

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

**ΕΙΣΑΓΩΓΗ ΣΤΗ ΒΙΟΠΛΗΡΟΦΟΡΙΚΗ**

**ΠΡΩΤΟ ΣΥΝΟΛΟ ΕΡΓΑΣΤΗΡΙΑΚΩΝ ΑΣΚΗΣΕΩΝ 2023-2024**

Ασπασία Τζερμιά

Α.Μ. 1067455

<Νίκος Αυγέρης>

Α.Μ.

Πάτρα, 2024

**Ερώτημα 3**

Οι k συμβολοσειρές μπορούν να αποθηκευτούν σε ένα γενικευμένο δέντρο επιθεμάτων σαν μια ενιαία συμβολοσειρά που θα είναι η συνένωση όλων των συμβολοσειρών, όπου ανάμεσα σε κάθε μία από τις συμβολοσειρές μπορεί να παρεμβάλλεται κάποιος ειδικός χαρακτήρας που δεν ανήκει στο αλφάβητό τους (για παράδειγμα ο χαρακτήρας “$”). Δηλαδή για δεδομένο σύνολο με τις συμβολοσειρές x1, x2, …, xk, το δέντρο θα κατασκευαστεί από τη συμβολοσειρά x1$x2$...xk$. Για να μπορούν να προστίθενται ή να αφαιρούνται από αυτό συμβολοσειρές πρέπει να μπορεί να διατηρεί και να προσαρμόζει «δυναμικά» τη δομή του, δηλαδή να είναι δυναμικό δέντρο επιθεμάτων (dynamic suffix tree). Πρέπει επίσης να μπορεί να γίνεται αναζήτηση για ταιριάσματα συμβολοσειρών στο δέντρο που δεν έχει σταθερή δομή (dynamic string matching).

Για τις παραπάνω λειτουργίες απαιτείται όμως περισσότερος χρόνος (για αναδιοργάνωση της δομής μετά από κάθε εισαγωγή ή διαγραφή και για αναζήτηση στο δέντρο) και χώρος (γιατί η κατάσταση του δέντρου πριν και μετά από κάθε αλλαγή πρέπει να διατηρείται στη μνήμη του υπολογιστή). Η αναζήτηση για ταιριάσματα μπορεί να γίνει πολύ χρονοβόρα, ειδικά όταν πρόκειται για βιολογικές ακολουθίες, που είναι μεγάλες σε μέγεθος. Επιπλέον, η διαγραφή μιας συμβολοσειράς xj γίνεται ουσιαστικά με διαγραφή των φύλλων στο δέντρο που αντιστοιχούν στις υποσυμβολοσειρές της xj, όμως η κατάσταση των εσωτερικών κόμβων του δέντρου δεν ανανεώνεται αμέσως στη μνήμη και έτσι ο συνολικός χώρος για την αποθήκευση της δομής αυξάνεται.

Μια λύση για βελτίωση της χρονικής και χωρικής πολυπλοκότητας είναι να αποθηκεύονται οι k συμβολοσειρές ως γραμμές ή στήλες ενός μητρώου που θα αποθηκεύεται στη μνήμη. Οι εισαγωγές/διαγραφές συμβολοσειρών μπορούν να γίνονται με εισαγωγή/διαγραφή γραμμών ή στηλών. Κάθε υποσυμβολοσειρά, που κατά την αναζήτηση ταιριάσματος θα είναι ουσιαστικά ένα υπομητρώο του μητρώου αυτού, θα αποθηκεύεται σε ένα ξεχωριστό dynamic suffix tree.

Μια άλλη λύση είναι τα dynamic Fully Compressed Suffix Trees (FCSTs). Αποτελούν βελτίωση των στατικών Compresses Suffix Trees για δυναμικά μεταβαλλόμενα δεδομένα. Ένα FCST αποτελείται από έναν πίνακα Compressed Suffix Array (που χρησιμοποιείται ως ευρετήριο για τις συμβολοσειρές που είναι αποθηκευμένες σε αυτό), από ένα δ-Sampled tree (που εκμεταλλεύεται τις ιδιότητες των suffix trees ώστε να μπορούν να αποθηκεύονται μόνο κάποιοι κρίσιμοι κόμβοι και να μη χρειάζεται αποθήκευση ολόκληρης της δενδρικής δομής) και από αντιστοιχίσεις ανάμεσα στις δύο αυτές δομές. Το FCST μεταβάλλεται με χρήση ενός κανόνα που τοποθετεί νέα στοιχεία από δεξιά προς τα αριστερά δημιουργώντας ένα suffix tree ως «ενδιάμεση» κατασκευή. Έτσι μπορούν να αναπαρασταθούν όλο και μεγαλύτερες συμβολοσειρές. Κάθε αλλαγή στο δέντρο γίνεται μόνο στα φύλλα και αλλαγές στους εσωτερικούς κόμβους γίνονται μόνο αν χρειάζεται. Τα δέντρα αυτά πετυχαίνουν βέλτιστη χωρική πολυπλοκότητα.

Πηγές:

1. Y. Choi, T.W. Lam, Dynamic suffix tree and two-dimensional texts management, Information Processing Letters, Volume 61, Issue 4, 1997, Pages 213-220, ISSN 0020-0190, <https://doi.org/10.1016/S0020-0190(97)00018-5> .
2. Russo, L.M.S., Navarro, G., Oliveira, A.L. (2008). Dynamic Fully-Compressed Suffix Trees. In: Ferragina, P., Landau, G.M. (eds) Combinatorial Pattern Matching. CPM 2008. Lecture Notes in Computer Science, vol 5029. Springer, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-540-69068-9_19>

**Ερώτημα 4**

Ως «κενό» ανάμεσα σε δύο εμφανίσεις μιας συμβολοσειράς μπορεί να οριστεί ο αριθμός των συμβόλων ανάμεσά τους. Ένα ζευγάρι εμφανίσεων είναι μέγιστο (maximal) όταν καμία από τις δύο δεν μπορεί να επεκταθεί προς τα δεξιά ή προς τα αριστερά χωρίς να πάψει να υπάρχει ταίριασμα μεταξύ τους. Ένα ζευγάρι είναι αριστερά μέγιστο (left-maximal) αν τα σύμβολα στα αριστερά των δύο εμφανίσεων είναι διαφορετικά και αντίστοιχα δεξιά μέγιστο (right-maximal) αν τα σύμβολα στα δεξιά είναι διαφορετικά. Έχει σημασία να προσδιοριστεί τι θα θεωρείται «κενό» ανάμεσα σε δύο εμφανίσεις για να φαίνεται πότε υπάρχει επικάλυψη και πότε δύο εμφανίσεις βρίσκονται σε μεγάλη απόσταση μεταξύ τους μέσα στη συμβολοσειρά. Σε κάποιες περιπτώσεις μπορεί μια συμβολοσειρά να επαναλαμβάνεται δύο διαδοχικές φορές, δηλαδή το κενό έχει μήκος 0 (tandem repeat).

Μια προσέγγιση [1] που βάζει περιορισμούς στα κενά ανάμεσα σε δύο εμφανίσεις της ίδιας συμβολοσειράς θεωρεί ότι ένα ζευγάρι εμφανίσεων συμβολοσειράς α είναι μέγιστο μόνο αν το κενό βρίσκεται στο διάστημα ανάμεσα σε δύο συναρτήσεις g1(|a|) και g2(|a|). Οι υποσυμβολοσειρές αποθηκεύονται σε ένα δυαδικό δέντρο επιθεμάτων. Τα μέγιστα ζευγάρια όλης της συμβολοσειράς μπορούν να βρεθούν αφού πρώτα βρεθούν όλα τα δεξιά-μέγιστα ζευγάρια στο δέντρο με κενά στο διάστημα ανάμεσα στις δύο συναρτήσεις και στη συνέχεια αφαιρεθούν από αυτά όλα τα δεξιά μέγιστα που δεν είναι και αριστερά μέγιστα. Για να μπορεί να γίνει αναζήτηση, χρησιμοποιούνται δέντρα αναζήτησης, για παράδειγμα AVL, όπου η πληροφορία αποθηκεύεται στους κόμβους Καλούνται leaf-list trees και κατασκευάζονται βήμα-βήμα με διαπέραση του αρχικού δέντρου και με βάση τις λίστες φύλλων (Leaf Lists) κάθε κόμβου. Για κάθε κόμβο ένα leaf list αποθηκεύει όλες τις υποσυμβολοσειρές που καταλήγουν σε συγκεκριμένο επίθεμα(δηλαδή τους δείκτες προς τα ανίστοιχα φύλλα του δέντρου). Για να μπορεί να γίνει το φιλτράρισμα χωρίς να χρειάζεται να βρεθούν όλα τα δεξιά μέγιστα κατασκευάζεται ένα επιπλέον βοηθητικό AVL δέντρο, το block-start tree, που διατηρεί συνδέσμους προς το leaf- list tree. Τα leaf-list trees αποθηκεύουν την πληροφορία και τα block-start trees αποθηκεύουν το υποσύνολο των στοιχείων που ξεκινούν σε κάθε «μπλοκ». Για τα φύλλα, τα δέντρα αυτά κατασκευάζονται απευθείας. Για τους εσωτερικούς κόμβους κατασκευάζονται με συγχωνεύσεις υποδέντρων. Όμως η ενημέρωση των AVL δέντρων κοστίζει σε χρόνο. Μια εναλλακτική δομή είναι η χρήση δέντρων σωρού (heap trees) και μιας στοίβας στην οποία θα αποθηκεύεται ουσιαστικά το μονοπάτι (backbone) από τη ρίζα προς κάποιο φύλλο που δημιουργείται ακολουθώντας σε κάθε βήμα το «δεξιότερο» παιδί. Οπότε στην περίπτωση αυτή το «κενό» θα περιορίζεται μόνο από ένα κάτω όριο g1(|α|) αντί για διάστημα. Αντίστοιχη είναι και η χρήση χρωματισμένων δέντρων (colored trees). Οι προσεγγίσεις αυτές έχουν ίδια χωρική πολυπλοκότητα χειρότερης περίπτωσης με την προηγούμενη, O(n), αλλά βελτιώνουν τη χρονική πολυπλοκότητα από O(nlogn + z) σε O(n + z), όπου n το μήκος της αρχικής συμβολοσειράς και z ο αριθμός των ζευγών μέγιστων εμφανίσεων.

Μια δεύτερη προσέγγιση [3] γενικεύει την έννοια των ζευγών εμφανίσεων για εμφανίσεις της ίδιας υποσυμβολοσειράς περισσότερες από δύο φορές στην ίδια συμβολοσειρά: τις πολυεμφανίσεις (multirepeats). Θεωρεί ένα σύνολο από N συμβολοσειρές, όπου κάθεμία έχει μήκος n ή (στην περίπτωση που δεν έχουν όλες το ίδιο μήκος) ο μέσος όρος των μηκών τους είναι n. Στόχος είναι να βρεθούν όλες οι επαναλήψεις μιας συμβολοσειράς από αλφάβητο μεγέθους σ που συναντώνται τουλάχιστον m φορές σε κάθε μια από q συμβολοσειρές του συνόλου, χωρίς περιορισμούς στο μήκος των κενών. Μια πολυεμφάνιση F χαρακτηρίζεται left ή right maximal με βάση τα ίδια κριτήρια με προηγουμένως, αν δηλαδή δεν μπορεί να επεκταθεί. Το multirepeat πρέπει να εμφανίζεται τον ίδιο αριθμό φορών σε κάθε συμβολοσειρά για να ληφθεί υπ’ όψη από τον αλγόριθμο. Γίνεται χρήση generalized suffix trees με την παραδοχή ότι είναι δυαδικά. Για να βρεθούν οι πολυεμφανίσεις χωρίς περιορισμούς στα κενά, συγκρίνονται με bottom-up τρόπο αναδρομικά κόμβοι που έχουν κοινό «πατέρα» στο δέντρο και συνενώνονται οι leaf-lists τους. Η χρονική πολυπλοκότητα του αλγορίθμου είναι O(σΝ^2 + z). Αν υπάρχουν περιορισμοί στα κενά, οι leaf-lists κατασκευάζονται ως finger search trees σε φάσεις και με τη βοήθεια ενός γράφου. Στην περίπτωση αυτή η πολυπλοκότητα γίνεται O( (c^2 + σ)Ν^2nlog(nN) + z ), όπου c ο περιορισμός για τα κενά. Οι αλγόριθμοι της παραπάνω προσέγγισης έχουν το μειονέκτημα ότι η χρονική πολυπλοκότητα παραμένει μικρή μόνο όταν το μήκος των κενών είναι μικρό.

Μια τρίτη προσέγγιση [2] εξετάζει την περιοδικότητα (periodicity) συμβολοσειρών. Μια συμβολοσειρά S είναι ημιπεριοδική (quasiperiodic) αν μπορεί να κατασκευαστεί από συνενώσεις και υπερθέσεις μιας μικρότερης συμβολοσειράς α, που λέγεται ότι «καλύπτει» την S. Αλλιώς, αν η S δεν μπορεί να καλυφθεί από κάποια μικρότερη υποσυμβολοσειρά, είναι superprimitive. Παρουσιάζεται ένας αλγόριθμος που βρίσκει τις μέγιστα ημιπεριοδικές υποσυμβολοσειρές σε μια συμβολοσειρά πρώτα με κατασκευή και μετά με bottom-up διαπέραση ενός suffix tree. Ο αλγόριθμος αυτός έχει χρονική πολυπλοκότητα O(nlogn).

Ένα πρόβλημα που φαίνεται να είναι κοινό στις παραπάνω προσεγγίσεις είναι ότι η τελική χρονική πολυπλοκότητα των αλγορίθμων μπορεί να γίνει μεγάλη γιατί η κατασκευή των δέντρων επιθεμάτων εξαρτάται από το μέγεθος και τη συχνότητα των κενών ανάμεσα σε δύο επαναλήψεις μιας συμβολοσειράς.

Μια εναλλακτική προσέγγιση [4] που θα μπορούσε να είναι αποδοτική μπορεί να είναι κάποια παραλλαγή της χρήσης k-μερών, όπου να χρησιμοποιείται ένα ευρετήριο με καταχωρημένες υποσυβολοσειρές της αρχικής συμβολοσειράς και κάθε φορά που προκύπτει νέα εμφάνιση μιας συμβολοσειράς να αυξάνεται ένας δείκτης που θα μετράει τον αριθμό των εμφανίσεών. Σε μια τέτοια παραλλαγή όμως πρέπει κάπως να ληφθούν υπ’ όψη και οι περιορισμοί για τα κενά. Επιπλέον, τα k-μερή είναι ευαίσθητα στις μεταλλάξεις, που στις βιολογικές συμβολοσειρές συμβαίνουν συχνά [5]. Γι’ αυτό ίσως να είναι αποδοτικότερη η μελέτη της κατανομής των k-μερών στις περιοχές της συμβολοσειράς όπου υπάρχουν ταιριάσματα μέσω διανυσμάτων (vector seeds), ή η χρήση με κάποιον τρόπο ομάδων από k-μερή που να είναι διασυνδεδεμένα μεταξύ τους (όπως τα strobemers).

Πηγές:

1. Gerth Stølting Brodal, Rune B. Lyngsø, Christian N. S. Pedersen, Jens Stoye: Finding Maximal Pairs with Bounded Gap. CPM 1999: 134-149
2. Gerth Stølting Brodal, Christian N. S. Pedersen: Finding Maximal Quasiperiodicities in Strings. CPM 2000: 397-411
3. A. Bakalis, Costas S. Iliopoulos, Christos Makris, Spyros Sioutas, Evangelos Theodoridis, Athanasios K. Tsakalidis, Kostas Tsichlas: Locating Maximal Multirepeats in Multiple Strings Under Various Constraints. Comput. J. 50(2): 178-185 (2007).
4. Back to sequences: find the origin of *k*-mers Anthony Baire, Pierre Peterlongo bioRxiv 2023.10.26.564040; doi: <https://doi.org/10.1101/2023.10.26.564040>
5. Sahlin K. Effective sequence similarity detection with strobemers. Genome Res. 2021 Nov;31(11):2080-2094. doi: 10.1101/gr.275648.121. Epub 2021 Oct 19. PMID: 34667119; PMCID: PMC8559714.

**Ερώτημα 5**

Ο αλγόριθμος θα μπορούσε να έχει την παρακάτω περιγραφή:

Ξεκινά από τη ρίζα.

Αρχικοποιεί έναν μετρητή για τους χαρακτήρες σε 0 και έναν άλλο μετρητή για τον αριθμό των εμφανίσεων του προτύπου σε 0. Ξεκινά την διαπέραση του δέντρου χαρακτήρα προς χαρακτήρα.

Για κάθε χαρακτήρα που συναντά, ελέγχει αν είναι ο επόμενος χαρακτήρας του προτύπου (αρχικά έλεγχος αν είναι ο πρώτος χαρακτήρας του προτύπου).

Αν είναι, αυξάνει τον μετρητή χαρακτήρων κατά 1 και συνεχίζει με τον επόμενο χαρακτήρα του κλάδου του δέντρου στον οποίο βρίσκεται.

Αν δεν ανήκει, μηδενίζει τον μετρητή χαρακτήρων και ξεκινά την ίδια διαδικασία από τον χαρακτήρα στον οποίο βρίσκεται.

Αν ο μετρητής γίνει ίσος με το μήκος του προτύπου (4 για το δέντρο της εκφώνησης), το πρότυπο έχει βρεθεί. Αυξάνει τον μετρητή εμφανίσεων κατά 1, μηδενίζει τον μετρητή χαρακτήρων και εμφανίζει στην έξοδο την συμβολοσειρά που ξεκινά από τη ρίζα και καταλήγει στη θέση αυτή του δέντρου. Ξεκινά την ίδια διαδικασία από τον χαρακτήρα στον οποίο βρίσκεται.

Αν φτάσει σε φύλλο και έχει βρεθεί το πρότυπο, εμφανίζει στην έξοδο την συμβολοσειρά που ξεκινά από τη ρίζα και καταλήγει στο φύλλο αυτό. Ξεκινά την ίδια διαδικασία στον επόμενο κλάδο του δέντρου όπως θα προσπελαζόταν το δέντρο σε μια κατά βάθος αναζήτηση.

Αν φτάσει σε φύλλο χωρίς να έχει βρει το πρότυπο, μηδενίζει τον μετρητή χαρακτήρων και ξεκίνα την ίδια διαδικασία στον επόμενο κλάδο του δέντρου όπως θα προσπελαζόταν σε μια κατά βάθος αναζήτηση.

**Ερώτημα 7**

import re

import collections

tf1 = 'BHTGTGGTYW'

p1 = r'([^A][^G](TGTGGT)[CT][AT])'

tf2 = 'WGACAGB'

p2 = r'([AT](GACAG)[^A])'

tf3 = 'BTGGGARD'

p3 = r'([^A](TGGGA)[AG][^C])'

#function to read the sequence

def readDna(file):

sequence = ''

with open(file, 'r') as dna:

for line in dna:

if not line[0] == '>':

sequence += line.rstrip()

dna.close()

return sequence

gen = readDna('Sequence.txt')

print(gen, '\n')

#function to find matches

def findTranscriptionFactors(s, file):

for i in s:

found1 = re.finditer(p1, s)

with open('Finds.txt', 'a') as matches:

print(f'Match found for ' + tf1 ' at position: {found1.start()} of genome\n')

matches.write(f'Match found for ' + tf1 ' at position: {found1.start()} of genome\n')

findTranscriptionFactors(gen, 'Finds.txt')

#function to complement the given sequence

def complementGenome(s, file):

complement = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}

emoneg = ''

with open(file, 'a') as changes\_file:

for i in s:

emoneg = complement[i]

changes\_file.write(emoneg)

changes\_file.close()

complementGenome(gen, 'Changes.txt')

#function to create the sequence after transcription to rna

def turnToRna(s, file):

transcr = {'T': 'U', 'A': 'A', 'C': 'C', 'G': 'G'}

rna = ''

with open(file, 'a') as transcr\_file:

for i in s:

rna = rna + transcr[i]

transcr\_file.write(rna)

transcr\_file.close()

turnToRna(gen, 'Rna.txt')

#function to count how many times each base appears in given sequence

#and to calculate percentage of CG compared to other bases

def countBases(s, file):

seq\_length = len(s)

print('Total number of bases:', seq\_length, '\n')

char\_count = collections.Counter(s)

print('Number of occurences of each base in sequence:', char\_count, '\n')

gc = 0

percent\_gc = 0

for c in s:

for i in range(len(s)):

if s[i] == 'C' or s[i] == 'G':

gc += 1

if gc > 0:

percent\_gc = (gc / seq\_length) \* 100

print('Percentage of "CG" compared to other bases in given sequence is:', percent\_gc, '\n')

with open(file, 'a') as count\_file:

count\_file.write(str(seq\_length))

count\_file.write('\n')

count\_file.write(str(char\_count))

count\_file.write('\n')

count\_file.write(str(percent\_gc))

count\_file.close()

countBases(gen, "Count.txt")