SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

FM-stablo

Andrea Bernat, Anel Hadžimuratagić, Robert Jambrečić

Voditelj: doc. dr. sc. Mirjana Domazet-Lošo

Zagreb, siječanj, 2020.

Sadržaj

[1. Uvod 1](#_Toc29932638)

[2. FM stablo 2](#_Toc29932639)

[2.1. Primjer 5](#_Toc29932640)

[2.2. Optimizacija 10](#_Toc29932641)

[3. Rezultati 11](#_Toc29932642)

[4. Zaključak 16](#_Toc29932643)

[5. Literatura 17](#_Toc29932644)

# Uvod

Jedan od osnovnih zadataka bioinformatike je prepoznavanje uzoraka u velikoj količini teksta. Problem prepoznavanja uzoraka je identificiranje pozicija pojavljenog uzorka u zadanom tekstu. Kod genskih podataka, uzorci i tekst se sastoje od malog abecednog skupa čiji elementi predstavljaju nukleotidne baze unutar DNK molekule. Kako bi se proces prepoznavanja uzorka ubrzao jedan od najpoznatijih pristupa je podudaranje indeksima. Prvo se izgradi struktura indeksa za zadani tekst, a zatim se pomoću izgrađenog indeksa pretražuje uzorak po tekstu. Standardni indeksi poput sufiksnog polja sastoje se od dviju osnovnih funkcija: prebrojavanje (engl*. count*) i određivanje položaja (engl*. locate*). Funkcija prebrojavanja će vratiti broj pojava uzorka unutar teksta dok će funkcija određivanja položaja dohvatiti sve pozicije na kojima se uzorak pojavljuje. Veliki problem spomenutih indeksa je taj što zahtijevaju previše memorije, pogotovo kod velikih tekstova. Kako bi se riješio ovaj problem razvijeni su različiti komprimirani indeksi, a jedni od njih su i FM indeksi. FM indeksi su dizajnirani tako da daju iste rezultate kao standardni indeksi, samo zauzimaju manje memorije. Nažalost, glavni nedostatak FM indeksa je što njihova funkcija za određivanje položaja je dosta sporija od standardnih indeksa. Prije nekoliko godina, točnije 2017. godine, skupina stručnjaka s MIT-a razvili su algoritam koji će značajno ubrzati funkciju određivanja položaja kod FM indeksa pod nazivom FM stablo. U ovome radu biti će objašnjen algoritam FM stablo koji pretvara prostor pretrage u stablo četverostrukog grananja i time omogućuje dohvaćanje više lokacija odjednom prolazeći kroz stablo.

# 2. FM stablo

FM-stablo sastoji se od FM indeksa i uzorkovanog sufiksnog polja *SA*. Zadatak SA je spremiti sve pozicije sufiksa za primljeni tekst *T*, počevši od leksički manjih pa do leksički većih nizeva. FM-indeksi sastoje se od prvog i posljednjeg stupca Burrows-Wheeler matrice koja nastaje kružnom rotacijom teksta te sortiranje redaka od leksički najmanjega do najvećega. Prvi stupac matrice označava se s *F*, a posljednji s *BWT(T).* U Tablici 1 prikazana je matrica i *SA* za ulazni tekst *T=acat$*, gdje znak *„$”* predstavlja kraj niza.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i | F |  | BWT |  | SA |
| 0 | $ | aca | t |  | 4 |
| 1 | a | cat | $ |  | 0 |
| 2 | a | t$a | c |  | 2 |
| 3 | c | at$ | a |  | 1 |
| 4 | t | $ac | a |  | 3 |

Tablica 1. Matrica M i sufiksno polje SA

Kako bi se odredio raspon redaka *[sp, ep]* matrice *M* koji počinju uzorkom *P*, FM-stablo kao funkciju prebrojavanja koristi funkciju algoritma FM-indeksa: *pretraga\_unazad(P, T).* Za algoritam je potrebno prethodno izračunati *C[s],* gdje je *s* slovo, a *C[s]* broj slova u *T* koja imaju manju leksičku vrijednost od *s*. Formula *ranks(BWT(T),index)* vraća broj pojavljivanja znaka *s* u *BWT(T)[0,index-1].*

*C[s]* vrijednosti za prethodni primjer nalaze se u Tablici 2. Slova su poredana od leksički najmanjega. Znak *$* ima najmanju leksičku vrijednost. Ne postoji slovo *s* manjom leksičkom vrijednošću od *$* te je zato *C[$]=0*. U tekstu *T* se jedan put ponavlja *$* i dva puta *a* pa je zato *C[c]=3* itd.

Dodatno, *C[s+1]* predstavlja prvi leksički znak veći od *s*. Za naš primjer vrijedi ako je *s=a*, onda je *C[a+1]=C[c].*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *s* | $ | a | c | t |
| *C[s]* | 0 | 1 | 3 | 4 |

Tablica 2. *C[s]* vrijednosti

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *i* | 0 | 1 | 2 | 3 | 4 |
| *BWT(T)* | t | $ | c | a | a |
| *Occ($)* | 0 | 1 | 1 | 1 | 1 |
| *Occ(a)* | 0 | 0 | 0 | 1 | 2 |
| *Occ(c)* | 0 | 0 | 1 | 1 | 1 |
| *Occ(t)* | 1 | 1 | 1 | 1 | 1 |

Tablica 3. Occ vraća broj pojavljivanja znaka *s* u *BWT(T)[0, i]*

Funkcija *pretraga\_unazad(P, T)* kreće od posljednjeg slova u *P* te se u svakom koraku pomiče za jedno slovo u lijevo. Na početku se se inicijaliziraju *sp=C[s]* i *ep=C[s+1]-1*. Time se dobivaju svi redci matrice *M* u kojima *F*(prvi stupac) sadrži *s*. U svim sljedećim koracima ako vrijedi *sp≤ep*, računa se LF mapiranje (engl. Last-to-First mapping) za *sp* i *ep* prema formuli *LF(l)=C[BWT(T)[l]]+rankl(BWT(T),l)* tj. *sp=C[s]+ranks(BWT(T),sp)* i *ep=C[s]+ranks(BWT(T),ep+1)-1*. U slučaju da je *sp>ep*, uzorak ne postoji u tekstu.

U nastavku je prikazano traženje intervala *[sp, ep]* za dani uzorak *P=ac* (*T=acat$,* tekst iz početnog primjera).

Kreće se od kraja *P* tj. znak *s=c.* *sp=C[c]=*3, *ep=C[c+1]-*1=*C[t]-*1=4-1=3

Sljedeći znak je *s=a*. *sp=C[a]+ranka(BWT(T)*,3)=1+0=1,

*ep=C[a]+ranka(BWT(T),*3+1)-1=1+1-1=1

Pomoću *sp* i *ep* može se izračunati broj uzoraka u tekstu: *ep-sp*+1=1-1+1=1

To je bilo prvo slovo uzorka *P* te je pretraga završena. Kako je *sp=ep=1*, to znači da se u Tablici 1 gleda redak kojemu je *i=1*. U slučaju da postoje sve vrijednosti u SA, može se odmah saznati lokacija uzorka u tekstu sa *SA[1]=0* tj. Uzorak *P=ac* kreće od *T[0].*

Ključ FM-stabla je njegova funkcija pronalaženja uzoraka u tekstu. Prednost FM-stabla je to što se više lokacija uzorka u tekstu može pronaći istovremeno.

Kako bi se smanjila potrošnja memorije *SA* se uzorkuje. Koristi se uzorkovanje po vrijednosti(engl. *value sampling*) tj. *S[i]* se uzorkuje samo ako vrijedi *SA[i]==0(mod D),* gdje je *D* udaljenost uzorkovanja. Za razumijevanje funkcije određivanja pozicija, važno je razumjeti sljedeće: ako je *sP* uzorkovan tj. Vrijednost *SA[i]* je poznata, onda se *SA[j]* koji pokazuje na jedan od uzoraka *P* u tekstu, može izračunati kao *SA[j]=SA[i]+1*. Također, ako je *ssP* (*ss* može biti bilo koja kombinacija slova abecede) uzorkovan, *SA[j]* od *P* može se izračunati kao *SA[j]=SA[i]+2*. Po istom principu se računa i za *s\*iP (i=1,...,D-1).*

## 2.1. Primjer

U nastavku je prikazan primjer koji demonstrira cijeli algoritam. Tekst *T=ACACATAACA$,* uzorak *P=ACA*, udaljenost uzorkovanja *D=4* te abeceda ∑ sadrži slova *[A,C,G,T].* Tablica 4 prikazuje matricu *M* te *SA*. Važno je napomenuti da se pri izvođenju algoritma ne sprema cijela matrica, već samo stupac *F* i *BWT*. Također, *SA* sadrži samo polja označena plavom bojom tj. one za koje vrijedi *SA[i]==0(mod 4)*).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |  |  |  |
|  | F |  |  |  |  |  |  |  |  |  | BWT |  |  | SA |
| 0 | $ | A | C | A | C | A | T | A | A | C | A |  |  | 10 |
| 1 | A | $ | A | C | A | C | A | T | A | A | C |  |  | 9 |
| 2 | A | A | C | A | $ | A | C | A | C | A | T |  |  | 6 |
| 3 | A | C | A | $ | A | C | A | C | A | T | A |  |  | 7 |
| 4 | A | C | A | C | A | T | A | A | C | A | $ |  |  | 0 |
| 5 | A | C | A | T | A | A | C | A | $ | A | C |  |  | 2 |
| 6 | A | T | A | A | C | A | $ | A | C | A | C |  |  | 4 |
| 7 | C | A | $ | A | C | A | C | A | T | A | A |  |  | 8 |
| 8 | C | A | C | A | T | A | A | C | A | $ | A |  |  | 1 |
| 9 | C | A | T | A | A | C | A | $ | A | C | A |  |  | 3 |
| 10 | T | A | A | C | A | $ | A | C | A | C | A |  |  | 5 |

Tablica 4. Matrica M i sufiksno polje SA

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S | $ | A | C | G | T |
| C[s] | 0 | 1 | 7 | 0 | 10 |

Tablica 5. *C*[s] vrijednosti

Tablica 5 sadrži vrijednosti polja *C* dok se u tablici 6 nalaze vrijednosti pojave svakog slova od indeksa 0 do svakog sljedećeg indeksa do kraja niza *BWT*.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| BWT(T) | A | C | T | A | $ | C | C | A | A | A | A |
| Occ($) | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Occ(A) | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 4 | 5 | 6 |
| Occ(C) | 0 | 1 | 1 | 1 | 1 | 2 | 3 | 3 | 3 | 3 | 3 |
| Occ(T) | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Tablica 6. *Occ[s]* vrijednosti

Prvo se prema funkciji *pretraga\_unazad(P, T)* računa interval matrice u kojima postoji traženi uzorak (objašnjeno na prošlom primjeru).

*P*=ACA

AC**A**  
*s*=A  
*sp*=*C[s]=C*[A]=1,*ep=C*[s+1]-1=*C*[A+1]-1=*C*[C]-1=7-1=6   
Redci koji počinju s A su [1,6].

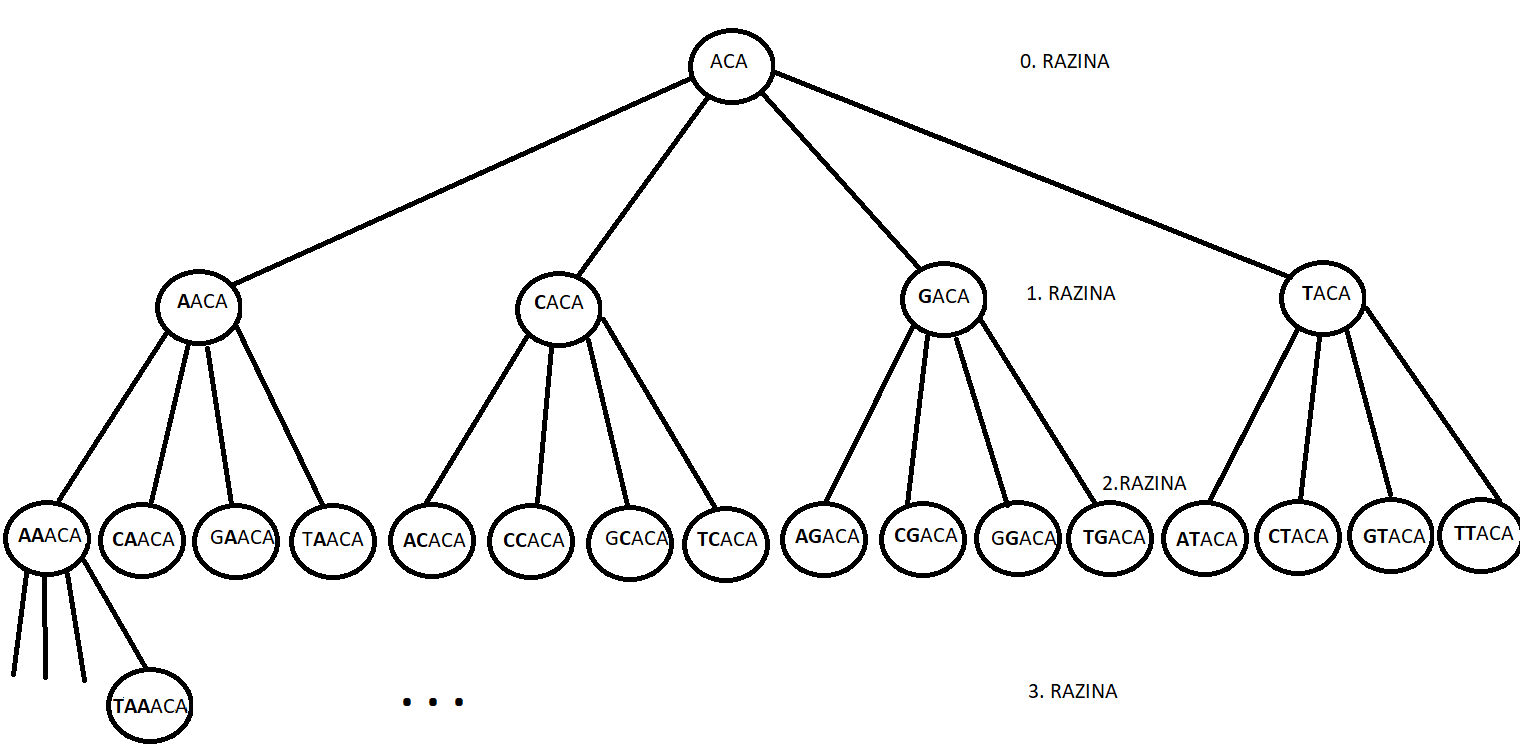
A**CA**  
*s*=C  
*sp=C[s]*+*ranks(BWT(T),sp*)=*C*[C]+*rankC(BWT(T),*1)=7+0=7  
ep=*C[s]+ranks(BWT(T),ep*+1)-1=*C*[C]+rankC(BWT(T),6+1)-1=7+3-1=9

Redci koji počinju s CA su [7,9].

**ACA**

*s*=A   
*sp=C*[A]+*rankA(BWT(T),*7)=1+2=3  
*ep=C*[A]+*rankA(BWT(T),*9+1)-1=1+5-1=5  
Redci u kojima se pojavljuje ACA su [*sp,ep*]=[3,5].

Nakon što su određeni *sp* i *ep* počinje izgradnja FM-stabla (Slika 1). Stablo se pretražuje BFS algoritmom tj. razinu po razinu, s lijeva na desno.



Slika 1. FM-stablo

Na 0. Razini nalazi se uzorak *P*=ACA i za njega vrijedi [*sp,ep*]=[3,5]. Potrebno je provjeriti da li postoje uzorkovani *SA[i]* u intervalu [3,5] (uzorkovani sufiksi su označeni plavom bojom u tablici). *SA*[4] je uzorkovan te se sprema u *R* 🡪 *SA*[4]+*razina*=*SA*[4]+0=0+0=0. Još nedostaju *SA*[3] i *SA*[5].

Na 1. razini se ispituju *S*=*sP*, gdje *s* može biti bilo koje slovo iz ∑ (A,C,G,T). Na toj razini se znači nalaze čvorovi **A**ACA, **C**ACA, **G**ACA i **T**ACA. Za svaki čvor računamo interval [*sp,ep*].

*S*=AP 🡪 *S*=**A**ACA

*sp=C[s]*+*ranks(BWT(T),sp)=C*[A]+*rankA*(*BWT(T),*3)=1+1=2  
*ep = C[s]+ranks(BWT(T),ep*+1) - 1 = *C*[A]+*rankA(BWT(T),*5+1) - 1=1+2-1=2

Redci gdje se pojavljuje **A**ACA su [2,2] tj. samo redak 2. Zatim se provjerava je li *SA*[2] uzorkovan. Kako redak nije uzrokovan nastavlja se dalje s 1. razinom.

*S*=C*P*🡪S=**C**ACA   
*sp=C*[C]+*rankC(BWT(T),*3)=7+1=8  
*ep= C*[C]+*rankC(BWT(T),*5+1)-1=7+2-1=8

*SA*[8] također nije uzrokovan, nastavlja se dalje.

*S*=**G**P🡪*S*=**G**ACA  
*C*[G]=0 jer se G ne pojavljuje u tekstu *T*, stoga se izračun za njega preskače.

*S*=**T**P🡪*S*=**T**ACA  
*sp=C*[T]+*rankT(BWT(T),*3)=10+1=11  
*ep= C*[T]+*rankT(BWT(T),*5+1)-1=7+2-1=10+1-1=10

Kako je *sp>ep* zaključuje se da uzorak **T**ACA ne postoji u *T*.

Kraj prve razine.

Na 2. razini provjerava se *S=s*AACA i *S=s*CACA.

Za čvor AACA dobiven je interval [sp,ep]=[2,2] te ga koriste sva njegova djeca.

Za **A**AACA i **C**AACA se dobiva da je *sp>ep* što znači da uzorci ne postoje. G se ne pojavljuje u tekstu *T* stoga se **G**AACA preskače.

*S*=**TA**P🡪*S*=**TA**ACA   
*sp=C*[T]+*rankT(BWT(T)*,2)=10+0=10  
*ep=C*[T]+*rankT(BWT(T),*2+1)-1=10+1-1=10  
*SA*[10] nije uzrokovan, nastavlja se dalje.

Za čvor **C**ACA dobiven je interval [*sp,ep*]=[8,8] te ga koriste sva njegova djeca.

*S*=**AC**P🡪*S*=**AC**ACA  
*sp=C*[A]+*rankA(BWT(T),*8)=1+3=4  
*ep=C*[A]+*rankA(BWT(T),*8+1)-1=1+4-1=4  
*SA*[4] je uzorkovan te se sprema u *R* 🡪 *SA*[4]+*razina*=0+2=2. Potrebno je locirati još jedan uzorak.

Za **CC**ACA i **TC**ACA se dobiva da je *sp>ep* što znači da uzorci ne postoje. G se ne pojavlja u tekstu *T* stoga se **GC**ACA preskače.

Čvor **G**ACA sadrži G stoga se preskače.

Kraj druge razine.

Na 3. razini se nastavlja obilaziti stablo s lijeva na desno gore navedenim postupkom. Dolazimo do čvora **ATA**ACA.

*S*=**ATA**P🡪*S*=**ATA**ACA  
*sp=C*[A]+*rankA(BWT(T),*10)=1+5=6  
*ep= C[A]+rankA(BWT(T),*10+1)-1=1+6-1=6

*SA*[6] je uzrokovan te se sprema u *R* 🡪 *SA*[6]+3=4+3=7. Pronađena su sva tri uzorka te algoritam završava.

Pri dohvaćanju *SA* uzorkovanih elemenata kako bismo znali koji je element uzorkovan koristi se niz *B* sastavljen od jedinica i nula, gdje 1 označava da je element na toj poziciji uzorkovan, a 0 da nije uzorkovan. Tako bi za naš primjer vrijedilo *B*=00001011000. Tablica 7 prikazuje vrijednosti pojave jedinica od indeksa 0 do svakog sljedećeg indeksa do kraja niza *B*.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| *B* | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| Occ(1) | 0 | 0 | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |

Tablica 7. *Occ[1]* vrijednosti

Kako bismo saznali je li element polja *SA* na poziciji *i* uzorkovan ispitat ćemo je li na poziciji *i* u nizu *B* postavljena jedinica, ako je kako bismo dohvatili element *SA*[*i*] koristit ćemo slijedeću funkciju:

*SA*[*i*]=*rank1(B,i) (2.2.1)*

Npr. za dohvat *SA*[4]=rank1(*B*,4) = *Occ*(1)[4]=0

## 2.2. Optimizacija

Broj rank operacija raste eksponencijalno svakom razinom, u svakoj se razini računa 2x4i rank operacija, gdje je *i* broj razine, 2 zbog računanja *sp* i *ep* u svakom koraku te 4 za broj slova u abecedi(A,C,G,T). Posljednja razina sadrži 2x4*D*-1 operacija što je više od zbroja operacija svih prethodnih razina. Iz tog razloga koristi se „*early leaf node calculation*“ optimizacija pomoću koje se izbjegava zadnja razina.

Optimizacija je bazirana na Teoremu 21[1]. Ako je *P*[*i*, *|P|*-1]=*T*[*SA*[*j*],*SA*[*j*]-1], gdje je *i=1,…,D*, a *j* je iz intervala u kojemu se pojavljuje *P*[*i*, |*P*|-1] tj. [*spi,epi*], onda vrijedi *SA*[*k*]=*SA*[*j*]-*i*, gdje je *k* iz intervala u kojemu se pojavljuje *P*.

Kako bi se dobio interval [*sp1, ep1*] računa se *P*[1, |*P*|-1]. Zatim se u *R* dodaje svaki *SA*[*j*] koji je uzrokovan i za koji vrijedi *T*[*SA*[*j*]-1]=*P*[0]. Konačno, za svaki *SA*[*j*] koji je pronađen u ovoj fazi umanjuje se za jedan.

Za navedeni primjer pretrage *P*=ACA umjesto obilaska 3. razine, možemo iskoristiti navedenu optimizaciju. Stoga, pretraga 3. razine može se svesti na slijedeći izračun:

*P*=ACA 🡪 *P*[1, |*P*|-1|=CA, računa se interval [*sp1,ep1*]=[7,9].

Za *j*=7 , *SA*[7] je uzorkovan te vrijedi *T*[*SA*[7]-1]=*P*[0] 🡪 *T*[8-1]=*P*[0] 🡪 A=A, stoga se dodaje u *R* 🡪 *SA*[7]-1=8-1=7

Za *j*=8 , *SA*[8] nije uzrokovan, nastavljamo dalje

Za *j*=9 , *SA*[9] također nije uzorkovan.

Kraj izračuna.

Ovim algoritmom se neće morati računati *D-1* razina te se time smanjuje broj „skupih“ rank operacija, stoga će time FM stablo značajno brže raditi za dulje tekstove.

Još jedan način optimizacije je korištenje funkcije određivanja položaja od FM-indeks algoritma ako je interval [*sp,ep*] manji od nekog definiranog praga, a inače računati preko funkcije određivanja položaja od FM stabla.

# 3. Rezultati

Generirali smo 10 sintetskih datoteka različitih duljina. Slova A,B,C,D i E u nazivu datoteke predstavljaju veličinu datoteke. Slovo A predstavlja duljinu od 100 znakova, B od 1000, C od 10000, D od 100000 i E od 1000000. Program smo testirali na tih 10 datoteka i na ecoli.txt koja predstavlja genome bakterije. Kao što možemo vidjeti u tablicama, naša implementacija je jedan do dva reda sporija u odnosu na originalnu, ali kada bi se gledala cjelokupna implementacija naš program radi brže, jer izrada njihovih uzoraka oduzima dosta vremena. Na manjim testnim podacima zauzeće memorije je u rangu s njihovom implementacijom, dok kod većih testnih datoteka zauzuće memorije se značajno razlikuje.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naša implementacija | | Originalna implementacija | |
| Uzorak | Vrijeme (**μs)** | Memorija (**kB**) | Vrijeme (**μs)** | Memorija (**kB**) |
| ACA | 251 | 7110 | 20 | 6781 |
| AGA | 124 | 7105 | 15 | 6654 |
| AGGA | 198 | 7120 | 8 | 6670 |
| AAATTT | 205 | 7160 | 5 | 6678 |

Tablica 8. Rezultati za datoteku test\_A\_1.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naša implementacija | | Originalna implementacija | |
| Uzorak | Vrijeme (**μs)** | Memorija (**kB**) | Vrijeme (**μs)** | Memorija (**kB**) |
| TAGA | 597 | 7140 | 45 | 6671 |
| CACATA | 229 | 7189 | 25 | 6598 |
| AGTACATGCATTGTGTGTGCGCTGCTAGCTTGGGTTTTTTT | 421 | 7128 | 17 | 6780 |
| TGCGCTGCTAGCTTGGG | 323 | 7126 | 24 | 6754 |

Tablica 9. Rezultati za datoteku test\_B\_1.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naša implementacija | | Originalna implementacija | |
| Uzorak | Vrijeme (**μs)** | Memorija (**kB**) | Vrijeme (**μs)** | Memorija (**kB**) |
| CAGACAGACA | 460 | 7132 | 14 | 6648 |
| GACAGATGTACGACCGCAGCAGACAGACAGACAGATGTAC | 227 | 7128 | 15 | 6678 |
| CAGACAGACAGATGTACGACCGCAGCAGACAGACAGACAGATGTACGACC | 390 | 7128 | 33 | 6746 |
| CGCAGCAGACAGACAGACAGATGTACGACCGCAGCAGACAGACAGACAGATGTACGACCG | 484 | 7132 | 32 | 6724 |

Tablica 10. Rezultati za datoteku test\_B\_2.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naša implementacija | | Originalna implementacija | |
| Uzorak | Vrijeme (**μs)** | Memorija (**kB**) | Vrijeme (**μs)** | Memorija (**kB**) |
| TGGGTTTTTTTCGAGTAATGA | 334 | 8025 | 22 | 7115 |
| ATATATATATATTTATTCTATAGATACAGATACATTGGCCATACAGTAGCAGATGACAGTAGACTGACA | 857 | 8227 | 18 | 7050 |
| ATA | 1058 | 8219 | 30 | 7127 |
| ATATATA | 499 | 8287 | 18 | 7111 |

Tablica 11. Rezultati za datoteku test\_C\_1.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naša implementacija | | Originalna implementacija | |
| Uzorak | Vrijeme (**μs)** | Memorija (**kB**) | Vrijeme (**μs)** | Memorija (**kB**) |
| TGCGACGTTCTAAGCGTTGGTCCATGTGAA | 320 | 8140 | 9 | 7016 |
| CATCCAGGATCACGTCGCCCTGAAAAAAAGATATCAGCAACTCTCCTCCT | 269 | 8132 | 16 | 7068 |
| AGAGGA | 171 | 8142 | 13 | 7102 |
| CGCAGCAGACAGACAGACAGATGTACGACCGCAGCAGACAGACAGACAGATGTACGACCG | 384 | 8180 | 37 | 7124 |

Tablica 12. Rezultati za datoteku test\_C\_2.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naša implementacija | | Originalna implementacija | |
| Uzorak | Vrijeme (**μs)** | Memorija (**kB**) | Vrijeme (**μs)** | Memorija (**kB**) |
| ATAGTAGATTCCAGGTTAAGATAGATAGGCTAGCAAATTTGACATGACATAGAGAGGGAACCTTAGGAATTGACTGACT | 1980 | 18760 | 20 | 8540 |
| ATACCTTAGATACAGATAGCCCCCACACGTAC | 2151 | 18884 | 22 | 8635 |
| CAGGTTAAGATAGATAGGCTAGCAAATTTGACATGACATAGAGAGGGAACCTTAGGAATTGACTGACTAATACCAAC | 1896 | 18676 | 19 | 8721 |
| ACATA | 5067 | 18383 | 26 | 8802 |

Tablica 13. Rezultati za datoteku test\_D\_1.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naša implementacija | | Originalna implementacija | |
| Uzorak | Vrijeme (**μs)** | Memorija (**kB**) | Vrijeme (**μs)** | Memorija (**kB**) |
| GATAGATAGATAGAGACAGACAAGACAGACAGACATAGAT | 1504 | 18068 | 42 | 8476 |
| AAGACAGACAGACATAGATAGATAGATAGATAGATAGATA | 1826 | 18168 | 23 | 8348 |
| GACATAGATAGATAGATAGATAGATAGATAGAGACAGACAAGACAGACAGACATAGATAGATAGATAGATAGATAGATAG | 507 | 16172 | 27 | 8336 |
| CAGACAGA | 1987 | 16048 | 44 | 8440 |

Tablica 14. Rezultati za datoteku test\_D\_2.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naša implementacija | | Originalna implementacija | |
| Uzorak | Vrijeme (**μs)** | Memorija (**kB**) | Vrijeme (**μs)** | Memorija (**kB**) |
| CATTGAGTAGCAAGGCACTTCCGGACTCAATGAAGGGCCGGGAAAGG | 4981 | 108997 | 43 | 15123 |
| CAAGGCGCGGCGG | 2396 | 109972 | 25 | 19123 |
| TTTCAGGTATCGTGCACGTAGGGTTGGACCGCACGCATGTAAGTCAAATCGCGCGTCTAGACTCAGCTCTATTTTAGTGGTAACGCACTCAAAG | 1530 | 109786 | 45 | 18529 |
| A | 52859 | 109986 | 1265 | 19005 |

Tablica 15. Rezultati za datoteku test\_E\_1.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naša implementacija | | Originalna implementacija | |
| Uzorak | Vrijeme (**μs)** | Memorija (**kB**) | Vrijeme (**μs)** | Memorija (**kB**) |
| ACCTTCCTAGTAACCGAGGACTAAGAATCTAT | 1027 | 106184 | 102 | 18994 |
| AGCT | 6275 | 106204 | 173 | 18912 |
| ATAACACCCCT | 3950 | 106248 | 53 | 18900 |
| TCATAATCGTTTGTAATTCAAAGCTTGATCAACATTGGATTGTCTTTCTCTTAAAGTATTATGCAGAATGGGGTACGCATACCATATAAA | 1064 | 106324 | 24 | 18918 |

Tablica 16. Rezultati za datoteku test\_E\_2.txt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naša implementacija | | Originalna implementacija | |
| Uzorak | Vrijeme (**μs)** | Memorija (**kB**) | Vrijeme (**μs)** | Memorija (**kB**) |
| AGCTTTTCA | 6419 | 248388 | 32 | 21460 |
| AACGGGCAATATGTCTCTGT | 3658 | 248388 | 16 | 21436 |
| TACAGGAAACACAGAAAAAAGCCCGCACCT | 2540 | 248392 | 21 | 21420 |
| CGGCGGTACATCAGTGGCAAATGCAGAACGTTTTCTGCGTGTTGCCGATATTCTGGAAAG | 1613 | 248388 | 12 | 21420 |

Tablica 17. Rezultati za datoteku ecoli.txt

# 4. Zaključak

U ovome radu pokušali smo implementirati i objasniti koncept rada algoritma FM stabla na temelju originalne implementacije. FM stablo značajno ubrzava dohvaćanje pozicija pojave nekog uzorka u genomskim podacima. FM stablo pri traženju uzorka koristi FM indeks te stvara višestruko stablo. Na taj način više pozicija pojave zadanog uzorka može biti dohvaćeno kroz višestruko stablo. Suprotno tome, postojeće metode unutar FM-indeksa dohvaćaju pozicije svih pojava zadanog uzorka jednu po jednu. FM-stablo smanjuje broj nepotrebnih izračuna "skupih" operacija i time predstavlja bolji lokalitet podataka. Također, koristili smo dvije metode za optimizaciju FM stabla navedenih u originalnoj implementaciji kako bismo dodatno smanjili broj nepotrebnih izračuna i time ubrzali algoritam. Naša implementacija je u odnosu na originalnu jedan do dva reda sporija, te iako je zauzeće memorije većinom u rangu s njihovim, ipak pri većim ulaznim podatcima njihovo zauzeće memorije se značajno razlikuje od našeg. Međutim, u obzir se mora uzeti i to da se naša implementacija temelji na nativnoj implementaciji, tj. njihova implementacija se temelji na programiranju niske razine (engl. *low level*) kako bi što više ubrzali i smanjili memoriju programa. Također, uočili smo pogrešku u radu njihovog programa s pretragom uzorka duljine 1, dok kod našeg to nije slučaj. U budućnosti bismo poradili na našim vještinama u implementiranju na niskoj razini kako bismo dodatno ubrzali naš algoritam te mu smanjili potrošnju memorije.

# 5. Literatura

[1] Haoyu Cheng, Ming Wu and Yun Xu (2017), FMtree: a fast locating algorithm of FM-indexes for genomic data

[2] Ben Langmead, Burrows-Wheeler Transform and FM Indeks

[3] Paulo G. S. da Fonseca and Israel B. F. da Silva (2017), Online Construction of Wavelet Trees