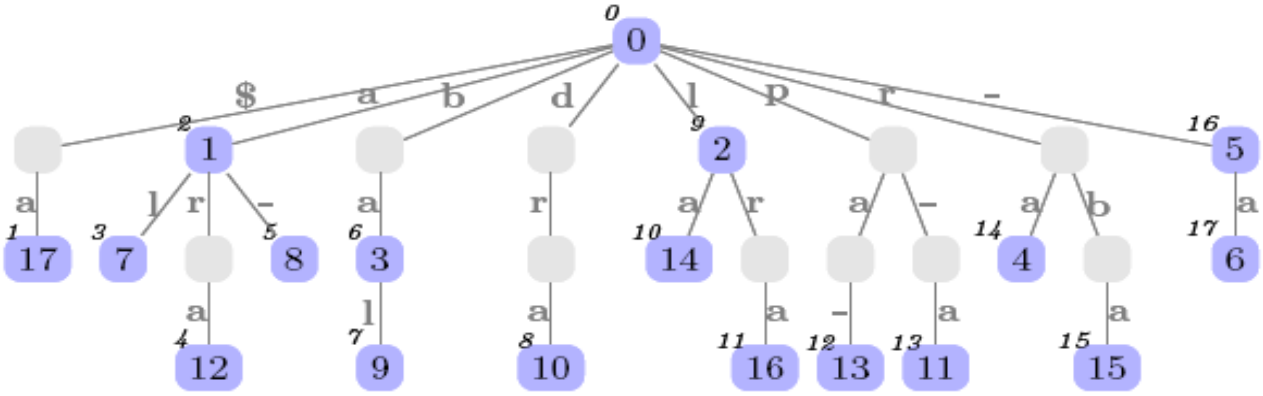
1) LZTrie: Είναι το δέντρο το οποίο περιέχει όλες τις φράσεις $B_0 \ldots B_n$ και σχηματίζεται μετά από εφαρμογή του αλγορίθμου συμπίεσης LZ78 στο κείμενο μας. Πρέπει να πούμε ότι το δέντρο αποτελείται από n+1 κόμβους οι οποίοι αναφέρονται σε ένα string ο καθένας.
2) RevTrie: Το δέντρο αυτό σχηματίζεται από όλα τα ανεστραμμένα strings $B_0^r \ldots B_n^r$. Στο δέντρο αυτό μπορεί να υπάρχουν κόμβοι οι οποίοι δεν αναπαριστούν κάποια φράση και ονομάζονται κενοί.
3) Node: Ενώνει τα αναγνωριστικά των κόμβων με τους αντίστοιχους κόμβους στο LZTrie.
4) Range: Είναι μια δομή η οποία επιτρέπει αναζήτηση δύο διαστάσεων σε ένα διάστημα [0…n]x[0…n]. Αποθηκεύουμε τους κόμβους έτσι ώστε στον οριζόντιο άξονα να βρίσκεται η preorder θέση του στο LZTrie και στον κάθετο να βρίσκεται η preorder θέση του στο RevTrie.

Βλέπουμε όλες τις δομές του LZ-Index για την φράση
“alabar_a_la_alabarda_para_apalabrarla$”

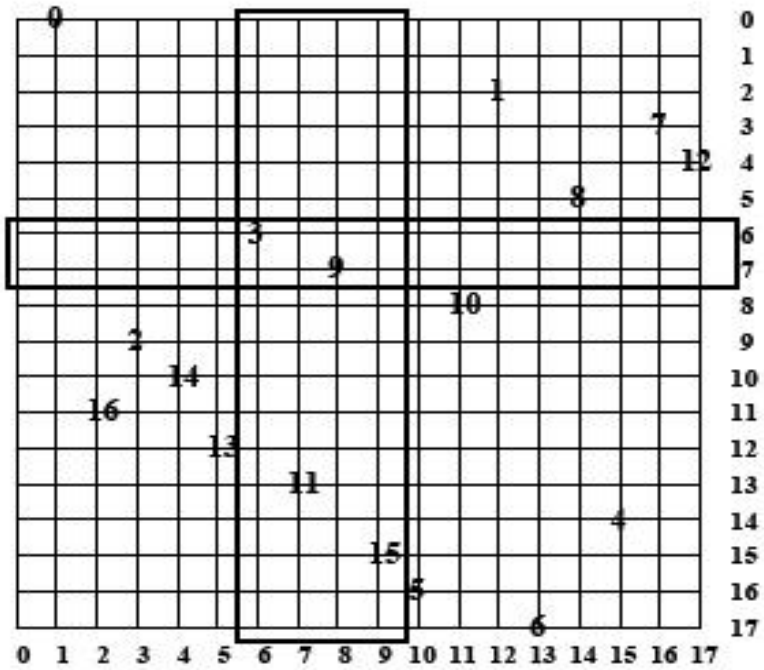
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
par:	(	(	(	)	(	(	)	)	(	)	(	(	)	(	)	(	)	)	(	(	)	)	)	(	(	(	)	)	)	(	(	(	)	)	)	)
ids:		1	17		3	15			14		4	12		10		16			6	11				2	7	9				5	8	13				
letts:		a	$		b	r			l		r	a		d		l			_	p				l	a	b				_	a	p				



Lempel-Ziv Trie (*LZTrie*) for the running example.



RevTrie data structure.



Range data structure. Horizontal coordinates are for *LZTrie* preorders, and vertical coordinates are for *RevTrie* preorders.

i :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$Node[i]$:	0	1	23	4	10	29	18	24	30	25	13	19	11	31	8	5	15	2

Node data structure, assuming that the parentheses sequence starts from position zero

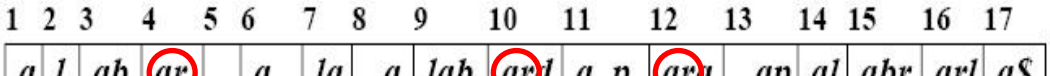
2.3 LZ-Index Search Algorithm

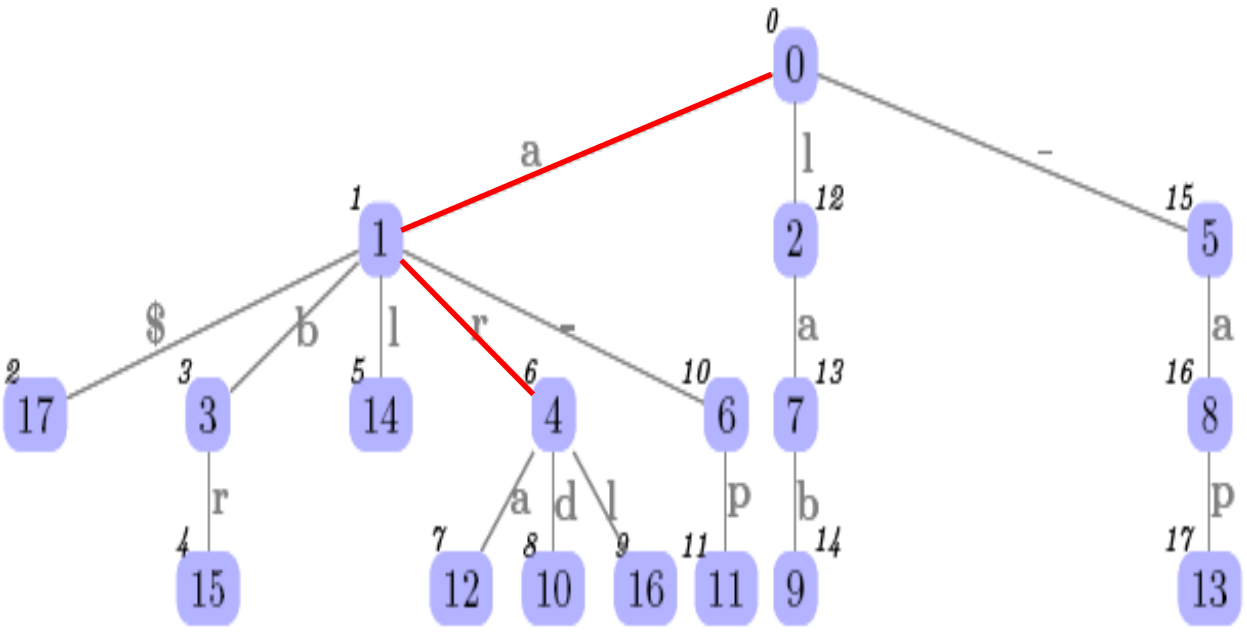
Υπάρχουν τριών διαφορτικών τύπων στιγμιότυπα για τα οποία πρέπει να κάνουμε αναζήτηση.

Στιγμιότυπα τύπου 1:

Πρόκειται για στιγμιότυπα τα οποία βρίσκονται σε ένα απλό block

Για παράδειγμα αναζητούμε το στιγμιότυπο ‘ar’



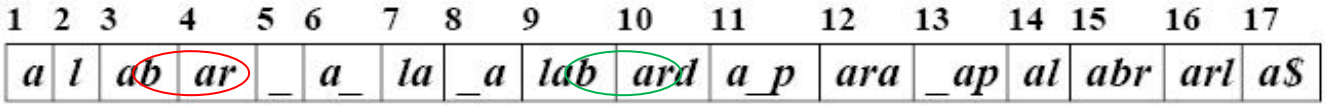


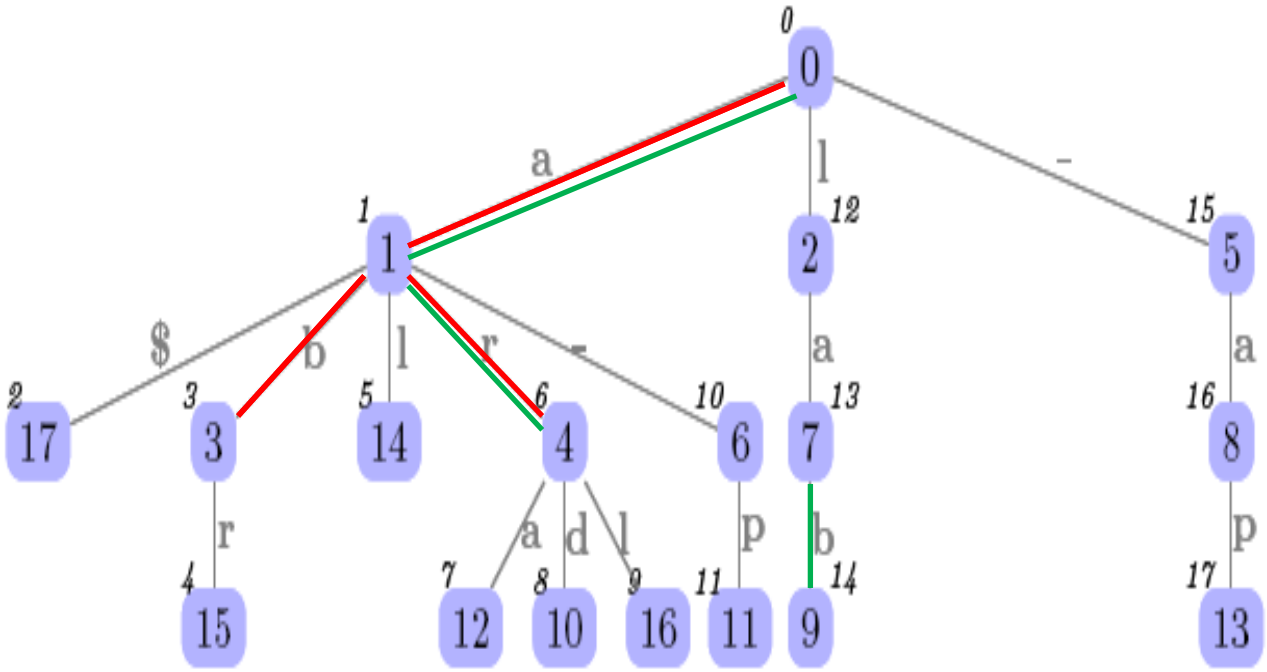
Lempel-Ziv Trie (*LZTrie*) for the running example.

Στιγμιότυπα τύπου 2:

Πρόκειται για στιγμιότυπα τα οποία βρίσκονται σε δύο block. Στην περίπτωση αυτή η φράση που αναζητούμε θα πρέπει να σπάσει σε κάθε δυνατό συνδυασμό.

Για παράδειγμα για την φράση “bar” θα πρέπει να αναζητήσουμε “b-ar” και ”ba-r”.



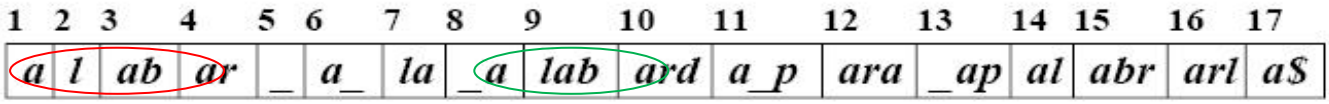


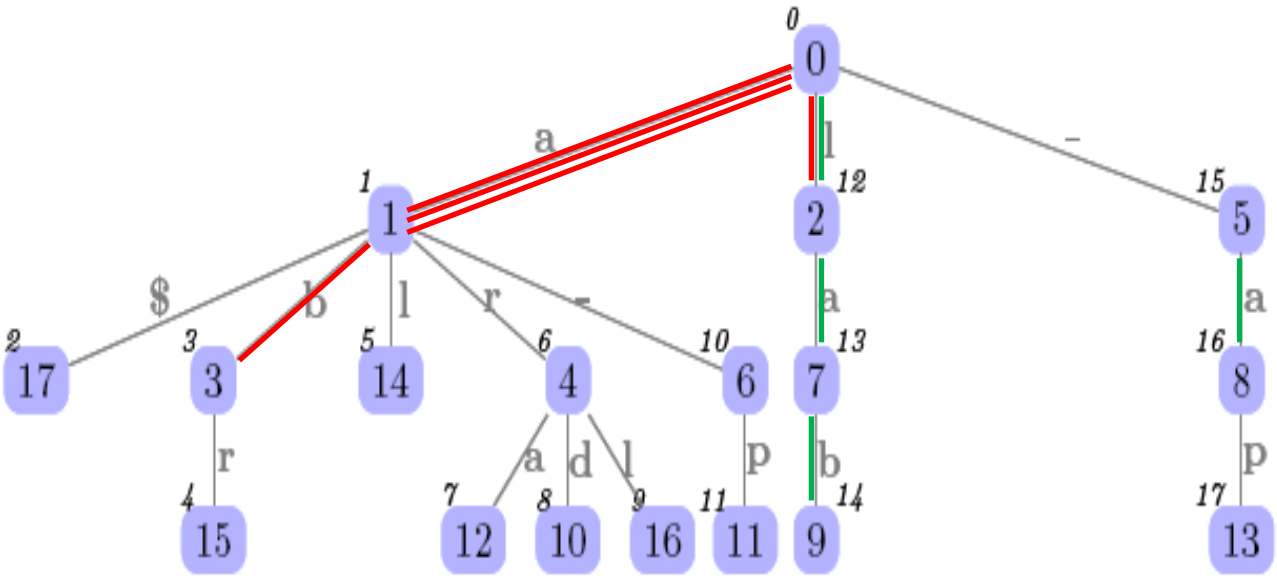
Lempel-Ziv Trie (*LZTrie*) for the running example.

Στιγμιότυπα τύπου 3:

Τα στιγμιότυπα αυτά καταλαμβάνουν τρία ή και περισσότερα block. Ξέρουμε ότι ο αλγόριθμος LZ78 μας εξασφαλίζει ότι ο κάθε κόμβος αντιπροσωπεύει ένα διαφορετικό string. Άρα υπάρχει ένα μοναδικό block το οποίο ταιριάζει στο $P_{i...j}$ για κάθε i και j. Αρχικά βρίσκουμε κάθε block το οποίο ταιριάζει σε κάθε substring $P_{i...j}$ και αποθηκεύουμε όλα τα νούμερα των blocks σε m πίνακες A_i,όπου το A_i αποθηκεύει κάθε block που αντιστοιχεί στο $P_{i...j}$ για όλα τα j. Έπειτα, προσπαθούμε να βρούμε μια αλληλουχία επιτυχών block $B_κ$,$B_{κ+1}$…. τα οποία ταιριάζουν συνεχόμενα με τα substring της φράσης που αναζητούμε. Υπάρχει πάντα μόνο ένας υποψήφιος που μπορεί να ακολουθεί το B_k στην φράση και αυτός είναι ο B_{k+1}. Τέλος, για κάθε μέγιστη αλληλουχία blocks $P_{i...j}=B_k...B_l$ που περιέχονται στη φράση μας, εξετάζουμε αν το B_{k-1} τελειώνει με $P_{1...i-1}$ και το B_{l+1} αρχίζει με $P_{j+1...m}$. Αν συμβαίνει κάτι τέτοιο τότε μπορούμε να αναφερουμε ένα στιγμιότυπο της φράσης που ψάχνουμε.

Για παράδειγμα η φράση ‘alaba’





Lempel-Ziv Trie (*LZTrie*) for the running example.

Κεφάλαιο 3
XBW Transformation

3.1 Ο μετασχηματισμός XBW

Ο XBW είναι ένας μετασχηματισμός ο οποίος εφαρμόζεται σε labeled trees. Χρησιμοποιεί path sorting για να γραμμικοποιήσει το δέντρο σε δύο πίνακες, από τους οποίους ο ένας περιέχει τη δομή του δέντρου και ο άλλος τις ετικέτες.

Ο μετασχηματισμός αυτός μας δίνει τη δυνατότητα να έχουμε τις παρακάτω λειτουργίες:

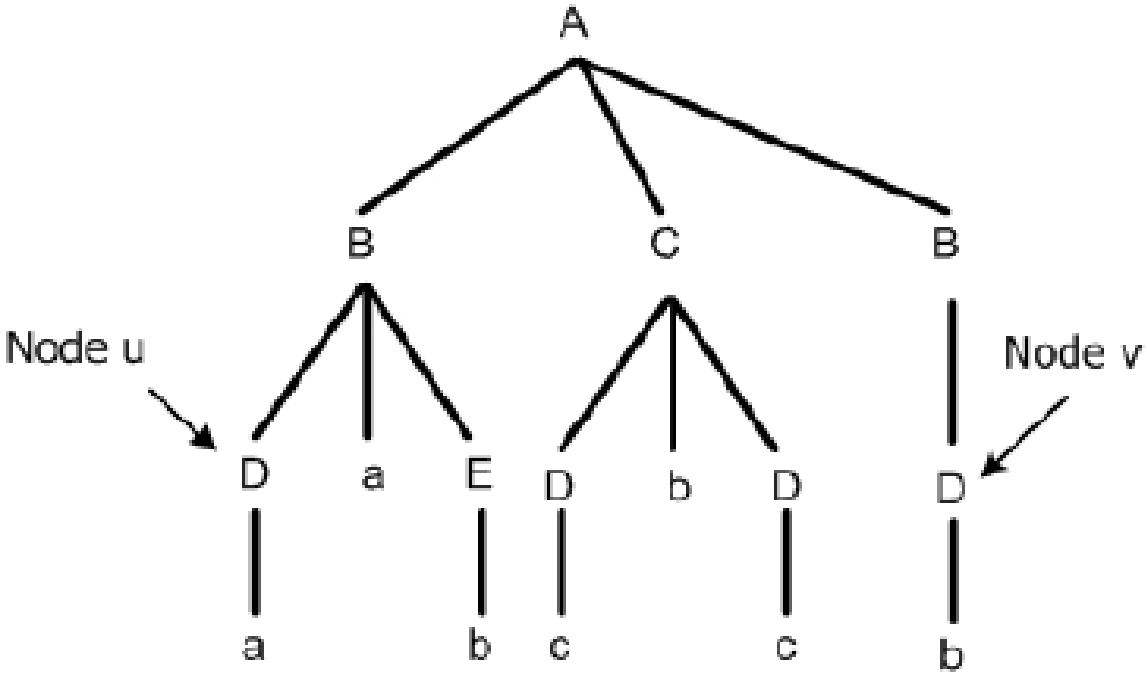
1) **Navigational Queries:** Ζητάμε τον πατέρα ενός κόμβου, το *i*th παιδί ενός κόμβου ή τον βαθμό ενός κόμβου.
2) **Visualization Queries:** Μπορούμε να ανακτήσουμε τους κόμβους που αποτελούν το υποδένδρο ενός κόμβου u.
3) **Subpath Queries:** Ζητάμε τους κόμβους (αριθμός στιγμιοτύπων) στους οποίους καταλήγει ένα μοναπάτι Π, το οποίο μπορεί να βρίσκεται οπουδήποτε στο δέντρο.

Ο μετασχηματισμός XBW μας επιτρέπει να μετατρέψουμε προβλήματα συμπίεσης και αναζήτησης σε απλά προβλήματα πράξεων με string.

Για την αναζήτηση αρκεί να χρησιμοποιήσουμε τις πράξεις rank και select που εφαρμόζονται σε strings. Για ένα string S[1,t] η $rank_c(S,q)$ επιστρέφει τον αριθμό των φορών που το σύμβολο c υπάρχει στο πρόθεμα S[1,q] και η $select_c(S,q)$ μας δίνει τη θέση του q-th στιγμιοτύπου του χαρακτήρα c.

3.2 Ο μετασχηματισμός XBW στην πράξη

Παρακάτω έχουμε ένα labeled tree πάνω στο οποίο θα εφαρμόσουμε τον μετασχηματισμό.

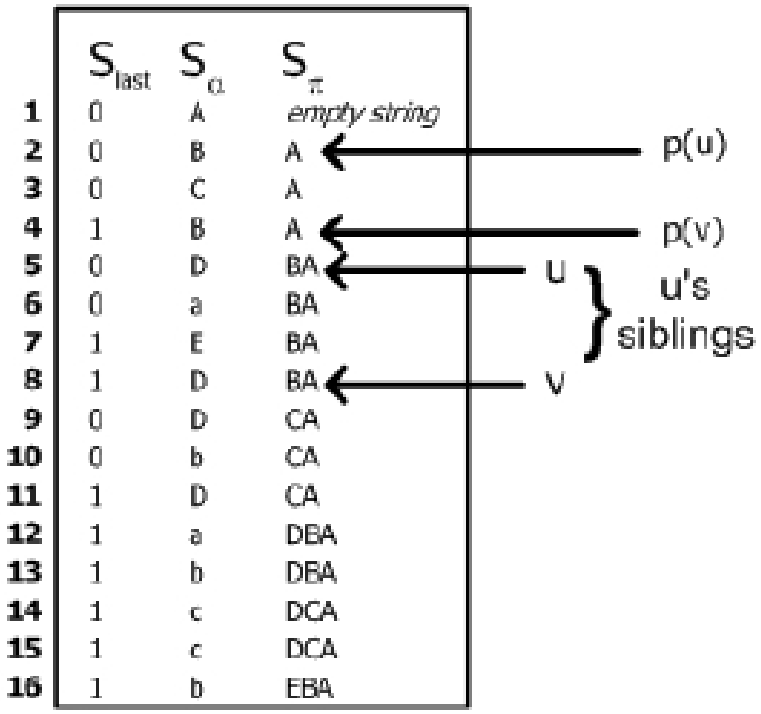


Στην πρώτη φάση του μετασχηματισμού διατρέχουμε το δέντρο σε preorder και παίρνουμε τους παρακάτω πίνακες.

S_{last}	S_{α}	S_{π}
0	A	*empty string*
0	B	A
0	D	BA
1	a	DBA
0	a	BA
1	E	BA
1	b	EBA
0	C	A
0	D	CA
1	c	DCA
0	b	CA
1	D	CA
1	c	DCA
1	B	A
1	D	BA
1	b	DBA

Ο πίνακας S_{last} περιέχει αναγνωριστικά για το αν ένα παιδί είναι το τελευταίο. Στην περίπτωση που είναι το τελευταίο παίρνει τιμή 1 αλλιώς 0. Ο πίνακας S_a περιέχει τα label όλων των κόμβων του δέντρου. Ο πίνακας S_π περιέχει όλα τα μονοπάτια από κάθε κόμβο μέχρι τη ρίζα (upward paths).

Τέλος, ταξινομούμε (stable sorting) τον πίνακα S_π και μαζί του φυσικά και τους άλλους δύο. Το αποτέλεσμα το βλέπουμε παρακάτω.



Αφού τελειώσουμε με την ταξινόμηση μπορούμε να διαγράψουμε τον πίνακα S_π με τα μονοπάτια και να κρατήσουμε τον S_a και τον S_{last}.

S_{last}	S_a
0	A
0	B
0	C
1	B
0	D
0	a
1	E
1	D
0	D
0	b
1	D
1	a
1	b
1	c
1	c
1	b

3.3 Ιδιότητες του μετασχηματισμού XBW

Στο σημείο αυτό χρειάζεται να αναφέρουμε κάποιες ιδιότητες του μετασχηματισμού.

Ιδιότητα 1η: Δομή του S

1. Ο πίνακας S_{last} έχει n bits ίσα με 1 (ένα για κάθε κόμβο ο οποίος είναι τελευταίο παιδί) και l=t-n bits ίσα με 0.
2. Ο πίνακας S_a είναι μια διάταξη των συμβόλων που αποτελούν τις ετικέτες των κόμβων του δέντρου T.
3. Ο πίνακας S_π περιλαμβάνει όλα τα μονοπάτια από τους κόμβους προς τη ρίζα και περιλαμβάνει μόνο εσωτερικούς κόμβους (όχι φύλλα).

Ιδιότητα 2η: Δομική σχέση ανάμεσα στο δέντρο και τους πίνακες του μετασχηματισμού

1) Η πρώτη τριπλέτα των πινάκων αναφέρεται στη ρίζα
2) Η τριπλέτα του κόμβου u προηγείται της τριπλέτας του κόμβου v εαν και μόνο εαν $\pi[u]<\pi[v]$ λεξικογραφηκά, ή $\pi[u]=\pi[v]$ και ο u προηγείται του v κατά την προδιάταξη.
3) Υποθέτουμε ότι τα $u_1,\ldots,u_z$ είναι παιδιά του κόμβου u. Οι τριπλέτες $s[u_1],\ldots.s[u_z]$ βρίσκονται σε συνεχόμενες θέσεις στο S με την ίδια σειρά. Επιπλέον, ο υποπίνακας $S_{last}[u_1\ldots u_z]$ μας δίνει τον αριθμό παιδιών του u, αφού $S_{last}[u_z]=1$ και $S_{last}[u_i]=0$ για όλα τα $1<=i<z$.Επομένως το μέγεθος του υποπίνακα είναι ίσο με τον αριθμό των παιδιών του γονέα u.
4) Υποθέτουμε ότι δύο κόμβοι u,z έχουν τον ίδιο μονοπάτι, τότε αν ο u προηγείται του z θα συνεχίσει να προηγείται του z και μετά την ταξινόμηση.

Ιδιότητα 3η: Υποθέτουμε ότι το c είναι η ετικέτα ενός εσωτερικού κόμβου και $S[j_1,j_2]$ είναι όλες οι τριπλέτες των οποίων τα μονοπάτια ξεκινούν με το σύμβολο c. Αν ο u είναι ο i-th κόμβος του οποίου ετικέτα είναι το σύμβολο c , τα παιδιά του βρίσκονται συνεχόμενα μέσα στο $S[j_1,j_2]$.

Εδώ φαίνεται ο αλγόριθμος ο οποίος μετατράπει ένα δέντρο σε αναπαράσταση XBW.

Algorithm PathSort($\mathcal{T}$)

1. Create the array IntNodes$[1, n]$ initially empty.
2. Visit the internal nodes of $\mathcal{T}$ in preorder. Let u denote the ith visited node. Write in IntNodes$[i]$ the symbol $\alpha[u]$, the level of u in $\mathcal{T}$, and the position in IntNodes of u's parent.
3. Let $j \in \{0, 1, 2\}$ be such that the number of nodes in IntNodes whose level is $\equiv j \pmod 3$ is at least $n/3$. Sort recursively the upwards paths starting at nodes at levels $\not\equiv j \pmod 3$.
4. Sort the upward paths starting at nodes at levels $\equiv j \bmod 3$ using the result of Step 3.
5. Merge the two sets of sorted paths.

3.4 Subpath Search in xbw(T)

Το χαρακτηριστικό του μετασχηματισμού xbw που μας ενδιαφέρει περισσότερο είναι η εύρεση μονοπατιών από κάθε κόμβο στη ρίζα. Επειδή εμάς μας ενδιαφέρει τις περισσότερες φορές το αντίθετο, δηλαδή η εύρεση ενός μονοπατιού με αρχή έναν κόμβο και φορά προς τα κάτω, μπορούμε το πρόβλημα αυτό να το λύσουμε με έναν πολύ απλό τρόπο. Βάζοντας στην ερώτηση το αντίστροφο string από αυτό που θέλουμε να βρούμε. Δηλαδή, αν ψάχνουμε για το “abcd” θα πρέπει να κάνουμε την ερώτηση για το μονοπάτι “dcba” και αυτό θα μας δώσει το αποτέλεσμα που ψάχνουμε. Ο αλγόριθμος μπορεί να βρει τον αριθμό των υπομονοπατιών που αντιστοιχούν στο string της ερώτησης. Από τις ιδιότητες του μετασχηματισμού είναι εύκολο να δούμε ότι όλα τα ζητούμενα μονοπάτια θα καταλαμβάνουν συνεχόμενες θέσεις στον πίνακα S.

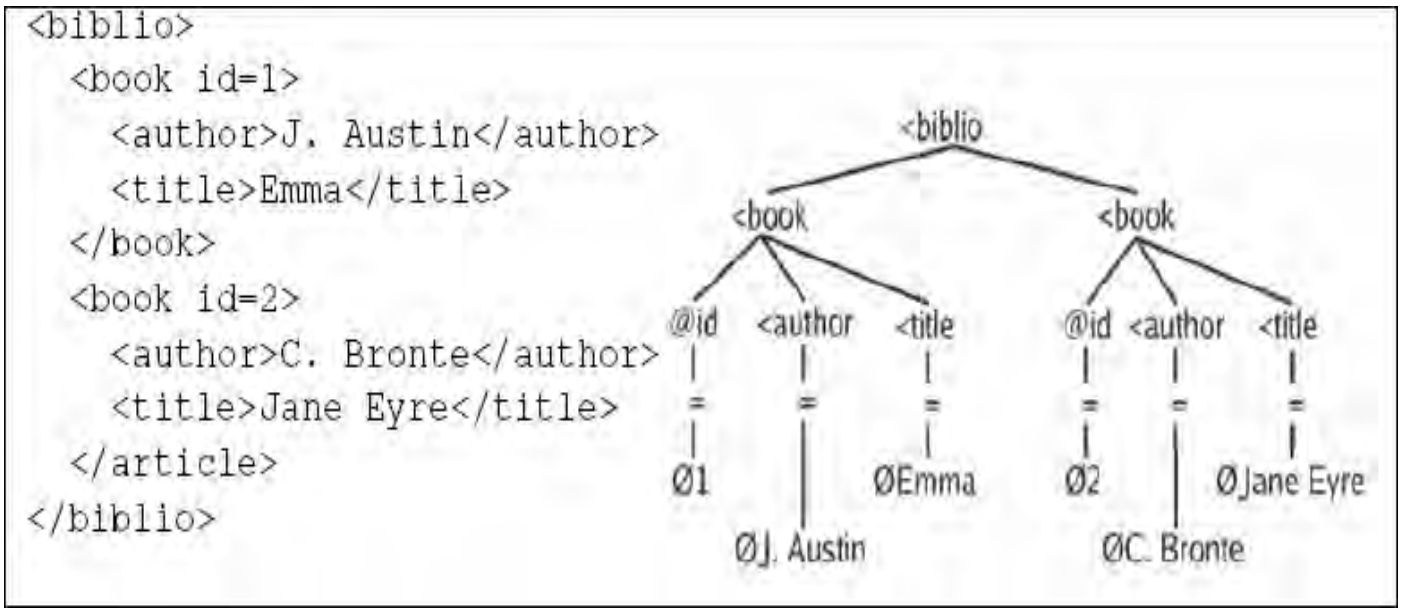
Algorithm SubPathSearch($q_1 q_2 \cdots q_k$)

1. $i = k$;
2. First $= \mathsf{F}(q_k)$; Last $= \mathsf{F}(q_k + 1) - 1$;
3. **while** (First $\leq$ Last) **and** ($i \geq 2$) **do**
4. $\quad i = i - 1$;
5. $\quad z = \mathsf{rank}_1(\mathcal{S}_{\mathsf{last}}, \mathsf{F}(q_i) - 1)$; { *z is # of nodes with label smaller than* q_i }
6. $\quad k_1 = \mathsf{rank}_{q_i}(\mathcal{S}_\alpha, \mathsf{First} - 1)$; { *# occurrences of* q_i *in* $\mathcal{S}_\alpha[1, \mathsf{First} - 1]$ }
7. $\quad \mathsf{First} = \mathsf{select}_1(\mathcal{S}_{\mathsf{last}}, z + k_1 - 1) + 1$; {*starting position of the* $(z + k_1)$*-th group of siblings* }
8. $\quad k_2 = \mathsf{rank}_{q_i}(\mathcal{S}_\alpha, \mathsf{Last})$; { *# of occurrences of* q_i *in* $\mathcal{S}_\alpha[1, \mathsf{Last}]$ }
9. $\quad \mathsf{Last} = \mathsf{select}_1(\mathcal{S}_{\mathsf{last}}, z + k_2)$; { *ending position of the* $(z + k_2)$*-th group of siblings* }
10. **if** (First $>$ Last) **then**
11. $\quad$ **return** “no path prefixed by $q_1 q_2 \cdots q_k$”;
12. **else return** (First, Last).

Compute $\mathcal{S}[a, b]$ as the set of triplets representing the parents of nodes with paths prefixed by $q_1 \ldots q_k$.

3.5 Ο XBW στην πράξη

Παρακάτω βλέπουμε ένα XML document και το αντίστοιχο δέντρο



Ακολούθως έχουμε τους πίνακες του μετασχηματισμού XBW

S_{last}	S_α	S_π
1	<biblio	*empty string*
0	<book	<biblio
0	@id	<book<biblio
1	=	@id<book<biblio
1	**0**1	=@id<book<biblio
0	<author	<book<biblio
1	=	<author<book<biblio
1	**0**J. Austin	=<author<book<biblio
1	<title	<book<biblio
1	=	<title<book<biblio
1	**0**Emma	=<title<book<biblio
1	<book	<biblio
0	@id	<book<biblio
1	=	@id<book<biblio
1	**0**2	=@id<book<biblio
0	<author	<book<biblio
1	=	<author<book<biblio
1	**0**C. Bronte	=<author<book<biblio
1	<title	<book<biblio
1	=	<title<book<biblio
1	**0**Jane Eyre	=<title<book<biblio

Μετά το Stable Sorting οι πίνακες γίνονται

Rk	$\mathcal{S}_{\text{last}}$	$\mathcal{S}_{\alpha}$	$\mathcal{S}_{\pi}$
1	1	<biblio	*empty string*
2	1	=	<author<book<biblio
3	1	=	<author<book<biblio
4	0	<book	<biblio
5	1	<book	<biblio
6	0	@id	<book<biblio
7	0	<author	<book<biblio
8	1	<title	<book<biblio
9	0	@id	<book<biblio
10	0	<author	<book<biblio
11	1	<title	<book<biblio
12	1	=	<title<book<biblio
13	1	=	<title<book<biblio
14	1	=	@id<book<biblio
15	1	=	@id<book<biblio
16	1	**0**J. Austin	=<author<book<biblio
17	1	**0**C. Bronte	=<author<book<biblio
18	1	**0**Emma	=<title<book<biblio
19	1	**0**Jane Eyre	=<title<book<biblio
20	1	**0**1	=@id<book<biblio
21	1	**0**2	=@id<book<biblio

Τελικά έχουμε τους παρακάτω δύο πίνακες (Σημ. μόνο ο S_{last} και ο S_a ανήκουν στον μετασχηματισμό)

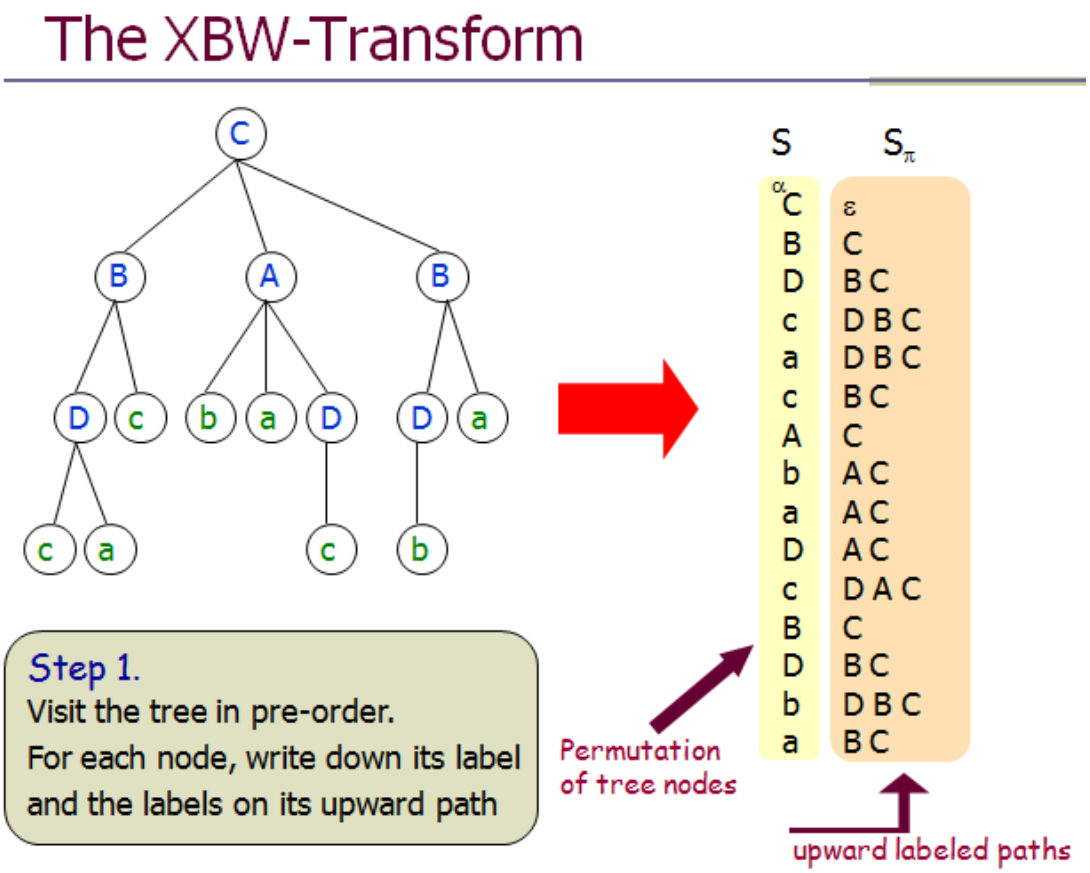
$$\widehat{\mathcal{S}}_{\text{last}} = 111010010011111$$

$$\widehat{\mathcal{S}}_{\alpha} = \text{<biblio==<book<book@id<author<title@id<author<title====}$$

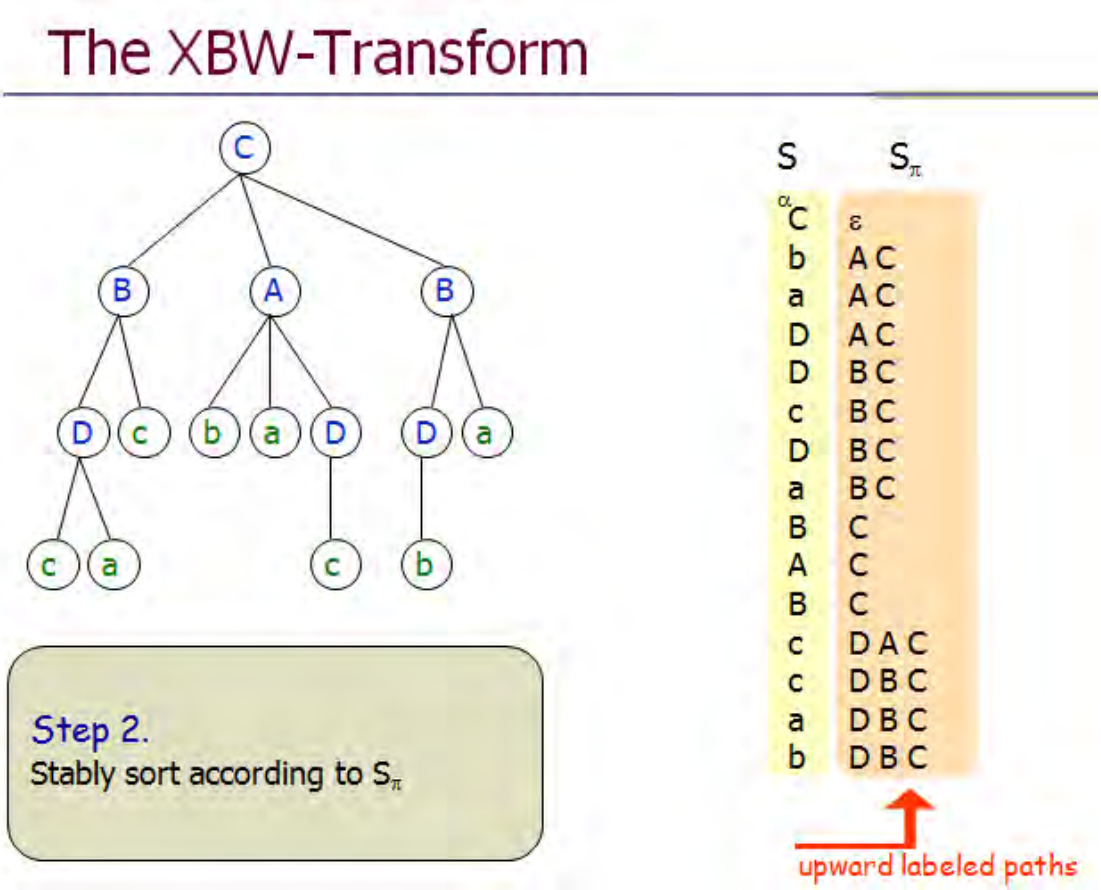
$$\widehat{\mathcal{S}}_{\text{pcdata}} = \textbf{0}\text{J. Austin}\textbf{0}\text{C. Bronte}\textbf{0}\text{Emma}\textbf{0}\text{Jane Eyre}\textbf{0}\textbf{1}\textbf{0}\textbf{2}$$

Παρακάτω θα δούμε βήμα βήμα την δημιουργία των πινάκων του μετασχηματισμού, τις ιδιότητες και τέλος τη διαδικασία της αναζήτησης

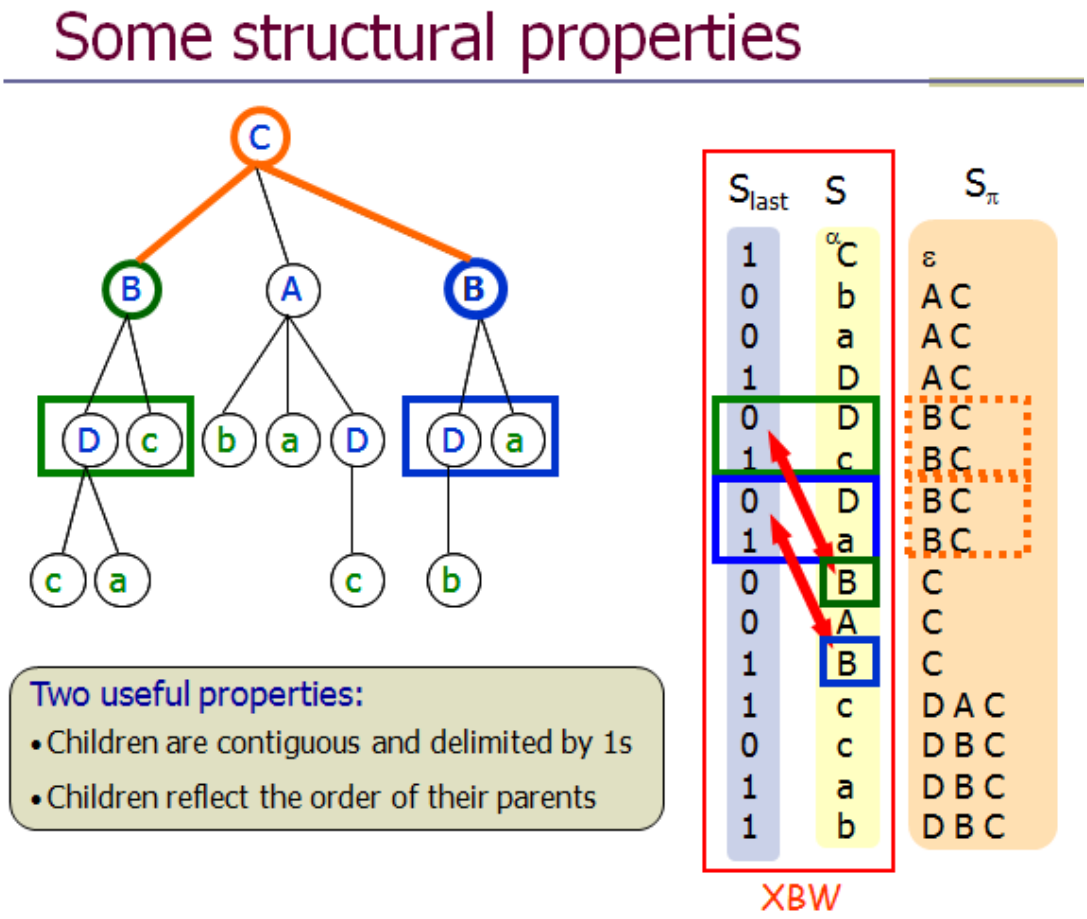
1) Ο μετασχηματισμός XBW



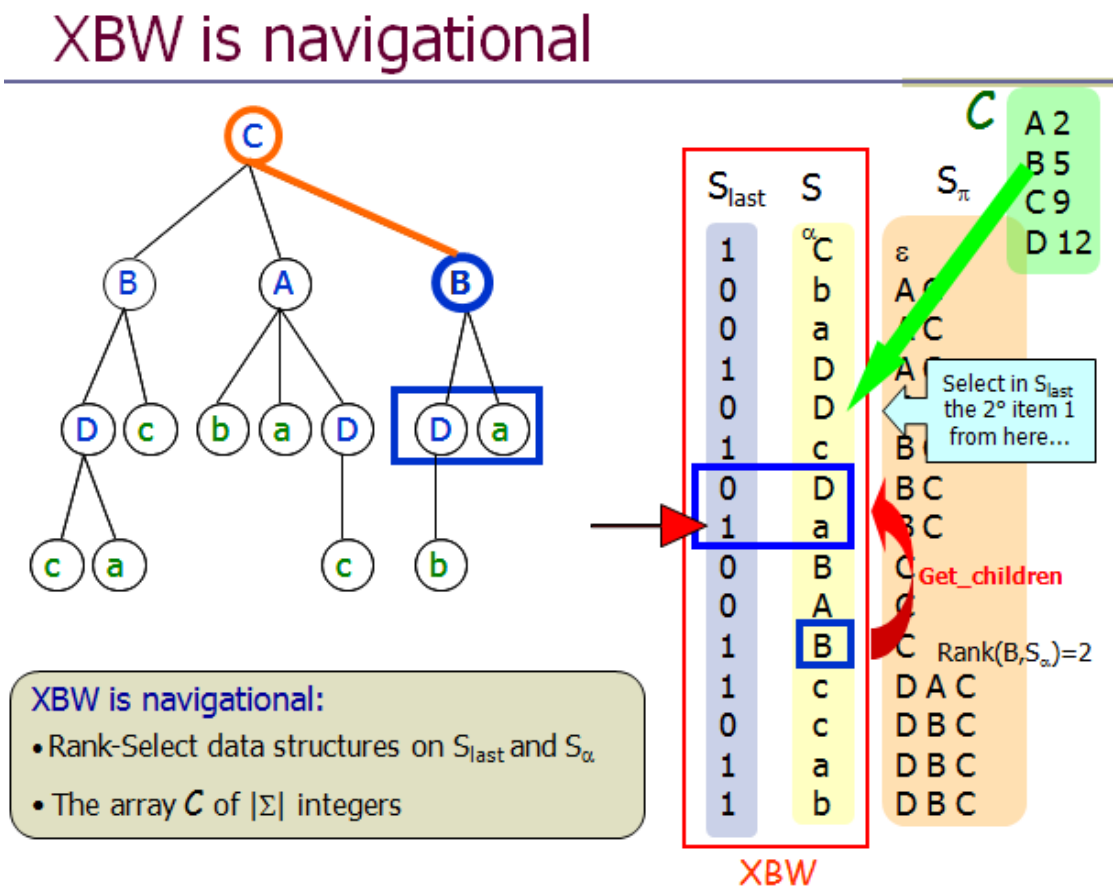
2) Ο μετασχηματισμός XBW μετά το Stable Sorting



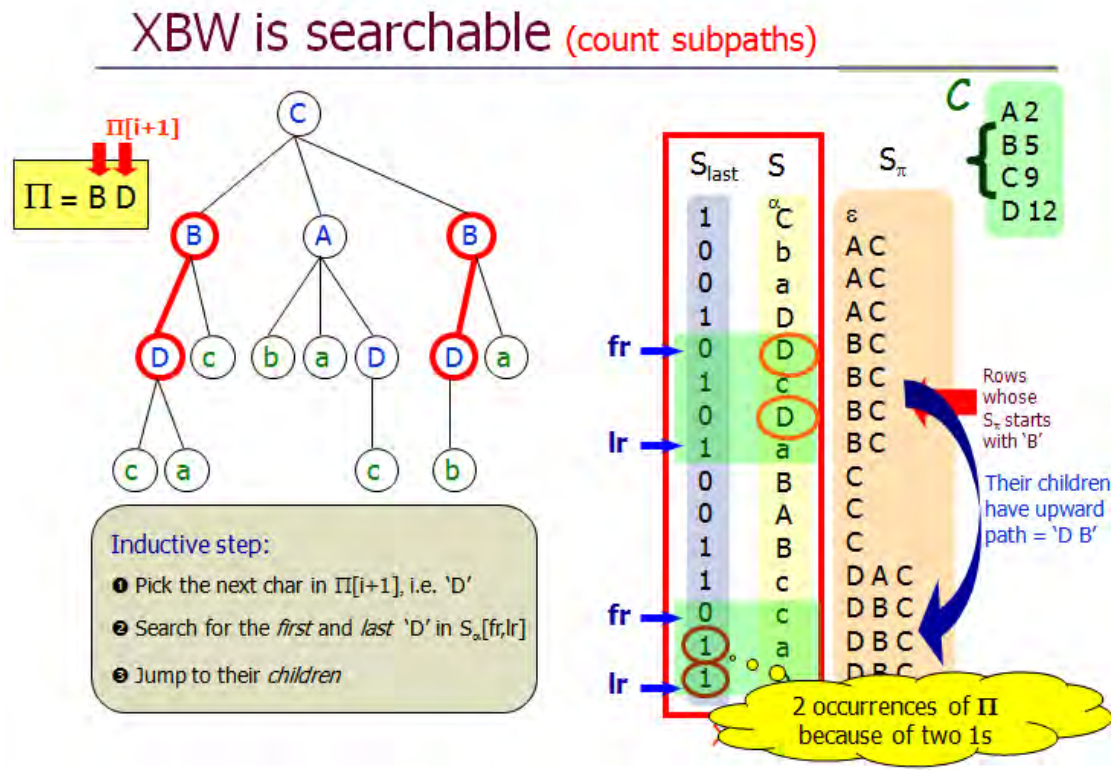
3) Κάποιες από τις δομικές ιδιότητες του μετασχηματισμού



4) Η πλοήγηση στο δέντρο μέσω των πινάκων



5) Η αναζήτηση



Κεφάλαιο 4
Χρήση του μετασχηματισμού XBW στον LZ-Index

Όπως ξέρουμε ο LZ-Index χρησιμοποιεί δύο δομές δέντρων το LZTrie και το ανεστραμμένο RevTrie. Η ιδέα της παρούσας έρευνας είναι να χρησιμοποιήσουμε έναν καλύτερο τρόπο για να αναπαραστήσουμε αυτά τα δύο δέντρα. Χρησιμοποιώντας τον μετασχηματισμό XBW μπορούμε να μειώσουμε και τις απαιτήσεις σε χώρο αλλά και τις απαιτήσεις σε χρόνο για την διαπέραση των δέντρων και κυρίως της εύρεσης μονοπατιών που είναι το δύνατό σημείο του μετασχηματισμού XBW και το πλέον αναγκαίο στοιχείο για τον LZ-Index. Όπως θα δούμε και παρακάτω όταν αναφερθούμε στην αναζήτηση το RevTrie δεν το χρειαζόμαστε. Άρα καταλήγουμε στο σημείο, ότι αντικαθιστούμε τις δύο δομές δέντρων με τους δύο πίνακες S_a και S_{last} του μετασχηματισμού XBW πετυχαίνοντας και οικονομία χώρου αλλά και καλύτερο χρόνο αναζήτησης.

4.1 Η δομή του LZ-Index με την χρήση του μετασχηματισμού XBW

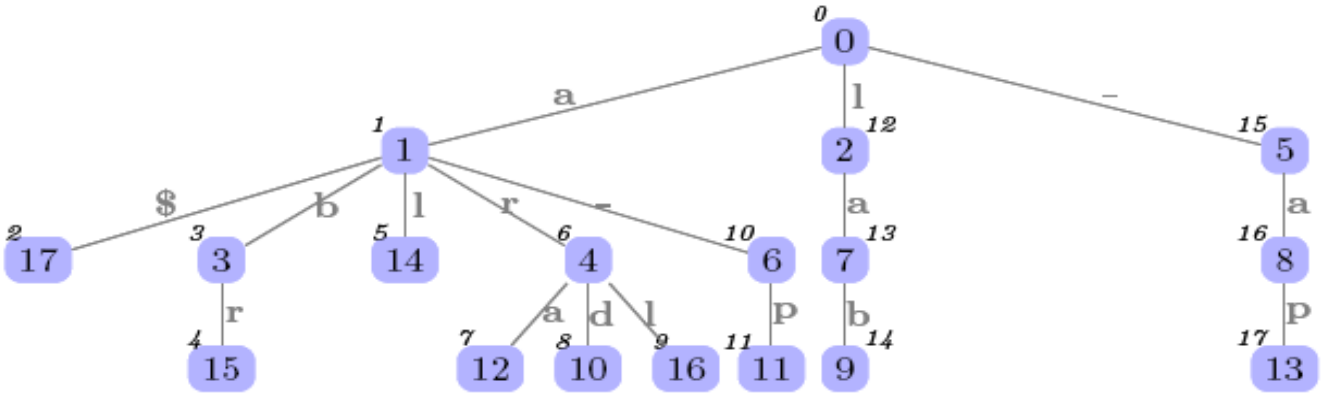
Πριν μιλήσουμε για τις δομές πρέπει να πούμε ότι είναι απαραίτητη η πρόσθεση dummy children στα φύλλα του δέντρου LZTrie έτσι ώστε να εφαρμόσουμε τον μετασχηματισμό XBW χωρίς να χαθεί πληροφορία.

- Στη θέση των δέντρων LZTrie και RevTrie έχουμε πλέον δύο πίνακες που προέρχονται από τον μετασχηματισμό XBW, τον S_a και τον S_{last}.
- *Par:* Έχουμε την δομή balanced parentheses από την οποία μπορούμε να πάρουμε την μορφή του αρχικού δέντρου. Στη δομή αυτή ανοίγουμε μια παρένθεση κατά την πρώτη μας επίσκεψη σε έναν κόμβο καθώς διατρέχουμε το δέντρο με την τεχνική της προδιάταξης. Ακολούθως στην τελική μας επίσκεψη σε έναν κόμβο κλείνουμε την παρένθεση.
- *Ids:* Η δομή αυτή περιέχει το αναγνωριστικό κάθε κόμβου (μόνο για non-dummy) και πάλι σε σειρά προδιατάξεως.
- *Pos:* Μας δίνει για κάθε κόμβο του μετασχηματισμού XBW την αρίθμηση για τον αντίστοιχο κόμβο σε σειρά προδιατάξεως.
- *Pos^{-1}:* Κάνει το αντίστροφο από το Pos, δηλαδή για κάθε κόμβο στη σειρά προδιατάξεως μας δίνει τον αντίστοιχό του από τον μετασχηματισμό XBW.

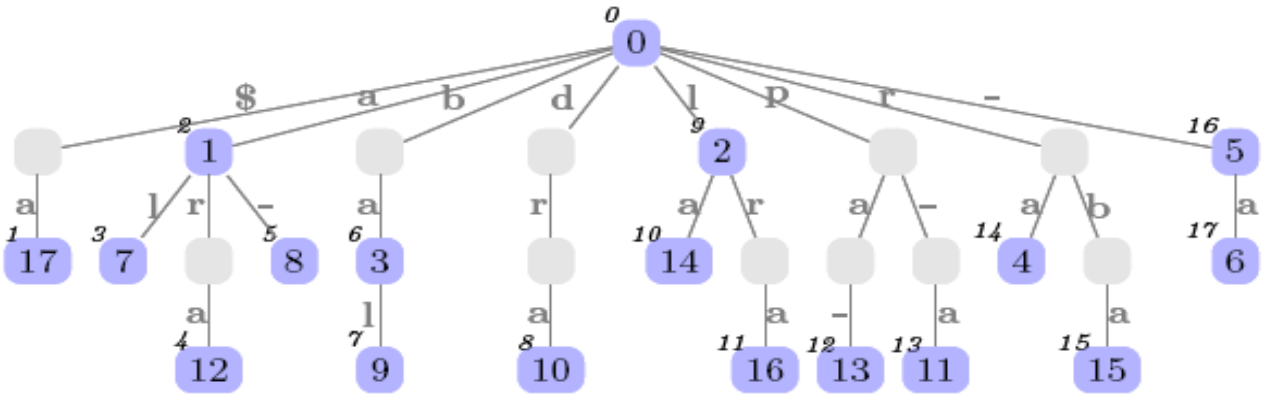
- *Range:* Είναι μια δομή ψαξίματος. Σε κάθε σημείο αποθηκεύει ένα αναγνωριστικό φράσης. Το κάθε σημείο έχι συντεταγμένες (x,y) από τις οποίες η x αναφέρεται στη θέση του κόμβου στον μετασχηματισμό xbw και η y αναφέρεται στη θέση του κόμβου στη σειρά προδιατάξεως.

Παρακάτω βλέπουμε τις δομές που αντιστοιχούν στον LZ-Index της φράσης “alabar_a_la_alabarda_para_apalabrarla$” πριν εφαρμοστεί ο μετασχηματισμός xbw.

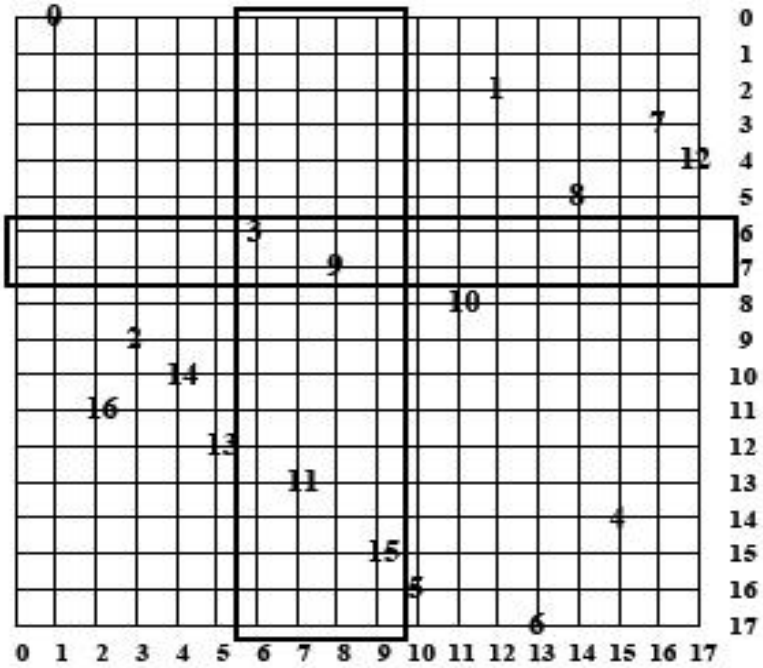
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
par:	(	(	(	)	(	(	)	)	(	)	(	(	)	(	)	(	)	)	(	(	)	)	)	(	(	(	)	)	)	(	(	(	)	)	)	)
ids:		1	17		3	15			14		4	12		10		16			6	11				2	7	9				5	8	13				
letts:		a	$		b	r			l		r	a		d		l			_	p				l	a	b				_	a	p				



Lempel-Ziv Trie (*LZTrie*) for the running example.



RevTrie data structure.



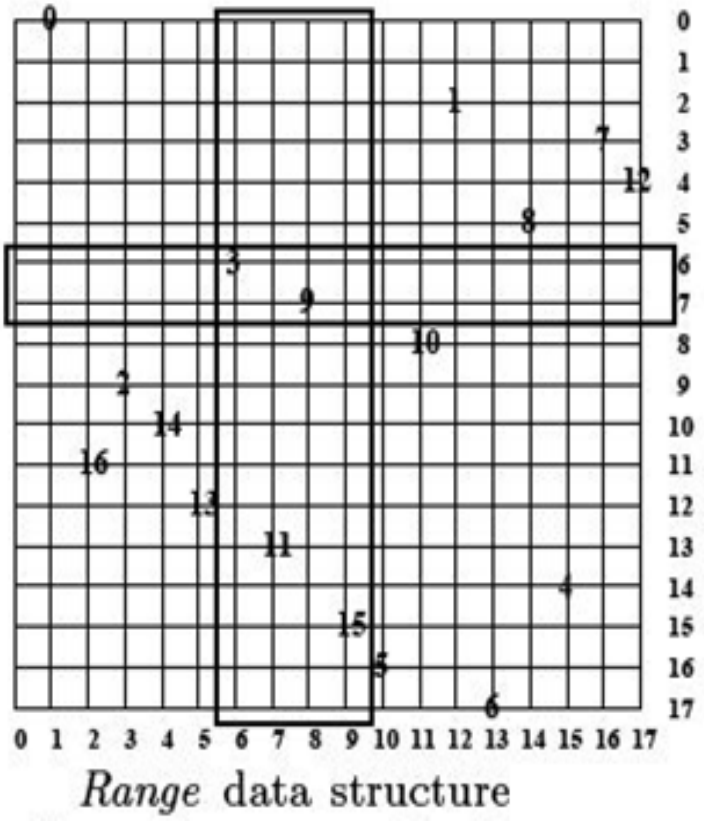
Range data structure. Horizontal coordinates are for *LZTrie* preorders, and vertical coordinates are for *RevTrie* preorders.

i :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$Node[i]$:	0	1	23	4	10	29	18	24	30	25	13	19	11	31	8	5	15	2

Node data structure, assuming that the parentheses sequence starts from position zero

Παρακάτω θα δούμε τις δομές του LZ-Index για την παραπάνω φράση αφού πρώτα έχει εφαρμοστεί ο μετασχηματισμός xbw.

i	S_{last}	S_α
1	0	a
2	0	l
3	1	_
4	1	Δ
5	0	$
6	0	b
7	0	l
8	0	r
9	1	_
10	1	b
11	1	Δ
12	1	p
13	1	r
14	1	Δ
15	1	Δ
16	1	a
17	1	Δ
18	1	Δ
19	1	Δ
20	1	Δ
21	0	a
22	0	d
23	1	l
24	1	Δ
25	1	a
26	1	p



Range data structure

	0			5			10			15			20			25			30			35			40			45			50	53
par:	(	(	(	(	)	)	(	(	(	)	)	)	(	(	(	)	)	(	(	)	)	(	(	(	)	)	)	)	(	(	(	(	)	)	)	)
B:	1	1	1	0			1	1	0				1	0				1	0				1	1	0				1	1	1	0			
ids:	0	1	17				3	15					14					4	12				10						16						
letts:	a	$	Δ				b	r	Δ				l	Δ				r	a	Δ			d	Δ					l	Δ					
Pos^{-1}:	1	5	4				6	13	24				7	17				8	21	11			22	15					23	18					

i :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
$Pos[i]$:	1	19	23	3	2	4	7	9	16	21	11	25	5	22	13	20	8	15	26	18	10	12	14	6	24	17

4.2 Αλγόριθμος Αναζήτησης

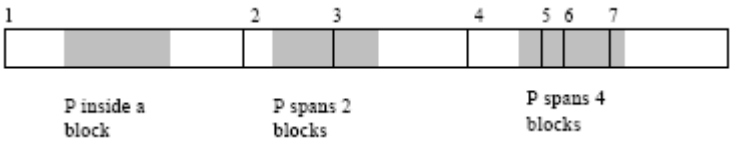
Για να βρούμε μια φάση P μήκους m χωρίζουμε την αναζήτηση σε τρία στάδια.

1) Στιγμιότυπα τύπου 1: Ψάχνουμε να βρούμε όλες τις φράσεις που έχουν την P σαν επίθεμα. Για να γίνει αυτό κάνουμε αναζήτηση της αντίστροφης φράσης P^r στη δομή xbw.Υποθέτουμε ότι παίρνουμε τις θέσεις $[x_1 \ldots x_2]$ οι οποίες αναφέρονται σε όλους τους κόμβους που έχουν σαν επίθεμα τη φράση P. Με άλλα λόγια, το διάνυσμα $[x_1 \ldots x_2]$ περιέχει τις ρίζες των υποδέντρων τα οποία περιέχουν τους κόμβους που ψάχνουμε.

2) Στιγμιότυπα τύπου 2: Για να βρούμε αυτού του είδους τις φράσεις, για κάθε δυνατή διαμέριση P[1…i] και P[i+1…m] της φράσης P, διασχίζουμε το δέντρο σε μορφή xbw από την ρίζα χρησιμοποιώντας τα σύμβολα P[i+1…m]. Αφού βρούμε τις ζητούμενες θέσεις στο xbw tree τις βρίσκουμε και στο preorder και έτσι βρίσκουμε όλους τους κόμβους οι οποίοι έχουν πρόθεμα P[i+1…m]. Ακολούθως κάνουμε αναζήτηση στο xbw tree και βρίσκουμε όλους τους κόμβους που τελειώνουν με P[1…i]. Τέλος, ψάχνουμε στη δομή Range για κόμβους των οποίων το αναγνωριστικό φράσης B_t τελειώνει με P[1…i] και για κόμβους B_{t+1} των οποίων αρχίζει με P[i+1…m].

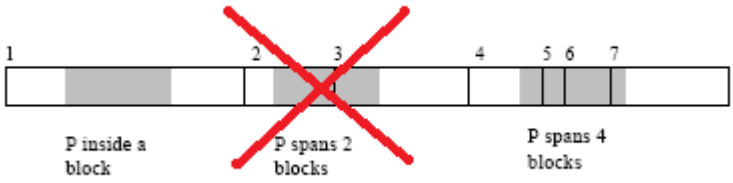
3) Στιγμιότυπα τύπου 3: Για να βρούμε αυτού του είδους φράσεις θα πρέπει για κάθε διαμέριση της φράσης $P[i \ldots j]=B_t \ldots B_l$ να βρούμε τις φράσεις B_{l+1} οι οποίες ξεκινάνε με P[j+1…m] και τις φράσεις B_{t-1} οι οποίες τελειώνουν με P[1…i-1]. Επειδή έχουμε βρει όλα τα substrings του P και ξέρουμε την σειρά όλων των απογόνων του P[j+1…m] ελέγχουμε αν ο κόμβος σε προδιάταξη στη θέση $ids^{-1}(l+1)$ ανήκει σε αυτό το διάνυσμα. Στον δεύτερο έλεγχο βρίσκουμε ποιοι κόμβοι B_{t-1} τελειώνουν σε P[1…i-1]. Εδώ χρειαζόμαστε το Pos^{-1} και η θέση είναι $Pos^{-1}(ids^{-1}(t-1))$.

Σχηματικά οι τρεις τύποι στιγμιοτύπων



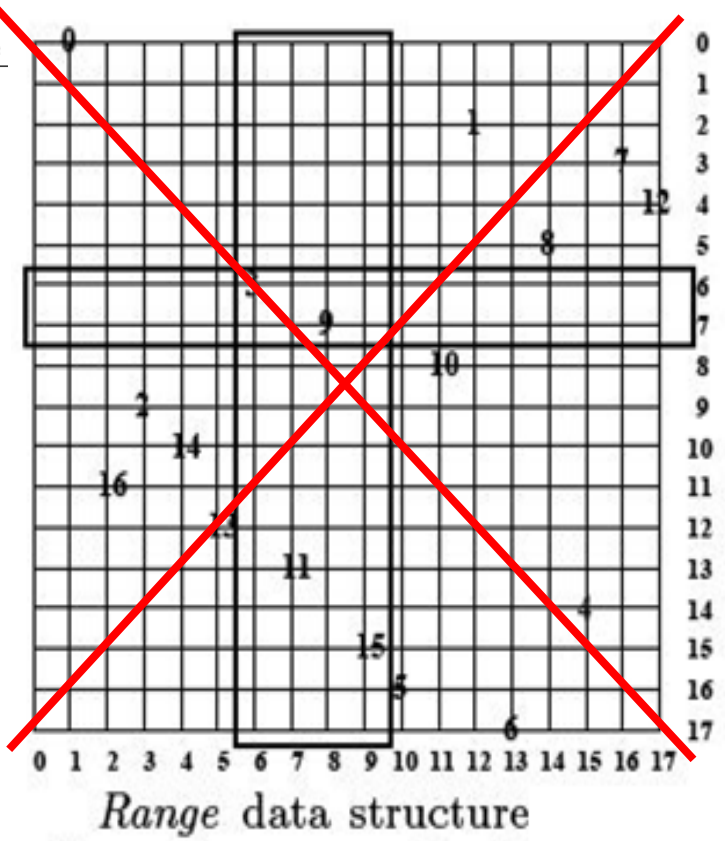
4.3 Βελτίωση

Βλέπουμε ότι στο στιγμιότυπο τύπου 2 χωρίζουμε στα δύο τη φράση που ψάχνουμε, δηλαδή P[1…i] και P[i+1…m]. Στο στιγμιότυπο τύπου 3 έχουμε τρεις διαμερίσεις ως εξής: P[1…i-1], P[i…j] και P[j+1…m]. Για i=j παρατηρούμε ότι είναι πολύ εύκολο να βρούμε τα στιγμιότυπα τύπου 2 με τον ίδιο τρόπο που βρίσκουμε τα στιγμιότυπα τύπου 3. Επομένως, γλυτώνουμε την δομή Range από την υλοποίησή μας κάτι που μειώνει τις απαιτήσεις και σε χώρο και σε χρόνο.



Επομένως οι απαιτούμενες δομές φαίνονται παρακάτω

i	S_{last}	S_α
1	0	a
2	0	l
3	1	_
4	1	Δ
5	0	$
6	0	b
7	0	l
8	0	r
9	1	_
10	1	b
11	1	Δ
12	1	p
13	1	r
14	1	Δ
15	1	Δ
16	1	a
17	1	Δ
18	1	Δ
19	1	Δ
20	1	Δ
21	0	a
22	0	d
23	1	l
24	1	Δ
25	1	a
26	1	p



	0				5				10				15				20				25				30				35				40				45				50		53
par:	(	(	(	(	)	)	(	(	(	)	)	)	)	(	(	)	)	(	(	)	)	)	(	(	(	)	)	)	)	)	(	(	(	(	)	)	)	)	(	(	(	(	)	)	)	)	)	)	)
B:	1	1	1	0			1	1	0					1	1	0					1	0			1	0			1	1	0					1	1	1	0					1	1	1	0			
ids:	0	1	17				3	15						14							4	12			10				16							6	11							2	7	9						
letts:	a	$	Δ				b	r	Δ					l	Δ						r	a	Δ		d	Δ			l	Δ						_	p	Δ						l	a	b	Δ						
Pos^{-1}:	1	5	4				6	13	24					7	17						8	21	11		22	15			23	18						9	26	20						2	16	10	14						

i:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
$Pos[i]$:	1	19	23	3	2	4	7	9	16	21	11	25	5	22	13	20	8	15	26	18	10	12	14	6	24	17

Κεφάλαιο 5
Επίλογος

Στην παραπάνω εργασία εξετάσαμε τη δομή LZ-Index η οποία έχει σαν σκοπό την αναζήτηση και ανάκτηση πληροφοριών μέσα από κείμενο. Αρχικά (Κεφάλαιο 1) είδαμε τις εφαρμογές και την χρησιμότητα τέτοιου είδους δομών. Έπειτα (Κεφάλαιο 2) αναλύσαμε τον αλγόριθμο LZ-Index και ακολούθως (Κεφάλαιο 3) εξετάσαμε τον μετασχηματισμό XBW ο οποίος εφαρμόζεται σε στατικά δένδρα ετικετών, όπως ακριβώς και οι δύο βασικές δενδρικές δομές του LZ-Index, με σκοπό τη μείωση του χώρου που καταλαμβάνουν και τον ταχύτερο χρόνο αναζήτησης φράσεων. Τέλος (Κεφάλαιο 4), δείξαμε πως αυτοί οι δύο αλγόριθμοι δύναται να συνδυασθούν έτσι ώστε να βελτιωθεί ο αλγόριθμος LZ-Index και σε χώρο και σε χρόνο.

Ο τομέας αυτός της επιστήμης υπολογιστών είναι πολύ σημαντικός και σίγουρα θα συνεχίσει να αναπτύσσεται με γρήγορους ρυθμούς. Μια πρόταση για περαιτέρω έρευνα είναι το κατά πόσο η παραπάνω συνδυαστική μορφή των δύο αλγορίθμων θα μπορούσε να αποκτήση και δυναμικά χαρακτηριστικά, όπως η πρόσθεση και η απόσβεση κόμβων.

Βιβλιογραφία

[1] Stronger Lempel-Ziv Based Compressed Text Indexing_
Diego Arroyuelo, Gonzalo Navarro, and Kunihiko Sadakane

[2] Structuring labeled trees for optimal succinctness, and beyond
Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini ,S. Muthukrishnan

[3] Compressing and indexing labeled trees, with applications
Paolo Ferragina† Fabrizio Luccio† Giovanni Manzini‡
S. Muthukrishnan

[4] D. Arroyuelo and G. Navarro. Space-efficient construction of LZ-index. In *Proc. ISAAC*, LNCS 3827, pages
1143–1152. Springer, 2005.

[5] D. Arroyuelo, G. Navarro, and K. Sadakane. Reducing the space requirement of LZ-index. In *Proc. CPM*, LNCS
4009, pages 319–330, 2006.

[6] J. Barbay, M. He, J. I. Munro, and S. S. Rao. Succinct indexes for strings, binary relations and multi-labeled
trees. In *Proc. SODA*, pages 680–689, 2007.

[7] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal
on Computing*, 17(3):427–462, 1988.

[8] P. Ferragina, R. Gonz´alez, G. Navarro, and R. Venturini. Compressed text indexes: From theory to practice!
2007. Submitted. http://pizzachili.dcc.uchile.cl/resources/cti jea07.pdf.

[9] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness,
and beyond. In *Proc FOCS*, pages 184–196, 2005.

[10] P. Ferragina, G. Manzini, V. M¨akinen, and G. Navarro. Compressed representations of sequences and full-text
indexes. *ACM Transactions on Algorithms (TALG)*, 3(2):article 20, 2007.

[11] A. Golynski, J. I. Munro, and S. S. Rao. Rank/select operations on large alphabets: A tool for text indexing. In
Proc. SODA, pages 368–373, 2006.

[12] R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string
matching. *SIAM J. Comput.*, 35(2):378–407, 2005.

[13] G. Manzini. An analysis of the Burrows-Wheeler transform. *Journal of the ACM*, 48(3):407–430, 2001.

[14] J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on
Computing*, 31(3):762–776, 2001.

[15] G. Navarro. Indexing text using the Ziv-Lempel trie. Technical Report TR/DCC-2003-0, Dept. of Computer
Science, Univ. of Chile, 2003.

[16] G. Navarro. Indexing text using the Ziv-Lempel trie. *Journal of Discrete Algorithms (JDA)*, 2(1):87–114, 2004.

[17] G. Navarro and V. M¨akinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):article 2, 2007.