Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακή Διατριβή
στα Πληροφοριακά Συστήματα

Wavelet Trees: Theory and Implementation

Θεόδωρος Δαρβούδης

Επιβλέπων Καθηγητής
Εμμανουήλ Χριστοδουλάκης

Σεπτέμβριος 2015

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Wavelet Trees: Theory and Implementation

Θεόδωρος Δαρβούδης

Επιβλέπων Καθηγητής
Εμμανουήλ Χριστοδουλάκης

Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε
προς μερική εκπλήρωση των απαιτήσεων για απόκτηση

μεταπτυχιακού τίτλου σπουδών
στα Πληροφοριακά Συστήματα

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

Σεπτέμβριος 2015

Περίληψη

Το αντικείμενο της παρούσας διατριβής είναι η ενδελεχής κατανόηση και παρουσίαση των χαρακτηριστικών και της λειτουργίας του wavelet tree, μιας πολλαπλών εφαρμογών δομής δεδομένων που αρχικά προτάθηκε για την συμπίεση κειμένου, δια της μελέτης της υφιστάμενης βιβλιογραφίας.

Επιπλέον, θα ανακτηθούν υφιστάμενες υλοποιήσεις της δομής, από ελεύθερες πηγές και θα ελεγχθεί η αποδοτικότητά τους με δοκιμές επί πραγματικών δεδομένων μεγάλου όγκου και διαφόρων ειδών.

Τέλος, θα πραγματοποιηθεί υλοποίηση της δομής, με την αξιοποίηση των προηγουμένων, και έμφαση στην ορθή λειτουργία επί δεδομένων μεγάλου όγκου, με την μέγιστη δυνατή λειτουργικότητα και αποτελεσματικότητα, συγκριτικά με τις υφιστάμενες δομές που θα ανακτηθούν από ελεύθερες πηγές.

Summary

The objective of this thesis is the understanding and presentation, of the characteristics of the wavelet tree, a multiple purpose data structure that was initially proposed for text compression applications, through a survey.

Furthermore, various implementations of the structure will be retrieved from free sources and their efficiency will be tested upon various sets of large data sets.

Finally, exploiting the aforementioned, an implementation of the structure will be produced, emphasising on functionality over large data sets while maximizing effectiveness and efficiency, comparing to various implementations of the structure, which will be retrieved from free sources.

Ευχαριστίες

Ευχαριστώ τους γονείς μου και την συμβία μου για την ανεκτικότητα και την κατανόησή τους, ως προς τον χρόνο που τους στέρησα ενόσω εκπονούσα την παρούσα.

Για την βοήθεια επί της εκπόνησης, αλλά και την υπομονή του, ευχαριστώ τον επιβλέποντα καθηγητή.

Περιεχόμενα

Κεφάλαιο 1
Εισαγωγή

Η συνεχής και με εκθετικό ρυθμό αύξηση του όγκου δεδομένων που υποστηρίζουν την ανθρώπινη δραστηριότητα αλλά και προκύπτουν από αυτήν, καθιστά επιβεβλημένη την ανάπτυξη και συνεχή βελτίωση των δομών καθώς και των αλγορίθμων, που θα συμπιέσουν αυτά τα δεδομένα, επιτυγχάνοντας έτσι την γρήγορη αφενός και την οικονομική αφετέρου, αποθήκευση τους σε σταθερά μέσα και διακίνηση τους δια των διαφόρων δικτύων.

Γενικά, η συμπίεση δεδομένων μπορεί να θεωρηθεί ως δοσοληψία, στην οποία το κέρδος είναι η μείωση του όγκου δεδομένων, ενώ το κόστος είναι οι απαραίτητοι επιπλέον υπολογισμοί (συμπίεση - αποσυμπίεση).

Συνεπεία, το σημαντικό ερώτημα είναι πάντοτε, αν μπορεί η πληροφορία να κωδικοποιηθεί πιο αποδοτικά, δηλαδή να βρεθεί ένας τρόπος να αποθηκεύει ή να μεταδοθεί με λιγότερα bit, χωρίς να απαιτείται σημαντικά μεγαλύτερος χρόνος για την προσπέλαση της.

1.1 Μη απωλεστική – απωλεστική συμπίεση

Ανάλογα με την σχέση της αρχικής πληροφορίας με αυτήν που προκύπτει από την αποσυμπίεση της αφού αυτή έχει συμπιεστεί, οι αλγόριθμοι συμπίεσης διακρίνονται σε μη απωλεστικούς ή αντιστρεπτούς (lossless compression) και σε απωλεστικούς ή μη αντιστρεπτούς (lossy compression).

1.1.1 Μη απωλεστική

Στην μη απωλεστική ή αντιστρεπτή συμπίεση, οι αλγόριθμοι δεν μεταβάλλουν τα χαρακτηριστικά της πληροφορίας κατά τη διάρκεια της συμπίεσης με αποτέλεσμα η πληροφορία που προκύπτει κατά την αποσυμπίεση να είναι ακριβές αντίγραφο της αρχικής.

Τεχνικές αυτής της μορφής χρησιμοποιούνται για τη συμπίεση δεδομένων τα οποία δε θα πρέπει να αλλοιωθούν κατά τη διαδικασία της συμπίεσης / αποσυμπίεσης. Συνήθως, αυτοί οι αλγόριθμοι εφαρμόζονται σε περιπτώσεις που δεν υπάρχει κανένα περιθώριο απωλειών. Για παράδειγμα, αν η πληροφορία που αποθηκεύεται ή μεταφέρεται είναι ένα πρόγραμμα υπολογιστή, ένα και μόνο αλλοιωμένο bit μπορεί να είναι αρκετό για να καταστήσει το πρόγραμμα άχρηστο.

Ευρέως διαδεδομένοι μη απωλεστικοί αλγόριθμοι είναι οι: Run Length Encoding (RLE), Huffman [06], LZW [26] (παρουσιάζονται συνοπτικά παρακάτω).

1.1.2 Απωλεστική

Στην απωλεστική συμπίεση αλλοιώνονται τα χαρακτηριστικά της αρχικής πληροφορίας κατά τη διαδικασία της συμπίεσης.

Τεχνικές αυτής της μορφής χρησιμοποιούνται κυρίως για τη συμπίεση ηχητικών σημάτων ή εικόνων, εκμεταλλευόμενες τα φυσιολογικά χαρακτηριστικά της ανθρώπινης όρασης και ακοής για να εισάγουν επιλεκτικά αλλοιώσεις στα δεδομένα οι οποίες γίνονται δύσκολα ή καθόλου αντιληπτές.

Ευρέως διαδεδομένοι απωλεστικοί αλγόριθμοι είναι οι: JPEG (για συμπίεση εικόνας), MPEG (για συμπίεση κινούμενης εικόνας) και MP3 (για συμπίεση ήχου).

1.2 Κωδικοποίηση πηγής – εντροπίας

Ανάλογα με τη μέθοδο που ακολουθείται για τη συμπίεση μίας πηγής πληροφορίας οι αλγόριθμοι συμπίεσης διακρίνονται σε κωδικοποίησης πηγής (source encoding) και σε κωδικοποίησης εντροπίας (entropy encoding).

1.2.1 Κωδικοποίηση πηγής

Στην κωδικοποίηση πηγής λαμβάνεται υπόψη η φύση των δεδομένων που συμπιέζονται. Οι μετασχηματισμοί τους οποίους υφίστανται τα αρχικά δεδομένα, με στόχο τη συμπίεση, εξαρτώνται άμεσα από τον τύπο τους. Για παράδειγμα, η ομιλία χαρακτηρίζεται από συχνά διαστήματα σιωπής, που μπορούν να συμπιεστούν δραστικά σε ένα ηχητικό σήμα δεδομένου ότι δεν περιέχουν καμία χρήσιμη ακουστική πληροφορία πέραν της διάρκειας τους. Δηλαδή, οι μετασχηματισμοί των δεδομένων κάνουν χρήση των ιδιαίτερων σημασιολογικών χαρακτηριστικών που αυτά εμπεριέχουν.

Οι τεχνικές κωδικοποίησης πηγής, στη γενική περίπτωση, επιτυγχάνουν μεγαλύτερα ποσοστά συμπίεσης σε σχέση με την κωδικοποίηση εντροπίας, μειονεκτούν όμως στη σταθερότητα, γιατί το ποσοστό συμπίεσης που επιτυγχάνουν διαφοροποιείται ανάλογα με το αντικείμενο που συμπιέζεται.

Η κωδικοποίηση πηγής μπορεί να λειτουργήσει και με απώλειες (απωλεστική) και χωρίς απώλειες (μη απωλεστική).

Οι τεχνικές κωδικοποίησης πηγής διακρίνονται περεταίρω σε:

- Προβλεπτικές: DPCM (Difference Pulse Code Modulation), DM (Difference Modulation)
- Μετασχηματισμού: FFT (Fast Fourier Transform), DCT (Discrete Cosine Transform)
- Στρωματοποίησης (Layered): Sub band Coding
- Διανυσματικές: Ταύτισης με προκαθορισμένα πρότυπα, Fractals

Οι παραπάνω τεχνικές απλώς αναφέρονται (δεν αναλύονται, αφού η ανάλυση αυτή δεν εμπίπτει στους στόχους της παρούσης διατριβής).

1.2.2 Κωδικοποίηση εντροπίας

Εντροπία καλείται ο (θεωρητικά) ελάχιστος αριθμός από bits ανά σύμβολο που απαιτείται για τη κωδικοποίηση - μετάδοση ενός μηνύματος.

Επειδή η εντροπία υποδεικνύει τη βέλτιστη συμπίεση (χωρίς απώλειες), που μπορεί να επιτευχθεί, η αποδοτικότητα κωδικοποίησης μιας μεθόδου συχνά συγκρίνεται με την εντροπία, η οποία υπολογίζεται με την εξίσωση του Shannon [25]:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i$$

όπου n ο αριθμός των συμβόλων που αποτελούν το μήνυμα και p_i η πιθανότητα εμφάνισης του συμβόλου i.

Συνοπτικά, παρουσιάζονται παρακάτω βασικές τεχνικές κωδικοποίησης εντροπίας (τεχνικές που χρησιμοποιούνται σε υλοποιήσεις που θα παρουσιάσουμε παρακάτω):

- Κωδικοποίηση μήκους διαδρομής (RLE - Run Length Encoding)

  Ο αλγόριθμος κωδικοποίησης μήκους διαδρομής αποτελεί μια από τις απλούστερες τεχνικές συμπίεσης. Βασίζεται στην παρατήρηση ότι σε πολλές περιπτώσεις μέσα σε μια ομάδα δεδομένων εμφανίζεται το ίδιο σύμβολο να επαναλαμβάνεται πολλές φορές στη σειρά. Ο αλγόριθμος διατρέχει την ακολουθία των bytes που αποτελούν τα δεδομένα προς συμπίεση και εντοπίζονται οι διαδοχικές επαναλήψεις του ίδιου χαρακτήρα. Στη συνέχεια αντικαθίστανται οι συνεχόμενες επαναλήψεις με το πλήθος τους, ακολουθούμενο από τον χαρακτήρα. Η μέθοδος είναι αποτελεσματική για την συμπίεση κειμένου με συχνές επαναλήψεις (τουλάχιστον τέσσερις) χαρακτήρων και εικόνων όπου η ίδια τιμή φωτεινότητας ή χρώματος επαναλαμβάνεται πολλές φορές.

- Κωδικοποίηση Huffman [06]

Στην κωδικοποίηση Huffman [06], επιτυγχάνεται συμπίεση καθώς ο κώδικας που αντιστοιχεί σε κάθε χαρακτήρα δεν έχει σταθερό μήκος αλλά μεταβλητό. Η κωδικοποίηση για τους πιο συχνά εμφανιζόμενους χαρακτήρες είναι μικρότερη από ό,τι για τους λιγότερο συχνά εμφανιζόμενους, με αποτέλεσμα το συνολικό μέγεθος να είναι μικρότερο, καθώς για τους συχνότερους χρησιμοποιούνται μόνο 2-3 bits και όχι 8 (κατά ASCII) ή 16 (κατά UNICODE). Η κωδικοποίηση βασίζεται στην στατιστική ανάλυση της εμφάνισης των συμβόλων σε μια ακολουθία δεδομένων.

- Κωδικοποίηση αντικατάστασης προτύπων (pattern substitution)

  Αναζήτηση των πιο συχνά χρησιμοποιούμενων ακολουθιών συμβόλων σε μια ακολουθία κι αντικατάστασή της εκάστης από αυτές από κωδικές λέξεις, πολύ μικρότερου μεγέθους. Η κωδικοποίηση αντικατάστασης προτύπων αποτελεί στην πραγματικότητα παραλλαγή της κωδικοποίησης μήκους διαδρομής. Ο ευρέως διαδεδομένος αλγόριθμος συμπίεσης LZW, ο οποίος παρουσιάστηκε από τους J. Ziv και A. Lempel το 1977 [27], και τροποποιήθηκε από τον T. Welch το 1984 [26] και χρησιμοποιείται από τα περισσότερα εμπορικά προγράμματα συμπίεσης δεδομένων, βασίζεται στη κωδικοποίηση αντικατάστασης προτύπων.

1.3 Αυτό-δεικτοδοτούμενες δομές δεδομένων

Σε όλες τις παραπάνω μεθόδους που αναφέρθηκαν, όταν αυτές εφαρμόζονται κατά μόνας, η οποιαδήποτε πρόσβαση στην πληροφορία απαιτεί πρώτα την αποσυμπίεση της.

Κατά την προηγούμενη δεκαετία εμφανίστηκαν οι αυτό-δεικτοδοτούμενες (self-indexing) δομές δεδομένων. Πρόκειται για δομές δεδομένων που δεν απαιτούν την αποθήκευση των αρχικών δεδομένων για να λειτουργήσουν, αλλά εμπεριέχουν τόσο τα δεδομένα σε συμπιεσμένη μορφή, όσο και δείκτες επ' αυτών, επιτρέποντας έτσι επιπλέον λειτουργικότητες επί των συμπιεσμένων δεδομένων, χωρίς να απαιτείται η προηγούμενη αποσυμπίεση τους. Τέτοιες λειτουργικότητες είναι για παράδειγμα η αναζήτηση του αριθμού επαναλήψεων ενός συμβόλου (rank) και η εύρεση της θέσης της νιοστής επανάληψης ενός συμβόλου (select).

Αρκετοί αλγόριθμοι και δομές έχουν παρουσιαστεί [01] [05] [08] [11] [16] [18] [22], στα πλαίσια της έρευνας επί των self-indexing δομών δεδομένων με την δομή wavelet tree να αποτελεί μέρος αυτών.

Αντικείμενο της παρούσας διατριβής είναι η ενδελεχής κατανόηση της δομής δεδομένων wavelet tree δια μέσου της μελέτης της υφιστάμενης βιβλιογραφίας, η παρουσίαση των χαρακτηριστικών της, η ανάκτηση υλοποιήσεων της από τρίτους (από ελεύθερες πηγές), ο έλεγχος της απόδοσης αυτών και τέλος η ανάπτυξη μια υλοποίησης, βασιζόμενης στα αποτελέσματα των παραπάνω, η λειτουργικότητα και η απόδοση της οποίας θα συγκριθεί με τις υλοποιήσεις τρίτων.

Για την διευκόλυνση της προσέγγισης του αναγνώστη, δίδεται ένα παράδειγμα εφαρμογής των wavelet trees: Υποθέτουμε το αρχείο προσβάσεων μιας δημοφιλούς ιστοσελίδας, για παράδειγμα της Wikipedia. Το αρχείο αποτελεί μια ακολουθία από τίτλους λημμάτων (URLs) με την ημερομηνία και ώρα της πρόσβασης τους και ταξινομείται με βάση αυτά. Χρησιμοποιώντας ένα wavelet tree κάθε λήμμα απεικονίζεται σαν ένα διακεκριμένο σύμβολο. Το σύνολο των διακεκριμένων συμβόλων αποτελεί το αλφάβητο του αρχείου. Με την αποθήκευση των λημμάτων ως σύμβολα, σε μορφή δυαδικού δένδρου είναι δυνατή η αναζήτηση του αριθμού επαναλήψεων (προσβάσεις) ενός λήμματος σ' όλο ή μέρος του αρχείου καθώς και ο εντοπισμός της θέσης της νιοστής επανάληψης (πρόσβασης) ενός λήμματος εντός της ακολουθίας.

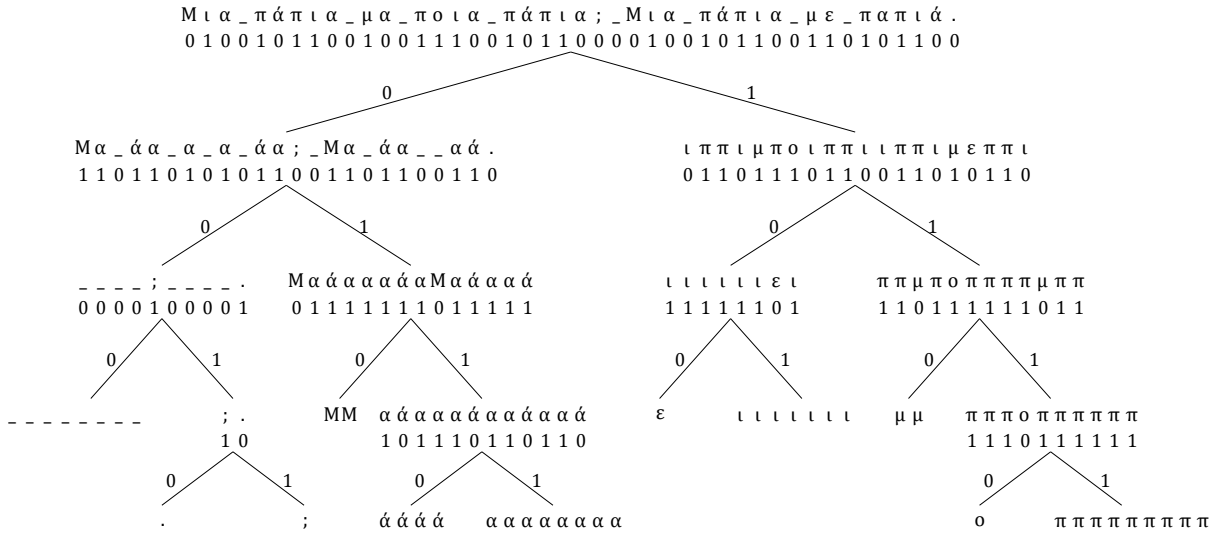
Κεφάλαιο 2
Παρουσίαση του wavelet tree

Τα wavelet trees πρωτοεμφανίζονται σε μια εργασία του Bernard Chazelle [03], ως μέρος μια δομής με σκοπό την αναζήτηση συμβολοσειρών εντός μια ακολουθίας, ενώ παρουσιάζονται για πρώτη φορά στην εργασία "High-Order Entropy-Compressed Text Indexes" [15] των R. Grossi, A. Gupta, J. S. Vitter, το 2003, σαν διακριτή δομή δεδομένων για την απεικόνιση μιας ακολουθίας συμβόλων και την άντληση πληροφοριών απ' αυτήν.

2.1 Δομή του wavelet tree

Το wavelet tree λειτουργεί κατά βάση ως ένας μηχανισμός που επιτρέπει την αποδοτική απεικόνιση ενός συνόλου συμβόλων από μη δυαδικό αλφάβητο (για παράδειγμα του ελληνικού αλφάβητου πλέον συμβόλων στίξεως και βοηθητικών όπως κενού και αλλαγής γραμμής), δια μέσω της υποκατάστασής του από το δυαδικό αλφάβητο δηλαδή τα σύμβολα "1" και "0" (υποκατάσταση του μη δυαδικού αλφάβητου με δυαδικό) και την άντληση πληροφοριών από αυτό, με λογαριθμική απομείωση του απαιτούμενου αποθηκευτικού χώρου, ήτοι το wavelet tree οργανώνει μια ακολουθία ως μια ιεραρχία δυαδικών διανυσμάτων.

Συγκεκριμένα, για μια δεδομένη ακολουθία *A*, μήκους *μ*, που απαρτίζεται από σύμβολα που ανήκουν σ' ένα ταξινομημένο αλφάβητο *Σ* μεγέθους *σ*, το wavelet tree *T* είναι ένα πλήρες δυαδικό δένδρο, στους τερματικούς κόμβους (φύλλα) του οποίου αποθηκεύονται τα σύμβολα του δοθέντος αλφάβητου με ταξινόμηση από αριστερά προς τα δεξιά, ήτοι ο αριθμός των φύλλων του δένδρου ισούται με τον αριθμό των στοιχείων του αλφάβητου *σ*. Ο ριζικός κόμβος του δένδρου περιέχει την αρχική ακολουθία *A*, απεικονισμένη ως δυαδικό διάνυσμα, ήτοι το πρώτο μισό του συνολικού αλφάβητου κωδικοποιείται ως "0" ενώ το δεύτερο μισό ως "1". Όλοι οι εσωτερικοί κόμβοι του δένδρου αποτελούν υποσύνολα της αρχικής ακολουθίας *A*, αποθηκευμένα επίσης ως δυαδικά διανύσματα, ως εξής: το αριστερό υποδένδρο περιέχει όλους τους κωδικοποιημένους ως "0" χαρακτήρες ενώ το δεξί υποδένδρο περιέχει όλους τους κωδικοποιημένους ως "1" χαρακτήρες.



Σχήμα 2.1: Το wavelet tree για την ακολουθία *A* = "Μια πάπια μα ποια πάπια; Μια πάπια με παπιά". Τα κενά μεταξύ των λέξεων απεικονίζονται για πρακτικούς λόγους ως κάτω παύλες. Τα υποσύνολα της ακολουθίας για κάθε εσωτερικό κόμβο και οι λεζάντες επί των ακμών απεικονίζονται επεξηγηματικά. Το δένδρο αποθηκεύει μόνον την τοπολογία και τα δυαδικά διανύσματα.

Σε κάθε κόμβο του δένδρου αποθηκεύεται ένα δυαδικό διάνυσμα, όπως απεικονίζεται στο σχήμα 2.1, έτσι ώστε να είναι δυνατή η ανάκτηση οποιουδήποτε συμβόλου της ακολουθίας και η απάντηση σε ερωτήματα πλήθους επαναλήψεων ενός συμβόλου του δοθέντος αλφαβήτου (rank) και θέσης στην αρχική ακολουθία της νιοστής επανάληψής ενός συμβόλου του δοθέντος αλφαβήτου (select) .

Για της δημιουργία του wavelet tree εκ της αρχικής ακολουθίας ακολουθούμε τα παρακάτω βήματα: Τα σύμβολα της αρχικής ακολουθίας κωδικοποιούνται είτε ως "0" εφόσον είναι μικρότερα

ή ίσα του μεσαίου συμβόλου του αλφαβήτου, είτε ως "1" εφόσον είναι μεγαλύτερα και αποθηκεύονται ως δυαδικό διάνυσμα στον ριζικό κόμβο. Εν συνεχεία η ακολουθία διαιρείται σε δύο (2) υπό ακολουθίες με βάση το ως άνω κριτήριο (σύμβολα μικρότερα ή ίσα του μεσαίου συμβόλου αποθηκεύονται στην πρώτη (1η) υπό ακολουθία και σύμβολα μεγαλύτερα του μεσαίου συμβόλου αποθηκεύονται στην δεύτερη (2η) υπό ακολουθία). Τα σύμβολα και της πρώτης (1ης) και της δεύτερης (2ης) υπό ακολουθίας κωδικοποιούνται εκ νέου είτε ως "0" εφόσον είναι μικρότερα ή ίσα του μεσαίου συμβόλου του αλφαβήτου, είτε ως "1" εφόσον είναι μεγαλύτερα και αποθηκεύονται ως δυαδικά διανύσματα αντίστοιχα στο αριστερό και δεξί παιδί του ριζικού κόμβου. Η διαδικασία εκτελείται αναδρομικά έως ότου η κάθε υπό ακολουθία να περιέχει μόνον ένα (1) σύμβολο.

Είναι φανερό ότι το ύψος του wavelet tree ισούται με το λογάριθμο με βάση το δύο (2) του μεγέθους του αλφαβήτου ($\log_2 \sigma$).

Για την ανάκτηση οποιουδήποτε συμβόλου του αλφαβήτου ακολουθούμε την παρακάτω διαδικασία: Ας υποθέσουμε ότι επιθυμούμε να προσπελάσουμε το ενδέκατο σύμβολο της ακολουθίας του σχήματος 2.1. Το ενδέκατο σύμβολο του δυαδικού διανύσματος του ριζικού κόμβου είναι "1". Καταμετρούμε το σύνολο των συμβόλων "1" μέχρι την ενδέκατη (11η) θέση του δυαδικού διανύσματος του ριζικού κόμβου και βρίσκουμε πέντε (5). Στη συνέχεια μεταβαίνουμε στο δεξί υποδένδρο αφού το σύμβολο που προσπελάσαμε ήταν "1". Στο υποδένδρο ανακτούμε το πέμπτο (5ο) σύμβολο του δυαδικού διανύσματος και βρίσκουμε ότι είναι "1". Καταμετρούμε το σύνολο των συμβόλων "1" μέχρι την Πέμπτη (5η) θέση του δυαδικού διανύσματος του κόμβου και βρίσκουμε τρία (3). Στη συνέχεια μεταβαίνουμε στο δεξί υποδένδρο αφού το σύμβολο που προσπελάσαμε ήταν "1". Στο υποδένδρο ανακτούμε το πέμπτο (3ο) σύμβολο του δυαδικού διανύσματος και βρίσκουμε ότι είναι "0". Καταμετρούμε το σύνολο των συμβόλων "0" μέχρι την τρίτη (3η) θέση του δυαδικού διανύσματος του κόμβου και βρίσκουμε ένα (1). Στη συνέχεια μεταβαίνουμε στο αριστερό υποδένδρο αφού το σύμβολο που προσπελάσαμε ήταν "0", ο οποίος είναι τερματικός και αντιστοιχεί στο σύμβολο "μ" (γνωστό εκ της ταξινομήσεως του αλφαβήτου). Με την ανωτέρω διαδικασία έχουμε καταφέρει να ανακτήσουμε τον ενδέκατο (11ο) χαρακτήρα ("μ") της αρχικής ακολουθίας με τη βοήθεια των δυαδικών διανυσμάτων στους κόμβους του wavelet tree.

Θα εξηγήσουμε στη συνέχεια πως πραγματοποιείται η καταμέτρηση του πλήθους των επαναλήψεων ενός συμβόλου του δοθέντος αλφαβήτου (rank), στη δομή του wavelet tree.

Στο παράδειγμα του σχήματος 2.1 της προηγούμενης σελίδας, θα απαντήσουμε στο ερώτημα του αριθμού επαναλήψεων του συμβόλου “ι” στην ακολουθία. Ξεκινώντας από τον ριζικό κόμβο, γνωρίζουμε ότι το σύμβολο “ι” έχει κωδικοποιηθεί σαν “1”, αφού το πραγματικό αλφάβητό της ακολουθίας μας είναι γνωστό και ταξινομημένο, είναι γνωστό σε πιο μέρος της ακολουθίας βρίσκεται το κάθε σύμβολο, στο 1ο μισό που έχει κωδικοποιηθεί ως “0” ή το δεύτερο μισό που έχει κωδικοποιηθεί ως ”1”. Στην συνέχεια μετράμε το σύνολο των συμβόλων “1” στο δυαδικό διάνυσμα του ριζικού κόμβου και βρίσκουμε 20. Αυτό σημαίνει πως υπάρχουν στην ακολουθία μας 20 σύμβολα που είναι είτε “ε” είτε “ι” είτε “μ” είτε “ο” είτε “π”. Στη συνέχεια μεταβαίνουμε στο δεξί υποδένδρο αφού το σύμβολο “ι” είναι κωδικοποιημένο ως “1”. Με το ίδιο τρόπο γνωρίζουμε ότι εδώ το σύμβολο “ι” είναι κωδικοποιημένο ως “0”. Καταμετρούμε λοιπόν το σύνολο των συμβόλων “0” στο δυαδικό διάνυσμα και βρίσκουμε 8 σύμβολα που είναι είτε “ε” είτε “ι”. Στη συνέχεια μεταβαίνουμε στο αριστερό υποδένδρο αφού το σύμβολο “ι” είναι κωδικοποιημένο ως “0”. Γνωρίζουμε ότι εδώ το σύμβολο “ι” είναι κωδικοποιημένο ως “1”. Καταμετρούμε λοιπόν το σύνολο των συμβόλων “1” στο δυαδικό διάνυσμα και βρίσκουμε 7 σύμβολα που είναι αποκλειστικά “ι”. Άρα έχουμε την απάντησή μας, ήτοι στην αρχική ακολουθία απαντώνται επτά (7) “ι”.

Είναι φανερό ότι η ανωτέρω διαδικασία μπορεί να περιοριστεί μόνον σε ένα μέρος της αρχικής ακολουθίας ή να επεκταθεί στο σύνολό της όπως στο παράδειγμα. Μπορεί δηλαδή το ερώτημα να τίθεται με τη μορφή: πόσα “ι” απαντώνται μέχρι την θέση 15 της αρχικής ακολουθίας; Η απάντηση θα είναι δύο (2). Αρκεί στην διαδικασία ,που περιεγράφηκε στην προηγούμενη παράγραφο, να περιορίζουμε την καταμέτρηση στο μέρος της ακολουθίας που απαιτείται.

Τέλος θα εξηγήσουμε πως πραγματοποιείται η εύρεση της θέσης στην αρχική ακολουθία της νιοστής επανάληψής ενός συμβόλου του δοθέντος αλφαβήτου (select), στη δομή του wavelet tree.

Στο παράδειγμα του σχήματος 2.1 και πάλι, θα αναζητήσουμε την θέση στην αρχική ακολουθία της τρίτης (3ης) επανάληψης του συμβόλου “ι”. Ξεκινώντας αυτή τη φορά από τον τερματικό κόμβο που αντιστοιχεί στο σύμβολο “ι” θα πραγματοποιήσουμε μια αντίστροφη διαδρομή (από τον τερματικό στο ριζικό κόμβο) στη δενδρική δομή. Μεταβαίνουμε στον πατρικό κόμβο, όπου γνωρίζουμε ότι το σύμβολο “ι” είναι κωδικοποιημένο ως “1” αφού ο τερματικός κόμβος είναι το δεξί παιδί του. Εντοπίζουμε την τρίτη (3η) επανάληψη του συμβόλου “1” στην τρίτη (3η) θέση του δυαδικού διανύσματος. Μεταβαίνουμε στον πατρικό κόμβο, όπου γνωρίζουμε ότι το σύμβολο “ι” είναι κωδικοποιημένο ως “0” αφού ο κόμβος εκ του οποίου προερχόμαστε είναι το αριστερό παιδί του. Εντοπίζουμε την τρίτη (3η) επανάληψη του συμβόλου “0” στην όγδοη (8η) θέση του δυαδικού

διανύσματος. Μεταβαίνουμε στον πατρικό κόμβο (που στο παράδειγμά μας είναι ο ριζικός κόμβος), όπου γνωρίζουμε ότι το σύμβολο "ι" είναι κωδικοποιημένο ως "1" αφού ο κόμβος εκ του οποίου προερχόμαστε είναι το δεξί παιδί του. Εντοπίζουμε την όγδοη (8η) επανάληψη του συμβόλου "1" στην δέκατη έκτη (16η) θέση του δυαδικού διανύσματος. Άρα η τρίτη (3η) επανάληψη του συμβόλου "ι" βρίσκεται στην δέκατη έκτη (16η) θέση της αρχικής ακολουθίας.

2.2 Χρήσεις του wavelet tree

Το wavelet tree, όπως αναλύθηκε ανωτέρω, απαιτεί το πολύ $\mu\lceil\log_2\sigma\rceil$ bit και μπορεί να δημιουργηθεί σε χρόνο $O(\mu\log_2\sigma)$, όπου μ, το μέγεθος της αρχικής ακολουθίας και σ, το μέγεθος του αλφαβήτου. Υποστηρίζει δε την ανάκτηση οποιοδήποτε συμβόλου της ακολουθίας (access) καθώς και την καταμέτρηση του πλήθους των επαναλήψεων ενός συμβόλου του δοθέντος αλφαβήτου (rank) σε χρόνο $O(\log_2\sigma)$, με τη χρήση της δομής καταμέτρησης του πλήθους επαναλήψεων συμβόλου σε δυαδική ακολουθία του Jacobsen [17], η οποία επιτυγχάνει την καταμέτρηση σε σταθερό χρόνο.

Επιπλέον υποστηρίζεται η εύρεση της θέσης στην αρχική ακολουθία της νιοστής επανάληψης ενός συμβόλου του δοθέντος αλφαβήτου (select) σε χρόνο επίσης $O(\log_2\sigma)$, με τη χρήση της δομής εύρεση της νιοστής επανάληψής συμβόλου σε δυαδική ακολουθία του Clark [04], η οποία επιτυγχάνει την εύρεση σε σταθερό χρόνο.

Στην πράξη όλα τα δυαδικά διανύσματα που αποθηκεύονται στους κόμβους του wavelet tree έχουν αλυσιδωτή δομή και δεν αποθηκεύονται δείκτες στον πατρικό για τους θυγατρικού κόμβους [05], δομή η οποία απαιτεί την ταξινόμηση και την συνέχεια του αλφαβήτου, ήτοι το αλφάβητο θα πρέπει να κωδικοποιείται σε μια συνεχή ακολουθία (0…..σ-1).

Και οι δύο ανωτέρω υποστηρικτικές δομές, δεν επιτυγχάνουν καμία συμπίεση επί των δυαδικών διανυσμάτων στα οποία εφαρμόζονται.

Έχει προταθεί από τον Pagh [23] και αναπτύχθηκε περεταίρω από τους R. Raman, V. Raman, και S. Rao [24] δομή, γνωστή ως RRR, από τα ονόματα των τριών, η οποία επιτυγχάνει μεν, σύμφωνα με μετρήσεις των Claude και Navarro [05], συμπίεση των δυαδικών διανυσμάτων κατά 50% περίπου, με κόστος όμως τον τετραπλασιασμό του χρόνου που απαιτείται για τις λειτουργικότητες

ανάκτησης συμβόλου, χρόνου καταμέτρησης του πλήθους επαναλήψεων συμβόλου και εύρεσης της νιοστής επανάληψής συμβόλου.

Η απόδοση του wavelet tree, σαν αυτόνομου μηχανισμού συμπίεσης [07], είναι άμεσα συναρτώμενη από το μέγεθος των συμβόλων του αλφαβήτου και του μέγεθος του αλφαβήτου. Ο ρυθμός συμπίεσης είναι ανάλογος του μεγέθους του συμβόλου και αντιστρόφως ανάλογος του μεγέθους του αλφαβήτου. Για παράδειγμα, εάν ένα έχουμε ένα αλφάβητο που αποτελείται από εκατό (100) σύμβολα, το καθένα από τα οποία απαιτεί 2 byte (16 bit) για την αποθήκευσή του και μια ακολουθία που περιλαμβάνει δέκα χιλιάδες (10.000) σύμβολα, τότε ο αποθηκευτικό χώρος που θα απαιτείται για την αποθήκευση της ασυμπίεστης ακολουθίας θα είναι είκοσι χιλιάδες (20.000) byte, το ύψος του wavelet tree θα είναι $\log_2 100 \cong 7$, το οποίο σημαίνει ότι κάθε σύμβολο θα απαιτεί για την αποθήκευσή του στο wavelet tree 7 bit και συνεπεία ο ρυθμός συμπίεσης θα είναι $7/16 \cong 44\%$. Τα παραπάνω ισχύουν για wavelet tree με ασυμπίεστα δυαδικά διανύσματα. Εφόσον τα δυαδικά διανύσματα συμπιεστούν κατά RRR (ως προηγούμενη παράγραφος) μπορεί να επιτευχθεί περεταίρω συμπίεση ως και 50%, όπως προαναφέρθηκε.

Το wavelet tree, παρά τις δυνατότητές του ως αυτόνομος μηχανισμός συμπίεσης, χρησιμοποιείται κυρίως μαζί με άλλες μεθόδους, όπως οι κωδικοποιήσεις Huffman [06], αντικατάστασης προτύπων και μήκους διαδρομής (RLE - Run Length Encoding) [07] [09] [20].

Ως συνέπεια του βασικού δομικού χαρακτηριστικού του wavelet tree, ήτοι της απεικόνισης της αριστερής ακμής κάθε κόμβου ως “0” και της δεξιάς ως “1” που επιτρέπει την απεικόνιση κάθε διακριτού συμβόλου κάθε πιθανού αλφάβητου στους τερματικούς κόμβους της δενδρικής δομής, προκύπτει ότι μπορεί να επιτευχθεί μεγαλύτερος ρυθμός συμπίεσης χρησιμοποιώντας αντί της δυαδικής απεικόνισης κάθε διακριτού συμβόλου (αυτόνομο wavelet tree) την κωδικοποίηση του κατά Huffman [01] [05] [14] [16] [18]. Σ’ αυτήν την προσέγγιση το δένδρο δεν είναι ισοζυγισμένο με αποτέλεσμα η χρονική πολυπλοκότητα διεργασιών όπως ανάκτησης συμβόλων, καταμέτρηση πλήθους επαναλήψεων συμβόλου (rank) και εύρεσης νιοστής επανάληψης συμβόλου (select) επιδεινώνεται θεωρητικά σε $O(\sigma)$ από $O(\log_2 \sigma)$. Στην πραγματικότητα όμως η χρονική πολυπλοκότητα των ανωτέρω διεργασιών οριοθετείται σε $O(\log_2 \sigma)$, ως αποτέλεσμα της εντροπίας και του μικρότερου ύψους του δένδρου που δημιουργείται και άρα μικρότερης διαδρομής για την ανάκτηση κάθε τερματικού κόμβου.

Στην υλοποίηση wavelet tree που αναλύεται στο επόμενο κεφάλαιο έχει χρησιμοποιηθεί, μια απλή ως προς την υλοποίηση, κωδικοποίηση αντικατάστασης προτύπων, με ικανοποιητική αποδοτικότητα.

Η κωδικοποίηση μήκους διαδρομής (RLE - Run Length Encoding), έχει χρησιμοποιηθεί σε συνδυασμό με την μετατροπή Burrows-Wheeler [02], όπου η απόδοση παρουσιάζεται σημαντικά αυξημένη λόγω του συνδυασμού ουσιαστικά τριών μεθόδων [08] [11] [22]. Πρώτα εφαρμόζεται η μετατροπή Burrows-Wheeler [02] επί της αρχικής ακολουθίας, στη συνέχεια η έξοδος αυτής κωδικοποιείται με τη μέθοδο μήκους διαδρομής, η οποία εν τέλει αποθηκεύεται στο wavelet tree.

Πολύ περιληπτικά, η μετατροπή Burrows-Wheeler [02] είναι μια μέθοδος που επιτρέπει την αναδιάταξη των συμβόλων μιας ακολουθίας έτσι ώστε επαναλαμβανόμενα σύμβολα να τοποθετούνται συναπτά με σκοπό την ευκολότερη και αποδοτικότερη συμπίεση. Η μετατροπή επιτυγχάνεται με την λεξικογραφική ταξινόμηση όλων ακολουθιών που προκύπτουν από την σειριακή περιστροφή των συμβόλων της αρχικής ακολουθίας κατά μία θέση τη φορά. Το σύνολο των τελευταίων συμβόλων κάθε ακολουθίας δίνει το αποτέλεσμα. Στο πίνακα που ακολουθεί δίδεται το παράδειγμα μετατροπής Burrows-Wheeler [02] της ακολουθίας "^BANANA|" (με το σύμβολο "|" προσδιορίζεται το τέλος της ακολουθίας):

Μετατροπή Burrows-Wheeler			
Έισοδος (αρχική ακολουθία)	**Περιστροφή συμβόλων κατά μία θέση τη φορά**	**Λεξικογραφική ταξινόμιση**	**Έξοδος (τελευταίο σύμβολο κάθε ταξινομημένης ακολουθίας)**
^BANANA|	^BANANA|	**A**NANA|^B	BNN^AA|A
	|^BANANA	**A**NA|^BAN	
	A|^BANAN	**A**|^BANAN	
	NA|^BANA	**B**ANANA|^	
	ANA|^BAN	**N**ANA|^BA	
	NANA|^BA	**N**A|^BANA	
	ANANA|^B	**^**BANANA|	
	BANANA|^	|^BANANA	

Πίνακας 2.1: Μετατροπή Burrows-Wheeler [02] της ακολουθίας "^BANANA|".

Η αρχική ακολουθία μπορεί να ανακτηθεί από την είσοδο (έξοδο της μετατροπής Burrows-Wheeler) δια της επαναλαμβανόμενης λεξικογραφικής ταξινόμησης και ένωσης εισόδου και ταξινομημένης ακολουθίας. Στο πίνακα που ακολουθεί δίδεται το παράδειγμα ανάκτησης της ακολουθίας "^BANANA|" μετά την μετατροπή Burrows-Wheeler [02] στο σχήμα 2.2:

Αναστροφή Burrows-Wheeler							
(1) **Είσοδος**	**(2)** **1η ταξ/ση** **(ταξ/ση 1)**	**(3)** **1η προσθήκη** **(1+2)**	**(4)** **2η ταξ/ση** **(ταξ/ση 3)**	**(5)** **2η προσθήκη** **(1+4)**	**(6)** **3η ταξ/ση** **(ταξ/ση 5)**	**(7)** **3η προσθήκη** **(1+6)**	**(8)** **4η ταξ/ση** **(ταξ/ση 7)**
B	A	BA	AN	BAN	ANA	BANA	ANAN
N	A	NA	AN	NAN	ANA	NANA	ANA|
N	A	NA	A|	NA|	A|^	NA|^	A|^B
^	B	^B	BA	^BA	BAN	^BAN	BANA
A	N	AN	NA	ANA	NAN	ANAN	NANA
A	N	AN	NA	ANA	NA|	ANA|	NA|^
|	^	|^	^B	|^B	^BA	|^BA	^BAN
A	|	A|	|^	A|^	|^B	A|^B	|^BA
(9) **4η προσθήκη** **(1+8)**	**(10)** **5η ταξ/ση** **(ταξ/ση 9)**	**(11)** **5η προσθήκη** **(1+10)**	**(12)** **6η ταξ/ση** **(ταξ/ση 11)**	**(13)** **6η προσθήκη** **(1+12)**	**(14)** **7η ταξ/ση** **(ταξ/ση 13)**	**(15)** **7η προσθήκη** **(1+14)**	**(16)** **8η ταξ/ση** **(ταξ/ση 15)**
BANAN	ANANA	BANANA	ANANA|	BANANA|	ANANA|^	BANANA|^	ANANA|^B
NANA|	ANA|^	NANA|^	ANA|^B	NANA|^B	ANA|^BA	NANA|^BA	ANA|^BAN
NA|^B	A|^BA	NA|^BA	A|^BAN	NA|^BAN	A|^BANA	NA|^BANA	A|^BANAN
^BANA	BANAN	^BANAN	BANANA	^BANANA	BANANA|	^BANANA|	BANANA|^
ANANA	NANA|	ANANA|	NANA|^	ANANA|^	NANA|^B	ANANA|^B	NANA|^BA
ANA|^	NA|^B	ANA|^B	NA|^BA	ANA|^BA	NA|^BAN	ANA|^BAN	NA|^BANA
|^BAN	^BANA	|^BANA	^BANAN	|^BANAN	^BANANA	|^BANANA	^BANANA|
A|^BA	|^BAN	A|^BAN	|^BANA	A|^BANA	|^BANAN	A|^BANAN	|^BANANA
Έξοδος ^BANANA|							

Πίνακας 2.2: Αναστροφή Burrows-Wheeler [02] της ακολουθίας "^BANANA|".

Η έξοδος είναι εκείνη η γραμμή της τελευταία στήλης όπου το σύμβολο "|" βρίσκεται στο τέλος της ακολουθίας.

Στον παραπάνω συνδυασμό μεθόδων προστέθηκε από τους Paolo Ferragina και Giovanni Manzini [10] η δομή FM-index (Full-text index in Minute space), που επιτρέπει ταχύτερη εύρεση δείγματος στην ακολουθία που έχει υποστεί την μετατροπή Burrows-Wheeler [02], εφαρμόζοντας αναζήτηση που ξεκινάει από το τελευταίο σύμβολο της ακολουθίας και μετακινείται προς το αρχικό εφόσον η αναζήτηση είναι επιτυχής για το προηγούμενο σύμβολο. Διάφορες εκδοχές αυτής της προσέγγισης έχουν παρουσιαστεί [11], όπου παρατηρούνται διάφοροι συσχετισμοί χωρικής και χρονικής πολυπλοκότητας.

Κεφάλαιο 3
Υλοποίηση του wavelet tree

Η υλοποίηση του wavelet tree (στο εξής η υλοποίηση θα αναφέρεται ως wavelet tree), μετά την ενδελεχή κατανόηση του τρόπου λειτουργίας δια της μελέτης της υφιστάμενης βιβλιογραφίας καθώς και την μελέτη υφιστάμενων υλοποιήσεων τρίτων, αναλύεται σε δύο μέρη.

- Το κατεξοχήν wavelet tree που εκτελεί τις διεργασίες δημιουργίας, αποθήκευσης και ανάκτησης του, καθώς και των διεργασιών ανάκτησης συμβόλων του δοθέντος αλφαβήτου, καταμέτρησης του πλήθους επαναλήψεων συμβόλου και εύρεσης της νιοστής επανάληψης συμβόλου.

- Τον μηχανισμό συμπίεσης, που από όσο γνωρίζουμε, αποτελεί μια πρωτότυπη ιδέα, σε ότι αφορά τον τρόπο κατάτμησης της αρχικής ακολουθίας, παρότι κατηγοριοποιείται σαφώς, στην ευρύτερη κωδικοποίηση αντικατάστασης προτύπων (pattern substitution).

Στο σύνολό του το wavelet tree υλοποιήθηκε σε γλώσσα C και σε περιβάλλον linux, με την βοήθεια της πλατφόρμας NetBeans IDE 8.0.2 και μεταγλωττίστηκε με τον gcc 4.9.3.

Όλα τα αρχεία κώδικα (source files) επισυνάπτονται της παρούσης καθώς και make file για την μεταγλώττιση του προγράμματος.

Το πρόγραμμα έχει ελεγχθεί τόσο για την μεταγλώττισή του όσο και την εκτέλεσή του σε περιβάλλον Linux Mint 17.2 64-bit, Ubuntu 14.04.3 LTS 64-bit και Microsoft® Windows (Windows 7 professional N 64-bit, περιβάλλον Cygwin 2.2.1).

3.1 Το wavelet tree

Η υλοποίηση του κατεξοχήν wavelet tree, δεν παρουσιάζει ιδιαιτερότητες, σε σχέση με υλοποιήσεις τρίτων, σε ότι αφορά τους γενικούς αλγορίθμους που χρησιμοποιήθηκαν.

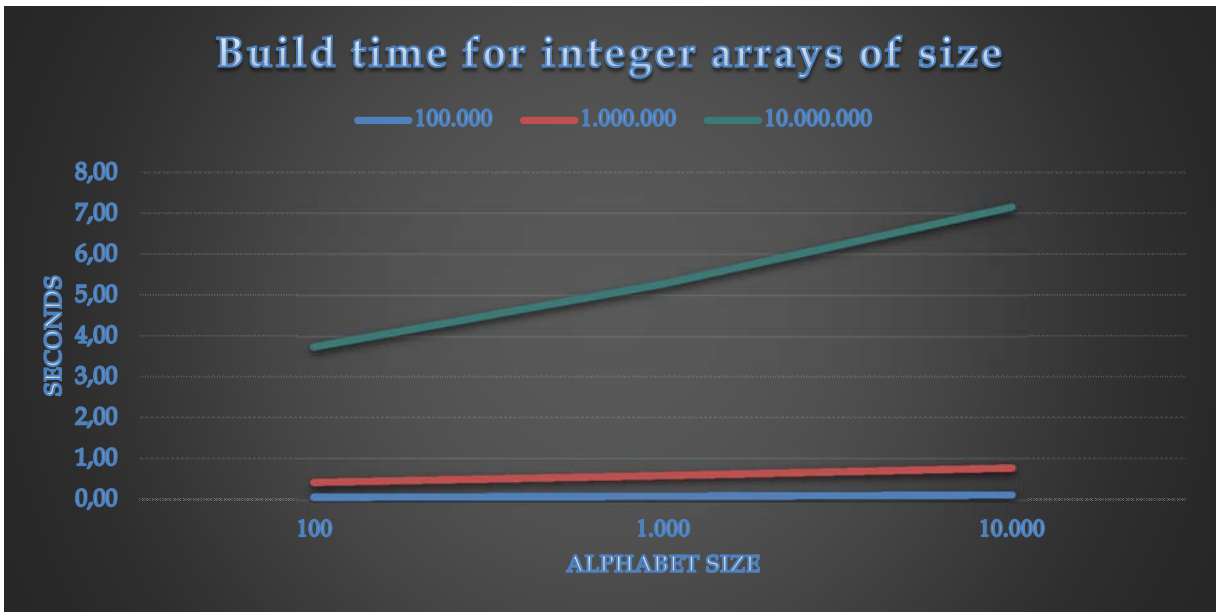
Για την υλοποίηση των δυαδικών διανυσμάτων έχει χρησιμοποιηθεί η δομή καταμέτρησης του πλήθους επαναλήψεων συμβόλου σε δυαδική ακολουθία του Jacobsen [17], η οποία επιτυγχάνει την καταμέτρηση σε σταθερό χρόνο καθώς και η δομή εύρεση της νιοστής επανάληψής συμβόλου σε δυαδική ακολουθία του Clark [04], η οποία επίσης επιτυγχάνει την εύρεση σε σταθερό χρόνο.

Για την δημιουργία του wavelet tree έχουν χρησιμοποιηθεί οι αλγόριθμοι που έχουν αναλυτικά περιγραφεί στο προηγούμενο κεφάλαιο ενότητα 2.1.

Στα συνημμένα της παρούσης συγκαταλέγεται και η υλοποίηση χωρίς τον μηχανισμό συμπίεσης, όπου χρησιμοποιούνται ως ακολουθία εισόδου μ ακέραιοι αριθμοί συνεχόμενου αλφάβητου ($0 \ldots \sigma$).

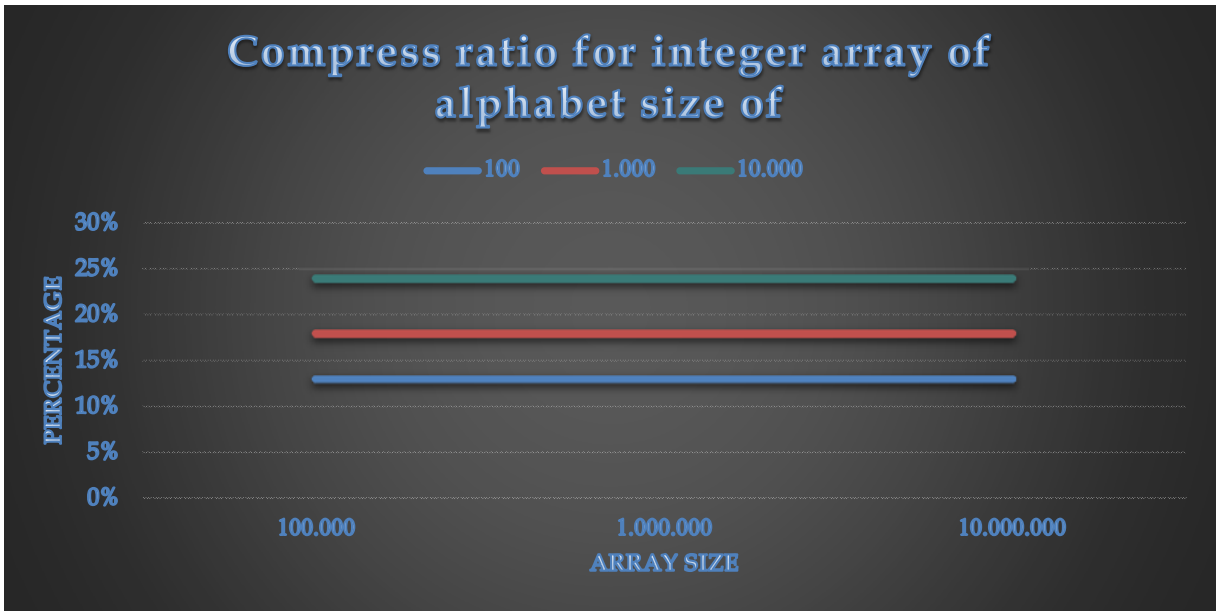
Ακολουθεί σειρά γραφημάτων όπου εμφανίζεται η απόδοση της ως άνω υλοποίησης στους επιμέρους τομείς του χρόνου δημιουργίας του wavelet tree, του ρυθμού συμπίεσης, της ταχύτητας ανάκτησης συμβόλων (πραγματοποιήθηκαν 50 χιλ. ανακτήσεις τυχαίων συμβόλων), της καταμέτρησης πλήθους επαναλήψεων συμβόλου (πραγματοποιήθηκαν 50 χιλ. καταμετρήσεις επαναλήψεων τυχαίων συμβόλων) και εύρεσης της νιοστής επανάληψής συμβόλου (πραγματοποιήθηκαν 50 χιλ. αναζητήσεις νιοστής επανάληψης τυχαίων συμβόλων), για τρία (3) διαφορετικά μεγέθη ακολουθίας και επίσης τρία (3) διαφορετικά μεγέθη αλφαβήτου.

Στο γράφημα 3.1, φαίνεται ότι ο χρόνος δημιουργίας του wavelet tree είναι συνάρτηση κυρίως του μεγέθους της ακολουθίας και πολύ λιγότερο του μεγέθους του αλφαβήτου, όπως προκύπτει και από την πολυπλοκότητα του αλγορίθμου δημιουργίας $O(\mu \log_2 \sigma)$.



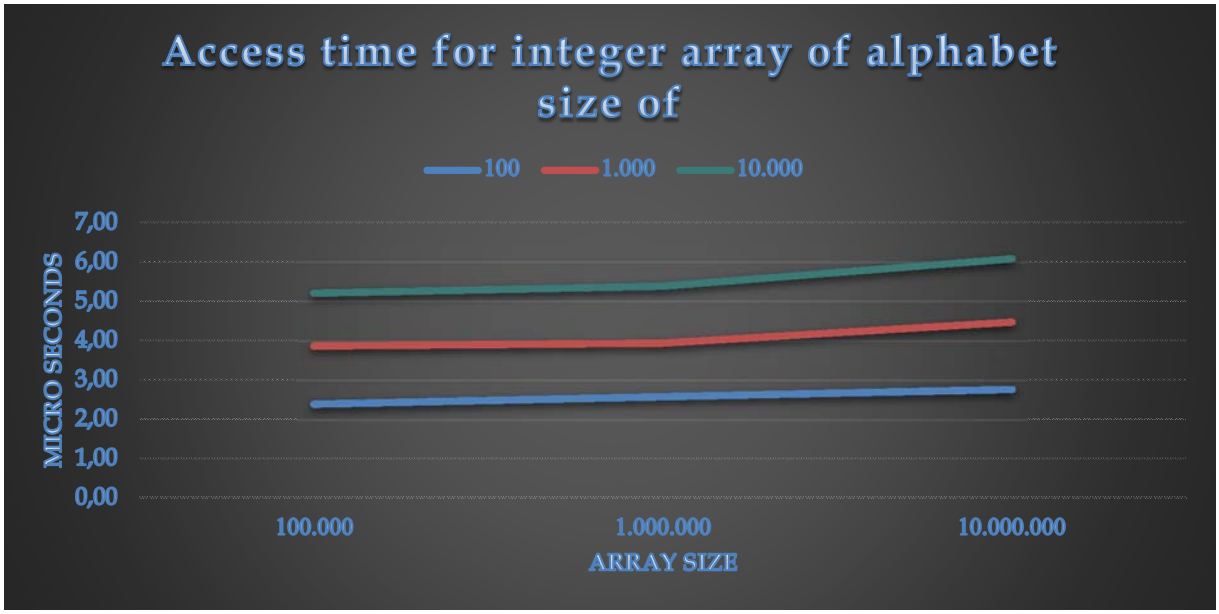
Γράφημα 3.1: Χρόνος δημιουργίας του wavelet tree (σε δευτερόλεπτα) για ακολουθίες 100 χιλ.., 1.000 χιλ. και 10.000 χιλ. ακεραίων και μεγέθη αλφαβήτου 100, 1.000 και 10.000 συμβόλων.

Στο γράφημα 3.2, φαίνεται ότι ο ρυθμός συμπίεσης του wavelet tree ως προς την αρχική ακολουθία, είναι απόλυτη συνάρτηση του μεγέθους του αλφαβήτου και δεν επηρεάζεται διόλου από το μέγεθος της ακολουθίας (ευθείες γραμμές), αφού το wavelet tree απαιτεί το πολύ $\mu\lceil\log_2 \sigma\rceil$ bit για την αναπαράσταση της αρχικής ακολουθίας. Όσο αυξάνεται το μέγεθος του αλφαβήτου, μειώνεται ο ρυθμός συμπίεσης, με λογαριθμικό όμως ρυθμό, όπως φαίνεται από την παραπάνω σχέση.

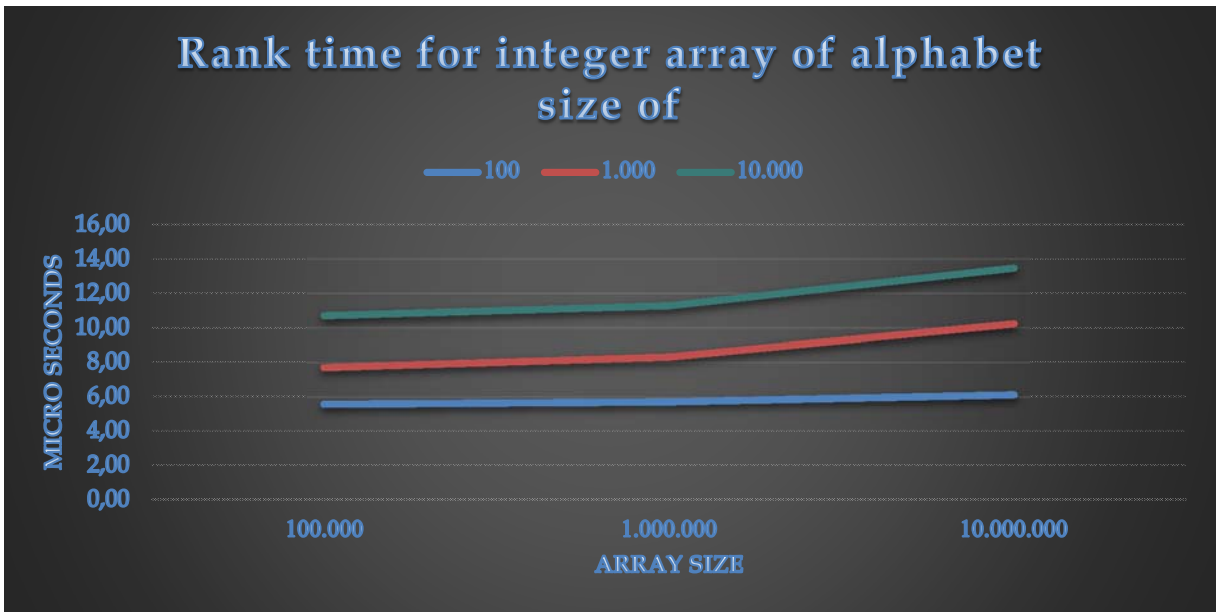


Γράφημα 3.2: Ρυθμός συμπίεσης του wavelet tree για μεγέθη αλφαβήτου 100, 1.000 και 10.000 συμβόλων, επί ακολουθιών 100 χιλ.., 1.000 χιλ. και 10.000 χιλ. ακεραίων.

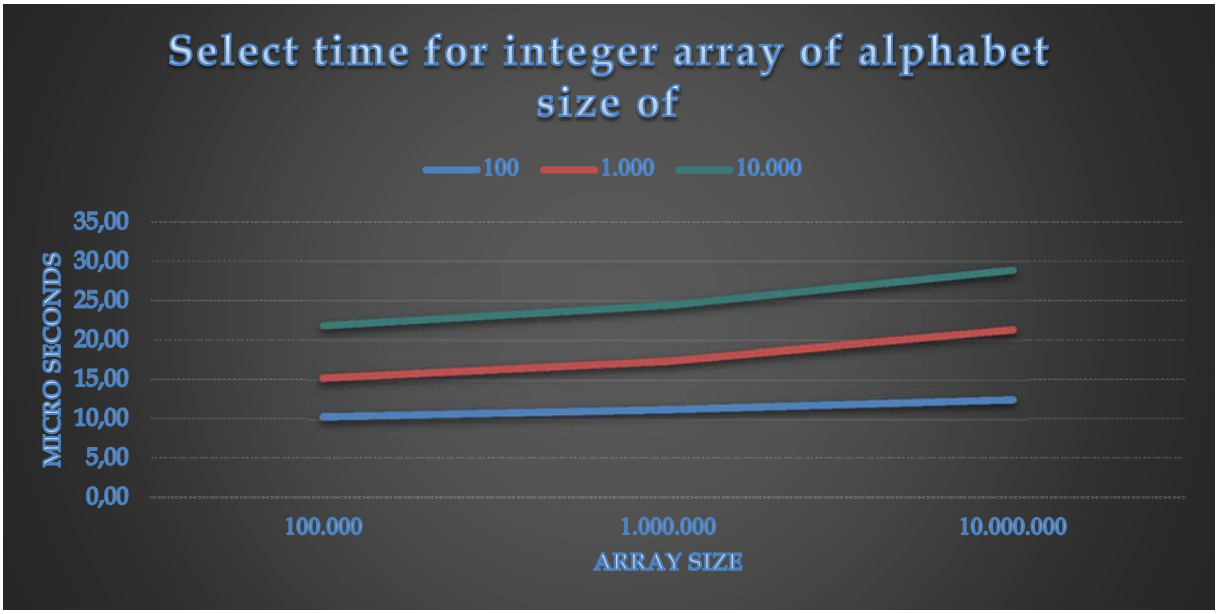
Στα τρία γραφήματα που ακολουθούν παρουσιάζεται η ταχύτητα, ανάκτησης συμβόλων, καταμέτρησης πλήθους επαναλήψεων συμβόλου και εύρεσης της νιοστής επανάληψής συμβόλου. Και οι τρεις παραπάνω λειτουργίες απαιτούν χρόνο $O(\log_2 \sigma)$. Συνεπεία συναρτώνται κατά κύριο λόγο, του μεγέθους του αλφαβήτου. Η μικρή εξάρτηση που φαίνεται να υπάρχει από το μέγεθος της ακολουθίας οφείλεται, στους αλγορίθμους υλοποίησης των δυαδικών διανυσμάτων.



Γράφημα 3.3: Χρόνος ανάκτησης συμβόλου, για μεγέθη αλφαβήτου 100, 1.000 και 10.000 συμβόλων, επί ακολουθιών 100 χιλ.., 1.000 χιλ. και 10.000 χιλ. ακεραίων (πραγματοποιήθηκαν 50 χιλ. ανακτήσεις τυχαίων συμβόλων και παρουσιάζεται ο μέσος χρόνος ανάκτησης συμβόλου σε εκατομμυριοστά του δευτερολέπτου).



Γράφημα 3.4: Χρόνος καταμέτρησης πλήθους επαναλήψεων συμβόλου, για μεγέθη αλφαβήτου 100, 1.000 και 10.000 συμβόλων, επί ακολουθιών 100 χιλ.., 1.000 χιλ. και 10.000 χιλ. ακεραίων (πραγματοποιήθηκαν 50 χιλ. καταμετρήσεις επαναλήψεων τυχαίων συμβόλων και παρουσιάζεται ο μέσος χρόνος καταμέτρησης επαναλήψεων συμβόλου σε εκατομμυριοστά του δευτερολέπτου).



Γράφημα 3.5: Χρόνος εύρεσης της νιοστής επανάληψής συμβόλου, για μεγέθη αλφαβήτου 100, 1.000 και 10.000 συμβόλων επί, ακολουθιών 100 χιλ.., 1.000 χιλ. και 10.000 χιλ. ακεραίων (πραγματοποιήθηκαν 50 χιλ. αναζητήσεις νιοστής επανάληψης τυχαίων συμβόλων και παρουσιάζεται ο μέσος χρόνος εύρεσης της νιοστής επανάληψης συμβόλου σε εκατομμυριοστά του δευτερολέπτου).

Οι επιδόσεις της υλοποίησης συγκρίθηκαν με την βιβλιοθήκη wat-array 0.0.6, που ανακτήθηκε από την ιστοσελίδα https://code.google.com/p/wat-array/downloads/list η οποία προσφέρει παρόμοιες λειτουργικότητες. Βρέθηκαν παρόμοιες επιδόσεις για κατασκευή wavelet tree, ρυθμό συμπίεσης, ταχύτητα ανάκτησης συμβόλων, καταμέτρηση πλήθους επαναλήψεων συμβόλου και εύρεση της νιοστής επανάληψής συμβόλου.

3.2 Ο μηχανισμός συμπίεσης

Η υλοποίηση του μηχανισμού συμπίεσης βασίστηκε στην απλή ιδέα της κατάτμησης της αρχικής ακολουθίας σε λέξεις οι οποίες ξεκινούν με ένα οποιοδήποτε σύμβολο πλην των κενών συμβόλων (tab, newline, vertical tab, form feed, carriage return, και space) και τελειώνουν με το επόμενο κενό σύμβολο που ακολουθείται από μη κενό σύμβολο ή το τέλος της ακολουθίας.

Για παράδειγμα η ακολουθία “I'll be delighted to explain (the simplest way I can) to you how this idea works!!!” θα κατατμηθεί στις λέξεις:

1. “I'll ”
2. “be ”

3. “delighted”
4. “to”
5. “explain”
6. “(the”
7. “simplest”
8. “way”
9. “I”
10. “can)”
11. “to”
12. “you”
13. “how”
14. “this”
15. “idea”
16. “works!!!”

Με την κατάτμηση αυτή καμία λέξη δεν μπορεί να είναι μικρότερη από δύο χαρακτήρες (αφού κάθε χαρακτήρας θα ακολουθείται από ένα χαρακτήρα κενού ή το τέλος της ακολουθίας) και επιπλέον το μέγεθος κάθε φυσικής λέξης αυξάνεται τουλάχιστον κατά ένα χαρακτήρα αφού συμπεριλαμβάνει το κενό που την ακολουθεί ενώ σημεία στίξης και οποιοιδήποτε άλλοι χαρακτήρες ενώνονται με τη φυσική λέξη που προηγείται ή ακολουθεί.

Στη συνέχεια οι λέξεις ταξινομούνται λεξικογραφικά και αντιστοιχίζονται στον αύξοντα φυσικό αριθμό που τους αναλογεί βάση της ταξινόμησης (αντικατάσταση προτύπων).

Αν υποθέσουμε πως έχουμε ένα κείμενο στην αγγλική γλώσσα όπου ο κάθε χαρακτήρας κωδικοποιείται κατά το πρότυπο ASCII, δηλαδή με 7 bit, και το σύνολο των λέξεων (αλφάβητο) είναι 100.000 (απαιτούνται $\log_2 100.000 = 17\ bit$ για την αποθήκευση στο wavelet tree), τότε για κάθε λέξη μεγαλύτερη των δύο χαρακτήρων επιτυγχάνεται συμπίεση. Για λέξεις δύο (2) χαρακτήρων (2 * 7 bit = 14 bit) απαιτούνται 17 bit για την αποθήκευση τους στο wavelet tree. Για λέξεις τριών (3) χαρακτήρων (3 * 7 bit = 21 bit) απαιτούνται πάλι17 bit για την αποθήκευση τους (συμπίεση 17 / 21 = 80,9%). Για λέξεις τεσσάρων (4) χαρακτήρων (4 * 7 bit = 28 bit) απαιτούνται

πάλι17 bit για την αποθήκευση τους (συμπίεση 17 / 28 = 60,7%). Για λέξεις πέντε (5) χαρακτήρων (5 * 7 bit = 35 bit) απαιτούνται πάλι 17 bit για την αποθήκευση τους (συμπίεση 17 / 35 = 48,6%). Για λέξεις δέκα (10) χαρακτήρων (10 * 7 bit = 70 bit) απαιτούνται πάλι17 bit για την αποθήκευση τους (συμπίεση 17 / 70 = 24,3%) και ούτω καθ' εξής.

Γίνεται φανερό ότι ο μηχανισμός επιτυγχάνει υψηλούς ρυθμούς συμπίεσης για το συντριπτικά μεγαλύτερο μέρος των λέξεων (πλην αυτών που απαρτίζονται από δύο χαρακτήρες, δηλαδή έναν χαρακτήρα πλέον το κενό που ακολουθεί).

Το μειονέκτημα της κατάτμησης είναι πως η ίδια φυσική λέξη θα κωδικοποιείται ως διαφορετική εφόσον ακολουθείται από ένα σημείο στίξης (για παράδειγμα οι ακολουθίες "name" και "name?" θα λογίζονται ως διαφορετικές λέξεις) ή εφόσον προηγείται ή ακολουθεί αυτήν οποιοσδήποτε άλλος χαρακτήρας. Αυτό αυξάνει σημαντικά τον αριθμό των λέξεων, ήτοι το αλφάβητο της ακολουθίας. Επιπλέον οι ταξινομημένες λέξεις πρέπει να αποθηκεύονται, ώστε να μπορούν να ανακτηθούν όταν αυτό απαιτείται.

Παρά τα παραπάνω μειονεκτήματα οι ρυθμοί συμπίεσης που επιτυγχάνονται είναι υψηλοί και ανταγωνιστικοί άλλων υλοποιήσεων (όπως θα δούμε στο επόμενο κεφάλαιο).

Επιπλέον η χρήση ως συμβόλων του αλφαβήτου, λέξεων αντί για χαρακτήρες καθιστά την ανάκτηση, αναζήτηση, καταμέτρηση επαναλήψεων δείγματος και εύρεση νιοστής επανάληψης δείγματος, ευκολότερη κα ταχύτερη για προφανείς λόγους.

Για την ταξινόμηση των λέξεων πριν την αποθήκευσή τους στο wavelet tree χρησιμοποιήθηκε ένα απλό δυαδικό δένδρο αναζήτησης, το οποίο διατρέχεται δις προ της δημιουργίας του wavelet tree. Μία για την λεξικογραφική ταξινόμηση των λέξεων και μία δεύτερη για την αντιστοίχιση των λέξεων της αρχικής ακολουθίας με φυσικούς ακεραίους.

Απ' όσο γνωρίζουμε η μεθοδολογία κατάτμησης ακολουθίας, που παρουσιάσαμε ανωτέρω, προκειμένου να παραχθεί εξ αυτής αυτό-δεικτοδοτούμενη δομή με σκοπό αφ' ενός την συμπίεση της αρχικής ακολουθίας και αφετέρου της επιτάχυνση αναζητήσεων επ' αυτής, δεν έχει παρουσιαστεί στο παρελθόν.

Η μεθοδολογία κρίνεται κατάλληλη για την συμπίεση ακολουθιών κειμένου σε φυσικές γλώσσες και οποιαδήποτε ακολουθία έχει χαρακτηριστικά παρόμοια φυσικής γλώσσας όπως είναι γλώσσες προγραμματισμού ηλεκτρονικών υπολογιστών και γλώσσες σήμανσης (XML, HTML). Δεν είναι κατάλληλη για βιολογικές σειρές (DNA, proteins), λόγω του πολύ μικρού αλφάβητου και κυρίως της περιορισμένης εμφάνισης κενών στις ακολουθίες.

Κεφάλαιο 4
Συγκριτική δοκιμή υλοποιήσεων

Πέραν της υλοποίησης που αναπτύχθηκε, με σκοπό την κατανόηση της λειτουργίας του wavelet tree, επελέγησαν τρεις διαφορετικές υλοποιήσεις τρίτων, από ελεύθερες πηγές, προκειμένου να συγκριθούν με αυτήν, ως προς την λειτουργικότητα και την απόδοση.

4.1 Ταυτότητα της συγκριτικής δοκιμής

4.1.1 Επιλεγείσες υλοποιήσεις

Σε κάθε μία εκ των επιλεγέντων υλοποιήσεων τρίτων η δομή wavelet tree χρησιμοποιείται ως μέρος ενός αλγορίθμου συμπίεσης σε συνδυασμό με κάποια άλλη τεχνική. Συγκεκριμένα επελέγησαν οι κάτωθι:

- **Succinct Suffix Array.**

  Το Succinct Suffix Array δημιουργήθηκε από τους Veli Mäkinen και Rodrigo González και ανακτήθηκε από την ιστοσελίδα Pizza&Chili Corpus (http://pizzachili.di.unipi.it/index.html). Πρόκειται για την 6η έκδοση της υλοποίησης που βασίστηκε στην εργασία των δημιουργών "New search algorithms and time/space tradeoffs for succinct suffix arrays" [19]. Πρόκειται για έναν FM-index που κωδικοποιεί κατά Huffman την ακολουθία και αποθηκεύεται στο wavelet tree, που υλοποιείται με χρήση ασυμπίεστων δυαδικών διανυσμάτων. Η ανάπτυξή του έχει πραγματοποιηθεί σε C++.

- **Alphabet Friendly FM-index.**

  Το Alphabet Friendly FM-index δημιουργήθηκε από τον Rodrigo González και ανακτήθηκε από την ιστοσελίδα Pizza&Chili Corpus (http://pizzachili.di.unipi.it/index.html). Πρόκειται για την 3η έκδοση της υλοποίησης που βασίστηκε στην εργασία των P. Ferragina, G. Manzini, V. Mäakinen και G. Navarro "An Alphabet-Friendly FM-index" [12] καθώς και την σχετική εργασία των ιδίων "Compressed Representations of Sequences and Full-Text Indexes" [13]. Πρόκειται για έναν FM-index που δημιουργείται κατακερματίζοντας το κείμενο με σκοπό την εφαρμογή της μετατροπής Burrows-Wheeler [02] που εν συνεχεία αποθηκεύεται στο wavelet tree, που υλοποιείται με χρήση συμπιεσμένων δυαδικών διανυσμάτων RRR [24]. Ο κατακερματισμός του κειμένου πραγματοποιείται με τρόπο τέτοιο, ώστε να μεγιστοποιείται η συμπίεση του υποκειμένου. Η ανάπτυξή του έχει πραγματοποιηθεί σε C++.

- **FM-index του Matthias Petri.**

  Το FM-index δημιουργήθηκε από τον Matthias Petri και ανακτήθηκε από την ιστοσελίδα GitHub (https://github.com/mpetri/FM-Index). Πρόκειται για έναν FM-index που δημιουργείται κατακερματίζοντας το κείμενο με σκοπό την εφαρμογή της μετατροπής Burrows-Wheeler [02] που εν συνεχεία αποθηκεύεται στο wavelet tree, που υλοποιείται με χρήση συμπιεσμένων δυαδικών διανυσμάτων RRR [24]. Ο κατακερματισμός του κειμένου πραγματοποιείται με τρόπο τέτοιο, ώστε να μεγιστοποιείται η συμπίεση του υποκειμένου. Η ανάπτυξή του έχει πραγματοποιηθεί σε C++.

Οι ως άνω υλοποιήσεις επελέγησαν με κριτήριο την απρόσκοπτη λειτουργία τους καθώς και την αποδοτικότητά τους με βάση την συγκριτική μελέτη "Compressed text indexes: From theory to practice" των Paolo Ferragina , Rodrigo González , Gonzalo Navarro και Rossano Venturini [09], στην οποία συγκρίνονται μεταξύ άλλων οι δύο πρώτες εκ των τριών, οι οποίες επιτυγχάνουν τις καλύτερες αποδόσεις στις επιμέρους μετρήσεις της εν λόγω συγκριτικής μελέτης.

4.1.2 Υποκείμενο των δοκιμών

Επελέγησαν τρεις διαφορετικοί τύποι δεδομένων, από την συλλογή δεδομένων της ιστοσελίδας Pizza&Chili Corpus (http://pizzachili.di.unipi.it/texts.html). Η επιλογή έγινε βάσει δύο κριτηρίων:

1. Πραγματικά δεδομένα διαφόρων ειδών
2. Δεδομένα μεγάλου όγκου.

Συγκεκριμένα επελέγησαν:

- Αγγλικά κείμενα. Περιλαμβάνει μια συλλογή από κείμενα στην αγγλική γλώσσα επιλεγμένα από τις συλλογές etext02-etext05, διαθέσιμες στην ιστοσελίδα "Gutenberg Project" (https://www.gutenberg.org/). Έχουν αφαιρεθεί κεφαλίδες της ιστοσελίδας ώστε να παραμείνει μόνον πραγματικό κείμενο.
- Κώδικας (source program code). Περιλαμβάνει κώδικα σε C/C++/Java, που προήλθε από την συνένωση όλων των αρχείων .c, .h, .C και .java των διανομών linux-2.6.11.6 (ftp.kernel.org) και gcc-4.0.0 (ftp.gnu.org).
- XML (structured text). Περιλαμβάνει, σε XML μορφοποίηση, βιβλιογραφικές αναφορές από επιστημονικά περιοδικά πληροφορικής καθώς και συνέδρια πληροφορικής. Ανακτήθηκε από DBLP archive (dblp.uni-trier.de).

Σημειώνεται ότι δεν έγιναν μετρήσεις επί αρχείων βιολογικών σειρών (DNA, proteins), όπως συνηθίζεται σε αντίστοιχες έρευνες, λόγω της φύσης της υλοποίησης wavelet tree, που λειτουργεί αποδοτικά σε μεγαλύτερα αλφάβητα από αυτά των βιολογικών σειρών καθώς και ακολουθίες που περιέχουν σημαντικό πλήθος κενών μεταξύ των συμβόλων, όπως είναι κείμενο, κώδικας και XML.

Πραγματοποιήθηκαν μετρήσεις επί των παραπάνω τύπων αρχείων για μεγέθη 50MB, 100MB και 200MB που διατίθενται στην ιστοσελίδα Pizza&Chili Corpus. Επειδή δεν παρατηρήθηκαν αποκλείσεις στην απόδοση λόγω του μεγέθους των αρχείων, παρουσιάζονται μόνον οι μετρήσεις που αφορούν τα αρχεία μεγέθους 200MB.

Στον πίνακα που ακολουθεί παρουσιάζονται το μέγεθος του πραγματικού αλφάβητου καθώς η εντροπία καθενός από τα ανωτέρω αρχεία.

Αρχείο	**Μέγεθος πραγματικού αλφάβητου**	**Εντροπία (bps)**
Αγγλικό κείμενο	90	4.60
Κώδικας	230	5.47
XML	96	5.26

Πίνακας 4.1: Βασικά δεδομένα υποκείμενων αρχείων.

4.1.3 Μετρήσεις

Πραγματοποιήθηκαν έξι διαφορετικές μετρήσεις:

- Απαιτούμενος χρόνος για την δημιουργία του wavelet tree (build time). Ο χρόνος που απαιτείται, σε δευτερόλεπτα, για την δημιουργία του wavelet tree, από την αρχική ακολουθία. Στις εν λόγω μετρήσεις περιλήφθηκαν και οι επιδόσεις των gzip και bzip2, προκειμένου να τεθεί ορθότερα το πλαίσιο της συγκριτικής δοκιμής και να τονιστεί η διαφοροποίηση των εν λόγω αλγορίθμων από τις εφαρμογές συμπίεσης - αποσυμπίεσης.

- Μέγεθος του wavelet tree (index size). Το μέγεθος του wavelet tree, σε απόλυτο μέγεθος, σε Mbytes, καθώς και ως ποσοστό του μεγέθους της αρχικής ακολουθίας (compress ratio). Στις εν λόγω μετρήσεις περιλήφθηκαν επίσης και οι επιδόσεις των gzip και bzip2.

- Χρόνος ανάκτησης του wavelet tree από αποθηκευτικό μέσο (load time). Ο χρόνος που απαιτείται, σε δευτερόλεπτα, για την ανάκτηση του wavelet tree, αφού αυτό έχει αποθηκευτεί ως αρχείο, σε αποθηκευτικό μέσο (σκληρό δίσκο).

- Χρόνος ανάκτησης συμβόλων (access time). Ο χρόνος απλής ανάκτησης συμβόλων της ακολουθίας, σε δευτερόλεπτα.

- Χρόνος καταμέτρησης επαναλήψεων δείγματος (count occurrences time). Ο χρόνος καταμέτρησης επαναλήψεων, σειράς συμβόλων του αλφαβήτου (δείγματος), στην ακολουθία, σε χιλιοστά του δευτερολέπτου (milliseconds) ανά δείγμα.

- Χρόνος εύρεση της θέσης στην αρχική ακολουθία της νιοστής επανάληψής ενός δείγματος (locate occurrence time). Ο χρόνος εύρεσης της θέσης στην αρχική ακολουθία της νιοστής επανάληψής, σειράς συμβόλων του αλφαβήτου (δείγματος), σε χιλιοστά του δευτερολέπτου (milliseconds) ανά επανάληψη.

4.2 Παρουσίαση αποτελεσμάτων

Παρακάτω παρουσιάζονται τα αποτελέσματα των δοκιμών.

Όλες οι δοκιμές πραγματοποιήθηκαν σε σύστημα με επεξεργαστή Intel core i5-2500 3.3GHz, 8GB RAM και λειτουργικό σύστημα Linux Mint 17.2 - 64bit. Οι μεταγλωττίσεις (compilation) όλων των υλοποιήσεων τρίτων έχουν πραγματοποιηθεί με τον g++ 4.9.3.

Σε όλους του πίνακες αποτελεσμάτων που ακολουθούν, εμφανίζονται με έντονα στοιχεία οι βέλτιστες αποδόσεις σε κάθε κατηγορία που παρουσιάζεται (μεταξύ των συγκρινόμενων υλοποιήσεων – δεν υπολογίζονται οι επιδόσεις των gzip και bzip2 στους πίνακες που αυτές συμπεριλαμβάνονται).

Ως wt, αναφέρεται στους πίνακες η υλοποίηση που αναπτύχθηκε στα πλαίσια της παρούσης διατριβής.

4.2.1 Χρόνος δημιουργίας του wavelet tree (build time)

Ο χρόνος που απαιτείται, για την δημιουργία του wavelet tree, από τις αρχικές ακολουθίες, ήτοι αγγλικό κείμενο, κώδικα γλωσσών προγραμματισμού και γλώσσας σήμανσης (XML), ονομαστικού μεγέθους εκάστου 200MB (πραγματικού 209,7MB), εμφανίζεται, για κάθε μία από τις υλοποιήσεις, στον παρακάτω πίνακα:

Index	Build time in seconds		
	English text 200MB (209,7)	**Sources** 200MB (209,7)	**xml** 200MB (209,7)
wt	70,290	53,825	**33,982**
Succinct Suffix Array (Veli Mäkinen, R. González)	**57,210**	**41,552**	41,082
Alphabet Friendly FM-index (Rodrigo González)	214,781	193,205	127,349
FM-index (Matthias Petri)	128,974	123,715	111,064
gzip (compression + decompression time)	≅13	≅7	≅5
bzip2 (compression + decompression time)	≅ 24	≅ 21	≅ 26

Πίνακας 4.2: Χρόνος δημιουργίας του wavelet tree (build time) σε δευτερόλεπτα.

Το wt επιτυγχάνει ανταγωνιστικούς χρόνους δημιουργίας της δομής, παρά την χρήση (δις) απλού δυαδικού δένδρου αναζήτησης στον αλγόριθμο συμπίεσης (βλέπε 3.2). Υπολείπεται μόνον του Succinct Suffix Array για το αγγλικό κείμενο και τον κώδικα γλωσσών προγραμματισμού, λόγω της απλότητας της συγκεκριμένης υλοποίησης, ενώ προπορεύεται στην γλώσσα σήμανσης (XML), λόγω του μηχανισμού συμπίεσης που ευνοείται ιδιαίτερα από τη δομή των εν λόγω αρχείων (συχνές επαναλήψεις μεγάλων ακολουθιών που χωρίζονται με κενά).

Ο υπέρ-τριπλάσιος χρόνος που απαιτεί ο Alphabet Friendly FM-index για την δημιουργία του wavelet tree, σε όλους τους τύπους δεδομένων, είναι το κόστος της χρήσης συμπιεσμένων δυαδικών διανυσμάτων. Ο FM-index του Matthias Petri που επίσης χρησιμοποιεί συμπιεσμένα δυαδικά διανύσματα, απαιτεί μεν περισσότερο χρόνο, αλλά όχι όσο ο Alphabet Friendly FM-index, λόγω του ότι δεν κατακερματίζει την ακολουθία.

Σημειώνεται, ότι ο χρόνος που απαιτείται συνολικά για την συμπίεση και την αποσυμπίεση των ίδιων αρχείων στο ίδιο σύστημα, σε gzip μορφή αλλά και για την συμπίεση από το bzip2, είναι σημαντικά μικρότερος, σε σχέση με αυτόν που απαιτείται από όλες τις εξεταζόμενες υλοποιήσεις, όπως φαίνεται στον πίνακα 4.2. Λαμβάνεται υπόψη ο συνολικός χρόνος συμπίεσης και αποσυμπίεσης, αφού για κάθε επεξεργασία των δεδομένων που έχουν συμπιεστεί σε gzip μορφή ή από το bzip2, απαιτείται προηγουμένως η αποσυμπίεση τους. Οι σαφώς ανώτερες επιδόσεις, οφείλονται

στην ανυπαρξία δεικτών επί των δεδομένων σε αυτές τις δομές. Συνεπεία, η σύγκριση μεταξύ αυτών των δομών και των υλοποιήσεων που εξετάζονται εδώ, δεν είναι δυνατή.

4.2.2 Μέγεθος του wavelet tree (index size)

Το μέγεθος του wavelet tree, σε απόλυτο μέγεθος, σε Mbytes, καθώς και ως ποσοστό του μεγέθους της αρχικής ακολουθίας (compress ratio), ήτοι αγγλικό κείμενο, κώδικα γλωσσών προγραμματισμού και γλώσσας σήμανσης (XML), ονομαστικού μεγέθους εκάστου 200MB (πραγματικού 209,7MB), εμφανίζεται, για κάθε μία από τις υλοποιήσεις, στον παρακάτω πίνακα:

Index	**Index space in Mbytes**		
	English text 200MB (209,7)	**Sources** 200MB (209,7)	**xml** 200MB (209,7)
wt	118,5 (56,5%)	136,5 (65,1%)	102,8 (49,0%)
Succinct Suffix Array (Veli Mäkinen, R. González)	125,8 (60,0%)	151,0 (72,0%)	144,6 (69,0%)
Alphabet Friendly FM-index (Rodrigo González)	88,1 **(42,0%)**	102,8 **(49,0%)**	71,4 **(34,0%)**
FM-index (Matthias Petri)	129,5 (61,8%)	122,5 (58,4%)	103,6 (49,4%)
gzip	77,5 (37,0%)	47,3 (22,6%)	36,7 (17,5%)
bzip2	57,5 (27,4%)	39,1 (18,6%)	23,8 (11,3%)

Πίνακας 4.3: Μέγεθος του wavelet tree (index size) σε Mbytes και ως ποσοστό του μεγέθους της αρχικής ακολουθίας (compress ratio) .

Η υπεροχή του Alphabet Friendly FM-index είναι εδώ αισθητή, λόγω των συμπιεσμένων δυαδικών διανυσμάτων RRR [24] που χρησιμοποιούνται στους κόμβους του wavelet tree, εις βάρος, όπως προαναφέρθηκε, του χρόνου δημιουργίας του wavelet tree.

Ακολουθεί το wt, το οποίο πετυχαίνει τον 2ο υψηλότερο ρυθμό συμπίεσης για αγγλικό κείμενο και γλώσσας σήμανσης (XML), χάρη στον μηχανισμό συμπίεσης που χρησιμοποιεί και παρότι δεν εφαρμόζει καμία άλλη τεχνική συμπίεσης όπως κωδικοποίηση Huffman [06] ή μετατροπή Burrows-Wheeler [02].

Ο FM-index του Matthias Petri που επίσης χρησιμοποιεί συμπιεσμένα δυαδικά διανύσματα δεν επιτυγχάνει ικανοποιητικά αποτελέσματα, εκτός από την ακολουθία γλώσσας σήμανσης (XML), λόγω του αλγορίθμου συμπίεσης που υιοθετεί.

Σημειώνεται, ότι και εδώ η συμπίεση των ίδιων αρχείων στο ίδιο σύστημα, σε gzip μορφή αλλά και από το bzip2, επιτυγχάνει καλύτερες επιδόσεις, σε σχέση με τις εξεταζόμενες υλοποιήσεις, όπως φαίνεται στον πίνακα 4.3. Οι ανώτερες επιδόσεις, οφείλονται, και πάλι, στην ανυπαρξία δεικτών επί των δεδομένων σε αυτές τις δομές. Συνεπεία, η σύγκριση μεταξύ αυτών των δομών και των υλοποιήσεων που εξετάζονται εδώ, δεν είναι δυνατή, αφού οι δομές αυτές δεν προσφέρουν καμία επιπλέον λειτουργικότητα επί των δεδομένων που συμπιέζουν (όπως αυτές που μετρούνται παρακάτω) και απαιτούν την αποσυμπίεση των δεδομένων για την οποιαδήποτε επεξεργασία τους ή αναζήτηση επί αυτών.

4.2.3 Χρόνος ανάκτησης του wavelet tree από αποθηκευτικό μέσο (load time)

Ο χρόνος που απαιτείται, για την ανάκτηση του wavelet tree, αφού αυτό έχει αποθηκευτεί ως αρχείο, σε αποθηκευτικό μέσο (σκληρό δίσκο) και για τις τρεις (3) κατηγορίες αρχικής ακολουθίας, παρουσιάζεται στον παρακάτω πίνακα:

Index	**Load time in seconds**		
	English text 200MB (209,7)	**Sources** 200MB (209,7)	**xml** 200MB (209,7)
wt	0,17	0,63	0,71
Succinct Suffix Array (Veli Mäkinen, R. González)	**0,13**	**0,08**	**0,08**
Alphabet Friendly FM-index (Rodrigo González)	0,16	0,21	0,19
FM-index (Matthias Petri)	3,97	3,91	3,61

Πίνακας 4.4: Χρόνος ανάκτησης του wavelet tree από αποθηκευτικό μέσο (load time) σε δευτερόλεπτα.

Ενώ όλες οι υλοποιήσεις τρίτων παρουσιάζουν μικρή απόκλιση στους χρόνους ανάκτησης για κάθε κατηγορία ακολουθίας, το wt παρουσιάζει σημαντική απόκλιση, στις ακολουθίες κώδικα

γλωσσών προγραμματισμού και γλώσσας σήμανσης (XML), γεγονός που οφείλεται στον συνδυασμό του μεγέθους του αλφαβήτου και του μέσου μήκους των συμβόλων του αλφαβήτου, δεδομένου του ότι αυτά αποθηκεύονται στη δομή και ανακτώνται πριν την ανάκτηση του wavelet tree.

Για τον κώδικα γλωσσών προγραμματισμού, το αλφάβητο είναι περίπου τριπλάσιο αυτού του αγγλικού κειμένου με περίπου ίδιο μέσο μήκος συμβόλων. Για την δε γλώσσα σήμανσης (XML), το αλφάβητο είναι περίπου κατά 50% μεγαλύτερο αυτού του αγγλικού κειμένου με πολύ μεγαλύτερο όμως μέσο μήκος συμβόλων.

Ο FM-index του Matthias Petri αποκλίνει σημαντικά στο χρόνο ανάκτησης, λόγω της χρήσης συμπιεσμένων δυαδικών διανυσμάτων και του τρόπου που τα υλοποιεί.

4.2.4 Χρόνος ανάκτησης συμβόλων (access time)

Ο χρόνος απλής ανάκτησης συμβόλων και για τις τρεις (3) κατηγορίες αρχικής ακολουθίας, παρουσιάζεται στον παρακάτω πίνακα:

Index	**Access time for 10.000 characters in seconds**		
	English text 200MB (209,7)	**Sources** 200MB (209,7)	**xml** 200MB (209,7)
wt	**0,018**	**0,017**	**0,007**
Succinct Suffix Array (Veli Mäkinen, R. González)	0,022	0,022	0,022
Alphabet Friendly FM-index (Rodrigo González)	0,022	0,022	0,014
FM-index (Matthias Petri)	0,159	0,158	0,160

Πίνακας 4.5: Χρόνος ανάκτησης συμβόλων (access time), 10.000 χαρακτήρων της αρχικής ακολουθίας, σε δευτερόλεπτα.

Για τους ελέγχους έχει επιλεγεί η ανάκτηση δέκα χιλιάδων (10.000) συνεχόμενων χαρακτήρων από τυχαία σημεία των ακολουθιών.

Το wt προπορεύεται έναντι όλων των υπόλοιπων υλοποιήσεων, αφού λόγω του μηχανισμού συμπίεσης, ανακτά λέξη με κάθε διάσχιση του δένδρου όταν οι υπόλοιπες ανακτούν έναν χαρακτήρα

με κάθε διάσχιση, που επιβαρύνει σημαντικά το χρόνο ανάκτησης συμβόλων ακόμη κι αν λάβουμε υπόψη μας πως το ύψος του δένδρου είναι μικρότερο (λόγω μικρότερου αλφαβήτου).

Ειδικά δε σε ακολουθίες γλώσσας σήμανσης (XML) η απόδοση είναι πολύ υψηλή, λόγω του μεγαλύτερου μέσου μήκους συμβόλου (λέξης) που προκύπτει από την φύση της ακολουθίας.

Και εδώ ο FM-index του Matthias Petri αποκλίνει σημαντικά.

4.2.5 Χρόνος καταμέτρησης επαναλήψεων δείγματος (count occurrences time)

Ο χρόνος καταμέτρησης επαναλήψεων, σειράς συμβόλων του αλφαβήτου (δείγματος), και για τις τρεις (3) κατηγορίες αρχικής ακολουθίας, παρουσιάζεται, σε χιλιοστά του δευτερολέπτου ανά δείγμα, στον παρακάτω πίνακα:

Index	**Count occurrences of 1.000 patterns (20 characters wide) In milli seconds / pattern**		
	English text 200MB (209,7)	**Sources** 200MB (209,7)	**xml** 200MB (209,7)
wt	2,24	1,36	0,81
Succinct Suffix Array (Veli Mäkinen, R. González)	2,13	1,74	0,94
Alphabet Friendly FM-index (Rodrigo González)	**1,67**	**1,30**	**0,71**
FM-index (Matthias Petri)	2,28	2,56	2,67

Πίνακας 4.6: Χρόνος καταμέτρησης επαναλήψεων δείγματος (count occurrences time), 1.000 τυχαίων δειγμάτων εκ της αρχικής ακολουθίας έκαστο μεγέθους είκοσι (20) χαρακτήρων, σε χιλιοστά του δευτερολέπτου ανά δείγμα.

Για τους ελέγχους έχουν επιλεγεί από τις αρχικές ακολουθίες χίλια (1.000) δείγματα, μεγέθους είκοσι χαρακτήρων έκαστο.

Οι επιδόσεις του wt είναι και εδώ ικανοποιητικές, ειδικά στις ακολουθίες κώδικα γλωσσών προγραμματισμού και γλώσσας σήμανσης (XML), παρότι υπολείπονται του Alphabet Friendly FM-index.

Σημειώνεται ότι υπάρχουν περιθώρια βελτίωσης του αλγόριθμου αναζήτησης του δείγματος εντός του λεξικογραφικά ταξινομημένου αλφαβήτου, που θα αναπτυχθούν σε μια $2^{η}$ έκδοση του wt.

4.2.6 Χρόνος εύρεση της θέσης στην αρχική ακολουθία της νιοστής επανάληψής ενός δείγματος (locate occurrence time)

Ο χρόνος εύρεσης της θέσης στην αρχική ακολουθία της νιοστής επανάληψής, σειράς συμβόλων του αλφαβήτου (δείγματος), και για τις τρεις (3) κατηγορίες αρχικής ακολουθίας, παρουσιάζεται, σε χιλιοστά του δευτερολέπτου ανά επανάληψη, στον παρακάτω πίνακα:

Index	**Locate occurrences of 1.000 patterns (20 characters wide) In milli seconds / occurrence**		
	English text 200MB (209,7)	**Sources** 200MB (209,7)	**xml** 200MB (209,7)
wt	2,42	3,11	2,01
Succinct Suffix Array (Veli Mäkinen, R. González)	2,81	3,43	1,18
Alphabet Friendly FM-index (Rodrigo González)	**2,19**	**2,58**	**0,69**
FM-index (Matthias Petri)	2,52	3,25	2,13

Πίνακας 4.7: Χρόνος εύρεση της θέσης στην αρχική ακολουθία της νιοστής επανάληψής ενός δείγματος (locate occurrence time), 1.000 τυχαίων δειγμάτων εκ της αρχικής ακολουθίας έκαστο μεγέθους είκοσι (20) χαρακτήρων, σε χιλιοστά του δευτερολέπτου ανά επανάληψη.

Και εδώ η απόδοση του wt είναι ικανοποιητική, ειδικά στις ακολουθίες αγγλικού κειμένου και κώδικα γλωσσών προγραμματισμού, ενώ υπολείπεται σημαντικά στην ακολουθία γλώσσας σήμανσης (XML) από τον Succinct Suffix Array και πολύ περισσότερο από τον Alphabet Friendly FM-index, ο οποίος απεικονίζει εδώ όλη την δυναμική του FM-index (backward searching), σε συνδυασμό με την μεθοδολογία κατακερματισμού της ακολουθίας που υλοποιεί και που φαίνεται να ευνοείται ιδιαίτερα από αυτόν τον τύπο ακολουθίας.

Κεφάλαιο 5
Επίλογος

Το wavelet tree, από το 2003 που προτάθηκε για 1η φορά ως αυτό-δεικτοδοτούμενη δομή, έχει καθιερωθεί κυρίως ως μέρος αλγορίθμων (Huffman coding wavelet trees, μετατροπή Burrows-Wheeler, FM-index) αυτό-δεικτοδοτούμενων δομών, ενώ σπανίως εμφανίζεται ως αυτόνομη δομή. Οι υλοποιήσεις του δε, κρίνονται ώριμες, χωρίς σημαντικά περιθώρια βελτίωσης.

Στο πλαίσιο αυτό η υλοποίηση του wavelet tree που αναπτύχθηκε στα πλαίσια της παρούσης δεν υπολείπεται μεν των υφιστάμενων υλοποιήσεων αλλά και δεν προάγει περαιτέρω την δομή. Οι διαφοροποιήσεις που παρατηρούνται αφορούν την υλοποίηση των δυαδικών διανυσμάτων, υιοθετώντας είτε την συμπιεσμένη (με κόστος στον χρόνο δημιουργίας του wavelet tree) είτε την α-συμπίεστη (ταχύτερη δημιουργία του wavelet tree).

Περιθώρια βελτίωσης και έρευνας υφίστανται στον συνολικό αλγόριθμο συμπίεσης.

Πρέπει να αναφερθεί, ότι η απόδοση κάθε συνολικού αλγόριθμου συμπίεσης, τόσο ως προς τον ρυθμό συμπίεσης που επιτυγχάνει όσο και ως προς την ταχύτητα προσπέλασης των δεδομένων, είναι συνάρτηση των ιδιαίτερων χαρακτηριστικών της ακολουθίας στην οποία καλείται να εφαρμοστεί.

Στην παρούσα διατριβή παρουσιάστηκε, μια πρωτότυπη μεθοδολογία, συμπίεσης ακολουθιών συμβόλων φυσικών γλωσσών ή συμβόλων που προσομοιάζουν σε σημαντικό βαθμό φυσικές γλώσσες, η οποία βασίζεται στη δομή αυτών των ακολουθιών.

Η εν λόγω μεθοδολογία αναπτύχτηκε χρησιμοποιώντας καταρχήν για την λεξικογραφική ταξινόμηση των υποκειμένων, απλό δυαδικό δένδρο αναζήτησης, που ενδεχομένως δεν αποτελεί την βέλτιστη λύση λεξικογραφικής ταξινόμησης. Επιπλέον έχουν χρησιμοποιηθεί για την αναζήτηση δειγμάτων επί των ακολουθιών συναρτήσεις από την βασική βιβλιοθήκη της ANSI C, όπως η strstr(), που επίσης δεν αποτελούν τις βέλτιστες λύσεις για τους σκοπούς που χρησιμοποιήθηκαν. Σε μια δεύτερη έκδοση της υλοποίησης θα ενσωματωθούν προηγμένες μέθοδοι λεξικογραφικής ταξινόμησης αλλά και αναζήτησης δειγμάτων εντός ταξινομημένων ακολουθιών που θα συνάδουν με την μεθοδολογία υπό το πρίσμα της συνολικής βελτίωσης της απόδοσης της.

Τέλος, εκτιμάται πως είναι δυνατή η βελτίωση της μεθοδολογίας, τόσο ως προς τον ρυθμό συμπίεσης που επιτυγχάνει, όσο και ως προς την ταχύτητα αναζήτησης, εφόσον συνδυαστεί με άλλες μεθοδολογίες όπως η κωδικοποίηση Huffman [06], η μετατροπή Burrows-Wheeler [02] ή ο FM-index [10]. Στα πλαίσια μιας ενδεχόμενης μελλοντικής διδακτορικής διατριβής θα ερευνηθούν τα ανωτέρω παράλληλα με την βελτιστοποίηση των επιμέρους μεθόδων που χρησιμοποιήθηκαν, στην προτεινόμενη μεθοδολογία, πάντα σ' ένα πλαίσιο σύγκρισης με υλοποιήσεις δομών που επιδιώκουν τους αυτούς στόχους, της συμπίεσης μεγάλων ακολουθιών και της ταχύτερης δυνατής πρόσβασης σε αυτές χωρίς την ανάγκη προηγούμενης αποσυμπίεσης τους.

Βιβλιογραφία

[01] N. Brisaboa, Y. Cillero, A. Farina, S. Ladra, O. Pedreira. «A New Approach for Document Indexing using Wavelet Trees». DEXA Workshops, σ. 69-73, 2007.

[02] M. Burrows, D. Wheeler. «A block sorting lossless data compression algorithm, Technical Report 124», 1994, Digital Equipment Corporation.

[03] B. Chazelle. «A Functional Approach to Data Structures and Its Use in Multidimensional Searching». SIAM Journal on Computing 17 (3), σ. 427–462, 1988.

[04] D. Clark. «Compact Pat Trees». PhD thesis, University of Waterloo, 1996.

[05] F. Claude and G. Navarro. «Practical Rank/Select Queries over Arbitrary Sequences». In Proc. of the 15th International Conference on String Processing and Information Retrieval (SPIRE), σ. 176–187, 2008.

[06] D. Huffman. «A Method for the Construction of Minimum-Redundancy Codes». Proc. of the Institute of Radio Engineers (IRE), 40(9), σ. 1098–1101, 1952.

[07] P. Ferragina, R. Giancarlo, G. Manzini. «The Myriad Virtues of Wavelet Trees». ICALP, σ. 561-572, 2006.

[08] P. Ferragina, R. Giancarlo, G. Manzini, M. Sciortino: «Boosting textual compression in optimal linear time». Journal of the ACM, 52(4), σ. 688-713, 2005.

[09] P. Ferragina, R. González, G. Navarro, R. Venturini. «Compressed text indexes: From theory to practice». Journal of Experimental Algorithmics (JEA), 13, 2009.

[10] Ferragina, P. and Manzini, G. «An experimental study of an opportunistic index». In Proceedings 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). 269–278, 2001.

[11] P. Ferragina, G. Manzini. «Indexing Compressed Texts». Journal of the ACM 52 (4), σ. 552–581, 2005.

[12] P. Ferragina, G. Manzini, V. Mäakinen, G. Navarro. «An Alphabet-Friendly FM-Index». 11th International Symposium on String Processing and Information Retrieval (SPIRE), LNCS 3246, σ. 150–160, 2004.

[13] P. Ferragina, G. Manzini, V. Mäakinen, G. Navarro. «Compressed Representations of Sequences and Full-Text Indexes». ACM Transactions on Algorithms 3 (2) article 20, 2007.

[14] L. Foschini, R. Grossi, A. Gupta, J. Vitter. «When indexing equals compression: experiments with compressing suffix arrays and applications». ACM Transactions on Algorithms, 2(4), σ. 611-639, 2006.

[15] R. Grossi, A. Gupta, J. S. Vitter. «High-Order Entropy-Compressed Text Indexes». 14th Symposium on Discrete Algorithms (SODA), σ. 841–850, 2003.

[16] R. Grossi, A. Gupta, J. S. Vitter, B. Xu. «Wavelet Trees: From Theory to Practice». In Proc. First International Conference on Data Compression, Communications and Processing (CCP), σ. 210–221, 2011.

[17] G. Jacobsen. «Space-efficient static trees and graphs». In Proc. of the 30th Annual Symposium on Foundations of Computer Science (FOCS), σ. 549–554, 1989.

[18] V. Mäakinen, G. Navarro. «Implicit Compression Boosting with Applications to Self-Indexing». 14th International Symposium on String Processing and Information Retrieval (SPIRE), LNCS 4726, σ. 214–226, 2007.

[19] V. Mäakinen, G. Navarro. «New search algorithms and time/space tradeoffs for succinct suffix arrays». Technical Report C-2004-20 (April), University of Helsinki, Finland.

[20] V. Mäakinen, G. Navarro. «Succinct Suffix Arrays Based on Run-Length Encoding». Nordic Journal of Computing 12 (1), σ. 40–66, 2005.

[21] G. Navarro. «Indexing Text Using the Ziv-Lempel Trie». Journal of Discrete Algorithms 2 (1), σ. 87–114, 2004.

[22] G. Navarro, V. Mäakinen. «Compressed Full Text Indexes». ACM Computing Surveys 39 (1) article 2, 2007.

[23] R. Pagh. «Low Redundancy in Static Dictionaries with O(1) Worst Case Lookup Time». In Proc. of the 26th International Colloquium on Automata, Languages and Programming (ICALP), σ. 595–604, 1999.

[24] R. Raman, V. Raman, S. Rao. «Succinct Indexable Dictionaries with Applications to Encoding K-Ary Trees and Multisets». 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), σ. 233–242, 2002.

[25] C. E. Shannon, W. Weaver. «The Mathematical Theory of Communication», University of Illinois Press, 1949.

[26] T. Welch. «A technique for high-performance data compression». IEEE Computer 17 (6), σ. 8–19, 1984.

[27] J. Ziv, A. Lempel. «A universal algorithm for sequential data compression». IEEE Trans. Info Theory, vol. IT-23, σ. 337 - 343, 1977.