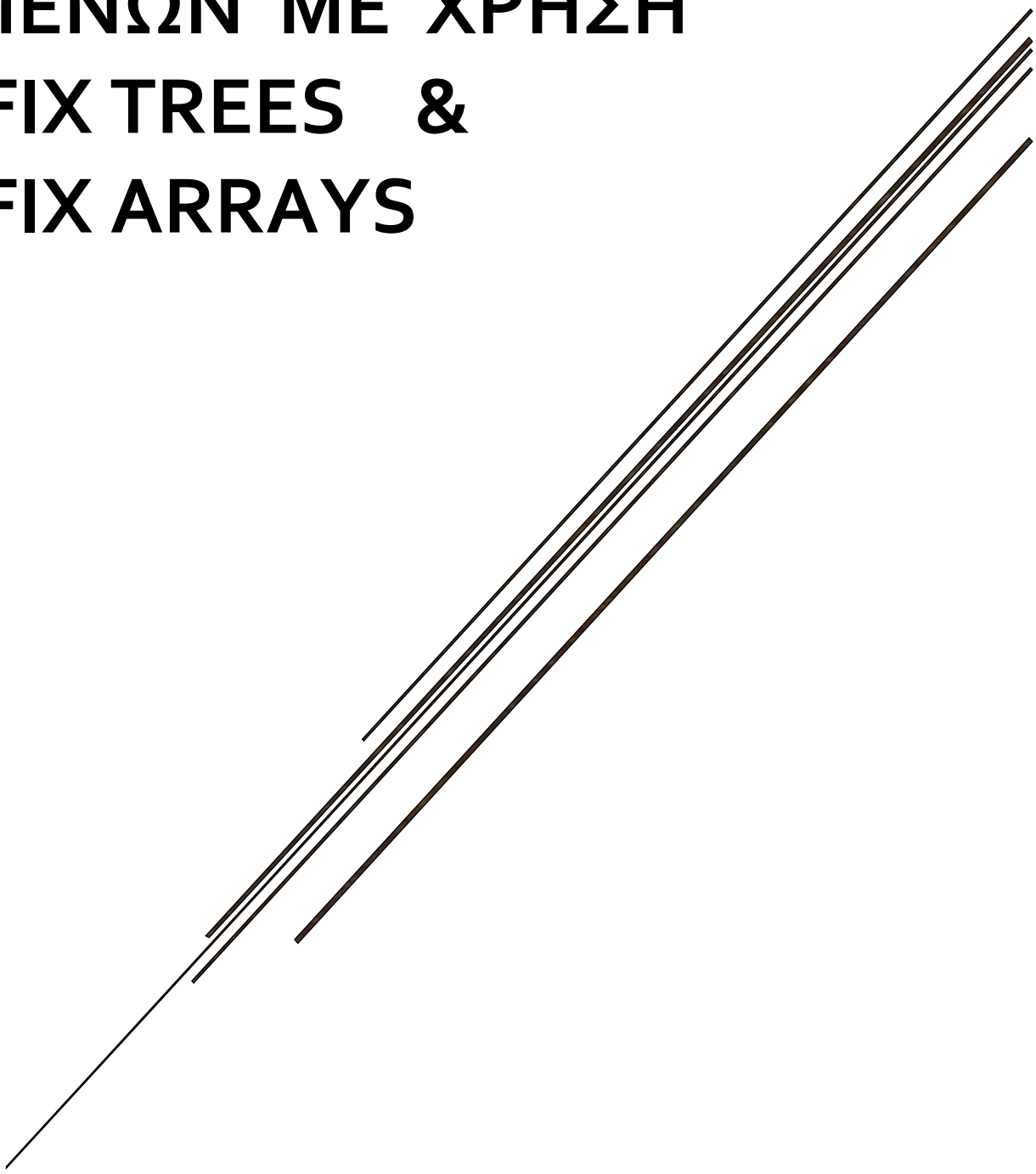
ΔΕΙΚΤΟΔΟΤΗΣΗ ΚΕΙΜΕΝΩΝ ΜΕ ΧΡΗΣΗ SUFFIX TREES & SUFFIX ARRAYS



Μπουλαφέντης Χρήστος ΑΜ:1059612 & Σκύρλα Αγάθη ΑΜ:1064888

Περιεχόμενα

Εισαγωγή

Αυτή η εργασία αφορά τα Suffix Trees και Suffix Arrays. Αρχικά, θα δούμε κάποιους βασικούς ορισμούς, για να γίνει πιο κατανοητή η λειτουργία των Δέντρων Επιθεμάτων και Πινάκων. Έπειτα, θα γίνει μία αναδρομή στο παρελθόν και στην ιστορία ανάπτυξης αυτών των δομών, από την πρωταρχική μορφή μέχρι την σημερινή σύγχρονη ανάπτυξη τους. Ακόμη, ακολουθεί μία πιο λεπτομερής παρουσίαση των Suffix Trees και Suffix Arrays, καθώς και συνήθης χρήσεις των δομών αυτών, όπως και νέα είδη δομών τα οποία βασίστηκαν στα δέντρα/πίνακες επιθέματος. Τέλος, θα δούμε κάποια συμπεράσματα πάνω σε αυτά που προ-αναφέραμε.

1. Βασικοί Ορισμοί

1.1 Δομές Δεδομένων

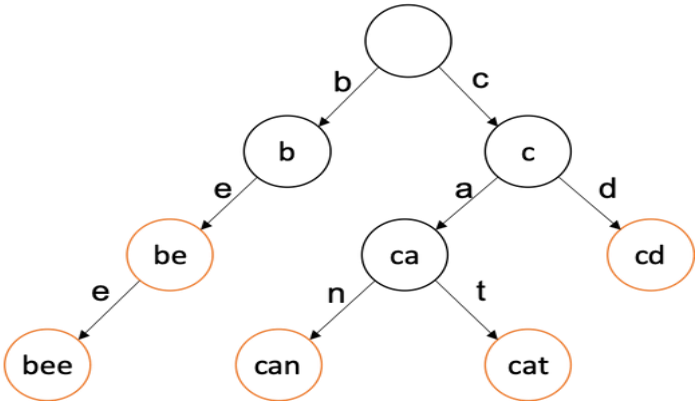
Στην επιστήμη των υπολογιστών, ορίζουμε ως δομές δεδομένων ένα σύνολο πολλών και διαφορετικών τρόπων οργάνωσης και αποθήκευσης δεδομένων. Κάθε μία από αυτές είναι ένα σύνολο αποθηκευμένων πληροφοριών, τα οποία τα επεξεργάζεται ένα σύνολο λειτουργειών[1]. Κάθε μορφή δομής αποτελείτε από ένα σύνολο κόμβων(nodes).Οι βασικές λειτουργίες τους είναι οι εξής:

- Προσπέλαση
- Εισαγωγή
- Διαγραφή
- Αναζήτηση
- Ταξινόμηση
- Αντιγραφή
- Συγχώνευση
- Διαχωρισμός

Μία δομή δεδομένων είναι αποδοτικότερη από μία άλλη δομή με κριτήριο κάποια άλλη λειτουργία[2].

1.2 TRIE

Ένα Trie είναι μία δομή δεδομένων που ονομάζεται επίσης Prefix Tree(Δέντρο Προθέματος) ή Digital Tree(Ψηφιακό Δέντρο)[3].Συγκεκριμένα ανήκει στις Υβριδικές Δομές δεδομένων συνδυάζει την χρήση δεικτών και πινάκων, όπως επίσης είναι δέντρο πολλαπλών δρόμων (multi-way). Χρησιμοποιείται για την εύκολη αποθήκευση και ανάκτηση πληροφορίας σε λέξεις, συμβολοσειρές, επιθέματα και λοιπά, σε χρόνο ανάλογο τους μήκους τους. [4]



Εικόνα 1: Παράδειγμα TRIE για τη συμβολοσειρά 'bee,can,cat,cd'.

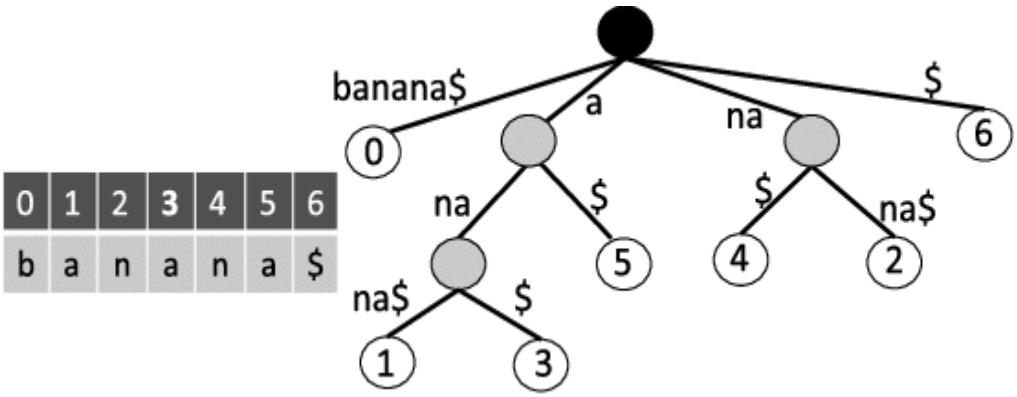
```c
#include <stdio.h>
#include <stdlib.h>

// define character size
#define CHAR_SIZE 26

// A Trie node
struct Trie
{
    int isLeaf;    // 1 when node is a leaf node
    struct Trie* character[CHAR_SIZE];
};

// Function that returns a new Trie node
struct Trie* getNewTrieNode()
{
    struct Trie* node = (struct Trie*)malloc(sizeof(struct Trie));
    node->isLeaf = 0;

    for (int i = 0; i < CHAR_SIZE; i++)
        node->character[i] = NULL;

    return node;
}

// Iterative function to insert a string in Trie
void insert(struct Trie *head, char* str)
{
    // start from root node
    struct Trie* curr = head;
    while (*str)
    {
        // create a new node if path doesn't exists
        if (curr->character[*str - 'a'] == NULL)
            curr->character[*str - 'a'] = getNewTrieNode();

        // go to next node
        curr = curr->character[*str - 'a'];

        // move to next character
        str++;
    }

    // mark current node as leaf
    curr->isLeaf = 1;
}

// Iterative function to search a string in Trie. It returns 1
// if the string is found in the Trie, else it returns 0
int search(struct Trie* head, char* str)
{
    // return 0 if Trie is empty
    if (head == NULL)
        return 0;

    struct Trie* curr = head;
    while (*str)
    {
        // go to next node
        curr = curr->character[*str - 'a'];

        // if string is invalid (reached end of path in Trie)
        if (curr == NULL)
            return 0;

        // move to next character
        str++;
    }

    // if current node is a leaf and we have reached the
    // end of the string, return 1
    return curr->isLeaf;
}
```

```c
    // if current node is a leaf and we have reached the
    // end of the string, return 1
    return curr->isLeaf;
}

// returns 1 if given node has any children
int haveChildren(struct Trie* curr)
{
    for (int i = 0; i < CHAR_SIZE; i++)
        if (curr->character[i])
            return 1;    // child found

    return 0;
}

// Recursive function to delete a string in Trie
int deletion(struct Trie **curr, char* str)
{
    // return if Trie is empty
    if (*curr == NULL)
        return 0;

    // if we have not reached the end of the string
    if (*str)
    {
        // recur for the node corresponding to next character in
        // the string and if it returns 1, delete current node
        // (if it is non-leaf)
        if (*curr != NULL && (*curr)->character[*str - 'a'] != NULL &&
            deletion(&((*curr)->character[*str - 'a']), str + 1) &&
            (*curr)->isLeaf == 0)
        {
            if (!haveChildren(*curr))
            {
                free(*curr);
                (*curr) = NULL;
                return 1;
            }
            else {
                return 0;
            }
        }
    }

    // if we have reached the end of the string
    if (*str == '\0' && (*curr)->isLeaf)
    {
        // if current node is a leaf node and don't have any children
        if (!haveChildren(*curr))
        {
            free(*curr); // delete current node
            (*curr) = NULL;
            return 1; // delete non-leaf parent nodes
        }

        // if current node is a leaf node and have children
        else
        {
            // mark current node as non-leaf node (DON'T DELETE IT)
            (*curr)->isLeaf = 0;
            return 0;       // don't delete its parent nodes
        }
    }

    return 0;
}

// Trie Implementation in C - Insertion, Searching and Deletion
int main()
{
    struct Trie* head = getNewTrieNode();

    insert(head, "hello");
    printf("%d ", search(head, "hello"));       // print 1

    insert(head, "helloworld");
    printf("%d ", search(head, "helloworld"));  // print 1

    printf("%d ", search(head, "helll"));       // print 0 (Not present)

    insert(head, "hell");
    printf("%d ", search(head, "hell"));        // print 1

    insert(head, "h");
    printf("%d \n", search(head, "h"));         // print 1 + newline

    deletion(&head, "hello");
    printf("%d ", search(head, "hello"));       // print 0 (hello deleted)
    printf("%d ", search(head, "helloworld"));  // print 1
    printf("%d \n", search(head, "hell"));      // print 1 + newline

    deletion(&head, "h");
    printf("%d ", search(head, "h"));           // print 0 (h deleted)
    printf("%d ", search(head, "hell"));        // print 1
    printf("%d\n", search(head, "helloworld")); // print 1 + newline

    deletion(&head, "helloworld");
    printf("%d ", search(head, "helloworld"));  // print 0
    printf("%d ", search(head, "hell"));        // print 1

    deletion(&head, "hell");
    printf("%d\n", search(head, "hell"));       // print 0 + newline

    if (head == NULL)
        printf("Trie empty!!\n");               // Trie is empty now

    printf("%d ", search(head, "hell"));        // print 0

    return 0;
}
```

Εικόνα 2: Παράδειγμα Κώδικα TRIE σε γλώσσα C. [5]

1.3 Suffix Trees

Ένα Suffix Tree ή αλλιώς Δέντρο Επιθέματος, αποτελεί μία Δομή Δεδομένων ψηφιακού δέντρου (Digital Tree), η οποία είναι βασισμένη σε όλα τα επιθέματα του δεδομένου κειμένου π.χ T=[$t_i, t_j, \dots, t_n$]='$'.έχει ρίζα και είναι ένα κατευθυνόμενο δέντρο, το οποίο περιέχει n φύλα (δηλαδή τα τελικά επίπεδα του δέντρου) αριθμημένα από το 1-n. Οι δείκτες στα επιθέματα $t_i, t_j, \dots, t_n$ αποθηκεύονται στα φύλα του δέντρου ως τιμές[6].Τα Suffix Trees επιτρέπουν πολλές, γρήγορες και πολύπλοκες υλοποιήσεις συμβολοσειρών και όχι μόνο.



Εικόνα 3:Παράδειγμα Suffix Tree για το κείμενο 'banana'.

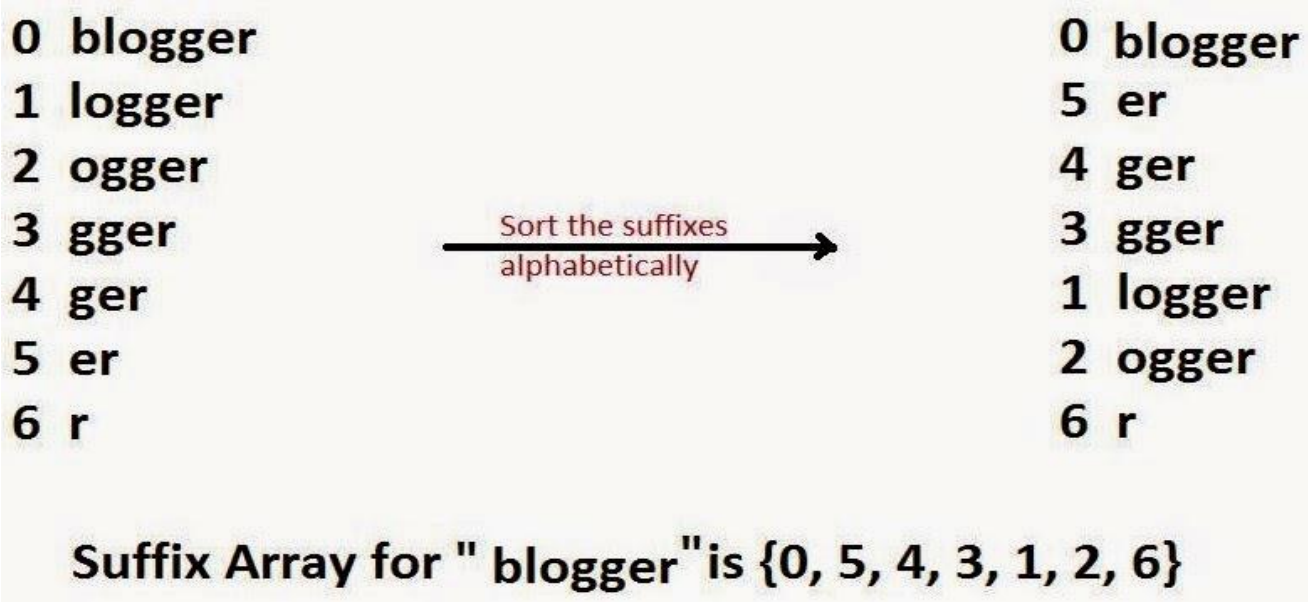
Αναζήτηση-Δέντρου-Επιθέματος $(S, P = p_1 p_2 \dots p_m)$

(1) $i \leftarrow 1$
(2) **μέχρις_ότου** *true* **κάνε** {
(3) **αν** S είναι ένα φύλλο που δείχνει στο j **τότε** {
(4) **αν** $p_i \dots p_m = t_{j+i-1} \dots t_{j+m-1}$
(5) **τότε επίστρεψε** S
(6) **αλλιώς επίστρεψε** *null*
}
(7) **αν** υπάρχει μια ακμή $(S \xrightarrow{p'_1 \dots p'_s} S' \wedge p'_1 = p_i$ **τότε** {
(8) $j \leftarrow 0$
(9) **μέχρις_ότου** $j < s \wedge i + j \le m \wedge p'_{j+1} = p_{i+j}$ **κάνε** $j \leftarrow j+1$
(10) $i \leftarrow i + j$
(11) **αν** $i > m$ **τότε επίστρεψε** S'
(12) **αν** $j < s$ **τότε επίστρεψε** *null*
(13) $S \leftarrow S'$
}
(14) **αλλιώς επίστρεψε** *null*

Εικόνα 4: Παράδειγμα Ψευδοκώδικα για την αναζήτηση Suffix Tree,της συμβολοσειράς P. [7]

1.4 Suffix Array

Ένας πίνακας επιθέματος(Suffix Array), είναι ένας ταξινομημένος πίνακας που περιέχει όλα τα επιθέματα μίας συμβολοσειράς[8]. Ανήκει στην κατηγορία των Δομών Δεδομένων και είναι μία εναλλακτική και πιο αποδοτική ως προς το χώρο, αναπαράσταση Suffix Tree(Δέντρο Επιθέματος). Η κύρια ιδέα είναι ότι, εάν γνωρίζουμε ότι η συμβολοσειρά $t_{i+1} \dots t_n < t_{j+1} \dots t_n$ και $t_i = t_j$ τότε μπορούμε να συμπεράνουμε ότι $t_i \dots t_n < t_j \dots t_n$, χωρίς να καμία σύγκριση συμβολοσειράς[9].



Εικόνα 5: Παράδειγμα Suffix Array για την συμβολοσειρά 'blogger'.

2 Ιστορική Αναδρομή

Ας ξεκινήσουμε με μία αναδρομή στο παρελθόν, ώστε να κατανοήσουμε την σύνδεση των Δέντρων Επιθεμάτων(Suffix Trees) με τους Πίνακες Επιθέματος (Suffix Arrays), καθώς και την εξέλιξη τους τα τελευταία πενήντα χρόνια.

Το πρόβλημα εύρεσης κοινής συμβολοσειράς, έγινε γνωστό από τον Donald Knuth ο οποίος υποστήριξε ότι απαιτεί χρόνο $\Omega(nlogn)$ [10]. Έπειτα, το 1973 η θεωρία του Knuth καταρρίφθηκε από τον Peter Weiner, με την προϋπόθεση ότι το αλφάβητο της συμβολοσειράς να είναι εξ' αρχής γνωστό. Ο Weiner, δημοσίευσε έναν αλγόριθμο, ο οποίος έλυνε το πρόβλημα του Knuth σε γραμμικό χρόνο O(n).Την δομή αυτή την ονόμασε Bi-tree,η οποία είχε το χαρακτηριστικό ότι διάβαζε τη συμβολοσειρά από δεξιά προς τα αριστερά και

επομένως, ήταν εκτός γραμμής(offline) [11]. Τρία χρόνια αργότερα το 1976, ο McCreight δημοσίευσε παρόμοια δομή με του Weiner, με την διαφορά ότι ο αλγόριθμός του διάβαζε την συμβολοσειρά από το τα αριστερά προς τα δεξιά και επομένως ήταν και αυτή εκτός γραμμής(offline). Την δομή αυτή την ονόμασε Δέντρο Επιθέματος(Suffix Tree), η οποία ήταν η πρώτη αναφορά σε αυτά και είχε γραμμικό χρόνο κατασκευής O(n) [12]. Αυτή η ανακάλυψη αμέσως πυροδότησε πολλές νέες έρευνες, για την εύρεση ενός online αλγορίθμου. Αργότερα, το 1981 οι Michael Rodeh, Vaugham Pratt και Shimon Even εκτέλεσαν τη μέθοδο συμπίεσης των Suffix Trees από τους Lempel και Ziv, σε βέλτιστο γραμμικό χρόνο[13]. Ακολουθώντας την δουλειά των Weiner και Pratt, το 1985, ο Alberto Apostolico δημοσίευσε ένα paper στο οποίο παρουσίασε επιπλέον λειτουργίες των προηγούμενων υλοποιήσεων[14]. Όμως, παρατηρήθηκε ότι, το Suffix Tree στις εφαρμογές του όπου απαιτούταν, κατανάλωνε μεγαλύτερο χώρο, σε σχέση με το χώρο που παρέχονταν από την πηγή. Για περίπου μία πενταετία, ο Stefan Kurtz και οι συνεργάτες του, αφιέρωσαν μεγάλο κομμάτι της δουλειάς τους ώστε να κάνουν κατανομή του δέντρου και μερικές από τις δομές του, στη μνήμη του υπολογιστή[15]. Ύστερα, το 1994 οι Burrows και Wheeler πρότειναν μία μέθοδο συμπίεσης βασισμένη στην ταξινόμηση με Suffix(Suffix Sorting) [16]. Το 1990 πρωτοεμφανίστηκε ο ορισμός του Suffix-Array (Πίνακας επιθέματος), από τους Udi Manber και Eugene W. Myers.Οι οποίοι απάλειψαν τη δομή του Suffix Tree, όμως η δομή αυτή ήταν ικανή ώστε να υλοποιήσει μερικές μόνο από τις διεργασίες του δέντρου επιθέματος, με απαιτούμενο χώρο ίσο με δύο ακεραίους σε κάθε χαρακτήρα του κειμένου και με χρόνο αναζήτησης O(|P|+logn) [17]. Το 2000 οι Paolo Feraggina και Giovanni Manzini παρουσίασαν την δομή FM-index,η οποία ήταν ένα συμπιεσμένο Suffix Array βασισμένη πάνω στον μετασχηματισμό των Burrows και Wheeler[18]. Η δομή αυτή, είχε την ιδιότητα να είναι μικρότερη του αρχικού κειμένου και να μην χρειάζεται αποσυμπίεση κατά την αναζήτηση. Την ίδια χρονιά, οι Grossi και Vitter παρουσίασαν ένα σύγχρονο succinct Suffix Tree-Array(σύντομο Δέντρο Επιθέματος-Πίνακα) με O(n) bits, για δυαδικό αλφάβητο. Στην αρχή, πίστευαν πως ήταν μία διαφορετική δομή δεδομένων από το Suffix Tree, διότι το Suffix Array χρησιμοποιούσε γραμμικό χρόνο O(n logn) για την κατασκευή οποιουδήποτε αλφαβήτου, σε αντίθεση με το δέντρο επιθέματος[19]. Έπειτα από έρευνες όμως, αυτή η διάκριση υποχώρησε. Αφού το 2001, ο Toru Kasai έδειξε ότι το Suffix Tree μπορεί να αναπαρασταθεί σε γραμμικό χρόνο από το Suffix Array[20]. Έτσι λοιπόν, συμπέρανε ότι ο Πίνακας Επιθέματος ήταν μία περιληπτική αναπαράσταση του Δέντρου Επιθέματος, όπως και το αντίστροφο. Έπειτα, το 2003 τρείς ομάδες παρουσίασαν, με διάφορες αλλαγές στον αλγόριθμο του Farach για την κατασκευή του Suffix Tree, την πρώτη απευθείας κατασκευή ενός suffix array σε γραμμικό χρόνο. Εκδόθηκε το 2007 μία μελέτη, από τους

Hon, Lam, Sung και Yui, η οποία αφορούσε την βελτίωση, ως προς τον χώρο κι τον χρόνο στο Compressed Suffix Array(CSA) των Vitter και Grossi. Αργότερα, το 2011 οι Nong, Zang και Chan δημιούργησαν έναν αλγόριθμο ,με γραμμικό χρόνο, γρήγορη απόδοση και χαμηλή απαίτηση χώρου, για την κατασκευή ενός Πίνακα Επιθέματος[21]. Ταυτόχρονα με την εξέλιξη των Suffix Arrays, ο Sadakane το 2007 παρουσίασε μία νέα δομή για Compressed Suffix Tree(CST,Συμπιεσμένο Δέντρο Επιθέματος), στην οποία διατηρούσε όλες τις λειτουργίες του Suffix Tree και απαιτούσε χώρο O(n log|A|). Δηλαδή, η δομή αυτή πρόσθετε 6n bits επιπλέον, στο μέγεθος του CSA[22]. Επίσης την ίδια χρονιά, δημοσιεύτηκε μία βελτιωμένη δομή του Sadakane από τους Fischer και Heun, η οποία απαιτούσε μόνο 4n bits επιπλέον στο μέγεθος του CSA[23]. Παρά τις βελτιώσεις του απαιτούμενου χώρου, οι δύο προαναφερόμενες δομές, ήταν εντός του ορίου Θ(n), το όριο αυτό έσπασαν το 2008 οι Russo, Navarro και Oliviera εκδίδοντας τα Fully Compressed Suffix Trees(FCST, Πλήρως Συμπιεσμένα Δέντρα Επιθέματος) [24]. Ο απαιτούμενος χώρος μειώθηκε στο O(nlogσ), αυξάνοντας όμως σημαντικά τον χρόνο των διεργασιών. Τέλος, μία πενταετία αργότερα, το 2014 οι Russo και Navarro βελτίωσαν το FCST, δημοσιεύοντας ένα paper με τίτλο “Fast Fully Compressed Suffix Tree”?, όπου οι διεργασίες απαιτούσαν πολύ-λογαριθμικό χρόνο. [25]

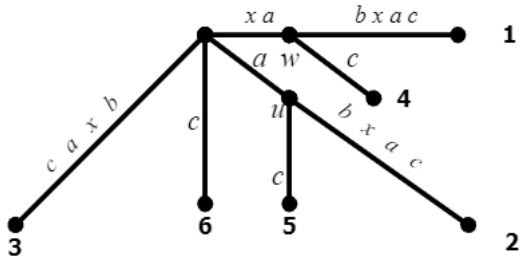
3 Αναλυτικά

Με μια ματιά:

Σε αυτό το κεφάλαιο, θα παρουσιάσουμε κάποιες δομές δεδομένων, οι οποίες διαβάζοντας τα paper μας κίνησαν το ενδιαφέρον. Εκτός αυτού, θεωρήσαμε ότι παίζουν σημαντικό ρόλο στην κατανόηση της εξέλιξης όχι μόνο των Suffix Trees αλλά και των Suffix Arrays, καθώς επίσης και των νέων ειδών που διαμορφώθηκαν από την αρχική μορφή αυτών των δύο. Επιπλέον, παρατίθενται εικόνες ή και μικρά παραδείγματα για την κατανόηση αυτών των δομών.

3.1 Suffix Trees

Όπως είδαμε τα Suffix Trees δημιουργήθηκαν από τον McCreight, οποίος δημοσίευσε παρόμοια δομή με του αρχικού αλγορίθμου του Weiner (ο οποίος δεν χρησιμοποίησε τον όρο 'Suffix Tree' αλλά 'Bi-tree'), με την διαφορά ότι ο αλγόριθμός του είχε γραμμικό χρόνο κατασκευής O(n). Τα δέντρα επιθέματος είναι μία κατευθυνόμενη δεντρική δομή με ακριβώς n φύλλα, για μία συμβολοσειρά A μεγέθους n, τα φύλλα είναι αριθμημένα από το 1 έως το n. Κάθε εσωτερικός κόμβος, που δεν έχει ρίζα, έχει τουλάχιστον δύο παιδιά και κάθε πλευρά αντιστοιχεί σε μία μη-μηδενική υπο-συμβολοσειρά της A. Εάν ενώσουμε τις υπο-συμβολοσειρές, με διαδρομή από την ρίζα προς κάποιο φύλλο(έστω με αριθμό i), τότε σχηματίζουμε το επίθεμα την συμβολοσειράς A που ξεκινά από την θέση i, δηλαδή A[i…n]. [26]

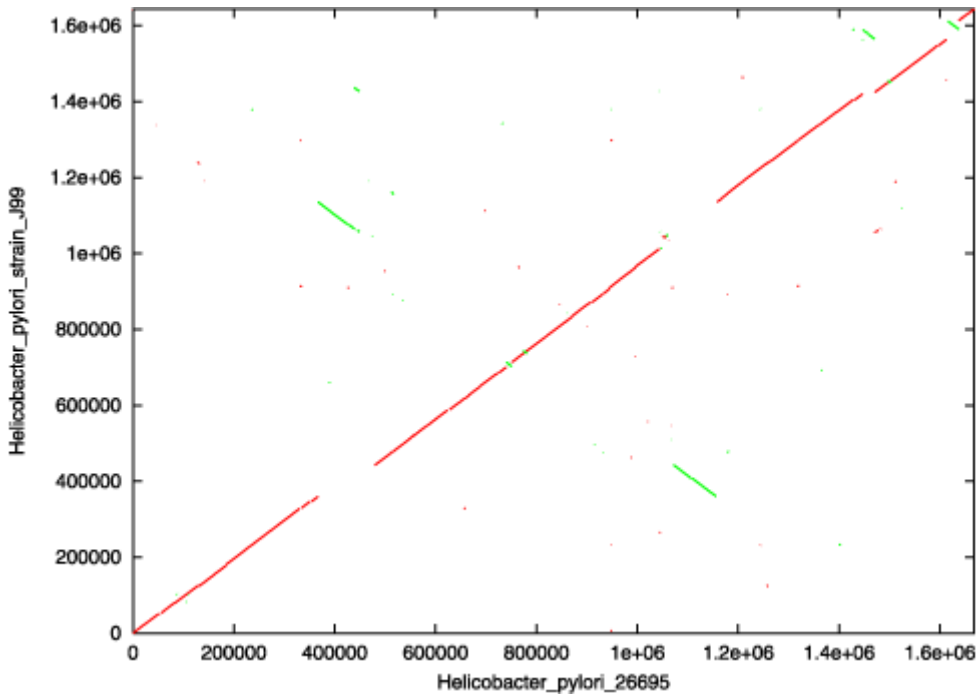


Εικόνα 6: Δέντρο Επιθέματος για την συμβολοσειρά S=[xabxa].

Γενικότερα, τα Suffix Trees είναι οι πιο πολύπλευρες δομές δεδομένων, καθώς μπορούν να χρησιμοποιηθούν σε πολλές εφαρμογές. Μερικές από αυτές είναι:

- Να λύνουν το πρόβλημα με τις σειρές στην διόρθωση κειμένου.
- Να αναζητούν κείμενα.
- Εφαρμογές στην υπολογιστική βιολογία.
- Εφαρμογές στην βιοπληροφορική.
- Τα Δέντρα Επιθέματος μπορούν να δώσουν απαντήσεις σε δύσκολα ερωτήματα πάνω στο μοτίβο του DNA.
- Τα Δέντρα Επιθέματος μπορούν να δώσουν απαντήσεις σε δύσκολα ερωτήματα στις αλληλουχίες πρωτεΐνης.
- Αναζητήσεις στο Διαδίκτυο.
- Εύρεση του μακρύτερου επαναλαμβανόμενης υπό-συμβολοσειράς.
- Βρίσκουν τη μεγαλύτερη κοινή υπό-συμβολοσειρά.
- Εύρεση του μακρύτερου palindrome σε μια συμβολοσειρά.

Για παράδειγμα, τα Suffix Tree χρησιμοποιούνται από ένα κοινώς διαδεδομένο πρόγραμμα για εύρεση κοινών αλληλουχιών σε συμβολοσειρές DNA,RNA και πρωτεΐνες, το Mummer[27].



Στο παραπάνω σχήμα είναι το αποτέλεσμα σύγκρισης δύο συμβολοσειρών-γονιδίων. Όπου υπάρχει ταύτιση μεταξύ των δύο γονιδίων παρουσιάζεται η κόκκινη γραμμή που δημιουργείται από σημεία και όποτε υπάρχει ανάποδη ταύτιση μεταξύ τους δημιουργείται η πράσινη αντίστοιχα. Με το τρόπο αυτό γίνεται πολύ εύκολη η ανίχνευση του ποσοστού ομοιότητας των δύο συμβολοσειρών και των σημείων στα οποία είναι κοινές.

3.2 Suffix Array

Όπως αναφέρθηκε παραπάνω, τα Suffix Arrays δημιουργήθηκαν από τους Mamber και Myers σε μια προσπάθεια δημιουργίας μιας δομής που θα αποτελούσε υποκατάστατο των Suffix Trees και θα έλυνε το πρόβλημα του μεγάλου μεγέθους μνήμης που αυτά απαιτούσαν. Παρ 'όλα αυτά, αποδείχθηκε πως τα Suffix Trees και τα Suffix Arrays συνδέονται στενά. Τα Suffix Arrays από μόνα τους δεν προσφέρουν όλες τις διεργασίες που μπορούν να εκτελέσουν τα Suffix Trees[28]. Πλέον, ο βέλτιστος ως προς χρόνο και χώρο

αλγόριθμος για δημιουργία Suffix Array είναι από τους Huo, Li και Li (2016) [29].

Παρακάτω παρουσιάζεται ένα σύντομο παράδειγμα της βασικής λογικής πίσω από την δημιουργία ενός Suffix Array για την συμβολοσειρά «abracadabra» [30]:

Text	a	b	r	a	c	a	d	a	b	r	a
Index	0	1	2	3	4	5	6	7	8	9	10

Αρχικά, αναθέτουμε σε κάθε prefix της συμβολοσειράς μια τιμή που δεικτοδοτεί την αρχική θέση του στην συμβολοσειρά:

Suffix											Index
a	b	r	a	c	a	d	a	b	r	a	0
	b	r	a	c	a	d	a	b	r	a	1
		r	a	c	a	d	a	b	r	a	2
			a	c	a	d	a	b	r	a	3
				c	a	d	a	b	r	a	4
					a	d	a	b	r	a	5
						d	a	b	r	a	6
							a	b	r	a	7
								b	r	a	8
									r	a	9
										a	10

Στην συνέχεια ταξινομούνται τα prefixes που συμβολίζονται με τους δείκτες που τους δώσαμε πριν με αλφαβητική σειρά και παράγουμε το Suffix Array:

Sorted Suffix											Index
a											10
a	b	r	a								7
a	b	r	a	c	a	d	a	b	r	a	0
a	c	a	d	a	b	r	a				3
a	d	a	b	r	a						5
b	r	a									8
b	r	a	c	a	d	a	b	r	a		1
c	a	d	a	b	r	a					4
d	a	b	r	a							6
r	a										9
r	a	c	a	d	a	b	r	a			2

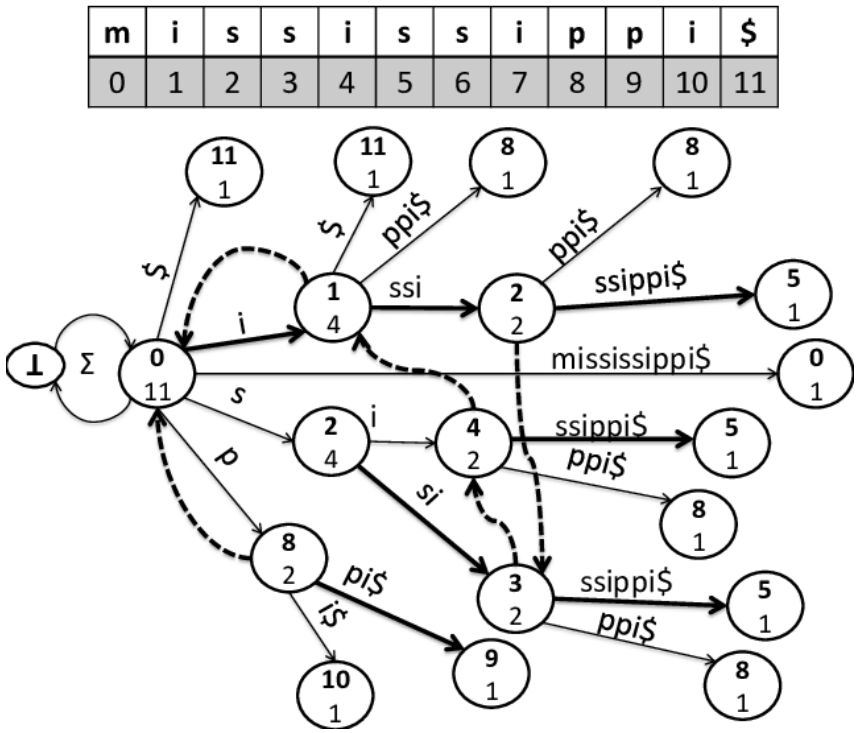
10	7	0	3	5	8	1	4	6	9	2

Εικόνα 7: Παράδειγμα Suffix Array

Αξιοσημείωτο είναι να σημειωθεί πως αν και, όπως είπαμε προηγουμένως, τα Suffix Arrays δεν διατηρούν τις λειτουργίες των Suffix Trees, έχουν βρεθεί τρόποι «θυσιάζοντας» αποθηκευτικό χώρο να έχουν την πλήρη λειτουργικότητα των τελευταίων, όπως τα Enhanced Suffix Array[31].

3.3 Compressed Suffix Trees

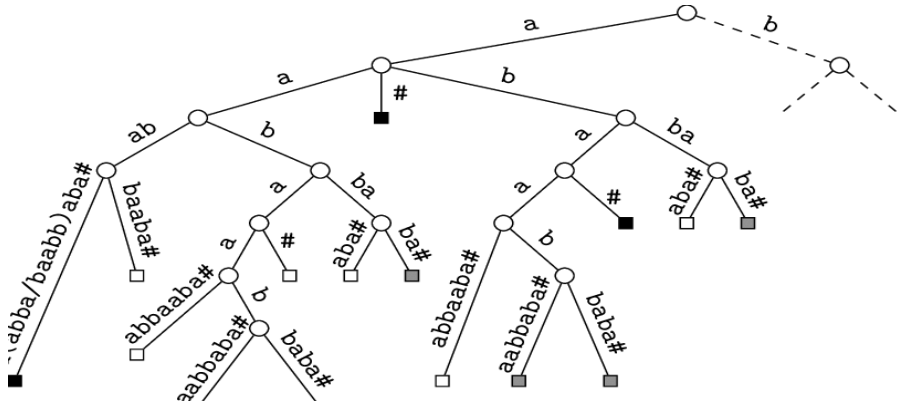
Ήταν ευρέως γνωστό ότι τα Suffix Trees κατανάλωναν μεγάλο χώρο, καθώς για μία μεγάλη συμβολοσειρά κατανάλωναν δέκα με είκοσι bytes για κάθε χαρακτήρα. Για να αντιμετωπιστεί αυτό το πρόβλημα προσπάθησαν να αναπαραστήσουν τα δέντρα επιθέματος με συμπιεσμένες δομές δεδομένων. Έτσι, οδηγήθηκαν στην δημιουργία των Compressed Suffix Trees (CST, Συμπιεσμένα Δέντρα Επιθέματος). Ο Sadakane , παρουσίασε ένα CST, το οποίο λειτουργεί σε μικροδευτερόλεπτα ακόμη και νανοδευτερόλεπτα, χρησιμοποιώντας μόλις 12 bpc (bits per character) για απλές εφαρμογές. Το πιο μικρό CST, σήμερα, λέγεται Fully Compressed Suffix Tree (FCST, Πλήρως Συμπιεσμένο Δέντρο Επιθέματος) το οποίο χρησιμοποιεί 5 bpc. Είναι γεγονός ότι, εάν υπάρξουν και άλλες μειώσεις στην παρουσίαση του FCST, θα υπάρξουν μειώσεις και στο υλικό μέρος, στην επικοινωνία αλλά και στα ενεργειακά κόστη, όταν εφαρμόζονται πολύπλοκες αναζητήσεις σε μεγάλες γονιδιωματικές βάσεις δεδομένων . [32]



Εικόνα 8: Compressed Suffix Tree για την λέξη "mississipi".

3.4 Suffix Tree of an Alignment

Το Suffix Tree of an Alignment(STA, Δέντρο Επιθέματος Ευθυγράμμισης), επιτρέπει συγκεκριμένους διαχωρισμούς στις ετικέτες οι οποίες βρίσκονται στην άκρη του Suffix Tree, με σκοπό την μείωση του μεγέθους του δέντρου, καθώς μπορούν να διατηρούν την λειτουργικότητά του. Όμως, το STA δεν έχει εφαρμοστεί, καθώς χωρίζει την ακολουθία σε μεγάλες περιοχές , οι οποίες μπορούν να διακριθούν η μία από την άλλη. Η εξάρτησή του αυτή, κάνει το STA ευαίσθητο σε μικρές αλλαγές είτε στην ακολουθία είτε στις περιοχές, που μπορεί να εμποδίσει την δυνατότητα κλιμάκωσής του. [33]



Εικόνα 9: Suffix Tree of an Aligment για την συμβολοσειρά "aaabaa".

3.5 Compressed Suffix Arrays

Το Compressed Suffix Array(CSA, Συμπιεσμένος Πίνακας Επιθέματος), είναι μία δομή δεδομένων συμπίεσης την οποία παρουσίασαν οι Grossi και Vitter. Πιο συγκεκριμένα, το CSA είναι ένα ευρετήριο κειμένου, του οποίου η λειτουργικότητα μοιάζει με των Suffix Array. Δεδομένου ενός κειμένου X χαρακτήρων n από ένα αλφάβητο Σ ένα CSA, υποστηρίζει την αναζήτηση αυθαίρετων μοτίβων στο X. Για ένα μοτίβο εισαγωγής P χαρακτήρων m, ο χρόνος αναζήτησης είναι συνήθως O (m) ή O (m + log (n)). Ο χρόνος και ο χώρος για την κατασκευή ενός πίνακα συμπιεσμένου επιθήματος είναι συνήθως O (n). [34] Οι Grossi και Vitter, διευθέτησαν το πρόβλημα του εάν είναι δυνατό να επιτευχθεί ταυτόχρονη γρήγορη απόδοση και να σπάσει το φράγμα του διαστήματος (n log n). Χρησιμοποιήθηκε μια ιεραρχική αποσύνθεση του Suffix Array με βάση τη συνάρτηση γειτονικού Φ, η οποία ορίζεται ως εξής:

$$\Phi(i) = j, \text{ if } SA[j] = (SA[i] + 1) \bmod n$$

Αρχικά, ομαδοποιούμε εννοιολογικά όλα τα επίθημα με βάση το πρόθεμα 1-συμβόλου, ας πούμε x, το καθένα των επιθημάτων, έτσι ώστε όλα τα επίθημα σε μια ομάδα να ξεκινούν με το χαρακτήρα x. Το λέμε αυτό εννοιολογική ομάδα μια ομάδα x. Για το παράδειγμα στο σχήμα 1, τα πρώτα τέσσερα μικρότερα επίθημα αντιστοιχούν στην ομάδα a, αφού αυτά τα επίθημα έχουν τον πρώτο χαρακτήρα a. Προφανώς, το πρόθεμα 1-συμβόλου, x, των επιθημάτων που αντιστοιχούν στην ομάδα x δεν συνεισφέρει στο σχετική σειρά τω επιθημάτων. Εάν αφαιρέσουμε το πρόθεμα 1-συμβόλου x από τα

επίθημα στο x-group, τότε η σχετική σειρά των επιθημάτων που προκύπτουν δεν αλλάζει, αν και μπορεί να μην βρίσκεστε πλέον σε γειτονικό τμήμα. Καλούμε, τη συλλογή των επιθημάτων που προκύπτουν αποκτήθηκε από την ομάδα x μια λίστα x. Με άλλα λόγια, εάν μια ομάδα x περιέχει θέσεις επιθήματος

k = SA [i] και h = SA [i + 1] (αντιστοιχεί στα επίθημα T [k..n] και T [h..n]

αντίστοιχα, και

T [k] = T [h] = x), και T [SA [p] .. n] = T [k + 1..n] και T [SA [q] .. n] = T [h + 1..n] και μετά p <q.

Για παράδειγμα, η α-ομάδα περιέχει τέσσερα επίθημα με τις τάξεις 0, 1, 2, 3, αντίστοιχα. Όλα αυτά τα επίθημα ξεκινούν με ένα, καταργούμε ένα από αυτά τα επίθημα και λαμβάνουμε τέσσερα νέα επίθημα, των οποίων η κατάταξη σχηματίζει μια αυξανόμενη ακολουθία θέσεων, δηλαδή, 6, 14, 17 και 23. Η συλλογή αυτών των x-λιστών είναι ακριβώς οι Σ λίστες που εισήγαγαν οι Grossi και Vitter[35]:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
T	a	b	f	g	d	b	f	b	g	d	f	c	c	b	g	a	c	e	f	c	e	g	c	d	e	f	g	b	f	c	a	d	b	g	a	f
SA	0	15	30	34	5	27	1	13	32	7	29	12	11	22	16	19	4	31	23	9	17	24	20	35	6	28	10	18	25	2	14	33	26	21	3	8
Φ	6	14	17	23	24	25	29	30	31	35	2	7	11	18	20	22	4	8	21	26	27	28	33	0	9	10	12	15	32	34	1	3	5	13	16	19

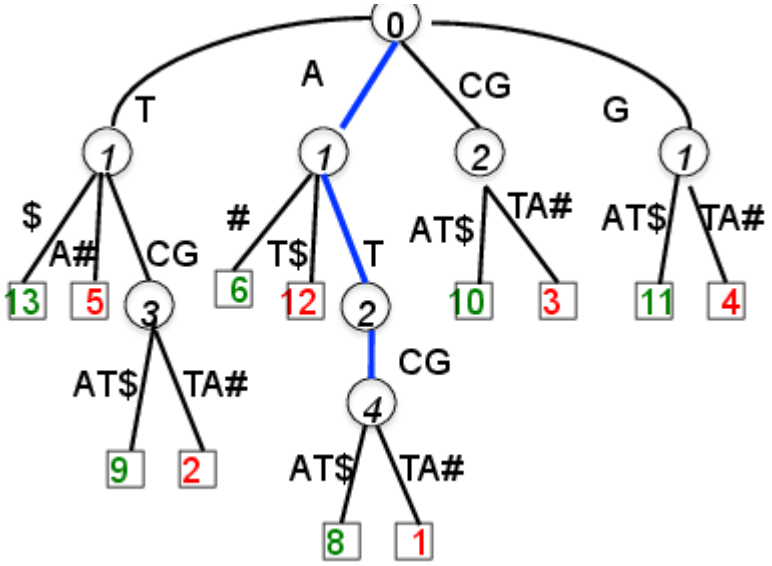
Εικόνα 10: Suffix Array με την γειτονική συνάρτηση Φ.

3.6 Generalized Suffix Trees

Για να επιτευχθεί η εύρεση της μέγιστης κοινής υπο-συμβολοσειράς δύο ή περισσότερων συμβολοσειρών, χρησιμοποιούνται τα Generalized Suffix Trees. Η λογική είναι πως μετατρέπουμε όλες τις συμβολοσειρές για τις οποίες θέλουμε να βρούμε την μέγιστη κοινή συμβολοσειρά σε μία ενιαία συμβολοσειρά. Για να σηματοδοτήσουμε το τέλος κάθε συμβολοσειράς, ορίζουμε ένα ξεχωριστό σύμβολο για την κάθε μία τους και δημιουργούμε το Suffix Tree.

Για παράδειγμα χρησιμοποιούμε τις συμβολοσειρές: ATCGTA και ATCGAT. Και τις φτιάχνουμε σε μία συμβολοσειρά: ATCGTA#ATCGAT$

```
ATCGTA#
      # 7
     A# 6
    TA# 5
   GTA# 4
  CGTA# 3
 TCGTA# 2
ATCGTA# 1

ATCGAT$
      $ 14
     T$ 13
    AT$ 12
   GAT$ 11
  CGAT$ 10
 TCGAT$ 9
ATCGAT$ 8
```



Εικόνα 11:Παράδειγμα Generalized Suffix Tree

Έτσι για να βρούμε της μέγιστης κοινής υπο-συμβολοσειράς ψάχνουμε για το μονοπάτι με το μεγαλύτερο βάθος στο οποίο υπάρχουν φύλλα που ανήκουν και στις δύο προ-αναφέρουσες συμβολοσειρές.

Με αυτήν την δόμηση των συμβολοσειρών διευκολύνεται και η πραγματοποίηση άλλων λειτουργιών, όπως «Τελευταίος κοινός πρόγονος» και «Εύρεση μέγιστου Palindrome» [36].

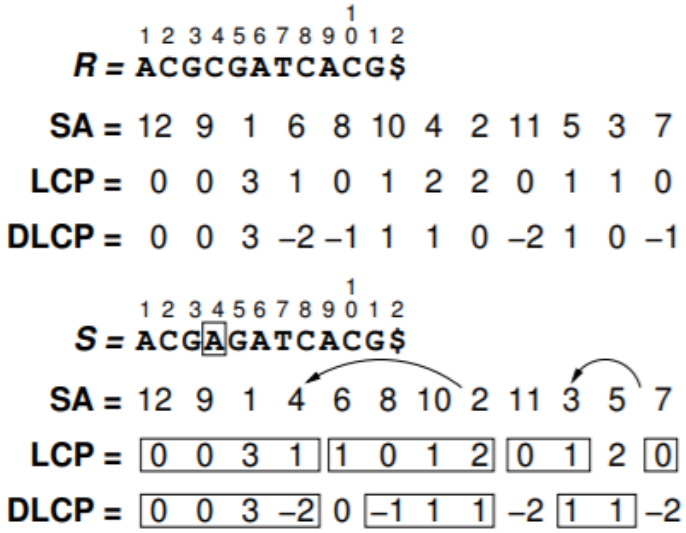
3.7 Relative Suffix Trees

Το Relative Suffix Tree(RST, Σχετικό Δέντρο Επιθέματος), είναι σχετικό με το Συμπιεσμένο Δέντρο Επιθέματος(CST). Το οποίο παρουσίασαν οι Andrea Farruggia, Travis Gagie, Gonzalo Navarro, Simon J. Puglisi και Jouni Siren , στην εργασία τους με τίτλο «Relative Suffix Trees», η οποία εκδόθηκε το τον Μάϊο του 2018.Αποτελείται από το FM-index και το LRCP (relative longest common prefix, σχετικό μακρύτερο κοινό πρόθεμα) πίνακα το οποίο βασίζεται σε σχετική συμπίεση Lempel-Ziv(RLZ). Το RST εκμεταλλεύεται την επανάληψη από τα κείμενα σε μία συλλογή ενώ παράλληλα προσφέρει ένα CST, για κάθε κείμενο. [37] Η προαιρετική σχετική δομή μπορεί να δημιουργηθεί ή να φορτωθεί από δίσκο για να επιταχύνει τους αλγόριθμους με βάση την αναζήτηση προς τα εμπρός.

Το Σχετικό Δέντρο Επιθέματος παράγει μία ακολουθία φράσεων:

$$w_i = (p_i, l_i, c_i)$$

Με μέγιστο μέτρο φράσεων 1024 και το $|c_i| > 0$ για κάθε i.



Εικόνα 12: Παράδειγμα Relative Suffix Tree

3.8 Grammar Compressed Topology

Μία από αυτές τις εκδοχές παρουσιάστηκε το 2016 από τους Navarro και Pereira στο paper «Faster Compressed Suffix Trees for Repetitive Collections» με όνομα Grammar-Compressed Topology (GCT). Η εκδοχή αυτή πετυχαίνει πολύ μικρό μέγεθος απαιτούμενου αποθηκευτικού χώρου για γονίδια που έχουν μεγάλο ποσοστό επανάληψης και πολύ καλύτερους χρόνους στις διεργασίες που εκτελεί. Η εκδοχή αυτή είναι βασισμένη στο Succinct Tree[38] και διατηρεί όλες τις λειτουργίες του. Η γραμματική συμπίεση που χρησιμοποίησαν υλοποιήθηκε με τον αλγόριθμο RePair των Larsson και Moffat[39], που διαδοχικά ορίζει ένα νέο γράμμα για τα δυο πιο συχνά εμφανιζόμενα ζεύγη χαρακτήρων έως ότου το τελευταίο ζεύγος να παρουσιάζεται μόνο μία φορά.

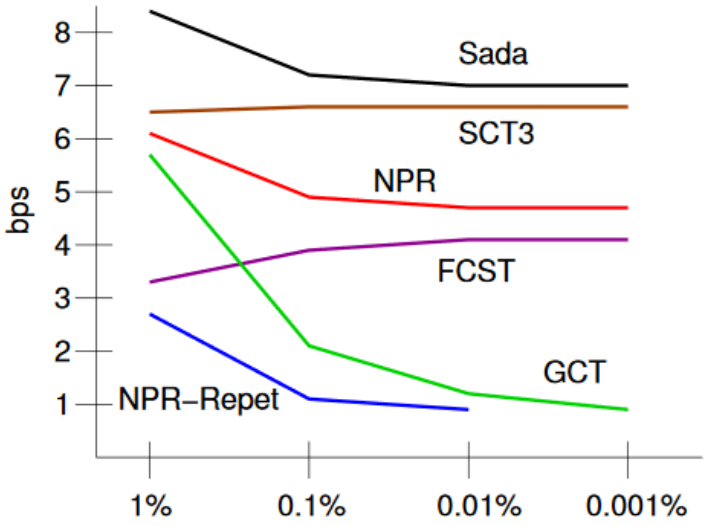


Fig. 20: Approximate space figures for the different CSTs as repetitiveness increases on the synthetic collections. For the space of GCT we take the sweet point, whereas for the others we show their minimum space in the plots.

Operation	Time (μs)	NPR-Repet	FCST	NPR	SCT3	Sada
fChild	3–10	2–3	2–3	2	(1)	(2–3)
tDepth	5–10	2–4	3–4	2–4	0–1	(2)
nSibling	10–50	2	2	1	(1–2)	(1–2)
parent	10–40	2	2	1	(1)	(3–4)
tAncestor	10–30	3–4	3–4	2–3	1–2	(1–2)
LCA	30–100	1–2	(1)	1	(1)	(1)
sLink	60–300	1–2	0–1	0–1	(1–2)	(1–2)
sDepth	50–100	0–1	1	0	0	0
sAncestor	250–700	0–1	2–3	0	0–1	0
letter	4–10	0	0	(1)		1
child	300–1000	1–2	1	0–1	(0–2)	(0–1)

Table II: Operation time ranges for the GCT and orders of magnitude of difference with alternative CSTs (the other structure is slower by that order, unless the number is in parentheses, in which case it is faster by that order). The space increases left to right, in general terms.

Όπως φαίνεται από τα παραπάνω σχεδιαγράμματα, ο απαιτούμενος χώρος εμφανίζει σημαντική μείωση όσο αυξάνεται η «επαναληπτικότητα» της συμβολοσειράς, όπως είχε προβλεφθεί, ιδιότητα την οποία δεν εμφανίζουν άλλες εκδοχές και επομένως σε συμβολοσειρές όπως τα γονίδια, που χαρακτηρίζονται από την επανάληψη prefixes, η εξοικονόμηση χώρου είναι πολύ μεγάλη. Επίσης, από τον παραπάνω πίνακα είναι εμφανής η σημαντική βελτίωση του χρόνου, για τις περισσότερες περιπτώσεις, για εκτέλεση κάποιων διεργασιών[40].

4 Συμπεράσματα

Τα Suffix Trees και τα Suffix Arrays, παρά το ότι είχαν πρωτοεμφανιστεί πριν από σχεδόν μισό αιώνα, αποτελούν δομές που μέχρι και σήμερα χρησιμοποιούνται ευρέως σε πολλούς τομείς. [41] Υπάρχουν πάρα πολλές τροποποιήσεις των δομών από τις οποίες μπορεί κανείς να διαλέξει ώστε να βρει τον βέλτιστο αλγόριθμο που να ταιριάζει καλύτερα στο πρόβλημα που θέλει να λύσει. Κλασικό παράδειγμα αποτελεί η χρήση ειδικών Suffix Trees και Arrays στην επιστήμη της βιολογίας, που ευνοούνται σε πολύ μεγάλο βαθμό από την επανάληψη επιθεμάτων(prefixes), πολλές φορές μέσα στην συμβολοσειρά και αυτός είναι ο λόγος για τον οποίο είναι κοινώς χρησιμοποιούμενα στην αναπαράσταση γονιδίων. Το «αλφάβητο» των γονιδίων αποτελείται από τέσσερα γράμματα, το C,U,A και G και επομένως είναι προφανές πως λόγω του πολύ μεγάλου μήκους της συμβολοσειράς-γονιδίου, υπάρχουν πολλά σε πλήθος αλλά και μεγάλα σε μέγεθος, επαναλαμβανόμενα επιθέματα. Τέλος είναι αξιοσημείωτο, ότι οι συνεχείς διαφοροποιήσεις των μορφών των δόμων αυτών, αποτελούν αποδεικτικό του ότι πρόκειται για μία «ζωντανή» δομή η οποία έχει αντέξει και θα αντέξει στον χρόνο και θα συνεχίζει να εξελίσσεται, όπως έχει κάνει μέχρι τώρα, για να έχουμε την βέλτιστη εκδοχή του για κάθε πρόβλημα.

Αναφορές

[1]https://el.wikipedia.org/wiki/%CE%94%CE%BF%CE%BC%CE%AE_%CE%B4%CE%B5%CE%B4%CE%BF%CE%BC%CE%AD%CE%BD%CF%89%CE%BD

[2]http://sylogos7oulykeioy.weebly.com/uploads/9/9/7/7/9977539/kef3.pdf

[3] https://en.wikipedia.org/wiki/Trie

[4] Βιβλίο:Richardo Baeza-Yates & Berthier Ribeiro-Neto Ανάκτηση Πληροφορίας, Σελίδα 339.

[5] https://www.techiedelight.com/trie-implementation-insert-search-delete/

[6] https://web.cs.elte.hu/blobs/diplomamunkak/msc_alkmat/2013/vasarhelyi_balint_mark.pdf

[7] Βιβλίο:Richardo Baeza-Yates & Berthier Ribeiro-Neto Ανάκτηση Πληροφορίας, Σελίδα 342.

[8] https://en.wikipedia.org/wiki/Suffix_array

[9] Βιβλίο:Richardo Baeza-Yates & Berthier Ribeiro-Neto Ανάκτηση Πληροφορίας, Σελίδα 344

[10] R. M. Karp, R. E. Miller, and A. L. Rosenberg. Rapididentification of repeated patterns in strings, trees andarrays. InProceedings of the 4th ACM Symposium onthe Theory of Computing, pages 125–136, Denver, CO,**1972**

[11] P. Weiner. Linear pattern matching algorithms. InProceedings of the 14th Annual IEEE Symposium onSwitching and Automata Theory, pages 1–11,Washington, DC, **1973**

[12] E. M. McCreight. A space-economical suffix treeconstruction algorithm.J. Algorithms, 23(2):262–272,**1976**.

[13] M. Rodeh, V. Pratt, and S. Even. Linear algorithm fordata compression via string matching.J. Assoc.Comput. Mach., 28(1):16–24, **1981**.

[14] A. Apostolico. The myriad virtues of suffix trees. InA. Apostolico and Z. Galil, editors,CombinatorialAlgorithms on Words, volume 12 ofNATO AdvancedScience Institutes, Series F, pages 85–96.Springer-Verlag, Berlin, **1985**.

[15] S. Kurtz. Reducing the space requirements of suffixtrees.Softw. Pract. Exp., 29(13):1149–1171, **1999**

[16] M. Burrows and D. J. Wheeler. A block-sortinglossless data compression algorithm. Technical Report124, Digital Equipments Corporation, May **1994**.

[17] U. Manber and G. Myers. Suffix arrays: a new methodfor on-line string searches. InProceedings of the 1stACM-SIAM Annual Symposium on DiscreteAlgorithms, pages 319–327, San Francisco, CA, **1990**.

[18] P. Ferragina and G. Manzini. Opportunistic data structures with applications. InFOCS, pages 390–398,**2000**.

[19] R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In Proceedings ACM Symposium onthe Theory of Computing, pages 397–406, Portland,Oregon, **2000**

[20] T. Kasai, G. Lee, H. Arimura, S. Arikawa, andK. Park. Linear-time longest-common-prefixcomputation in suffix arrays and its applications. InCPM, pages 181–192. Springer-Verlag, **2001**

[21] G. Nong, S. Zhang, and W. H. Chan. Two efficientalgorithms for linear time suffix array construction.IEEE Trans. Comput., 60(10):1471–1484, **2011**.

[22] Sadakane, Kunihiko. "Compressed suffix trees with full functionality." Theory of Computing Systems 41.4 (2007): 589-607.

[23] Fischer, Johannes, and Volker Heun. "A new succinct representation of RMQ-information and improvements in the enhanced suffix array." International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies. Springer, Berlin, Heidelberg, 2007.

[24] Russo, Luís MS, Gonzalo Navarro, and Arlindo L. Oliveira. "Fully compressed suffix trees." ACM transactions on algorithms (TALG) 7.4 (2011): 1-34.

[25] Navarro, Gonzalo, and Luís MS Russo. "Fast fully-compressed suffix trees." 2014 Data Compression Conference. IEEE, 2014.

[26] Βιβλίο Βάσεις Δεδομένων σελ. 161.

[27] http://mummer.sourceforge.net/manual/

[28] Apostolico, Alberto, et al. "40 years of suffix trees." *Communications of the ACM* 59.4 (2016): 66-73.

[29] Li, Zhize, Jian Li, and Hongwei Huo. "Optimal in-place suffix sorting." *International Symposium on String Processing and Information Retrieval*. Springer, Cham, 2018.

[30] http://sary.sourceforge.net/docs/suffix-array.html

[31] Abouelhoda, Mohamed Ibrahim, Stefan Kurtz, and Enno Ohlebusch. "The enhanced suffix array and its applications to genome analysis." *International Workshop on Algorithms in Bioinformatics*. Springer, Berlin, Heidelberg, 2002.

[32] Farruggia, Andrea, et al. "Relative suffix trees." The computer journal 61.5 (2018): 773-788.

[33] Farruggia, Andrea, et al. "Relative suffix trees." The computer journal 61.5 (2018): 773-788.

[34] https://en.wikipedia.org/wiki/Compressed_suffix_array#Open_Source_Implementations

[35] https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1041.7659&rep=rep1&type=pdf

[36] https://www.cs.tau.ac.il/~rshamir/algmb/presentations/Suffix-Trees.pdf

[37] Farruggia, Andrea, et al. "Relative suffix trees." The computer journal 61.5 (2018): 773-788.

[38] Jacobson, Guy. "Succinct static data structures." *Ph. D. thesis, Carnegie Mellon University* (1988).

[39] *N. Jesper Larsson and Alistair Moffat. 1999. Offline Dictionary-Based Compression. In Proceedings of the Conference on Data Compression (DCC '99). IEEE Computer Society, USA, 296*

[40] Navarro, Gonzalo, and Alberto Ordóñez Pereira. "Faster compressed suffix trees for repetitive collections." *Journal of Experimental Algorithmics (JEA)* 21 (2016): 1-38.

[41] Apostolico, Alberto, et al. "40 years of suffix trees." *Communications of the ACM* 59.4 (2016): 66-73.