ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Τεχνικές κατασκευής δένδρων επιθεμάτων πολύ μεγάλου μεγέθους και χρήσης τους για γρήγορη αναζήτηση βιολογικών δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΠΟΛΥΧΡΟΝΟΠΟΥΛΟΥ ΒΑΣΙΛΕΙΟΥ

Επιβλέπων : Σελλής Τιμολέων
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβρης 2009

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Τεχνικές κατασκευής δένδρων επιθεμάτων πολύ μεγάλου μεγέθους και χρήσης τους για γρήγορη αναζήτηση βιολογικών δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΠΟΛΥΧΡΟΝΟΠΟΥΛΟΥ ΒΑΣΙΛΕΙΟΥ

Επιβλέπων : Σελλής Τιμολέων
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 26η Οκτωβρίου 2009.

.....	.....	.....
Σελλής Τιμολέων	Βασιλείου Ιωάννης	Κοζύρης Νεκτάριος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβρης 2009

.....

ΠΟΛΥΧΡΟΝΟΠΟΥΛΟΣ ΒΑΣΙΛΕΙΟΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τα δένδρα επιθεμάτων χρησιμοποιούνται ευρύτατα ως ευρετήρια για ακολουθίες βιολογικών δεδομένων. Τα τελευταία χρόνια παρατηρείται πολύ μεγάλη αύξηση του όγκου αυτού του τύπου δεδομένων λόγω της συνεχούς μείωσης του κόστους για ακολουθιοποίηση του γενετικού υλικού. Το γεγονός αυτό δημιουργεί την ανάγκη για αποδοτικούς τρόπους κατασκευής των δένδρων επιθεμάτων στον δίσκο, αφού τα ευρετήρια για ακολουθίες πολύ μεγάλου μεγέθους δεν χωρούν στην διαθέσιμη μνήμη. Στα πλαίσια της διπλωματικής (α) μελετήσαμε τους κυριότερους αλγορίθμους που έχουν προταθεί για την λύση του προβλήματος και (β) υλοποιήσαμε σειρά γνωστών αλγορίθμων που χρησιμοποιούν δένδρα επιθεμάτων πάνω σε μεγάλου μεγέθους δένδρα που κατασκευάστηκαν από αυτούς τους αλγορίθμους. Για το σκοπό αυτό χρησιμοποιήσαμε και επεκτείναμε την ανοιχτού κώδικα υλοποίηση του αλγορίθμου TRELLIS (ο οποίος είναι ο πιο γρήγορος από τους αλγορίθμους που έχουν προταθεί). Τα πειράματά μας αποκαλύπτουν τη συμπεριφορά των δένδρων μεγάλου μεγέθους όταν χρησιμοποιούνται από τους αλγορίθμους που υλοποιήσαμε.

Λέξεις Κλειδιά: << DNA, παραγωγή ευρετηρίων για ακολουθίες, βιολογικά δεδομένα, δένδρα επιθεμάτων, εξωτερική μνήμη, κατώτατος κοινός πρόγονος, υβριδικός δυναμικός προγραμματισμός >>

Abstract

The recent advance in DNA sequencing technologies has resulted in large DNA sequence databases. Suffix trees are widely used as indexes for answering sequence queries on large data sequences. There are efficient in-memory algorithms that construct suffix trees, but these algorithms do not scale up when the tree exceeds the available memory. Several disk-based suffix tree algorithms have been recently proposed. The aim of this diploma thesis is (i) to study the state-of-the-art algorithms that have been proposed so far to solve the problem, and (ii) to implement algorithms answering sequence queries using large suffix trees constructed by these methods. For this reason we use and extend the open source implementation of TRELLIS (which is the fastest among the proposed algorithms). We also provide extensive experimental evaluation to analyze the behavior of these algorithms.

Keywords: << DNA, sequence indexing, biological data, suffix trees, external memory, lowest common ancestor, hybrid dynamic programming >>

Πίνακας περιεχομένων

1 *Εισαγωγή*

1.1 Αντικείμενο της διπλωματικής

Η ανακάλυψη του DNA και του ρόλου του ως φορέα των γενετικών πληροφοριών, η κατανόηση των νόμων που διέπουν την γενετική έκφραση και οι μέθοδοι της γενετικής μηχανικής αποτέλεσαν κάποιες από τις μεγαλύτερες επιστημονικές τομές του 20^{ου} αιώνα. Η δυνατότητα της αναπαράστασης του γενετικού υλικού με τη μορφή ακολουθιακών δεδομένων και η ανάγκη αποδοτικής επεξεργασίας του για την εξαγωγή στατιστικών συμπερασμάτων και την αποσαφήνηση του ρόλου που παίζει στις λειτουργίες της ζωής, εγκαινίασαν τον κλάδο της Υπολογιστικής Βιολογίας.

Στις μέρες μας παρατηρείται μεγάλη αύξηση του μεγέθους των δεδομένων ακολουθιακού χαρακτήρα τα οποία πρέπει να επεξεργαστούν και να αναλυθούν. Η αύξηση αυτή οφείλεται κυρίως στη σημαντική μείωση του κόστους (σε χρόνο και χρήματα) για την παραγωγή τέτοιων δεδομένων στο πεδίο της βιολογίας. Για την αποδοτική επεξεργασία και ανάλυση ακολουθιακών δεδομένων έχουν προταθεί διάφορα ευρετήρια. Το σημαντικότερο από αυτά είναι το δένδρο επιθεμάτων (suffix tree). Καθώς το μέγεθος των δεδομένων που πρέπει να ευρετηριοποιηθεί αυξάνει, μεγαλώνει επίσης και το μέγεθος του αντίστοιχου ευρετηρίου. Επειδή το δένδρο επιθεμάτων είναι πάντα αρκετά μεγαλύτερο από τα δεδομένα, είναι πια σύνηθες να μην επαρκεί το μέγεθος της κύριας μνήμης για την αποθήκευσή του. Αντικείμενο αυτής της διπλωματικής είναι η μελέτη των δένδρων επιθεμάτων για αυτές ακριβώς τις περιπτώσεις. Στα πλαίσια αυτά μελετώνται σύγχρονες τεχνικές κατασκευής που έχουν προταθεί στη βιβλιογραφία και υλοποιούνται γνωστοί αλγόριθμοι αναζήτησης και επεξεργασίας ακολουθιών που τα χρησιμοποιούν.

Να σημειωθεί ότι τα προβλήματα που μελετώνται είναι υπαρκτά και έχουν τεράστιο ενδιαφέρον για την έρευνα στις βιοεπιστήμες. Οι βιολογικές ακολουθίες παρουσιάζουν

κάποια χαρακτηριστικά στην δομή τους, η κατανόηση και η στατιστική ανάλυση των οποίων οδηγεί σε ιδιαίτερα χρήσιμα ερευνητικά συμπεράσματα. Ενδεικτικά παραδείγματα είναι η ύπαρξη συμπληρωματικών παλινδρόμων, η ύπαρξη πολλαπλών επαναλήψεων ποικίλλου μήκους, η παρεμβολή περιοχών χωρίς γενετική έκφραση (εσώνια) με άγνωστο ακόμα λειτουργικό ρόλο κλπ. Επιπλέον, οι διάφορες μεταλλάξεις του γενετικού υλικού κατευθυνόμενες από την φυσική επιλογή οδηγούν σε διαφοροποιήσεις μεταξύ γονιδιωμάτων των ατόμων και των πληθυσμών, και η στατιστική ανάλυση των διαφορών είναι ένα χρήσιμο εργαλείο της γενετικής πληθυσμών αλλά και της κατανόησης κληρονομικών ασθενειών.

1.2 Συνεισφορά

Η σημαντικότερη συνεισφορά της συγκεκριμένης διπλωματικής εργασίας συνίσταται στα ακόλουθα σημεία:

1.2.1 Θεωρητική μελέτη των τεχνολογιών αποδοτικής κατασκευής δένδρων επιθεμάτων μεγάλου μεγέθους

Ο πυρήνας της θεωρητικής μελέτης της συγκεκριμένης διπλωματικής είναι οι αλγόριθμοι αποδοτικής κατασκευής δένδρων επιθεμάτων, των οποίων το μέγεθος είναι μεγαλύτερο από τη διαθέσιμη κύρια μνήμη. Ήδη από τη δεκαετία του ’70 έχουν παρουσιαστεί αποδοτικοί τρόποι για την κατασκευή των δένδρων επιθεμάτων όταν υπάρχει διαθέσιμη επαρκής μνήμη, με χαρακτηριστικό παράδειγμα τον αλγόριθμο του Ukkonen [Ukk95]. Το τοπίο δείχνει να αλλαζει εντελώς όταν το δένδρο δεν χωράει στην μνήμη. Ο λόγος είναι ότι κατά την κατασκευή του δένδρου επιθεμάτων με τέτοιους αλγορίθμους, οι διασχίσεις που συμβαίνουν στο δένδρο δεν εμφανίζουν καλά χαρακτηριστικά τοπικότητας. Αυτό μεταφράζεται σε υπερβολικά μεγάλο φόρτο εισόδου/ εξόδου στο δίσκο. Το συγκεκριμένο πρόβλημα άρχισε να μελετάται κατά τις αρχές της παρούσας δεκαετίας. Στα πλαίσια της διπλωματικής, παρακολουθήσαμε και καταγράψαμε τις εξελίξεις στο συγκεκριμένο πρόβλημα. Συγκεκριμένα, μελετήθηκαν οι πιο αποτελεσματικές μέθοδοι που υπάρχουν στην βιβλιογραφία για κατασκευή του δένδρου επιθεμάτων στον δίσκο. Μελετήθηκε η προσέγγιση της Hunt [HAI01] που κατακερματίζει το δένδρο στην βάση προθεμάτων σταθερού μήκους, οι αλγόριθμοι Dyna-Cluster [CYL05] και TOP-Q [BH04] που χρησιμοποιούν έξυπνες στρατηγικές buffering, ο αλγόριθμος TDD [THP04] που υλοποιεί μια παραλλαγή του αλγοριθμου wotdeager [GKJ03]. Τέλος μελετήθηκε διεξοδικά ο αλγόριθμος TRELLIS [PZ07], που, απ’ όσο γνωρίζουμε, είναι η πιο ώριμη δουλειά στο χώρο. Ο συγκεκριμένος αλγόριθμος φαίνεται να είναι ο ταχύτερος, δίνει τη δυνατότητα για διατήρηση των

συνδέσμων επιθέματος (που είναι ιδιαίτερα χρήσιμοι για διάφορους αλγορίθμους) και παρέχει ανοιχτού κώδικα υλοποίηση.

1.2.2 Θεωρητική μελέτη αλγορίθμων ανάλυσης και αναζήτησης ακολουθιακών δεδομένων

Πέρα απ' αυτά μελετήθηκαν και παρουσιάστηκαν γνωστοί αλγόριθμοι που επιλύουν προβλήματα ανάλυσης και αναζήτησης ακολουθιακών δεδομένων. Κάποιοι από αυτούς τους αλγορίθμους χρησιμοποιούν δένδρα επιθεμάτων και κάποιοι άλλοι δεν χρησιμοποιούν. Η παρουσίαση των πρώτων έγινε γιατί τη συμπεριφορά αυτών ακριβώς των αλγορίθμων είναι που θέλουμε να δοκιμάσουμε, όταν το δένδρο που χρησιμοποιούν ξεπερνά σε μέγεθος την κύρια μνήμη. Η παρουσίαση των δεύτερων έγινε γιατί αυτούς του αλγορίθμους τους χρησιμοποιούμε ως βάση σύγκρισης για την αποδοτικότητα των αλγορίθμων που χρησιμοποιούν τα δένδρα επιθεμάτων. Οι αλγόριθμοι αυτοί ομαδοποιούνται σε αλγορίθμους για την ακριβή ταύτιση προτύπου, αλγορίθμους για την προσεγγιστική ταύτιση προτύπου και αλγορίθμους εύρεσης κοινών προθεμάτων συγκεκριμένου μήκους. Οι τελευταίες περιπτώσεις προϋποθέτουν ότι το δένδρο επιθεμάτων απαντά σε σταθερό χρόνο ερωτήματα μέγιστης κοινής επέκτασης (LCE). Αυτά τα ερωτήματα στο δένδρο επιθεμάτων ανάγονται σε ερωτήματα κατώτατου κοινού προγόνου (LCA), για τα οποία υπάρχει αλγόριθμος που μπορεί να τα απαντήσει σε σταθερό χρόνο, με την προϋπόθεση ότι το δένδρο έχει υποστεί κατάλληλη προεπεξεργασία. Για το λόγο αυτό μελετούμε θεωρητικά και παρουσιάζουμε επίσης το συγκεκριμένο άλγοριθμο.

1.2.3 Επέκταση του συστήματος ανοιχτού κώδικα TRELLIS και υλοποίηση αλγορίθμων που χρησιμοποιούν το αντίστοιχο δένδρο

Όπως αναφέρθηκε και προηγουμένως, ένα άλλο κομμάτι της παρούσας διπλωματικής είναι η μελέτη της χρήσης των παραγόμενων δένδρων επιθεμάτων κατά την απάντηση διαφόρων γνωστών τύπων ερωτημάτων. Στα πλαίσια αυτά, υλοποιήσαμε καθιερωμένους αλγορίθμους που χρησιμοποιούν δένδρα επιθεμάτων μεγάλου μεγέθους για να απαντήσουν γνωστά ερωτήματα πάνω σε ακολουθιακά δεδομένα. Για την αποδοτική κατασκευή των δένδρων επιθεμάτων, χρησιμοποιήσαμε την ανοιχτού κώδικα υλοποίηση του αλγορίθμου TRELLIS, η οποία παρέχεται από τους συγγραφείς της αντίστοιχης ερευνητικής εργασίας. Επεκτείναμε τον στοιχειώδη κώδικα για διάσχιση του παραγόμενου δένδρου που παρεχόταν και παραλλάξαμε τη μορφή των αρχείων του παραγόμενου δένδρου κατάλληλα, βελτιώνοντας θέματα αποδοτικότητας.

Πιο συγκεκριμένα, οι σημαντικότερες βελτιωτικές παρεμβάσεις που εφαρμόσαμε στο δένδρο ήταν η δημιουργία αποδοτικότερης δομής για τη διατήρηση των προθεμάτων και η επέκταση του δένδρου με την κατάλληλη πληροφορία προκειμένου να απαντώνται ερωτήματα LCA σε σταθερό χρόνο. Για την πρώτη παρέμβαση, δημιουργήσαμε ένα trie, το οποίο αποθηκεύεται στον δίσκο. Το μικρό του μέγεθος επιτρέπει να φορτώνεται στην μνήμη για χρήση από τις εφαρμογές χωρίς να τροποποιούνται οι παραδοχές που έχουν γίνει για το μέγεθος της διαθέσιμης μνήμης. Στην αρχική υλοποίηση το σύνολο των προθεμάτων ήταν αποθηκευμένα σειριακά σε ένα αρχείο κειμένου, επιβαρύνοντας κάθε μετέπειτα εφαρμογή με επιπλέον κόστος για την ανάκτησης και την επεξεργασίας. Η δεύτερη παρέμβαση, αφορά στην προεπεξεργασία των αρχείων των φύλλων και των εσωτερικών κόμβων για την δυνατότητα ανάκτησης του κατώτατου κοινού προγόνου (LCA) δύο κόμβων του δένδρου σε σταθερό χρόνο. Με αυτή την παράμβαση μπορούμε να απαντούμε σε σταθερό χρόνο ερωτήματα μέγιστης κοινής επέκτασης, τα οποία βρίσκονται στον πυρήνα αρκετών αλγορίθμων.

Από άποψη αλγορίθμων πάνω σε δένδρα επιθεμάτων, υλοποιήθηκαν καθιερωμένοι αλγόριθμοι για ακριβή και προσεγγιστική εύρεση προτύπου. Κάποιοι από τους αλγορίθμους κάνουν χρήση συνδέσμων επιθέματος και κάποιοι όχι.

1.2.4 Πειραματική αξιολόγηση

Τα πειράματα που πραγματοποιήθηκαν προσπαθούν να δείξουν τη συμπεριφορά αυτών των αλγοριθμών, ενώ χρησιμοποιούνται αλγόριθμοι στη μνήμη ως baseline προσεγγίσεις.

Μπορεί οι αλγόριθμοι για την κατασκευή δένδρων επιθεμάτων μεγάλου μεγέθους να έχουν μελετηθεί εξονυχιστικά για την απόδοση κατασκευής τους, όμως, απ' όσο γνωρίζουμε, δεν έχει γίνει μέχρι τώρα κάποια συστηματική μελέτη για την εφαρμογή γνωστών αλγορίθμων πάνω σε αυτά, ούτε έχουν γίνει αντίστοιχες υλοποιήσεις. Η συγκεκριμένη διπλωματική έρχεται να καλύψει αυτήν ακριβώς την ανάγκη.

1.3 Οργάνωση του κειμένου

Το κείμενο της διπλωματικής αποτελείται από 8 κεφάλαια. Στο παρόν πρώτο κεφάλαιο αναφέρεται το αντικείμενο της διπλωματικής, και κάποιες γενικές πληροφορίες για το οικείο πεδίο έρευνας. Περιγράφεται συνοπτικά η συνεισφορά της διπλωματικής και ακολουθεί η περιγραφή της οργάνωσης του κειμένου. Στο δεύτερο κεφάλαιο επιχειρείται η παροχή στον αναγνώστη του απαραίτητου θεωρητικού υποβάθρου για την κατανόηση της εργασίας. Μετά από την εξοικείωση με κάποιους βασικούς όρους και την παρουσίαση κάποιων στοιχειωδών γνώσεων βιολογίας, ακολουθεί μια ανασκόπηση βασικών αλγορίθμων πάνω σε ακολουθιακά δεδομένα, με χρήση ευρετηρίων και χωρίς. Μετά την παρουσίαση των σημαντικότερων

αλγορίθμων ακριβούς αναζήτησης προτύπων χωρίς χρήση ευρετηρίου, με έμφαση στους πιο δημοφιλείς (Knuth-Morris-Pratt [KMP77], Boyer-Moore [BM77]), παρουσιάζονται οι ορισμοί για τα δένδρα και τους πίνακες επιθεμάτων, και οι μέθοδοι αναζήτησης προτύπων σε αυτά. Ακολουθεί η περιγραφή του προβλήματος της μέγιστης κοινής επέκτασης ακολουθιών, και η αναγωγή του στο πρόβλημα του κατώτατου κοινού προγόνου (LCA) δύο κόμβων του δένδρου επιθεμάτων. Ορίζεται το πρόβλημα της προσεγγιστικής ταύτισης προτύπου, και περιγράφονται οι μέθοδοι επίλυσης του με χρήση ευρετηρίων και χωρίς. Τέλος, περιγράφεται αναλυτικά ο αλγόριθμος του Ukkonen [Ukk95]. Στο τρίτο κεφάλαιο γίνεται μια ανασκόπηση των αλγορίθμων για την αποθήκευση δένδρων επιθεμάτων στον δίσκο. Παρουσιάζεται ο αλγόριθμος της Hunt [HAI01], οι αλγόριθμοι DynaCluster [CYL05] και TOP-Q [BH04], ο αλγόριθμος TDD [THP04], και ενδελεχέστερα από όλους ο αλγόριθμος TRELLIS [PZ07] που σύμφωνα με πειραματικά αποτελέσματα έχει την καλύτερη απόδοση. Στο τέταρτο κεφάλαιο αναφέρονται οι υλοποιήσεις που πραγματοποιήσαμε πάνω στο σύστημα των δένδρων του TRELLIS και περιγράφονται αναλυτικά οι παρεμβάσεις στο σύστημα των αρχείων. Στο πέμπτο κεφάλαιο παρατίθεται η πειραματική αξιολόγηση των υλοποιήσεων. Παρουσιάζονται διαγραμματικά τα αποτελέσματα των χρονομετήσεων και γίνεται ποιοτική αποτίμηση και σύγκριση των αποτελεσμάτων με τα αποτελέσματα μεθόδων που δεν κάνουν χρήση ευρετηρίων. Στο έκτο κεφάλαιο γίνεται μία σύνοψη της διπλωματικής και των αποτελεσμάτων και προτείνονται μελλοντικές επεκτάσεις και το έβδομο κεφάλαιο παρατίθεται η βιβλιογραφία.

2 *Θεωρητικό υπόβαθρο και σχετικές εργασίες*

2.1 Βιολογικοί όροι και αναπαραστάσεις στον Η/Υ

2.1.1. Βιολογικά μακρομόρια, DNA, RNA και πρωτεΐνες

Η κληρονομικότητα των χαρακτηριστικών ανθρώπων και ζώων αλλά και η ποικιλομορφία μεταξύ ατόμων του ιδίου είδους ήταν γνωστή στον άνθρωπο ήδη από τους προϊστορικούς χρόνους. Πολλά ζώα εξημερώθηκαν από τον άνθρωπο κατά την ύστερη Παλαιολιθική περίοδο, και με τεχνητή επιλογή, με ελεγχόμενες δηλαδή διασταυρώσεις, απέκτησαν σταδιακά επιθυμητές ιδιότητες. Η τεχνητή επιλογή και διάφορες άλλες τεχνικές βελτίωσης των χαρακτηριστικών ήταν επίσης γνωστές για πλήθος αγροτικών προϊόντων. Τα επιτεύγματα αυτά έδωσαν με τη σειρά τους μεγάλη ώθηση στην ανάπτυξη τομέων της οικονομίας, για παράδειγμα, η δημιουργία ικανών ποιμενικών σκύλων βοήθησε στην περαιτέρω ανάπτυξη και μαζικοποίηση της κτηνοτροφίας. Η παρατήρηση άλλωστε ότι διασταυρώσεις μεταξύ κοντινών συγγενών έχουν ως συνέπεια την αύξηση της θνησιμότητας των απογόνων οδήγησε από πολύ νωρίς τις ανθρώπινες κοινωνίες στην απαγόρευση τέτοιων επαφών. Ο άνθρωπος λοιπόν από την αυγή της ιστορίας προσπάθησε να θέσει υπό τον έλεγχο του τον μηχανισμό μεταβίβασης των χαρακτηριστικών από γενιά σε γενιά. Ωστόσο ως και τα μέσα του 20^{ου} αιώνα ο ακριβής μηχανισμός μεταβίβασης των χαρακτηριστικών και ο φορέας στον οποίον εδράζονται οι γενετικές πληροφορίες ήταν άγνωστοι.

Το DNA (δεσοξυριβοζονουκλεϊκό οξύ) απομονώθηκε για πρώτη φορά το 1869 από τον Ελβετό ιατρό Friedrich Miescher ο οποίος το ονόμασε νουκλεΐνη αφού εδράζεται στον

πυρήνα του κυττάρου. Ωστόσο χρειάστηκε να περάσουν πολλές δεκαετίες μέχρι να ανακαλυφθεί ο ρόλος του ως γενετικού υλικού των κυττάρων, και το 1953 οι James Watson και Francis Crick πρότειναν το μοντέλο της διπλής έλικας για την δομή του DNA. Το μόριο του DNA αποτελείται από δύο πολυνουκλεοτιδικές αλυσίδες. Ο άξονας κάθε αλυσίδας αποτελείται από δεσοξυριβόζη και τη φωσφορική ομάδα κάθε νουκλεοτιδίου, ενώ κάθετα σε κάθε αλυσίδα προεξέχουν αζωτούχες βάσεις. Οι βάσεις του DNA είναι η αδενίνη (A), η θυμίνη (T), η γουανίνη (G) και η κυτοσίνη (C). Οι βάσεις έχουν την ιδιότητα της συμπληρωματικότητας, δηλαδή τα ζεύγη A-T και C-G όταν βρίσκονται αντικριστά αναπτύσσουν μεταξύ τους δεσμούς υδρογόνου και συγκρατούνται μαζί. Η αλληλουχία των βάσεων σε έναν κλώνο συνιστά την πληροφορία του γενετικού υλικού, αποτελεί δηλαδή την πληροφορία που κωδικοποιεί το πρόγραμμα κατασκευής των πρωτεϊνών. Η πληροφορία που κωδικοποιεί λοιπόν η πρώτη αλυσίδα καθορίζει μονοσήμαντα την πληροφορία που είναι κωδικοποιημένη στην δεύτερη αλυσίδα και οι δύο αλυσίδες είναι συμπληρωματικές και συγκρατούνται με έλκτικες δυνάμεις σχηματίζοντας στο χώρο δομή διπλής έλικας. Ο Crick λίγα χρόνια αργότερα εισήγαγε το Κέντρικο Δόγμα της Μοριακής Βιολογίας εγκαινιάζοντας παράλληλα και την αντίστοιχη επιστήμη. Το Δόγμα αυτό περιγράφει τη σχέση μεταξύ DNA, RNA και πρωτεϊνών και είναι σε γενικές γραμμές αποδεκτό μέχρι σήμερα. Σύμφωνα με αυτό, το γενετικό υλικό ενός κυττάρου βρίσκεται αποθηκευμένο στο δίκλωνο DNA του πυρήνα του, ενώ με μια διαδικασία ημισυντηρητικού αναδιπλασιασμού μεταφέρει αυτούσιες τις γενετικές πληροφορίες στα κύτταρα απογόνους του. Το DNA που γενικά είναι δίκλωνο μεταγράφεται σε RNA, που είναι ένα γενικά μονόκλωνο μόριο παρεμφερούς δομής, το οποίο μεταφέρει την γενετική πληροφορία στα ριβοσώματα στα οποία το RNA μεταφράζεται σε πρωτεΐνες. Το RNA περιέχει τις βάσεις A, C, και G του DNA και την ουρακίλη (U) στη θέση της θυμίνης (T). Ισχύει η αντίστοιχη συμπληρωματικότητα A-U. Το DNA περιέχει λοιπόν κωδικοποιημένη την πληροφορία κατασκευής πρωτεϊνών που είναι απαραίτητες για την λειτουργία των κυττάρων. Παρόλο που ο μεγάλος όγκος του DNA εδράζεται στον πυρήνα, DNA περιέχουν και άλλα οργανίδια του κυττάρου όπως τα μιτοχόνδρια, και κάποια ιδιαίτερα χαρακτηριστικά του μιτοχονδριακού DNA όπως και το γεγονός ότι στον άνθρωπο και σε άλλους διπλοειδείς οργανισμούς το μιτοχονδριακό DNA μεταφέρεται μόνο από τον μητρικό πρόγονο (τα μιτοχόνδρια των σπερματοζωαρίων καταστρέφονται αμέσως μετά την γονιμοποίηση) τα καθιστούν ιδανικά για μελέτες γενετικής πληθυσμών.

Οι πρωτεΐνες ή πολυπεπτίδια είναι μακρομόρια που αποτελούνται από αμινοξέα και έχουν ρόλο είτε δομικό είτε λειτουργικό. Κάποιες πρωτεϊνες δηλαδή αποτελούν δομικά στοιχεία των κυττάρων και κάποιες άλλες λειτουργούν ως βιοκαταλύτες επιταγχύνοντας διάφορες βιοχημικές αντιδράσεις. Τα αμινοξέα είναι 20 το πλήθος και μπορούν να συνδεθούν μεταξύ τους κατά εκατοντάδες ή και χιλιάδες με όλους τους δυνατούς τρόπους δίνοντας έτσι ένα

πρακτικά απεριόριστο πλήθος πιθανών πρωτεϊνών. Εκτός από την συγκεκριμένη αλληλουχία των αμινοξέων που συνιστούν μία πρωτεΐνη, που ονομάζεται πρωτοταγής δομή της πρωτεΐνης, κρίσιμος παράγοντας για την λειτουργικότητα μιας πρωτεϊνης συνιστά η αναδίπλωση και η μακροσκοπική τοπολογία των διαφόρων πτυχώσεων της πρωτεϊνης, γνωστές ως δευτεροταγής και τριτοταγής δομή. Το γενετικό υλικό περιέχει κωδικοποιημένη την αλληλουχία των αμινοξέων που συνιστούν μια πρωτεΐνη σε μια περιοχή που ονομάζεται γονίδιο. Ένα γονίδιο λοιπόν περιέχει το πρόγραμμα κατασκευής της πρωτεΐνης ενώ μετά την πρωτεϊνοσύνθεση η πρωτεΐνη αποκτά την επιθυμητή δευτεροταγή και τριτοταγή δομή με την βοήθεια άλλων πρωτεϊνών και βιοχημικών διεργασιών. Κάποια γονίδια ονομάζονται ρυθμιστικά γονίδια καθώς οι πρωτεΐνες που κωδικοποιούν ενεργοποιούν με την σειρά τους πλήθος άλλων γονιδίων. Το σύνολο του γενετικού υλικού ενός οργανισμού ονομάζεται γονιδίωμα. Μέσα σε κάθε γονίδιο υπάρχουν περιοχές που κωδικοποιούν αμινοξέα και ονομάζονται εξώνια, και άλλες περιοχές που ονομάζονται εσώνια και δεν κωδικοποιούν πληροφορία για κατασκευή πρωτεϊνών και η χρησιμότητα τους είναι πεδίο επιστημονικής έρευνας. Γενικά οι χημικές διαδικασίες του αναδιπλασιασμού, της μεταγραφής, της μετάφρασης, οι διεργασίες που τις πυροδοτούν, η διαφοροποίηση, η διαφορετική δηλαδή έκφραση του γενετικού υλικού σε κύτταρα διαφορετικών ιστών, και γενικότερα η γενετική έκφραση είναι λειτουργία εξαιρετικά πολύπλοκη και στην λεπτομέρεια της δεν είναι ακόμα αποσαφηνισμένη. Η αναλυτική περιγραφή των παραπάνω διαδικασίων υπερβαίνει τους σκοπούς του παρόντος εισαγωγικού σημειώματος και ο αναγνώστης που θέλει να μελετήσει ενδελεχέστερα αυτό το πεδίο θα πρέπει να καταφύγει σε σχετική βιβλιογραφία.

2.1 Γενετικός κώδικας και βιολογικά δεδομένα

2.1.1 Γενετικός κώδικας

Όπως αναφέραμε οι πληροφορίες που αφορούν στην κατασκεύη μίας πρωτεΐνης βρίσκονται κωδικοποιημένες στην αλληλουχία των βάσεων. Το σύνολο των κανόνων σύμφωνα με τους οποίους μια συγκεκριμένη αλληλουχία βάσεων μεταφράζεται σε μια αλληλουχία αμνοξέων ονομάζεται **γενετικός κώδικας**. Κάθε λέξη της γλώσσας του γενετικού κώδικα αποτελείται από 3 βάσεις, ονομάζεται κωδικόνιο και κωδικοποιεί είτε την κατασκευή ενός αμινοξέος είτε εντολές έναρξης και λήξης της πρωτεϊνοσύνθεσης. Η πρωτεϊνοσύνθεση γίνεται όπως αναφέραμε στα ριβοσώματα στη βάση της πληροφορίας που μεταφέρει το RNA και άρα ο γενετικός κώδικας έχει ως αλφάβητο του τις βάσεις του RNA, A, U, C και G. Τα κωδικόνια αυτά βέβαια εχουν προκύψει από την μεταγραφή ενός τμήματος DNA και άρα βρίσκονται κωδικοποιημένα με το αλφάβητο του DNA (A, T, C ,G) στον αντίστοιχο συμπληρωματικό κλώνο. Γίνεται αμέσως αντιληπτό ότι ενώ τα αμινοξέα είναι 20, το πλήθος των λέξεων με

αλφάβητο 4 βάσεων και μήκους 3 είναι 4^3=64 άρα υπάρχει περιθώριο για κωδικοποίηση αμονοξέων από περισσότερα του ενός κωδικονίου. Πράγματι με την εξαίρεση 2 αμινοξέων, υπάρχουν περισσότερες από μία κωδικοποιήσεις για ένα αμινοξύ. Αυτό εξασφαλίζει την ανθεκτικότητα του γενετικού υλικού σε μεταλλάξεις, μία τυχαία μετάλλαξη δηλαδή σε κάποια βάση έχει σημαντικές πιθανότητες να μην επηρεάσει την γενετική έκφραση. Το αξιοσημείωτο με τον γενετικό κώδικα είναι ότι είναι οικουμενικός, δηλαδή με ελάχιστες εξαιρέσεις, είναι πανομοιότυπος για το σύνολο των έμβιων, κάτι που σημαίνει ότι πρωτεΐνες που χρησιμοποιούνται από τον ανθρώπινο οργανισμό μπορούν με μεθόδους γενετικής μηχανικής να παραχθούν εργαστηριακά με την βοήθεια βακτηρίων.

2.1.2 Βιολογικά δεδομένα

Το σύνολο επομένως της πληροφορίας που είναι απαραίτητη για την πρωτεϊνοσύνθεση, βρίσκεται αποθηκευμένο στην σειρα με την οποία απαντώνται οι αζωτούχες βάσεις κάθετα στις έλικες. Ζητούμενο είναι η δημιουργία ψηφιακών ακολουθιακών δεδομένων από αυτά τα βιολογικά δεδομένα.

Τα μηχανήματα που μετατρέπουν τα βιολογικά δεδομένα σε ψηφιακά ακολουθιακά δεδομένα ονομάζονται ακολουθιοποιητές. Η τεχνολογία που χρησιμοποιούν είναι αντικείμενο εντατικής έρευνας και βελτιώνεται συνεχώς, ώστε σήμερα, σε εύλογο χρόνο και με μικρό σχετικά κόστος να είναι δυνατή η ακολουθιοποίηση εκατοντάδων χιλιάδων βάσεων. Τα μηχανήματα αυτά είναι διασυνδεδεμένα σε υπολογιστή και πέραν της ακολουθιοποίησης έχουν την δυνατότητα να εκτιμήσουν τον βαθμό αξιοπιστίας της ταυτοποίησης του κάθε νουκλεοτιδίου που αναλύεται. Ένα πρόβλημα που παρουσιάζεται κατά την χαρτογράφηση, είναι το γεγονός ότι τα βιολογικά δεδομένα που συλλέγονται στο εργαστήριο είναι συνήθως κατακερματισμένα σε θραύσματα, και επομένως πρέπει να γίνει σωστά η επανασυγκόλληση με την χρήση του υπολογιστή. Το πρόβλημα αυτό δεν είναι εύκολο καθώς λόγω των επαναληπτικών δομών που έχει το γενετικό υλικό, συχνά η τοποθέτηση των τμημάτων δεν γίνεται με μονοσήμαντο τρόπο [Παπ07].

Γίνεται φανερό ότι η αλληλουχία του DNA μετά από την χαρτογράφηση είναι μια ψηφιακή πληροφορία, που μπορεί να παρασταθεί με ένα αλφάβητο 4 χαρακτήρων, και να αποθηκευθεί σε οποιοδήποτε ψηφιακό αποθηκευτικό μέσο και να υποστεί επεξεργασία και ανάλυση από ηλεκτρονικό υπολογιστή, μπορεί δηλαδή να αποτελέσει αντικείμενο της επιστήμης υπολογιστών.

2.1.3 Η Χαρτογράφηση του Ανθρώπινου Γονιδιώματος

Το 1990 η κυβέρνηση των Ηνωμένων Πολιτειών εκπόνησε ένα πρόγραμμα με την ονομασία Human Genome Project με στόχο την πλήρη χαρτογράφηση του ανθρώπινου γονιδιώματος,

προϋπολογισμού $3 δις. Το πρόγραμμα ολοκληρώθηκε τυπικά το 2003 ωστόσο μέχρι και σήμερα ένα ποσοστό περίπου 8% του γονιδιώματος δεν έχει χαρτογράφηθεί. Πρόκειται για τις λεγόμενες ετεροχρωματικές περιοχές και περιοχές στο κέντρο των χρωμοσωμάτων που έχουν εξαιρετική πυκνότητα και είναι πολύ δυσχερής η χαρτογράφηση τους.

Κατά τη διαδικασία της χαρτογράφησης προέκυψαν πολλά νέα ευρήματα. Έγινε φανερό ότι ο άνθρωπος διάθετει 25000-30000 γονίδια, πλήθος αντίστοιχο με αυτό του ποντικού και μόλις 3 φορές περισσότερα από αυτά ενός βακτηριδίου. Σε ένα μεγάλο ποσοστό μάλιστα τα γονίδια αυτά είναι ακριβώς τα ίδια. με αυτά του ποντικού. Επίσης προέκυψε ότι τα γονιδιώματα διαφορετικών ανθρώπων είναι κατά 99,99% ίδια ανεξαρτήτως φυλής. Το απλό ανθρώπινο γονιδιώμα, δηλαδή ένα χρωμόσωμα από κάθε ένα από τα 23 ζεύγη χρωμοσωμάτων, έχει περίπου $3{,}3 \cdot 10^9$ ζεύγη βάσεων (bps-base pairs) και άρα το πλήρες γονιδίωμα ενός ανθρώπου (46 χρωμοσώματα) μπορεί να αποθηκευτεί σε 6,6 GB σε κάποιο ψηφιακό μέσο αποθήκευσης και χρησιμοποιώντας συμπιεσμένη μορφή, δηλαδή χρησιμοποιώντας 2 bits για κάθε βάση, σε περίπου 1,7 GB.

Τα ποσά που απαιτήθηκαν για την χαρτογράφηση του γονιδιώματος μπορεί να φαίνονται αστρονομικά, και παραμένουν ως και σήμερα πολύ ψηλά και απροσπέλαστα για ιδιώτες, ωστόσο το κόστος που απαιτείται για την χαρτογράφηση μειώνεται με φρενήρη ρυθμό, και διατίθενται τεράστια ποσά στην έρευνα για νέες τεχνολογίες που θα μειώσουν περαιτέρω αυτό το κόστος. Το 2006 το κόστος για την πλήρη χαρτογράφηση ενός ανθρώπινου γονιδιώματος υπολογιζόταν σε $100 εκατ. με $150 εκατ. ενώ τον Αύγουστο του 2009 ο καθηγητής Stephen Quake του Stanford University υποστήριξε ότι έκανε χαρτογράφηση του γονιδιώματός του με μόλις $50.000, παρουσιάζοντας στο περιοδικό Nature Biotechnology μία νέα μέθοδο που μειώνει αρκετά το κόστος και προβλέποντας ότι πολύ σύντομα το κόστος θα έχει πέσει σε $1000 [Tel09].

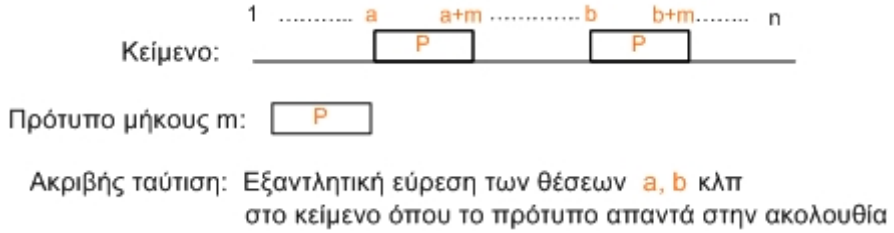
Αναμένεται λοιπόν μία έκρηξη βιολογικών δεδομένων, κατί που γεννά ένα αίτημα για αποδοτικούς τρόπους αποθήκευσης, ανάλυσης και διαχείρισης. Η έρευνα για αποδοτική αποθήκευση και ανάλυση τέτοιων δεδομένων ξεκίνησε ήδη από τη δεκαετία του ’70 προτείνοντας διάφορους τρόπους ευρετηριοποίησης των δεδομένων. Μία από τις πιο σημαντικές λειτουργίες πάνω στα βιολογικά δεδομένα είναι η αναζήτηση προτύπων, υποακολουθιών δηλαδή της βασικής ακολουθίας.

2.2 Αλγόριθμοι αναζήτησης σε βιολογικά δεδομένα

2.2.1 Ακριβής ταύτιση προτύπου (exact matching)

2.2.1.1 Το πρόβλημα

Το πρόβλημα της ακριβούς ταύτισης προτύπου συνίσταται στην εύρεση όλων των θέσεων στις οποίες εμφανίζεται κάποια υποακολουθία-πρότυπο μέσα σε μια (αρκετά μεγαλύτερη συνήθως) ακολουθία, που ονομάζεται κείμενο. Το πρόβλημα αυτό δεν είναι μόνο μία ενδιαφέρουσα πρόκληση για τους θεωρητικούς, αλλά έχει και μεγάλο πρακτικό ενδιαφέρον καθώς βρίσκεται στον πυρήνα πολλών εφαρμογών από διάφορους ερευνητικούς τομείς: Υπολογιστική Βιολογία (Computational Biology), Επεξεργασία Σήματος (Signal Processing), Ανάκτηση Κειμένου (Text Retrieval) κ.τ.λ. Όσον αφορά τα βιολογικά δεδομένα, είναι προφανές ότι το πρόβλημα αυτό μπορεί να απαντήσει πολλά πρακτικά ερωτήματα, τα οποία σχετίζονται με την αναζήτηση ομοιοτήτων μεταξύ γονιδιωμάτων διαφορετικών ειδών, τον εντοπισμό όμοιων γονιδίων, την εύρεση όμοιων αλληλουχιών αμινοξέων σε διαφορετικές πρωτεΐνες και γενικότερα την παραγωγή χρήσιμων στατιστικών συμπερασμάτων για τη δομή του γονιδιώματος. Η απαίτηση της ακριβούς ταύτισης, μπορεί να μοιάζει αρκετά περιοριστική, όμως ακόμα και στην περίπτωση της προσεγγιστικής ταύτισης προτύπου (βλέπε 2.2.3) πολλοί αλγόριθμοι απαιτούν στον πυρήνα τους την ακριβή εύρεση προτύπου.



Για τις ανάγκες της ανάλυσης μας θα συμβολίσουμε την ακολουθία-πρότυπο ως P= P[1…m], μήκους m και το κείμενο ως S=[1…n], μήκους n. Θεωρούμε ότι οι ακολουθίες μας κατασκευάζονται πάνω στο ίδιο αλφάβητο Σ και ότι η πληθικότητα του αλφαβήτου (δηλαδή το πλήθος των διαφορετικών συμβόλων που υπάρχουν στο αλφάβητο) συμβολίζεται με σ. Για παράδειγμα, αν οι ακολουθίες είναι κομμάτια DNA, τότε το αλφάβητο που χρησιμοποιείται είναι το Σ={A,T,C,G} με πληθικότητα σ=4.

Οι αλγόριθμοι για την ακριβή ταύτιση προτύπων μπορούν να ταξινομηθούν σε δύο κατηγορίες. Στην πρώτη κατηγορία είναι οι αλγόριθμοι που δεν απαιτούν προεπεξεργασία της ακολουθίας κειμένου (μπορεί όμως να απαιτούν την προεπεξεργασία του προτύπου). Στην δεύτερη κατηγορία είναι οι αλγόριθμοι που απαιτούν την προεπεξεργασία της

ακολουθίας κειμένου, για την παραγωγή κάποιας μορφή ευρετηρίου, που στη συνέχεια χρησιμοποιείται κατά την αναζήτηση.

2.2.1.2 Αλγόριθμοι χωρίς προεπεξεργασία του κειμένου

Οι αλγόριθμοι που αναζητούν το πρότυπο δίχως προεπεξεργασία του κειμένου, χρησιμοποιούν γενικά την μέθοδο του κυλιόμενου παραθύρου. Παράθυρο ονομάζεται μια συγκεκριμένη υποπεριοχή του κειμένου μας, συνήθως μεγέθους m. Αρχικά το παράθυρο στοιχίζεται έτσι ώστε το αριστερό του άκρο να βλέπει την αρχή του κειμένου. Στην συνέχεια γίνεται χαρακτήρας προς χαρακτήρας σύγκριση κειμένου και προτύπου εντός του παραθύρου. Μετά από κάποια αποτυχημένη σύγκριση χαρακτήρα ή μία πλήρη ταύτιση του κειμένου που περιέχεται στο παράθυρο, το παράθυρο ολισθαίνει προς τα δεξιά. Η προηγούμενη διαδικασία επαναλαμβάνεται έως ότου το δεξί άκρο του παραθύρου ξεπεράσει το τέλος του κειμένου.

Ο κάθε αλγόριθμος εφαρμόζει την παραπάνω διαδικασία με παραλλαγές στον τρόπο ολίσθησης του παραθύρου και στην επιλογή των θέσεων αναζήτησης. Η απλούστερη εκδοχή είναι ο αλγόριθμος Brute-force, που ταυτίζεται με αυτά που παρουσιάστηκαν παραπάνω, με χρονική πολυπλοκότητα O(mn), ανεξάρτητα μάλιστα με την σειρά αναζήτησης εντός του κειμένου. Η πολυπλοκότητα του δηλαδή μένει ίδια, ανεξάρτητα με την κατεύθυνση που αυτές γίνονται ή με την επιλογή των θέσεων, αφού ο αλγόριθμος συνολικά εκτελεί σε κάθε περίπτωση σχεδόν τον ίδιο αριθμό συγκρίσεων. Οι διάφορες παραλλαγές και βελτιώσεις του Brute-force ταξινομούνται σε τέσσερεις κατηγορίες:

- τους αλγόριθμους που κάνουν αναζήτηση με τον φυσικό τρόπο, δηλαδή από τα αριστερά προς τα δεξιά, .όπως ο Knuth-Morris-Pratt [KMP77], ο Shift-OR κ.ά...
- τους αλγορίθμους που κάνουν αναζήτηση από τα δεξιά προς τα αριστερά και είναι γενικά οι πιο αποδοτικοί, .όπως ο Boyer-Moore [BM77] και πλήθος παραλλαγών του.
- τους αλγορίθμους που κάνουν τις συγκρίσεις με κάποια καθορισμένη σειρά και έχουν τα καλύτερα θεωρητικά άνω όρια, όπως ο Galil-Seiferas [GS83] , ο KMP Skip and Search [CLP98] κ.ά.
- και τους αλγόριθμους για τους οποίους η σειρά με την οποία γίνονται οι συγκρίσεις είναι ανεξάρτητη από την απόδοσή τους. .όπως ο ίδιος ο Brute-Force, ο Smith [Smi91] κ.ά

Ο hash αλγόριθμος προτάθηκε από τον Harisson και αναλύθηκε περαιτέρω από τους Karp και Rabin [KR87] και προτείνει την χρήση μιας hashing συνάρτησης για την μείωση των απαιτούμενων συγκρίσεων. Η ιδέα είναι να γίνονται συγκρίσεις μονάχα στα παράθυρα που

‘ομοιάζουν’ με το πρότυπο. Ο έλεγχος της ομοιότητας βασίζεται στην χρήση μιας κατάλληλης συνάρτησης hashing. Παράθυρα που αποτυγχάνουν στον έλεγχο της hashing συνάρτησης είναι σίγουρο ότι δεν θα ταιριάζουν με το πρότυπο και επομένως αποφεύγεται να γίνουν συγκρίσεις γι’ αυτά. Η απαιτούμενη προεπεξεργασία του προτύπου για την δημιουργία της hashing συνάρτησης απαιτεί σταθερό χώρο και χρονική πολυπλοκότητα O(m) ενώ η αναζήτηση στην μέση περίπτωση, με κάποιες πιθανοτικά εύλογες παραδοχές, η πολυπλοκότητα είναι O(m+n). Στην χειρότερη περίπτωση ωστόσο, όταν δηλαδή αναζητείται το a^m μέσα στο a^n η πολυπλοκότητα είναι O(mn).

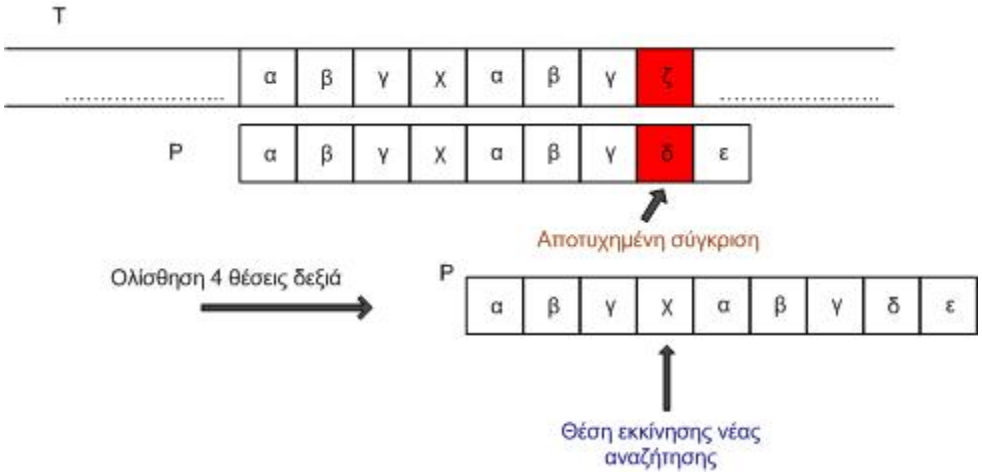
Ο αλγόριθμος Shift-Or [BG92] είναι αποδοτικός μόνο όταν το μέγεθος του προτύπου είναι αντίστοιχο με το μέγεθος της λέξης της μνήμης, αρκετά μικρό δηλαδή. Συνεπώς, η αποδοτικότητα του, εξαιτίας των bitwise τεχνικών που χρησιμοποιεί, βασίζεται στο unit-cost RAM μοντέλο σύμφωνα με το οποίο η ανάκτηση μιας λέξης από τη μνήμη και οι bitwise πράξεις σε λέξεις της μνήμης έχουν κόστος O(1). Ο αλγόριθμος έχει μια φάση προεπεξεργασίας πολυπλοκότητας O(m+σ) σε χώρο και χρόνο ενώ η φάση της αναζήτησης έχει χρονική πολυπλοκότητα O(n) και είναι ανεξάρτητη του μεγέθους του προτύπου και του αλφαβήτου.

Η προεπεξεργασία του αλγορίθμου αφορά στην δημιουργία σ το πλήθος δυαδικών αριθμών S_c μεγέθους m, ένας για κάθε χαρακτήρα c του αλφαβήτου. Κάθε bit i του αριθμού S_c για κάποιο σύμβολο του αλφαβήτου c, τίθεται $S_c(i)=0$ αν και μόνον αν P[i]=c, και 1 σε άλλη περίπτωση. Άρα κατά την προεπεξεργασία ο αλγόριθμος αποθηκεύει όλες τις θέσεις που εμφανίζεται μέσα στο πρότυπο κάποιο σύμβολο του αλφαβήτου, για όλα τα σύμβολα. Η αναζήτηση του προτύπου στο κείμενο γίνεται με χρήση ενός άλλου δυαδικού αριθμού R_j, με m το πλήθος bits όπου $R_j(i)=0$ όταν P[0,i]=T[j-i,j] και 1 σε κάθε άλλη περίπτωση. Γίνεται φανερό ότι μια πλήρης ταύτιση του προτύπου στην θέση j του κειμένου γίνεται όταν $R_j(m-1)=0$. Μπορεί να αποδειχθεί ότι ο δυαδικός αριθμός R_{j+1} για οποιοδήποτε $j<n$ προκύπτει αναδρομικά από τον R_j μέσω της σχέσης R_{j+1}= Left_Shift(R_j) OR $S_{T[j+1]}$. Επομένως αρκεί ο αλγόριθμος να θέσει όλα τα bits για το R_0 ίσα με 1 και στην συνέχεια με την παραπάνω δυαδική σχέση υπολογίζονται όλα τα επόμενης τάξης R καθώς διασχίζεται το κείμενο. Αφού οι δυαδικές πράξεις σύμφωνα με το μοντέλο εκτελούνται σε σταθερό χρόνο και γίνονται n επαναλήψεις ο αλγόριθμος κατά την αναζήτηση έχει πολυπλοκότητα O(n).

Ο πρώτος γνήσια γραμμικός αλγόριθμος ήταν ο αλγόριθμος Morris-Pratt [MP70] και σε βελτιωμένη μορφή ο Knuth-Morris-Pratt [KMP77]. Συνιστά μια βελτίωση του Brute-force και αξιοποιεί τις συγκρίσεις που γίνονται κατά τις αναζητήσεις που δεν οδηγούνται σε ταύτιση. Η προεπεξεργασία έχει πολυπλοκότητα χώρου και χρόνου O(m) ενώ η φάση της αναζήτησης έχει πολυπλοκότητα χρόνου O(n+m). Στην χειρότερη περίπτωση ο αλγόριθμος κατά την φάση της αναζήτησης θα κάνει 2n-1 συγκρίσεις ενώ ο αριθμός των συγκρίσεων που

θα γίνουν για έναν συγκεκριμένο χαρακτήρα του κειμένου φράσσεται από το $\log_{\Phi} m$ όπου Φ η χρυσή τομή ($\Phi=\frac{1+\sqrt{5}}{2}$)

Η ιδέα για τη μέθοδο ολίσθησης που εφαρμόζει ο αλγόριθμος Knuth-Morris-Pratt συνίσταται στην εκμετάλλευση της ύπαρξης επαναλαμβανόμενων υποακολουθιών μέσα στο πρότυπο ώστε μετά από μία αποτυχημένη ταύτιση κατά την διάρκεια της αναζήτησης, το κυλιόμενο παράθυρο ολισθαίνει παραπάνω από 1 χαρακτήρες και στην συνέχεια ξεκινάει την επόμενη σύγκριση όχι από τον πρώτο χαρακτήρα του προτύπου, αλλά από κάποιον εσωτερικό χαρακτήρα αφού είναι βέβαιο ότι κάποιο πρόθεμα του P ταυτίζεται με την αντίστοιχη υποακολουθία του κειμένου. Για παράδειγμα, όταν έχουμε το πρότυπο P= 'αβγχαβγδε', και είμαστε στη φάση αναζήτησης σε κάποιο σημείο του κειμένου, λόγω της ταύτισης των υποακολουθιών P[1..3]=P[5..7]='αβγ', αν υπάρξει μια αποτυχημένη σύγκριση για την θέση 8 του P, διαπιστώνουμε αμέσως ότι μπορούμε να ολιθήσουμε το παράθυρο για να συνεχίσουμε την αναζήτηση 4 θέσεις δεξιά. Επιπλέον, η νέα αναζήτηση μπορεί να ξεκινήσει όχι από το πρώτο γράμμα του προτύπου αλλά από τη θέση 4, την θέση δηλαδή του T στην οποία έλαβε χώρα η τελευταία αποτυχημένη αναζήτηση, αφού οι πρώτες 3 θέσεις είναι βέβαιο ότι ταυτίζονται.

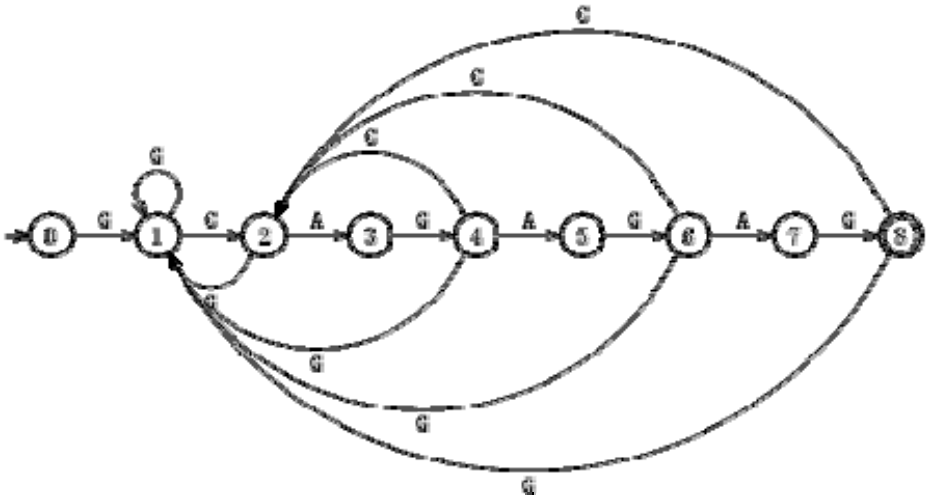


Έχει ενδιαφέρον ότι το κριτήριο για την ολίσθηση δεν έχει καθόλου να κάνει με την δομή του κειμένου T, και είναι ανεξάρτητο από αυτό, δηλαδή αν υπάρξει αποτυχημένη ταύτιση σε κάποιο σημείο της αναζήτησης, ο αριθμός των ολισθήσεων και η νέα θέση έναρξης της αναζήτησης εξαρτάται μόνο από το P.

Η γενική ιδέα είναι αρκετά κατανοητή, ο αλγόριθμος όμως πρέπει να βρει μια μέθοδο τυποποίησης του τρόπου με τον οποίο γίνονται οι ολισθήσεις. Για τον σκοπό αυτό ορίζεται για κάθε θέση i μέσα στο πρότυπο P, την ποσότητα $sp_i'(P)$ ως το μήκος του μεγαλύτερου γνήσιου επιθέματος του P[1..i] που ταυτίζεται με ένα πρόθεμα του P, και για το οποίο οι χαρακτήρες P(i+1) και $P(sp_i'+1)$ είναι διαφορετικοί. Στο παραπάνω παράδειγμα λοιπόν

sp_1'=0 και sp_8'=3. Ο αλγόριθμος λοιπόν, μετά από μια προεπεξεργασία ανάλογη του μεγέθους του προτύπου για να ανακτήσει τις τιμές sp_i' για όλες τις θέσεις, κάνει τις ολισθήσεις ως εξής. Αν σε κάποιο σημείο του κυλιόμενου παραθύρου, κατά τις συγκρίσεις από τα αριστερά προς τα δεξιά, υπάρξει μια αποτυχημένη σύγκριση μεταξύ των θέσεων i+1 του προτύπου και της θέσης k του κειμένου, γίνεται ολίσθηση προς τα δεξιά (με σημείο αναφοράς το κείμενο T) ώστε η υποακολουθία P[1.. sp_i'] να στοιχηθεί με την T[k- sp_i'..k-1], γίνονται δηλαδή ακριβώς i+1-(sp_i'+1)=i- sp_i' ολισθήσεις προς τα δεξιά, και ο χαρακτήρας P(sp_i'-1) στοιχίζεται με τον χαρακτήρα T(k). Αν υπάρξει πλήρης ταύτιση, δηλαδή δεν υπάρξει κάποια αποτυχημένη σύγκριση, τότε το P ολισθαίνει m- sp_m' θέσεις. Η ολίσθηση αυτή εξασφαλίζει ότι το πρόθεμα P[1.. sp_i'] του ολισθημένου P ταυτίζεται με την αντίστοιχη απέναντι υποακολουθία του T. Η επόμενη σύγκριση λοιπόν γίνεται μεταξύ των χαρακτήρων T(k) και P(sp_i'-1). Το όφελος λοιπόν είναι διπλό, το κυλιόμενο παράθυρο ολισθαίνει πάνω από μία φορά, και επιπλέον η αναζήτηση θα γίνει μόνο σε ένα κλάσμα των χαρακτήρων του προτύπου. Αφού το κυλιόμενο παράθυρο ποτέ δεν ολισθαίνει αριστερά, σε κάθε νέα ολίσθηση, οι συγκρίσεις περιλαμβάνουν το πολύ έναν χαρακτήρα που είχε συγκριθεί και στην προηγούμενη θέση του κυλιόμενου παραθύρου. Η διπλή αυτή σύγκριση γίνεται κάθε φορά που δεν υπάρχει πλήρης ταύτιση. Συνεπώς, ο αριθμός των συγκρίσεων φράσσεται από το n+s όπου s είναι ο αριθμός των ολισθήσεων που γίνονται από τον αλγόριθμο. Αλλά οι ολισθήσεις δεν μπορούν να ξεπερνούν το n, αφού μετά από n ελάχιστες ολισθήσεις, δηλαδή ολισθήσεις μίας θέσεις, το τέλος του P θα βρίσκεται πάντα δεξιότερα του τέλους του T. Άρα, ο αριθμός των συγκρίσεων είναι μικρότερος από 2n.

Μια άλλη μέθοδος για την αναζήτηση προτύπων είναι αυτή της χρήσης ενός Ντετερμινιστικού Πεπερασμένου Αυτόματου (DFA) [CH97] που μπορεί να αναγνωρίζει την γλώσσα Σ*x. Η αναζήτηση του x στο y ανάγεται στον συντακτικό έλεγχο του y με την χρήση του DFA ξεκινώντας από την αρχική κατάσταση του Αυτομάτου. Κάθε άφιξη στην τερματική κατάσταση ισοδυναμεί με μία εύρεση του προτύπου. Η κατασκευή του Αυτομάτου γίνεται σε χώρο O(mσ) και χρόνο O(m+σ) ενώ η φάση της αναζήτησης έχει πολυπλοκότητα χρόνου O(n). Αν έχω για παράδειγμα το κείμενο 'GCATCGCAGAGAGTATACAGTACG' η αναζήτηση με χρήση αυτομάτου κάποιου προτύπου θα γίνει με βάση το διάγραμμα καταστάσεων που φαίνεται στο σχήμα. Κάθε κατάσταση έχει τον αριθμό του προθέματος του κειμένου στο οποίο αντιστοιχεί, ενώ οι μεταβάσεις που απουσιάζουν είναι αυτές που οδηγούν στην αρχική κατάσταση (κατάσταση 0).



Ο αλγόριθμος Apostolico-Chorchermore [AC91] βελτιώνει περαιτέρω το άνω φράγμα των συγκρίσεων, έχει χωρική και χρονική πολυπλοκότητα προεπεξεργασίας O(m), χρονική πολυπλοκότητα αναζήτησης O(n) και στην χειρότερη περίπτωση αριθμό συγκρίσεων $\frac{3}{2}n$. Ο αλγόριθμος Not-so-Naive στην χειρότερη περίπτωση έχει πολυπλοκότητα O(mn) ωστόσο στη μέση περίπτωση είναι αμυδρά υπογραμμικός.

Ο αλγόριθμος Boyer-Moore [BM77] θεωρείται ένας από τους πιο αποδοτικούς αλγόριθμους για τις συνήθεις εφαρμογές, και για αυτό το λόγο χρησιμοποιείται κατεξοχήν σε εφαρμογές επεξεργαστών κειμένου για λειτουργίες αναζήτησης. Πραγματοποιεί τις συγκρίσεις σε ένα παράθυρο από τα δεξιά προς τα αριστερά και όταν η ταύτιση αποτύχει σε κάποια σύγκριση κάνει έξυπνη ολίσθηση στα δεξιά με χρήση δύο προϋπολογισμένων συναρτήσεων. Η προεπεξεργασία έχει πολυπλοκότητα χώρου και χρόνου O(m+σ). Στην χειρότερη περίπτωση έχει πολυπλοκότητα αναζήτησης O(mn) ,ωστόσο όταν αναζητά μη περιοδικά πρότυπα έχει άνω όριο 3n συγκρίσεις, ενώ στην καλύτερη περίπτωση έχει χρονική πολυπλοκότητα αναζήτησης $O\left(\frac{n}{m}\right)$ που είναι το καλύτερο κάτω όριο για αλγορίθμους χωρίς προεπεξεργασία του κειμένου.

Οι δύο προϋπολογισμένες συναρτήσεις που χρησιμοποιούνται για να κάνουν την ολίσθηση του δένδρου προς τα δεξιά βασίζονται σε δύο κανόνες:

- στον κανόνα του ‘κακού’ χαρακτήρα και
- στον κανόνα του ‘καλού’ επιθέματος.

Ο κανόνας του 'κακού' χαρακτήρα: Για να γίνει κατανοητός ο κανόνας του κακού χαρακτήρα ας υποτεθεί ότι το τελευταίος, δηλαδή ο δεξιότερος χαρακτήρας του προτύπου P , έστω y, είναι στοιχισμένος απέναντι από τον χαρακτήρα x του κειμένου T. Αφού η σύγκριση

ξεκινάει από τα δεξιά προς τα αριστερά, η πρώτη σύγκριση που θα γίνει θα είναι αποτυχημένη. Όταν συμβεί λοιπόν αυτή η αποτυχημένη σύγκριση, η θέση όπου βρίσκεται η δεξιότερη εμφάνιση του χαρακτήρα x μέσα στο πρότυπο P είναι γνωστή, είναι δυνατόν να γίνει ολίσθηση του προτύπου προς τα δεξιά έτσι ώστε το δεξιότερο x μέσα στο πρότυπο να βρίσκεται ακριβώς κάτω από το x στο T όπου έγινε η αμέσως προηγούμενη σύγκριση. Οποιαδήποτε μικρότερη ολίσθηση θα οδηγούσε σε άμεση αποτυχημένη σύγκριση, αφού ο χαρακτήρας x θα συγκρινόταν με κάποιον διαφορετικό του. Άρα η ολίσθηση αυτή είναι ορθή. Επιπλέον, αν ο χαρακτήρας x δεν απαντάται καμία φορά μέσα στο P, μπορεί να γίνει πλήρης ολίσθηση του P μετά το σημείο όπου συνέβη η αποτυχημένη σύγκριση με το T.

Για να τυποποιηθεί αυτή η διαπίστωση ορίζεται για κάθε σύμβολο x του αλφαβήτου η απόσταση R(x) όπου απαντάται ο δεξιότερος χαρακτήρας x μέσα στο πρότυπο. Το R(x) ορίζεται ως μηδέν αν το x δεν απαντάται μέσα στο P. Είναι εύκολο μετά από μια γραμμική προεπεξεργασία του προτύπου να ανακτηθούν οι R(x) τιμές. Στην συνέχεια αυτές οι τιμές χρησιμοποιούνται για την εφαρμογή του κανόνα του 'κακού' χαρακτήρα. Ας υποτεθεί ότι για μια συγκεκριμένη στοίχηση του P απέναντι από το T, δηλαδή για μία συγκεκριμένη θέση του κυλιόμενου παραθύρου, οι δεξιότεροι m-i χαρακτήρες του P ταυτίζονται με τους αντίστοιχους χαρακτήρες του T, αλλά ο επόμενος χαρακτήρας προς τα αριστερά είναι διαφορετικός από τον αντίστοιχο του, έστω στην θέση k του T. Ο κανόνας του 'κακού' χαρακτήρα λέει ότι μόλις η σύγκριση φθάσει σε αυτόν τον χαρακτήρα και αποτύχει, το πρότυπο P πρέπει να ολισθήσει max[1,i-R(T(k))] θέσεις. Δηλαδή αν η δεξιότερη εμφάνιση του χαρακτήρα T(k) στο P βρίσκεται στη θέση που είναι αριστερότερα από το i, συμπεριλαμβανομένης της πιθανότητας ότι j=0, δηλαδή ο χαρακτήρας να μην υπάρχει καθόλου μέσα στο πρότυπο ή να βρίσκεται δεξιά του i, τότε γίνεται ολίσθηση του P ώστε ο χαρακτήρας j του P να βρίσκεται ακριβώς από κάτω από τον χαρακτήρα k του T. Αλλιώς, γίνεται ολίσθηση κατά 1 χαρακτήρα.

Ο κανόνας του κακού χαρακτήρα είναι μια χρήσιμη ευρετική μέθοδος όταν οι συγκρίσεις αποτυγχάνουν κοντά στο δεξιό τέλος του P, αλλά δεν έχει καμία αποτελεσματικότητα όταν ο χαρακτήρας του T στην θέση k που δεν ταυτίστηκε με τον αντίστοιχο του P, βρίσκεται μέσα στο P σε κάποια θέση δεξιότερα του σημείου της αποτυχημένης ταύτισης, δηλαδή δεξιά του i. Αυτό είναι σύνηθες όταν το αλφάβητο είναι μικρό και το κείμενο περιέχει πολλές παρόμοιες, αλλά όχι ίδιες υποακολουθίες. Αυτή η κατάσταση είναι πολύ συνηθισμένη για το DNA, που έχει σ=4, αλλά ακόμα και για τις πρωτεΐνες, που έχουν μεγέθος αλφαβήτου σ=20 (όσα και τα αμινοξέα), αφού συχνά αποτελούνται από περιοχές μεγάλης ομοιότητας.

Ο επεκταμένος κανόνας του κακού χαρακτήρα είναι αρκετά πιο αποτελεσματικός. Δεν αναζητεί την δεξιότερη εμφάνιση του x στο P, αλλά την εγγύτερη εμφάνιση του x στα αριστερά του i και κάνει την ολίσθηση ανάλογα. Αναζητεί δηλαδή, την δεξιότερη εμφάνιση

που είναι στα αριστερά του i, και όχι την δεξιότερη εμφάνιση γενικά μέσα στο πρότυπο. Ο επεκταμένος κανόνας του κακού χαρακτήρα οδηγεί πάντα σε μεγαύτερου μήκους ολισθήσεις επομένως ο μόνος λόγος να μην εφαρμοστεί είναι το κόστος της υλοποίησης κατά την επεξεργασία, σε σχέση με τον απλό κανόνα του κακού χαρακτήρα.

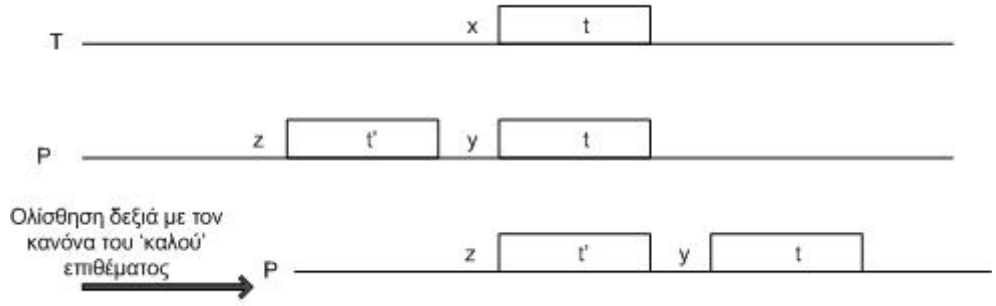
Η προεπεξεργασία για την μετέπειτα εφαρμογή του επεκταμένου κανόνα του κακού χαρακτήρα πρέπει να βρει για κάθε θέση μέσα στο P και για κάθε σύμβολο x του αλφαβήτου, την θέση της εγγύτερης θέσης του x στα αριστερά του i. Η πιο προφανής προσέγγιση για να γίνει αυτό είναι με τη χρήση ενός διδιάστατου πίνακα με διαστάσεις m επί σ, που θα αποθηκεύει αυτές τις πληροφορίες. Αν το αλφάβητο είναι σχετικά μικρό αυτή η προσέγγιση είναι αρκετά αποδοτική στην κατασκευή και στον χώρο που χρησιμοποιεί, και η πληροφορία ανακτάται σε σταθερό χρόνο από τον διδιάστατο πίνακα. Αν το αλφάβητο είναι πολύ μεγάλο τότε για να αποφευχθεί η δημιουργία ενός τεράστιου διδιάστατου αραιού πίνακα στη μνήμη μπορούν να χρησιμοποιηθούν λίστες που να αποθηκεύουν τις θέσεις που εμφανίζονται τα διάφορα σύμβολα, ώστε σύμβολα που δεν εμφανίζονται καθόλου μέσα στο πρότυπο να μην καταλαμβάνουν καθόλου χώρο στην μνήμη. Αυτό επιβαρύνει βέβαια τον χρόνο εκτέλεσης του αλγορίθμου καθώς υπάρχει το επιπλέον κόστος της αναζήτησης εντός των λιστών, ωστόσο η επιβάρυνση δεν είναι σημαντική στην μεση περίπτωση.

Ο κανόνας του 'καλού' επιθέματος: Ο κανόνας του ‘κακού’ χαρακτήτα είναι πολύ αποτελεσματικός στην πράξη ειδικά για κείμενα φυσικής γλώσσας, ωστόσο αποδεικνύεται λιγότερο αποτελεσματικός για μικρά αλφάβητα και δεν οδηγεί σε γραμμικό χρόνο αναζήτησης στην χείριστη περίπτωση. Γι’αυτό το λόγο υπάρχει ένας άλλος κανόνας για τις ολισθήσεις, ο κανόνας του ‘καλού’ επιθέματος. Ας υποτεθεί ότι ότι για κάποια στοίχηση του P και του T, δηλαδή για κάποια συγκεκριμένη θέση του κυλιόμενου παραθύρου της αναζήτησης, μια υποακολουθία t του κειμένου ταυτίζεται με ένα επίθεμα του P, αλλά υπάρχει μια αποτυχημένη σύγκριση στην αμέσως επόμενη θέση αριστερά. Βρίσκεται, αν υπάρχει, το δεξιότερο αντίγραφο του t στο P, έστω t’, τέτοιο ώστε το t’ δεν είναι επίθεμα του P (δεν είναι δηλαδή στην ίδια θέση με το t) και ο χαρακτήρας στα αριστερά του t’ στο P διαφέρει από τον χαρακτήρα στα αριστερά του P στο t. Αν η υποακολουθία t’ υπάρχει, το P ολισθαίνει δεξιά ώστε η υπακολουθία t’ στο P να είναι κάτω από την υποακολουθία t στο T. Αν η υποακολουθία t’ δεν υπάρχει, ολισθαίνει το αριστερό άκρο του P πέραν του αριστερού άκρου του t στο T στον ελάχιστο βαθμό που ένα πρόθεμα του ολισθημένου προτύπου ταιριάζει με ένα επίθεμα του t στο T. Αν μία τέτοια ολίσθηση δεν είναι δυνατή, τότε το P ολισθαίνει m θέσεις προς τα δεξιά. Κάθε φορά που υπάρχει μία πλήρης ταύτιση του P μέσα στο T το P ολισθαίνει στον ελάχιστο βαθμό ώστε ένα γνήσιο πρόθεμα του ολισθημένου P να ταυτίζεται με ένα επίθεμα της προηγούμενης πλήρους ταύτισης πάνω στο T. Αν μια τέτοια ολίσθηση δεν είναι δυνατή τότε το πρότυπο ολισθαίνει πλήρως, δηλαδή m θέσεις προς τα δεξιά. Η αρχική

περιγραφή του αλγορίθμου Boyer-Moore πρότεινε μία πιο ασθενή εκδοχή για τον κανόνα του ‘καλού’ επιθέματος. Αυτή η εκδοχή απαιτεί το ολισθημένο πρότυπο να συμφωνεί με το t αλλά χωρίς να προσδιορίζει ότι ο επόμενος χαρακτήρας στα αριστερά του t δεν ταυτίζεται. Και στην ισχυρή και στην ασθενή εκδοχή του κανόνα του ‘καλού’ επιθέματος αποδεικνύεται ότι η χρήση του ποτέ δεν ολισθαίνει το P πέρα από μία ταύτιση του μέσα στο T. Με την χρήση της ασθενέστερης εκδοχής πάντως δεν μπορεί να αποδειχθεί η γραμμική πολυπλοκότητα του αλγορίθμου για την χείριστη περίπτωση αν και στην γενική περίπτωση ο αλγόριθμος είναι γραμμικός.

Ο κανόνας του ‘καλού’ επιθέματος στερείται βέβαια της ευλογοφάνειας που έχει ο κανόνας του ‘κακού’ χαρακτήρα ώστοσο με μία προεπεξεργασία πολυπλοκότητας O(m) ο αλγόριθμος αποθηκεύει όλες τις απαραίτητες πληροφορίες ώστε να απαντά γρήγορα στα κριτήρια του κανόνα κατά την διάρκεια της αναζήτησης.

Επειδή οι δύο αυτοί κανόνες προτείνουν διαφορετικά μήκη ολισθήσεων, ο αλγόριθμος υπολογίζει τις ολισθήσεις και από τις δύο προϋπολογισμένες συναρτήσεις και στην συνέχεια πραγματοποιεί την μεγαλύτερη σε μήκος ολίσθηση.



Υπάρχει πλήθος παραλλαγών και βελτιώσεων του Boyer-Moore για συγκεκριμένα προβλήματα, ο αλγόριθμος Quick Search [Sun90] με βέλτιστη απόδοση για μικρά πρότυπα και μεγάλα αλφάβητα (όχι κατάλληλος για βιολογικά δεδομένα), ο Turbo BM [CCG+91], ο Reverse Colussi [Col94], ο Apostolico-Giancarlo [AG86], ο Reverse Factor [Lec92], ο Berry-Ravindran [BR99]κ.ά.

Ο αλγόριθμος Galil-Seiferas [GS83] διαμερίζει το πρότυπο, και αναζητεί το αριστερό μισό ενώ συνεχίζει με το δεξί αν η αναζήτηση του αριστερού δεν αποτύχει. Η προεπεξεργασία έχει πολυπλοκότητα O(m) ενώ η αναζήτηση O(n) και στην χειρότερη περίπτωση πραγματοποιούνται 5n συγκρίσεις. Υπάρχουν και άλλοι αλγόριθμοι όπου η σειρά με την οποία γίνονται οι συγκρίσεις είναι επιλεγμένη, όπως ο KMP Skip and Search [CLP98] που είναι παραλλαγή του Knuth-Morris-Pratt, ο Optimal Mismatch [Sun90], ο Maximal Shift [Sun90], κ.ά.

Τέλος υπάρχουν αλγόριθμοι των οποίων η απόδοση είναι ανεξάρτητη της σειράς με την οποία γίνονται οι συγκρίσεις, όπως ο Quick Search που ήδη αναφέρθηκε, ο Horspool [Hor80]

που είναι επίσης παραλλαγή του Boyer-Moore, ο Tuned Boyer-Moore [HS91], ο αλγοριθμος του Smith [Smi91] κ.ά.

Παρά την πολύ σημαντική πρόοδο που έχει σημειωθεί με αυτούς τους αλγορίθμους, οι περισσότεροι είναι στην μέση περίπτωση γραμμικοί, πρέπει δηλαδή να πραγματοποιήσουν συγκρίσεις που είναι αντίστοιχες του μεγέθους του κειμένου, συνεπώς για πολύ μεγάλες ακολουθίες όπως είναι αυτές που αποτελούνται από βιολογικά δεδομένα, φαίνεται αρκετά ασύμφορη η αναζήτηση προτύπων με αυτόν τον τρόπο, δίχως δηλαδή την προεπεξεργασία του κειμένου για παραγωγή ευρετηρίου που θα χρησιμοποιηθεί για την αποδοτική αναζήτηση.

2.2.1.2 Αλγόριθμοι με προεπεξεργασία του κειμένου

Όπως αναφέρθηκε και προηγουμένως, αν έχει κατασκευαστεί πάνω στο κείμενο κάποιου είδους ευρετήριο (μέσω κάποιας φάσης προεπεξεργασίας), μπορούν να υπάρξουν πιο αποδοτικοί αλγόριθμοι για ακριβή εύρεση προτύπου. Ευρετήρια όπως τα ανεστραμμένα αρχεία (inverted files) που συνηθίζονται στις αναζητήσεις κειμένου φυσικής γλώσσας, δεν είναι κατάλληλα για την περίπτωση βιολογικών κειμένων, όπως το γονιδίωμα, επειδή τέτοιου είδους κείμενα δεν χωρίζονται σε λέξεις. Τα πιο συνηθισμένα ήδη ευρετηρίων που χρησιμοποιούνται σε αυτές τις περιπτώσεις είναι οι πίνακες και τα δένδρα επιθεμάτων (suffix arrays & suffix trees).

Ένα οποιοδήποτε κείμενο y μήκους m έχει ακριβώς m επιθέματα, θεωρώντας ότι και ολόκληρη η ακολουθία του κειμένου είναι επίθεμα. Το κάθε επίθεμα μπορεί να αριθμηθεί στην βάση ενός αύξοντα αριθμού i, ώστε το εκάστοτε αριθμημένο επίθεμα να είναι η υποακολουθία y[i..m]. Χρησιμοποιώντας το κλασσικό παράδειγμα της ακολουθίας ‘abracadabra’, η αριθμηση των επιθεμάτων είναι αυτή που φαίνεται στον παρακάτω πίνακα.

Κείμενο	a	b	r	a	c	a	d	a	b	r	a
Δείκτης	1	2	3	4	5	6	7	8	9	10	11

Ο αντίστοιχος πίνακας που θα περιέχει τα επιθέματα της συγκεκριμένης ακολουθίας είναι λοιπόν:

Επίθεμα	Δείκτης
abracadabra	1
bracadabra	2
racadabra	3

acadabra	4
cadabra	5
adabra	6
dabra	7
abra	8
bra	9
ra	10
a	11

Η ιδέα για την ευρετηριοποίηση με βάση τον πίνακα επιθεμάτων συνίσταται στην ταξινόμηση του προηγούμενου πίνακα με βάση τη λεξικογραφική σειρά των επιθεμάτων. Ο πίνακας που παράγεται είναι ο «πίνακας επιθεμάτων». Στο προηγούμενο παράδειγμα, ο αντίστοιχος πίνακας επιθεμάτων είναι ο ακόλουθος:

Επίθεμα	Δείκτης
a	11
abra	8
abracadabra	1
acadabra	4
Adabra	6
Bra	9
Bracadabra	2
Cadabra	5
Dabra	7
Ra	10
Racadabra	3

Η αναζήτηση κάποιου προτύπου, μεταφράζεται μετά από αυτό σε δυαδική αναζήτηση (binary search) μέσα σε αυτόν τον πίνακα.

Γίνεται φανερό ότι μετά την ταξινόμηση, δεν είναι απαραίτητο να αποθηκεύεται το σύνολο των χαρακτήρων των επιθεμάτων αφού όταν στην μνήμη υπάρχει η αρχική ακολουθία και ο αύξοντα αριθμός του επιθέματος μπορεί αμέσως να ανακτηθεί το επίθεμα. Ο πίνακας επιθεμάτων λοιπόν που θα χρησιμοποιηθεί ως ευρετήριο για το κείμενο είναι ο παρακάτω:

11	8	1	4	6	7	2	5	7	10	3

Η δυαδική αναζήτηση για κάποιο πρότυπο μεγέθους m θα πρέπει να αναζητά το πρότυπο πραγματοποιώντας όμως τις συγκρίσεις μόνο για τους πρώτους m χαρακτήρες των επιθεμάτων, αφού ένα πρότυπο είναι υποακολουθία του κειμένου αν και μόνον αν είναι πρόθεμα κάποιου επιθέματος, ενώ το πρόθεμα δύναται να είναι μη γνήσιο, να είναι δηλαδή ολόκληρο το επίθεμα. Αφού η αναζήτηση επιστρέψει την θέση του επιθέματος που βρέθηκε με την κλασσική μέθοδο, πρέπει να ελέγξει τα γειτονικά επιθέματα του πίνακα επιθεμάτων προς τα δεξιά και προς τα αριστερά για τα οποία η σύγκριση των πρώτων m χαρακτήρων είναι επιτυχής ώστε να επιστραφούν εξαντλητικά όλες οι θέσεις στο κείμενο όπου εντοπίζεται το πρότυπο. Αν η αναζήτηση προς τα αριστερά ή τα δεξιά αποτύχει σε κάποια θέση, τότε σταματάει η αναζήτηση προς αυτή την κατεύθυνση καθώς είναι αδύνατο λόγω της ταξινόμησης να υπάρξει επιτυχία σε επόμενη θέση.

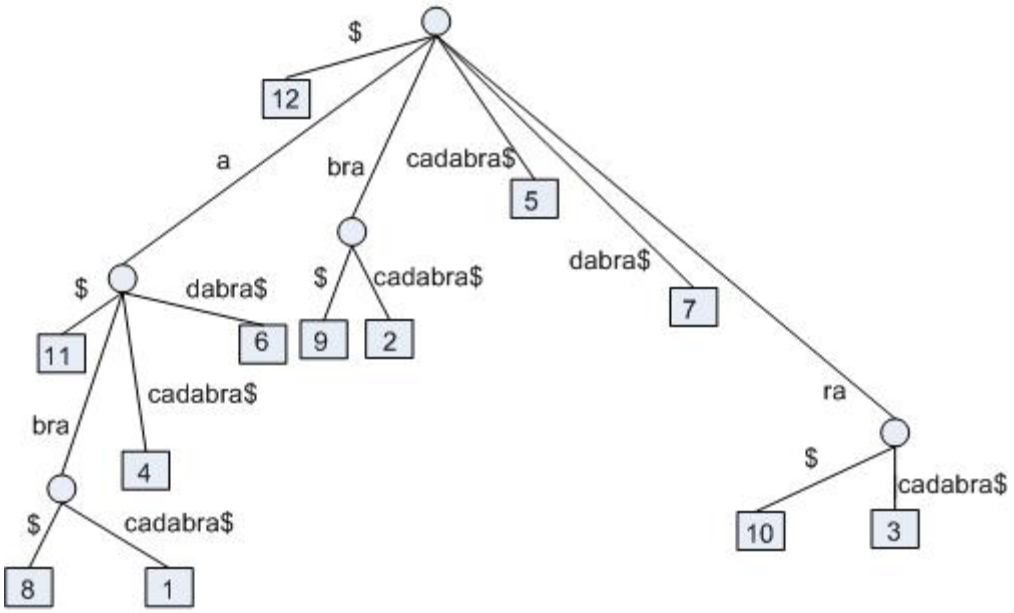
Αφού γενικά το πρόβλημα της ταξινόμησης είναι χρονικής πολυπλοκότητας Θ(nlogn) και η δυαδική αναζήτηση έχει πολυπλοκότητα O(logn), η αναζήτηση προτύπων με τον πίνακα επιθεμάτων έχει προεπεξεργασία χρόνου O(nlogn) και χώρου O(1) (heapsort), και πολυπλοκότητα στη φάση της αναζήτησης O(mlogn) αφού σε κάθε σύγκριση ακολουθιών γίνονται το πολύ m συγκρίσεις. Ο αλγόριθμος αυτός είναι δηλαδή αποδοτικότερος από όλους τους αλγορίθμους ακριβούς ταύτισης προτύπου χωρίς προεπεξεργασία του κειμένου.

Μια άλλη δομή που χρησιμοποιείται επίσης για την ευρετηριοποίηση ακολουθιών είναι το δένδρο επιθεμάτων. Το δένδρο επιθεμάτων ορίζεται ως ένα δένδρο που έχει ρίζα, και κατευθυνόμενες ακμές, και περιέχει ακριβώς n φύλλα αριθμημένα από το 1 ως το n. Κάθε ακμή έχει μια ετικέτα που αντιστοιχεί σε κάποια υπάρχουσα υποακολουθία του κειμένου. Το φύλλο i του δένδρου επιθεμάτων αντιστοιχεί στο επίθεμα της αντίστοιχης ακολουθίας-κειμένου που ξεκινά από τη θέση i. Ο συνδυασμός των ετικετών που βρίσκονται πάνω στο μονοπάτι που ξεκινά από τη ρίζα του δένδρου και καταλήγει στο φύλλο i είναι η ακολουθία του συγκεκριμένου επιθέματος. Κάθε εσωτερικός κόμβος έχει τουλάχιστον δύο παιδιά. Δυο ακμές που εξέρχονται από κάποιον κόμβο δεν μπορούν να έχουν ετικέτα που να ξεκινά με τον ίδιο χαρακτήρα.

Ένα πρόβλημα που δημιουργείται είναι ότι αν κάποιο επίθεμα του S είναι πρόθεμα κάποιου άλλου μεγαλύτερου σε μήκος επιθέματος τότε ο παραπάνω ορισμός δεν εξασφαλίζει την ύπαρξη ακριβώς n φύλλων. Αυτό συμβαίνει γιατί το μονοπάτι που οδηγεί στο επίθεμα αυτό δεν θα κατέληγε σε φύλλο, αλλά σε κάποιον εσωτερικό κόμβο. Για να επιλυθεί αυτό το πρόβλημα γίνεται η υπόθεση ότι το y διαθέτει κάποιον τερματικό χαρακτήρα ο οποίος δεν απαντά σε καμία άλλη θέση της ακολουθίας και ο οποίος στα βιολογικά δεδομένα κατά σύμβαση συμβολίζεται με $. Προσθέτουμε δηλαδή στο τέλος της ακολουθίας-κειμένου τον χαρακτήρα Με την προσθήκη αυτή το δένδρο επιθεμάτων θα έχει πλέον ακριβώς n+1 φύλλα που θα κωδικοποιούν όλα του τα επιθέματα, συμπεριλαμβανομένου και του $. Το πλήθος των

χαρακτήρων της υποακολουθίας μιας ακμής που καταλήγει σε κάποιον κόμβο ονομάζεται βάθος ακολουθίας.

Σύμφωνα με τα παραπάνω για να ευρετηριοποιηθεί το κείμενο ‘abracadabra’ θα πρέπει να προστεθεί ο χαρακτήρας $ στο τέλος του και στη συνέχεια να κατασκευαστεί το μονοπάτι από τη ρίζα ως τα φύλλα για όλα τα επιθέματα τηρώντας τους περιορισμούς του ορισμού του δένδρου επιθεμάτων. Στο παρακάτω σχήμα φαίνεται το δένδρο επιθεμάτων για το κείμενο αυτό.



Κάθε αναζήτηση αντιστοιχεί με μια διάσχιση του δένδρου, η οποία ξεκινά από τη ρίζα. Η διάσχιση γίνεται με βάση την ακολουθία-πρότυπο: Ξεκινώντας από την αρχή της ακολουθίας-προτύπου, διαβάζουμε έναν-έναν τους χαρακτήρες της. Για κάθε χαρακτήρα του προτύπου που διαβάζουμε, ελέγχουμε να δούμε αν αυτός υπάρχει ακριβώς μετά από το σημείο που είμαστε στο δένδρο κατά την τρέχουσα στιγμή. Αν είμαστε σε κάποιον κόμβο του δένδρου, ελέγχουμε τον πρώτο χαρακτήρα για κάθε εξερχόμενη ακμή. Αν είμαστε στα μισά της ετικέτας κάποιας ακμής, απλώς ελέγχουμε τον επόμενο χαρακτήρα. Αν σε κάποιο σημείο διαβάσουμε κάποιον χαρακτήρα από το πρότυπο και δεν μπορούμε να τον ταιριάσουμε, τότε αυτό θα πει ότι το συγκεκριμένο πρότυπο δεν υπάρχει στο κείμενο του δένδρου επιθεμάτων. Αν καθώς διασχίζουμε το δένδρο διαβάσουμε όλο το πρότυπο, τότε το σημείο στο οποίο έχουμε σταματήσει τη διάσχιση, ορίζει ένα υποδένδρο, στο οποίο όλα τα φύλλα είναι οι θέσεις στις οποίες εμφανίζεται το επιθυμητό πρότυπο στο κείμενο. Ας υποθέσουμε ότι με χρήση του δένδρου επιθεμάτων του προηγούμενου παραδείγματος θέλουμε να αναζητήσουμε το πρότυπο ‘bra’ στο κείμενο ‘abracadabra’. Αρχικά προσπελαύνουμε τη ρίζα και ψάχνουμε να βρούμε την ακμή που έχει ως ετικέτα μια υποακολουθία με πρώτο χαρακτήρα το ‘b’. Αν

αυτή δεν υπάρχει τότε το πρότυπο δεν υπάρχει μέσα στην ακολουθία. Στο παράδειγμα όμως υπάρχει όντως ακμή που ξεκινάει με b, και έχει ως ετικέτα την υποακολουθία bra. Συγκρίνουμε χαρακτήρα προς χαρακτήρα το πρότυπο με την υποακολουθία της ακμής. Στο παράδειγμα μας η σύγκριση είναι επιτυχής και εξανλούνται και οι χαρακτήρες του προτύπου. Άρα η αναζήτηση ολοκληρώνεται σε εσωτερικό κόμβο και οι θέσεις στην αρχική ακολουθία είναι οι δείκτες όλων των φύλλων απογόνων του εσωτερικού κόμβου στον οποίο βρισκόμαστε, δηλαδή οι θέσεις 2 και 9. Αντίστοιχα αν αναζητούσαμε την υποακολουθία 'abra' θα φθάναμε από τη ρίζα στον επόμενο κόμβο μέσω της ακμής που έχει ως ετικέτα το a. Συγκρίνουμε χαρακτήρα προς χαρακτήρα την υποακολουθία της ακμής και του προτύπου δηλαδή στην εκφυλισμένη αυτή περίπτωση μόνο τον χαρακτήρα a. Η σύγκριση τελειώνει χωρίς να εξαντληθούν οι χαρακτήρες του προτύπου αφού απομένουν οι τρεις τελευταίοι χαρακτήρες δηλαδή το 'bra'. Εφαρμόζουμε αναδρομικά την ίδια διαδικασία όπως και για τη ρίζα, στον εσωτερικό αυτό κόμβο, αναζητούμε δηλαδή την ακμή που εξέρχεται από τον κόμβο και ξεκινάει με 'b', ο οποίος υπάρχει και έχει ως ετικέτα την υποκολουθία bra. Η αναζήτηση τελειώνει πάλι σε εσωτερικό κόμβο και έχει ως αποτέλεσμα τις θέσεις 1 και 8 όπου βρίσκεται η υποακολουθία 'abra'. Αν η υποακολουθία που αναζητούμε απαντά μόνο μία φορά στο κείμενο τότε η αναζήτηση θα σταματήσει σε κάποια ακμή που καταλήγει σε φύλλο.

Είναι προφανές ότι η αναζήτηση κάποιου προτύπου έχει πολυπλοκότητα O(m), ανεξάρτητη δηλαδή από το μέγεθος της ακολουθίας, και άρα πολύ καλύτερη όλων των μεθόδων χωρίς προεπεξεργασία του κειμένου, αλλά και των (απλών) πινάκων επιθεμάτων, γεγονός που καθιστά το δένδρο επιθεμάτων ένα εξαιρετικό ευρετήριο για αναζήτηση προτύπων. Η υποακολουθία που κωδικοποιεί κάθε ακμή δεν χρειάζεται να αποθηκεύεται στο δένδρο, αφού μπορεί να παρασταθεί με δύο δείκτες start και end που κωδικοποιούν την υποακολουθία S[start..end], συνεπώς αρκεί η αποθήκευση δύο δεικτών. Για μικρά αλφάβητα όπως αυτό ακολουθιών DNA, για το οποίο $|\Sigma|=5$, (λαμβάνοντας υπόψη και την προσθήκη του τερματικού χαρακτήρα) είναι αρκετά εύκολο να συμβολισθεί κάθε κόμβος του δένδρου ως μια δομή δεδομένων σταθερού μεγέθους με 7 εγγραφές, 2 για την υποακολουθία της προσπίπτουσας ακμής, και κάθε μία από τις άλλες 5 να περιέχει έναν δείκτη σε κόμβους-παιδιά ανάλογα με ποιον από τους 5 χαρακτήρες του αλφαβήτου ξεκινά η υποακολουθία της αντίστοιχης εξερχόμενης ακμής, για την γρήγορη προσπέλαση των κόμβων μέσω της κατάλληλης ακμής. Αν το αλφάβητο είναι αρκετά μεγάλο, μια τέτοια προσέγγιση οδηγεί μάλλον σε μεγάλη σπατάλη χώρου και πρέπει να χρησιμοποιηθούν άλλες στρατηγικές. Ζητήματα αποδοτικής κατασκευής και αποθήκευσης των δένδρων επιθεμάτων θα αναλυθούν στην ενότητα 2.3 και το κεφάλαιο 3.

Γενικά ένας πίνακας επιθεμάτων μπορεί να χαρακτηριστεί ως μια συμπιεσμένη μορφή εν΄ςο δένδρου επιθεμάτων. Ο χώρος που πιάνουν είναι σημαντικά μικρότερος από τα δένδρα επιθεμάτων με αποτέλεσμα να γίνουν αρκετά δημοφιλή. Αυτό το κέρδος σε χώρο όμως συνοδεύεται από ένα κόστος σε χρόνο αναζήτησης. Το 2004 [AKO04] προέκυψαν πολύ ενδιαφέροντα ερευνητικά αποτελέσματα σχετικά με τα δένδρα και τους πίνακες επιθεμάτων. Οι Abouelhoda, Kurtz και Ohlebusch έδειξαν ότι οποιοσδήποτε αλγόριθμος που υλοποιείται με χρήση των δένδρων επιθεμάτων μπορεί να υλοποιηθεί με την ίδια πολυπλοκότητα με χρήση ενός **αναβαθμισμένου πίνακα επιθεμάτων.** Ο αναβαθμισμένος πίνακας επιθεμάτων, είναι ένας πίνακας επιθεμάτων κατά τα γνωστά, με 4 επιπλέον στήλες. Η βασική αλλαγή είναι η προσθήκη της στήλης που αποθηκεύει το μήκος του μέγιστου κοινού προθέματος (longest common prefix-lcp) μεταξύ του επιθέματος της τρέχουσας γραμμής και του επιθέματος της προηγούμενης γραμμής. Οι υπόλοιπες στήλες κωδικοποιούν πληροφορίες για τις σχέσεις γονέων-παιδιών σε ένα virtual δένδρο επιθεμάτων.

2.2.2 Εύρεση της μέγιστης κοινής επέκτασης υποακολουθιών (LCE)

2.2.2.1 Το πρόβλημα – η σημασία

Το πρόβλημα της εύρεσης της μέγιστης κοινής επέκτασης υποακολουθιών είναι ένα κλασσικό πρόβλημα πάνω στα ακολουθιακά δεδομένα με πλήθος χρήσεων σε αρκετούς αλγορίθμους. Αν θεωρήσουμε δύο ακολουθίες S_1 και S_2 και δύο θέσεις i και j σε αυτά αντίστοιχα, το πρόβλημα συνίσταται στην έυρεση του μέγιστου μήκους max_length για την οποία οι υποακολουθίες S_1[i..i+max_length] και S_2[j..j+max_length] ταυτίζονται. ...

Για παράδειγμα, αν S_1=’αβγδεζνξ’ και S_2=’ααβγδεζηθ’ η μέγιστη κοινή επέκταση στην θέση 4 της S_1 και την θέση 5 της S_2 είναι η ακολουθία ‘δεζ’.

Η εύρεση του LCE είναι κεντρική στον αλγόριθμο για την εύρεση παλινδρόμων σε κάποια ακολουθία. Τα παλίνδρομα είναι ακολουθίες που διαβάζονται με τον ίδιο τρόπο είτε από αριστερά προς τα δεξιά είτε από τα δεξιά προς τα αριστερά. Το γενετικό υλικό των θηλαστικών βρίθει συμπληρωματικών παλινδρόμων, ακολουθιών δηλαδή που γίνονται παλίνδρομα αν το μισό τους κομμάτι αντικατασταθεί με τις αντίστοιχες συμπληρωματικές βάσεις. Γενικότερα στα γονιδιώματα, παρατηρούνται πάρα πολλές επαναληπτικές δομές, η

χρησιμότητα και η λειτουργία των οποίων είναι αντικείμενο εντατικής έρευνας και η αποδοτική εύρεση του LCE συμβάλλει στην αποδοτική στατιστική ανάλυση του γονιδιώματος.

Το πρόβλημα του LCE και η αποδοτική του επίλυση είναι επίσης κομβικό για τον αλγόριθμο της ακριβούς ταύτισης προτύπου με χαρακτήρες μπαλαντέρ (exact matching with wildcards) και τον αλγόριθμο k-mismatch που είναι στην ουσία μία παραλλαγή του. Το πρόβλημα της ακριβούς ταύτισης προτύπου με χαρακτήρες μπαλαντέρ, είναι το γνωστό πρόβλημα της αναζήτησης ενός προτύπου σε κάποια ακολουθία όπου όμως το πρότυπο περιέχει χαρακτήρες μπαλαντέρ, χαρακτήρες δηλαδή που αντιπροσωπεύουν οποιονδήποτε χαρακτήρα του αλφαβήτου. Αντίστοιχα, ο k-mismatch κάνει αναζήτηση του προτύπου όπου επιτρέπει ως και k λάθη στις συγκρίσεις των χαρακτήρων, περιέχει δηλαδή, κάνοντας μια αναλογία με το προηγούμενο, ως και k χαρακτήρες μπαλαντέρ αλλά σε οποιαδήποτε θέση. Εκτός από αυτούς τους δύο αλγορίθμους το LCE είναι στον πυρήνα και άλλων αλγορίθμων προσεγγιστικής ταύτισης όπως ο αλγόριθμος υβριδικού δυναμικού προγραμματισμού k-difference που θα αναλυθεί παρακάτω.

2.2.2.2. Η αναγωγή στο LCA

Είναι αρκετά εύκολο να δειχθεί ότι για ακολουθίες που έχουν ευρετηριοποιηθεί υπό μορφή δένδρου επιθεμάτων η εύρεση της μέγιστης κοινής επέκτασης δύο θέσεων της ακολουθίας ανάγεται στην εύρεση του κατώτατου κοινού προγόνου (Lowest Common Ancestor-LCA) δύο φύλλων του δένδρου.

Πράγματι, δύο οποιεσδήποτε θέσεις, έστω i και j, μέσα στην ακολουθία αντιστοιχούν σε δύο φύλλα που κωδικοποιούν τα αντίστοιχα επιθέματα S[i..n] kai S[j..n]. Η ακολουθία που παράγεται αν συνδυαστούν με τη σειρά οι ετικέτες από τις ακμές που συμμετέχουν στη διαδρομή από τη ρίζα ως αντίστοιχο φύλλο είναι το αντίστοιχο επίθεμα. Αν τα επιθέματα έχουν κάποιο κοινό κομμάτι, κατά μήκος του οποίου ταυτίζονται, θα ταυτίζονται και τα αντίστοιχα κομμάτια των μονοπατιών από τη ρίζα. Αν σε κάποια θέση πάψουν να είναι κοινά, τότε τα αντίστοιχα μονοπάτια στο δένδρο γίνονται διακριτά, όπως επιτάσσει ο ορισμός του δένδρου επιθεμάτων. Συνεπώς η υποακολουθία που αντιστοιχεί στον κατώτατο κοινό πρόγονο (που θα είναι πάντοτε κάποιος εσωτερικός κόμβος εκτός αν i=j) είναι η μέγιστη κοινή επέκταση (LCE) των i και j.

Το συμπέρασμα αυτό είναι πολύ χρήσιμο καθώς υπάρχει ένας εξαιρετικός αλγόριθμος που μετά από κάποια γραμμική προεπεξεργασία ενός δένδρου, είναι σε θέση να ανακτά τον LCA οποιωνδήποτε κόμβων του δένδρου σε σταθερό χρόνο. Αν επιπλέον φροντίσουμε σε κάθε κόμβο να έχουμε αποθηκευμένο το μήκος της υποακολουθίας την οποία συμβολίζει, μπορούμε να έχουμε το ζητούμενο μήκος της μέγιστης κοινής επέκτασης δύο θέσεων της

ακολουθίας, και άρα να επιλύουμε το πρόβλημα της εύρεσης του LCE με χρονική πολυπλοκότητα O(1).

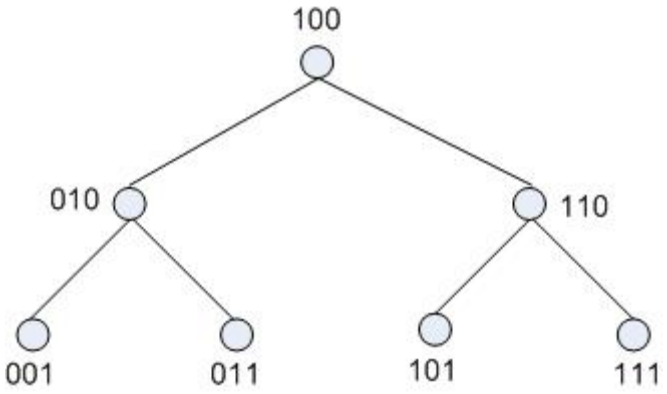
2.2.2.3 Ο αλγόριθμος σταθερού χρόνου για το LCA

Ο αλγόριθμος για την εύρεση του LCA με το αξιοσημείωτο αυτό αποτέλεσμα του σταθερού χρόνου προτάθηκε αρχικά από τους Harel και Tarjan [HT84] και στην συνέχεια απλοποιήθηκε από τους Schieber και Vishkin [SV88]. Η παραδοχή που πρέπει να γίνει για να ισχύει η βέλτιστη αυτή πολυπλοκότητα είναι ότι ο υπολογισμός γίνεται με μία υπολογιστική μηχανή για την οποία ισχύει το μοντέλο μονάδας κόστους RAM (unit-cost RAM model).

Σύμφωνα με το μοντέλο μονάδας κόστους RAM αν η μνήμη έχει χωρητικότητα O(n), συνεπώς χωράει ένα δένδρο n φύλλων, τότε οποιοσδήποτε αριθμός μήκους O(logn) bits μπορεί να γραφθεί, να διαβαστεί και να χρησιμοποιηθεί ως διεύθυνση σε σταθερό χρόνο. Επίσης δύο αριθμοί μήκους O(logn) μπορούν να συγκριθούν, να προσθαφαιρεθούν, να πολλαπλασιαστούν και να διαιρεθούν σε σταθερό χρόνο. Αυτό είναι το στάνταρ μοντέλο μονάδας κόστους RAM (standard unit-cost RAM model), ωστόσο θα θεωρήσουμε ότι και οι λειτουργίες της δυαδικής ολίσθησης και των δυαδικών πράξεων AND, OR και XOR για αριθμούς μήκους O(logn) γίνονται σε σταθερό χρόνο, κάτι που ισχύει μεν για πολλές υπάρχουσες μηχανές διευκολύνει δε την περιγραφή του αλγορίθμου. Ωστόσο, ο αλγόριθμος εξακολουθεί με κάποιες τροποποιήσεις να έχει το ίδιο αποτέλεσμα σταθερού χρόνου και για το στάνταρ μοντέλο μονάδας κόστους RAM.

Προκειμένου να ισχύει η εύρεση LCA σε σταθερό χρόνο, θα πρέπει το δένδρο να υποστεί μια κατάλληλη προεπεξεργασία. Η γραμμική προεπεξεργασία του αλγορίθμου αποβλέπει στην απεικόνιση των κόμβων του δένδρου στους κόμβους ενός άλλου υποθετικού δυαδικού δένδρου. Στην συνέχεια το πρόβλημα της εύρεσης του LCA ανάγεται στην εύρεση του LCA των απεικονισμένων κόμβων στο δυαδικό δένδρο B.

Ας υποθέσουμε ότι έχουμε ένα πλήρες δυαδικό δένδρο B, με p φύλλα. Αφού το δένδρο είναι πλήρες, όλα τα φύλλα έχουν το ίδιο βάθος από τη ρίζα, $d=\log_2 p$. Σε κάθε κόμβο υ αναθέτουμε έναν δυαδικό αριθμό μήκους d+1 bits ο οποίος κωδικοποιεί το μοναδικό μονοπάτι από τη ρίζα μέχρι τον κόμβο. Ξεκινώντας από το πιο αριστερό bit, ο ν-οστό bit του αριθμού αυτού αντιστοιχεί στην ν-οστή ακμή στο μονοπάτι από τη ρίζα ως τον υ. Ένα 0 στο ν-οστό bit υποδεικνύει ότι η ν-οστή ακμή του μονοπατιού καταλήγει σε ένα αριστερό παιδί, ενώ το 1 υποδεικνύει μια ακμή σε ένα παιδί δεξιά. Για παράδειγμα, ένα μονοπάτι που πάει δύο φορές αριστερά, μια φορά δεξιά και μετά πάλι αριστερά, καταλήγει σε έναν κόμβο ο αριθμός μονοπατιού του οποίου ξεκινάει με 0010. Ο κάθε αριθμός μονοπατιού συμπληρώνεται με ένα 1 και με όσα μηδενικά χρειάζονται για να φθάσει τον αριθμό των d+1 bits. Ένα παράδειγμα πλήρους δυαδικού δένδρου με την παραπάνω αρίθμηση των κόμβων φαίνεται στο σχήμα.



Αν θελήσουμε να βρούμε τον LCA δύο κόμβων σε ένα τέτοιο δυαδικό δένδρο, θα πρέπει να πραγματοποιήσουμε την δυαδική πράξη XOR μεταξύ των αριθμών μονοπατιού των δύο κόμβων. Αν στο αποτέλεσμα του XOR το πρώτο 1-bit βρίσκεται στη θέση k αυτό σημαίνει ότι τα μονοπάτια των δύο κόμβων ταυτίζονται για τις πρώτες k-1 ακμές και στην συνέχεια αποκλίνουν. Ο αριθμός μονοπατιού του κόμβου LCA αποτελείται λοιπόν από τα k-1 αριστερά bits του αριθμού μονοπατιού είτε του πρώτου είτε του δεύτερου κόμβου, ακολουθούμενος από 1 και από όσα μηδενικά χρειάζονται για να φθάσει τον αριθμό των d+1 bits. Για την εύρεση του λοιπόν χρειάζεται να πραγματοποιήσουμε την πράξη XOR, να βρούμε την θέση k του αριστερότερου 1-bit, να ολισθήσουμε τον αριθμό μονοπατιού κάποιου από τους δύο κόμβους δεξιά κατά d+1-k θέσεις, να θέσουμε το πιο δεξί bit στην τιμή 1, και να ολισθήσουμε ξανά αριστερά κατά d+1-k θέσεις. Σύμφωνα με την υπόθεση όλες αυτές οι πράξεις σε επίπεδο bit γίνονται σε σταθερό χρόνο, και άρα μετά την αρίθμηση των κόμβων ενός πλήρους δυαδικού δένδρου σύμφωνα με την κωδικοποίηση των μονοπατιών που περιγράφθηκε παραπάνω, ο LCA δύο οποιωνδήποτε κόμβων μπορεί να ανακτηθεί σε O(1).

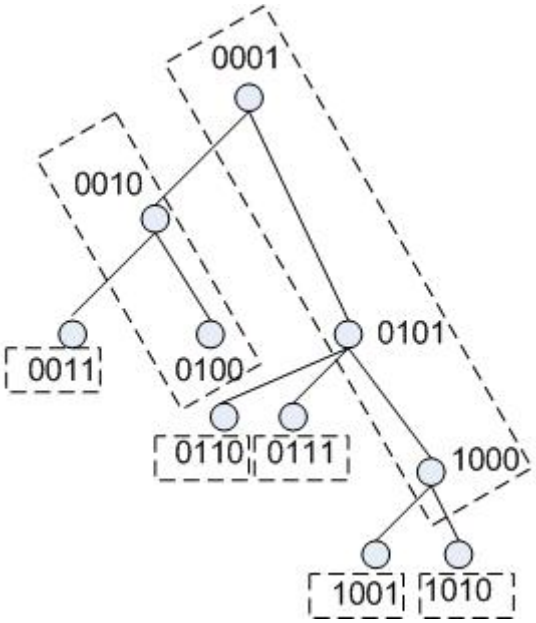
Επομένως, αν μπορούμε να απεικονίσουμε κάποιο επιθυμητό δένδρο T σε ένα δυαδικό δένδρο B, τότε θα μπορούμε να έχουμε σταθερού χρόνου εύρεση LCA για τους κόμβους του T. Το πρώτο βήμα για την απεικόνιση ενός δένδρου T σε κάποιο δυαδικό δένδρο B είναι η αρίθμηση των κόμβων του T κάνοντας διάσχιση προτεραιότητας βάθους (depth-first), η γνωστή από την θεωρία των δομών δεδομένων pre-order αρίθμηση. Με την αρίθμηση αυτή επιτυγχάνεται η εξής ιδιότητα: Οι κόμβοι οποιουδήποτε υποδένδρου με q κόμβους που έχουν ως ρίζα έναν κόμβο με αρίθμηση k, έχουν αρίθμηση από k+1 ως k+q-1, ο πρόγονος δηλαδή κάποιου κόμβου έχει πάντοτε αρίθμηση μικρότερη του απογόνου του.

Ορίζουμε την συνάρτηση h(k), για οποιονδήποτε αριθμό k, ως την θέση του πρώτου από τα δεξιά 1-bit, στην δυαδική αναπαράσταση του k. Ορίζουμε επίσης το ύψος ενός κόμβου ενός δυαδικού δένδρου B, ως τον αριθμό των κόμβων στον μονοπάτι από τον κόμβο μέχρι κάποιο φύλλο. Σύμφωνα με αυτό λοιπόν, το ύψος ενός φύλλου είναι 0 ενώ το ύψος της ρίζας, αν

έχουμε p φύλλα, είναι $\log_2 p$. Γίνεται φανερό ότι το ύψος ενός κόμβου δυαδικού δένδρου που έχει αριθμό μονοπατιού k, είναι h(k).

Για έναν κόμβο υ του Τ, όπου το υ είναι η pre-order αρίθμηση του κόμβου αλλά συμβολίζει επίσης τον ίδιο τον κόμβο αφού η αρίθμηση αυτή είναι μονοσήμαντη, ορίζουμε ως I(υ) τον κόμβο ω στο δένδρο Τ, όπου το ω είναι πάλι η pre-order αρίθμηση του κόμβου, για τον οποίο h(ω) είναι το μέγιστο h( ) όλων των κόμβων στο υποδένδρο με ρίζα το υ, συμπεριλαμβανομένου και του υ. Μπορεί να αποδειχθεί ότι για οποιονδήποτε κόμβο υ του δένδρου Τ, υπάρχει ένας μοναδικός κόμβος ω για τον οποίο h(ω) είναι το μέγιστο όλων των κόμβων του υποδένδρου του υ, δηλαδή η συνάρτηση υ→I(υ) είναι καλά ορισμένη.

Η συνάρτηση υ→I(υ) είναι κεντρική για την απεικόνιση των κόμβων του Τ σε ένα άλλο υποθετικό δυαδικό δένδρο Β. Το στοίχημα είναι να διατηρηθούν κατά την απεικόνιση αρκετές προγονικές σχέσεις που υπάρχουν στο Τ και στο Β, ώστε ώστε οι σχέσεις lca στο Β να χρησιμοποιηθούν για να προσδιοριστούν οι σχέσεις lca στο Τ. Ως πρώτο βήμα για την κατανόηση της απεικόνισης, διαμερίζουμε τους κόμβους του Τ σε σύνολα κόμβων των οποίων η I τιμή είναι η ίδια. Το σύνολο που περιέχει όλους τους κόμβους για τους οποίους η τιμή I είναι ίδια και έχει μια συγκεκριμένη τιμή ονομάζεται **δρόμος**. Δηλαδή, *δύο κόμβοι υ και u ανήκουν στον ίδιο δρόμο, αν και μόνον αν I(u)=I(υ)*. Αλγοριθμικά μπορούμε να ανακτήσουμε το I(υ), για κάθε κόμβο υ, κάνοντας μία από τα κάτω προς τα πάνω διάσχιση του Τ. Για κάθε φύλλο, I(υ)= υ, ενώ για κάθε εσωτερικό κόμβο I(υ)=υ αν h(υ) είναι μεγαλύτερο από το h(I(υ')) για κάθε παιδί υ' του υ, ειδάλλως το I(υ) τίθεται ίσο με το παιδί u' για το οποίο το h(I(υ΄)) είναι το μέγιστο ανάμεσα σε όλα τα παιδιά του υ. Κάθε δρόμος λοιπόν αποτελεί ένα ανοδικό μονοπάτι στο δένδρο Τ. Και αφού οι τιμές h(I( )) ποτέ δεν μειώνονται κατά μήκος κάποιου ανοδικού μονοπατιού έπεται ότι *για οποιονδήποτε κόμβο υ, ο κόμβος I(υ) είναι ο βαθύτερος κόμβος στον δρόμο που περιέχει τον κόμβο υ*. Τα παραπάνω γίνονται περισσότερο κατανοητά με τη βοήθεια του παρακάτω σχήματος, όπου οι κόμβοι ενός δένδρου μετά την pre-order αρίθμηση, διαμερίζονται σε 7 δρόμους. Δρόμοι είναι τα σύνολα των κόμβων που περιέχονται μέσα στα ορθογώνια με τις διακεκομμένες γραμμές.



Ορίζουμε στη συνέχεια την **κεφαλή** κάθε δρόμου ως τον κόμβο του δρόμου που βρίσκεται πιο κοντά στην ρίζα. Για παράδειγμα στον δρόμο με τους κόμβους 0010 (2) και 0100 (4) του σχήματος, κεφαλή είναι ο κόμβος 0010.

Μπορούμε να κάνουμε τώρα την απεικόνιση στο δυαδικό δένδρο B. Κάθε κόμβος υ του δένδρου T, απεικονίζεται στον κόμβο I(υ). Για κάθε κόμβο υ υπάρχει ένα μοναδικό I(υ), ωστόσο είναι δυνατόν διαφορετικοί κόμβοι να έχουν την ίδια I τιμή και να ανήκουν στον ίδιο δρόμο, άρα όλοι οι κόμβοι ενός δρόμου απεικονίζονται στον ίδιο κόμβο του δένδρου B. Η μη αμφιμονοσήμαντη αυτή αντιστοιχία μοιάζει να αντιβαίνει στη διαίσθηση μας και φαίνεται ίσως ακόμα και παράλογη, αφού η απώλεια πληροφορίας δείχνει να μην συντηρεί τις προγονικές σχέσεις. Ωστόσο, όπως θα δούμε αναλυτικά παρακάτω συντηρούμε όσες προγονικές σχέσεις χρειαζόμαστε για να μπορέσουμε στη συνέχεια να ανακτήσουμε τον LCA. Συνοπτικά λοιπόν η γραμμική προεπεξεργασία του δένδρου T συνίσταται στα εξής 4 βήματα:

- Διάσχιση με προτεραιότητα βάθους του δένδρου T και preorder αρίθμηση των κόμβων. Για κάθε κόμβο πρέπει επιπλέον να κρατάμε έναν δείκτη στον γονιό του, μετατρέπουμε δηλαδή τις ακμές σε διπλής κατεύθυνσης.
- Χρήση του bottom-up αλγορίθμου που περιγράφηκε παραπάνω για τον υπολογισμό της τιμής I(υ) για κάθε κόμβο υ. Για κάθε αριθμό k για τον οποίο I(u)=k δημιουργείται ένας δείκτης L(I(u)) που δείχνει στην κεφαλή του δρόμου στον οποίο ανήκει ο κόμβος k.

- Έστω B ένα πλήρες δυαδικό δένδρο με βαθος $d=\lceil \log_2 n \rceil - 1$, κάθε κόμβος του οποίου παριστάνεται με τον αριθμό μονοπατιού από την ρίζα ως τον κόμβο. Απεικόνιση κάθε κόμβου υ του T στον κόμβο I(υ) στο B. Το βήμα αυτό είναι απλά εννοιολογικό, δεν έχει δηλαδή υπολογιστικό κόστος, ωστόσο το υποθετικό δένδρο B μας βοηθά στην κατανόηση του αλγορίθμου.
- Τα παραπάνω 3 βήματα είναι ο πυρήνας της προεπεξεργασίας, ωστόσο είναι απαραίτητο ένα τελευταίο τεχνικό βήμα κατά το οποίο αποθηκεύεται ένας αριθμός για κάθε κόμβο υ που παρέχει κάποια πληροφορία για το που απεικονίζονται στο B οι πρόγονοι του υ στο T. Ο αριθμός αυτός συμβολίζεται με A_υ, και κάθε bit $A_\upsilon(i)$ τίθεται στην τιμή 1 αν και μόνον αν ο κόμβος υ έχει κάποιον πρόγονο στο T που απεικονίζεται σε ύψος i στο B, δηλαδή αν και μόνον αν το υ έχει κάποιον πρόγονο u για τον τέτοιον ώστε h(I(u))=i. Ο υπολογισμός του δυαδικού αριθμού γίνεται με την διάσχιση του δένδρου από πάνω προς τα κάτω. Αν ο κόμβος υ' είναι γονέας του υ τότε ο αριθμός A_υ προκύπτει από την αντιγραφή του αριθμού $A_{\upsilon'}$ και την ανάθεση στο bit $A_\upsilon(i)$ της τιμής 1 αν h(I(υ))=i.

Σχετικά με τις προγονικές σχέσεις αποδεικνύεται ότι *αν ο κόμβος z είναι πρόγονος του x στο δένδρο T, τότε ο I(z) είναι πρόγονος του I(x) στο B*. Πρόγονος εδώ νοείται με την μη γνήσια έννοια, δηλαδή κάθε κόμβος θεωρείται πρόγονος του εαυτού του. Διατυπωμένο διαφορετικά, αν ο z είναι πρόγονος του x στο T, τότε είτε οι x και z βρίσκονται στον ίδιο δρόμο (απεικονίζονται λοιπόν στο I(z)=I(x)) ή ο κόμβος I(z) είναι γνήσιος πρόγονος του κόμβου I(x) στο B.

Έστω ότι ο άγνωστος κόμβος z είναι ο LCA των κόμβων x και y. Έστω επίσης ότι b είναι ο LCA των απεικονίσεων των x και y στο δένδρο B, δηλαδή των κόμβων I(x) και I(y). Όπως περιγράψαμε παραπάνω, για ένα δυαδικό δένδρο με αρίθμηση μονοπατιών,ο LCA μπορεί με χρήση δυαδικών πράξεων να ανακτηθεί σε σταθερό χρόνο. Συνεπώς ο b μπορεί να υπολογιστεί σε σταθερό χρόνο. Μπορεί να αποδειχθεί, ότι αν j είναι η μικρότερη θέση που είναι μεγαλύτερη ή ίση με h(b) για την οποία οι αριθμοί A_x και A_y έχουν αμφότεροι την τιμή 1 στην θέση j, τότε ο κόμβος I(z) βρίσκεται σε ύψος j στο B, δηλαδή h(I(z))=j. Για δύο οποιουσδήποτε κόμβους x,y λοιπόν μπορεί να υπολιγιστεί το h(I(z)), όπου z=lca(x,y), σε σταθερό χρόνο.

Θα δειχθεί τώρα πως με δεδομένα τα x, y και το h(I(z)), δηλαδή το ύψος της απεικόνισης του z στο B, μπορεί να βρεθεί ο ίδιος ο κόμβος z σε σταθερό χρόνο. Ας θεωρηθεί ο δρόμος που περιέχει το z στο T. Το ανοδικό μονοπάτι στο T από το x στο z μπαίνει στον δρόμο του z σε κάποιον κόμβο $\bar{x}$, που είναι πιθανώς ο ίδιος ο z, και συνεχίζει κατά μήκος του μονοπατιού μέχρι να συναντήσει τον ίδιο τον z. Αντίστοιχα το ανοδικό μονοπάτι του T από το y στο z

μπαίνει στον δρόμο του z σε κάποιον κόμβο $\bar{y}$ και συνεχίζει κατά μήκος του μονοπατιού μέχρι να συναντήσει τον ίδιο τον z. Αφού οι δρόμοι είναι πάντοτε ανοδικά μονοπάτια προς τη ρίζα και ο z είναι ο LCA των x και y, έπεται ότι ο z θα είναι είτε ο $\bar{x}$ είτε ο $\bar{y}$. Συγκεκριμένα θα είναι ο πιο ψηλός από τους $\bar{x}$ και $\bar{y}$, και με βάση το σχήμα αρίθμησης, z=$\bar{x}$ αν και μόνον αν $\bar{x}<\bar{y}$.

Από τα παραπάνω συμπεραίνουμε ότι για να βρούμε το z αρκεί να βρούμε τα $\bar{x}$ και $\bar{y}$ αφού το z προκύπτει από αυτά. Θα περιγράψουμε την διαδικασία εύρεσης του $\bar{x}$. Με τον ίδιο ακριβώς τρόπο βρίσκεται και το $\bar{y}$ και στην συνέχεια ανακτάται ο z. Ο κόμβος x απεικονίζεται στο δένδρο B στον κόμβο I(x) που ανήκει στο υποδένδρο με ρίζα το I(z), όπου h(I(z))=j. Αν λοιπόν h(I(x))=j, τότε τα x και z ανήκουν στον ίδιο δρόμο, άρα $\bar{x}$= x και τελειώσαμε. Ας υποτεθεί λοιπόν ότι x≠$\bar{x}$.

Έστω ω (που είναι πιθανώς ο x) ο κόμβος του T στο μονοπάτι από το x στο z που είναι ακριβώς από κάτω από τον δρόμο που περιέχει το z. Αφού x≠$\bar{x}$, το x δεν είναι στον ίδιο δρόμο με τον z, άρα ο κόμβος ω υπάρχει. Από το h(I(z)) που έχει υπολογιστεί σε σταθερό χρόνο, και από το A_x που είναι προϋπολογισμένο θα ανακτηθεί τα h(I(ω)) και στην συνέχεια τα I(ω),ω και $\bar{x}$.

Αφού το ω βρίσκεται στο υποδένδρο με ρίζα z στο T και δεν είναι στον δρόμο του z, το ω απεικονίζεται στο δένδρο B σε κάποιον κόμβο με ύψος αυστηρά μικρότερο από το ύψος του I(z). Συγκεκριμένα, από όλους τους κόμβους που βρίσκονται στο μονοπάτι από το x στο z και δεν ανήκουν στον δρόμο του z, ο ω απεικονίζεται στο B στον κόμβο με το μέγιστο ύψος. Άρα το h(I(ω)) πρέπει να είναι το πιο σημαντικό 1-bit του A_x που βρίσκεται σε θέση μικρότερη του j. Με τις παραδοχές που έχουν γίνει, η θέση του bit αυτού και άρα και το h(I(ω)) μπορεί να υπολογιστεί σε σταθερό χρόνο.

Έστω h(I(ω))=k. Θα βρεθεί τώρα ο I(ω). Ο ω είτε ταυτίζεται με τον x είτε είναι γνήσιος πρόγονος του x στο T, οπότε αφού η προγονική σχέση διατηρείται στο B, είτε I(ω)=I(x) είτε ο κόμβος I(ω) είναι γνήσιος πρόγονος του κόμβου I(x) στο δένδρο B. Επιπλέον εκ του γεγονότος ότι οι αριθμοι I κωδικοποιούν μονοπάτια στο B, έπεται ότι οι αριθμοί I(x) και I(ω) είναι ίδιοι για τα bits που είναι αριστερά του k, και ο I(ω) έχει ένα 1-bit στη θέση k ακολουθούμενη με μηδενικά μέχρι το δεξί άκρο. Άρα το I(ω) ανακτάται από το I(x) και το k με δυαδικές πράξεις. Με δεδομένο το I(ω) μπορούμε να βρούμε το ω, αφού ω=L(I(ω)), είναι δηλαδή εξ'υποθέσεως η κεφαλή του αντίστοιχου δρόμου. Ο γονιός του ω είναι ο κόμβος $\bar{x}$, ο οποίος μπορεί να ανακτηθεί σε σταθερό χρόνο αφού κατά την προεπεξεργασία έχουν

δεικτοδοτηθεί οι γονείς των κόμβων. Αντίστοιχα υπολογίζεται και ο $\bar{y}$, και τελικά ο z=lca(x,y) σε σταθερό χρόνο. Συνοπτικά τα βήματα του αλγορίθμου είναι τα παρακάτω:

1. Ευρεση του κατώτατου κοινού προγόνου b στο δένδρο B των κόμβων I(x) και I(y)
2. Εύρεση της μικρότερης θέσης j που είναι μεγαλύτερη ή ίση με το h(b) τέτοιας ώστε αμφότεροι οι αριθμοί A_x και A_y έχουν 1-bits στην θέση j. Η τιμή j είναι τότε ίση με h(I(z)).
3. Εύρεση του κόμβου $\bar{x}$, του εγγύτερου στο x κόμβου που ανήκει στον δρόμο του z

3α. Ευρεση της θέσης *l* του δεξιότερου 1-bit στο A_x

3β. Αν *l=j* τότε x=$\bar{x}$ και μετάβαση στο βήμα 4

(ειδάλλως όταν *l<j*)

3γ. Εύρεση της θέσης k του αριστερότερου 1-bit στο A_x που βρίσκεται στα δεξιά της θέσης j. Σχηματισμός του αριθμού που αποτελείται από τα bits του I(x) στα αριστερά της θέσης k, ακολουθούμενα από ένα 1-bit στη θέση k, και όλο μηδενικά μετά. Ο αριθμός αυτός είναι ο I(ω) παρότι ακόμα δεν γνωρίζουμε τον ω. Εύρεση του L(I(ω)) που είναι ο κόμβος ω εξ'υποθέσεως. Ο κόμβος $\bar{x}$ είναι ο πατέρας του κόμβου ω στο T.

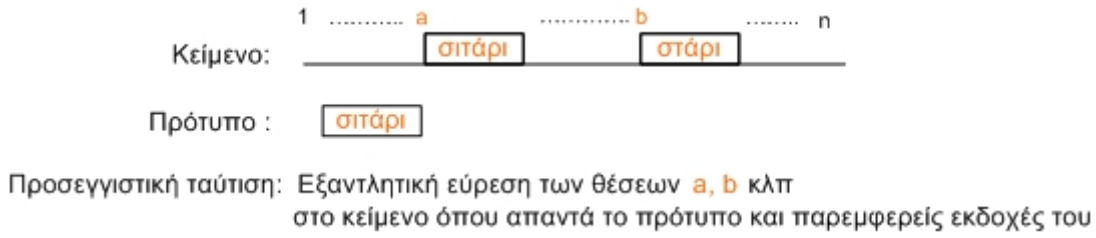
4. Εύρεση του $\bar{y}$, του εγγύτερου κόμβου στο y που ανήκει στον ίδιο δρόμο με το z, με την ίδια προσέγγιση όπως στο βήμα 3
5. Αν $\bar{x} < \bar{y}$ τότε z=$\bar{y}$ αλλιώς z=$\bar{x}$

Είδαμε λοιπόν πως μετά από μια γραμμική προεπεξεργασία ο αλγόριθμος αυτός μπορεί να ανακτά τον κατώτερο κοινό πρόγονο για δύο οποιουσδήποτε κόμβους του δένδρου με πολυπλοκότητα O(1). Όπως αναφέραμε παραπάνω, ακόμα και αν χρησιμοποιούμε το στάνταρ μοντέλο σύμφωνα με το οποίο οι δυαδικές πράξεις δεν ανήκουν σε αυτές που μπορούν να πραγματοποιηθούν σε σταθερό χρόνο, είναι έυκολο να γίνουν κάποιες επιμέρους τροποποιήσεις κατά την προεπεξεργασία, ώστε τελικά ο αλγόριθμος να έχει το ίδιο αποτέλεσμα O(1) και για το στάνταρ μοντέλο μονάδας κόστους RAM.

2.2.3 Προσεγγιστική ταύτιση προτύπου (approximate matching)

2.2.3.1 Το πρόβλημα

Αρκετές φορές είναι επιθυμητό να γίνεται μη ακριβής ταύτιση με κάποιο πρότυπο. Αναζήτουμε δηλαδή μέσα σε μία ακολουθία υποακολουθίες που μοιάζουν αρκετά με το πρότυπο. Αναφέραμε ήδη παραδείγματα προσεγγιστικής ταύτισης, όπως το πρόβλημα της ταύτισης με χαρακτήρα μπαλαντέρ, και το k-mismatch πρόβλημα. Η χρησιμότητα της προσεγγιστικής αναζήτησης είναι σίγουρα μεγαλύτερη από αυτή της ακριβούς αναζήτησης, κάτι που υπονοείται και από το μεγαλύτερο πλήθος εφαρμογών. Χαρακτηριστικό είναι και πάλι το παράδειγμα των βιολογικών δεδομένων. Όπως αναφέραμε η απαραίτητη προϋπόθεση για την εξέλιξη των ειδών, είναι οι αλλαγές που προκαλούνται από τυχαίες μεταλλάξεις του γενετικού τους υλικού, ενώ και άτομα του ιδίου είδους έχουν διαφορές στα χαρακτηριστικά τους που οφείλονται σε διαφορές σε σημεία του γενετικού τους υλικού. Ενώ κάποιες από αυτές τις μεταλλάξεις υπονοούν δραματικές αλλαγές σε δομές και λειτουργίες, ένα μεγάλο μέρος τους μπορεί να μην έχει καμία συνέπεια. Ως αποτέλεσμα μια ακολουθία πριν από κάποια τέτοια μετάλλαξη και η ίδια ακολουθία μετά, παρόλο που θα είναι διαφορετική σε μερικούς από τους χαρακτήρες της, έχει ακριβώς την ίδια λειτουργικότητα και επομένως μπορεί να θεωρηθεί ως πανομοιότυπη. Τέτοιες καταστάσεις είναι ο κανόνας στη βιολογία. Η ουσία λοιπόν εδώ είναι ότι όταν κάποιος αναζητά να βρει μια αλληλουχία, συνήθως ενδιαφέρεται και για παρόμοιες αλληλουχίες, επειδή οι μικροαλλαγές που έχουν δεν επηρεάζουν τις λειτουργίες στις οποίες αυτές εμπλέκονται. Τέτοιες ακολουθίες προσπαθούν να εντοπίσουν οι αλγόριθμοι προσεγγιστικής ταύτισης προτύπου. Η προσεγγιστική ταύτιση κάποιου προτύπου μπορεί να ανακαλύψει ομοιότητες σε μη πανομοιότυπα γονιδιώματα, αλλά και να αποκαλύψει τον βαθμό των διαφορών. Υπάρχουν πλήθος εφαρμογών των αλγορίθμων προσεγγιστικής ταύτισης στην μοριακή βιολογία, θα εστιάσουμε ωστόσο κυρίως στις τυπικές και τεχνικές πτυχές του προβλήματος. Μια πολύ κατατοπιστική ανασκόπηση των αλγορίθμων για προσεγγιστική ταύτιση γίνεται στο [Nav01].



Ένα κρίσιμο ζήτημα που τίθεται για το πρόβλημα της προσεγγιστικής ταύτισης είναι η κατάλληλη τυποποίηση του βαθμού ομοιότητας, ή αντίστροφα, του βαθμού διαφοράς μεταξύ δύο ακολουθιών. Η πιο κοινή και απλή τυποποίηση εστιάζει στον μετασχηματισμό της μίας ακολουθίας σε μία άλλη με μία σειρά επιτρεπόμενων συντακτικών πράξεων. Το ελάχιστο

πλήθος συντακτικών πράξεων που οδηγούν στο μετασχηματισμό μιας ακολουθίας σε μία άλλη ονομάζεται **συντακτική απόσταση** (edit distance) των ακολουθιών. Οι συντακτικές πράξεις που είναι επιτρεπτές για την μετατροπή της μίας ακολουθίας στην άλλη είναι η εισαγωγή ενός χαρακτήρα στην πρώτη ακολουθία (insertion), η διαγραφή ενός χαρακτήρα από την πρώτη ακολουθία (deletion), και η αντικατάσταση ενός χαρακτήρα στην δεύτερη ακολουθία με έναν χαρακτήρα από την πρώτη ακολουθία (replacement). Για παράδειγμα αν συμβολίσουμε με I την πράξη της εισαγωγής, με D την πράξη της διαγραφής, με R την πράξη της αντικατάστασης και με M την μη-πράξη της ταύτισης, τότε η ακολουθία 'μεσιτών' μπορεί να μετατραπεί στην ακολουθία 'αμέσως' (με τη σύμβαση ότι οι χαρακτήρες που τονίζονται είναι ίδιοι με τους αντίστοιχους μη τονισμένους) ως ακολούθως:

Πράξη	I	M	M	M	D	D	M	R
Ακολουθία 1	-	μ	ε	σ	ι	τ	ώ	ν
Ακολουθια 2	α	μ	έ	σ	-	-	ω	ς

Η ακολουθία με αλφάβητο I,D,M,R που περιγράφει τον μετασχηματισμό μίας ακολουθίας σε μία άλλη ονομάζεται **συντακτικό μεταγραφής** (edit transcript) ή εν συντομία, μεταγραφή.

2.2.3.2 Αλγόριθμοι χωρίς προεπεξεργασία (dynamic programming)

Ο υπολογισμός της συντακτικής απόστασης βρίσκεται στον πυρήνα της προσεγγιστικής αναζήτησης ενός προτύπου μέσα σε μία άλλη. Συγκεκριμένα, ο αλγόριθμος για την εύρεση της συντακτικής απόστασης δύο ακολουθιών, μπορεί με μία πολύ μικρή παραλλαγή να χρησιμοποιηθεί ως έχει για την προσεγγιστική αναζήτηση. Θα εξετάσουμε πως μπορούμε να υπολογίσουμε με χρήση δυναμικού προγραμματισμού την συντακτική απόσταση δύο ακολουθιών και στην συνέχεια θα δούμε πως αυτή η μέθοδος μπορεί να παραλλαχθεί για να μετατραπεί σε έναν αλγόριθμο προσεγγιστικής αναζήτησης. Η μέθοδος του δυναμικού προγραμματισμού είναι κεντρική για πλήθος αλγορίθμων πάνω σε ακολουθίες. Στην πιο κοινή της εκδοχή υποθέτουμε ότι δεν διαθέτουμε κανένα είδος ευρετηρίου για τις ακολουθίες.

2.2.3.2.1 Υπολογισμός της συντακτικής απόστασης

Έστω δύο ακολουθίες S_1 και S_2, με $|S_1|=m$ και S_2 $|S_2|=n$. Συμβολίζεται με D(i,j) η συντακτική απόσταση των προθεμάτων $S_1[1..i]$ και $S_2[1..j]$. Όπως θα ξεκαθαριστεί στη συνέχεια, η βάση των αλγορίθμων του δυναμικού προγραμματισμού είναι η παρατήρηση ότι αυτή η συντακτική απόσταση μπορεί να υπολογιστεί μέσω κάποιας αναδρομικής σχέσης από τις

συντακτικές αποστάσεις κάποιων ακόμα μικρότερων προθεμάτων. Ακολουθώντας τον παραπάνω συμβολισμό, αν η ακολουθία S_1 έχει m γράμματα και η ακολουθία S_2 έχει n γράμματα, η συντακτική απόσταση των S_1 και S_2 θα συμβολίζεται ως D(m,n) και μπορεί να υπολογιστεί και αυτή μέσω αναδρομής. Για να υπολογιστεί το D(m,n) μέσω δυναμικού προγραμματισμού θα χρησιμοποιηθούν όλα τα D(i,j) για όλους τους συνδυασμούς i και j, όπου το $0 \leq i \leq m$ και $0 \leq j \leq n$. Αυτή είναι η στάνταρ προσέγγιση δυναμικού προγραμματισμού που χρησιμοποιείται σε πλήθος υπολογιστικών προβλημάτων και αποτελείται από τρία βασικά συστατικά:

- την αναδρομική σχέση (recursive relation),
- τον υπολογισμό με πίνακα (tabular computation) και
- την προς τα πίσω αναζήτηση (traceback).

Η αναδρομική σχέση αναδεικνύει μία σχέση μεταξύ της τιμής του D(i,j), για i και j θετικά, με τιμές του D για ζεύγη δεικτών μικρότερα από i και j. Όταν δεν υπάρχουν μικρότεροι δείκτες πρέπει οι τιμές του D να έχουν οριστεί αυθαίρετα ως αρχικές συνθήκες για το D(i,j). Για το πρόβλημα της συντακτικής απόστασης οι αρχικές συνθήκες που χρησιμοποιούνται είναι:

D(i,0)=i και D(0,j)=j.

Η αρχική συνθήκη D(i,0)=i είναι προφανώς σωστή, δίνει δηλαδή τον σωστό αριθμό που απαιτείται από τον ορισμό, αφού ο μόνος τρόπος να μετασχηματιστούν οι πρώτοι i χαρακτήρες της S_1 σε μηδέν χαρακτήρες της S_2 είναι με την διαγραφή όλων των i χαρακτήρων του S_1. Αντίστοιχα η συνθήκη D(0,j)=j είναι σωστή αφού πρέπει να εισαχθούν j χαρακτήρες για να μετατρέψουν μηδέν χαρακτήρες της ακολουθίας S_1 σε j χαρακτήρες της ακολουθίας S_2.

Η αναδρομική σχέση για το D(i,j) όταν $i,j>0$ είναι:

$D(i,j)= \min[D(i-1,j)+1, D(i,j-1)+1, D(i-1,j-1)+t(i,j)]$,

όπου t(i,j) ορίζεται να έχει τιμή 1 αν $S_1(i) \neq S_2(j)$ και τιμή 0 όταν $S_1(i)=S_2(j)$

Μετά την κατάρτιση της αναδρομικής σχέσης το δεύτερο βήμα που πρέπει να γίνει είναι η χρήση της για τον αποδοτικό υπολογισμό του D(m,n). Η πιο απλοϊκή μέθοδος για να γίνει αυτό θα ήταν η ευθεία χρήση του αναδρομικού ορισμού για την σύνταξη ενός κώδικα για την συνάρτηση D(i,j) με οποιαδήποτε γλώσσα προγραμματισμού που επιτρέπει την αναδρομή. Η μέθοδος αυτή όμως δεν είναι καθόλου αποδοτική καθότι ο αριθμός των αναδρομικών κλήσεων αυξάνει εκθετικά με την αύξηση των m και n. Ωστόσο, υπάρχουν μόνο $(m+1)\cdot(n+1)$ συνδυασμοί των i και j άρα υπάρχουν μόνο $(m+1)\cdot(n+1)$ διακριτές αναδρομικές κλήσεις που είναι δυνατόν να γίνουν. Η μη αποδοτικότητα λοιπόν της προσέγγισης από τα πάνω προς τα κάτω έγκειται στο γεγονός ότι γίνεται ένα τεράστιο πλήθος αναδρομικών κλήσεων για συγκεκριμένα ζεύγη i και j. Το κλειδί για την επίτευξη ενός κατά πολλές τάξεις

αποδοτικότερου υπολογισμού του D(n,m) είναι να εγκαταληφθεί η απλότητα του υπολογισμού από τα πάνω προς τα κάτω και να γίνει ο υπολογισμός από κάτω προς τα πάνω.

Στην από-κάτω-προς-πάνω προσέγγιση υπολογίζεται πρώτα την τιμή για το D(i,j) για τις μικρότερες δυνατές τιμές των i και j, και στην συνέχεια υπολογίζονται τιμές του D(i,j) για αυξανόμενες τιμές των i και j. Συνήθως, αυτός ο υπολογισμός οργανώνεται με τη βοήθεια ενός **πίνακα δυναμικού προγραμματισμού** μεγέθους (m+1)·(n+1). Για δύο ενδεικτικές ακολουθίες, 'writer' και 'vintner' όπως φαίνεται και στο σχήμα ο πίνακας δυναμικού προγραμματισμού κρατάει όλες τις πιθανές τιμές για τα i και j.

D(i,j)			w	r	i	t	e	r	s
		0	1	2	3	4	5	6	7
	0	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
v	1	**1**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
i	2	**2**	**2**	**2**	**2**	**3**	**4**	**5**	**6**
n	3	**3**	**3**	**3**	**3**	**3**	**4**	**5**	**6**
t	4	**4**	**4**	**4**	**4**	*			
n	5	**5**							
e	6	**6**							
r	7	**7**							

Η πρώτη γραμμή του πίνακα αντιστοιχεί στο i=0 και λέγεται γραμμή 0. Αντίστοιχα, η πρώτη στήλη του πίνακα αντιστοιχεί στο j=0 και λέγεται στήλη 0. Τα κελιά που ανήκουν στη γραμμή 0 και τη στήλη 0 αντιστοιχούν στις αρχικές συνθήκες της αναδρομικής σχέσης, επομένως οι τιμές τους καθορίζονται από αυτές. Στη συνέχεια τα εναπομείναντα m·n στοιχεία του πίνακα μπορούν να συμπληρωθούν μέσω του υπολογισμού μιας γραμμής τη φορά, με σειρά αυξανόμενου i. Μέσα σε κάθε γραμμή, τα στοιχεία θα συμπληρώνονται με σειρά αυξανόμενου j. Εκτός αυτού όμως είναι εύκολο να αποδειχθεί ότι ο πίνακας μπορεί να υπολογιστεί και στήλη προς στήλη. Τέλος, είναι εφικτό να συμπληρωθεί ο πίνακας συμπληρώνοντας διαδοχικές αντιδιαγώνιες.

Αφού κάθε υπολογισμός του στοιχείου του πίνακα μπορεί να γίνει σε σταθερό χρόνο, και αφού η πλήρης συμπλήρωση του πίνακα απαιτεί την συμπλήρωση (m+1)·(n+1) στοιχείων, η χρήση του δυναμικού προγραμματισμού υπολογίζει την συντακτική απόσταση δύο ακολουθιών μήκους m και n σε χρόνο O(mn)

Αφού συμπληρωθεί ο πίνακας δυναμικού προγραμματισμού τίθεται το ερώτημα πως μπορούμε να εξαγάγουμε το σχετικό βέλτιστο συντακτικό μεταγραφής. Ο ευκολότερος τρόπος είναι να δημιουργούμε κατάλληλους δείκτες κατά την συμπλήρωση του πίνακα.

Συγκεκριμένα, όταν υπολογίζεται η τιμή του στοιχείου (i,j) θέτουμε έναν δείκτη από το στοιχείο (i,j)

- προς το στοιχείο (i,j-1) αν D(i,j)=D(i,j-1)+1,
- προς το στοιχείο (i-1,j) όταν D(i,j)=D(i-1,j)+1 και
- προς το στοιχείο (i-1,j-1) όταν D(i,j)=D(i-1,j-1) + t(i,j).

Ο κανόνας αυτός ισχύει και για τα στοιχεία της γραμμής 0 και της στήλης 0. Έτσι με εξαίρεση το στοιχείο (0,0) κάθε στοιχείο στην γραμμή 0 δείχνει στο στοιχείο αριστερά του και κάθε στοιχείο της στήλης 0 δείχνει στο στοιχείο από πάνω του. Για τα άλλα στοιχεία είναι πιθανό (και σύνηθες) να έχουν παραπάνω από ένα δείκτη προς άλλα στοιχεία. Οι δείκτες αυτοί επιτρέπουν με εύκολο τρόπο να ανακτήσουμε ένα βέλτιστο συντακτικό μεταγραφής. Πρέπει απλά να ακολουθήσουμε ένα (από τα πιθανώς πολλά) μονοπάτι από το στοιχείο (m,n) προς το στοιχείο (0,0). Το συντακτικό μεταγραφής προκύπτει από το μονοπάτι αυτό όπου κάθε φορά που έχουμε οριζόντια ακμή από το στοιχείο (i,j) στο στοιχείο (i,j-1) το ερμηνεύουμε ως εισαγωγή (I) του χαρακτήρα $S_2(j)$ στην S_1 , κάθε φορά που έχουμε μια κάθετη ακμή από το στοιχείο (i,j) στο στοιχείο (i-1,j) το ερμηνεύουμε ως διαγραφή (D) του χαρακτήρα $S_1(i)$ από το S_1, και κάθε φορά που συναντάμε διαγώνιο ακμή από το (i,j) προς το στοιχείο (i-1,j-1) το ερμηνεύουμε ως αντικατάσταση του $S_1(i)$ από το $S_2(j)$ όταν $S_1(i)\neq S_2(j)$ και ως ταύτιση όταν $S_1(i)=S_2(j)$. Αν υπάρχει παραπάνω από ένας δείκτης από το στοιχείο (m,n) τότε το μονοπάτι μπορεί να ξεκινήσει από οποιονδήποτε από αυτούς τους δείκτες ανακτώντας και το αντίστοιχο σύμβολο για το συντακτικό μεταγραφής. Αντίστοιχα και για κάθε εσωτερικό στοιχείο του μονοπατιού που διαθέτει πάνω από έναν δείκτη το μονοπάτι μπορεί να συνεχίσει από οποιονδήποτε από αυτούς τους δείκτες. Επιπλέον αφού όλα τα στοιχεία εκτός από το (0,0) έχουν κάποιον δείκτη που ξεκινάει από αυτά, το μονοπάτι αυτό δεν μπορεί ποτέ να σταματάει στα μέσα της διαδρομής και άρα θα φθάνει στο (0,0).

Σε κάθε βήμα το μονοπάτι πάει είτε μια γραμμή πιο ψηλά, είτε μια στήλη πιο αριστερά, είτε και τα 2 ταυτόχρονα. Αφού υπάρχουν m γραμμές και n στήλες, το μονοπάτι θα έχει το πολύ n+m βήματα (όταν δεν υπάρχουν καθόλου διαγώνιες) συνεπώς η ανάκτηση του βέλτιστου συντακτικού μεταγραφής μπορεί να γίνει με χρήση του πίνακα δυναμικού προγραμματισμού σε χρόνο O(n+m). Ο πλήρης πίνακας του δυναμικού προγραμματισμού με όλους τους δείκτες για τις ακολουθίες ‘writers’ και ‘vintner’ φαίνεται παρακάτω:

D(i,j)			w	r	i	t	e	r	s
		0	1	2	3	4	5	6	7
	0	**0**	**← 1**	**← 2**	**← 3**	**← 4**	**← 5**	**← 6**	**← 7**
v	1	**↑1**	**↖1**	**↖← 2**	**↖← 3**	**↖← 4**	**↖← 5**	**↖← 6**	**↖← 7**
i	2	**↑2**	**↖↑2**	**↖2**	**↖2**	**← 3**	**← 4**	**← 5**	**← 6**
n	3	**↑3**	**↖↑3**	**↖↑3**	**↖↑3**	**↖3**	**↖↑4**	**↖↑5**	**↖↑6**
t	4	**↑4**	**↖↑4**	**↖↑4**	**↖↑4**	**↖3**	**↖↑4**	**↖↑5**	**↖↑6**
n	5	**↑5**	**↖↑5**	**↖↑5**	**↖↑5**	**↑4**	**↖4**	**↖↑5**	**↖↑6**
e	6	**↑6**	**↖↑6**	**↖↑6**	**↖↑6**	**↑5**	**↖4**	**↖↑5**	**↖↑6**
r	7	**↑7**	**↖↑7**	**↖6**	**↖↑7**	**↑6**	**↑5**	**↖4**	**←5**

Η μέθοδος εύρεσης του βέλτιστου συντακτικού μεταγραφής με τη χρήση των δεικτών δεν ανακτά μόνο μερικές από τις βέλτιστες μεταγραφές, αλλά *όλες* τις δυνατές βέλτιστες μεταγραφές, δηλαδή το σύνολο των διαφορετικών μονοπατιών πάνω στον πίνακα του δυναμικού προγραμματισμού αναπαριστούν όλες τις διαφορετικές βέλτιστες μεταγραφές που μπορούν να υπάρξουν για τις δύο ακολουθίες.

2.2.3.2.2 Προσεγγιστική αναζήτηση

Είδαμε λοιπόν πως μπορούμε να χρησιμοποιήσουμε τον πίνακα του δυναμικού προγραμματισμού για την εύρεση της συντακτικής απόστασης δύο ακολουθιών και όλων των βέλτιστων συντακτικών μεταγραφών. Θα δούμε τώρα ότι με μία μικρή παραλλαγή ο αλγόριθμος αυτός μπορεί να χρησιμοποιηθεί για προσεγγιστική αναζήτηση προτύπων. Έστω ότι η ακολουθία S_1=P[1..m] είναι το πρότυπο που αναζητούμε να βρούμε μέσα στην ακολουθία S_2=T[1..n], η οποία είναι το κείμενο. Μπορούμε να βρούμε όλες τις προσεγγιστικές ταυτίσεις του P μέσα στο T χρησιμοποιώντας τον δυναμικό προγραμματισμό κάνοντας την εξής αλλαγή: θέτουμε όλα τα στοιχεία της πρώτης γραμμής ίσα με το μηδέν. Κατά τα άλλα ο αλγόριθμος ακολουθεί τις ίδιες αρχικοποιήσεις και τα ίδια βήματα με την αλγόριθμο για τον υπολογισμό της βέλτιστης συντακτικής απόστασης. Με την συμπλήρωση με μηδενικά της πρώτης γραμμής αναπαριστούμε το γεγονός ότι από κάθε σημείο του κειμένου T μπορεί να ξεκινάει μια προσεγγιστική ταύτιση. Μετά την συμπλήρωση του πίνακα δυναμικού προγραμματισμού μπορούμε να εξάγουμε τα συμπεράσματα για την προσεγγιστική ταύτιση από την τελευταία γραμμή του πίνακα του δυναμικού

προγραμματισμού. Ο αριθμός της στήλης του στοιχείου της τελευταίας γραμμής είναι η θέση μέσα στο κείμενο T όπου τελειώνει μία εμφάνιση του προτύπου P με **πλήθος διαφορών** ίσο με την τιμή του περιεχομένου του στοιχείου. Τα περιεχόμενα των στοιχειών του πίνακα δυναμικού προγραμματισμού λοιπόν δεν αναπαριστούν πλέον συντακτική απόσταση, και αναφέρονται σε μία παρεμφερή έννοια, το πλήθος των διαφορών, όπου διαφορές μπορεί να είναι είτε **μη-ταυτίσεις** (mismatches) είτε **κενά** στο πρότυπο ή στο κείμενο. Οι μη-ταυτίσεις συνιστούν το ισοδύναμο της πράξης της αντικατάστασης, ενώ ένα κενό στο πρότυπο ή στο κείμενο στην γλώσσα της συντακτικής απόστασης ισοδυναμεί με εισαγωγές ή διαγραφές από το πρότυπο αντίστοιχα.

Στις συνήθεις εφαρμογές βέβαια, οι ζητούμενες διαφορές είναι μικρές, δηλαδή μας ενδιαφέρουν οι εμφανίσεις του προτύπου μέσα στο κείμενο με διαφορές μικρότερες από κάποιο φράγμα. Άλλωστε όταν οι διαφορές είναι της τάξεως του μεγέθους του προτύπου m, η ταύτιση δεν έχει νόημα καθώς με διαδοχικές διαγραφές και εισαγωγές των κατάλληλων χαρακτήρων, το πρότυπο μπορεί να μετασχηματίζεται σε μια εντελώς διαφορετική ακολουθία και άρα να απαντάται κυριολεκτικά σε όλες τις θέσεις του κειμένου. Η προσεγγιστική ταύτιση λοιπόν έχει νόημα για φραγμένο πλήθος διαφορών, και μάλιστα αρκετά μικρότερο από το μέγεθος του προτύπου m. Στο σχήμα μπορούμε να δούμε τον πίνακα του δυναμικού προγραμματισμού για την αναζήτηση του προτύπου ‘survey’ μέσα στο κείμενο ‘surgery’, όπου μας ενδιαφέρουν οι ταυτίσεις όπου το πλήθος διαφορών δεν ξεπερνάει το 2, και που είναι τονισμένες με κόκκινο χρώμα στην τελευταία γραμμή του πίνακα. Παρατηρούμε την διαφορά στην πρώτη γραμμή, που αρχικοποιείται με μηδενικά:

			s	u	r	g	e	r	y
		0	1	2	3	4	5	6	7
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
s	1	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
u	2	**2**	**1**	**0**	**1**	**2**	**2**	**2**	**2**
r	3	**3**	**2**	**1**	**0**	**1**	**2**	**2**	**3**
v	4	**4**	**3**	**2**	**1**	**1**	**2**	**3**	**3**
e	5	**5**	**4**	**3**	**2**	**2**	**1**	**2**	**3**
y	6	**6**	**5**	**4**	**3**	**3**	**2**	**2**	**2**

Ο αλγόριθμος βέβαια επιστρέφει την τελική θέση όπου απαντάται το πρότυπο μέσα στο κείμενο. Το ποια είναι η αρχική θέση είναι ένα ερώτημα που δεν απαντάται. Για την εύρεση της αρχικής θέσης μπορούν να χρησιμοποιηθούν οι αντίστοιχοι δείκτες που

χρησιμοποιήθηκαν και προηγουμένως για την εύρεση του βέλτιστου συντακτικού μεταγραφής, και να ακολουθηθεί το σχετικό μονοπάτι μέχρι να την γραμμή 1. Η μία (από τις πιθανώς πολλές) θέσεις της στήλης όπου θα καταλήξει αυτό το μονοπάτι είναι η αρχική θέση όπου απαντάται το πρότυπο με τον επιθυμητό αριθμό των διαφορών. Το μονοπάτι των δεικτών μπορεί να μας οδηγήσει στην πλήρη στοίχηση του προτύπου με το αντίστοιχο σημείο του κειμένου όπου γίνεται η ταύτιση, δηλαδή μια αντιστοίχηση των δύο ακολουθιών με παρεμβολή κενών στις κατάλληλες θέσεις. Μια άλλη τεχνική για να βρίσκουμε τις αρχικές θέσεις, είναι να κάνουμε την ίδια διαδικασία για τις αντίστοιχες ανεστραμμένες ακολουθίες χωρίς την χρήση δεικτών. Στο παράδειγμα μας λοιπόν του προτύπου ‘survey’ και του κειμένου ‘surgery’ θα είχαμε το πρότυπο ‘yevrus’ και το κείμενο ‘yregrus’. Τα αντίστοιχα αποτελέσματα στην τελευταία γραμμή θα αντιπροσώπευαν τις τελικές θέσεις για τις αντίστροφες ακολουθίες, και τις αρχικές θέσεις για τις κανονικές ακολουθίες.

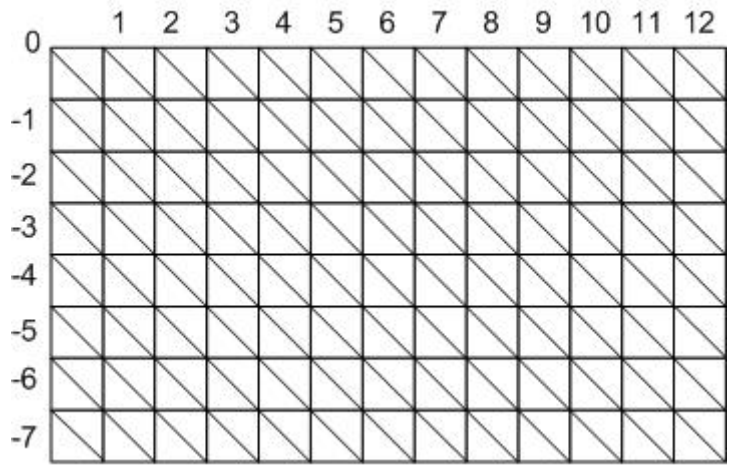
Η ανάλυση της χρονικής πολυπλοκότητας είναι αντίστοιχη με πριν, ο αλγόριθμος δηλαδή πραγματοποιείται σε χρόνο O(mn). Όσον αφορά την χωρική πολυπλοκότητα των αλγορίθμων, αν δεν μας ενδιαφέρει να κρατάμε τους δείκτες για να κάνουμε προς τα πίσω αναζήτηση και μας αρκούν οι τελικές θέσεις, το πρόβλημα χρειάζεται χώρο O(m), αφού σε κάθε φάση της εκτέλεσης του αλγορίθμου η τιμή κάποιου στοιχείου του πίνακα μπορεί να προκύψει από το πάνω στοιχείο της τρέχουσας στήλης και από δύο στοιχεία της προηγούμενης στήλης, το πρώτο στην ίδια γραμμή και το δεύτερο στην αμέσως παραπάνω, άρα πρέπει να έχουμε αποθηκευμένα τα στοιχεία μόνο μιας στήλης.

2.2.3.3 Αλγόριθμοι με προεπεξεργασία (hybrid dynamic programming)

Όταν κάνουμε προσεγγιστική αναζήτηση με φραγμένο πλήθος διαφορών, αν έχουμε την δυνατότητα να ευρετηριοποιήσουμε τις ακολουθίες υπό μορφή δένδρων επιθεμάτων, τότε υπάρχει αλγόριθμος που μπορεί μετά από μία γραμμική προεπεξεργασία των ευρετηρίων να απαντά στο πρόβλημα k-διαφορών σε χρόνο O(kn). Η μέθοδος αυτή ονομάζεται υβριδικός δυναμικός προγραμματισμός, επειδή χρησιμοποιεί τον αλγόριθμο σταθερού χρόνου για την απάντηση ερωτημάτων μέγιστης κοινής επέκτασης (LCE) σε δένδρα επιθεμάτων για να επιλύει υποπροβλήματα μέσα στο πλαίσιο μιας προσέγγισης δυναμικού προγραμματισμού. Το αποτέλεσμα της πολυπλοκότητας O(kn) βρέθηκε πρώτη φορά από τους Landau και Vishkin και από τον Myers και συνεχίζει να επεκτείνεται.

Για την περιγραφή του αλγορίθμου είναι χρήσιμοι κάποιοι ορισμοί για την εισαγωγή χρήσιμης ορολογίας. Καταρχάς, ορίζεται η **κύρια διαγώνιος** του πίνακα δυναμικού προγραμματισμού ως η διαγώνιος που αποτελείται από τα στοιχεία (i,i) όπου $0 \leq i \leq m \leq n$. Οι διαγώνιες που βρίσκονται πάνω από την κύρια διαγώνιο αριθμούνται από το 1 ως το n, η διαγώνιος που ξεκινά στο στοιχείο (0,i) είναι η διαγώνιος i. Οι διαγώνιες που βρίσκονται

κάτω από την κύρια διαγώνιο αριθμούνται με αρνητικούς ακέραιους από το -1 εως το –m, επομένως η διαγώνιος που ξεκινάει με το στοιχείο (i,0) είναι η διαγώνιος –i. Ο πίνακας δυναμικού προγραμματισμού με όλες τις αριθμημένες διαγωνίους φαίνεται στο σχήμα που ακολουθεί.



Ως **d-μονοπάτι** ορίζεται ένα μονοπάτι στον πίνακα του δυναμικού προγραμματισμού που ξεκινάει από τη γραμμή 0 και προσδιορίζει ακριβώς d μη-ταυτίσεις και κενά, δηλαδή ακριβώς d διαφορές. Ως **απώτατο d-μονοπάτι** της διαγωνίου i ορίζεται το d-μονοπάτι που τελειώνει σε στοιχείο της διαγωνίου i και η στήλη c (κατά μήκος της διαγωνίου i) όπου τελειώνει είναι μεγαλύτερη ή ίση από οποιαδήποτε άλλη στήλη όπου τελειώνει κάποιο άλλο d-μονοπάτι της διαγωνίου i. Δηλαδή, το απώτατο d-μονοπάτι της διαγωνίου i είναι το d-μονοπάτι που φθάνει μακρύτερα από όλα τα άλλα μονοπάτια που τελειώνουν κατά μήκος της i.

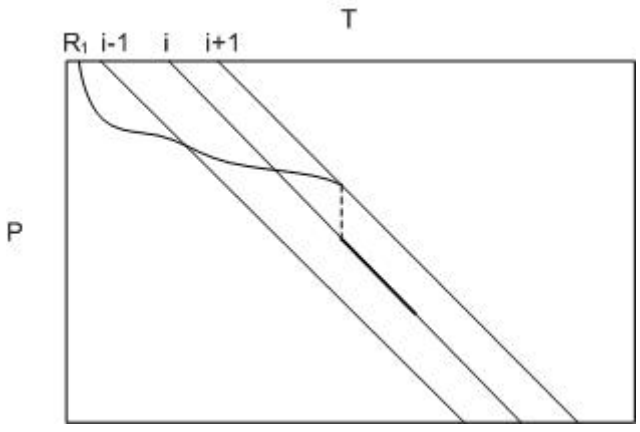
Αν k είναι το φράγμα των διαφορών, ο αλγόριθμος εκτελεί k επαναλήψεις όπου η κάθε μία εκτελείται σε χρόνο O(n). Σε κάθε επανάληψη $d \leq k$, ο αλγόριθμος βρίσκει το τέλος του απώτατου d-μονοπατιού για κάθε διαγώνιο i, όπου $-m \leq i \leq n$. Το απώτατο μονοπάτι για την διαγώνιο i βρίσκεται από τα απώτατα (d-1)-μονοπάτια των διαγωνίων i-1,i και i+1. Η κάθε επανάληψη γίνεται σε χρόνο O(m+n) και αφού m<n, ο χρόνος είναι O(n) για κάθε επανάληψη. Συνεπώς, για τις k-επαναλήψεις εμφανίζεται το επιθυμητό O(kn) όριο. Ο χώρος που χρειάζεται ο αλγόριθμος είναι αντίστοιχα φραγμένος.

Για την πρώτη επανάληψη, όταν d=0, το απώτατο d-μονοπάτι που τελειώνει στην διαγώνιο i αντιστοιχεί στην μέγιστη κοινή επέκταση των T[i..n] και P[1..m]. Αυτό σημβαίνει αφού το 0-μονοπάτι δεν επιτρέπει καμία διαφορά μεταξύ των ακολουθιών, είναι δηλαδή στην ουσία ακριβής ταύτιση προτύπου. Όπως παρουσιάστηκε αναλυτικά στο 2.2.2 η μέγιστη κοινή επέκταση μπορεί να υπολογιστεί σε σταθερό χρόνο. Για d>0 το απώτατο d-μονοπάτι της i-

διαγωνίου μπορεί να υπολογιστεί από ένα από τα 3 παρακάτω μονοπάτια που τελειώνουν στην διαγώνιο i:

- Το μονοπάτι R_1 είναι το απώτατο (d-1)-μονοπάτι της διαγωνίου i+1, συνεχιζόμενο με μία κάθετη ακμή προς την διαγώνιο i, που αντιπροσωπεύει ένα κενό στο κείμενο T, και ακολουθούμενο από την μέγιστη επέκταση κατά μήκος της διαγωνίου i για την οποία το πρότυπο P και το κείμενο T ταυτίζονται στις αντίστοιχες θέσεις. Αφού το R_1 ξεκινάει με ένα (d-1)-μονοπάτι και προσθέτει ένα ακόμα κενό με την κάθετη ακμή, δηλαδή μία ακόμα διαφορά, το R_1 είναι ένα d-μονοπάτι
- Το μονοπάτι R_2 είναι το απώτατο (d-1)-μονοπάτι της διαγωνίου i-1, ακολουθούμενο από μια οριζόντια ακμή στην διαγώνιο i, που αντιπροσωπεύει ένα κενό στο πρότυπο P, ακολουθούμενο από την μέγιστη επέκταση κατά μήκος της διαγωνίου που αντιστοιχεί σε ταύτιση προτύπου και κειμένου. Αντίστοιχα με πριν, το R_2 είναι ένα d-μονοπάτι
- Το μονοπάτι R_3 είναι το απώτατο (d-1)-μονοπάτι της διαγωνίου i, ακολουθούμενο από μια διαγώνιο ακμή κατά μήκος της διαγωνίου μήκους 1 χαρακτήρα, που αντιστοιχεί σε μία μη-ταύτιση μεταξύ του προτύπου και του κειμένου (αν υπήρχε ταύτιση τότε το (d-1)-μονοπάτι δεν θα ήταν το απώτατο), και ακολουθούμενο από την μέγιστη επέκταση κατά μήκος της διαγωνίου που αντιστοιχεί σε ταύτιση προτύπου και κειμένου. Αντίστοιχα με πριν, το μονοπάτι R_3 είναι ένα d-μονοπάτι.

Στο σχήμα φαίνεται το R_1 μονοπάτι κατά τα τρία στάδια της διαδρομής του, από το τέλος ενός (d-1)-μονοπατιού της i+1 διαγωνίου, συνεχίζοντας με μία κάθετη ακμή προς την διαγώνιο i, και στη συνέχεια κατά μήκος της διαγωνίου, ορίζοντας την μέγιστη κοινή επέκταση στις αντίστοιχες θέσεις κειμένου και προτύπου:



Καθένα από τα μονοπάτια R_1, R_2 και R_3 τελειώνει με μια μέγιστη επέκταση που αντιστοιχεί σε ταύτιση μεταξύ του προτύπου P και του κειμένου T. Στην περίπτωση του R_1 (ή του R_2) οι αρχικές θέσεις των δύο υποακολουθιών δίνονται από το τελευταίο σημείο εισόδου του R_1(ή του R_2) στην διαγώνιο i, ενώ στην περίπτωση του R_1 είναι απλά η θέση μετά την τελευταία

μη-ταύτιση του R_3. Οι μέγιστες αυτές επεκτάσεις μπορούν βεβαίως να βρεθούν σε σταθερό χρόνο, όπως έχουμε δει και προηγουμένως.

Μπορεί να αποδειχθεί ότι *καθένα από τα 3 μονοπάτια R_1, R_2 και R_3, είναι d-μονοπάτια της διαγωνίου i* και ότι *το απώτατο d-μονοπάτι της διαγωνίου i είναι από αυτά το μονοπάτι που φθάνει πιο μακριά κατά μήκος της διαγωνίου i.*

Επομένως ο αλγόριθμος στο πρώτο βήμα βρίσκει όλες τις μέγιστες κοινές επεκτάσεις μεταξύ του P[1..m] και του T[i..n] όπου το i τρέχει για 1 ως n, υπολογίζοντας έτσι τα 0-μονοπάτια. Στην συνέχεια κάνει k επαναλήψεις για d από 1 έως k, και σε κάθε επανάληψη υπολογίζεται για κάθε διαγώνιο i, για i από –m έως n, οι θέσεις όπου τελειώνουν τα μονοπάτια R_1,R_2 και R_3 με χρήση των (d-1)-μονοπατιών των διαγωνίων i+1,i-1 και i, και το μονοπάτι από τα 3 που φθάνει πιο μακριά είναι το νέο απώτατο d-μονοπάτι για την διαγώνιο i. Στο τέλος κάθε επανάληψης για το d, όλα τα απώτατα d-μονοπάτια που φθάνουν στην γραμμή n, αντιπροσωπεύουν τις τελικές θέσεις όπου εμφανίζεται το πρότυπο στο κείμενο με ακριβώς d διαφορές.

Όλες οι τιμές για τα d-μονοπάτια, για κάθε d, μπορούν να αποθηκευτούν σε συνολικό χώρο O[k(m+n)] και αφού m≤n, ο χώρος που χρειάζεται ο αλγόριθμος είναι O(kn). Αν δεν μας ενδιαφέρει να κάνουμε την πλήρη στοίχηση προτύπου και κειμένου και να βρούμε τις αρχικές θέσεις που γίνονται οι ταυτίσεις, τότε μετά τον υπολογισμό των d-μονοπατιών, τα προηγούμενα (d-1)-μονοπάτια είναι περιττά για τον υπολογισμό των (d+1)-μονοπατιών και δεν είναι απαραίτητο να κρατούνται αποθηκευμένα. Επομένως αν μας ενδιαφέρουν μόνο οι τελικές θέσεις όπου γίνονται οι ταυτίσεις ο αλγόριθμος χρησιμοποιεί χώρο O(n+m), δηλαδή O(n).

Οι ανακτήσεις των (d-1)-μονοπατιών σε κάθε επανάληψη d γίνονται σε σταθερό χρόνο αφού είναι γνωστά, και στην συνέχεια για κάθε ένα από τα R_1,R_2 και R_3 γίνονται τρεις υπολογισμοί LCE, που λόγω του αλγορίθμου με τη χρήση των ευρετηρίων γίνεται σε σταθερό χρόνο. Επομένως για κάθε d ο υπολογισμός των μονοπατιών απαιτεί χρόνο ανάλογο του πλήθος των διαγωνίων, άρα O(m+n) δηλαδή O(n) αφού m≤n. Για k επαναλήψεις λοιπόν ο αλγόριθμος χρειάζεται χρόνο O(km).

2.3 Αποδοτική κατασκευή δένδρου επιθεμάτων στη μνήμη

2.3.1 Απλοϊκή προσέγγιση.

Στα προηγούμενα κεφάλαια έγινε αντιληπτό πως με χρήση των δένδρων επιθεμάτων μπορεί με αποδοτικό τρόπο να γίνει ταύτιση προτύπων. Τίθεται όμως το ερώτημα, δεδομένης μίας ακολουθίας S μεγέθους n, ποια είναι η πολυπλοκότητα της κατασκευής του δένδρου

επιθεμάτων στη μνήμη. Η απλοϊκή μέθοδος για την κατασκευή, ξεκινά εισάγοντας με τη σειρά όλα τα επιθέματα. Στο πρώτο βήμα, εισάγεται με χρήση μίας μοναδικής ακμής όλο το κείμενο και ο τερματικός χαρακτήρας , δηλαδή η ακολουθία S[0..n-1]$, και στα επόμενα βήματα εισάγονται διαδοχικά όλα τα επιθέματα S[i..n-1]$ στο ολοένα διογκούμενο δένδρο, για i από 1 έως n-1.

Ας συμβολισθεί με N_i το δένδρο κατά την εξέλιξη της κατασκευής όταν έχουν εισαχθεί όλα τα επιθέματα από το 0 ως το i. Το πρώτο δένδρο λοιπόν, N_0 περιλαμβάνει μία ακμή μεταξύ της ρίζας και ενός φύλλου με αριθμό 0. Η ακμή έχει ως ετικέτα όλο το κείμενο και τον τερματικό χαρακτήρα. Το δένδρο N_{i+1} κατασκευάζεται από το δένδρο N_i ως εξής: ξεκινώντας από τη ρίζα του N_i βρίσκουμε το μέγιστο μονοπάτι του οποίου η ετικέτα ταυτίζεται με ένα πρόθεμα του S[i+1..n-1]$. Το μονοπάτι αυτό βρίσκεται συγκρίνοντας τους χαρακτήρες του S[i+1..n-1] με τους χαρακτήρες ενός μοναδικού μονοπατιού, μέχρι η σύγκριση να μην οδηγεί σε ταύτιση. Είναι βέβαιο ότι το μονοπάτι αυτό είναι μοναδικό αφού εξ'ορισμού δύο ακμές που εξέρχονται από τον ίδιο κόμβο δεν μπορούν να έχουν ως ετικέτα μια υποακολουθία που ξεκινάει από τον ίδιο χαρακτήρα. Επίσης είναι βέβαιο ότι κάποια στιγμή δεν θα είναι δυνατή περαιτέρω ταύτιση, αφού λόγω του τερματικού χαρακτήρα δεν υπάρχει κανένα επίθεμα του κειμένου που να είναι πρόθεμα κάποιου άλλου επιθέματος. Όταν φθάσει σε αυτό το σημείο, ο αλγόριθμος βρίσκεται είτε σε κάποιον κόμβο είτε σε κάποιο σημείο στο μήκος μίας ακμής. Αν βρίσκεται πάνω σε ακμή, η οποία έχει ως ετικέτα την υποακολουθία [s, e] όπου τα s, e είναι οι θέσεις αρχής και τέλους μέσα στο κείμενο y, τότε ακριβώς μετά από τον χαρακτήρα με θέση στο κείμενο w στον οποίο συνέβη η τελευταία επιτυχής ταύτιση κατά την σύγκριση και ακριβώς πριν τον χαρακτήρα όπου αυτή η ταύτιση δεν ήταν επιτυχής, δηλαδή τον επόμενο του w, εισάγεται ένας νέος κόμβος και η αρχική ακμή διασπάται σε δύο, μία με ετικέτα την ακολουθία [s, w] που προσπίπτει στον νέο κόμβο και μία με ετικέτα [w+1,e] που εξέρχεται από τον νέο κόμβο. Στην συνέχεια, είτε η σύγκριση σταμάτησε σε κάποιο κόμβο είτε ο αλγόριθμος βρίσκεται στον νέο κόμβο που δημιουργήθηκε, δημιουργείται επιπλέον μία ακμή που εξέρχεται από τον τρέχοντα κόμβο, που έχει ως ετικέτα τους υπόλοιπους χαρακτήρες του επιθέματος που εισάγεται.

Η παραπάνω μέθοδος κατασκευής εξασφαλίζει σε κάθε περίπτωση ότι δύο ακμές που εξέρχονται από τον ίδιο κόμβο, δεν έχουν ως ετικέτα υποακολουθίες που ξεκινούν με τον ίδιο χαρακτήρα, αφού η σύγκριση σταματά στο σημείο που δεν γίνεται ταύτιση και στην συνέχεια κατασκευάζονται οι νέοι κόμβοι. Υποθέτοντας ότι έχουμε φραγμένο αλφάβητο, η απλοϊκή αυτή μέθοδος έχει πολυπλοκότητα $O(n^2)$, αφού γίνονται n+1 επαναλήψεις για την εισαγωγή n+1 επιθεμάτων και σε κάθε επανάληψη διασχίζονται O(n) χαρακτήρες στο μονοπάτι του εκάστοτε N_i δένδρου.

2.3.2 Η προσέγγιση του Ukkonen

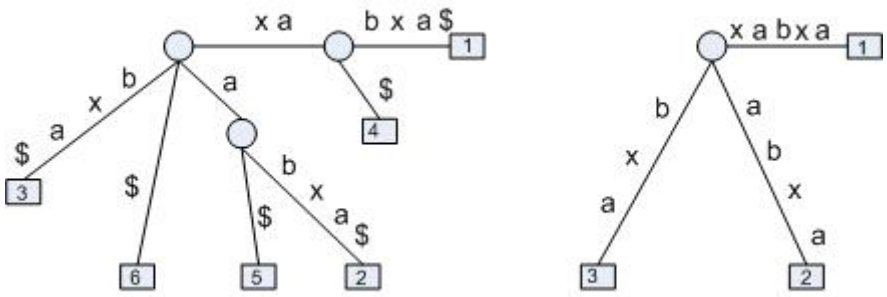
Ήδη από τα πρώτα χρόνια εμφάνισης των δένδρων επιθεμάτων, διατυπώθηκαν γραμμικοί ως προς το μέγεθος της εισόδου αλγόριθμοι κατασκευής. Ο πρώτος από αυτούς διατυπώθηκε από τον P. Weiner το 1973 [Wei73]. Ήταν λίγο δυσνόητος, όμως η σημασία του αποτελέσματος ήταν τόσο πρωτοφανής, ώστε φημολογείται ότι ο Donald Knuth, τον είχε χαρακτηρίσει ως αλγόριθμο της χρονιάς. Το 1976, ο E. M. McCreight [McC76] παρουσίασε έναν νέο γραμμικό ως προς την είσοδο αλγόριθμο, ο οποίος είναι πιο αποδοτικός από άποψη χώρου. Τέλος το 1995 ο E. Ukkonen παρουσίασε έναν νέο αλγόριθμο [Ukk95], που ενώ διατηρεί όλα τα χαρακτηριστικά του αλγορίθμου του McCreight είναι πιο εύκολος στην κατανόηση.

2.3.2.1 Ορολογία και γενική περιγραφή του αλγορίθμου

Στο επίκεντρο του αλγορίθμου βρίσκεται η έννοια του πεπλεγμένου δένδρου επιθεμάτων (implicit suffix tree). Το πεπλεγμένο δένδρο επιθεμάτων μιας ακολουθίας S προκύπτει από το δένδρο επιθεμάτων για την ακολουθία S$ μετά από:

- διαγραφή του τερματικού συμβόλου $ από τις ετικέτες των ακμών του δένδρου,
- αφαίρεση όλων των ακμών που δεν έχουν καμία ετικέτα (άρα αφαίρεση όλων των ακμών που είχαν ως ετικέτα μόνο τον χαρακτήρα $ πριν την διαγραφή του), και
- απομάκρυνση των κόμβων που δεν έχουν τουλάχιστον δύο παιδιά μέσω της συγχώνευσης προσπίπτουσας και εξερχόμενης ακμής και των αντίστοιχων ετικετών τους.

Ένα πεπλεγμένο δένδρο επιθεμάτων για μία ακολουθία S θα έχει λιγότερα φύλλα από το δένδρο επιθεμάτων για το S$ αν και μόνον αν τουλάχιστον ένα από τα επιθέματα του S είναι πρόθεμα κάποιου άλλου επιθέματος. Ο τερματικός χαρακτήρας άλλωστε προστέθηκε στην ακολουθία ακριβώς για να διασφαλίσει αυτή την συνθήκη. Αν ωστόσο η ακολουθία S τελειώνει με έναν χαρακτήρα που δεν απαντά σε κανένα άλλο σημείο του κειμένου τότε το πεπλεγμένο δένδρο επιθεμάτων για το S θα έχει ένα φύλλο για κάθε επίθεμα και θα είναι ένα γνήσιο δένδρο επιθεμάτων. Στο σχήμα φαίνεται το δένδρο επιθεμάτων για την ακολουθία xabxa$ και το αντίστοιχο πεπλεγμένο δένδρο επιθεμάτων όπου έχουν γίνει οι αφαιρέσεις τερματικών χαρακτήρων, ακμών, κόμβων, και οι συγχωνεύσεις ακμών για να γίνει πιο κατανοητή η έννοια του πεπλεγμένου δένδρου επιθεμάτων



Παρόλο που το πεπλεγμένο δένδρο επιθεμάτων δεν διαθέτει ένα φύλλο για κάθε επίθεμα, περιέχει όλα τα επιθέματα, υπάρχει δηλαδή μονοπάτι από τη ρίζα για κάθε επίθεμα, αλλά το μονοπάτι αυτό δεν καταλήγει σε κάθε περίπτωση σε κάποιο φύλλο, και ως εκ τούτου για τα επιθέματα για τα οποία το μονοπάτι τελειώνει σε κάποιο εσωτερικό κόμβο ή σε κάποια ακμή που δεν καταλήγει σε φύλλο δεν μπορεί να ανακτηθεί ο αριθμός επιθέματός τους, και άρα κατά την αναζήτηση κάποιας υποακολουθίας δεν μπορούν να ανακτηθούν εξαντλητικά όλες οι θέσεις μέσα στην ακολουθία όπου αυτή απαντά. Το πεπλεγμένο δένδρο επιθεμάτων φέρει λοιπόν λιγότερη πληροφορία από ότι το γνήσιο δένδρο επιθεμάτων. Θα συμβολίσουμε το πεπλεγμένο δένδρο επιθεμάτων της ακολουθίας S[1..i] με I_i όπου το i μπορεί να πάρει τιμές μεταξύ του 1 και του n.

Ο αλγόριθμος του Ukkonen κατασκευάζει ένα πεπλεγμένο δένδρο επιθεμάτων I_i για κάθε πρόθεμα S[1..i] ξεκινώντας από το I_1 και αυξάνοντας κατά 1 μέχρι την κατασκευή του I_n. Στο τέλος, το γνήσιο δένδρο επιθεμάτων κατασκευάζεται από το I_n και η ολική χρονική πολυπλοκότητα είναι O(n). Θα παρουσιαστεί αρχικά μία μέθοδος πολυπλοκότητας $O(n^3)$ και στην συνέχεια με βελτιστοποίηση της υλοποίησης θα φθάσουμε στο όριο πολυπλοκότητας που αξιώνεται.

Ο αλγόριθμος χωρίζεται σε n ***φάσεις*** (μία για κάθε δυνατό πρόθεμα του κειμένου). Σε κάθε φάση i ο αλγόριθμος κατασκευάζει το δένδρο I_i από το I_{i-1}. Κάθε φάση i αποτελείται από i σε πλήθος ***επεκτάσεις***, μία για κάθε επίθεμα του (τρέχοντος) προθέματος S[1..i]. Κατά την επέκταση j της φάσης i, ο αλγόριθμος βρίσκει πρώτα το τέλος του μονοπατιού που ξεκινάει από τη ρίζα για την υποακολουθία S[j..i-1], και στην συνέχεια το επεκτείνει προσθέτοντας τον χαρακτήρα S(i) στο τέλος του μονοπατιού (εκτός αν ο χαρακτήρας υπάρχει ήδη εκεί). Επομένως κατά την i φάση η ακολουθία S[1..i] τοποθετείται πρώτη κατά την πρώτη επέκταση και ακολουθούν οι S[2..i] στην δεύτερη επέκταση, η S[3..i] στην τρίτη επέκταση κοκ. Αφού το σύνολο των επιθεμάτων της ακολουθίας χρειάζεται χώρο $\Theta(n^2)$ και αφού η πολυπλοκότητα του αλγορίθμου δεν μπορεί να είναι μικρότερη από το μέγεθος της εξόδου του, για να μπορέσουμε να φθάσουμε τη γραμμική πολυπλοκότητα θεωρούμε ότι οι ακμές των δένδρων επιθεμάτων είναι συμπιεσμένες με χρήση δεικτών τέλους και αρχής μέσα στην

αρχική ακολουθία όπως περιγράφηκε παραπάνω κατά τον ορισμό των δένδρων επιθεμάτων, και άρα για την παράσταση μιας υποακολουθίας οσοδήποτε μεγάλου μήκους, χρειάζεται χώρος O(1) αφού απαιτείται σταθερός αριθμός δύο δεικτών και άρα για την παράσταση όλων των επιθεμάτων απαιτείται χώρος O(n).

2.3.2.2 Κανόνες επέκτασης επιθεμάτων

Για να καταστήσουμε αλγόριθμο την αφαιρετική περιγραφή των επεκτάσεων που δόθηκε παραπάνω πρέπει να οριστούν με λεπτομέρεια τα βήματα κατά την επέκταση προθέματος. Έστω S[j..i-1]=β το j-οστό επίθεμα του S[1..i-1]. Κατά την επέκταση j της φάσης i, όταν ο αλγόριθμος βρει το τέλος του β στο τρέχον δένδρο, επεκτείνει το β ώστε να εξασφαλίσει ότι το επίθεμα βS(i) βρίσκεται στο νέο πεπλεγμένο δένδρο. Η επέκταση αυτή γίνεται με βάση έναν από τους ακόλουθους τρεις κανόνες:

Κανόνας 1: Αν ο αλγόριθμος βρίσκει το τέλος του β σε κάποιο φύλλο του τρέχοντος δένδρου, τότε για την επέκταση πρέπει απλώς να προστεθεί ο χαρακτήρας S(i) στο τέλος της ετικέτας της προσκείμενης στο φύλλο ακμής.

Κανόνας 2: Αν δεν υπάρχει μονοπάτι αμέσως μετά το τέλος του μονοπατιού του β που να ξεκινάει με τον χαρακτήρα S(i) (αλλά τουλάχιστον ένα μονοπάτι συνεχίζει μετά το τέλος του β), τότε στο συγκεκριμένο σημείο δημιουργείται ένας νέος κόμβος (αν δεν υπάρχει ήδη κάποιος εκεί) και κάτω από αυτόν δημιουργείται μια νέα ακμή που καταλήγει σε νέο φύλλο με ετικέτα τον νέο χαρακτήρα.

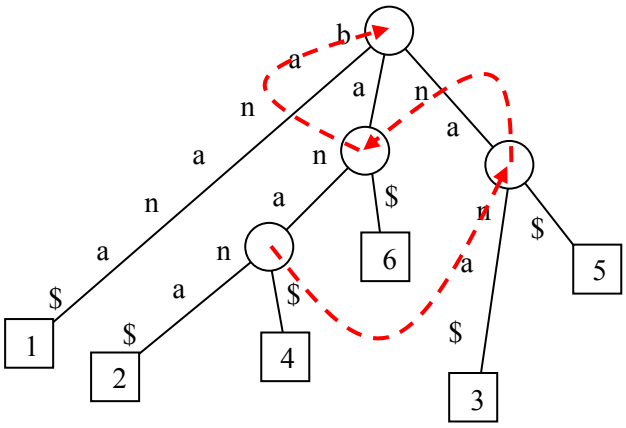
Κανόνας 3: Αν κάποιο μονοπάτι που ξεκινάει αμέσως μετά το τέλος του μονοπατιού του β ξεκινάει με τον χαρακτήρα S(i), τότε η ακολουθία βS(i) βρίσκεται ήδη μέσα στο δένδρο, άρα δεν πραγματοποιείται καμία ενέργεια. Αυτό γιατί σε ένα πεπλεγμένο δένδρο επιθεμάτων το τέλος ενός επιθέματος δεν χρειάζεται να καταδειχθεί.

Χρησιμοποιώντας τους παραπάνω κανόνες παρατηρούμε ότι αφού βρεθεί το τέλος του μονοπατιού του β μέσα στο τρέχον δένδρο, η επέκταση μπορεί να πραγματοποιηθεί σε σταθερό χρόνο. Τίθεται λοιπόν το αίτημα για αποδοτική εύρεση του τέλους του μονοπατιού για όλα τα επιθέματα του S[1..i-1]. Η απλοϊκή προσέγγιση βρίσκει το τέλος του μονοπατιού του β σε χρόνο $O(|\beta|)$ κάνοντας απλή διάσχιση του δένδρου ξεκινώντας από τη ρίζα. Με αυτή την προσέγγιση, η επέκταση j της φάσης i θα χρειαζόταν χρόνο O(i-j), το δένδρο I_i θα μπορούσε να κατασκευαστεί από το I_{i-1} σε χρόνο $O(i^2)$ και το δένδρο I_n σε συνολικό χρόνο $O(n^3)$. Η πολυπλοκότητα αυτή είναι χειρότερη από τον απλοϊκό τρόπο κατασκευής δένδρων επιθεμάτων που περιγράφηκε παραπάνω με πολυπλοκότητα $O(n^2)$, είναι όμως πιο εύκολο να περιγραφεί ο αλγόριθμος του Ukkonen ως βελτιστοποίηση της $O(n^3)$ παραπάνω μεθόδου. Η βελτιστοποίηση αυτή θα γίνει με τη χρήση διάφορων παρατηρήσεων και τρικ υλοποίησης.

Κάθε τρικ από μόνο του δείχνει ως μια εύλογη ευρετική μέθοδος για την επιτάχυνση του αλγορίθμου, ωστόσο δρώντας ξεχωριστά κάθε ένα από αυτά τα τρικς δεν μειώνει απαραίτητα το όριο πολυπλοκότητας για την χειρότερη περίπτωση. Ωστόσο, όταν λαμβάνονται όλα μαζί επιτυγχάνουν μια γραμμική πολυπλοκότητα στην χειρότερη περίπτωση. Ένα από τα πιο σημαντικά εργαλεία για την επιτάχυνση είναι οι **σύνδεσμοι επιθέματος** (suffix links)**.**

2.3.2.3 Σύνδεσμοι επιθέματος

Έστω χα μία ακολουθία, όπου x είναι ένας χαρακτήρας και α μια (πιθανώς κενή) υποακολουθία. Για έναν εσωτερικό κόμβο υ, ο οποίος έχει μονοπάτι από τη ρίζα έως αυτόν που κωδικοποιεί την ακολουθία χα, αν υπάρχει ένας άλλος κόμβος s(υ) με μονοπάτι α, τότε ο δείκτης από το υ στο s(υ) ονομάζεται σύνδεσμος επιθέματος (suffix link). Θεωρούμε επίσης κατά σύμβαση ότι αν το α είναι κενό ο σύνδεσμος επιθέματος δείχνει στη ρίζα, που δεν θεωρείται εσωτερικός κόμβος και δεν έχει σύνδεσμο επιθέματος που να ξεκινάει από αυτήν. Αν και από τον ορισμό δεν υπονοείται ότι κάθε εσωτερικός κόμβος ενός πεπλεγμένου δένδρου επιθεμάτων θα έχει έναν σύνδεσμο επιθεμάτων, στην πραγματικότητα, θα έχει. Ακολουθεί ως παράδειγμα το δένδρο επιθεμάτων της ακολουθίας «banana» με ενσωματωμένους του συνδέσμους επιθέματος:



Αντίστοιχα, οι σύνδεσμοι επιθέματος μπορούν να υπάρχουν και στα πεπλεγμένα δένδρα επιθεμάτων, που κατασκευάζονται στις διάφορες φάσεις του αλγορίθμου του Ukkonen. Μπορούμε να αποδείξουμε ότι αν ένας νέος εσωτερικός κόμβος υ με μονοπάτι-ετικέτα χα προστεθεί στο δένδρο κατά την επέκταση j κάποιας φάσης i, τότε είτε το μονοπάτι με ετικέτα την ακολουθία α τελειώνει σε έναν ήδη υπάρχοντα εσωτερικό κόμβο, είτε ένας εσωτερικός κόμβος με μονοπάτι-ετικέτα α θα δημιουργηθεί στην αμέσως επόμενη επέκταση j+1 της ίδιας φάσης. Πράγματι, όταν δημιουργείται ένας νέος εσωτερικός κόμβος κατά την επέκταση j τότε έχουμε την εφαρμογή του Κανόνα 2 κατά την επέκταση του επιθέματος αφού είναι ο μόνος

κανόνας κατά τον οποίο δημιουργούνται νέοι εσωτερικοί κόμβοι, δηλαδή το μονοπάτι-ετικέτα χα συνέχιζε με κάποιον χαρακτήρα διάφορο του S(i), έστω c. Αυτό σημαίνει, ότι στην επόμενη επέκταση j+1 θα υπάρχει οπωσδήποτε στο δένδρο κάποιο μονοπάτι-ετικέτα α και θα συνεχίζει και αυτό με τον χαρακτήρα c και ίσως και με κάποιον άλλον. Αν συνεχίζει μόνο με τον χαρακτήρα c ο Κανόνας 2 θα δημιουργήσει έναν νέο εσωτερικό κόμβο, αν συνεχίζει με πάνω από έναν χαρακτήρες τότε υπάρχει ήδη εσωτερικός κόμβος στο τέλος του μονοπατιού με ετικέτα το α. Άρα *κάθε νέος εσωτερικός κόμβος θα έχει έναν σύνδεσμο επιθέματος μέχρι το τέλος της επόμενης επέκτασης.*

Επομένως, μόλις ολοκληρωθούν όλες οι επεκτάσεις μιας φάσης, το πεπλεγμένο δένδρο επιθεμάτων θα διαθέτει όλους τους συνδέσμους επιθέματος. Αυτοί οι σύνδεσμοι χρησιμοποιούνται προκειμένου να επιτύχουν την επιτάχυνση της διάσχισης του μονοπατιού από τη ρίζα σε κάθε επέκταση. Για την πρώτη επέκταση, για τη φάση i-1, μπορούμε να βρούμε το τέλος του μονοπατιού, αν σε κάθε φάση κρατάμε έναν δείκτη στο φύλλο που αντιπροσωπεύει την πλήρη ακολουθία της φάσης S[1..i-1]. Στην επόμενη φάση, η πρώτη επέκταση αρκεί να χρησιμοποιήσει τον δείκτη αυτό, αφού η ακολουθία S[1..i] εισάγεται στο νέο πεπλεγμένο δένδρο (πάντα) με χρήση του κανόνα 1. Η πρώτη επέκταση κάθε φάσης λοιπόν είναι ειδική περίπτωση και λαμβάνει χώρα σε σταθερό χρόνο, δεδομένου ότι κρατάμε έναν δείκτη στο φύλλο όπου καταλήγει το μεγαλύτερο σε μήκος μονοπάτι του δένδρου. Αν συμβολίσουμε με υ τον κόμβο από τον οποίο προσπίπτει η ακμή στο φύλλο 1, και την ακολουθία S[1..i] με χα όπου κατά τα γνωστά x είναι ένας χαρακτήρας και α μία ακολουθία, στην δεύτερη επέκταση ο αλγόριθμος θα πρέπει να βρει το τέλος του μονοπατιού για το S[2..i]=α. Ο κόμβος υ είναι είτε εσωτερικός κόμβος είτε η ρίζα, δεν μπορεί δηλαδή εξ ορισμού να είναι φύλλο. Αν είναι η ρίζα η αναζήτηση για το μονοπάτι του α θα πρέπει να ξεκινήσει από τη ρίζα κατά τα γνωστά. Αν όμως είναι εσωτερικός κόμβος τότε ο κόμβος αυτός σύμφωνα με τα παραπάνω θα διαθέτει κάποιον σύνδεσμο επιθέματος που θα δείχνει σε κάποιον κόμβο s(υ) που κωδικοποιεί ένα μονοπάτι που είναι πρόθεμα του α, επειδή δημιουργήθηκε σε προηγούμενη φάση, και το μονοπάτι του α θα βρίσκεται μέσα στο υποδένδρο του s(υ). Γίνεται φανερό ότι η διάσχιση του δένδρου για την εύρεση του μονοπατιού του α μπορεί να αρχίσει από τον κόμβο s(υ). Η ίδια ιδέα επαναλαμβάνεται για τις επόμενες επεκτάσεις, με την διαφορά ότι για τις επεκτάσεις j με j>2, αν ο κόμβος στον οποίο βρισκόμαστε έχει ήδη κάποιον σύνδεσμο επιθέματος, δεν είναι δηλαδή ένας νεοδημιουργηθείς κόμβος, τότε διασχίζουμε το δένδρο με χρήση αυτού του συνδέσμου, δεν χρειάζεται δηλαδή να γίνει μετάβαση πάνω, στον κόμβο υ. Σε κάθε περίπτωση πάντως, ο αλγόριθμος θα χρειαστεί να ανεβεί το πολύ έναν κόμβο από την τρέχουσα θέση του για την αναζήτηση του συνδέσμου επιθέματος. Κάθε επέκταση λοιπόν j, με j≥2, που

πραγματοποιείται κατά την φάση i, με την χρήση των συνδέσμων επιθέματος μπορεί να υλοποιηθεί με τα ακόλουθα βήματα:

Βήμα 1. Εύρεση του πρώτου κόμβου υ που βρίσκεται πάνω στο τέλος του μονοπατιού για το S[j-1..i-1] ή ακριβώς πάνω από αυτό και διαθέτει σύνδεσμο επιθέματος ή είναι η ρίζα. Το βήμα αυτό απαιτεί το πολύ διάσχιση ενός κόμβου ξεκινώντας από το τέλος του S[j-1..i-1] στο τρέχον δένδρο. Έστω γ η (πιθανώς κενή) ακολουθία-ετικέτα μεταξύ του κόμβου υ και του τέλους της ακολουθίας S[j-1..i-1]

Βήμα 2. Αν ο υ δεν είναι η ρίζα, μετάβαση με την χρήση του συνδέσμου επιθεμάτων στον κόμβο s(υ) και μετά κατάβαση προς τα κάτω μέχρι το τέλος του μονοπατιού για το γ. Αν το υ είναι η ρίζα τότε διάσχιση του μονοπατιού για το S[j..i-1] όπως και με τον απλοϊκό αλγόριθμο

Βήμα 3. Χρήση των κανόνων επέκτασης επιθεμάτων, για να εξασφαλιστεί ότι η ακολουθία S[j..i-1]S(i) βρίσκεται μέσα στο δένδρο

Βημα 4. Αν κατά την επέκταση j-1 είχε δημιουργηθεί ένας νέος εσωτερικός κόμβος ω, τότε η ακολουθία S[j..i-1]S(i) θα πρέπει να τελειώνει στον κόμβο s(ω). Δημιουργία του συνδέσμου επιθέματος (ω,s(ω))....

2.3.2.4 Τα τρικ του αλγορίθμου

Τρικ 1- Skip and count: Η παραπάνω περιγραφή φαίνεται όντως να βελτιώνει τον χρόνο διάσχισης του δένδρου, ωστόσο δεν βελτιώνει το όριο πολυπλοκότητας. Για την βελτίωση του όριου πολυπλοκότητας είναι απαραίτητη η εφαρμογή του τρικ skip and count. Ας θεωρήσουμε ότι με χρήση κάποιου συνδέσμου επιθέματος μεταβαίνουμε στον κόμβο s(υ). Όπως είπαμε το μονοπάτι του α θα τελειώνει μέσα στο υποδένδρο του s(υ). Αν θεωρήσουμε ότι το μονοπάτι από τη ρίζα ως το s(υ) έχει μήκος $|\beta|$, τότε το μήκος του μονοπατιού γ που θα πρέπει να διασχίσουμε στο υποδένδρο του s(υ) έχει μήκος g=$|\gamma| = |\alpha| - |\beta|$. Με μία απλοϊκή προσέγγιση, η διάσχιση του μονοπατιού μέχρι το τέλος του μονοπατιού για το α, θα γινόταν κάνοντας σύγκριση των χαρακτήρων έναν προς έναν σε χρόνο ανάλογο του g. Το τρικ του skip και count βοηθάει να βελτιωθεί ο χρόνος αυτού του κομματιού. Αφού δεν υπάρχουν δύο ακμές που ξεκινούν από έναν κόμβο και έχουν ως ετικέτα μία υποακολουθία με τον ίδιο πρώτο χαρακτήρα, ο πρώτος χαρακτήρας του γ θα εμφανίζεται σε ακριβώς μία ακμή που εξέρχεται από το s(υ). Αν ο αριθμός των χαρακτήρων πάνω στην ακμή είναι g', και g'<g τότε ο αλγόριθμος δεν χρειάζεται να εξετάσει κανέναν άλλον από τους χαρακτήρες πάνω στην ακμή, και μπορεί να μεταβεί κατευθείαν στον επόμενο κόμβο του μονοπατιού. Με το ίδιο σκεπτικό στον επόμενο κόμβο χρειάζεται να εξεταστεί μόνο ο πρώτος χαρακτήρας, και συνεχίζουμε διασχίζοντας κόμβους μέχρι να φθάσουμε στο τέλος του μονοπατιού για το α.

Καθ αυτό τον τρόπο, η διάσχιση γίνεται ανάλογη του αριθμού των κόμβων στο μονοπάτι του γ και όχι ανάλογη του μήκους του γ. Με μία απλή συλλογιστική μπορεί να αποδειχθεί ότι το βάθος ενός κόμβου υ στο δένδρο σε μια δεδομένη στιγμή (όπου βάθος θεωρούμε τον αριθμό τον κόμβων από τη ρίζα ως τον υ) είναι το πολύ κατά ένα μεγαλύτερο από το βάθος του s(υ). Καθόλη τη διάρκεια της φάσης λοιπόν, αφού κατά την ανάβαση προς τα πάνω για την εύρεση του κόμβου με τον σύνδεσμο επιθέματος το βάθος μειώνεται το πολύ κατά ένα, στη συνέχεια με τον σύνδεσμο επιθέματος το βαθος μειώνεται πάλι το πολύ κατά ένα, και καταλήγει σε κάποιον κόμβο μεγαλύτερου βαθους. Αφού κανένας κόμβος δεν μπορεί να έχει βάθος μεγαλύτερο του n, η συνολική αυξομοίωση του βαθους καθόλη τη διάρκεια της φάσης φράσσεται από το 3n, και αφού η χρονική πολυπλοκότητα είναι ανάλογη του αριθμού των κόμβων που παρεμβάλλονται στα μονοπάτια, η χρονική πολυπλοκότητα για κάθε φάση είναι O(n). Αφού υπάρχουν n φάσεις, η συνολική χρονική πολυπλοκότητα με την χρήση των συνδέσμων επιθέματος πέφτει σε $O(n^2)$.

Τρικ 2: Για τη χρήση ενός δεύτερου τρικ για την περαιτέρω επιτάχυνση του αλγορίθμου μας είναι χρήσιμη μια παρατήρηση για τους κανόνες επέκτασης των επιθεμάτων. Παρατηρούμε ότι ο κανόνας 3, αν εφαρμοστεί μία φορά κατά τη διάρκεια μιας φάσης, εφαρμόζεται στη συνέχεια για όλες τις επόμενες επεκτάσεις της φάσης αυτής. Πράγματι, αν για την επέκταση j της φάσης i+1 εφαρμοστεί ο κανόνας 3, αυτό σημαίνει ότι η ακολουθία S[j..i] έχει ήδη τον χαρακτήρα S(i+1),άρα το ίδιο θα ισχύει και για όλες τα επιθέματα της S[j..i], δηλαδή τα S[j+1..i], S[j+2..i] κοκ. Επομένως το δεύτερο τρικ συνίσταται στην ολοκλήρωση κάποιας φάσης, πριν το τέλος των επεκτάσεων, όταν για οποιαδήποτε επέκταση εφαρμοστεί ο κανόνας 3.

Και πάλι, το τρικ αυτό συνιστά μία καλή ευρετική μέθοδο, ωστόσο από μόνο του δεν γίνεται σαφές πως βελτιώνει το όριο της πολυπλοκότητας για την χείριστη περίπτωση. Για αυτό, χρειάζεται μια ακόμα παρατήρηση και ένα τρικ

Τρικ 3: Το τρίτο τρικ βασίζεται στην εξής παρατήρηση: Αν ένα φύλλο δημιουργηθεί κατά τη διάρκεια της κατασκευής ενός πεπλεγμένου δένδρου I_i τότε αυτό θα παραμείνει φύλλο για όλες τα διαδοχικά πεπλεγμένα δένδρα επιθεμάτων που θα δημιουργήσει ο αλγόριθμος. Πράγματι, ο αλγόριθμος δεν διαθέτει κανέναν κανόνα επέκτασης με τον οποίο μια ακμή φύλλου μπορεί να επεκταθεί πέραν του τρέχοντος φύλλου. Με άλλα λόγια, αν κατά την επέκταση j υπάρχει ένα φύλλο με αριθμό επιθέματος j, τότε ο κανόνας 1 θα χρησιμοποιηθεί για όλες τις μελλοντικές επεκτάσεις j σε όλες τις φάσεις του αλγορίθμου που θα ακολουθήσουν. Στην φάση 1 υπάρχει μόνο μία επέκταση κατά την οποία δημιουργείται το φύλλο 1, επομένως σε κάθε φάση i υπάρχει μια αρχική ακολουθία διαδοχικών επεκτάσεων (ξεκινώντας με την επέκταση 1), κατά την οποία εφαρμόζονται οι κανόνες 1 ή 2. Αν συμβολίσουμε με j_i την τελευταία αυτή επέκταση, αφού κάθε νέα δημιουργία φύλλου

προϋποθέτει την εφαρμογή του κανόνα 2, συμπεραίνουμε ότι $j_i \leq j_{i+1}$ δηλαδή η ακολουθία αυτή δεν μπορεί ποτέ να συρρικνώνεται μεταξύ των φάσεων. Μπορούμε λοιπόν κατά την φάση i+1 να εφαρμόσουμε την επέκταση των ακμών με χρήση του κανόνα 1 για για όλες τις επεκτάσεις από 1 έως j_i με υπολογιστικό κόστος O(1) ένα εφαρμόσουμε το εξής τρικ: Για κάθε νέο φύλλο που δημιουργείται με τον κανόνα 2, κατά την φάση i+1, και έχει ως μονοπάτι ετικέτα στην ακμή φύλλου του την υποακολουθία S[p..i+1], κατά την συμπίεση της ετικέτας ακμής με δείκτες αρχή και τέλους δεν χρησιμοποιούμε το ζεύγος [p,i+1], άλλα το ζέυγος [p,e], όπου ο e είναι ένας **καθολικός δείκτης** που δείχνει στο εκάστοτε τελευταίο χαρακτήρα της αρχικής ακολουθίας που θα εισαχθεί στο δένδρο στην τρέχουσα φάση, άρα είναι e=i+1, για την φάση i+1. Κατά την εκκίνηση μιας νέας φάσης πρέπει το e να αυξηθεί κατά 1, πράξη που γίνεται σε σταθερό χρόνο και αυτόματα ενημερώνει όλα τα φύλλα χωρίς να χρειάζεται να εισαχθεί ο καινούριος τελευταίος χαρακτήρας με χρήση του κανόνα 1. Όλες οι επεκτάσεις λοιπόν από 1 έως j_i κατά την φάση i+1 μπορούν να πραγματοποιηθούν σε σταθερό χρόνο.

Η κάθε φάση i+1 λοιπόν μπορεί με χρήση των τρικ 1 και 2 να πραγματοποιηθεί με τα ακόλουθα βήματα:

Βήμα 1. Αύξηση του καθολικού δείκτη e σε i+1, πραγματοποιώντας έτσι λόγω του τρικ 3 όλες τις επεκτάσεις από το 1 εως το j_i

Βήμα 2. Χρησιμοποίηση του αλγορίθμου για τις επεκτάσεις, για πλήρη διαδοχικό υπολογισμό των επεκτάσεων από j_i +1 έως την πρώτη επέκταση j^* κατά την οποία εφαρμόζεται ο κανόνας επέκτασης 3, ή μέχρι το τέλος όλων των επεκτάσεων αν ο κανόνας 3 δεν εφαρμόζεται πουθενά. Έτσι, λόγω του τρικ 2 πραγματοποιούνται όλες οι πρόσθετες επεκτάσεις j^* έως και i+1

Βήμα 3. Το j_{i+1} τίθεται ίσο με j^*-1 για την προετοιμασία της επόμενης φάσης

Θα δειχθεί στη συνέχεια ότι ο αλγόριθμος του Ukkonen με χρήση των συνδέσμων επιθέματος και με τα τρικ υλοποίησης 1, 2 και 3 μπορεί να κατασκευάζει όλα τα πεπλεγμένα δένδρα επιθεμάτων I_1 έως I_n σε συνολικό χρόνο O(n). Ας συμβολισθεί με j' μια επέκταση ανεξαρτήτως φάσης που γίνεται με πλήρη τρόπο, δηλαδή χωρίς την χρήση των τρικ 2 και 3. Κατά την διάρκεια εκτέλεσης όλου του αλγορίθμου το j' ξεκινάει από το 1 για την πρώτη φάση, και συνεχίζει να αυξάνεται. Κατά την διάρκεια μιας φάσης i το j' συνεχίζει από κει που ήταν φθάνει μέχρι την επέκταση j^*, στην συνέχεια μεταβαίνει στην επόμενη φάση αφού όλες οι υπόλοιπες επεκτάσεις είναι περιττές λόγω του τρικ 2, η πρώτη επέκταση της επόμενης φάσης συνεχίζει χωρίς να αυξάνεται το j' και συνεχίζονται οι επόμενες επεκτάσεις μέχρι την νέα τελευταία επέκταση j^* κοκ. Αφού το j' φράσσεται από το n, και αφού υπάρχουν n φάσεις, και το j' πότε δεν μειώνεται αλλά αυξάνεται κατά την διάρκεια των φάσεων ενώ μένει ίδιο κατά την μετάβαση από μια φάση στην επόμενη, και αφού υπάρχουν ακριβώς n μεταβάσεις από φάση σε φάση, ο αλγόριθμος εκτελεί ακριβώς 2n πλήρεις επεκτάσεις. Όσον αφορά τον

αριθμό των διασχίσεων κόμβων, όπως δείξαμε προηγουμένως κατά την διάρκεια μίας φάσης είναι O(n). Πρέπει όμως να δείξουμε ότι είναι O(n) καθόλη την διάρκεια του αλγορίθμου. Όπως αναφέρθηκε, κατά την μετάβαση από μία φάση σε μία άλλη, το j' παραμένει σταθερό. Συνεπώς το βάθος του κόμβου παραμένει σταθερό κατά την μετάβαση από μία φάση σε μία άλλη. Κατά την διάρκεια μιας πλήρους επέκτασης όπως είδαμε, πραγματοποιούνται το πολύ δύο διασχίσεις προς τα πάνω, άρα μειώνεται το βάθος κόμβου και στην συνέχεια κατά την διάσχιση προς τα κάτω αυξάνεται το βάθος κατά των αριθμό των κόμβων που διασχίζονται προς τα κάτω. Αφού υπάρχουν 2n αναλυτικές επεκτάσεις, και το μέγιστο βάθος κόμβου δεν μπορεί να ξεπερνάει το n, έπεται ότι ο μέγιστος αριθμός διασχίσεων κόμβων είναι O(n). Άρα ο αλγόριθμος κατασκευάζει το I_n δένδρο μετά από n φάσεις σε χρόνο O(n).

2.3.2.5 Δημιουργία του γνήσιου δένδρου επιθεμάτων

Μετά την δημιουργία του I_n πεπλεγμένου δένδρου, μπορεί να κατασκευάστεί το γνήσιο δένδρο επιθεμάτων σε επιπλέον χρόνο O(n). Προστίθεται ο τερματικός χαρακτήρας στην ακολουθία και συνεχίζουμε την διαδικασία του αλγορίθμου από τη φάση όπου είχε σταματήσει. Το δένδρο που θα δημιουργηθεί δεν είναι πεπλεγμένο λόγω του μοναδικού τερματικού χαρακτήρα, δηλαδή σε κάθε επίθεμα θα αντιστοιχεί και ένα φύλλο. Απομένει η αντικατάσταση του καθολικού δείκτη e σε κάθε φύλλο του δένδρου με την τιμή n, το οποίο μπορεί να γίνει με μία πλήρη διάσχιση όλων των φύλλων του δένδρου, κάτι που απαιτεί χρόνο O(n). Άρα ο αλγόριθμος του Ukkonen μπορεί να κατασκευάσει το γνήσιο δένδρο επιθεμάτων της ακολουθίας S και όλους τους συνδέσμους επιθέματος των εσωτερικών του κόμβων σε συνολικό χρόνο O(n).

3 *Διαχείριση δένδρων επιθεμάτων που ξεπερνούν σε μέγεθος την μνήμη*

3.1 Το πρόβλημα

Όπως παρουσιάστηκε στο κεφάλαιο 2, τα δένδρα επιθεμάτων είναι μία από τις πιο σημαντικές δομές για την υλοποίηση αλγορίθμων σε ακολουθιακά δεδομένα και για την ευρετηριοποίηση και ανάλυση βιολογικών ακολουθιών. Δυστυχώς, όταν αυτοί οι αλγόριθμοι υλοποιούνται σε πραγματικά βιολογικά δεδομένα, που σε πολλές περιπτώσεις είναι αρκετά μεγάλα για να χωρέσουν στην μνήμη, παρατηρούνται φαινόμενα συμφόρησης (bottlenecks) αφού θα πρέπει να υπάρξει αποθήκευση κάποιου μέρους των δεδομένων στον δίσκο. Αυτό έχει σαν αποτέλεσμα την ραγδαία πτώση της απόδοσης των αλγορίθμων είτε αυτό αφορά αλγορίθμους κατασκευής των δένδρων, είτε αλγορίθμους αναζήτησης δεδομένων σε υπάρχοντα δένδρα. Ένα χαρακτηριστικό των δένδρων επιθεμάτων που επιβαρύνει την κατάσταση είναι ότι σαν δομή δεν έχουν τοπικότητα της αναφοράς (locality of reference) αφού η αναζήτηση από την ρίζα μπορεί να καταλήξει κυριολεκτικά σε οποιοδήποτε από τα πάρα πολλά σε αριθμό φύλλα, και αν η δομή δεν χωράει στην μνήμη, οποιαδήποτε αναζήτηση προτύπου απαιτεί να γίνει προσπέλαση στον δίσκο αρκετές φορές .

Επιπλέον, ο χώρος που καταλαμβάνει το δένδρο επιθεμάτων είναι πολύ μεγάλος και έτσι το δένδρο ως δομή μπορεί να θεωρηθεί αρκετά σπάταλο σε σχέση με το μέγεθος της ακολουθίας από την οποία κατασκεύαζεται. Πράγματι, αν έχουμε μια ακολουθία μεγέθους n του αλφαβήτου Σ={A,C,T,G}, δηλαδή με σ=4, μπορεί να αποθηκευτεί σε nlogσ = 2n bits, ενώ το δένδρο επιθεμάτων της ακολουθίας αυτής καταλαμβάνει O(nlogn) bits. Στην πράξη η διαφορά μεγέθους μπορεί πολύ εύκολα να φθάσει ένα λόγο 50:1.

Το γεγονός αυτό δημιουργεί την ανάγκη για νέους αλγορίθμους που μπορούν να αυξάνουν την απόδοση κατασκευής, της αποθήκευσης και της αναζήτησης. Συχνά η απόδοση για ένα από αυτά βελτιώνεται εις βάρος των υπολοίπων οπότε η χρήση κάθε αλγορίθμου εξαρτάται από τη στάθμιση που γίνεται για τις ανάγκες της κάθε συγκεκριμένης εφαρμογής. Προσεγγίσεις όπως ο αλγόριθμος της Hunt [HAI01], ο TDD, ο TOP-Q, ο TRELLIS, κ.ά. που στοχεύουν στην αποδοτική κατασκευή δένδρων επιθεμάτων για μεγάλα δεδομένα βασίζονται σε μία στρατηγική κατακερματισμού της ακολουθίας και του δένδρου επιθεμάτων στην βάση κάποιου κριτηρίου. Αρκετοί αλγόριθμοι, όπως ο TDD, εγκαταλείπουν την χρήση συνδέσμων επιθέματος κατά την κατασκευή των δένδρων. Πέραν της χρησιμότητας τους για την γρήγορη διάσχιση των κόμβων κατά την δημιουργία των δένδρων με τον αλγόριθμο του Ukkonen οι σύνδεσμοι επιθέματος είναι απαραίτητοι για σημαντικό αριθμό αλγορίθμων, και αν υπάρχει σχεδιασμός να εφαρμοστεί κάποιος τέτοιος αλγόριθμος θα πρέπει να κατασκευαστούν οι σύνδεσμοι επιθέματος από το κατασκευασμένο δένδρο.

3.2 Κατακερματισμός στην βάση προθεμάτων σταθερού μήκους

Η Ela Hunt [HAI01] ήταν η πρώτη που παρουσίασε έναν αλγόριθμο για κατασκευή δένδρου επιθεμάτων στον δίσκο. Τα δένδρα της Hunt εκμεταλλεύονται την αναδρομική δομή των δένδρων επιθεμάτων. Αν υ είναι ένας οποιοσδήποτε κόμβος του δένδρου που έχει ως ετικέτα την ακολουθία p, τότε το υποδένδρο με ρίζα το υ περιέχει όλα τα επιθέματα που έχουν πρόθεμα το p. Το δένδρο λοιπόν μπορεί να διαμεριστεί στην βάση προθεματικών υποδένδρων.

Ο αλγόριθμος αρχικά βρίσκει την συχνότητα με την οποία απαντούν στην ακολουθία όλα τα προθέματα κάποιου συγκεκριμένου μήκους. Αν το μήκος αυτό είναι αρκούντως μεγάλο είναι πιθανό όλα τα προθεματικά υποδένδρα να χωρούν στην μνήμη. Όμως κάθε ένα από τα προθεματικά υποδένδρα δεν είναι πλέον ένα δένδρο επιθεμάτων αφού οι ακολουθίες που περιέχει δεν είναι όλα τα επιθέματα μιας ακολουθίας, αλλά μόνο συγκεκριμένα επιθέματα. Συνεπώς για την κατασκευή δεν μπορεί να χρησιμοποιηθεί ο αλγόριθμος του Ukkonen, αφού βασίζεται στην χρήση συνδέσμων επιθέματος για κάθε εσωτερικό κόμβο, οι οποίοι δεν μπορούν να ανακτηθούν αφού τα περισσότερα επιθέματα των υποακολουθιών βρίσκονται σε άλλους διαμερισμούς. Η δημιουργία των δένδρων, υποδένδρο προς υποδένδρο, πραγματοποιείται εγκταλείποντας εντελώς την χρήση συνδέσμων επιθέματος, οι οποίοι αν είναι επιθυμητό να υπάρχουν στο δένδρο θα πρέπει να βρεθούν μετά την ολοκλήρωση της

κατασκευής με κάποιο υπολογιστικό κόστος. Η κατασκευή λοιπόν των δένδρων και ο αλγόριθμος της Hunt έχει χρονική πολυπλοκότητα $O(n^2)$.

Ένα επιπλέον πρόβλημα είναι ότι ο αλγόριθμος βασίζεται στην υπόθεση ότι τα δεδομένα παρουσιάζουν μια σχεδόν ομοιόμορφη κατανομή, ως προς την συχνότητα των διαφόρων υποακολουθιών. Αυτό δεν ισχύει για τα βιολογικά δεδομένα, για παράδειγμα, στο ανθρώπινο γονιδίωμα οι βάσεις A, C, G και T απαντούν με συχνότητα 30%, 20%, 20% και 30% αντίστοιχα. Αν λοιπόν διαμερίζαμε το δένδρο στην βάση προθεμάτων μήκους 1 χαρακτήρα, δύο από τις διαμερίσεις (αυτές με πρόθεμα A και T) θα ήταν κατά 50% μεγαλύτερες από τις άλλες 2. Για προθέματα μεγαλύτερου μήκους η διασπορά είναι ακόμα μεγαλύτερη, και στην γενική περίπτωση, είναι δυνατόν πολλά υποδένδρα να χωρούν στην μνήμη και άλλες να μην χωρούν για κάποιο μήκος προθέματος. Για την επίλυση αυτού του προβλήματος προτάθηκε η χρήση bin-packing τεχνικών με ταυτόχρονη αύξηση του μήκους προθέματος. Δηλαδή αρχικά αυξάνεται το μήκος προθέματος όσο χρειάζεται ώστε όλα τα υποδένδρα να χωρούν στην μνήμη, και στην συνέχεια αναζητείται ένας βέλτιστος τρόπος ώστε για τα 'μικρά' υποδένδρα που χωράνε πολλά μαζί στην μνήμη να κατασκευάζεται ταυτόχρονα ένα μεγαλύτερο υποδένδρο για παραπάνω από ένα προθέματα. Τα πιο μεγάλα υποδένδρα ανεβαίνουν στην μνήμη μόνα τους, ή γίνεται ταυτόχρονη κατασκευή για μικρότερο αριθμό προθεμάτων. Η αύξηση όμως του μήκους προθέματος και η αναζήτηση των συχνοτήτων μέσα στην ακολουθία για όλα τα προθέματα έχει μεγάλο υπολογιστικό κόστος αφού για μήκος προθέματος μ, υπάρχουν 4^{μ} πιθανά προθέματα σε ακολουθίες DNA. Επιπλέον, η επίλυση του bin-packing είναι ένα δύσκολο πρόβλημα και η βέλτιστη επίλυση του είναι NP-hard.

Στην αρχική του εκδοχή ο αλγόριθμος πρότεινε την αποθήκευση του δένδρου στον δίσκο σε πλήρη μορφή, όχι δηλαδή κατακερματισμένο όπως στην μνήμη. Σε μεταγενέστερη εκδοχή το δένδρο επιθεμάτων αποθηκεύεται στον δίσκο κατακερματισμένο, ως δάσος. Η δημιουργός του αλγορίθμου υποστηρίζει ότι ο αλγοριθμος έχει ικανοποιητική απόδοση όταν το μέγεθος της μνήμης είναι τουλάχιστον έξι φορές μεγαλύτερο από την ακολουθία.

Ο αλγόριθμος της Hunt βελτιώθηκε από τον Japp [Jap04], που προτείνει μία μέθοδο συμπίεσης των δεδομένων και παράλληλης κατασκευής των δένδρων επιθεμάτων, μετά από κάποια προεπεξεργασία, ώστε να είναι δυνατή η χρήση συνδέσμων επιθέματος. Γίνεται βέβαια πάλι η υπόθεση ότι οι συχνότητες υποακολουθιών στα δεδομένα είναι ομοιόμορφα κατανεμημένες και οι διαμερισμοί περίπου ίσοι σε μέγεθος, επομένως η προσέγγιση αυτή πάσχει λόγω της ανομοιομορφίας των βιολογικών δεδομένων.

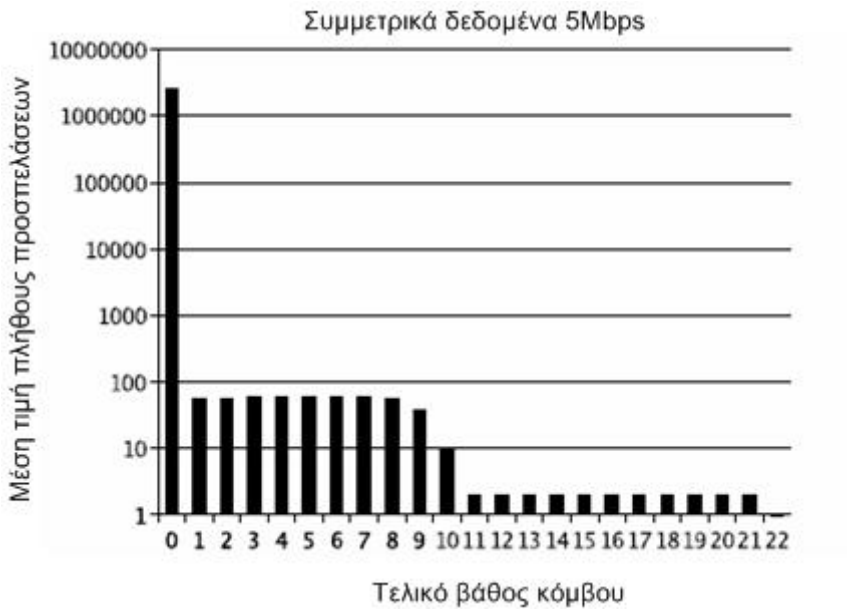
3.3 Οι αλγόριθμοι DynaCluster και TOP-Q

Οι αλγόριθμοι Dyna-Cluster [CYL05] και TOP-Q [BH04] αντιμετωπίζουν το πρόβλημα της ανισοκατανομής των δεδομένων αφού τα δένδρα που κατασκευάζουν έχουν τοπικότητα αναφοράς και άρα κατασκευάζονται αποτελεσματικά στην μνήμη με μικρή συμφόρηση, διατηρώντας την χρήση συνδέσμων επιθέματος.

Ο DynaCluster βασίζει την κατασκευή του δένδρου σε μία δυναμική ομαδοποίηση των δεδομένων της ακολουθίας. Τα ομαδοποιημένα δεδομένα έχουν τοπικότητα της αναφοράς, και το δένδρο κατασκευάζεται σε φάσεις, μία ομάδα ανά φάση. Ακολουθεί μια προαιρετική φάση για την δημιουργία των συνδέσμων επιθέματος, που είναι όμως πολύ χρονοβόρος σε σύγκριση με την φάση της κατασκευής.

Όπως αναφέρθηκε παραπάνω, το πρόβλημα με τον κατακερματισμό των βιολογικών δεδομένων στην βάση προθεμάτων συνίσταται στο γεγονός ότι οι διάφορες υποακολουθίες δεν έχουν την ίδια πιθανότητα εμφάνισης μέσα στην ακολουθία. Ο αλγόριθμος TOP-Q επιχειρεί να εφαρμόσει μια στρατηγική έξυπνου buffering, βασιζόμενος σε στοχαστικές ιδιότητες των κόμβων του δένδρου επιθεμάτων κατά την φάση της κατασκευής.

Ο αλγόριθμος για την κατασκευή του δένδρου επιθεμάτων μιας ακολουθίας S όπως είδαμε κατασκευάζει το δένδρο με χρήση συνδέσμων επιθέματος, εισάγοντας σε κάθε επανάληψη ένα νέο επίθεμα μέσα στο δένδρο. Σε κάθε επέκταση μίας φάσης είναι δυνατόν να εισάγονται μέσα στο δένδρο νέοι κόμβοι με διάσπαση υπαρχόντων ακμών. Το πόσο συχνά ένας κόμβος θα προσπελαστεί κατά την διάρκεια του αλγορίθμου έχει να κάνει με τις στοχαστικές ιδιότητες της ακολουθίας. Έχουν γίνει πολλές προσπάθειες για την ταξινόμηση των ακολουθιών στην βάση των στοχαστικών τους ιδιοτήτων. Ένα από τα απλούστερα μοντέλα που προτείνεται για να προσεγγιστούν οι στοχαστικές ιδιότητες ακολουθιών βιολογικών δεδομένων είναι αυτό των γεννητριών Bernoulli.



Σε αυτό το μοντέλο τα σύμβολα του αλφαβήτου κατά την δημιουργία της ακολουθίας επιλέγονται ανεξάρτητα από τα υπόλοιπα σύμβολα που έχουν επιλεγεί, η ακολουθία δηλαδή μπορεί να θεωρηθεί ως το αποτέλεσμα διαδοχικών δοκιμών Bernoulli. Αν επιπλέον όλα τα σύμβολα έχουν την ίδια πιθανότητα εμφάνισης στις δοκιμές, τότε η ακολουθία αποκαλείται συμμετρική, αλλιώς ασύμμετρη. Προφανώς μια συμμετρική ακολουθία, καθώς το μέγεθος της ακολουθίας μεγαλώνει τείνοντας προς το άπειρο, θα τείνει σε μια ακολουθία όπου οι διάφορες υποακολουθίες έχουν ομοιόμορφη κατανομή. Αν παρατηρήσουμε τα στατιστικά προσπέλασης των κόμβων για την δημιουργία ενός δένδρου επιθεμάτων με τον αλγόριθμο του Ukkonen σε μία συμμετρική ακολουθία βλέπουμε ότι η σχέση μεταξύ του μέσου αριθμού των προσπελάσεων που γίνονται σε έναν κόμβο και του τελικού βάθους του κόμβου στο δένδρου καταδεικνύει ότι κόμβοι που βρίσκονται ψηλά στο δένδρο επιθεμάτων προσπελαύνονται περισσότερες φορές από ότι κόμβοι που βρίσκονται χαμηλότερα στο δένδρο.

Το ίδιο συμπέρασμα προκύπτει και από τα στατιστικά στοιχεία για μια ενδεικτική ακολουθία πραγματικών βιολογικών δεδομένων όπως φαίνεται από το σχήμα. Φαίνεται λοιπόν εύλογο να επιλέξουμε τους κόμβους που είναι ψηλότερα στο δένδρο για να αποθηκευτούν σε κάποια γρήγορη μνήμη cache ώστε να εξυπηρετούνται οι προσπελάσεις γρηγορότερα.

Ο αλγόριθμος λοιπόν πρέπει να έχει την δυνατότητα να αναγνωρίσει κατά την διάρκεια της κατασκευής του δένδρου τους κόμβους που έχουν μεγαλύτερη πιθανότητα να καταλήξουν σε μικρότερο βάθος στο δένδρο. Παρόλο που το βάθος ενός εσωτερικού κόμβου δεν μπορεί να υπολογιστεί επακριβώς, μια απλή παρατήρηση της δομής του δένδρου επιθεμάτων μας δίνει έναν τρόπο για μια ικανοποιητική εκτίμηση αυτής της τιμής με ακρίβεια. Θεωρώντας μια συμμετρική ακολουθία σημαντικού μεγέθους για ένα στοχαστικό μοντέλο Bernoulli, παρατηρούμε ότι

- Υποακολουθίες ίσου μήκους είναι εξίσου πιθανό να εμφανίζονται
- Αν s είναι μία υποακολουθία με μήκος $|s|$, τότε ο αριθμός των υποακολουθιών που απαντούν σε άλλα σημεία της ακολουθίας και έχουν ένα κοινό πρόθεμα με το s είναι ευθέως ανάλογος του $|s|$

Από τα παραπάνω συμπεραίνουμε ότι *όσο μεγαλύτερη σε μήκος είναι η ετικέτα μίας ακμής του δένδρου επιθεμάτων μιας συμμετρικής ακολουθίας Bernoulli, τόσο μεγαλύτερη είναι η πιθανότητα η ακμή αυτή να διασπαστεί καθώς θα εισάγονται νέα κόμβοι στο δένδρο.* Προφανώς όταν η ακμή φθάσει το μήκος 1, δεν μπορεί πλέον να διασπαστεί.

Από την παραπάνω διαπίστωση συμπεραίνουμε ότι ένας κόμβος μπορεί να συνεχίσει να κατεβαίνει στο δένδρο επιθεμάτων έως ότου η ακμή που προσπίπτει σε αυτόν έχει μόνο ένα σύμβολο ως ετικέτα. Κατά την διάσπαση μιας ακμής βέβαια αυξάνεται το βάθος όχι μόνο του κόμβου στον οποίον κατέληγε η ακμή αλλά και όλων των κόμβων του αντίστοιχου

υποδένδρου. Επομένως, το μήκος της υποακολουθίας όλου του μονοπατιού από τη ρίζα ως τον κόμβο είναι ένα άνω όριο για το τελικό βάθος κάθε κόμβου. Αν μεριμνήσουμε να κρατάμε το μήκος της υποακολουθίας μέσα σε κάθε κόμβο κατά την κατασκευή, μπορούμε να ανακτούμε γρήγορα αυτή την ποσότητα για κάθε κόμβο, και να την χρησιμοποιούμε ως εκτιμήτρια για το τελικό του βάθος. Με στατιστική ανάλυση της εκτίμησης βάθους κατά την κατασκευή δένδρων για πραγματικές ακολουθίες βιολογικών δεδομένων και του τελικού βάθους των κόμβων, προκύπτει το συμπέρασμα ότι για μικρό ποσοστό των κόμβων δεν γίνεται ικανοποιητική εκτίμηση, ενώ όσο αυξάνεται το μέγεθος του δένδρου επιθεμάτων τόσο μειώνεται η συχνότητα των εσφαλμένων εκτιμήσεων.

Για να εκμεταλλευτεί τις παραπάνω διαπιστώσεις, ο αλγόριθμος θεωρεί ότι κάθε σελίδα στον δίσκο περιέχει μια συλλογή κόμβων του δένδρου επιθεμάτων, είτε εσωτερικούς είτε φύλλα, αλλά όχι μια ανάμιξη και των δυο. Η ανάμιξη αποφεύγεται καθώς οι κόμβοι που είναι φύλλα έχουν μικρότερο μέγεθος εγγραφής επομένως σε μια σελίδα που έχει αποκλειστικά φύλλα μπορεί να επιτευχθεί μεγαλύτερη πυκνότητα κόμβων απ' ότι σε μια σελίδα εσωτερικών κόμβων. Η κάθε σελίδα χαρακτηρίζεται από την μέση τιμή του μήκους μονοπατιού όλων των κόμβων που περιέχει. Χρησιμοποιώντας την εκτίμηση βάθους για κάθε σελίδα, *η προτεραιότητα για παραμονή στην μνήμη δίνεται σε σελίδες με μικρότερο μήκος μονοπατιού.* Η μέθοδος αυτή ονομάζεται TOP (κορυφή) για να καταδείξει το γεγονός ότι προσπαθεί να κρατήσει στην μνήμη της σελίδες του δένδρου επιθεμάτων που εκτιμάται ότι περιέχουν τους κορυφαίους κόμβους του δένδρου.

Η τελική εκδοχή του αλγορίθμου ονομάζεται TOP-Q και περιλαμβάνει μια περαιτέρω βελτίωση στην παραπάνω μέθοδο. Η μέθοδος TOP εκμεταλλεύεται την συχνότερη στατιστικά προσπέλαση σε κόμβους με μικρότερα μονοπάτια, ωστόσο αγνοεί την ύπαρξη συσχετιζόμενων προσπελάσεων κατά την κατασκευή του δένδρου επιθεμάτων. Κατά την εφαρμογή του TOP , ένας μεγάλος αριθμός σελίδων που απομακρύνονται από το buffer και κρατούνται στον δίσκο, φαίνεται να προσπελαύνεται πολύ λίγες επαναλήψεις αργότερα, μετά από επεξεργασία μερικών δεκάδων επιθεμάτων. Η πολιτική έξωσης από την μνήμη των σελίδων που εφαρμόζει ο TOP, δείχνει να μην είναι πολύ εύστοχη για σημαντικό αριθμό σελίδων. Ο TOP-Q χωρίζει το buffer σε μία στοίβα σελίδων που διατηρούνται με την σειρά του μήκους των μονοπατιών τους, και μια μικρού μεγέθους ουρά στην οποία κρατούνται οι σελίδες που έχουν υποστεί έξωση από τον σωρό. Οι σελίδες που βρίσκονται στη στοίβα επιλέγονται για έξωση όπως και με την μέθοδο TOP. Ωστόσο, δεν απομακρύνονται από την μνήμη, αφού εισάγονται στην FIFO δομή της ουράς. Η ύπαρξη λοιπόν της ουράς εξασφαλίζει την καθυστέρηση της έξωσης των σελίδων από την μνήμη, ικανοποιώντας σχεδόν όλες τις αναφορές που θα γίνουν σε κόμβους των σελίδων αυτών μέσα σε ένα μικρό περιθώριο μετά την έξωση από την στοίβα. Οι εμπνευστές του αλγορίθμου έδειξαν ότι χρησιμοποιώντας μια

ουρά μόλις 10 σελίδων για συνολικό πλήθος χιλιάδων ή και εκατοντάδων χιλιάδων σελίδων, η απόδοση ήταν αρκετά βελτιωμένη.

Συγκρινόμενος με άλλους αλγορίθμους buffering ο TOP-Q παρουσιάζει μεγαλύτερο λόγο επιτυχίας (hit-rate) από τον LRU (Least Recently Used) και συχνά από τον αλγόριθμο 2Q, όταν χρησιμοποιείται για την κατασκευή δένδρων για βιολογικά δεδομένα. Ο πολύ δημοφιλής μάλιστα αλγόριθμος LRU, κατά την κατασκευή δένδρων επιθεμάτων, έχει σχεδόν τον ίδιο ρυθμό επιτυχίας με μία πολιτική έξωσης από την μνήμη σελίδων που επιλέγονται με τυχαίο τρόπο (Random Selection strategy). Αυτό είναι ενδεικτικό της μεγάλης έλλειψης τοπικότητας της αναφοράς που έχουν ως δομές τα δένδρα επιθεμάτων. Ο μεγαλύτερος λόγος επιτυχίας οδηγεί φυσικά και σε ταχύτερους χρόνους κατασκευής. Ο αλγόριθμος μειώνει κατά 75% τις λειτουργίες Εισόδου-Εξόδου για τον δίσκο, κρατώντας στο buffer μόλις το 25% του δένδρου.

Οι αλγόριθμοι DynaCluster και TOP-Q δείχνουν λοιπόν να υπερβαίνουν με επιτυχία την έλλειψη ισοκατανομής στα βιολογικά δεδομένα, ωστόσο από τα πειραματικά ευρήματα καταδεικνύεται ότι είναι αποδοτικά για μεγέθη ακολουθιών της τάξης των 100Mbps, ενώ η απόδοση πέφτει ραγδαία όταν το μήκος των ακολουθιών αυξάνεται, μπορούν λοιπόν να χρησιμοποιηθούν για την παραγωγή ευρετηρίων για ακολουθίες μεγέθους χρωμοσώματος αλλά δεν είναι κατάλληλες για την παραγωγή ευρετηρίων για το σύνολο του γονιδιώματος. (≈ 3Gbps)

3.4 Ο αλγόριθμος TDD

Ο αλγόριθμος TDD (Top-Down Disk Based) [THP04] είναι ο πρώτος αλγόριθμος που αξίωσε την παραγωγή ευρετηρίου με σχετικά αποδοτικό τρόπο για το σύνολο του ανθρώπινου γονιδιώματος. Βασίζεται στην μέθοδο Διαμερισμού και Εγγραφής Μόνο από-Πάνω-προς-τα-Κάτω (Partition and Write Only Top Down), που συνίσταται στην διαμέριση της ακολουθίας και την δημιουργία του υποδένδρου για κάθε διαμέριση με χρήση του αλγορίθμου wotd-eager.

3.4.1 Ο αλγόριθμος wotdeager

Ο αλγόριθμος wotdeager [GKJ03] διασφαλίζει ότι οποιοσδήποτε κόμβος του δένδρου θα γραφτεί μόνο μια φορά ξεκινώντας την κατασκευή από τη ρίζα, και κατεβαίνοντας κατασκευάζει διαδοχικά τα παιδιά της ρίζας, τα παιδιά των παιδιών της ρίζας κλπ. Αρχικά όλα τα επιθέματα της ακολουθίας τοποθετούνται σε έναν πίνακα και ταξινομούνται γρήγορα σε γραμμικό χρόνο με χρήση του αλγορίθμου count-sort στην βάση του πρώτου τους χαρακτήρα. Η ταξινόμηση μας επιτρέπει να χωρίσουμε τα επιθέματα σε 5 κατηγορίες,

ανάλογα με το αν ξεκινούν με A,T,C,G και $ αν έχουμε να κάνουμε με βιολογικά δεδομένα, ή σε κατηγορίες πλήθους όσο το μέγεθος του εκάστοτε αλφαβήτου αν έχουμε να κάνουμε με άλλου τύπου δεδομένα.

Για κάθε μία από τις κατηγορίες κάνουμε τα εξής. Αν μια κατηγορία δεν περιέχει κανένα επίθεμα δεν κάνουμε τίποτα. Αν περιέχει παραπάνω από 2 επιθέματα τότε είναι σίγουρο ότι υπάρχει κάποιος εσωτερικός κόμβος που ξεκινάει από τη ρίζα για αυτόν τον αρχικό χαρακτήρα που αντιστοιχεί σε αυτήν την κατηγορία. Κατασκευάζουμε λοιπόν αυτόν τον εσωτερικό κόμβο, και ως ετικέτα στην ακμή του βάζουμε το μέγιστο κοινό πρόθεμα (longest common prefix) όλων των επιθεμάτων που ανήκουν σε αυτή την κατηγορία. Αν περιέχει μόνο ένα επίθεμα, τότε από τη ρίζα ξεκινάει κάποιο φύλλο, οπότε κατασκευάζεται το φύλλο αυτό και ως ετικέτα έχει όλο το αντίστοιχο επίθεμα. Η ίδια διαδικασία επαναλαμβάνεται για όλες τις κατηγορίες.

Μόλις ολοκληρωθεί η κατασκευή των παιδιών της ρίζας και των αντίστοιχων ακμών η κατασκευή του δένδρου συνεχίζεται ως εξής. Για κάθε κατηγορία επιθεμάτων, αυξάνουμε τους δείκτες των επιθεμάτων που ανήκουν στην κατηγορία αυτή κατά το μήκος του μέγιστου κοινού προθέματος, και στην συνέχεια επαναλαμβάνουμε την ίδια διαδικασία για τα επιθέματα της κατηγορίας που είχε γίνει πριν. Τα επιθέματα λοιπόν θα ταξινομηθούν πάλι στην βάση του πρώτου χαρακτήρα και θα δημιουργηθούν 5 υποκατηγορίες. Θα δημιουργηθούν λοιπόν τα αντίστοιχα παιδιά του κάθε παιδιού της ρίζας. Η διαδικασία μπορεί να επαναληφθεί γραμμικά μέχρι την ολοκλήρωση της κατασκευής του δένδρου.

3.4.2 Η μέθοδος PWOTD

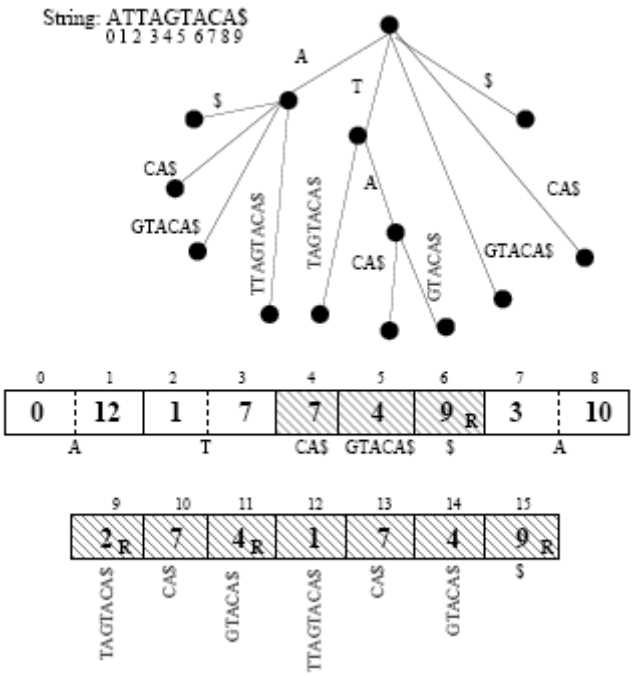
Η μέθοδος PWOTD αρχικά διαμερίζει την ακολουθία στην βάση προθεμάτων σταθερού μήκους, και για καθεμία από αυτές κατασκευάζει το δένδρο με χρήση του αλγορίθμου του wotdeager. Η μέθοδος απαιτεί την ύπαρξη τεσσάρων δομών δεδομένων για την κατασκευή των δένδρων. Οι δομές αυτές είναι:

- ο πίνακας της ακολουθίας εισόδου,
- ένας πίνακας με επιθέματα,
- ένας προσωρινός πίνακας και
- ένα δένδρο επιθεμάτων.

Αν το σταθερό μήκος προθεμάτων με βάση το οποίο γίνεται η διαμέριση είναι prefixlen ο πίνακας των επιθεμάτων γεμίζει με επιθέματα από μια διαμέριση μετά την αφαίρεση των πρώτων prefixlen χαρακτήρων. Ας θεωρηθεί ότι έχουμε την ακολουθία ATTAGTACA$ και κάνοντας διαμέριση στην βάση προθεμάτων μήκους 1, εκτελείται βήμα προς βήμα η διαδικασία της κατασκευής για την διαμέριση με πρόθεμα T, την οποία θα ονομάσουμε T-

διαμέριση. Τα επιθέματα σε αυτήν την διαμέριση βρίσκονται στις θέσεις 1,2 και 5 και άρα στον πίνακα επιθεμάτων θα αποθηκευτούν αυξημένα κατά 1 αφού prefixlen=1. Ο πίνακας λοιπόν των προθεμάτων είναι {2,3,6}. Το επόμενο βήμα απαιτεί την ταξινόμηση του πίνακα των επιθεμάτων στην βάση του πρώτου τους χαρακτήρα. Μόλις ολοκληρωθεί η ταξινόμηση αυτή, με ένα πέρασμα του πίνακα μετρώνται για κάθε χαρακτήρα τις εμφανίσεις του ως πρώτου χαρακτήρα σε κάθε επίθεμα και αποθηκεύονται οι δείκτες των επιθεμάτων στον Προσωρινό Πίνακα (temp). Μπορεί λοιπόν να προσδιοριστεί ο αριθμός των επιθεμάτων που έχει κάθε μία κατηγορία και τα αντίστοιχα όρια τους μέσα στον ταξινομημένο πίνακα των επιθεμάτων. Η κατηγορία Α (δηλαδή η κατηγορία των επιθεμάτων που ξεκινούν με Α) ξεκινάει στην θέση 0 του πίνακα (θεωρώντας υλοποίηση όπου το πρώτο στοιχείο των πινάκων έχει δείκτη 0) και έχει δύο στοιχεία, η κατηγορία Τ θα ξεκινάει στην θέση 2 και έχει 1 στοιχείο. Η κατηγορίες για τα G, C και $ είναι κενές. Αφού η κατηγορία Α έχει 2 επιθέματα θα αντιστοιχεί σε εσωτερικό κόμβο. Τα δύο αυτά επιθέματα ορίζουν πλήρως το υποδένδρο κάτω από αυτόν τον κόμβο. Επομένως φυλάσσεται χώρος στο δένδρο για να γραφθεί αυτός ο κόμβος όταν επεκταθεί με το πλήρες υποδένδρο του και ο κόμβος εισάγεται σε μία στοίβα. Αφού η Τ-κατηγορία έχει μόνο ένα μέλος, είναι κόμβος φύλλου και θα γραφθεί απευθείας στο δένδρο. Αφού δεν υπάρχουν άλλα παιδιά προς επεξεργασία, δεν υπάρχουν περαιτέρω εγγραφές στην στοίβα, και ο κόμβος αυτός θα εξαχθεί από την στοίβα πρώτος.

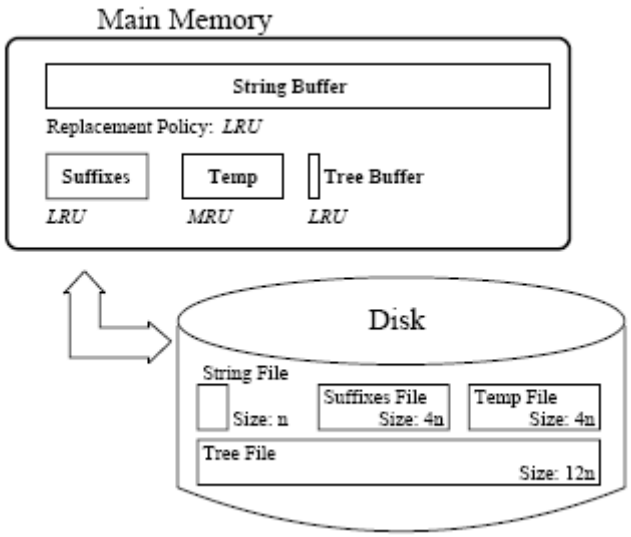
Μόλις ο κόμβος εξαχθεί από την στοίβα, ανακτάται το μέγιστο κοινό πρόθεμα όλων των κόμβων στην κατηγορία {3,6}. Εξετάζεται η θέση 4 (G) και η θέση 7 (C) και προσδιορίζεται το μήκος του μέγιστου κοινού προθέματος, που είναι 1. Κάθε δείκτης επιθέματος λοιπόν για αυτήν την κατηγορία αυξάνεται κατά 1 και το αποτέλεσμα υφίσταται αντίστοιχη επεξεργασία με πριν (ταξινόμηση στην βάση του πρώτου χαρακτήρα, υποκατηγορίες, δημιουργία εσωτερικών κόμβων και φύλλων). Ο υπολογισμός συνεχίζεται έως ότου όλοι οι κόμβοι επεκτείνονται με την δημιουργία των υποδένδρων τους και η στοίβα είναι κενή. Στο σχήμα φαίνεται το πλήρες δένδρο επιθεμάτων για όλες τις διαμερίσεις και η αναπαράσταση του



3.4.3 Απόδοση

Τίθεται το ερώτημα ποια θα ήταν η αποδοτικότερη μέθοδος buffering για κάθε μια από τις 4 δομές που είναι απαραίτητες για την κατασκευή του δένδρου με την μέθοδο PWTOD. Με μια απλή συλλογιστική συμπεραίνουμε ότι για τον Προσωρινό Πίνακα η πιο κατάλληλη μέθοδος είναι αυτή του MRU (most-recently used) αφού στον πίνακα αυτόν θα γίνει αναφορά μόνο κατά την ταξινόμηση σε 2 διαδοχικές σαρώσεις, μία για να αντιγραφούν τα επιθέματα από τον Πίνακα Επιθεμάτων, και μια για να αντιγραφούν τα ταξινομημένα επιθέματα πίσω στον Πίνακα Επιθεμάτων. Στην συνέχεια δεν θα υπάρξει ποτέ αναφορά σε αυτά τα δεδομένα. Αντίστοιχα οι κατάλληλη μέθοδος για τις υπόλοιπες δομές είναι η LRU (Least Recently Used).

Ο αλγόριθμος TDD γράφει το δένδρο από τα πάνω προς τα κάτω και δεν χρησιμοποιεί τους συνδέσμους επιθεμάτων, άρα εγκαταλείπει την γραμμική πολυπλοκότητα του Ukkonen. Ωστόσο, αν οι δομές χωρούν όλες στην μνήμη ο αλγόριθμος στην μέση περίπτωση έχει καλύτερη απόδοση από τον αλγόριθμο του Ukkonen. Αυτό οφείλεται στο γεγονός ότι ο αλγόριθμος του Ukkonen δεν έχει καθόλου τοπικότητα της αναφοράς και κατά την κατασκευή θα γίνουν προσπελάσεις σε πολλά διαφορετικά σημεία στην μνήμη. Σε σύγχρονους επεξεργαστες με cache ο TDD έχει καλύτερη απόδοση αφού η από πάνω προς τα κάτω κατασκευή εξασφαλίζει πολύ μεγάλη τοπικότητα της αναφοράς, άρα θα είναι πολύ περισσότερες οι φορές που ο αλγόριθμος θα διαβάζει τα ζητούμενα δεδομενα από την γρήγορη μνήμη cache, παρόλο που ο Ukkonen έχει καλύτερη θεωρητική πολυπλοκότητα για την χείριστη περίπτωση.



Για μεγαλύτερες ακολουθίες που δεν χωρούν στην μνήμη η buffering στρατηγική που εφαρμόζει ο αλγόριθμος σε συνδυασμό με την μέθοδο κατασκευής εξασφαλίζουν την αποδοτική ευρετηριοποίηση των δεδομένων, ακόμα και για πολύ ογκώδη δεδομένα επιπέδου γονιδιώματος.

Αν η ακολουθία δεν χωράει πλήρως στην μνήμη ο αλγόριθμος παρουσιάζει μεγάλη καθυστέρηση αν χρησιμοποιηθεί ως έχει λόγω των λειτουργιών I/O που γίνονται κατά την κατασκευή. Για την περίπτωση αυτή οι δημιουργοί του αλγορίθμου πρότειναν μια βελτίωση του TDD, τον ST-Merge [TTH+05], που χωρίζει την ακολουθία σε συνεχόμενες υποακολουθίες, χρησιμοποιεί τον TDD για την κατασκευή ενός δένδρου για κάθε μία από τις υποακολουθίες, και στην συνέχεια δημιουργεί το πλήρες δένδρο συγχωνεύοντας τα μικρότερα δένδρα.

Ο αλγόριθμος TDD και η παραλλαγή ST-Merge δείχθηκε πειραματικά ότι έχουν καλύτερη απόδοση από όλους τους προηγούμενους αλγορίθμους για κατασκευή των δένδρων στον δίσκο, συμπεριλαμβανομένου του TOP-Q. Το μεγαλύτερο μειονέκτημα σε σύγκριση με τον TOP-Q είναι η εγκατάλειψη των συνδέσμων επιθέματος, και αν υπάρχει απαίτηση το δένδρο να διαθέτει συνδέσμους επιθέματος, θα πρέπει να βρεθούν μετά την κατασκευή του δένδρου με το αντίστοιχο υπολογιστικό κόστος.

3.5 Η προσέγγιση του TRELLIS

Το 2007 προτάθηκε ο αλγόριθμος TRELLIS [PZ07] για την αποδοτική κατασκευή δένδρων επιθεμάτων στον δίσκο. Το καινοτόμο στοιχείο του TRELLIS είναι ότι χρησιμοποιεί ως κριτήριο για τον κατακερματισμό του δένδρου επιθεμάτων **προθέματα μεταβλητού μήκους**. Ο TRELLIS έχει 4 βασικά βήματα:

1. Φάση Δημιουργίας Προθεμάτων: Το πρώτο βήμα δημιουργεί μια λίστα προθεμάτων μεταβλητού μήκους που περιορίζει το πρόβλημα της ανισοκατανομής των δεδομένων που εμφανίζεται κατά τον κατακερματισμό στην βάση προθεμάτων σταθερού μήκους. Επιλέγονται τα προθέματα που απαντούν στην ακολουθία εισόδου με συχνότητα μικρότερη από μια επιλεγμένη τιμή κατωφλίου.
2. Φάση Διαμερισμού: Στο δεύτερο βήμα η ακολουθία εισόδου διαμερίζεται, και ο αλγόριθμος παράγει υποδένδρα επιθεμάτων του κάθε διαμερισμού.
3. Φάση Συγχώνευσης: Σε αυτό το βήμα ο αλγόριθμος συγχωνεύει τα προθεματικά υποδένδρα που παρήχθησαν στην προηγούμενη φάση από όλους τους διαμερισμούς για κάποιο πρόθεμα, σε ένα μεγάλο προθεματικό υποδένδρο, ένα για κάθε ένα από τα προθέματα μεταβλητού μήκους που προέκυψαν από τη φάση 1
4. Φάση Ανάκτησης Συνδέσμων Επιθέματος: Το τελευταίο βήμα είναι προαιρετικό· κατασκευάζει το σύνολο των συνδέσμων επιθέματος αν χρειάζονται.

3.5.1 Φάση Δημιουργίας Προθεμάτων

Όπως είδαμε, η ιδέα του κατακερματισμού του δένδρου επιθεμάτων στην βάση προθεμάτων σταθερού μήκους πάσχει, αφού η ανομοιομορφία των πραγματικών βιολογικών δεδομένων οδηγεί σε ανισορροπία μεγέθους των κατασκευαζόμενων προθεματικών υποδένδρων. Για κάποια μήκη προθεμάτων μπορεί μόνο ένα ποσοστό των υποδένδρων να χωράει στην μνήμη, το οποίο επιβάλλει είτε κατασκευή των υποδένδρων που δεν χωράνε κατευθείαν στον δίσκο με χρονική επιβάρυνση λόγω του I/O φορτίου που περιλαμβάνει μια τέτοια κατασκευή, είτε αύξηση του μήκους των προθεμάτων. Η αύξηση του μήκους των προθεμάτων δημιουργεί όμως και αυτή προβλήματα, καθώς μπορεί να οδηγήσει σε πάρα πολλά υποδένδρα κάποια από τα οποία να είναι πολύ μικρότερα από το άνω όριο με συνέπεια την σπατάλη πόρων. Bin-packing τεχνικές που έχουν προταθεί έχουν το μειονέκτημα ότι πάσχουν από μεγάλο υπολογιστικό κόστος.

Αντί για την χρήση προθεμάτων σταθερού μήκους, ο αλγόριθμος TRELLIS χρησιμοποιεί προθέματα μεταβλητού μήκους. Η χρήση των προθεμάτων μεταβλητού μήκους βασίζεται στο γεγονός ότι δεν χρειάζονται να επεκταθούν όλα τα προθέματα σε μεγαλύτερο μήκος ώστε τα υποδένδρα που ορίζουν να χωρούν στην μνήμη.

Έστω $\Omega=\{P_0, P_1, P_2,\ldots, P_\omega\}$ το σύνολο των προθεμάτων μεταβλητού μήκους της ακολουθίας S. Έστω $freq(P_i)$ η απόλυτη συχνότητα ενός προθέματος P_i στην ακολουθία S. Το σύνολο P, πληθικότητας p, που θα χρησιμοποιηθεί για τον κατακερματισμό του δένδρου, ορίζεται ως το σύνολο που περιέχει όλα τα προθέματα P_i για το οποία $freq(P_i)\leq t$ και τα οποία είναι τα ελάχιστου μήκους προθέματα για τα οποία ισχύει η σχέση ανάμεσα στο σύνολο όλων των προθεμάτων που είναι η επέκταση τους. Στο σύνολο P δεν είναι δηλαδή επιτρεπτό να

περιέχεται ένα πρόθεμα P_0 το οποίο να έχει με την σειρά του ως γνήσιο πρόθεμα κάποιο άλλο μέλος του συνόλου P. Η σταθερά t που ονομάζεται κατώφλι, επιλέγεται ανάλογα με την διαθέσιμη μνήμη. Ο ορισμός αυτός και η χρήση των προθεμάτων μεταβλητού μήκους για την κατασκευή δένδρων επιθεμάτων οφείλεται στις στατιστικές ιδιότητες των συχνοτήτων διάφορων προθεμάτων μέσα σε γονιδιώματα. Στον πίνακα βλέπουμε για $t=10^6$, πόσα από τα προθέματα στο ανθρώπινο γονιδίωμα για κάθε μήκος έχουν συχνότητες μεγαλύτερες του κατωφλίου

Μήκος	#Προθέματα	Μήκος	#Προθέματα
1	4	5	781
2	16	6	930
3	64	7	68
4	253	8-15	2

Ήδη από μήκος 8 και πάνω μόνο 2 προθέματα έχουν συχνότητα πάνω από t ενώ για μήκος 16 δεν υπάρχει κανένα πρόθεμα το οποίο να έχει συχνότητα μεγαλύτερη από t.

Ο αλγόριθμος χρησιμοποιεί μια αποδοτική μέθοδο σάρωσης της ακολουθίας για να ανακτήσει τις συχνότητες των προθεμάτων μεταβλητού μήκους. Κατά την διάρκεια κάθε σάρωσης, επεξεργάζεται τα προθέματα μέχρι ένα συγκεκριμένο μήκος, έτσι ώστε οι δομές δεδομένων που χρησιμοποιούνται να χωρούν στην μνήμη. Ας υποτεθεί ότι ο μέγιστος αριθμός προθεμάτων που μπορούν να υπολογιστούν σε κάθε στάδιο φράσσεται επίσης από το κατώφλι t. Έστω L_i το μέγιστο μήκος προθέματος μετά την i-οστή σάρωση και έστω EP_i το σύνολο των προθεμάτων που χρειάζονται περαιτέρω επέκταση (δεν χωρούν) στην επόμενη σάρωση. Τότε τίθεται το L_i στην κατάλληλη τιμή ώστε να ικανοποιείται η ανισότητα $\left|EP_i\right| \sum_{j=1}^{L_i-L_{i-1}} 4^j \le t$ όπου $\left|EP_i\right|=1$ και $L_0=0$, και μπορούμε να υπολογίσουμε το αντίστοιχο πλήθος προθεμάτων μεταβλητού μήκους. Για αλφάβητο Σ διαφορετικό από το βιολογικό ο τύπος έχει την γενική μορφή $\left|EP_i\right| \sum_{j=1}^{L_i-L_{i-1}} \sigma^j$, όπου σ είναι το μέγεθος του αλφαβήτου $\left|\Sigma\right|$. Σε κάθε στάδιο εισάγονται στο P τα μικρότερου μήκους προθέματα που έχουν συχνότητα μικρότερη του κατωφλίου και απορρίπτονται όλες οι επεκτάσεις. Για το ανθρώπινο γονιδίωμα και για κατώφλι $t=10^6$, η φάση υπολογισμού των προθεμάτων χρειάζεται να κάνει μόνο δύο σαρώσεις με $L_1=8$ και $L_2=16$. Όπως αναφέρθηκε για μήκος μεγαλύτερο του 16 δεν υπάρχει κανένα πρόθεμα που να έχει συχνότητα μεγαλύτερη του κατωφλίου.Το σύνολο P για το ανθρώπινο γονιδίωμα με το προαναφερθέν κατώφλι, περιέχει προθέματα μήκους από 4 ως

16, και υπάρχουν συνολικά 6400 προθέματα μεταβλητού μήκους. Για μικρότερου μεγέθους κομμάτια του ανθρώπινου γονιδιώματος ο αριθμός των προθεμάτων είναι μικρότερος.

3.5.2 Φάση Διαμερισμού: Δημιουργία Προθεματικών Υποδένδρων

Ο TRELLIS διαιρεί την ακολουθία εισόδου S σε $r=\left\lceil \frac{n+1}{t} \right\rceil$ διαδοχικές υποακολουθίες ή διαμερισμοί. Έστω R_i η i-οστή διαμέριση και έστω T_R το δένδρο που περιέχει όλα τα επιθέματα της S που ξεκινούν στην R_i, τα οποία θα ονομαστούν *υποδένδρα επιθεμάτων*.

Κάθε διαμερισμός μπορεί να ανέβει στην μνήμη στο σύνολο της και να υποστεί επεξεργασία, αφού από σχεδιασμού κάθε υποδένδρο T_R μπορεί να έχει το πολύ t επιθέματα, και το κατώφλι t επιλέγεται έτσι ώστε ένα δένδρο επιθεμάτων με t φύλλα να μπορεί να χωρέσει στη μνήμη. Για την δημιουργία των υποδένδρων επιλέγεται ο αλγόριθμος του Ukkonen λόγω της γραμμικής του πολυπλοκότητας. Η υιοθέτηση του αλγόριθμου του Ukkonen είναι εφικτή για αυτήν την εφαρμογή αφού η διαμέριση είναι σε συνεχόμενα κομμάτια της ακολουθίας και επομένως είναι εφικτό κατά τις επεκτάσεις να ανακτώνται οι σύνδεσμοι επιθέματος που είναι απαραίτητοι. Όταν ολοκληρώνεται η κατασκευή του υποδένδρου επιθεμάτων για κάποιον διαμερισμό το δένδρο αυτό χωρίζεται περαιτέρω σε αρκετά υποδένδρα πριν αποθηκευτεί στον δίσκο. Ο χωρισμός γίνεται με βάση το σύνολο P, δηλαδή το T_{R_i} χωρίζεται σε μικρότερα υποδένδρα T_{R_i,P_j} που θα ονομαστούν *προθεματικά υποδένδρα διαμερισμών*, και περιέχουν μόνο τα επιθέματα της R_i που ξεκινούν από το πρόθεμα $P_j \in P$. Επομένως για κάθε διαμέριση R_i κατασκευάζονται το πολύ p αρχεία στον δίσκο.

Η άμεση εφαρμογή του αλγορίθμου του Ukkonen για κάθε έναν από τους διαμερισμούς R_i οδηγεί στην δημιουργία πεπλεγμένων (υπο)-δένδρων επιθεμάτων όπου κάποια από τα επιθέματα (αυτά που είναι προθέματα άλλων επιθεμάτων), θα έχουν μονοπάτια που τελειώνουν σε κάποια εσωτερική ακμή και όχι σε φύλλα. Μόνο η τελευταία διαμέριση R_{r-1} που περιέχει τον τερματικό χαρακτήρα θα είναι ένα γνήσιο δένδρο επιθεμάτων, αφού η μοναδικότητα του τερματικού χαρακτήρα εγγυάται ότι κανένα επίθεμα δεν είναι πρόθεμα άλλου επιθέματος, και άρα όλα τα επιθέματα θα έχουν μονοπάτια που καταλήγουν σε φύλλα. Πρέπει όμως να βρούμε κάποιον τρόπο ώστε κάθε επίθεμα να είναι με σαφήνεια αποθηκευμένο μέσα στα υποδένδρα επιθεμάτων σε κάποιο φύλλο. Πρέπει δηλαδή κάθε υποδένδρο επιθεμάτων που προκύπτει από τους διαμερισμούς να έχει ακριβώς $|R_i|$, ήτοι, t φύλλα.

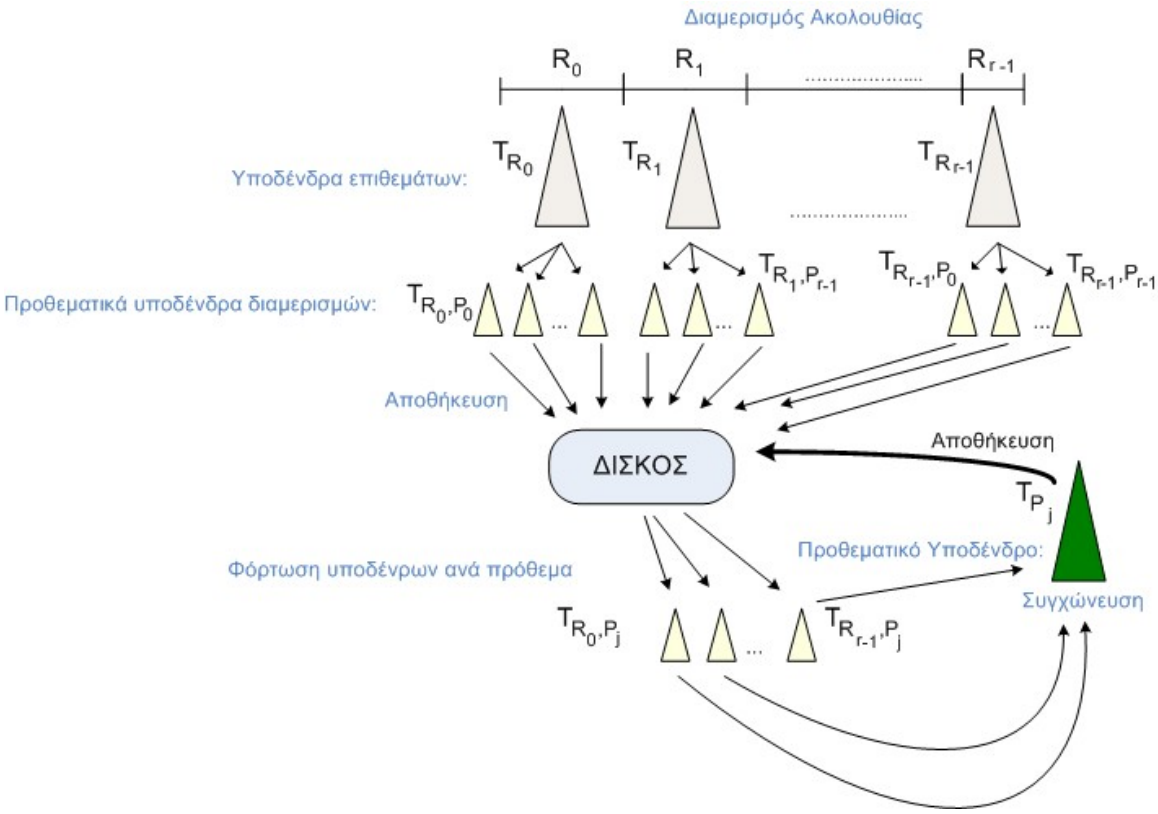
Ένας τρόπος να επιλύσουμε αυτό το πρόβλημα είναι να προστεθεί ο τερματικός χαρακτήρας στο τέλος κάθε υποακολουθίας R_i. Αυτό θα έχει ως συνέπεια όλα τα δένδρα T_{R_i} να έχουν ακριβώς $|R_i|$ φύλλα. Ωστόσο, αυτό θα έχει ως συνέπεια προβλήματα κατά την φάση της συγχώνευσης αφού ο τερματικός χαρακτήρας δεν βρίσκεται στην πραγματικότητα σε αυτή την θέση. Μια άλλη στρατηγική είναι να αφαιρέσουμε τον τερματικό χαρακτήρα μετά την δημιουργία από όλες τις ακμές κάτι που θα είχε όμως κόστος O(t) αφού θα έπρεπε να διασχίσουμε όλα τα φύλλα. Ο TRELLIS επιλύει το πρόβλημα αυτό συνεχίζοντας να διαβάζει κάποιους χαρακτήρες της υποακολουθίας R_i εως ότου στο δένδρο T_{R_i} να υπάρχουν ακριβώς t φύλλα. Αυτό θα συμβεί όταν συναντηθεί μια υποακολουθία που είναι μοναδική μέσα στον διαμερισμό, κατά τον ίδιο τρόπο που ο τερματικός χαρακτήρας απαντάται με μοναδικό τρόπο μέσα στην ακολουθία. Μόλις ολοκληρωθεί η τελευταία φάση του αλγορίθμου του Ukkonen για την ακολουθία που θα περιέχει το μοναδικό αυτό πρότυπο στο τέλος της, είναι σίγουρο ότι όλα τα επιθέματα του διαμερισμού αναπαριστώνται με σαφήνεια ως φύλλα στο υποδένδρο επιθεμάτων. Το βήμα αυτό δεν προσθέτει περαιτέρω φόρτο αφού κατά τεκμήριο χρειάζονται ελάχιστοι χαρακτήρες $c<<t$ για να καταστήσουν το πεπλεγμένο δένδρο γνήσιο. Για το ανθρώπινο γονιδιώμα με $t=10^6$ χρειάστηκαν να διαβαστούν το πολύ 1000 επιπλέον χαρακτήρες από κάθε διαμερισμό.

3.5.3 Συγχώνευση των Προθεματικών Υποδένρων Διαμερισμών

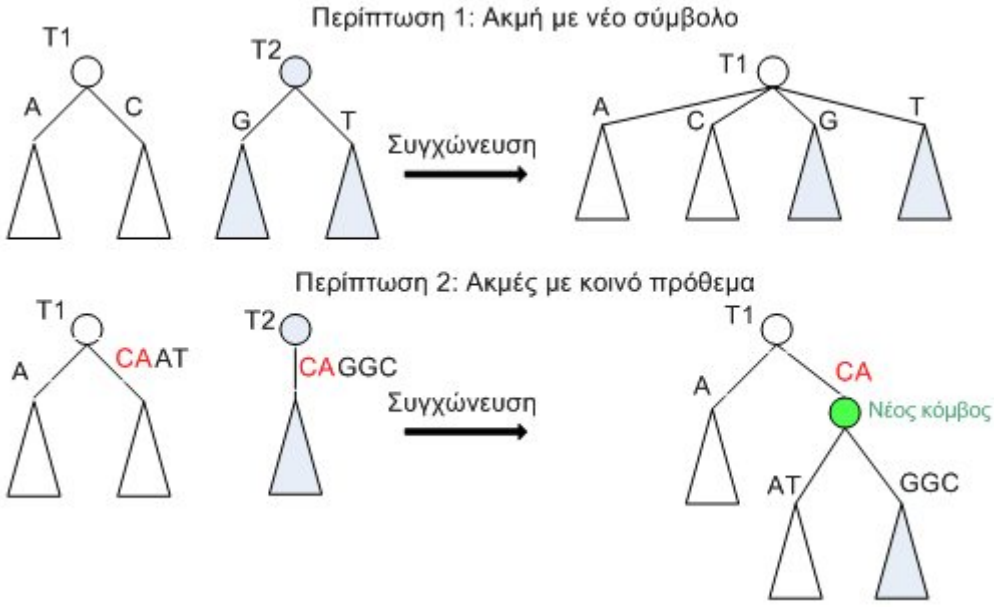
Μέχρι το σημείο αυτό ο αλγόριθμος έχει μεν εισάγει όλα τα επιθέματα της ακολουθίας σε κάποια δενδρική δομή, ωστόσο στον δίσκο υπάρχουν συνολικά $p \cdot r$ διαφορετικά αρχεία. Είναι απαραίτητο να συγχωνευτούν τα αρχεία που περιέχουν επιθέματα με το ίδιο πρόθεμα. Όλα τα προθεματικά υποδένδρα διαμερίσεων, που προέκυψαν από τις διαμερίσεις και έχουν το ίδιο πρόθεμα φορτώνονται στην μνήμη. Η μνήμη είναι βέβαιο ότι τα χωράει αφού η επιλογή των προθεμάτων μεταβλητού μήκους έγινε στην βάση της ανισότητας $freq(P_j) \leq t$, άρα τα δένδρα αυτά θα έχουν συνολικό άθροισμα φύλλων το πολύ t. Τα δένδρα σταδιακά συγχωνεύονται και αποθηκεύονται ως ένα αρχείο πίσω στον δίσκο. Κάθε δένδρο που προκύπτει από μια τέτοια συγχώνευση θα ονομαστεί *προθεματικό υποδένδρο*.

Πιο συγκεκριμένα, η συγχώνευση για ένα πρόθεμα P_j γίνεται σε στάδια. Σε κάθε στάδιο i, έστω M_i το συγχωνευμένο δένδρο που προκύπτει από την συγχώνευση των προθεματικών υποδένδρων διαμερίσεων T_{R_0,P_j} έως T_{R_i,P_j}. Στο επόμενο στάδιο φορτώνουμε στην μνήμη το T_{R_{i+1},P_j} το συγχωνεύουμε με το M_i και έχουμε το νέο δένδρο M_{i+1} και ούτω καθεξής για i από 0 ως r-1. Το τελικό συγχωνευμένο δένδρο M_{r-1} είναι το πλήρες προθεματικό υποδένδρο T_{P_j} το οποίο αποθηκεύεται πίσω στον δίσκο. Ο αλγόριθμος επαναλαμβάνει αυτή την

διαδικασία εως ότου να έχουν δημιουργηθεί προθεματικά υποδένδρα για όλα τα προθέματα μεταβλητού μήκους P_j , όπου $0 \leq j < p$.



Στο σχήμα φαίνονται τα βήματα του αλγορίθμου μετά τον υπολογισμό των προθεμάτων μεταβλητού μήκους μέχρι την δημιουργία των προθεματικών υποδένδρων.



Δεδομένων δύο δένδρων M_{r-1} και T_{R_{i+1},P_j} , ο αλγόριθμος για την συγχώνευση των δένδρων ξεκινάει μια αναδρομή από τις ρίζες των δύο δένδρων, και για οποιουσδήποτε κόμβους υ_1 και

υ_2 στα δύο δένδρα που έχουν το ίδιο μονοπάτι-ετικέτα, συγχωνεύει τις αντίστοιχες ακμές των παιδιών. Στο σχήμα φαίνεται με ένα παράδειγμα η διαδικασία της συγχώνευσης για τις δύο διακριτές περιπτώσεις που μπορούν να προκύψουν κατά την εκτέλεση του αλγορίθμου. Στην πρώτη περίπτωση υπάρχει μία ακμή κάτω από τον κόμβο υ_2 που ξεκινάει με ένα καινούριο σύμβολο, οπότε απλα θα αντιγραφεί στο τρέχοντα κόμβο υ_1 του συγχωνευόμενου δένδρου. Στην δέυτερη περίπτωση υπάρχουν οι ακμές e_1 και e_2 κάτω από τους κόμβους υ_1 και υ_2 που έχουν ένα κοινό πρόθεμα ως ετικέτα , έστω $e_1=\alpha\beta$ και $e_2=\alpha\gamma$ όπου $\alpha,\beta,\gamma \in \Sigma^*$, οπότε δημιουργείται ένας νέος εσωτερικός κόμβος με ετικέτα α, ο οποίος με τη σειρά του έχει δύο παιδιά, με ακμές που έχουν ετικέτες β και γ. Για παράδειγμα αν e_1 =CAAT και e_2 =CAGGC δημιουργείται νέος κόμβος με ετικέτα ακμής CA και παιδιά με ετικέτες ακμής AT και GGC.

Η συγχώνευση του δένδρου προϋποθέτει τυχαία προσπέλαση στην ακολουθία εισόδου S. Ο αλγοριθμός επεξεργάζεται κατά τεκμήριο ακολουθίες βιολογικών δεδομένων. Υπάρχει λοιπόν κωδικοποίηση/συμπίεση κάθε συμβόλου σε 2 bits στην μνήμη, άρα για το σύνολο του γονιδιώματος (≈3,3 Gbps) χρειάζονται περίπου 750 MB μνήμης για την αποθήκευση στη μνήμη, μέγεθος αρκετά εύλογο για έναν σύγχρονο υπολογιστή. Ως εκ τούτου δεν χρειάζεται κάποια διαχείριση buffer για την ακολουθία εισόδου, και θεωρείται ότι η πρόσβαση στην ακολουθία δεν επιφέρει περαιτέρω επιβάρυνση λόγω πρόσθετων I/O λειτουργιών.

Είναι επίσης ενδιαφέρον να σημειωθεί ότι η διαδικασία με την οποία γίνεται η συγχώνευση διαφέρει από αυτήν που πραγματοποιεί η παραλλαγή ST-Merge του αλγορίθμου TDD. Στον ST-Merge, τα δένδρα συγχωνεύονται μαζί για να δημιουργήσουν ένα πλήρες δένδρο επιθεμάτων για όλη την αρχική ακολουθία, ενώ ο TRELLIS συγχωνεύει μόνο τα υποδένδρα του ίδιου προθέματος δημιουργώντας στον δίσκο p διακριτά προθεματικά υποδένδρα. Κάθε πλήρες υποδένδρο της μνήμης γράφεται στον δίσκο με προτεραιότητα βάθους, σε ξεχωριστό αρχείο για τους εσωτερικούς κόμβους και σε άλλο αρχείο για τα φύλλα. Οι λεπτομέρειες της υλοποίησης του TRELLIS θα αναλυθούν στο κεφάλαιο 4.

3.5.4 Φάση ανάκτησης συνδέσμων επιθέματος

Η ύπαρξη των συνδέσμων επιθέματος είναι απαραίτητη για την εξασφάλιση της αποδοτικότητας σε πολλούς αλγορίθμους επεξεργασίας ακολουθιών, όπως η στοίχηση γονιδιώματος με δικλείδες ταύτισης (genome alignment via matching anchors) [BDP03, DPC+02 και HKO02] και η αναζήτηση δίδυμων επαναλήψεων (tandem repeats search) [GS04]. Ο TRELLIS διαθέτει μια μετα-κατασκευαστική φάση ανάκτησης των συνδέσμων επιθέματος, αν αυτοί ζητούνται. Η επιλογή αυτή επιτρέπει στους αλγόριθμους που χρησιμοποιούν συνδέσμους επιθέματος να εφαρμοστούν απευθείας πάνω στα δένδρα που εδράζονται στον δίσκο.

Παρόλο που ο TRELLIS χρησιμοποιεί τον αλγόριθμο του Ukkonen για την κατασκευή των δέδρων για να δημιουργήσει τα υποδένδρα επιθέματος, που στην συνέχεια χωρίζονται σε προθεματικά υποδένδρα διαμέρισης, μετά την φάση της συγχώνευσης δεν διαθέτουν όλοι οι εσωτερικοί κόμβοι σύνδεσμο επιθέματος. Η συγχώνευση μπορεί να καταργήσει κόμβους που διαθέτουν κάποιον σύνδεσμο επιθέματος και να δημιουργήσει κάποιον νέο εσωτερικό κόμβο που δεν διαθέτει. Στην πράξη, ένα μικρό ποσοστό από τους κόμβους στα προθεματικά υποδένδρα εξακολουθεί να διαθέτει συνδέσμους επιθέματος. Διαφαίνεται λοιπόν ως πιο αποδοτικό να κατασκευαστούν οι σύνδεσμοι επιθέματος από την αρχή παρά να γίνει εκμετάλλευση των υπαρχόντων συνδέσμων. Ο TRELLIS λοιπόν στην ουσία αγνοεί τους συνδέσμους που είναι κατασκευασμένοι μειώνοντας με αυτό τον τρόπο και της λειτουργίες I/O που θα γίνουν αφού μειώνονται οι πληροφορίες που πρέπει να κρατιούνται ανά κόμβο κατά την φάση της συγχώνευσης.

Η γενική ιδέα της ανάκτησης των συνδέσμων επιθέματος έχει ως εξής. Όπως αναφέρθηκε το δένδρο επιθεμάτων είναι στην μορφη πολλών προθεματικών υποδένδρων T_{P_j} για j $\in$ [0,m-1].

Ο TRELLIS ανακτά τα προθέματα για ένα προθεματικό δένδρο κάθε φορά. Προχωράει με μία διάσχιση προτεραιότητας βάθους· ξεκινώντας από τα παιδιά της ρίζας, για κάθε εσωτερικό κόμβο υ, ο αλγόριθμος εντοπίζει τον γονιό του p(υ) και τον σύνδεσμο επιθέματος sl(p(υ)) που έχει υπολογιστεί πριν. Ο σύνδεσμος επιθέματος στην συνήθη περίπτωση (αλλά όχι πάντα) δείχνει σε ένα άλλο προθεματικό υποδένδρο T_{P_α}, το οποίο φορτώνεται στην μνήμη την πρώτη φορά που προσπελαύνεται. Στην συνέχεια ο αλγόριθμος κάνει skip-and-count κάτω από τον sl(p(υ)) και ανακτά τον κόμβο sl(υ). Μόλις βρεθεί, η τοποθεσία στον δίσκο (φυσικός δείκτης στην θέση μέσα στο αρχείο) του κόμβου αυτού προστίθεται στο υ και ο αλγόριθμος συνεχίζει αναδρομικά για όλα τα παιδιά του υ που είναι εσωτερικοί κόμβοι. Με αυτό τον τρόπο διασφαλίζεται ότι όλοι επόμενοι σύνδεσμοι επιθέματος για τα παιδιά του υ θα βρεθούν κοντά στον sl(υ) μέσα στο T_{P_α}. Με την διάσχιση με προτεραίοτητα βάθους ανακτώνται λοιπόν όλοι οι συνδεσμοι επιθέματος για το T_{P_j} προσπελαύνοτας πρόσθετα δένδρα T_{P_α} όπως απαιτείται. Λόγω αυτής της διάσχισης, τα προθέματα P_α προσπελαύνονται με λεξικογραφική σειρά (αφού τα παιδιά κάθε κόμβου προσπελαύνονται στον πίνακα σε συγκεκριμένη σειρά πχ. A,C,G,T). Για παράδειγμα, αν το σύνολο των προθεμάτων μεταβλητού μήκους είναι το P={AC,CA,CC,CG,CTA,CTC,CTG,CTT}, οι σύνδεσμοι επιθέματος που προέρχονται από το T_{AC} μπορούν να καταλήγουν στο T_{CA}, T_{CC}.. T_{CTT}. Άρα οι σύνδεσμοι επιθέματος από το T_{AC} προς το T_{CA} θα κατασκευαστούν πρώτοι μετά οι σύνδεσμοι προς το T_{CC}, T_{CG} κοκ. Σε μια δεδομένη χρονική στιγμή, πρόσβαση γίνεται στους εσωτερικοί κόμβοι το πολύ δύο προθεματικών υποδένδρων. Οι κόμβοι που είναι φύλλα, που όπως

αναφέρθηκε αποθηκεύονται σε ξεχωριστό αρχείο, δεν διαθέτουν συνδέσμους επιθέματος και ως εκ τούτου δεν χρειάζεται να γίνει πρόσβαση σε αυτούς.

Ο TRELLIS λοιπόν αποφεύγει όσο το δυνατόν περισσότερο τις λειτουργίες I/O στον δίσκο για να ανακτήσει αποδοτικά τους συνδέσμους επιθέματος. Η φόρτωση του συνόλου των εσωτερικών κόμβων των δύο απαραίτητων προθεματικών υποδένδρων στην κύρια μνήμη πριν την ανάκτηση των συνδέσμων επιθέματος οδηγεί σε πολύ ταχύτερη λειτουργία σε σύγκριση με το διάβασμα από τον δίσκο. Για το παραπάνω παράδειγμα, κατά την ανάκτηση των συνδέσμων επιθέματος για το T_{CC}, το T_{CA} θα φορτωθεί στην μνήμη και μετά θα φύγει από την μνήμη και θα το διαδεχθεί το T_{CG} κοκ. Το δένδρο T_{CC} θα παραμένει μόνιμα στην μνήμη όσο κρατάει η κατασκευή των συνδέσμων του και μόλις η ανάκτηση ολοκληρωθεί, θα γραφεί στο δίσκο όπου κάθε εγγραφή για έναν κόμβο θα έχει μεγεθυνθεί για να έχει ως εγγραφή και τον σύνδεσμο επιθεμάτων. Με την κατάλληλη επιλογή του t για να χωράει πλήρως την ακολουθία εισόδου και δύο σύνολα εσωτερικών κόμβων (βλέπε 3.5.5), το βήμα της ανάκτησης των συνδέσμων επιθέματος μπορεί να γίνει με χρήση της διαθέσιμης μνήμης.

Κατά την φάση αυτή αναδεικνύεται το πλεονέκτημα της αποθήκευσης του πλήρους δένδρου επιθεμάτων του TRELLIS ως ένα δάσος από προθεματικά υποδένδρα, σε σχέση με την αποθήκευση που κάνει ο TDD, και η παραλλαγή του ST-Merge, όπου αποθηκεύεται το πλήρες δένδρο στον δίσκο χωρίς συνδέσμους επιθεμάτων. Μια οποιαδήποτε υλοποίηση για την ανάκτηση των συνδέσμων στο δένδρο που κατασκευάζουν αυτοί οι αλγόριθμοι θα ήταν πολύ χρονοβόρος, αφού, με δεδομένο ότι το πλήρες δένδρο των επιθεμάτων δεν χωράει στην μνήμη, κατά την προσπέλαση των κόμβων του δένδρου θα παρουσιαζόταν μεγάλη συμφόρηση λόγω των I/O λειτουργιών του δίσκου.

3.5.5 Επιλογή του κατωφλίου για την μνήμη

Σε όλα τα βήματα του αλγορίθμου κάναμε την σύμβαση ότι στην μνήμη μπορούν να χωρέσουν δένδρα με το πολύ t φύλλα,δηλαδή ο αριθμός των επιθεμάτων που υφίστανται επεξεργασία στην μνήμη σε μία δεδομένη στιγμή δεν πρέπει να ξεπερνάει το t. Η παράμετρος t χρησιμοποιήθηκε στα πρώτα 2 βήματα του αλγόριθμου για να βρεθεί το σύνολο P των προθεμάτων μεταβητού μήκους και για να καθοριστεί το μήκος των διαμερίσεων. Ένα εύλογο ερώτημα είναι ποια είναι η κατάλληλη επιλογή του κατωφλίου, δηλαδή ποια είναι η ελάχιστη τιμή που μπορεί να επιλεγεί ώστε η απαραίτητη μνήμη για την υλοποίηση του αλγορίθμου να μην ξεπεράσει σε μέγεθος την διαθέσιμη μνήμη σε όλα τα στάδια του αλγορίθμου, συμπεριλαμβανομένου του προαιρετικού σταδίου της ανάκτησης των συνδέσμων επιθέματος.

Έστω M η διαθέσιμη μνήμη. Για τυπικές ακολουθίες DNA ο αριθμός των εσωτερικών κόμβων ενός δένδρου επιθεμάτων δεν ξεπερνάει σε πλήθος το 70% των κόμβων-φύλλων, ενώ

τα φύλλα είναι ακριβώς n, όσο το μήκος της ακολουθίας. Η απαραίτητη μνήμη για την εφαρμογή του αλγορίθμου είναι αυτή για την αποθήκευση της ακολουθίας εισόδου και αυτή που χρειάζεται για την αναπαράσταση των υποδένδρων που είναι υπο επεξεργασία σε κάθε δεδομένη στιγμή. Η υλοποίηση του TRELLIS απαιτεί το πολύ 40 bytes άνα εσωτερικό κόμβο (36 bytes αν δεν υπάρχουν σύνδεσμοι επιθέματος) και 16 bytes άνα φύλλο. Τα παιδιά κάθε εσωτερικού κόμβου αποθηκεύονται σε έναν πίνακα σταθερού μεγέθους με σ+1 εγγραφές, δηλαδή 5 εγγραφές για ακολουθίες DNA. Μετά την φάση συγχώνευσης κάθε προθεματικό υποδένδρο διαθέτει εσωτερικούς κόμβους πλήθους της τάξεως 0,7t. Κατά την φάση ανάκτησης των συνδέσμων επιθέματος ο χρειάζονται φορτωμένα στην μνήμη οι εσωτερικοί κόμβοι από δύο δένδρα, άρα 0,7t + 0,7t = 1,4t. Θεωρώντας ότι κάθε σύμβολο της ακολουθίας αποθηκεύεται σε 2 bit, χρειαζόμαστε $\frac{n}{4}$ bytes για την αποθήκευση της ακολουθίας εισόδου.

Άρα, έχοντας ως δεδομένη την διαθέσιμη μνήμη M, το κατώφλι t επιλέγεται σύμφωνα με τα παραπάνω ως η μεγαλύτερη τιμή που ικανοποιεί την ανισότητα:

$$M \geq \frac{n}{4} + [(1{,}4 + 40) + 16]t \Rightarrow t \leq \frac{M - \frac{n}{4}}{72}$$

Αν το δεξί μέλος της ανισότητας είναι αρνητικό, τότε ο αλγόριθμος δεν μπορεί να εκτελεστεί θεωρώντας τις παραπάνω παραδοχές για τις δομές δεδομένων που βρίσκονται στην μνήμη, αφού η μνήμη δεν τις χωράει.

3.5.6 Ανάλυση υπολογιστικής πολυπλοκότητας

3.5.6.1 Πολυπλοκότητα της Φάσης δημιουργίας προθεμάτων

Όπως αναφέρθηκε η φάση της δημιουργίας των προθεμάτων μεταβλητού μήκους γίνεται σε στάδια σε κάθε ένα από τα οποία γίνεται σάρωση της ακολουθίας εισόδου. Σε κάθε στάδιο μόνο τα προθέματα μέχρι ένα συγκεκριμένο μήκος υπολογίζονται, και το μήκος αυτό αυξάνεται σε επόμενα στάδια. Επίσης, εξ'ορισμού, σε κάθε στάδιο μόνο τα προθέματα των οποίων οι συχνότητες υπερβαίνουν το κατώφλι επεκτείνονται. Τα υπόλοιπα προθέματα αποθηκεύονται στο σύνολο P. Στο τέλος του υπολογισμού έχουμε το σύνολο $P=\{P_0, P_1, P_2,\ldots, P_p\}$, p διαφορετικών προθεμάτων μεταβλητού μήκους, το καθένα από τα οποία εμφανίζεται στην ακολουθία με συχνότητα το πολύ t.

Έστω $\lambda=\max\{|P_i|\}$ το μεγαλύτερο μήκος κάποιου προθέματος που ανακτάται και εισάγεται στο σύνολο P. Αν το λ είναι γνωστό εξαρχής (από εκτίμηση ή από στατιστικά δεδομένα), και γίνεται μία μοναδική σάρωση στην ακολουθία εισόδου για να υπολογιστεί το σύνολο P, σε

κάθε θέση του S θα έπρεπε να ενημερωθούν οι συχνότητες λ προθεμάτων (των υποακολουθιών μήκους 1 εως λ που ξεκινούν από την δεδομένη θέση), που θα οδηγούσε σε πολυπλοκότητα χρόνου O(nλ) και πολυπλοκότητα χώρου O(n+$|\Sigma^{\lambda+1}|$). Στην πράξη, η προσέγγιση των πολλών σταδίων εκμεταλλεύεται την 'κλαδευτική' ιδιότητα του κατωφλίου συχνότητας t, και όσο αυξάνονται τα στάδια i και μεγαλώνει ο πίνακας P, τα προθέματα που ελέγχονται είναι πολύ λιγότερα από $|\Sigma^{i}|$ αφού τα προθέματα που είναι επεκτάσεις προθεμάτων που βρίσκονται ήδη στο P δεν χρειάζεται να ελεγχθούν κάτι που οδηγεί σε πολύ μειωμένες απαιτήσεις χώρου και χρόνου σε σχέση με το θεωρητικό όριο πολυπλοκότητας. Για το ανθρώπινο γονιδίωμα με κατώφλι $t=10^6$ μόνο δύο σαρώσεις αρκούν.

3.5.6..2 Πολυπλοκότητα της Φάσης Διαμερισμού

Κατά την φάση διαμερισμού η ακολουθία εισόδου σπάει σε $r=\left\lceil \frac{n+1}{i} \right\rceil$ διαμερισμούς (R_i, i ∈ [0,r-1]), και εφαρμόζεται ο αλγόριθμος του Ukkonen για να κατασκευαστεί ένα δένδρο επιθεμάτων T_{R_i} για κάθε διαμερισμό R_i. Αφού κάθε διαμερισμός είναι υποακολουθία μήκους (το πολύ) t, κάθε υποδένδρο επιθεμάτων T_{R_i} χρειάζεται O(t) χώρο και χρόνο για να κατασκευαστεί. Επιπλέον χρειάζεται μία πλήρης διάσχιση ώστε κάθε T_{P_i} να διαμεριστεί στα προθεματικά υποδένδρα διαμέρισης T_{R_i,P_j} (για j ∈[0,p-1]). Η πολυπλοκότητα κατά μήκος όλων των διαμερισμών r είναι r·O(t)=O(n). Μόνο μία σάρωση της ακολουθίας S είναι απαραίτητη σε αυτό το στάδιο. Όπως αναφέρθηκε στην 3.5.2 ο αλγόριθμος σαρώνει c χαρακτήρες μετά το τέλος κάθε διαμερισμού για να μετατραπούν τα πεπλεγμένα δένδρα σε γνήσια δένδρα. Για τυπικές βιολογικές ακολουθίες το πλήθος των επιπλέον χαρακτήρων που διαβάζεται είναι πολύ μικρό, ωστόσο στην ακραία περίπτωση μίας ακολουθίας ενός μόνο συμβόλου αυτή η προσέγγιση μπορεί να δημιουργήσει πρόβλημα αφού η σάρωση θα συνεχιστεί χωρίς να συναντήσει κάποια μοναδική υποακολουθία. Για να αποφευχθεί η δυσλειτουργία σε ακραίες περιπτώσεις ακολουθιών θα πρέπει να υιοθετηθεί κάποια άλλη στρατηγική για να διασφαλιστεί ότι τα υποδένδρα των διαμερισμών δεν είναι πεπλεγμένα, όπως η εισαγωγή ενός τερματικού χαρακτήρα στο τέλος κάθε διαμερισμού. Θεωρείται όμως ότι ο αλγοριθμος θα επεξεργάζεται πάντα πραγματικά βιολογικά δεδομένα τα οποία δεν παρουσιάζουν τέτοιες συμπεριφορές .

Από άποψη λειτουργιών I/O στον δίσκο, κατά την φάση διαμερισμού, υπάρχουν r·p υποδένδρα T_{R_i,P_j} το καθένα μεγέθους $O\left(\frac{t}{p}\right)$ που γράφονται στον δίσκο. Αφού

$r \cdot p \cdot \frac{t}{p} = n$, όλα τα επιθέματα του πλήρους δένδρου γράφονται στο δίσκο μια φορά. Αυτό γίνεται σε O(rp) προσπελάσεις του δίσκου, αφού ο TRELLIS διαβάζει ή γράφει τα δεδομένα στο δίσκο σε ένα βήμα. Ο αλγόριθμος δηλαδή δεν έχει κάποια περίπλοκη στρατηγική buffering για την διαχείριση του τρόπου που αποθηκεύονται και διαβάζονται οι κόμβοι όπως οι περισσότεροι υπάρχοντες αλγόριθμοι, απλά γράφει και διαβάζει τα απαραίτητα υποδένδρα ολόκληρα προς και από τον δίσκο και στην συνέχεια τα επεξεργάζεται στην μνήμη. Αυτό είναι εφικτό γιατί το μέγεθος των υποδένδρων που παραμένουν στην μνήμη σε οποιαδήποτε στιγμή φράσσεται από το κατώφλι, το οποίο έχει υπολογιστεί σύμφωνα με το μέγεθος της διαθέσιμης μνήμης. Οι κόμβοι των δένδρων γράφονται στον δίσκο (και διαβάζονται πάλι από αυτόν) με προτεραιότητα βάθους. Η πολιτική προσπέλασης του δίσκου που εφαρμόζει ο TRELLIS (με εγγραφές/αναγνώσεις ολόκληρων υποδένδρων κάθε φορά) είναι διαφορετικοί από αυτή των αλγορίθμων όπου κάθε προσπέλαση στον δίσκο αφορά ένα μικρό αριθμό κόμβων, συνήθως ανάλογο με το μέγεθος της σελίδας.

3.5.6.3 Πολυπλοκότητα της Φάσης Συγχώνευσης

Στην φάση της συγχώνευσης τα προθεματικά υποδένδρα T_{R_i,P_j} από όλες τις διαμερίσεις συγχωνεύονται στο προθεματικό υποδένδρο T_{P_j} για κάθε πρόθεμα μεταβλητού μήκους P_j,

Για κάθε πράξη συγχώνευσης πρέπει να διασχιστούν σ=4 ακμές για κάθε κόμβο και πρέπει να συγκριθούν τόσοι χαρακτήρες όσους διαθέτει το μέγιστο κοινό πρόθεμα (LCP) των ακμών που συγχωνεύονται. Έστω μ το μέσο μήκος LCP για όλες τις πράξεις συγχώνευσης. Ο μέσος χρόνος λοιπόν για κάθε συγχώνευση θα είναι 4μ=O(μ). Για όλα τα προθεματικά υποδένδρα ο συνολικός χρόνος συγχώνευσης είναι O(μn) αφού το πλήθος των κόμβων και των ακμών στο πλήρες δένδρο επιθεμάτων φράσσεται από O(n). Αφού μ=O(n) στην χείριστη περίπτωση, η συνολική χρονική πολυπλοκότητα της συγχώνευσης είναι $O(n^2)$. Ωστόσο, μ=O(n) είναι μια πολύ απαισιόδοξη εκτίμηση για όριο του μέσο μήκους του LCP. Από στατιστικά δεδομένα προκύπτει ότι το μέσο μήκος LCP για μεγάλες ακολουθίες DNA είναι περίπου 30, υποδηλώνοντας ότι ότι κατά μέσο όρο p=logn. Συνεπώς, στην πράξη η πολυπλοκότητα της συγχώνευσης είναι O(nlogn) και στην χείριστη περίπτωση $O(n^2)$

Η συνολική χωρική πολυπλοκότητα της φάσης συγχώνευσης παραμένει O(n), αφού υπάρχουν O(n) κόμβοι στο πλήρες δένδρο επιθεμάτων και ο χώρος που απαιτείται για την ακολουθία εισόδου είναι επίσης O(n). Η ακολουθία εισόδου θα σαρωθεί αρκετές φορές αφού σε κάθε πράξη συγχώνευσης κόμβων διαβάζονται κατά μέσο όρο μ χαρακτήρες από την S.

3.5.6..4 Πολυπλοκότητα της Φάσης Ανάκτησης Συνδέσμων Επιθέματος

Στην φάση αυτή ανακτώνται οι σύνδεσμοι επιθεμάτων για όλους τους O(n) εσωτερικούς κόμβους των προθεματικών υποδένδρων. Σε κάθε εσωτερικό κόμβο, γίνεται διάσχιση μια ακμή προς τα πάνω για να βρεθεί ο κόμβος γονέας που διαθέτει σύνδεσμο επιθέματος. Στην συνέχεια ακολουθείται ο σύνδεσμος, γίνεται skip/count προς τα κάτω για έναν αριθμό κόμβων και προστίθεται ο σύνδεσμος επιθέματος για τον δεδομένο κόμβο. Η ανάβαση κατά έναν κόμβο και η εγγραφή του συνδέσμου επιθέματος είναι λειτουργίες που χρειάζονται σταθερό χρόνο. Έχει επίσης αποδειχθεί (βλέπε 2.3.2.4) ότι όλα τα skip/count βήματα κατά την δημιουργία του δένδρου είναι το πολύ 3n=O(n). Άρα, η συνολική χρονική πολυπλοκότητα για την ανάκτηση των συνδέσμων επιθέματος είναι O(n). Η χρονική πολυπλοκότητα παραμένει επίσης O(n), αφού κάθε στιγμή ο αλγόριθμος προσπελαύνει την ακολουθία εισόδου και το πολύ δύο προθεματικά υποδένδρα.

Όσον αφορά το κόστος των λειτουργιών I/O στον δίσκο, η ανάκτηση των συνδέσμων απαιτεί να παραμένουν όλουι οι εσωτερικοί κόμβοι στην μνήμη. Έστω ότι το δένδρο για το οποίο ανακτούμε τους κόμβους είναι το T_{P_j}. Κατά την διάσχιση με προτεραιότητα βάθους πρέπει να κρατείται στην μνήμη το σύνολο των εσωτερικών κόμβων ενός άλλου δένδρου, έστω T_{P_α}. Μόλις υπάρξει ανάγκη να διαβαστεί κάποιος κόμβος από το T_{P_α} τότε αυτό φορτώνεται όλο στον δίσκο σε ένα βήμα, αντικαθιστώντας ένα προηγούμενο δένδρο που μπορεί να είχε διαβαστεί. Παρόλο που όλοι οι εσωτερικοί κόμβοι ανεβαίνουν στην μνήμη, μόνο οι κόμβοι στους οποίους καταλήγει κάποιος σύνδεσμος επιθέματος από το T_{P_j} θα χρησιμοποιηθούν. Επίσης, μόλις ολοκληρωθεί η ανάκτηση των συνδέσμων επιθέματος, το δένδρο γράφεται στον δίσκο ολόκληρο σε ένα βήμα.Το συνολικό I/O κόστος είναι δύσκολο να χαρακτηριστεί, αλλά στην χειρότερη περίπτωση κάθε προθεματικό υποδένδρο μπορεί να διαβαστεί p φορές, μια φορά για καθένα από τα p προθέματα μεταβλητού μήκους. Αυτό θα υποννούσε O(p) αναγνώσεις από το δένδρο του δίσκου στην χειρότερη περίπτωση. Στην πράξη πάντως κάθε προθεματικό υποδένδρο T_{P_j} έχει συνδέσμους σε λίγα μόνο άλλα δένδρα T_{P_α} απαιτώντας πολύ λίγα φορτώματα δένδρων άνα πρόθεμα. Η εγγραφή όλων των προθεματικών δένδρων στον δίσκο μετά την ανάκτηση των συνδέσμων επιθέματος ισοδυναμεί με την ανάκτηση των συνδέσμων για το πλήρες δένδρο, αφού αυτό είναι πλέον κατακερματισμένο ως δάσος προθεματικών υποδένδρων.

Εξετάζοντας τα παραπάνω συμπεράσματα για την χωροχρονική πολυπλοκότητα του αλγορίθμου στις διάφορες φάσεις, συμπεραίνεται ότι η φάση της συγχώνευσης έχει δυνητικά το μεγαλύτερο υπολογιστικό κόστος λόγω της $O(n^2)$ πολυπλοκότητας για την χείριστη περίπτωση, που δίνει στον TRELLIS επίσης πολυπλοκότητα $O(n^2)$ για την χείριστη

περίπτωση. Στην μέση περίπτωση, η πολυπλοκότητα της συγχώνευσης, και άρα του TRELLIS, είναι O(nlogn). Η χωρική πολυπλοκότητα παραμένει O(n).

4 *Υλοποίηση αλγορίθμων αποτίμησης ερωτημάτων στα δένδρα επιθεμάτων*

4.1 Γενική εικόνα του συστήματος και παρεμβάσεις στα αρχεία των δένδρων

4.1.1 Το σύστημα αρχείων του TRELLIS

Με τα προγράμματα ανοιχτού κώδικα που διατίθενται από την δημιουργό του TRELLIS, μπορεί να κατασκευαστεί το δένδρο επιθεμάτων για οποιαδήποτε ακολουθία DNA. Λεπτομέρειες για την εκτέλεση των προγραμμάτων του TRELLIS από την γραμμή εντολών υπάρχουν στο παράρτημα. Η ακολουθία εισόδου πρέπει να έχει ως αλφάβητο το σύνολο Σ={1,2,3,4}, όπου τα αριθμητικά 1, 2, 3 και 4 αντιστοιχούν στις βάσεις A, C, G και T αντίστοιχα. Μετά την ολοκλήρωση των 3 πρώτων φάσεων της κατασκευής (υπολογισμος προθεμάτων μεταβλητού μήκους, διαμερισμός, συγχώνευση) οι πληροφορίες που συνιστούν το δέντρο βρίσκονται αποθηκευμένες σε τρία διαφορετικά σημεία στο δίσκο:

- στο αρχείο **prefix.txt** που βρίσκεται στον ίδιο φάκελο με το αρχείο εισόδου,
- στο φάκελο **tempBinData** που περιέχει τα αρχεία εσωτερικών κόμβων, και
- στον φάκελο **binData** που περιέχει τα αρχεία φύλλων.

Το αρχείο prefix.txt, περιέχει όλα τα προθέματα και τις αντίστοιχες συχνότητες με τις οποίες αυτά παρουσιάζονται στην ακολουθία εισόδου. Η πρώτη γραμμή περιέχει ένα πρόθεμα και η δεύτερη την συχνότητα του, η τρίτη γραμμή περιέχει το δεύτερο πρόθεμα και η τέταρτη την συχνότητα του κοκ. Το αρχείο δηλαδή είναι της παρακάτω μορφής:

Γραμμή 1: 11

Γραμμή 2: 38401

Γραμμή 3: 12

Γραμμή 4: 41320

.....

Γραμμή 2j+1:1122

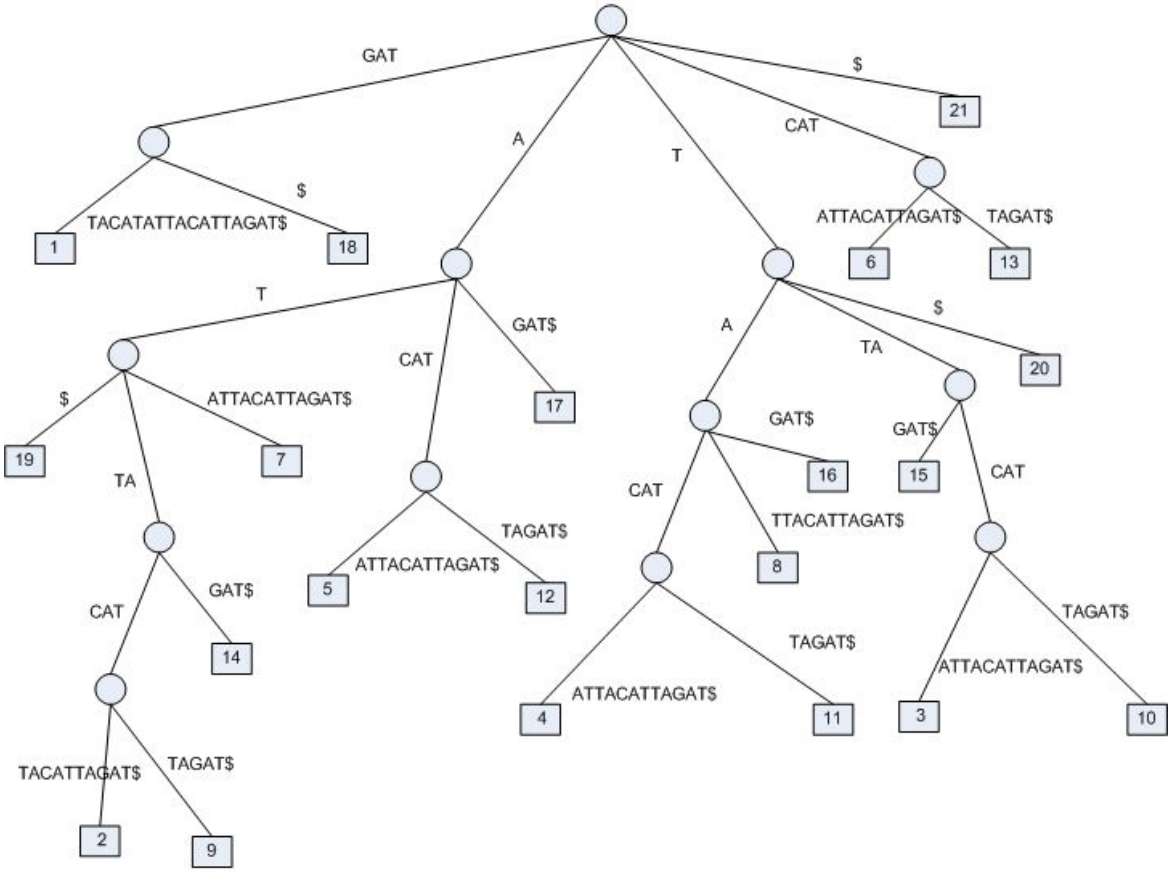
Γραμμή 2(j+1):56787

.....

Γραμμή 2(p-1)+1: 4444

Γραμμή 2p: 45456

Ο συνολικός αριθμός των προθεμάτων μεταβλητού μήκους, δηλαδή η πληθικότητα του συνόλου P, είναι p. Άρα το πρόθεμα 11 (που αντιστοιχεί στην ακολουθία "AA") απαντάται 38401 φορές στην ακολουθία εισόδου, το πρόθεμα 12 ("AC") απαντάται 41320 φορές, και ούτω καθεξής μέχρι να παρατεθούν όλα τα προθέματα. Αυτή είναι η συνήθης εικόνα ενός αρχείου προθεμάτων που αφορά ακολουθίες μεγέθους κάποιων Mbps, όπου οι συχνότητες είναι της τάξης κάποιων δεκάδων χιλιάδων. Τα συνήθη μεγέθη ακολουθιών που επεξεργάζεται η εφαρμογή είναι βέβαια τέτοιου μεγέθους και πολύ μεγαλύτερα, και δεν προσφέρονται να χρησιμοποιηθούν ως παράδειγμα για να αναλυθεί διεξοδικά η διαδικασία. Θα θεωρήσουμε λοιπόν μια πολύ μικρή ακολουθία μήκους 20 συμβόλων, η οποία θα χρησιμοποιηθεί για να είναι δυνατή μία εποπτεία των βημάτων του αλγορίθμου. Έστω λοιπόν η ακολουθία S=’GATTACATATTACATTAGAT$’. Το πλήρες δένδρο επιθεμάτων της S, το οποίο θα μπορούσε να δημιουργηθεί ως έχει αν χωράει ολόκληρο στην μνήμη, φαίνεται στο σχήμα.



Υποθέτουμε λοιπόν ότι διαθέτουμε έναν υπολογιστή με μνήμη που δεν χωράει το δένδρο. Το κατώφλι της μνήμης, δηλαδή ο μέγιστος αριθμός κόμβων-φύλλων που μπορεί να βρίσκεται φορτωμένος στην μνήμη σε κάθε στιγμή, ορίζεται t=3. Η πρώτη λειτουργία που πρέπει να γίνει είναι η εύρεση των συχνοτήτων των διαφόρων προθεμάτων. Δεκτές γίνονται μόνο οι συχνότητες που δεν ξεπερνούν το κατώφλι, αυτές δηλαδή για τις οποίες θα κατασκευαστεί εν συνεχεία κάποιο προθεματικό υποδένδρο, ενώ για προθέματα που έχουν συχνότητα μεγαλύτερη από το κατώφλι η σάρωση συνεχίζεται για επεκτάσεις μεγαλύτερου μήκους.

Για την παραπάνω ακολουθία έχουμε λοιπόν τον υπολογισμό των προθεμάτων μεταβλητού μήκους όπως φαίνεται στον παρακάτω πίνακα:

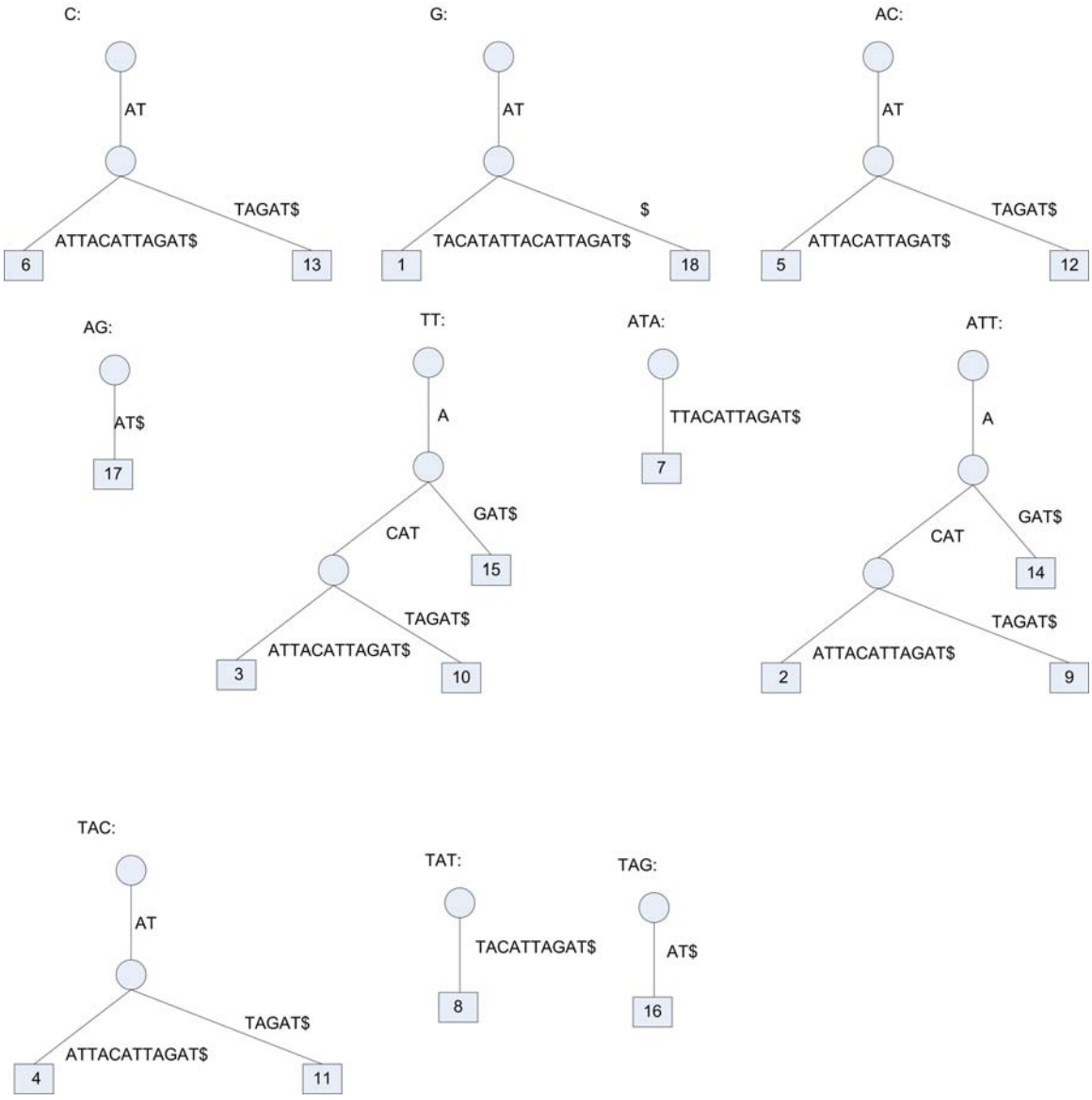
Σάρωση 1	Σάρωση 2(επέκταση A,T)	Σάρωση 3 (επέκταση των AT, TA)
A : 8 απορρίπτεται	AA : -	ATA: 1 δεκτό
C : 2 δεκτό	AC : 2 δεκτό	ATC: -
G : 2 δεκτό	AG : 1 δεκτό	ATG: -
T : 8 απορρίπτεται	AT : 5 απορρίπτεται	ATT: 3 δεκτό
	TA: 4 απορρίπτεται	TAA: -
	TC: -	TAC: 2 δεκτό

	TG: -	TAG: 1 δεκτό
	TT: 3 δεκτό	TAT: 1 δεκτό

Άρα το σύνολο των προθεμάτων μεταβλητού μήκους στην βάση των οποίων θα κατασκευαστούν τα προθεματικά υποδένδρα είναι P={C, G, AC, AG, TT, ATA, ATT, TAC, TAG, TAT}.

Στην συνέχεια η ακολουθία θα διαμεριστεί σε $\left\lceil \frac{21}{3} \right\rceil$=7 διαμερισμούς και για κάθε έναν θα κατασκευαστεί το αντίστοιχο υποδένδρο επιθεμάτων. Τα υποδένδρα επιθεμάτων θα διαμεριστούν περαιτέρω με βάση τα προθέματα του συνόλου P, και για κάθε υποδένδρο θα αποθηκευτούν στον δίσκο τα αντίστοιχα προθεματικά υποδένδρα διαμερισμών. Μετά την αποθήκευση των προθεματικών υποδένδρων διαμερισμών στον δίσκο, θα φορτωθούν στην μνήμη αυτά που ανήκουν στο ίδιο πρόθεμα και θα συγχωνευτούν σε ένα προθεματικό υποδένδρο. Η ίδια διαδικασία θα επαναληφθεί για όλα τα προθέματα που ανήκουν στο P.

Στο παρακάτω σχήμα φαίνονται τα προθεματικά υποδένδρα που κατασκευάζονται από τον αλγόριθμο για το παραπάνω παράδειγμα.



Παρατηρούμε ότι τα επιθέματα 19,20 και 21 **δεν αποθηκεύονται**. Αυτό συμβαίνει επειδή τα προθέματα AT και T ξεπερνούν σε συχνότητα την συχνότητα κατωφλίου, άρα δεν μπορεί να δημιουργηθεί ένα προθεματικό υποδένδρο που να αποθηκεύσει τα επιθέματα αυτά αφού το αλφάβητο των προθεμάτων δεν περιλαμβάνει τον τερματικό χαρακτήρα $. Το πρόβλημα αυτό όμως, υπό κανονικές συνθήκες και για το είδος των δεδομένων που πραγματεύεται η εφαρμογή TRELLIS δεν δημιουργεί επιπλοκές. Στις συνήθεις εφαρμογές, οι αναζητήσεις προτύπων που γίνονται σε δένδρα επιθεμάτων βιολογικών δεδομένων αφορούν πρότυπα που έχουν ελάχιστο μήκος της τάξης κάποιων δεκάδων συμβόλων. Δημοφιλή προγράμματα στοίχησης γονιδιώματος όπως το MUMer έχουν ελάχιστο μήκος αναζήτησης 40 σύμβολα. Γίνεται φανερό δηλαδή ότι στην πράξη δεν γίνονται ερωτήσεις με μέγεθος μικρότερο του 40. Από την άλλη, όπως αναφέρθηκε στο 3.5.1, για μέγεθος κατωφλίου $t=10^6$ που είναι σχετικά μικρό (για πολύ μικρά κλάσματα του γονιδιώματος και μικρό μέγεθος μνήμης, άρα μεγάλο αριθμό partitions, το κατώφλι είναι μεγαλύτερο του 10^6) δεν υπάρχουν προθέματα με μήκος

μεγαλύτερο των 15 χαρακτήρων η συχνότητα των οποίων υπερβαίνει το κατώφλι. Για μεγαλύτερη τιμή κατωφλίου το άνω αυτό όριο θα ήταν ακόμα πιο μικρό. Άρα τα επιθέματα που δεν θα αποθηκευθούν είναι στην πιο ακραία περίπτωση 15 χαρακτήρες πριν το τέλος της γενετικής ακολουθίας, τα επιθέματα όμως αυτά ή κλάσματα αυτών των επιθεμάτων δεν θα αναζητηθούν ποτέ αφου όπως είπαμε τα queries είναι μεγέθους τουλάχιστον 40 χαρακτήρων. Σίγουρα βέβαια, υπάρχουν πολλοί τρόποι να ξεπεραστεί το πρόβλημα αυτό αν κάποιος θέλει να αποθηκεύει διεξοδικά όλα τα επιθέματα, για την περίπτωση που θα ήθελε να κάνει μικρότερα queries σε κάποια ειδική εφαρμογή, ο κώδικας του TRELLIS πάντως δεν έχει κάποια πρόβλεψη.

Όπως έχει ήδη αναφερθεί, ο αλγόριθμος αποθηκεύει τους εσωτερικούς κόμβους και τα φύλλα σε διαφορετικές θέσεις εκμεταλλευόμενος το γεγονός ότι χρειάζονται λιγότερες πληροφορίες για την αναπαράσταση των φύλλων. Ο φάκελος tempBinData περιέχει τα αρχεία των εσωτερικών κόμβων. Ο αριθμός των αρχείων είναι ίσος με τον αριθμό των προθεμάτων που περιέχονται στο prefix.txt, και καθένα από αυτά περιέχει τους εσωτερικούς κόμβους του υποδέντρου που αντιστοιχεί στο κάθε πρόθεμα. Η ονοματοδοσία των αρχείων γίνεται με τον αριθμό που αντιστοιχεί στο κάθε πρόθεμα με τη σειρά που παρουσιάζονται στο αρχείο prefix.txt, άρα το αρχείο tempBinData/0 αναφέρεται στο υποδένδρο που περιέχει τους εσωτερικούς κόμβους με πρόθεμα το πρώτο πρόθεμα του prefix.txt, το αρχείο tempBinData/1 είναι το υποδένδρο των εσωτερικών κόμβων με το δεύτερο πρόθεμα κοκ. Η σειρά των προθεμάτων μέσα στο αρχείο prefix.txt είναι λεξικογραφική στο εύρος ενός συγκεκριμένου μήκους προθέματος. Δηλαδή τα προθέματα ενός συγκεκριμένου μήκους είναι τοποθετημένα σε λεξικογραφική σειρά, και ακολουθούν τα προθέματα μεγαλύτερου μήκους πάλι σε λεξικογραφική σειρά κοκ.

Για το παράδειγμα, τα πρόθεματα C, G, AC, AG, TT, ATA, ATT, TAC, TAG και TAT αποθηκεύονται στα αρχεία tempBinData/0, tempBinData/1, tempBinData/2, tempBinData/3, tempBinData/4, tempBinData/5, tempBinData/6, tempBinData/7, tempBinData/8 και tempBinData/9 αντίστοιχα.

Το κάθε αρχείο εσωτερικών κόμβων είναι ένα αρχείο τυχαίας προσπέλασης, η κάθε εγγραφή του οποίου έχει μέγεθος 28 bytes, και την ακόλουθη δομή:

[*δείκτης αρχής*, *δείκτης τέλους,, παιδί_0, παιδί_1, παιδί_2, παιδί_3, παιδί_4*]

Το καθένα από τα πεδία της εγγραφής είναι ένας μη προσεσημασμένος ακέραιος 4 bytes (UI4Bytes, εξ'ου και το μέγεθος της κάθε εγγραφής 7 X 4 =28 Bytes)

Το πεδίο <*δείκτης αρχής*> αναφέρεται στη θέση μέσα στην ακολουθία εισόδου όπου αρχίζει μια εμφάνιση της υποακολουθίας που κωδικοποιείται από την προγονική ακμή που προσπίπτει στον τρέχοντα κόμβο. Για παράδειγμα, το μοναδικό παιδί της ρίζας στο υποδένδρο του προθέματος C στο παραπάνω σχήμα, η προγονική ακμή του οποίου

κωδικοποιεί την ακολουθία AT θα μπορούσε να έχει <δείκτη αρχής> είτε το 2 είτε το 7 είτε το 9 κοκ, αφού όλα αυτά αναφέρονται σε θέσεις μέσα στην ακολουθία εισόδου όπου ξεκινάει η ακολουθία AT, δεν έχει δηλαδή σημασία ποια ακριβώς θέση χρησιμοποιείται από όλες τις πιθανές θέσεις που υπάρχει η συγκεκριμένη υποακολουθία μέσα στην ακολουθία εισόδου, σημασία έχει να είναι μια σωστή θέση, ώστε να μπορεί να γίνεται η σύγκριση όταν αναζητούμε υποακολουθίες.

Το πεδίο <*δείκτης τέλους*> αναφέρεται στη θέση μέσα στην ακολουθία εισόδου όπου αρχίζει μια εμφάνιση της υποακολουθίας που κωδικοποιείται από την προγονική ακμή που προσπίπτει στον τρέχοντα κόμβο. Χρησιμοποιώντας το ίδιο παράδειγμα με τον <*δείκτη αρχής*> ο <*δείκτης τέλους*> μπορεί να είναι είτε το 5, είτε το 8, είτε το 10 κοκ έχοντας βέβαια τον αντίστοιχο δείκτη αρχής. Δηλαδή τα εύρη μέσα στην ακολουθία εισόδου που μπορούν να κωδικοποιηθούν για την ακμή AT, είναι τα [2,5], [7,8], [9,10] κοκ.

Με αυτόν τον τρόπο δεν αποθηκεύεται η καθαυτό πληροφορία της ακολουθίας εισόδου (με εξαίρεση τους αρχικούς χαρακτήρες κάθε υποακολουθίας) . Πρόκειται για την κωδικοποίηση των ακμών που χρησιμοποιήθηκε και από τον αλγόριθμο του Ukkonen και επιτρέπει το δένδρο επιθεμάτων να αποθηκευτεί σε χώρο O(n), όπου n το μήκος της ακολουθίας. Για μια οποιαδήποτε ακμή μέσα στα δένδρα που κωδικοποιεί κάποια υποακολουθία , μπορούμε να καθορίσουμε τον αρχικό χαρακτήρα χωρίς να καταφύγουμε στην αρχική ακολουθία εισόδου, από το αν η ακμή προέρχεται από το δείκτη <παιδί 1>,<παιδί 2> κοκ αφού κάθε τέτοιος δείκτης δείχνει κάπου ανάλογα με αν η υποακολουθία-ετικέτα της εξερχόμενης ακμής ξεκινάει με A,C κοκ. Άρα η πληροφορία για τον πρώτο χαρακτήρα κάθε υποακολουθίας που κωδικοποιούν οι ακμές του δένδρου είναι αποθηκευμένη μέσα στο δένδρο, ενώ οι υπόλοιποι χαρακτήρες της ακολουθίας που βρίσκονται στις ακμές προκύπτουν με προσπέλαση στην ακολουθία για το κατάλληλο εύρος θέσεων.

Εξαίρεση αποτελεί ο root κόμβος που κατά σύμβαση έχει <*δείκτη αρχής*> το 0 και <*δείκτη τέλους*> το MAX_INT. Θεωρείται ότι γνωρίζουμε από πριν το πρόθεμα που αφορά το κάθε υποδένδρο, αφού έχουμε επιλέξει να ανοίξουμε το συγκεκριμένο αρχείο εσωτερικών κόμβων.

Όσον αφορά τα πεδία <*παιδί_**> αυτά περιέχουν ένα δείκτη που υποδηλώνει τη θέση της εγγραφής των κόμβων-παιδιών του τρέχοντα κόμβου. Τα <*παιδί_0*>, <*παιδί_1*>, <*παιδί_2*>, <*παιδί_3*>, <*παιδί_4*> περιέχουν τους δείκτες σε κόμβους-παιδιά των οποίων η προσπίπτουσα ακμή ξεκινάει με $, A, C, G, και T αντίστοιχα.

- Αν κάποιο από τα παιδιά δεν υπάρχει τότε ο αντίστοιχος δείκτης περιέχει την τιμή μηδέν. Για παράδειγμα, στο προθεματικό υποδένδρο TT του σχήματός μας, ο εσωτερικός κόμβος με προγονική ακμή την A έχει δύο παιδιά, το ένα με ακμή που ξεκινάει με C και το άλλο με ακμή που ξεκινάει με G. Άρα οι δείκτες <παιδί0>,

<παιδί1>, <παιδί4> είναι μηδενικοί αφού δεν υπάρχει ακμή με αρχικό χαρακτήρα το $, το A και το T.

- Αν το παιδί είναι φύλλο, τότε ο αντίστοιχος δείκτης κωδικοποιεί το ποια είναι η θέση του στο αντίστοιχο αρχείο φύλλων του υποδένδρου. Η τιμή αυτή δεν μπορεί ποτέ να γίνει μεγαλύτερη από t, γιατί από το σχεδιασμό που έχει γίνει δεν υπάρχει υποδένδρο με περισσότερα από t φύλλα. Επομένως αν ο αντίστοιχος δείκτης περιέχει την τιμή i, η πληροφορία θα βρεθεί στο σημείο i*8 του αντίστοιχου αρχείου φύλλων, αφού το μέγεθος της κάθε εγγραφής του αρχείου φύλλων είναι 8 χαρακτήρες
  Γνωρίζοντας δηλαδή εκ των προτέρων ότι σε οποιοδήποτε υποδένδρο, υπάρχουν το πολύ t φύλλα, το TRELLIS ορίζει κατά σύμβαση τους δείκτες σε φύλλα στο διάστημα [1,t] . Αν λοιπόν συναντήσουμε έναν δείκτη σε αυτό το διάστημα θα πρέπει να αναζητήσουμε τον αντίστοιχο κόμβο στο αρχείο των δένδρων και όχι στο αρχείο των εσωτερικών κόμβων. Χρησιμοποιώντας πάλι ως παραδειγμα τον εσωτερικό κόμβο με προγονική ακμή A του υποδένδρου TT του σχήματος, ο δείκτης στο δεξί παιδί που είναι φύλλο θα είναι στο διάστημα [1,3] και το φύλλο θα αναζητηθεί στο αρχείο L_44 (44=TT) που είναι το αρχείο φύλλων του αντίστοιχου υποδένδρου
- Αν το παιδί είναι και αυτό εσωτερικός κόμβος, τότε η πληροφορία του θα είναι αποθηκευμένη στο ίδιο αρχείο και ο αντίστοιχος δείκτης κωδικοποιεί το ποια είναι η θέση του μέσα στο αρχείο αυτό. Για να ξεχωρίζει από τις περιπτώσεις των φύλλων έχει γίνει η σύμβαση να προστίθεται το κατώφλι t στον αριθμό της εγγραφής. Επομένως αν ο αντίστοιχος δείκτης περιέχει την τιμή i, η πληροφορία θα βρεθεί στο σημείο (i-t-1)*28 του αρχείου. Είναι προφανές ότι οποιαδήποτε τιμή δείκτη μεγαλύτερη από t αναφέρεται σε εσωτερικό κόμβο. Π.χ. στον ίδιο κόμβο με πριν, το αριστερό παιδί που είναι εσωτερικός κόμβος κωδικοποιείται από κάποιο δείκτη μεγαλύτερο του κατωφλίου, άρα μεγαλύτερο του 3, και αφού στο υποδένδρο έχουμε μόνο τρεις εσωτερικούς κόμβους (και ο ένας είναι η ρίζα που είναι κατά σύμβαση στην αρχή του αρχείου) ο δείκτης μπορεί είτε να είναι ίσος με 5 ή με 6,άρα ο αντίστοιχος κόμβος θα αναζητηθεί είτε στην δεύτερη έγγραφη (byte 28) είτε στην τρίτη εγγραφή (2*28=byte 56) του αρχείου εσωτερικών κόμβων αφού η κάθε εγγραφή του αρχείου εσωτερικών κόμβων έχει μήκος 28 bytes.

Στο φάκελο binData βρίσκονται αποθηκευμένα τα αρχεία των φύλλων για κάθε προθεματικό υποδένδρο. Η ονοματοδοσία των αρχείων των φύλλων είναι διαφορετική από την ονοματοδοσία για τα αρχεία εσωτερικών κόμβων. Εν προκειμένω, τα φύλλα του υποδένδρου του προθέματος 112 (AAC) βρίσκονται στο αρχείο binData/L_112, δηλαδή το κάθε όνομα αρχείου ξεκινάει με "L_" ακολουθούμενο από το ίδιο το κείμενο του προθέματος. Άρα για τα δένδρα του παραδείγματος τα φύλλα εσωτερικών κόμβων για τα τα πρόθεματα C, G, AC,

AG, TT, ATA, ATT, TAC, TAG και TAT βρίσκονται στα αρχεία binData/L_2, binData/L_3, binData/L_12, binData/L_13, binData/L_44, binData/L_141, binData/L_144, binData/L_412, binData/L_413 και binData/L_414 αντίστοιχα.

Το κάθε αρχείο φύλλων είναι ένα αρχείο τυχαίας προσπέλασης, η κάθε εγγραφή του οποίου έχει μέγεθος 8 bytes, και την ακόλουθη δομή:

[*δείκτης_αρχής*, *αριθμός_επιθέματος*]

Το καθένα από τα πεδία της εγγραφής είναι ένας όπως και για τους εσωτερικούς κόμβους ένας μη προσεσημασμένος ακέραιος 4 bytes (UI4Bytes). Συνεπώς, το μέγεθος της εγγραφής του αρχείου φύλλων είναι 8 Bytes.

Ο <*δείκτης τέλους*> των εγγραφών των αρχείων εσωτερικών κόμβων είναι περιττός αφού είναι αυτονόητο ότι πρόκειται για το τέλος της ακολουθίας εισόδου. Ο αριθμός επιθέματος είναι η θέση του επιθέματος στην ακολουθία εισόδου. Για παράδειγμα, στο παραπάνω σχήμα το φύλλο που έχει ως μονοπάτι ετικέτα το επίθεμα 6 στο υποδένδρο C, έχει δείκτη αρχής το 9 όπου ξεκινάει το ATTACATTAGAT$ και αριθμό επιθέματος το 6. Υποννοείται ότι το εύρος μέσα στην ακολουθία εισόδου είναι το [6,20] αφού η ακολουθία εισόδου έχει μήκος 20 χαρακτήρες, ο δείκτης τέλους δηλαδή δεν αποθηκεύεται αφού είναι ο τελευταίος χαρακτήρας της ακολουθιας εισοδου.

Όσον αφορά τους συνδέσμους επιθέματος, αυτοί ανακτώνται με αποδοτικό τρόπο όπως περιγράφεται στο 3.5.4. Μετά την ολοκλήρωση της ανάκτησης των συνδέσμων επιθέματος στον δίσκο έχει δημιουργηθεί ένας νέος φάκελος, ο tempBinDataL/ που περιέχει όλα τα αρχεία εσωτερικών κόμβων, με την ίδια ονοματοδοσία, τροποποιημένα ώστε να περιέχουν ένα index για τον σύνδεσμο επιθέματος. Οι σύνδεσμοι επιθέματος βέβαια αφορούν μόνο τους εσωτερικούς κόμβους. Έτσι η κάθε εγγραφή ενός αρχείου εσωτερικών κόμβων αυξάνεται σε μέγεθος από 28 σε 32 bytes και έχει την ακόλουθη δομή:

[*δείκτης_αρχής, δείκτης_τέλους,, παιδί_0, παιδί_1, παιδί_2, παιδί_3, παιδί_4, σύνδεσμος_επιθέματος*]

όμοια με πριν δηλαδή αλλά με προσθήκη του δείκτη για τον σύνδεσμο επιθέματος. Ο δείκτης αυτός είναι ένας δείκτης αρχείου εσωτερικών κόμβων όπως και οι υπόλοιποι, ακολουθεί δηλαδή ακριβώς τις ίδιες συμβάσεις, αναφέρεται όμως στην γενική περίπτωση σε άλλο αρχείο εσωτερικών κόμβων, και κωδικοποιεί την φυσική θέση στο αρχείο αυτό όπου βρίσκεται ο κόμβος sl(υ) του τρέχοντος κόμβου υ . Το υπολογιστικό κόστος της εύρεσης του αρχείου αυτού αφορά την υλοποίηση. Ο κόμβος στον οποίον καταλήγει ο σύνδεσμος επιθέματος θα βρίσκεται στο προθεματικό υποδένδρο του προθέματος της υποακολουθίας-ετικέτας του μονοπατιού του υ, αν της αφαιρεθεί ο πρώτος χαρακτήρας. Το επίθεμα του τρέχοντος κόμβου θα αναζητηθεί λοιπόν στο αντίστοιχο προθεματικό υποδένδρο στη θέση (*σύνδεσμος_επιθέματος*-t-1)*32 κατά τα γνωστά.

4.1.2 Παρεμβάσεις στο σύστημα αρχείων

Όπως αναφέρθηκε ένα κρίσιμο μέγεθος είναι το κατώφλι της μνήμης t. Στην ενότητα 3.5.5 περιγράφηκαν τα κριτήρια για την εύρεση του κατωφλίου. Ο χρήστης της πλατφόρμας δεν τεκμαίρεται ότι γνωρίζει τις λεπτομέρειες του αλγορίθμου, ούτε μπορεί να θεωρείται εύλογη η αξίωση να υπολογίζει ο ίδιος το κατώφλι από την διαθέσιμη μνήμη μέσω της ανισοτικής σχέσης. Το μέγεθος που παραμετροποιείται λοιπόν είναι η διαθέσιμη μνήμη του συστήματος που είναι εύλογο να θεωρείται γνωστή. Ο υπολογισμός της κατάλληλης τιμής για το κατώφλι γίνεται από τον αλγόριθμο.

Επιπλέον, το κατώφλι ως μέγεθος θα χρησιμοποιηθεί και μετά την κατασκευή, αφού όπως είδαμε οι δείκτες μέσα στις εγγραφές των αρχείων των κόμβων αναφέρονται σε φύλλα ή εσωτερικούς κόμβους ανάλογα με το αν έχουν τιμή μικρότερη ή μεγαλύτερη από το κατώφλι, επομένως η οποιαδήποτε προσπέλαση των αρχείων στον δίσκο προϋποθέτει γνώση της τιμής του κατωφλίου. Όπως είναι αποθηκευμένες οι πληροφορίες το κατώφλι δεν είναι αποθηκευμένο πουθενά ως μέγεθος. Ο χρήστης θα πρέπει να σημειώσει την τιμή που τυπώνεται στην οθόνη μετά την κατασκευή του δένδρου και να την περνάει ως παράμετρο μέσα από κάποια γραμμή εντολών για το τρέξιμο οποιουδήποτε προγράμματος που προσπελαύνει τα δένδρα. Στην γενική περίπτωση το κατώφλι είναι ένας πολυψήφιος αριθμός κάτι που καθιστά το σύστημα δύσχρηστο. Η πρώτη λοιπόν παρέμβαση στο πρόγραμμα ήταν η αποθήκευση κατά την κατασκευή της τιμής του κατωφλίου σε ένα αρχείο με όνομα threshold.txt ώστε να ανακτάται αυτόματα από οποιαδήποτε εφαρμογή χωρίς παραμετροποίηση.

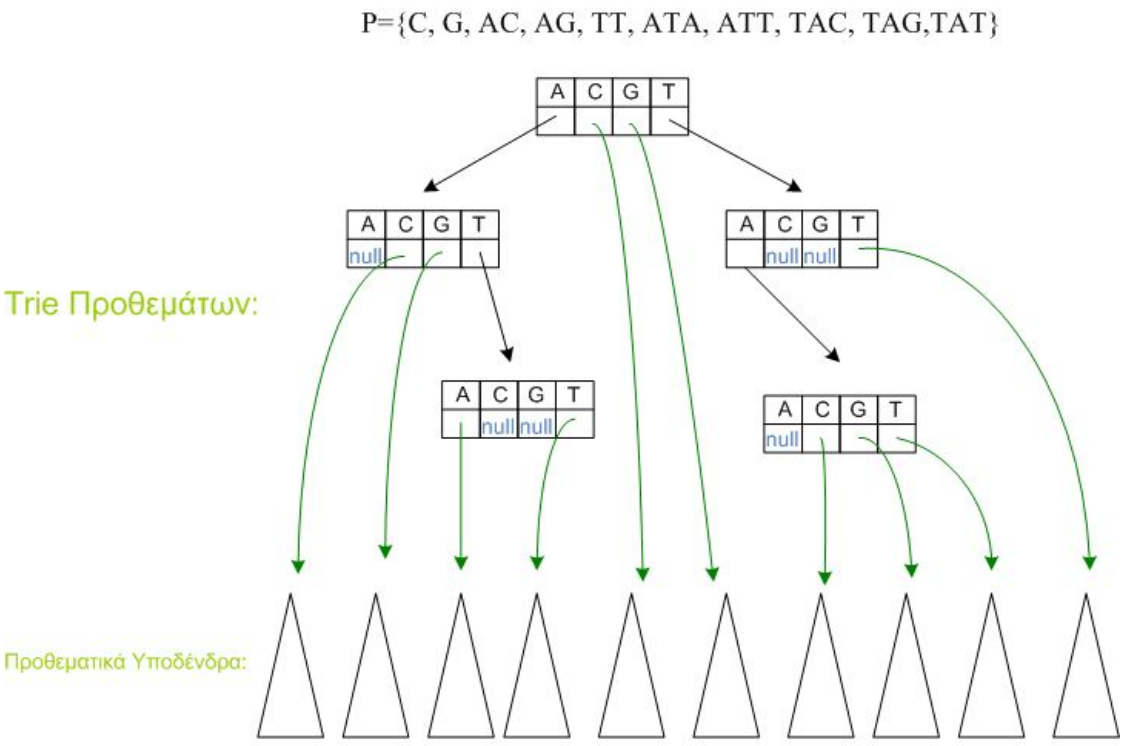
Τα προθέματα μεταβλητού μήκους βρίσκονται αποθηκευμένα στο αρχείο prefix.txt. Το πλήρες δένδρο επιθεμάτων είναι κατακερματισμένο ως δάσος στα διάφορα προθεματικά υποδένδρα. Στο πρόγραμμα του TRELLIS που υλοποιεί την προαιρετική φάση της ανάκτησης των συνδέσμων επιθέματος υπάρχει μια διαδικασία ανάκτησης των προθεμάτων για να προσπελαστεί το αντίστοιχο προθεματικό υποδένδρο. Σε πρώτη φάση είναι επιθυμητό να διαβαστούν όλα τα προθέματα για να είναι γνωστά και να βρίσκονται στην μνήμη. Για τις ανάγκες του προγράμματος είναι περιττή η συχνότητα του κάθε προθέματος άρα η ανάγνωση γίνεται γραμμή παρά γραμμή. Δημιουργείται ένα διάνυσμα (vector) από strings, ονόματι prefVec, στο οποίο αποθηκεύονται με διαδοχικά push_back τα προθέματα με τη σειρά που παρουσιάζονται στο prefix.txt. Παράλληλα με την αποθήκευση στο διάνυσμα, σε έναν χάρτη (map) <string, int>, ονόματι pmap, αποθηκεύονται επίσης τα προθέματα με κλειδί το string, δηλαδή την ακολουθία του προθέματος, και πληροφορία int τον αυξοντα αριθμό του κάθε προθέματος στο αρχείο prefix.txt. Ο λόγος που δημιουργείται αυτός ο χάρτης θα γίνει κατανοητός αμέσως παρακάτω. Μετά το πέρας της ανάγνωσης του αρχείου έχουμε λοιπόν δύο δομές στις οποίες είναι αποθηκευμένα τα προθέματα, το διάνυσμα prefVec και το χάρτη

pmap. Μόλις ολοκληρωθεί η ανάγνωση τα περιεχόμενα του prefVec (τα προθέματα δηλαδή) ταξινομούνται με χρήση του έτοιμου αλγόριθμου sort. Στην συνέχεια όταν αναζητείται κάποια υποακολουθία (εν προκειμένω το επίθεμα α μιας υποακολουθίας χα, όπου x ένας χαρακτήρας, ώστε να βρεθεί ο σύνδεσμος επιθέματος) πρέπει να βρεθεί το πρόθεμα της υποακολουθίας που βρίσκεται μέσα στο σύνολο P των προθεμάτων μεταβλητού μήκους. Το πρόγραμμα κάνει δυαδική αναζήτηση με χρήση της έτοιμης συνάρτησης της στάνταρ βιβλιοθήκης προτύπων της C++ (STL). Το αποτέλεσμα αποθηκεύεται σε κάποια μεταβλητή, έστω prefix, και κατόπιν ανοίγεται το αρχείο με όνομα pmap[prefix] εσωτερικών κόμβων που βρίσκεται στο φάκελο tempBinData. Κάποιο πρόγραμμα που πραγματοποιεί προσπελάσεις σε αρχεία φύλλων και χρησιμοποιεί την ίδια διαδικασία για την ανάκτηση των προθεμάτων θα άνοιγε το αρχείο "L_"+prefix στον φάκελο binData.

Η πρώτη παρατήρηση είναι ότι με την χρήση του map γίνεται μια πολυέξοδη αναζήτηση με κλειδί string, λόγω του διαφορετικού τρόπου ονοματοδοσίας των αρχείων εσωτερικών κόμβων και φύλλων. Μια προσέγγιση είναι να αλλάξει η ονοματοδοσία των αρχείων εσωτερικών κόμβων ώστε να ονομάζονται για παράδειγμα "I_"+prefix, ή οποιαδήποτε ονομασία θα περιείχε το πλήρες κείμενο του προθέματος, ή να αποθηκεύεται στην ίδια θέση, σε κάποιο struct δύο εγγραφών, ο αύξων αριθμός και το πλήρες κείμενο του προθέματος για να αποφευχθεί η χρήση του map. Μια δεύτερη παρατήρηση είναι ότι σε κάθε άνοιγμα αρχείου πρέπει να πραγματοποείται δυαδική αναζήτηση στο διάνυσμα των προθεμάτων. Σε κάθε σύγκριση ακολουθιών που πραγματοποιεί ο αλγόριθμος της δυαδικής αναζήτησης πραγματοποιούνται O(μ) συγκρίσεις χαρακτήρων, όπου μ το μέγιστο μήκος προθέματος στο P, άρα η αναζήτηση έχει συνολικό κόστος O(μlogp) όπου p το πλήθος των προθεμάτων μεταβλητού μήκους. Αν χρησιμοποιήσουμε την μέθοδο αυτή για την αναζήτηση κάποιου προτύπου μήκους m νοθεύεται η γραμμική πολυπλοκότητα O(m) της αναζήτησης κάποιου προτύπου με χρήση δένδρων επιθεμάτων αφού για τους πρώτους χαρακτήρες θα πρέπει να πραγματοποιηθεί δυαδική αναζήτηση για την εύρεση του αντίστοιχου προθέματος.

Υλοποίησαμε λοιπόν ένα trie για την αποθήκευση όλων των προθεμάτων και την γραμμική αναζήτηση των προθεμάτων σε αυτό. Το trie δημιουργείται κατά την κατασκευή των δένδρων και αποθηκεύεται στο αρχείο prefix_trie. Διατίθενται επιπλέον οι συναρτήσεις LoadTrie( ) και getPrefixFromTrie( ) για την φόρτωση στην μνήμη και την ανάκτηση προθεμάτων από το trie. Το trie αποθηκεύεται στην μνήμη σε πίνακα όπου κάθε στοιχείο του πίνακα είναι μια δομή 4 εγγραφών, μία για κάθε σύμβολο του αλφαβήτου. Ο τερματικός χαρακτήρας δεν περιλαμβάνεται στο αλφάβητο των προθεμάτων. Οι εγγραφές κάθε στοιχείου περιέχουν έναν δείκτη με την θέση που βρίσκεται ο κόμβος παιδί στον πίνακα. Αν ο κόμβος-παιδί για κάποιο σύμβολο δεν υπάρχει ο δείκτης αυτός έχει την τιμή MAX_INT, και θεωρείται null. Αν κάποιος κόμβος δείχνει σε κάποιο στοιχείο του πίνακα όπου όλα τα παιδιά

είναι null, η αναζήτηση τελειώνει και το πρόθεμα έχει βρεθεί. Το trie των προθεμάτων για την ακολουθία S του παραδείγματος φαίνεται παρακάτω



Με την χρήση του trie για την αναζήτηση προθεμάτων το κόστος της ανάκτησης ενός προθέματος σε συγκρίσεις είναι O(μ). Επιπλέον, το κόστος των συγκρίσεων για την αναζήτηση κάποιου προτύπου είναι O(m) αφού οι συγκρίσεις γίνονται γραμμικά μέσα στο trie για τους πρώτους χαρακτήρες και συνεχίζουν στο προθεματικό υποδένδρο. Παρεμβάλλεται βέβαια το I/O κόστος για την φόρτωση του προθεματικού υποδένδρου από τον δίσκο. Η ταχύτητα ανάκτησης των προθεμάτων αυξάνεται κατά έναν παράγοντα 2-4 σε σχέση με την δυαδική αναζήτηση, στην πράξη ωστόσο για πρότυπα μεγάλου μήκους ο χρόνος ανάκτησης των προθεμάτων είναι σε κάθε περίπτωση πολύ μικρός σε σχέση με τις υπόλοιπες λειτουργίες όπως το φόρτωμα των αρχείων στον δίσκο και η διάσχιση στο δένδρο, και η συνολική βελτίωση του χρόνου είναι μικρή. Η διαφορά είναι πιο αισθητή όταν αναζητούνται μικρότερα πρότυπα. Η δημιουργία του trie κάνει το σύστημα κομψότερο εννοιολογικά, αφού το πλήρες δένδρο επιθεμάτων δεν νοείται πλέον ως ένα κατακερματισμένο δάσος προθεματικών υποδένδρων, αλλά ως μια ενιαία δενδρική δομή.

4.1.3 Προεπεξεργασία των δένδρων για αποδοτική ανάκτηση του LCE

Όπως περιγράφθηκε στην 2.2.2.3 μετά από μια γραμμική προεπεξεργασία ενός δένδρου μπορεί να ανακτηθεί ο κατώτατος κοινός πρόγονος (LCA) δύο οποιωνδήποτε κόμβων του

δένδρου σε σταθερό χρόνο. Αν η προεπεξεργασία γίνει σε κάποιο δένδρο επιθεμάτων,το ερώτημα για τον κατώτατο κοινό πρόγονο δύο φύλλων του δένδρου ισοδυναμεί με την εύρεση της μέγιστης κοινής επέκτασης (LCE) των επιθεμάτων που αντιστοιχούν στα φύλλα. Η εύρεση του μήκους της μέγιστης κοινής επέκτασης είναι το μήκος μονοπατιού του LCA κόμβου, οπότε για να ισχύει η πολυπλοκότητα O(1) το μήκος του μονοπατιού πρέπει να παραμένει αποθηκευμένο σε κάθε κόμβο του δένδρου ώστε να ανακτάται όταν γίνεται η προσπέλαση του κόμβου χωρίς υπολογιστικό κόστος. Υλοποιήσαμε λοιπόν προεπεξεργασία των προθεματικών υποδένδρων που δημιουργούνται από το TRELLIS για την αποθήκευση όλων των πληροφοριών που χρειάζονται για την μετέπειτα ανάκτηση της μέγιστης κοινής επέκτασης δύο οποιοωνδήποτε θέσεων της ακολουθίας εισόδου σε χρόνο O(1). Η προεπεξεργασία αφορά όλα τα προθεματικά υποδένδρα.

Όπως αναφέραμε στην 2.2.2.3 το πρώτο βήμα της προεπεξεργασίας προυποθέτει την pre-order (προτεραιότητα βάθους) αρίθμηση των κόμβων του δένδρου. Έστω ένας κόμβος του δένδρου υ. Η pre-order αρίθμηση του κόμβου θα συμβολιστεί επίσης με υ, αφού για κάθε κόμβο η αρίθμηση που προκύπτει από την διάσχιση με προτεραιότητα βάθους είναι μοναδική, και η αναφορά σε μία αρίθμηση ισοδυναμεί με αναφορά σε έναν και μόνο κόμβο. Για τον αριθμό υ που προκύπτει από την pre-order αρίθμηση πρέπει στην συνέχεια να υπολογιστεί η συνάρτηση h(υ) για κάθε κόμβο υ και να αποθηκευτεί στον κόμβο.

Όπως αναφέρθηκε στην 4.1.1 τα αρχεία των εσωτερικών κόμβων του TRELLIS, δεν περιέχουν σε κάποιο πεδίο των εγγραφών τους το μήκος των μονοπατιών των αντίστοιχων κόμβων και είναι ζήτημα αυτού που υλοποιεί την οποιαδήποτε διάσχιση στα δένδρα να κρατάει σε κάποιον μετρητή την πληροφορία για το μονοπάτι κατά την διάσχιση, ενώ αν δοθεί ο δείκτης σε κάποιον κόμβο για άμεση προσπέλαση είναι αδύνατο να ανακτηθεί το μήκος του μονοπατιού . Γι' αυτό το λόγο το πρώτο βήμα της προεπεξεργασίας θα πρέπει να αποθηκεύει το μήκος του μονοπατιού του κάθε κόμβου κατά την διάσχιση για την pre-order αρίθμηση. Το δεύτερο βήμα του αλγορίθμου ανακτά τις τιμές I(υ) για κάθε κόμβο υ με μία από κάτω προς τα πάνω διάσχιση. Η μόνες τιμές I(υ) που μπορούν να υπολογιστούν χωρίς την χρήση άλλων τιμών I( ) κόμβων παιδιών του υ, είναι οι τιμές I( ) των φύλλων, για τις οποίες ισχύει I(υ)=υ. Ο υπολογισμός λοιπόν της I ( ) τιμής για τα κόμβους-φύλλα μπορεί να γίνει και αυτός στο πρώτο βήμα ταυτόχρονα με την pre-order αρίθμηση.

Τα παραπάνω βήματα αφορούν το πρώτο βήμα της προεπεξεργασίας για την γρήγορη απάντηση ερωτημάτων LCA για τους κόμβους του δένδρου. Ωστόσο, το ζητούμενο δεν είναι η απάντηση του ερωτήματος LCA, αλλά ερωτημάτων LCE για θέσεις της ακολουθίας. Οι είσοδοι λοιπόν της συνάρτησης δεν είναι κόμβοι δένδρων αλλά θέσεις μέσα στην ακολουθία. Κάθε θέση της ακολουθίας εισόδου αντιστοιχεί σε ένα επίθεμα, το οποίο αναπαρίσταται ως φύλλο σε κάποιο προθεματικό υποδένδρο. Πρέπει λοιπόν να μπορεί να γίνει μια γρήγορη

αντιστοίχηση οποιασδήποτε θέσης στην ακολουθία σε έναν pre-order κόμβο κάποιου προεπεξεργασμένου υποδένδρου. Ο αλγόριθμος θα πρέπει να μπορεί σε σταθερό χρόνο να βρίσκει τον κόμβο φύλλο που αντίστοιχει σε κάθε επίθεμα. Αν αυτό δεν είναι εφικτό, ο αλγόριθμος δεν θα είναι σταθερού χρόνου καθώς θα πρέπει να διασχίσει το δένδρο από την ρίζα μέχρι το αντίστοιχο φύλλο. Η διάσχιση από την ρίζα μέχρι το φύλλο μπορεί να εντοπίσει και τον LCA δύο φύλλων σε αριθμό βημάτων που φράσσεται από το βάθος του δένδρου, οπότε όλα τα υπόλοιπα βήματα του αλγορίθμου θα ήταν περιττά. Άρα χρειάζεται κάποια δομή στην μνήμη που με διεύθυνση την θέση στην ακολουθία να επιστρέφει τον κόμβο υ του προεπεξεργασμένου προθεματικού υποδένδρου όπου είναι αποθηκευμένη. Είναι απαραίτητος λοιπόν ένας πίνακας μεγέθους n, που σε κάθε θέση j, αποθηκεύει την pre-order αρίθμηση για τον κόμβο που αντιστοιχεί στο επίθεμα j. Για τον σκοπό αυτό, κατά την pre-order αρίθμηση των κόμβων στην προεπεξεργασία, κάθε φορά που η διάσχιση φθάνει σε κάποιον κόμβο-φύλλο υ που αντιστοιχεί σε κάποιο επίθεμα j, αποθηκεύει στην θέση j του πίνακα τον αριθμό υ. Μόλις τελειώσει η προεπεξεργασία για όλα τα προθεματικά υποδένδρα, ο πίνακας αυτός θα έχει τιμές για όλα τα επιθέματα της S που υπάρχουν στα δένδρα. Ο πίνακας αυτός θα αποθηκευτεί στο αρχείο suffix.dat.

Αναφέρθηκε στην 3.5.3 ότι το σύστημα TRELLIS αποθηκεύει στον δίσκο τα προθεματικά υποδένδρα για τους εσωτερικούς κόμβους και τα φύλλα με προτεραιότητα βάθους. Η αρίθμηση που γίνεται με βάση την σειρά που αποθηκεύονται μέσα στα αρχεία οι κόμβοι δεν είναι μια pre-order αρίθμηση αφού το δένδρο είναι διασπασμένο σε δύο κομμάτια, ένα για τους εσωτερικούς κόμβους και ένα για τα φύλλα. Η αποθήκευση σε διαφορετικές τοποθεσίες γίνεται για την εξοικονόμηση χώρου αφού οι εγγραφές για κόμβους-φύλλα απαιτούν λιγότερο χώρο από τις εγγραφές για εσωτερικούς κόμβους. Ο αλγόριθμος απαιτεί όμως μια pre-order διάσχιση και αρίθμηση για όλους τους κόμβους του πλήρους υποδένδρου. Ως εκ τούτου όλοι οι κόμβοι (εσωτερικοί και φύλλα) πρέπει να διασχιστούν με προτεραιότητα βάθους και να αποθηκευτούν με την σειρά που προκύπτει από την αρίθμηση αυτή σε ένα καινούριο ενιαίο αρχείο. Η επαναδιάταξη των κόμβων δηλαδή είναι ένα βήμα απαραίτητο αφού η με προτεραιότητα βάθους αποθήκευση που γίνεται από το TRELLIS δεν οδηγεί σε μία αρίθμηση που είναι ισοδύναμη με την pre-order αρίθμηση του πλήρους υποδένδρου.

Ένα τελευταίο απαραίτητο βήμα είναι η δημιουργία ενός δείκτη από κάθε κόμβο στον κόμβο γονέα του, αφού για την υλοποίηση του αλγορίθμου είναι απαραίτητο να μπορούμε σε σταθερό χρόνο να προσπελάσουμε τον γονέα οποιουδήποτε κόμβου. Τα παιδιά ωστόσο των κόμβων δεν είναι απαραίτητα για τον αλγόριθμο.

Επομένως κατά το πρώτο βήμα της προεπεξεργασίας πραγματοποιούνται τα ακόλουθα:

- Διάσχιση με προτεραιότητα βάθους (pre-order) όλων των κόμβων του προθεματικού υποδένδρου. Μετά την ολοκλήρωση του βήματος οι κόμβοι θα πρέπει να αποθηκευτούν στον δίσκο με σειρά προτεραιότητας βάθους.
- Υπολογισμός της τιμής h(υ) για κάθε κόμβο υ (όπου υ είναι η pre-order αριθμήσή του) και αποθήκευση στον κόμβο
- Υπολογισμός του μήκους του μονοπατιού του κόμβου και αποθήκευση του στον κόμβο. Το μήκος θα προκύψει από την πρόσθεση του μήκους της υποακολουθίας ετικέτας της προσπίπτουσας ακμής και του μήκους του μονοπατιού του κόμβου-γονέα. Η τιμή αυτή θα πρέπει να προωθηθεί αναδρομικά στα παιδιά του κόμβου για να συνεχιστεί ο υπολογισμός
- Δημιουργία δείκτη προς τον γονέα του κόμβου
- Υπολογισμός της τιμής I(υ) μόνο για τα φύλλα (I(υ)=υ) και αποθήκευσή της
- Αποθήκευση της τιμής υ για κάθε κόμβο-φύλλο στην θέση j ενός πίνακα, όπου j είναι ο <*αριθμός_επιθέματος*> του φύλλου. Μόλις ολοκληρωθεί η προεπεξεργασία για όλα τα υποδένδρα, ο πίνακας θα περιέχει την pre-order αρίθμηση όλων των επιθεμάτων που βρίσκονται στα προθεματικά υποδένδρα, και θα αποθηκευτεί στον δίσκο στο αρχείο suffix.dat.

Στην συνέχεια η προεπεξεργασία θα προχωρήσει στο επόμενο βήμα της που είναι αυτό του υπολογισμού των I τιμών για όλους τους κόμβους I. Η τιμή του I( ) προϋποθέτει την pre-order αρίθμηση και διασχίζει το δένδρο από τα κάτω προς τα πάνω (bottom-up), σύμφωνα με τα όσα περιγράφηκαν στην 2.2.2.3. Επιπλέον για κάθε τιμή I(υ) που υπολογίζεται πρέπει να βρεθεί και το L(I(υ)) και να αποθηκευτεί με τρόπο που να μπορεί να ανακτάται σε σταθερό χρόνο. Το L(I(υ)) δείχνει στην *κεφαλή* του *δρόμου* που ορίζεται από την τιμή I(υ) (βλέπε 2.2.2.3). Η κεφαλή υπολογίζεται εύκολα ως εξής. Ο κόμβος υ είναι η κεφαλή του δρόμου του αν η τιμή I(υ') του γονέα του είναι διαφορετική από την I(υ). Κατά την bottom-up διάσχιση πρέπει δηλαδή να ελέγχονται η τιμές I(υ) των παιδιών αμέσως μόλις υπολογιστεί η I (υ') τιμή του γονέα. Αν κάποια τιμή παιδιού είναι διαφορετική από αυτή του γονέα, τότε γίνεται μετάβαση στον κόμβο I(υ)=ω (είναι σίγουρο ότι υπάρχει αυτός ο κόμβος αφού η I(υ) τιμή ορίζεται ως ο κόμβος ω που για τον οποίον h(ω) είναι το μέγιστο σε όλο το υποδένδρο του υ) και αποθηκεύεται ο δείκτης στην κεφαλή, L(ω)=υ. Άρα είναι εφικτή η ανάκτηση της τιμής L(I(υ)) σε σταθερό χρόνο προσπελαύνοντας τον κόμβο I(υ) και εξετάζοντας τον δείκτη L.

Μετά την ολοκλήρωση του δεύτερου βήματος απομένει ο υπολογισμός του δυαδικού αριθμού $A_υ$ για κάθε κόμβο υ. Η διάσχιση γίνεται με προτεραιότητα βάθους όπως και στο πρώτο βήμα, και απαιτεί την γνώση των τιμών I των κόμβων, γιαυτό ο υπολογισμός αυτός δεν μπορεί να πραγματοποιηθεί κατά το πρώτο βήμα. Ο υπολογισμός πραγματοποιείται για

κάθε κόμβο με χρήση της τιμής A_v, του κόμβου γονέα και την πραγματοποίηση των κατάλληλων δυαδικών πράξεων (βλέπε 2.2.2.3).

Ονομάσαμε το πρόγραμμα με το οποίο πραγματοποιείται η προεπεξεργασία tree_preprocess. Δέχεται ως είσοδο μία παράμετρο που είναι ο φάκελος που περιέχει όλα τα αρχεία του TRELLIS. Στον φάκελο δηλαδή πρέπει να περιέχονται τα αρχεία prefix.txt, prefix_trie η ακολουθία εισόδου σε ένα αρχείο num με όνομα ίδιο με το όνομα του φακέλου, και οι δύο φάκελοι tempBinData και binData με τα αρχεία εσωτερικών κόμβων και φύλλων.

Μετά την ολοκλήρωση της προεπεξεργασίας όλα τα αρχεία του δίσκου που υπήρχαν από πριν παραμένουν ως είχαν. Επιπλέον στον δίσκο υπάρχουν:

- Τα αρχεία που για κάθε προθεματικό υποδένδρο περιέχουν όλες τις πληροφορίες για την ανάκτηση του LCA δύο κόμβων του υποδένδρου, και τοποθετούνται στον φάκελο binData/ μαζί με τα αρχεία φύλλων
- Το αρχείο suffix.dat μεγέθους που περιέχει τις πληροφορίες για την pre-order αρίθμηση (και θέση) κάθε επιθέματος μέσα στα νέα αρχεία.

Τα αρχεία με την πληροφορία για το LCA, που θα αποκαλούνται lca αρχεία, ονομάζονται "L_"+<κείμενο_προθέματος>+"_lca.dat" κατά τα πρότυπα της ονοματοδοσίας των φύλλων. Τα αρχεία επιλέχθηκε να μπουν στον φάκελο με τα φύλλα αυθαίρετα, θα μπορούσαν να είχαν αποθηκευθεί μαζί με τους εσωτερικούς κόμβους, ή αλλού σε ξεχωριστό φάκελο. Κάθε εγγραφή των αρχείων lca αντιπροσωπεύει έναν κόμβο και περιέχει 6 πεδία, με μήκος 4 bytes το καθένα, επομένως έχει μήκος 24 bytes. Η σειρά με την οποία είναι γραμμένοι οι κόμβοι στο αρχείο είναι η σειρά της pre-order αρίθμησης. Η δομή κάθε εγγραφής είναι:

[φυσική_θέση_στα_παλαιά_αρχεία, τιμή_I, δείκτης_γονέα, δυαδικός_αριθμός_A, μήκος_μονοπατιού, κεφαλή]

Οι δείκτες στα παιδιά της ακολουθίας είναι περιττοί για τον αλγόριθμο σταθερού χρόνου ανάκτησης του LCA και γι'αυτό δεν αποθηκεύονται. Η φυσική θέση στα παλαιά αρχεία, αποθηκεύεται για να μπορεί να γίνει αντιστοίχηση των κόμβων των lca αρχείων με τους κόμβους των παλαιών αρχείων αν κάτι τέτοιο είναι επιθυμητό. Φυσικά, ούτε αυτή η πληροφορία είναι απαραίτητη για τον αλγόριθμο του LCA. Με χρήση της τιμής αυτής μπορούμε μάλιστα να καθορίσουμε αν ο κόμβος είναι φύλλο η δένδρο από το εάν η τιμή είναι μικρότερη ή μεγαλύτερη από το κατώφλι.

Το αρχείο suffix.dat περιέχει εγγραφές που αντιστοιχούν σε επιθέματα στην ακολουθία εισόδου και έχουν 1 πεδίο 4 χαρακτήρων όπου αποθηκεύεται η τιμή της pre-order αρίθμησης (και άρα της φυσικής θέσης) όπου βρίσκεται το εκάστοτε επίθεμα στα lca αρχεία.

Με φόρτωση στην μνήμη ενός αρχείου lca, είναι διαθέσιμες όλες οι απαραίτητες πληροφορίες για την ανάκτηση του LCA σε σταθερό χρόνο για δύο κόμβους x και y του

δένδρου. Επιπλέον, με φόρτωση του αρχείου suffix.dat είναι δυνατή η ανάκτηση σε σταθερό χρόνο της μέγιστης κοινής επέκτασης δύο θέσεων της ακολουθίας που αντιστοιχούν σε επιθέματα που έχουν τα φύλλα τους στο ίδιο υποδένδρο. Οι δύο θέσεις αντιστοιχίζονται στους κόμβους στο επεξεργασμένο δένδρο,και εφαρμόζεται ο αλγόριθμος του LCA για τους κόμβους αυτούς. Η μέγιστη κοινή επέκταση είναι το άθροισμα του πεδίου <*μήκος_μονοπατιού*> του LCA και του μήκους του (κοινού, εξ'υποθέσεως) προθέματος των επεκτάσεων.

Είδαμε πως για να μπορεί να αξιωθεί το αποτέλεσμα σταθερού χρόνου, στην μνήμη πρέπει να βρίσκεται και ένας πίνακας επιθεμάτων το πολύ n στοιχείων και μεγέθους το πολύ 4n bytes, αφού χρησιμοποιούμε δείκτες τύπου μη προσεσημασμένου ακεραίου 4 χαρακτήρων. Το γεγονός αυτό τροποποιεί την τιμή του κατωφλίου που πρέπει να επιλεγεί πριν την κατασκευή των δένδρων αφού πρέπει να συνυπολογισθεί και η ανάγκη ύπαρξης μιας δομής στην μνήμη με μέγεθος 4n χαρακτήρες, αν σκοπεύεται στην συνέχεια να χρησιμοποιεί ο αλγόριθμος σταθερού χρόνου στα δένδρα. Για κάθε φύλλο και δένδρο η αναπαράσταση στην μνήμη είναι τώρα η ίδια και ίση με 24 bytes. Τα φύλλα είναι t και οι εσωτερικοί κόμβοι 0,7t, οπότε κάθε προεπεξεργασμένο υποδένδρο έχει μήκος της τάξεως των 24·1,7t=40,8t χαρακτήρων. Η ανισοτική σχέση για τον υπολογισμό του κατωφλίου, αν η βασική μνήμη είναι M, είναι:

$$M \geq \frac{5n}{4} + 40{,}8t \Rightarrow t \leq \frac{M - \frac{5n}{4}}{40{,}8}$$

και επιλέγεται η μεγαλύτερη τιμή του κατωφλίου για την οποία ισχύει η σχέση. Για αρκετά μεγάλα n η τιμή αυτή είναι μικρότερη από την τιμή που προκύπτει από την ανισοτική σχέση της 3.5.5 οπότε επιλέγεται οριστικά ως κατώφλι. Χρησιμοποιώντας δηλαδή τον αλγόριθμο σταθερού χρόνου καταβάλλεται ως αντίτιμο η μείωση της τιμής κατωφλίου, άρα θα υπάρχουν μικρότερα και περισσότερα προθεματικά υποδένδρα. Αν η τιμή αυτή είναι μεγαλύτερη από την τιμή που προκύπτει από την ανισοτική σχέση στην 3.5.5 επιλέγεται για κατώφλι η μικρότερη τιμή.

4.2 Υλοποιήσεις στα υπάρχοντα δένδρα του TRELLIS

4.2.1 Υλοποίηση ακριβούς ταύτισης προτύπου (exaxt matching)

Όπως αναλύθηκε στην 2.2.1.3 τα δένδρα επιθεμάτων χρησιμοποιούνται ως γρήγορα ευρετήρια για την ακριβή ταύτιση προτύπων μέσα σε κείμενο με πολυπλοκότητα O(m), όπου m το μήκος του προτύπου, ανεξάρτητη δηλαδή της ακολουθίας εισόδου. Η ακριβής ταύτιση είναι η πιο βασική και απλή χρήση των δένδρων επιθεμάτων. Η πρώτη υλοποίηση αφορά την

δημιουργία ενός προγράμματος για την πραγματοποίηση ερωτημάτων ακριβούς ταύτισης πάνω στο δένδρο. Το πρόγραμμα δέχεται ως είσοδο (παράμετρο):

- Έναν φάκελο όπου είναι αποθηκευμένα όλα τα αρχεία εισόδου, διατηρώντας την ίδια δομή φακέλων. Υπάρχουν δηλαδή τα αρχεία prefix.txt, threshold.txt, prefix_trie και στους υποφακέλους tempBinData και binData τα αρχεία εσωτερικών κόμβων και φύλλων. Υπάρχει ακόμα ένα αρχείο num με όνομα ίδιο με το όνομα του φακέλου όπου βρίσκεται αποθηκευμένη η ακολουθία εισόδου με βάση το αλφάβητο Σ={1,2,3,4}. Αν λοιπόν ο φάκελος ονομάζεται chr1_10MB, η ακολουθία εισόδου θα αναζητηθεί στο αρχείο chr1_10MB.num μέσα στο φάκελο.
- Το πρόγραμμα μπορεί να δέχεται και δεύτερη παράμετρο το όνομα του αρχείου που περιέχει την ακολουθία του προτύπου. Αν η παράμετρος αυτή παραλείπεται τότε by default η ακολουθία του προτύπου είναι το περιεχόμενο του αρχείου query.num που βρίσκεται στον ίδιο φάκελο με το πρόγραμμα.

Στην πρώτη φάση το πρόγραμμα φορτώνει στην μνήμη τις ακολουθιες της εισόδου και του προτύπου. Φορτώνονται στη μνήμη στο σύνολο τους ως αντικείμενα της κλάσης BitString με χρήση της μεθόδου init. Η κλάση BitString είναι διαθέσιμη από τους ήδη υπάρχοντες κώδικες του προγράμματος και χρησιμοποιείται από το TRELLIS κατά την κατασκευή των δένδρων για να διατηρείται με οικονομικό τρόπο στην μνήμη η ακολουθία εισόδου. Η κλάση εκμεταλλεύεται το γεγονός ότι έχουμε ειδικά δεδομένα με αλφάβητο μόλις 4 συμβόλων, και συνεπώς σε κάθε byte χωρούν 4 σύμβολα των δεδομένων μας. Γίνεται λοιπόν συμπίεση με αναλογία 4:1 στα δεδομένα εισόδου. Υπερφορτώνεται ο τελεστής [ ] και έτσι μπορούμε να έχουμε λειτουργίες οιονεί πίνακα. Επομένως αν η μεταβλητή inputString είναι αντικείμενο της κλάσης BitString, η εντολή inputString.init("human_genome.num") φορτώνει στη μνήμη σε συμπιεσμένη μορφή το περιεχόμενο του αρχείου human_genome.num, ενώ η παράσταση inputString[100] επιστρέφει τον εκατοστό χαρακτήρα του αρχείου εισόδου που είναι αποθηκευμένος σε συμπιεσμένη μορφή στη μνήμη, κάνοντας τις κατάλληλες bit-level πράξεις.

Στην συνέχεια φορτώνεται στην μνήμη η τιμή του κατωφλίου από το αρχείο threshold.txt και το trie των προθεμάτων από το αρχείο prefix_trie υπό μορφή πίνακα όπως περιγράφθηκε στην 4.1.2. Με χρήση της συνάρτησης getPrefixFromTrie( ) γίνεται διάσχιση του trie των προθεμάτων με είσοδο τους πρώτους χαρακτήρες της ακολουθίας του προτύπου. Αν μέσα στον trie των προθεμάτων δεν βρεθεί κάποιο πρόθεμα του προτύπου το πρότυπο δεν απαντά στην ακολουθία και η αναζήτηση τερματίζει τυπώνοντας μήνυμα μη επιτυχούς ταύτισης. Αν το πρόθεμα βρεθεί μέσα στο trie, και υπάρχουν και άλλοι χαρακτήρες του προτύπου προς αναζήτηση, το πρόγραμμα ανοίγει τα αρχεία εσωτερικών κόμβων και φύλλων του αντίστοιχου προθεματικού υποδένδρου.

Αφού ανοιχθούν τα αρχεία εσωτερικών κόμβων και φύλλων για το πρόθεμα που έχει βρεθεί πραγματοποείται η αναζήτηση εντός του δέντρου με την συνάρτηση search. Η συνάρτηση search κάνει χρήση των συναρτήσεων popIn και popLeaf που επιστρέφουν σε μεταβλητές τα πεδία των εγγραφών μιας συγκεκριμένης θέσης των αρχείων εσωτερικών κόμβων και φύλλων αντίστοιχα.

Η συνάρτηση search ξεκινάει ανακτώντας με μια popIn τον root κόμβο του δένδρου. Στη συνέχεια με κριτήριο τον πρώτο χαρακτήρα του προτύπου που ακολουθεί τους χαρακτήρες του προθέματος, αναζητά το αντίστοιχο παιδί του root. Γίνεται ανάκτηση του κόμβου-παιδιού με χρήση της popIn (ή της popLeaf) και στη συνέχεια γίνεται χαρακτήρα-προς-χαρακτήρα σύγκριση της ακολουθίας εισόδου και του προτύπου (patternString[i]==inputString[i] για κάποιον iterator i) στο ευρος που ορίζουν τα πεδία [<δείκτης αρχής>, <δείκτης τέλους>]. Αν η σύγκριση αποτύχει το αποτέλεσμα της αναζήτησης είναι αρνητικό και τυπώνεται σχετικό μήνυμα. Αν εξαντληθεί με επιτυχία το ευρος της υποακολουθίας που κωδικοποιεί η ακμή και παραμένουν χαρακτήρες προς αναζήτηση στο πρότυπο, πρέπει με νέο popIn (ή popLeaf αν το παιδί είναι φύλλο) να επαναληφθεί η διαδικασία αυτή για τον νέο κόμβο (ή φύλλο) που προκύπτει ως το παιδί στο οποίο αντιστοιχεί ο πρώτος χαρακτήρας στο πρότυπο μετά το πέρας της υποακολουθίας που μόλις ελέγχθηκε. Το αν έχουμε να κάνουμε με εσωτερικό κόμβο η φύλλο, και συνεπώς το αν θα χρησιμοποιηθεί η PopIn ή η PopLeaf, προκύπτει από το αν το index του παιδιού είναι μεγαλύτερο ή όχι από το κατώφλι t όπως περιγράφηκε παραπάνω. Η αναζήτηση συνεχίζει διασχίζοντας το δένδρο κατά βάθος όσο οι χαρακτήρα-προς-χαρακτήρα συγκρίσεις είναι επιτυχημένες και απομένουν χαρακτήρες στο πρότυπο.

Αν η αναζήτηση τερματίσει επιτυχώς σε εσωτερικό κόμβο τυπώνεται σχετικό μήνυμα, και στην συνέχεια με κλήση της RetrieveAllLeafs τυπώνεται μια λίστα με όλες τις θέσεις της ακολουθίας εισόδου στις οποίες βρίσκεται η υποακολουθία που εντοπίστηκε. Η συνάρτηση RetrieveAllLeafs ανακτά, με αναδρομική κλήση του εαυτού της για όλα τα παιδιά, όλα τα φύλλα που βρίσκονται κάτω από τον τρέχοντα κόμβο και τυπώνει το επίθεμα στο οποίο αντιστοιχεί το καθένα. Αν η αναζήτηση τερματίσει σε ακμή που καταλήγει σε φύλλο (που συμβαίνει στην πλειονότητα των περιπτώσεων για πρότυπα σημαντικού μεγέθους) η θέση είναι αποκλειστική ,τυπώνεται σχετικό μήνυμα και ο αριθμός του επιθέματος του φύλλου, η θέση δηλαδή όπου απαντά το προτύπο μέσα στην ακολουθία εισόδου. Αν σε οποιοδήποτε σημείο της αναζήτησης υπάρξει αποτυχημένη ταύτιση, το πρότυπο δεν υπάρχει μέσα στην ακολουθία εισόδου και τυπώνεται σχετικό μήνυμα.

Έχουμε διαθέσιμες αρκετές παραλλαγές του προγράμματος για την ακριβή ταύτιση για χρήση ανάλογα με τις ανάγκες. Η διαδικασία της αναζήτησης που περιγράφηκε παραπάνω παραμένει ίδια. Στην πρώτη παραλλαγή μετά το πέρας της αναζήτησης το πρόγραμμα τερματίζει. Στην δεύτερη το πρόγραμμα παραμένει σε λειτουργία διατηρώντας την ακολουθία

εισόδου στην μνήμη, και ζητώντας από τον χρήστη την εισαγωγή ονόματος αρχείου με άλλες ακολουθίες προς αναζήτηση, συνεχίζοντας τις αναζητήσεις μέχρι την εισαγωγή ενός χαρακτήρα για έξοδο. Τέλος, για γρήγορες αναζητήσεις, που βοήθησαν και σε ταχύτερο debug για τις υπόλοιπες υλοποιήσεις (για λόγους διασταύρωσης των αποτελεσμάτων) δημιουργήσαμε παραλλαγή όπου ο χρήστης εισάγει κατευθείαν από την γραμμή εντολών μια ακολουθία του αλφαβήτου {1,2,3,4} για άμεση αναζήτηση στο δένδρο.

4.2.2 Υλοποίηση προσομοίωσης στοίχησης

Όπως αναφέρθηκε οι σύνδεσμοι επιθέματος χρησιμοποιούνται σε πολλούς αλγορίθμους ακολουθιακών δεδομένων, και ένα από τα πλεονεκτήματα του TRELLIS είναι η δυνατότητα για γρήγορη ανάκτηση των συνδέσμων επιθέματος από τα κατασκευασμένα προθεματικά υποδένδρα. Υλοποιήσαμε λοιπόν μια ειδική αναζήτηση δεδομένων, για να επιβεβαιώσουμε την ταχύτητα προσπέλασης των κόμβων με την χρήση συνδέσμων επιθέματος. Η αναζήτηση αυτή βρίσκεται στον πυρήνα αλγορίθμων στοίχησης. Χρησιμοποιείται ως πρότυπο ένα κομμάτι της ακολουθίας εισόδου για να είναι σίγουρο ότι θα συμβεί επιτυχής ταύτιση. Ένα κυλιόμενο παράθυρο μικρότερου μεγέθους από το πρότυπο ορίζεται να ξεκινάει από κάποιο σημείο του προτύπου και πραγματοποιούνται διαδοχικές αναζητήσεις μέσα στο δένδρο, της περιοχής που ορίζεται από το παράθυρο με διαδοχική μετακύλιση του παραθύρου προς τα δεξιά κατά 1 θέση τη φορά.

Με αυτό τον τρόπο είναι εξασφαλισμένο ότι κάθε πρότυπο που αναζητείται περιέχει το πρώτο γνήσιο επίθεμα του προηγούμενου προτύπου, και έναν χαρακτήρα επιπλέον. Γίνεται αναζήτηση με δύο διαφορετικούς τρόπους. Στην πρώτη περίπτωση αναζητείται το πρότυπο με διάσχιση του δένδρου (αντίστοιχα με πριν) και στην δεύτερη περίπτωση η διάσχιση γίνεται με χρήση του συνδέσμου επιθέματος του κόμβου που βρίσκεται ακριβώς πάνω από την ακμή όπου ολοκληρώθηκε η τελευταία αναζήτηση. Αν η ακμή ολοκληρώθηκε πάνω στον εσωτερικό κόμβο χρησιμοποιείται ο σύνδεσμος επιθέματος του κόμβου αυτού. Προς αποφυγή παρανόησης πρέπει να διευκρινιστεί ότι η αναζήτηση θεωρείται ότι τελειώνει πάνω σε έναν κόμβο όταν συγκριθούν και βρεθούν όλοι οι χαρακτήρες της ακμής που προσπίπτει σε αυτόν. Αν έστω και ένας δεν συγκριθεί και η αναζήτηση ολοκληρωθεί, τότε ο σύνδεσμος επιθέματος πρέπει να αναζητηθεί στον κόμβο από τον οποίον εξέρχεται η ακμή, δηλαδή τον ακριβώς από πάνω. Όπως αναφέρθηκε πριν οι περισσότερες αναζητήσεις για πρότυπα σημαντικού μεγέθους ολοκληρώνονται σε φύλλο, εννοώντας ότι γίνεται προσπέλαση του φύλλου και σύγκριση των χαρακτήρων της ακμής που προσπίπτει σε φύλλο και όχι ότι συγκρίνονται όλοι οι χαρακτήρες του φύλλου μέχρι και το τέλος. Αν η αναζήτηση ολοκληρωθεί πάνω στο ίδιο το φύλλο σημαίνει ότι βρέθηκε ένα ολόκληρο επίθεμα άρα ο αντίστοιχος σύνδεσμος επιθέματος δεν υπάρχει αφού δεν ορίζεται για φύλλα. Μόλις

ανακτηθεί ο σύνδεσμος επιθέματος ανοιγεί το αντίστοιχο προθεματικό υποδένδρο και προσπελαύνεται η φυσική θέση στην οποία δείχνει ο σύνδεσμος που είναι μια θέση στο δένδρο κοντά στο πρότυπο. Ο χρόνος λοιπόν για την αναζήτηση με τον σύνδεσμο επιθέματος είναι μόνο το I/O κόστος για το ανέβασμα του δένδρου στην μνήμη, και μία επιπλέον σύγκριση προς τα κάτω αφού δεν είναι απαραίτητο να πραγματοποιηθούν συγκρίσεις και διάσχιση του δένδρου από την ρίζα.

Οι χρόνοι για την αναζήτηση με ευθύ τρόπο και την αναζήτηση μέσω συνδέσμου επιθέματος μετρούνται διαδοχικά, και αποθηκεύονται σε κάποιους μετρητές, και μετά από έναν ικανό αριθμό μετακυλίσεων του παραθύρου η αναζήτηση ολοκληρώνεται και τυπώνονται οι μέσες τιμές της ευθείας αναζήτησης και της αναζήτησης με χρήση του συνδέσμου επιθέματος ώστε να συγκριθεί η προσπέλαση κόμβου μετά και άνευ σύνδεσμων επιθέματος.

4.3 Υλοποιήσεις πάνω στα νέα προεπεξεργασμένα αρχεία

4.3.1 Υλοποίηση προσεγγιστικής ταύτισης με υβριδικό δυναμικό προγραμματισμό

Όπως περιγράφηκε στην 2.2.3.3 το πρόβλημα της προσεγγιστικής αναζήτησης κάποιου προτύπου μέσα σε κείμενο όπου το πλήθος των διαφορών μεταξύ των ταυτίσεων και του προτύπου είναι φραγμένο από κάποιον αριθμό k, μπορεί να επιλυθεί με χρήση υβριδικού δυναμικού προγραμματισμού σε χρόνο O(kn), και χώρο O(n+m), όπου n είναι το μήκος του κειμένου και m το μήκος του προτύπου. Ο αλγόριθμος προϋποθέτει την ύπαρξη δένδρου επιθεμάτων που έχει υποστεί προεπεξεργασία για την ανάκτηση σε σταθερό χρόνο της μέγιστης κοινής επέκτασης οποιωνδήποτε θέσεων της ακολουθίας.

Υλοποίησαμε λοιπόν τον αλγόριθμο των k-διαφορών με υβριδικό δυναμικό προγραμματισμό, κάνοντας χρήση των δένδρων που προέκυψαν από την προεπεξεργασία που περιγράφηκε στο 4.1.3. Ο αλγόριθμος σταθερού χρόνου προϋποθέτει την ύπαρξη ενός δένδρου επιθεμάτων που περιέχει και το κείμενο και το πρότυπο, επομένως πραγματοποιείται αναζήτηση ενός προτύπου που βρίσκεται ήδη σε κάποια θέση του προτύπου, και επιστρέφονται όλες οι θέσεις μέσα στο κείμενο όπου απαντά το πρότυπο με πλήθος διαφορών ως και k. Σε κάθε περίπτωση λοιπόν ο αλγόριθμος θα επιστρέψει μια 0-διαφορά (ακριβής ταύτιση) και όλες τις υπόλοιπες ταυτίσεις αν υπάρχουν.

Πριν την εφαρμογή του αλγορίθμου πρέπει να πραγματοποιηθεί η προεπεξεργασία των δένδρων με χρήση του προγράμματος tree_preprocess (βλέπε 4.1.3). Το πρόγραμμα για την προσεγγιστική ταύτιση δέχεται ως είσοδο τον φάκελο όπου βρίσκονται όλα τα αρχεία του

TRELLIS μετά την προεπεξεργασία, το εύρος μέσα στην ακολουθία εισόδου όπου βρίσκεται το πρότυπο και το φράγμα του πλήθους διαφορών.

Ο αλγόριθμος υλοποιεί πλήρως τον αλγόριθμο που περιγράφηκε στην 2.2.3.3. Δημιουργεί όλες τις n+m διαγωνίους του πίνακα δυναμικού προγραμματισμού και για κάθε μία από αυτές κρατάει την θέση όπου φθάνει το κάθε d-μονοπάτι. Κάθε φορά που ένα d-μονοπάτι φθάνει στην γραμμή m του πίνακα, υπάρχει ταύτιση με d διαφορές. Στην πρώτη επανάληψη για να βρεθούν τα απώτατα 0-μονοπάτια ο αλγόριθμος ανακτά με χρήση του αλγορίθμου σταθερού χρόνου όλες τις μέγιστες κοινές επεκτάσεις της αρχής του προτύπου για όλες τις θέσεις του κειμένου. Στην γραμμή m φθάνουν οι επεκτάσεις που αντιπροσωπεύουν όλες τις ακριβές ταυτίσεις μέσα στο κείμενο. Στην συνέχεια ανακτάται το απώτατο 1-μονοπάτι με χρήση των μονοπατιών R_1 ,R_2 και R_3 που προκύπτουν από τα 0-μονοπάτια των διαγωνίων i+1, i-1, i , για κάθε διαγώνιο i, με επέκταση της κατάλληλης ακμής μήκους 1 χαρακτήρα και τους υπολογισμούς των μέγιστων κοινών επεκτάσεων στις αντίστοιχες θέσεις προτύπου και κειμένου. Η διαδικασία επαναλαμβάνεται k φορές.

Κατά την εκτέλεση του αλγορίθμου για το απώτατο 0-μονοπάτι η θέση μέσα στο πρότυπο για την οποία γίνονται τα ερωτήματα της μέγιστης κοινής επέκτασης είναι ίδια, ο πρώτος χαρακτήρας του προτύπου. Επομένως για την ανάκτηση του LCE είναι απαραίτητο να διατηρείται το πολύ ένα προθεματικό δένδρο στην μνήμη, αυτό που αντιστοιχεί στο πρόθεμα του προτύπου. Για τους πρώτους χαρακτήρες η αναζήτηση γίνεται στο ίδιο το κείμενο του προθέματος μέσα στο trie και αν η επέκταση συνεχίζεται και μετά το κείμενο του προθέματος, γίνεται ένα LCE ερώτημα πάνω στο δένδρο και επιστρέφεται σε σταθερό χρόνο το μήκος της μέγιστης επέκτασης. Για τα υπόλοιπα απώτατα d-μονοπάτια οι θέσεις μέσα σε πρότυπο και κείμενο όπου αναζητείται η μέγιστη κοινή επέκταση είναι γενικά τυχαίες, και για την χρήση του αλγορίθμου σταθερού χρόνου θα πρέπει να φορτώνεται στην μνήμη το υποδένδρο που ορίζεται από το κοινό πρόθεμα των θέσεων του προτύπου και του κειμένου. Σε κάθε περίπτωση όμως, για τους πρώτους χαρακτήρες των επεκτάσεων η αναζήτηση θα πρέπει να γίνει μέσα στο trie των προθεμάτων με γραμμικό τρόπο. Υπάρχει λοιπόν μια μικρή νόθευση του αποτελέσματος για τον σταθερό χρόνο, αφού πλέον η πολυπλοκότητα για την εύρεση της μέγιστης κοινής επέκτασης είναι O(μ) όπου μ το μέγιστο μήκος προθέματος.

Η ανάκτηση του LCE μέσα στα δένδρα προϋποθέτει την χρήση του πίνακα των επιθεμάτων που είναι αποθηκευμένος στο αρχείο suffix.dat, και ο οποίος για τις ανάγκες του αλγορίθμου φορτώνεται στο σύνολο του στην μνήμη. Τα επιθέματα που ορίζουν οι θέσεις μέσα στο πρότυπο και το κείμενο για τις οποίες αναζητείται το LCE (και είναι οι θέσεις που έχουν προκύψει από τον πίνακα του δυναμικού προγραμματισμού μετατοπισμένες κατά το μήκος του προθέματος του εκάστοτε υποδένδρου) αντιστοιχίζονται σε κόμβους μέσα στα προεπεξεργασμένα δένδρα με χρήση του πίνακα των επιθεμάτων. Στην συνέχεια βρίσκεται ο

LCA των κόμβων αυτών πραγματοποιώντας όλα τα βήματα του αλγορίθμου που περιγράφεται στην 2.2.2.3. Ο αλγόριθμος περιλαμβάνει μόνο δυαδικές πράξεις και ανακτήσεις δεδομένων από την μνήμη επομένως πραγματοποείται σε σταθερό χρόνο. Το πεδίο <μήκος_μονοπατιού> του LCA κόμβου επιστρέφεται ως η τιμή της μέγιστης επέκτασης. Το συνολικό LCE είναι το άθροισμα του μήκους του μονοπατιού που επιστρέφει ο αλγόριθμος για το LCA και του μήκους του προθέματος.

Σε κάθε επανάληψη επιστρέφονται ως επιτυχείς ταυτίσεις η κάθε στήλη όπου κάποιο απώτατο d-μονοπάτι φθάνει στην γραμμή m. Η στήλη αυτή αντιπροσωπεύει την τελική θέση όπου απαντάται το πρότυπο μέσα στο κείμενο με d διαφορές. Η υλοποίηση μας κρατά στην μνήμη μόνο τα τελευταία μονοπάτια d που προκύπτουν σε κάθε επανάληψη άρα η πολυπλοκότητα χώρου είναι O(n+m). Αν είναι επιθυμητό να κρατούνται τα μονοπάτια που έχουν προκύψει από τις προηγούμενες επαναλήψεις για να βρεθούν οι αρχικές θέσεις των ταυτίσεων και η στοίχηση του προτύπου με το κείμενο στις θέσεις της ταύτισης, ο αλγόριθμος πρέπει να χρησιμοποιήσει χώρο O[k(m+n)], το οποίο για μεγάλα k είναι προβληματικό για τις παραδοχές που έχουν γίνει από τον TRELLIS για το κατώφλι και το μέγεθος της μνήμης (βλέπε 3.5.5 και 4.1.3).

4.3.2 Υλοποίηση ερωτήματος καθολικής εύρεσης της μέγιστης κοινής επέκτασης

Όπως αναλύθηκε στην 2.2.2.1 η εύρεση της μέγιστης κοινής επέκτασης δύο ακολουθιών είναι κεντρικό πρόβλημα για την αποδοτική επίλυση πολλών προβλημάτων για ακολουθιακά δεδομένα και χρήσιμο για την στατιστική ανάλυση βιολογικών δεδομένων. Ορίζεται το πρόβλημα της καθολικής εύρεσης του LCE με κάτω φράγμα, δηλαδή της εύρεσης του μήκους του LCE μιας θέσης της ακολουθίας με όλες τις άλλες θέσεις καθόλο το μήκος της ακολουθίας όπου το μήκος των μεγίστων κοινών επεκτάσεων που μας αφορούν είναι μεγαλύτερο από κάποιο φράγμα. Το πρόβλημα αυτό όσον αφορά βιολογικά δεδομένα έχει νόημα όταν οι πρώτοι χαρακτήρες κάποιου σημείου της ακολουθίας έχουν ένα συγκεκριμένο ειδικό βάρος, π.χ. κάποιο συγκεκριμένο ρυθμιστικό γονίδιο, ή μια αρχική αλυσίδα αμινοξέων που απαντά σε πολλές πρωτεϊνες. Μας αφορούν δηλαδή όλες οι άλλες θέσεις σε όλο το μήκος της βιολογικής ακολουθίας που απαντά κάποιο πρώτο κομμάτι της ακολουθίας που ορίζεται από την θέση, και οι μέγιστες επεκτάσεις αυτών.

Υλοποίησαμε λοιπόν ένα πρόγραμμα για την καθολική ανάκτηση των μέγιστων κοινών επεκτάσεων με κάτω φράγμα. Το πλεονέκτημα που διαθέτουμε για την απάντηση ενός τέτοιου ερωτήματος μετά την προεπεξεργασία των δένδρων, είναι ότι για ένα κάτω φράγμα που ξεπερνάει το μήκος του προθέματος της θέσης της ακολουθίας στην οποία εκτελείται ο αλγόριθμος, όλες οι αναζητήσεις για την μέγιστη κοινή επέκταση της θέσης αυτής και όλων των άλλων θέσεων της ακολουθίας μπορούν να γίνουν σε ένα προεπεξεργασμένο

προθεματικό υποδένδρο που παραμένει στην μνήμη. Αφού το μήκος των μεγίστων κοινών επεκτάσεων που μας αφορά ξεπερνά το μήκος του προθέματος η αναζήτηση για όλες τις άλλες θέσεις που είναι αποθηκευμένες σε άλλα προθεματικά υποδένδρα είναι περιττές αφού θα είναι σε κάθε περίπτωση μικρότερες του φράγματος. Η ανάκτηση του LCE θα γίνει για όλα τα φύλλα του δένδρου και θα επιστραφούν οι τιμές που είναι μεγαλύτερες από το φράγμα. Αφού η ανάκτηση του LCE για δύο κόμβους γίνεται σε χρόνο O(1) η ανάκτηση για τα t φύλλα γίνεται σε χρόνο O(t). Χωρίς χρήση ευρετηρίου η ανάκτηση των LCE μπορεί να γίνει σε χρόνο ανάλογο του μεγέθους της πλήρους ακολουθίας.

Το πρόγραμμα δέχεται ως είσοδο τον φάκελο όπου βρίσκονται τα δένδρα του TRELLIS μετά την προεπεξεργασία με το πρόγραμμα tree_preprocess, την θέση μέσα στην βιολογική ακολουθία για την οποία θα αναζητηθεί καθολικά η μέγιστη κοινή επέκταση, και το κάτω φράγμα. Φορτώνεται το trie των προθεμάτων και αναζητείται το πρόθεμα της θέσης εισόδου. Αν το μήκος του προθέματος είναι μικρότερο από το κάτω φράγμα τότε δεν είναι απαραίτητη η φόρτωση στην μνήμη όλης της ακολουθίας εισόδου. Φορτώνεται μόνο το προεπεξεργασμένο προθεματικό υποδένδρο, και με χρήση του αλγορίθμου σταθερού χρόνο γίνεται ανάκτηση του μήκους των μεγίστων κοινών επεκτάσεων για το φύλλο που αντιστοιχεί στην θέση είσοδου και όλα τα άλλα φύλλα του δένδρου σε χρόνο O(t). Ο αλγόριθμος τυπώνει τα αποτελέσματα που είναι μεγαλύτερα από το κάτω φράγμα και τερματίζει. Αν το μήκος του προθέματος είναι μεγαλύτερο από το κάτω φράγμα τότε η αναζήτηση γίνεται σε όλο το μήκος της ακολουθίας σε χρόνο ανάλογο του μήκους της ακολουθίας. Η χρονική πολυπλοκότητα O(t) ισχύει επομένως όταν το κάτω φράγμα είναι μεγαλύτερο από το μήκος του προθέματος της θέσης εισόδου, το οποίο στην γενική περίπτωση είναι αρκετά μικρό (κάτω από 10 σύμβολα).

4.4 Περιγραφή χρήσης της πλατφόρμας ερωτημάτων

Το compilation όλων των προγραμμάτων για την πλατφόρμα του TRELLIS μαζί με τα δικά μας προγράμματα γίνεται με την εκτέλεση από το τερματικό στον φάκελο όπου βρίσκονται τα αρχεία πηγαίου κώδικα της εντολής:

./make

Μόλις ολοκληρωθεί το compilation όλων των προγραμμάτων στον φάκελο υπάρχουν τα εκτελέσιμα αρχεία. Θα ήταν χρήσιμο ο φάκελος όπου βρίσκονται τα αρχεία να μπεί στο command path του συστήματος ώστε τα προγράμματα να μπορούν να εκτελεστούν ως εντολές από οποιονδήποτε φάκελο. Αν αυτό δεν γίνει για την εκτέλεση κάθε προγράμματος θα πρέπει πριν το όνομα του του να γράφουμε και τον φάκελο όπου βρίσκεται. Παρακάτω γίνεται η υπόθεση ότι τα προγράμματα βρίσκονται στο command path του συστήματος.

Τα αρχεία με τα βιολογικά δεδομένα όπως λαμβάνονται από την δημόσια βάση του NCBI είναι σε μορφή FASTA. Το TRELLIS παρέχει ένα πρόγραμμα για την μετατροπή των αρχείων FASTA που αναπαριστούν ακολουθίες DNA με το αλφάβητο {A,C,G,T} σε αρχεία NUM με το αλφάβητο {1,2,3,4} για την δυνατότητα επεξεργασίας από το TRELLIS. Το πρόγραμμα αυτό είναι το f2n (Fasta2Num). Έτσι αν στο αρχείο chr1.fa βρίσκεται αποθηκευμένη σε FASTA μορφή η γενετική ακολουθία του πρώτου ανθρώπινου χρωμοσώματος, με την εντολή:

f2n chr1.fa chr1_1MB.num n 1000000

δημιουργείται ένα αρχείο NUM που περιέχει το πρώτο 1MB του chr1_1MB.num. Η παράμετρος n ορίζει η έξοδος να είναι μορφής num και η τελευταία παράμετρος ορίζει το μέγεθος του αρχείου. Μπορούν έτσι να δημιουργηθούν διάφορα μεγέθη αρχείων. Αν η τελευταία παράμετρος οριστεί 0 τότε το παραγόμενο αρχείο NUM έχει το πλήρες μήκος του αρχικού αρχείου FASTA.

Στην συνέχεια η NUM ακολουθία μπορεί να χρησιμοποιηθεί από το TRELLIS για την παραγωγή ευρετηρίου. Για την δυνατότητα μετέπειτα χρήσης των προγραμμάτων χρησιμοποιώντας τον ελάχιστο αριθμό παραμέτρων φτιάχνουμε έναν νέο υποφάκελο με την ονομασία chr1_1MB και μεταφέρουμε το αρχείο NUM στον φάκελο αυτόν εκτελώντας διαδοχικά τις εντολές:

mkdir chr1_1MB

mv chr1_1MB.num chr1_1MB/

Το πρόγραμμα για την παραγωγή του ευρετηρίου δέχεται τρεις παραμέτρους:

- Το όνομα του αρχείου εισόδου. Το αρχείο εισόδου πρέπει να είναι σε μορφή NUM.
- Το μέγεθος της διαθέσιμης μνήμης σε KB για τον υπολογισμό του μέγεθος του κατωφλίου (βλέπε 3.5.5)
- Το εκτιμώμενο μέγιστο μήκος προθέματος για την επιτάχυνση της Φάσης Ανάκτησης των Προθεμάτων (βλέπε 3.5.6.1). Αν δεν μπορεί να χρησιμοποιηθεί μια καλή εκτίμηση, ο TRELLIS θα εκτελεστεί σωστά, με μία μικρή επιβάρυνση στο χρόνο για την ανάκτηση των προθεμάτων.

Η εντολή για την δημιουργία ευρετηρίου για την ακολουθία chr1_1MB.num είναι λοιπόν:

trellis –i chr1_1MB.num –m 8500 –l 3

Το μέγεθος της εισόδου και οι αντίστοιχες τιμές των παραμέτρων είναι αρκετά μικρές ώστε η κατασκευή να γίνει σε σύντομο χρονικό διάστημα. Το μέγεθος της μνήμης είναι αρκετά μικρό ώστε ακόμα και για αυτήν την μικρή ακολουθία να δημιουργούνται αρκετά προθεματικά υποδένδρα. Σκοπός του παραδείγματος είναι η πρώτη εξοικείωση με την πλατφόρμα. Αν ζητούμενο είναι η παραγωγή ευρετηρίων για πολύ μεγάλες ακολουθίες

πρέπει να χρησιμοποιηθεί η κατάλληλη ακολουθία εισόδου με τις αντίστοιχες προσαρμογές για το μεγέθος της διαθέσιμης μνήμης και του μήκους των προθεμάτων.

Μετά την παραγωγή των ευρετηρίων μπορούν να ανακτηθούν οι σύνδεσμοι επιθέματος με χρήση της εντολής:

addSFLinks .

Αν ο χρήστης βρίσκεται σε άλλον φάκελο πρέπει να προσδιορίσει τον φάκελο όπου βρίσκονται τα ευρετήρια και η ακολουθία εισόδου (που πρέπει να έχει το ίδιο όνομα με το όνομα του φακέλου):

addSFLinks ~/chr1_1MB

Μετά την εκτέλεση αυτών των προγραμμάτων μέσα στον φάκελο chr1_1MB υπάρχει:

- ο φάκελος tempBinData με τα αρχεία εσωτερικών κόμβων
- ο φάκελος tempBinDataL με τα αρχεία εσωτερικών κόμβων, όπου για κάθε κόμβο υπάρχει και ο σύνδεσμος επιθέματος του
- ο φάκελος binData που περιέχει τα αρχεία κόμβων-φύλλων
- Τα αρχεία prefix.txt και prefix_trie
- Το αρχείο threshold.txt που περιέχει το κατώφλι
- Η ακολουθία εισόδου (chr1_1MB.num)

Για την δυνατότητα ερωτημάτων ακριβούς ταύτισης πρέπει να διατίθενται αρχεία τύπου NUM που να περιέχουν το πρότυπο. Δημιουργήσαμε ένα μικρό πρόγραμμα με το όνομα generate_query_file που παίρνει από την ακολουθία εισόδου ένα κομμάτι με το επιθυμητό μήκος ξεκινώντας από μια συγκεκριμένη θέση της ακολουθίας εισόδου. Αν εκτελεστεί η εντολή:

generate_query_file chr1_1MB.num query.num 100 300000

θα δημιουργηθεί ένα αρχείο NUM με όνομα query.num με μέγεθος 100 σύμβολα, ξεκινώντας από την θέση 300000 μέσα στην ακολουθία εισόδου chr1_1MB.num. Αν παραληφθεί η τελευταία παράμετρος τότε το πρόγραμμα θα επιλέξει τυχαία κάποια θέση μέσα στην ακολουθία η οποία θα εκτυπωθεί.

Στην συνέχεια με το πρόγραμμα exact μπορούμε να εκτελέσουμε ερωτήματα ακριβούς ταύτισης στα δένδρα. Η εντολή:

exact .

αναζητά το πρότυπο που περιέχεται μέσα στο αρχείο query.num του τρέχοντος φακέλου με χρήση των δένδρων επιθεμάτων και εκτυπώνει εξαντλητικά όλες τις θέσεις όπου απαντάται καθώς και τον χρόνο που χρειάστηκε για την ταύτιση. Στον φάκελο πρέπει να έχει ολοκληρωθεί η δημιουργία του πλήρους ευρετηρίου εκτός των συνδέσμων επιθέματος που

δεν είναι απαραίτητη για την ακριβή ταύτιση. Αν ο χρήστης βρίσκεται σε άλλο φάκελο ο φάκελος όπου βρίσκονται τα δένδρα πρέπει να προσδιοριστεί:

exact ~/chr1_1MB

Το πρότυπο by default θα αναζητηθεί στο αρχείο query.num.Αν το πρότυπο δεν βρίσκεται σε αρχείο με όνομα query.num αλλά σε αρχείο με άλλο όνομα, το όνομα πρέπει να προσδιοριστεί με δεύτερη παράμετρο. Π.χ.:

exact ~/chr1_1MB query2.num

Οι αναζητήσεις με χρήση των συνδέσμων επιθέματος (βλέπε 4.2.1) γίνονται με χρήση του προγράμματος TestLinks. Η σύνταξη της εντολής είναι

TestLinks <φάκελος_ευρετηρίου> <δείκτης_έναρξης> <μεγεθος_ερωτήματος> <αριθμός_ερωτημάτων>

Για παράδειγμα η εντολή:

TestLinks ~/chr1_1MB 200000 200 20

πραγματοποιεί 20 ερωτήματα μεγέθους 200 συμβόλων ξεκινώντας από την θέση 200000 της ακολουθίας εισόδου με και χωρίς χρήση συνδέσμων επιθεμάτων και τυπώνει τους χρόνους που χρειάζονται σε κάθε περίπτωση. Για την εκτέλεση αυτού του προγράμματος πρέπει προηγουμένως να έχουν ανακτηθεί οι σύνδεσμοι επιθέματος με την εντολή addSFLinks.

Για την γραμμική προεπεξεργασία των δένδρων εκτελείται το πρόγραμμα process με σύνταξη:

preprocess <φάκελος_ευρετηρίου>

Έτσι, η εντολή:

preprocess chr1_1MB

προεπεξεργάζεται τα δένδρα του παραδείγματος. Μετά την προεπεξεργασία στον φάκελο υπάρχει το νέο αρχείο suffix.dat και στον υποφάκελο binData βρίσκονται τα νέα αρχεία L_*_lca.dat μαζί με τα παλαιά αρχεία των φύλλων.

Μετά την προεπεξεργασία μπορούν να πραγματοποιηθούν ερωτήματα προσεγγιστικής ταύτισης με χρήση υβριδικού προγραμματισμού με χρήση του προγράμματος hybrid.Το πρόγραμμα έχει την παρακάτω σύνταξη:

hybrid <φάκελος_ευρετηρίου> <δείκτης_αρχής> <δείκτης_τέλους> <μέγιστο_πλήθος_διαφορών>

Επομένως η εντολή :

hybrid chr1_1MB 400000 400020 3

αναζητά στην ακολουθία εισόδου όλες τις εμφανίσεις της υποακολουθίας [400000..400020] με πλήθος διαφορών το πολύ 3, και τυπώνει όλες τις τελικές θέσεις των ταυτίσεων, και το

πλήθος διαφορών τους. Σε κάθε περίπτωση θα τυπωθεί μια ακριβής ταύτιση (0 διαφορές) στην τελική θέση 400020 και όσες άλλες ταυτίσεις υπάρχουν στην ακολουθία.

Ερωτήματα καθολικής εύρεσης LCE με κάτω φράγμα πραγματοποιούνται με χρήση του προγράμματος global_lce. Η σύνταξη είναι:

global_lce <φάκελος_ευρετηρίου> <δείκτης_αρχής> <κάτω_φράγμα>

Επομένως η εντολή:

global_lce chr1_1MB 500000 8

επιστρέφει όλες τις θέσεις στην ακολουθία εισόδου των οποίων η μέγιστη κοινή επέκταση με την θέση 500000 της ακολουθίας είναι μεγαλύτερη από 8, και για κάθε θέση επιστρέφει και το μήκος της μέγιστης επέκτασης.

5 Αξιολόγηση

5.1 Γενική περιγραφή των πειραμάτων

Τα πειράματα για τις υλοποιήσεις που περιγράφηκαν στο κεφάλαιο 4 πραγματοποιήθηκαν σε ένα μηχάνημα 3 Ghz Intel Core 2 Duo CPU με 4GB RAM σε λειτουργικό Debian Linux. Οι διαθέσιμοι κώδικες του TRELLIS καθώς και οι κώδικες των δικών μας υλοποιήσεων είναι γραμμένοι σε C++, και μεταγλωττίστηκαν με τον GNU g++ compiler έκδοση 4.3.3. Η μνήμη που χρησιμοποιήθηκε για τα πειράματα δεν ήταν όλη η διαθέσιμη μνήμη αφού το TRELLIS ορίζει την διαθέσιμη μνήμη παραμετρικά. Η μνήμη ορίστηκε παραμετρικά στα 100MB έτσι ώστε η ακολουθία 20Mbps να έχει ως ευρετήρια λιγότερα από 30 προθεματικά υποδένδρα, και η ακολουθία 200Mbps περισσότερα από 200. Για την ευρετηριοποίηση ολόκληρου του ανθρώπινου γονιδιώματος καλό είναι να υπάρχει μια διαθέσιμη μνήμη τουλάχιστον 2GB ώστε το πλήθος των προθεματικών υποδένδρων να μην εκτινάσσεται. Οι βιολογικές ακολουθίες που χρησιμοποιήθηκαν στα πειράματα είναι κομμάτια του πρώτου ανθρώπινου χρωμοσώματος που είναι διαθέσιμο, μαζί με όλο το υπόλοιπο ανθρώπινο γονιδίωμα, στην δημόσια βάση δεδομένων του Εθνικού Κέντρου Πληροφοριών Βιοτεχνολογίας των Ηνωμένων Πολιτειών (NCBI).

Κατά την εκτέλεση των πειραμάτων χρησιμοποιήσαμε για την χρονομέτρηση τις μεθόδους της κλάσης TimeTracker που διατίθεται από το TRELLIS. Για να μην επηρεάζεται η χρονομέτρηση από λειτουργίες εκτύπωσης των αποτελεσμάτων στην οθόνη εισάγαμε τα αποτελέσματα κατά την εκτέλεση σε ένα αντικείμενο τύπου λίστας της στανταρ βιβλιοθήκης προτύπων της C++, και τυπώναμε τα αποτελέσματα στην οθόνη μετά την λήξη της χρονομέτρησης. Τα αποτελέσματα που είναι σε διαγραμματική μορφή παρήχθησαν με το εργαλείο gnuplot.

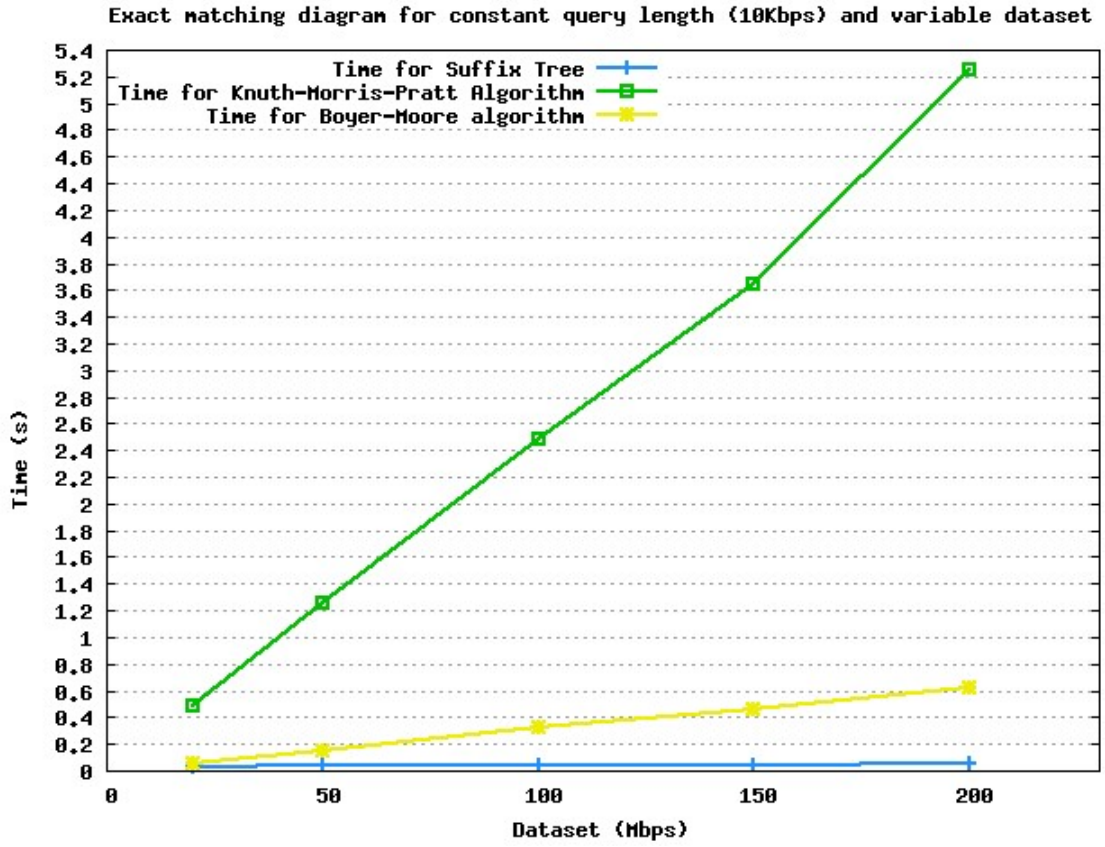
Για την πραγματοποίηση των πειραμάτων παραλλάξαμε τις υλοποιήσεις ώστε να πραγματοποιείται ένας ικανός αριθμός αναζητήσεων (περίπου 20) σε τυχαίες θέσεις της ακολουθίας εισόδου και να εκτυπώνεται η μέση τιμή του χρόνου για την πραγματοποίηση τους.

Όπως αναφέρθηκε, η ακολουθία εισόδου σε κάθε περίπτωση χωράει στην μνήμη, και θεωρούμε ότι παραμένει μονίμως φορτωμένη. Το ευρετήριο δεν χωράει στην μνήμη επομένως η χρονομέτρηση περιλαμβάνει και το I/O κόστος για την φόρτωση από τον δίσκο του αντίστοιχου προθεματικού υποδένδρου.

5.2 Πειράματα για ακριβή ταύτιση προτύπου

Η πρώτη σειρά πειραμάτων έχει ως σκοπό την μελέτη της συμπεριφοράς των δένδρων επιθεμάτων πολύ μεγάλου μεγέθους για την ακριβή ταύτιση προτύπου, που περιγράφηκε στην 4.2.1. Οι ακολουθίες που αναζητώνται είναι υπάρχουσες υποακολουθίες της βιολογικής ακολουθίας στην οποία κατασκευάζεται το δένδρο και άρα οι αναζητήσεις καταλήγουν πάντα σε επιτυχία. Ως εκ τούτου οι χρόνοι αναζήτησης αντιπροσωπεύουν το άνω όριο για κάθε μέγεθος προτύπου, αφού στην περίπτωση που δεν έχουμε επιτυχή ταύτιση η αναζήτηση μπορεί να σταματήσει ακόμα στην πρώτη σύγκριση.

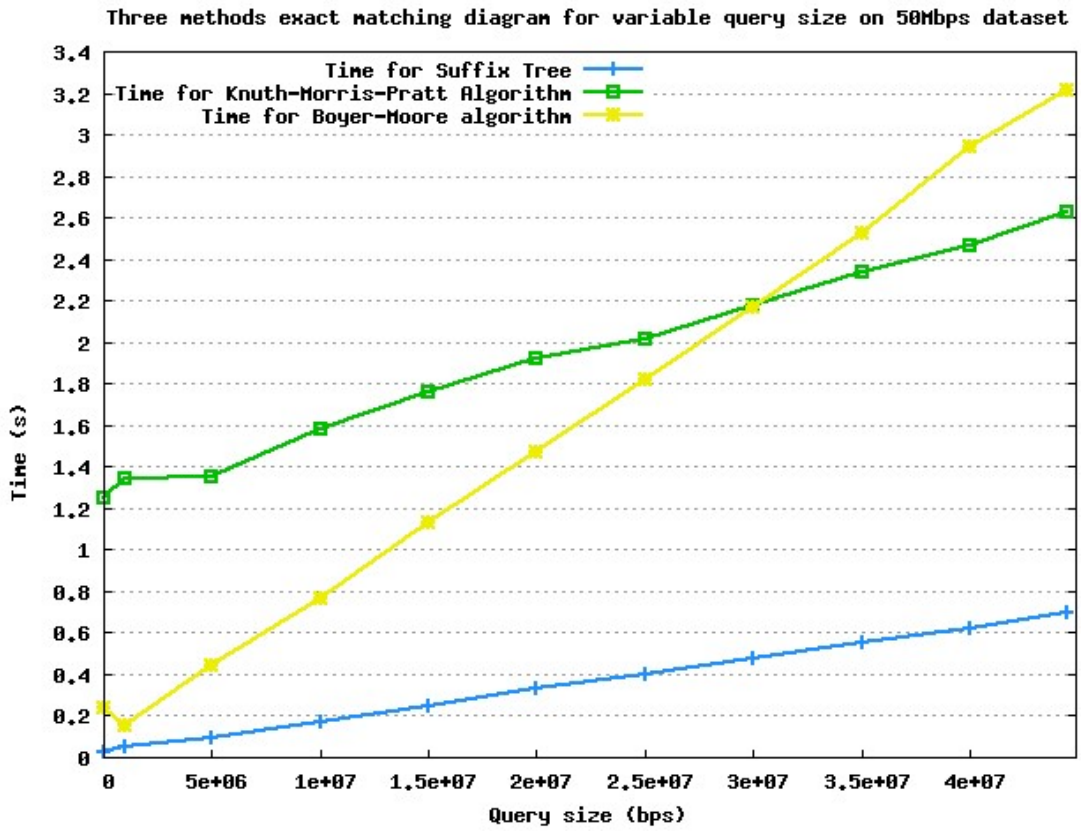
Στον πρώτο κύκλο των πειραμάτων πραγματοποιήσαμε αναζητήσεις για ένα αντιπροσωπευτικό μήκος προτύπου 10Kbps (K base pairs) σε δένδρα επιθεμάτων ακολουθιών 20, 50, 100, 150 και 200 Kbps. Οι αναζητήσεις πραγματοποιήθηκαν με τρεις τρόπους: με χρήση του ευρετηρίου των δένδρων επιθεμάτων του TRELLIS, με χρήση του αλγορίθμου Knuth-Morris-Pratt και με χρήση του αλγορίθμου Boyer-Moore. Στα διαγράμματα που ακολουθούν παριστάνονται οι προηγούμενες μετρήσεις.



Η ποιοτική μορφή του διαγράμματος είναι η αναμενόμενη. Παρατηρούμε ότι η αναζήτηση με χρήση του δένδρου επιθεμάτων είναι ταχύτερη και από τις δύο άλλες μεθόδους, ενώ ο χρόνος αναζήτησης με το δένδρο επιθεμάτων παραμένει σταθερός όσο αυξάνεται το μήκος της ακολουθίας, σε αντίθεση με τους αλγορίθμους χωρίς προεπεξεργασία που έχουν χρόνο αναζήτησης ανάλογο του μήκους της ακολουθίας.

Από τους δύο αλγορίθμους χωρίς χρήση ευρετηρίου γίνεται φανερό ότι ο αλγόριθμος Boyer-Moore έχει την καλύτερη απόδοση ενώ για την ακολουθία των 50Mbps η απόδοση του είναι σχεδόν ίδια με την απόδοση της αναζήτησης με το δένδρο επιθεμάτων για αυτό το μέγεθος προτύπου. Καθώς αυξάνεται το μέγεθος της ακολουθίας, η ψαλίδα μεγαλώνει και η αναζήτηση με το ευρετήριο είναι πολύ ταχύτερη των άλλων μεθόδων.

Στον δεύτερο κύκλο πειραμάτων κάναμε αναζητήσεις προτύπων με τις 3 μεθόδους για αυξανόμενο μήκος προτύπου σε μια ακολουθία μήκους 50Mbps.



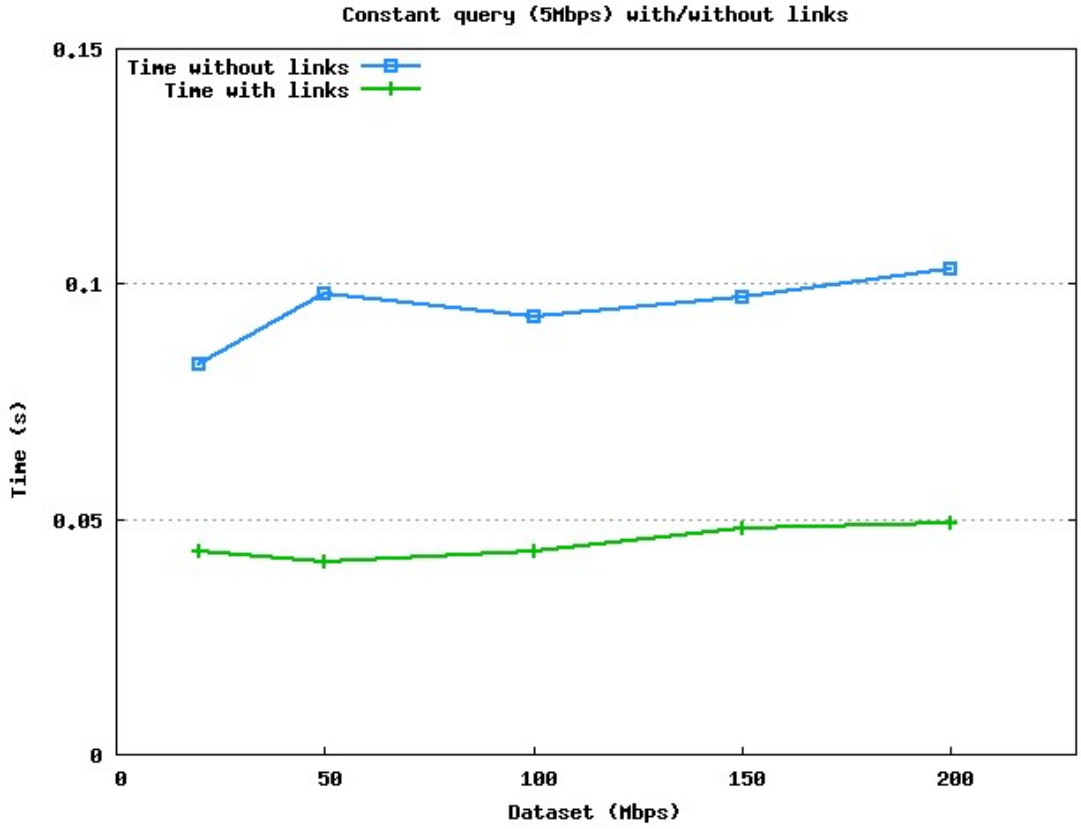
Τα αποτελέσματα είναι και σε αυτή την περίπτωση τα αναμενόμενα. Η αναζήτηση με το δένδρο επιθεμάτων είναι σε όλες τις περιπτώσεις ταχύτερη και η αναζήτηση είναι ανάλογη του μήκους του προτύπου. Η μέθοδος Boyer-Moore έχει καλύτερη συμπεριφορά για μικρά μήκη προτύπου αλλά γίνεται όλο και βραδύτερη καθώς το μήκος του προτύπου μεγαλώνει. Η μέθοδος Knuth-Morris-Pratt ξεκινάει με μεγάλους χρόνους ήδη για αρκετά μικρά πρότυπα, ωστόσο ο χρόνος μεταβάλλεται πιο ομαλά καθώς μεγαλώνει το μήκος του προτύπου σε σχέση με τον Boyer-Moore, και για μεγέθη προτύπου που φθάνουν στην τάξη του μεγέθους της ακολουθίας ο Knuth-Morris-Pratt είναι ταχύτερος από τον Boyer-Moore

5.3 Πειράματα για την αναζήτηση με χρήση συνδέσμων επιθέματος για προσομοίωση στοίχησης

Στη συνέχεια, πραγματοποίησαμε πειράματα για την υλοποίηση της αναζήτησης με χρήση συνδέσμων επιθέματος που προσομοιώνει την λειτουργία που βρίσκεται στον πυρήνα αλγορίθμων στοίχησης όπως περιγράφηκε στην 4.2.2. Για αυτή την αναζήτηση χρησιμοποιούνται πρότυπα ίδιου μεγέθους που λαμβάνονται από ένα κομμάτι της ακολουθίας εισόδου ώστε να υπάρχει πάντα επιτυχής ταύτιση. Από αυτό το κομμάτι της ακολουθίας εισόδου, λαμβάνονται πρότυπα με χρήση ενός κυλιόμενου παραθύρου που μετακινείται ένα βήμα τη φορά ώστε κατά τις διαδοχικές αναζητήσεις, κάθε επόμενο πρότυπο έχει ως πρόθεμα

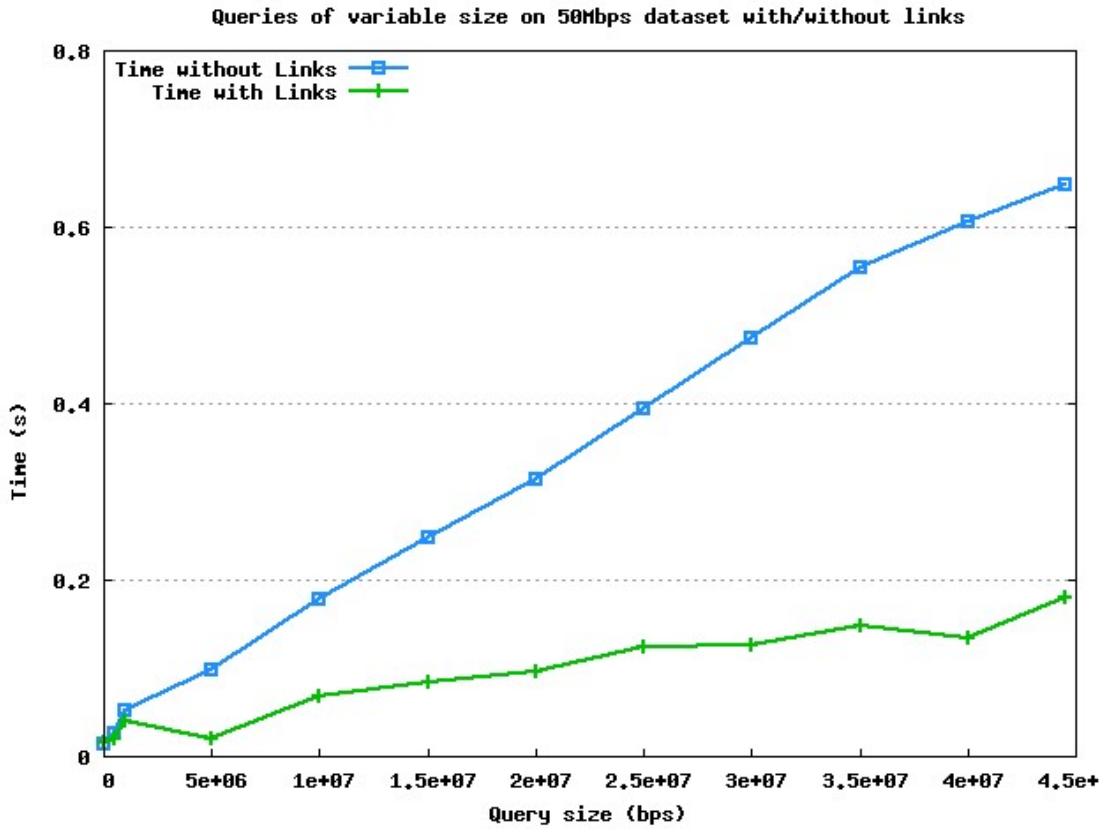
το πρώτο γνήσιο επίθεμα του προηγούμενου προτύπου. Η χρήση των συνδέσμων επιθέματος για τέτοιες αναζητήσεις είναι κομβική για την αποδοτική υλοποίηση αλγορίθμων. Τέτοιου τύπου ερωτήματα είναι πολύ πρακτικά και εμφανίζονται σε πολλές πραγματικές εφαρμογές όπως στον αλγόριθμο της γονιδιωματικής στοίχησης με δικλείδες ταύτισης (genome alignment via matching anchors), και στην αναζήτηση διδύμων επαναλήψεων (tandem repeats search).

Αντίστοιχα με τα προηγούμενα στον πρώτο κύκλο πειραμάτων χρησιμοποιήσαμε σταθερό μήκος προτύπου και μεταβαλλόμενο μήκος ακολουθίας. Το μήκος προτύπου που επιλέγχθηκε είναι αρκετά μεγαλύτερο από πριν (5Mbps) αφού για μικρά μήκη προτύπου η διαφορά είναι ανεπαίσθητη καθώς ο αλγόριθμος έχει έναν σταθερό χρόνο για να ανεβάσει τα προθεματικά υποδένδρα στην μνήμη που γενικά είναι αρκετά μεγαλύτερος από τον χρόνο για να πραγματοποιήσει συγκρίσεις στην μνήμη, και για μικρό αριθμό συγκρίσεων η διαφορά στους χρόνους είναι πολύ μικρή.



Στο διάγραμμα φαίνεται ότι ο χρόνος αναζήτησης παραμένει στην πράξη σταθερός όσο αυξάνεται το μήκος της ακολουθίας. Όπως αναμενόταν, η αναζήτηση με χρήση συνδέσμων επιθέματος είναι αρκετά ταχύτερη.

Στον δεύτερο κύκλο πειραμάτων κάναμε αναζητήσεις προτύπων μεταβλητού μήκους σε δένδρο επιθεμάτων που έχει κατασκευαστεί σε σταθερού μήκους ακολουθία με χρήση συνδέσμων επιθεμάτων και χωρίς. Τα αποτελέσματα φαίνονται στο επόμενο διάγραμμα.



Η αναζήτηση χωρίς την χρήση συνδέσμων επιθέματος παρουσιάζει γραμμικότητα ως προς το μήκος του προτύπου, ενώ η αναζήτηση με χρήση των συνδέσμων κινείται γενικά ανοδικά παραμένοντας όμως πολύ χαμηλά.

5.4 Πειράματα για την υλοποίηση του υβριδικού δυναμικού προγραμματισμού

Στο σημείο αυτό, πραγματοποιήσαμε πειράματα για την υλοποίηση του αλγορίθμου υβριδικού δυναμικού προγραμματισμού. Τα δένδρα που χρησιμοποιούμε έχουν προεπεξεργαστεί κατά τον τρόπο που περιγράφεται στην 4.3.1 προκειμένου να έχουμε σταθερού χρόνου απάντηση των ερωτημάτων LCE, κάτι που είναι προαπαιτούμενο για να λειτουργεί αποδοτικά ο συγκεκριμένος αλγόριθμος. Σαν baseline μέθοδο για σύγκριση των αποτελεσμάτων χρησιμοποιήσαμε την μέθοδο δυναμικού προγραμματισμού για την προσεγγιστική εύρεση προτύπου, που πρότεινε ο Ukkonen το 1985 (cut-off heuristic) [Ukk85] που μειώνει τον αριθμό των υπολογισμών που γίνονται στα στοιχεία του πίνακα του

δυναμικού προγραμματισμού. Για τον δυναμικό προγραμματισμού πραγματοποιήσαμε τους υπολογισμούς στήλη-στήλη μειώνοντας με αυτόν τον τρόπο την απαίτηση για μνήμη σε O(m+n) κάτι που επιτρέπει την εφαρμογή του αλγορίθμου χωρίς παραβίαση των παραδοχών που έχουν γίνει για το μέγεθος των δεδομένων που χωρούν στην μνήμη. Το μήκος του προτύπου επιλέχθηκε να είναι 40 bytes, και το πλήθος των διαφορών 3.

Τα αποτελέσματα των μετρήσεων φαίνονται στον πίνακα

Μήκος ακολουθίας (Mbps)	Ukkonen 85 (sec)	Υβριδικός (sec)
1	0,3	38
2	0,6	48
5	1,5	103
10	3,0	198

Τα αποτελέσματα αυτά είναι βεβαίως αποκαρδιωτικά. Παρά την θεωρητικά καλύτερη πολυπλοκότητα του υβριδικού δυναμικού προγραμματισμού, το γεγονός ότι το δένδρο βρίσκεται κατακερματισμένο πάνω στον δίσκο και τα ερωτήματα για το μήκος των μεγίστων κοινών επεκτάσεων γίνονται σε τυχαίες θέσεις του προτύπου και του κειμένου κατά την εκτέλεση του αλγορίθμου, οδηγούν στην απαίτηση για φόρτωση στην μνήμη ενός διαφορετικού προθεματικού υποδένδρου σε κάθε βήμα του αλγορίθμου. Το μεγάλο αυτό I/O κόστος οδηγεί σε ραγδαία πρώση της απόδοσης.

Για την επιβεβαίωση του μεγάλου κόστους που επιφέρουν οι κλήσεις για LCE σε τυχαίες θέσεις, πραγματοποιήσαμε μια παραλλαγή της υλοποίησης του υβριδικού δυναμικού προγραμματισμού βασιζόμενοι στην εξής παρατήρηση. Κατά τον υπολογισμό ενός απώτατου d-μονοπατιού, μπορούμε να καθορίσουμε εκ των προτέρων τις θέσεις στις οποίες θα γίνουν οι κλήσεις για μέγιστες κοινές επεκτάσεις για τον υπολογισμό κάποιων επόμενων (d+1)-μονοπατιών, από τις θέσεις που ορίζουν η οριζόντια, κάθετη και διαγώνιος ακμή (που θα είναι το πρώτο βήμα για την δημιουργία ενός R μονοπατιού) μετά το πέρας αυτού του d-μονοπατιού, εφαρμόζοντας το για όλα τα d-μονοπάτια. Για κάθε θέση λοιπόν του προτύπου αποθηκεύουμε όλες τις θέσεις στο κείμενο για τις οποίες θα χρειαστεί στην επόμενη d+1 επανάληψη ένας υπολογισμός μέγιστης κοινής επέκτασης και πριν την εκτέλεση της επόμενης επανάληψης κάνουμε όλους τους υπολογισμούς των μέγιστων κοινών επεκτάσεων με τη σειρά, φορτώνοντας και τα αντίστοιχα προθεματικά υποδένδρα με την σειρά. Βέβαια αφού υπάρχει απαίτηση σταθερού χρόνου, οι τιμές που θα υπολογιστούν θα πρέπει να αποθηκευτούν σε μια δομή που θα επιτρέπει την γρήγορη ανάκτηση τους αν χρειαστούν. Για τον λόγο αυτό χρειάζεται μια δομή πίνακα $m \cdot n$ που θα επιτρέπει την αποδοτική ανάκτηση των προϋπολογισμένων LCE τιμών κατά την εκτέλεση της επόμενης επανάληψης.

Η μέθοδος αυτή βελτιώνει κατά περίπου 3 φορές την απόδοση του υβριδικού δυναμικού προγραμματισμού επιβεβαιώνοντας την εκτίμηση ότι η μεγάλη καθυστέρηση οφείλεται στο επιπλέον I/O κόστος λόγω των συνεχών φορτώσεων των προθεματικών υποδένδρων. Ωστόσο, πέραν του ότι παραμένει πολύ βραδύτερο από την υλοποίηση δυναμικού προγραμματισμού κατευθείαν στην μνήμη, αφού και πάλι φορτώνονται και εκφορτώνονται από την μνήμη όλα τα προθεματικά υποδένδρα k φορές, η προϋπόθεση για την χρήση της τεράστιας m·n δομής στην μνήμη ακυρώνει τις παραδοχές που έχουν γίνει για το μέγεθος της μνήμης και εν τέλει τον λόγο για τον οποίο το δένδρο επιθεμάτων παραμένει στην μνήμη κατακερματισμένο ως ένα δάσος προθεματικών υποδένδρων. Αν δηλαδή έχουμε διαθέσιμη τόση μνήμη δεν υπάρχει λόγος το δένδρο να είναι κατακερματισμένο με αυτόν τον τρόπο.

Επιπλέον ο αλγόριθμος για την ανάκτηση του LCE στην υλοποίηση του υβριδικού δυναμικού προγραμματισμού είναι αρκετά νοθευμένος αφού στις περισσότερες περιπτώσεις η απάντηση στο ερώτημα LCE θα γίνει με σύγκριση μόνο των πρώτων χαρακτήρων του προθέματος. Πράγματι,αν έχουμε 30 προθεματικά υποδένδρα, λιγότερο από το 5% των ερωτημάτων LCE θα χρειαστεί να εφαρμόσει τον αλγόριθμο σταθερού χρόνου για το LCA μέσα σε κάποιο προθεματικό υποδένδρο. Οι υπόλοιπες επεκτάσεις είναι αρκετά μικρές και θα βρίσκονται μέσα στο κείμενο των προθεμάτων. Και βέβαια, ακόμα και για τις περιπτώσεις που θα χρησιμοποιηθεί ο αλγόριθμος σταθερού χρόνου έχουν προηγηθεί οι συγκρίσεις στο κείμενο του προθέματος, δηλαδή για όλα τα ερωτήματα LCE γίνονται συγκρίσεις στα κείμενα των προθεμάτων.

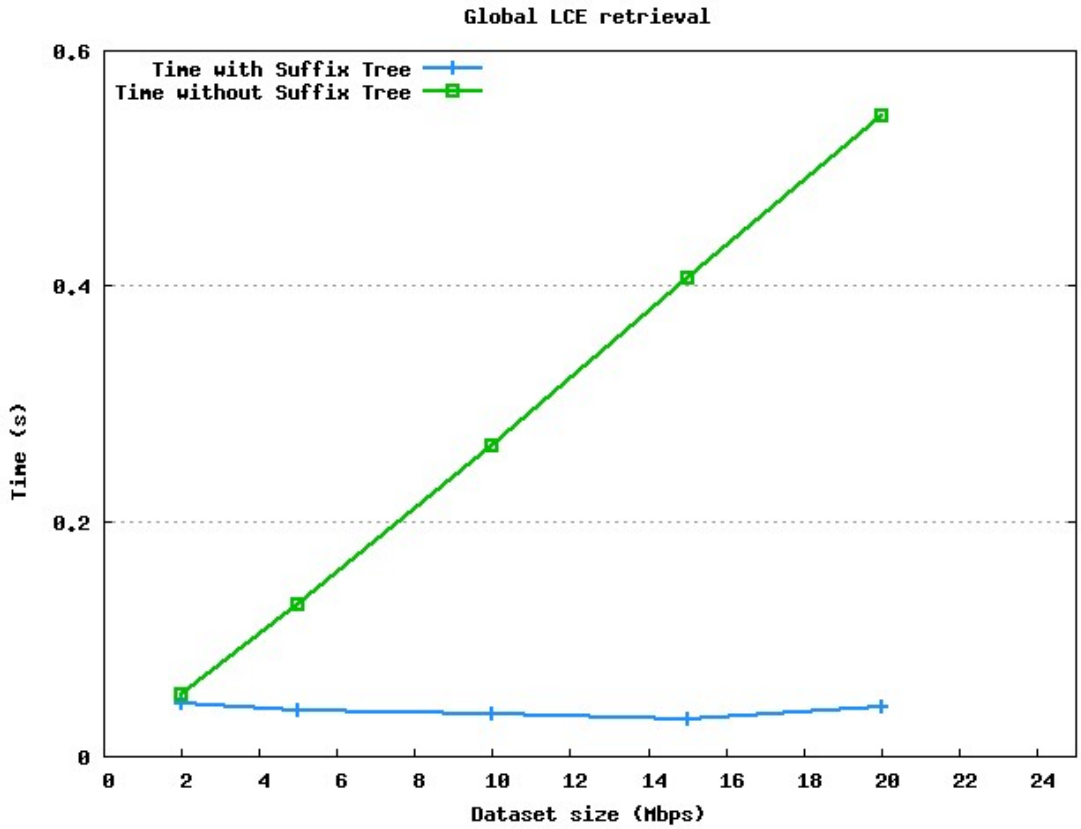
Διαμορφώνουμε λοιπόν το συμπέρασμα ότι τα δένδρα που παράγονται με τον κατακερματισμό στην βάση προθεματικών υποδένδρων δεν είναι κατάλληλα για την εφαρμογή του αλγορίθμου υβριδικού δυναμικού προγραμματισμού, λόγω της συμφόρησης που δημιουργείται από τα ερωτήματα για LCEs σε διαφορετικές θέσεις κειμένου και προτύπου.

5.5 Πειράματα για την υλοποίηση καθολικού LCE με κάτω φράγμα

Στη συνέχεια, πραγματοποιήσαμε πειράματα για την υλοποίηση της καθολικής ανάκτησης του LCE με χρήση των προεπεξεργασμένων δένδρων που περιγράφηκε στην 4.3.2. Όπως αναφέρθηκε τέτοια ερωτήματα είναι χρήσιμα για την ανάλυση της ομοιότητας περιοχών, των οποίων οι πρώτοι χαρακτήρες έχουν κάποια ειδική σημασία, για την εξαγωγή στατιστικών συμπερασμάτων. Ως baseline μέθοδο σύγκρισης χρησιμοποιήσαμε μια παραλλαγή του Brute-

Force αφού οι άλλες μέθοδοι ακριβούς ταύτισης προτύπου χωρίς προεπεξεργασία που χρησιμοποιήθηκαν στα πειράματα του 5.2 είναι δύσκολο να προσαρμοστούν για την εύρεση του LCE αφού δεν είναι γνωστό το μήκος κάθε επέκτασης και οι αλγόριθμοι απαιτούν γνώση εκ των προτέρων του προτύπου και του μήκους του για να κάνουν την απαραίτητη προεπεξεργασία στο πρότυπο.

Τα πειράματα εγιναν για διάφορα μεγέθη δεδομένων και για κάτω φράγμα 8 σύμβολα. Τα ευρετήρια παρήχθησαν από το TRELLIS και στην συνέχεια υπέστησαν την προεπεξεργασία από το πρόγραμμα μας για την δυνατότητα εφαρμογής του αλγορίθμου σταθερού χρόνου για το LCA.



Είναι φανερό ότι η χρήση του δένδρου επιθεμάτων επιτρέπει την απάντηση τέτοιων ερωτημάτων σε πολύ μικρό χρόνο. Ο χρόνος που χρειάζεται για την απάντηση των ερωτημάτων με χρήση του δένδρου είναι περίπου σταθερός ενώ μοιάζει ακόμα και να μειώνεται ανεπαίσθητα σε δύο περιπτώσεις. Εφόσον ισχύει η συνθήκη ότι το κάτω φράγμα είναι μεγαλύτερο του προθέματος, η αύξηση του μεγέθους της ακολουθίας με σταθερή την βασική μνήμη οδηγεί γενικά σε μικρότερη τιμή κατωφλίου, αφού η μεγαλύτερη ακολουθία καταλαμβάνει περισσότερο χώρο στην μνήμη μειώνοντας τον χώρο που είναι διαθέσιμος για τα δένδρα. (βλέπε 3.5.5). Αφού ο αλγόριθμος είναι πολυπλοκότητας O(t), όπου t το κατώφλι, αναμένεται αύξηση της απόδοσης για μικρότερο μέγεθος κατωφλίου.

5.6 Σύνοψη συμπερασμάτων

Από τα πειράματα λοιπόν αναδεικνύεται το μεγάλο πλεονέκτημα που έχουν τα δένδρα επιθεμάτων που παράγονται από το TRELLIS για την αναζήτηση προτύπων με ακριβή ταύτιση (exact matching). Τα ευρετήρια, παρότι βρίσκονται στο δίσκο, είναι σε κάθε περίπτωση ταχύτερα από τους δύο δημοφιλέστερους αλγορίθμους για την αναζήτηση χωρίς ευρετήρια, τον Boyer-Moore και τον Morris-Pratt, που εκτελούνται στην μνήμη.

Επιβεβαιώθηκε επίσης η ταχύτητα τις προσπέλασης εσωτερικών κόμβων με χρήση συνδέσμων επιθέματος, παρόλο που οι σύνδεσμοι δείχνουν γενικά σε διαφορετικά προθεματικά υποδένδρα, απαιτείται δηλαδή ένα επιπλέον κόστος εισόδου για τη φόρτωση ενός νέου προθεματικού υποδένδρου από τον δίσκο.

Τα αποτελέσματα για τον αλγόριθμο προσεγγιστικής ταύτισης με χρήση υβριδικού δυναμικού προγραμματισμού κατέδειξαν το μειονέκτημα του κατακερματισμού των δένδρων με αυτόν τον τρόπο για αλγορίθμους που δεν έχουν τοπικότητα της αναφοράς, αφού ο αλγόριθμος αποδείχθηκε αρκετά βραδύτερος από μια υλοποίηση στην μνήμη χωρίς το ευρετήριο.

Τέλος, η απάντηση ερωτημάτων LCE φάνηκε ότι γίνεται πάρα πολύ αποδοτικά με χρήση των δένδρων που έχουν υποστεί προεπεξεργασία για ανάκτηση του LCA σε σταθερό χρόνο.

6 *Επίλογος*

6.1 Σύνοψη και συμπεράσματα

Αντικείμενο αυτής της διπλωματικής ήταν η μελέτη των δένδρων επιθεμάτων για τις περιπτώσεις όπου το μέγεθός τους ξεπερνά τη διαθέσιμη μνήμη. Στα πλαίσια αυτά μελετήθηκαν σύγχρονες τεχνικές κατασκευής που έχουν προταθεί στη βιβλιογραφία και υλοποιήθηκαν γνωστοί αλγόριθμοι αναζήτησης και επεξεργασίας ακολουθιών που τα χρησιμοποιούν.

Όσον αφορά στη θεωρητική μελέτη, το μεγαλύτερο βάρος δόθηκε στους αλγορίθμους αποδοτικής κατασκευής δένδρων επιθεμάτων, των οποίων το μέγεθος είναι μεγαλύτερο από τη διαθέσιμη κύρια μνήμη. Καταρχάς διατυπώθηκε σαφώς το πρόβλημα και στη συνέχεια μελετήθηκαν οι πιο αποτελεσματικές μέθοδοι που υπάρχουν στην βιβλιογραφία για κατασκευή του δένδρου επιθεμάτων στον δίσκο. Μελετήθηκε η προσέγγιση της Hunt που κατακερματίζει το δένδρο στην βάση προθεμάτων σταθερού μήκους, οι αλγόριθμοι Dyna-Cluster και TOP-Q που χρησιμοποιούν έξυπνες στρατηγικές buffering, ο αλγόριθμος TDD που υλοποιεί μια παραλλαγή του αλγοριθμου wotdeager. Τέλος μελετήθηκε διεξοδικά ο αλγόριθμος TRELLIS, που, απ' όσο γνωρίζουμε, είναι η πιο ώριμη δουλειά στο χώρο. Ο συγκεκριμένος αλγόριθμος φαίνεται να είναι ο ταχύτερος, δίνει τη δυνατότητα για διατήρηση των συνδέσμων επιθέματος (που είναι ιδιαίτερα χρήσιμοι για διάφορους αλγορίθμους) και παρέχει ανοιχτού κώδικα υλοποίηση. Πέρα απ' αυτά μελετήθηκαν και παρουσιάστηκαν γνωστοί αλγόριθμοι που επιλύουν προβλήματα ανάλυσης και αναζήτησης ακολουθιακών δεδομένων. Κάποιοι από αυτούς τους αλγορίθμους χρησιμοποιούν δένδρα επιθεμάτων και κάποιοι άλλοι δεν χρησιμοποιούν. Η παρουσίαση των πρώτων έγινε γιατί τη συμπεριφορά αυτών ακριβώς των αλγορίθμων είναι που θέλουμε να δοκιμάσουμε, όταν το δένδρο που χρησιμοποιούν ξεπερνά σε μέγεθος την κύρια μνήμη. Η παρουσίαση των δεύτερων έγινε

γιατί αυτούς του αλγορίθμους τους χρησιμοποιούμε ως βάση σύγκρισης για την αποδοτικότητα των αλγορίθμων που χρησιμοποιούν τα δένδρα επιθεμάτων. Οι αλγόριθμοι αυτοί ομαδοποιούνται σε αλγορίθμους για την ακριβή ταύτιση προτύπου, αλγορίθμους για την προσεγγιστική ταύτιση προτύπου και αλγορίθμους εύρεσης κοινών προθεμάτων συγκεκριμένου μήκους. Οι τελευταίες περιπτώσεις προϋποθέτουν από το δένδρο επιθεμάτων να απαντά σε σταθέρο χρόνο ερωτήματα μέγιστης κοινής επέκτασης (LCE). Αυτά τα ερωτήματα στο δένδρο επιθεμάτων μεταφράζονται σε ερωτήματα κατώτατου κοινού προγόνου (LCA), για τα οποία υπάρχει αλγόριθμος που μπορεί να τα απαντήσει σε σταθερό χρόνο, με την προϋπόθεση ότι το δένδρο έχει υποστεί κατάλληλη προεπεξεργασία. Για το λόγο αυτό μελετήθηκε θεωρητικά και παρουσιάστηκε επίσης ο συγκεκριμένος άλγοριθμος.

Τα πειραματικά αποτελέσματα της διπλωματικής γενικά επιβεβαίωσαν τα πλεονεκτήματα της χρήσης των δένδρων επιθεμάτων, ακόμα και αν αυτά είναι πολύ μεγάλα για να χωρέσουν στη μνήμη. Ο τρόπος με τον οποίο αποθηκεύονται τα δένδρα από το TRELLIS δεν φαίνεται να παρουσιάζει προβλήματα στις περισσότερες των περιπτώσεων. Εξαίρεση αποτελεί ο αλγόριθμος του υβριδικού δυναμικού προγραμματισμού για την μη-ακριβή ταύτιση προτύπου, ο οποίος δείχνει να είναι πολύ βραδύτερος σε σχέση με την απευθείας υλοποίηση δυναμικού προγραμματισμού στην μνήμη. Ο λόγος για τον οποίο συμβαίνει αυτό είναι η έντονη μη τοπικότητα των διασχίσεων που συμβαίνουν στο δένδρο. Αυτό έχει ως αποτέλεσμα να χρειάζεται υπερβολικά συχνά να φορτωθούν και να εκφορτωθούν από την μνήμη υποδένδρα. Κομβικό σημείο σε αυτό είναι ότι η σελιδοποίηση του δένδρου γίνεται με βάση το πρόθεμα της ακολουθίας. Αυτή η σελιδοποίηση βοηθάει στην γρήγορη κατασκευή του δένδρου, όμως όταν χρησιμοποιείται σε κάποιον αλγόριθμο σαν τον υβριδικό δυναμικό προγραμματισμό, δεν ενδείκνυται. Ο λόγος είναι ότι στον συγκεκριμένο αλγόριθμο η διάσχιση ακολουθιών στο δένδρο γίνεται συχνά και με εντελώς τυχαίο τρόπο. Πρέπει λοιπόν είτε να αναζητηθούν παραλλαγές των υπαρχόντων αλγορίθμων για τέτοια προβλήματα ή να βρεθεί κάποιος διαφορετικός τρόπος σελιδοποίησης των δένδρων. Μια συνεισφορά της παρούσας διπλωματικής είναι η ανάδειξη αυτού του προβλήματος, το οποίο μπορεί να απασχολήσει επόμενες εργασίες. Ένα άλλο θέμα το οποίο αναδεικνύεται, είναι η μεγάλη σπατάλη χώρου που κάνουν τα δένδρα επιθεμάτων. Ενδεικτικά η ανοιχτού κώδικα υλοποίηση του TRELLIS για την αποθήκευση των δένδρων χρειάζεται περίπου 27 bytes για κάθε σύμβολο της ακολουθίας εισόδου. Τα σύμβολα του αλφαβήτου του DNA, επειδή έχουν μέγεθος αλφαβήτου 4, (συμπιεσμένα) καταλαμβάνουν χώρο 2 bits. Επομένως, ακόμα και ολόκληρο το γονιδίωμα ($\approx 3 \cdot 10^9$ σύμβολα) μπορεί να χωρέσει ολόκληρο στην μνήμη ενός σύγχρονου υπολογιστή, ενώ το αντίστοιχο ευρετήριο πρέπει να παραμένει κατακερματισμένο στον δίσκο. Το ζήτημα βέβαια αυτό είναι γνωστό, απλώς αναδειχτηκε επίσης κατά την ενασχόλησή μας με σχετικά ζητήματα.

Πάντως, παρά τα συγκεκριμένα μειονεκτήματα, παρατηρήσαμε ότι για πολλές περιπτώσεις τα δένδρα μπορούν να χρησιμοποιηθούν ως έχουν και να απαντήσουν αποδοτικότερα συγκεκριμένα ερωτήματα σε σχέση με παραδοσιακούς αλγορίθμους που δεν χρησιμοποιούν ευρετήρια.

6.2 Μελλοντικές επεκτάσεις

Όπως αναφέρθηκε ένα από τα μειονεκτήματα των δένδρων επιθεμάτων είναι η σπατάλη χώρου. Το TRELLIS φαίνεται ότι ακολουθεί αυτό τον κανόνα. Είναι μάλιστα αρκετά πιο σπάταλο από τον βασικό ανταγωνιστή του, τον αλγόριθμο TDD, αφού σε ακολουθίες μικρότερες από 1Gbps ο TRELLIS σπαταλά περίπου 27 bytes ανά σύμβολο ενώ ο TDD μόλις 9,7 bytes ανά σύμβολο. Σε επίπεδο γονιδιώματος, ο TDD γίνεται αρκετά πιο σπάταλος με περίπου 19 bytes/σύμβολο αλλά παραμένει λιγότερο σπάταλος από τον TRELLIS. Η πολύ καλή αυτή απόδοση του TDD οφείλεται στην χρήση του αλγορίθμου wordeager που έχει έναν από τους χαμηλότερους λόγους bytes/σύμβολο, κατά κανόνα μικρότερο από 10. Μια ενδιαφέρουσα πρόκληση για μια μελλοντική επέκταση είναι η συμπίεση των αρχείων του TRELLIS με τρόπο που να μην επηρεάζει ραγδαία την απόδοση των αλγορίθμων πάνω στα δένδρα. Η συμπίεση ιδανικά μπορεί να γίνεται online πριν την αποθήκευση των προθεματικών υποδένδρων στον δίσκο.

Αντικείμενο μιας μελλοντικής επέκτασης θα μπορούσε επίσης να είναι η περαιτέρω βελτίωση του χρόνου για την ανάκτηση του LCE που λόγω του κατακερματισμού δεν έχει απόδοση O(1) αλλά όπως αναφέρθηκε, O(μ), όπου μ το μέγιστο μήκος προθέματος μέσα στο P. Το μήκος αυτό είναι γενικά μικρό αλλά σε περιπτώσεις που γίνονται πολλά τέτοια ερωτήματα η διαφορά μπορεί να μην είναι ευκαταφρόνητη. Το πλήθος των προθεμάτων είναι αρκετά μικρό (6400 για ολόκληρο το γονιδίωμα με κατώφλι $t=10^6$). Με δεδομενο αυτό, αν προστεθεί μια στήλη στον πίνακα επιθεμάτων-κόμβων του αρχείου suffix.dat και για κάθε επίθεμα αποθηκεύεται, εκτός από τον κόμβο-φύλλο στα προεπεξεργασμένα αρχεία, το πρόθεμα στο οποίο αντιστοιχεί, (για άνω όριο 6400 προθέματα η στήλη αυτή μπορεί να επιβαρύνει με 2 bytes επιπλέον ανά στοιχείο, αφού σε 2 bytes μπορούν να αποθηκευτούν 65535 διαφορετικοί αριθμοί άρα η επιβάρυνση για την μνήμη είναι 2n), και σε έναν πίνακα μ·μ, που είναι αρκετά μικρός και μπορεί να θεωρηθεί ότι χωρά στην μνήμη χωρίς αλλαγή των παραδοχών, είναι προϋπολογισμένες όλες οι μέγιστες κοινές επεκτάσεις μεταξύ όλων των συνδυασμών προθεμάτων, μπορεί να ανακτηθεί η μέγιστη κοινή επέκταση των χαρακτήρων του προθέματος σε σταθερό χρόνο, και μάλιστα χωρίς προσπέλαση στην ακολουθία εισόδου. Αν η αναζήτηση γίνεται στο ίδιο πρόθεμα η ανάκτηση του LCE μπορεί να γίνει με εφαρμογή του αλγορίθμου σταθερού χρόνου στο αντίστοιχο προθεματικό υποδένδρο κατά τα γνωστά. Με αυτόν τον τρόπο η συνολική ανάκτηση του LCE θα είναι πολυπλοκότητας O(1) με την

αντίστοιχη επιβάρυνση στην μνήμη βέβαια για την προσθήκη της στήλης. Εκτός αυτού θα ήταν σημαντικό να μελετηθούν τρόπουι με τους οποίους να μπορούν να παρακαμφθούν τα προβλήματα απόδοσης που παρουσιάζονται σε κάποιους αλγορίθμους λόγω της σελιδοποίησης του δένδρου με βάση τα προθέματα. Κάποια παράλλαξη των αλγορίθμων ή του τρόπου διαμέρισης του δένδρου θα βοηθούσε προς αυτή την κατεύθυνση.

Ένα άλλο ζήτημα είναι ότι, στην τρέχουσα έκδοση το TRELLIS έχει την δυνατότητα αποθήκευσης μόνο ακολουθιών DNA (ή γενικώς αποθήκευσης ακολουθιών πάνω σε αλφάβητο μεγέθους 4 συμβόλων). Μια χρήσιμη επέκταση θα μπορούσε να δώσει στο TRELLIS την δυνατότητα παραγωγής ευρετηρίου για άλλου είδους δεδομένα, όπως ακολουθίες αμινοξέων, όπου το μέγεθος του αλφαβήτου είναι 20. Μια τέτοια επέκταση θα μετέβαλλε την δομή των κόμβων για να μπορούν να δεχθούν τις πρόσθετες πληροφορίες.

Τέλος, ως μελλοντική επέκταση θα μπορούσε να μελετηθεί η δυνατότητα εφαρμογής στο TRELLIS και στα προεπεξεργασμένα δένδρα, ενός από τους αλγορίθμους που κάνουν χρήση συνδέσμων επιθέματος όπως η στοίχηση με χρήση δικλείδων ταύτισης ή αλγορίθμων που κάνουν χρήση ερωτημάτων LCE όπως ο αλγόριθμος για την εύρεση παλινδρόμων [Gus97].

7 *Βιβλιογραφία*

[Παπ07] Παπαδόπουλος Μ., Τεχνικές Διαχείρισης Δεδομένων σε Βιοεπιστήμες, Διπλωματική Εργασία Ε.Μ.Π., 2007

[AC91] Apostolico A., Crochemore M., Optimal canonization of all substrings of a string, Information and Computation 95(1):76-95, 1991.

[AG86] Apostolico A., Giancarlo R., The Boyer-Moore-Galil string searching strategies revisited, SIAM Journal on Computing 15(1):98-105, 1986.

[AKO04] Abouelhoda M., Kurtz S., Ohlebusch E., Replacing Suffix Trees with Enhanced Suffix Arrays, Journal of Discrete Algorithms, 2, 53-86, 2004.

[BDP03] Bray N., Dubchak I., Pachter L., AVID: A global alignment program. Genome Research, 13(1):97–102, 2003.

[BG92] Baeza-Yates R, Gonnet G., A new approach to text searching, Communications of the ACM, 35(10):74-82, 1992.

[BH04] Bedathur S., Haritsa J., Engineering a fast online persistent suffix tree construction, 20th International Conference on Data Engineering, 2004.

[BM77] Boyer R., Moore J., A fast string searching algorithm, Communications of the ACM, 20:762-772, 1977.

[BR99] Berry T., Ravindran S., A fast string matching algorithm and experimental results, Prague Stringology Club Workshop`99, 6-26, 1999.

[CCG+91] Crochemore M., Czumaj A., Gasieniec L., Jarominek S., Lecroq T., Plandowski W., Rytter W., Deux méthodes pour accélérer l'algorithme de Boyer-Moore, Théorie des Automates et Applications, Actes des 2^{e} Journées Franco-Belges, D. Krob ed., Rouen, France, 1991

[CH97] Crochemore M., Hancart, C., Automata for Matching Patterns, Handbook of Formal Languages, Volume 2, Linear Modeling: Background and Application, G. Rozenberg and A. Salomaa ed., 399-462, Springer-Verlag, Berlin, 1997.

[CLP88] Charras C., Lecroq T., Pehoushek J., A very fast string matching algorithm for small alphabets and long patterns, in Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching , M. Farach-Colton ed., Piscataway, New Jersey, Lecture Notes in Computer Science 1448, pp 55-64, Springer-Verlag, Berlin, 1998.

[Col94] Colussi L., Fastest pattern matching in strings, Journal of Algorithms, 16(2):163-189, 1994.

[CYL05] Cheung C., Yu J., Lu H., Constructing suffix tree for gigabyte sequences with megabyte memory, IEEE Transactions on Knowledge and Data Engineering, 17(1):90–105, 2005.

[DPC+02] Delcher A., Phillippy A., Carlton J., Salzberg S., Fast algorithms for large-scale genome alignment and comparison. Nucleic Acids Research, 30(11):2478–2483, 2002.

[GK95] Giegerich R., Kurtz. S., A Comparison of Imperative and purely Functional Suffix tree constructions, Science of Programming, 1995.

[GKS03] Giegerich R., Kurtz S., Stoye J., Efficient implementation of lazy suffix trees.,Software Practice & Experience, 33(11):1035–1049, 2003

[GS83] Galil Z., Seiferas J., Time-space optimal string matching, Journal of Computer and System Science 26(3):280-294, 1983.

[GS94] Gusfield D., Stoye J., Linear time algorithms for finding and representing all the tandem repeats in a string. Journal of Computer and System Sciences, 69(4):525–546, 2004.

[Gus97] Gudfield D., Algorithms on Strings, Trees and Sequences, Cambridge University Press, 1997.

[HAI01] Hunt E., Atkinson M., Irving R., A database index to large biological sequences, 27th International Conference on Very Large Data Bases, 2001.

[HKO02] Höhl M., Kurtz S., Ohlebusch E., Efficient multiple genome alignment, Bioinformatics, 18 (supplement 1):312–320, 2002.

[Hor80] Horspool R., Practical fast searching in strings, Software - Practice & Experience, 10(6):501-506, 1980.

[HS91] Hume A., Sunday D., Fast string searching. Software - Practice & Experience 21(11):1221-1248, 1991.

[HT84] Harel D., Tarjan R., Fast algorithms for finding nearest common ancestors, SIAM J. Comput., 13:338-55, 1984

[Jap04] Japp R., The top-compressed suffix tree: A disk-resident index for large seqeuences, Bioinformatics Workshop, 21st Annual British National Conference On Databases, 2004.

[KMP77] Knuth D., Morris J., Pratt V., Fast pattern matching in strings, SIAM Journal on Computing 6(1):323-350, 1977.

[KR87] Karp R., Rabin M., Efficient randomized pattern-matching algorithms. IBM J. Res. Dev. 31(2):249-260, 1987.

[Lec92] Lecroq T., A variation on the Boyer-Moore algorithm, Theoretical Computer Science 92(1):119—144, 1992.

[McC76] E. McCreight. A space-economical suffix tree construction algorithm, Journal of the ACM, 23(2):262–272, 1976.

[MP70] Morris J., Pratt V., A linear pattern-matching algorithm, Technical Report 40, University of California, Berkeley, 1970.

[Nav01] Navarro G., A guided tour to approximate string matching, ACM Computing Surveys (CSUR), v.33 n.1, 31-88, 2001

[PZ07] Phophakdee B., Zaki M., Genome-scale Disk-based Suffix Tree Indexing, ACM SIGMOD International Conference on Management of Data, 2007.

[Smi91] Smith P., Experiments with a very fast substring search algorithm, Software - Practice & Experience 21(10):1065-1074, 1991.

[Sun90] Sunday D., A very fast substring search algorithm, Communications of the ACM, 33(8):132-142, 1990.

[SV88] Schieber B., Vishkin U., On finding lowest common ancestors: simplifications and parallelization, SIAM J. Comput., 17:1253-62, 1988

[Tel09] The Daily Telegraph, Deciphering a person's DNA to cost $1000, 11/8/2009.

[THP04] Tata S., Hankins R., Patel J., Practical suffix tree construction, 30th International Conference on VLDB, 2004.

[TTH+05] Tian Y., Tata S., Hankins R., Patel J., Practical methods for constructing suffix trees, VLDB Journal, 14(3):281–299, 2005.

[Ukk85] Ukkonen E., Algorithms for approximate string matching. Information and Control 64,100–118, 1985

[Ukk95] Ukkonen E., On-line construction of suffix trees, Algorithmica, 14(3), 1995.

[Wei73] Weiner P., Linear pattern matching algorithms, 14th IEEE Symp. on Switching and Automata Theory, 1972