SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

BIOINFORMATIKA

IZGRADNJA FM-INDEKSA KORIŠTENJEM STABLA VALIĆA

Danijel Sokač
Pamela Miletić

Zagreb, siječanj 2015.

Sadržaj

1. Uvod

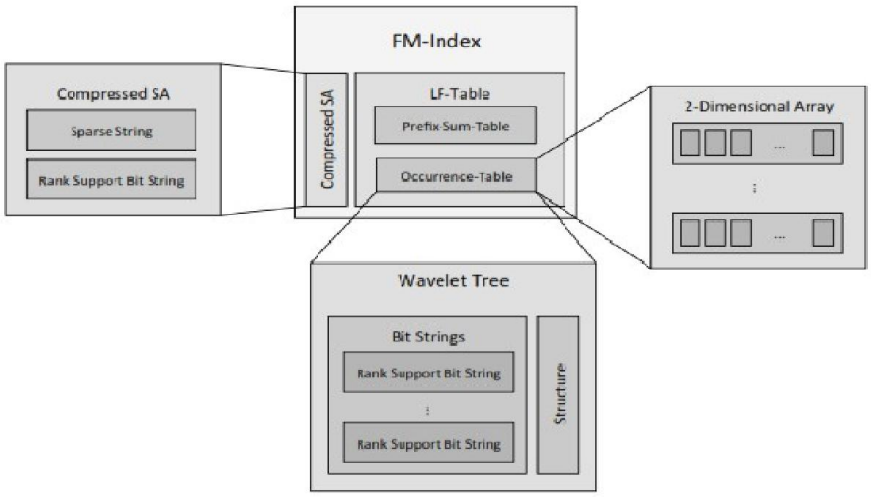
Nova otkrića u svijetu genoma sa sobom povlače i nova pitanja. Proučava se sve veći spektar podataka, a njih je potrebno negdje i pohraniti, ali i na efikasan i brz način doći do željenih informacija. Vrlo često istraživanja zahtijevaju identifikaciju sekvenci u genomima koji su od određenog značaja, odnosno dekodiranje podataka skrivenih u genomu. Određeni pristupi pomoću kojih se ovo pokušavalo postići, trošili su previše memorije budući da je potrebno analizirati veliku količinu podataka. Problem je rješen korištenjem FM indeksa.

FM indeks je memorijski učinkovit i vrlo brzi pristup pretraživanju uzoraka. Može se koristiti i u druge svrhe, ne samo za istraživanje sekvenci u genomima. Ovo je dobro rješenje za bilo koju aplikaciju u kojoj je potrebno pronaći željeni podniz unutar teksta, i to efikasno i u kratkome vremenu.

U ovome radu smo implementirali jedno takvo rješenje koje koristi FM-indeks te ga testirali koristeći genome bakterija. U nastavku je opisan algoritam i prikazani su koraci algoritma na jednostavno primjeru.

2. Opis algoritma

Cilj algoritma je omogućiti brzu pretragu bilo kojeg podniza u tekstu. U tu svrhu koristi se FM-indeks koji se sastoji od LF-tablice i sufiksnog polja. LF-tablica pak sadrži PrefixSum-tablicu i stablo valića (engl. *Wavelet Tree*). Navedeno se može vidjeti na idućoj slici (Sl 2.1).



Slika 2.1. FM-indeks

2.1 FM-indeks

FM-indeks (engl. *The Full-Text-Minut-Space index*) je podatkovna struktura izgrađena nad tekstom T koja podržava efikasno pretraživanje podnizova u T. FM-indeks je također i memorijski učinkovit. Ovaj indeks je zapravo kolekcija tablica s dodatnim funkcionalnostima kao što je prikazano na slici 2.1. FM-indeks sadrži LF-tablicu i sufiksno polje pomoću kojih se može odrediti polazni tekst ili pronaći odgovarajući podniz unutar teksta.

2.2 Sufiksno polje

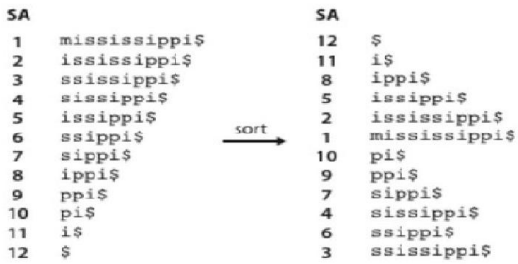
Sufiksno polje služi za određivanje točne pozicije podniza unutar teksta. Međutim, nije potrebno čuvati pozicije svih sufiksa od T jer LF-mapiranje omogućava rekonstrukciju podniza od T počinjući od bilo koje pozicije.

Sufiksno polje se izgrađuje tako da se na kraj niza dodaje znak $ koji je leksikografski manji od ostalih znakova iz abecede. Zatim se određuju svi sufiksi niza, te se oni poredaju po abecedi.

U najjednostavnijem obliku, sufiksno polje može biti konstruirano za string S[1..N] na način da se [1]:

1. Konstruira polje pokazivača za sve sufikse S[1..N], S[2..N],…, S[N..N].
2. Sortiraju se pokazivači po leksikografskom redoslijedu sufiksa.

Primjer sortiranja stringa 'mississippi' sa znakom $ pokazano je na idućoj slici (Sl 2.2.1).



Slika 2.2.1 Sortiranje

U ovome radu smo koristili Kärkkäinen-Sandersov algoritam za izgradnju sufiksnog polja.

2.2.1 Kärkkäinen-Sandersov algoritam

Za neki niz S duljine n nad abecedom Σ provode se idući koraci:

1. Rekurzivno se sortira 2n/3 sufiksa s_i (i mod 3 ≠ 0, tj. i = 1, 2);
2. Sortira se n/3 sufiksa s_i (i mod 3 = 0) koristeći rezultat koraka (1);
3. Napraviti spajanje sortiranih polja dobivenih u koracima (1) i (2).

2.3 LF-tablica

LF-mapiranje (eng. *LF-mapping; Last-to-Front mapping; Last-to-First mapping*) opisuje vezu, odnosno mapiranje, zadnjeg i prvog stupca u listi leksikografski sortiranih cikličkih rotacija niza S, a temelji se na sljedećem: i-ta pojava znaka c u zadnjem stupcu (stupcu L) leksikografski sortiranih cikličkih rotacija odgovara i-toj pojavi znaka c u prvom stupcu (stupcu F). Dakle, BWT[i] se nalazi u stupcu F na mjestu LF[i].

LF-mapiranjem možemo iz niza B, uz prisutnost stupca F, izgraditi početni niz S. Prvi stupac F sačinjavaju leksikografski sortirani znakovi iz S, a time i B, što znači da nam je samo B dovoljan da bi odredili S. Sama LF-tablica se sastoji od Prefix-Sum tablice i tablice pojava koja može biti izgrađena kao 2D polje ili kao stablo valića.

2.4 Stablo valića

Stablo valića (engl. *Wavelet Tree*) je binarno stablo niza bitova koje predstavlja tekst T. To je posebna struktura podataka koja, kada se sprema korištenjem RRR sekvenci, može riješiti operacije u vremenu O(log A). A predstavlja veličinu abecede. RRR vrši kompresiju stabla valića.

Stablo valića omogućava dohvat broja pojavljivanja znaka do određenog indeksa.

2.5 PrefixSum-tablica

Svaki element ove tablice predstavlja broj znakova u T koji su leksikografski prije određenog znaka.

2.6 FM indeks i Burrows-Wheeler transformacija

FM indeks (engl. *Full-text indeks in Minute space*) se temelji na korištenju Burrows-Wheeler transformacije (BWT) kao podloge za sažimanje i sufiksna polja.

2.6.1 Burrows-Wheeler transformacija

BW transformacija reverzibilno permutira string na način da se znakovi iz ponavljajućih podstringova grupiraju zajedno. Ovaj je postupak koristan zbog brzog sažimanja teksta.

Koraci su idući [1]:

1. Na kraj teksta je potrebno dodati znak $ koji je leksikografski manji od svakog znaka iz teksta T;
2. Stvori se konceptualna matrica u kojoj je prvi redak jednak T, a svaki idući je ciklički pomak prethodnog za jedno mjesto udesno;
3. U dobivenoj matrici sortirati retke u leksikografskom poretku. Na taj način smo dobili matricu $M_{T;}$
4. Iz posljednjeg stupca M_T se dobiva transformirani tekst L.

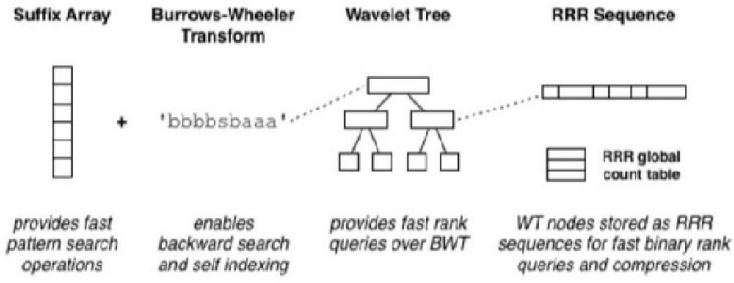
Postupak je povezan sa sufiksnim poljem: BWT[i]=S[SA[i]–1,BWT[1]=$, gdje je S originalni string, sufiksno polje SA i BWT je Burrows Wheeler transformacija.

Zaključno, i-ti simbol BWT je simbol neposredno prije i-tog sufiksa (Sl 3.1.1).

BWT	SA	
i	12	$
p	11	i$
s	8	ippi$
s	5	issippi$
m	2	ississippi$
$	1	mississippi$
p	10	pi$
i	9	ppi$
s	7	sippi$
s	4	sissippi$
i	6	ssippi$
i	3	ssissippi$

Slika 3.1.1 BWT i SA

Ferragina i Manzini su predložili da se kombinira BWT i sufiksno polje čime se kreira FM indeks koji omogućava pretraživanje unazad. Prikaz cjelokupnog postupka je na idućoj slici (Sl 3.1.2).



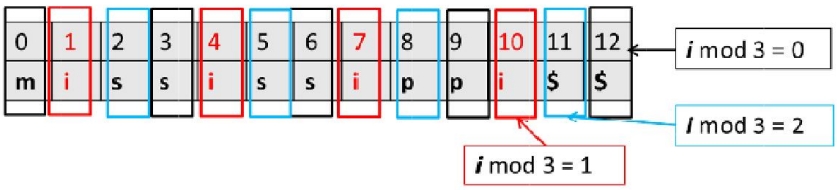
Slika 3.1.2 SA+BWT+Wavelet Tree+RRR

3. Primjer izvođenja algoritma

Kako bismo još bolje objasnili algoritam, prikazat ćemo korak po korak algoritma na jednostavnom primjeru. Uzet ćemo tekst 'mississippi'. Slijede opisi kroz sve faze algoritma [2].

3.1 Kärkkäinen-Sandersov algoritam

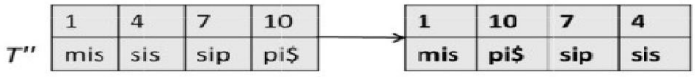
Kako smo rekli, uzimamo za niz S = mississippi$$. Dodali smo dva znaka $ i time je duljina niza višekratnik broja 3. Sada je potrebno iz S stvoriti T` na način da se spajaju podnizovi S[i, i+2], gdje je i mod 3 ≠ 0. T' je ississippi$$ssissippi, što se lako može uočiti iz slike niže (Sl 3.1.1).



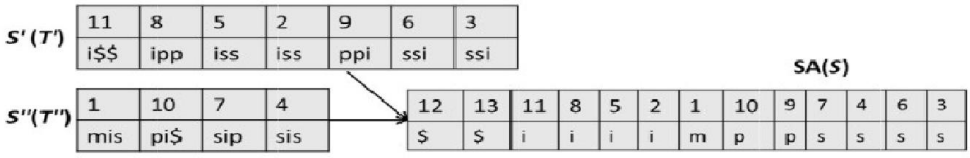
Slika 3.1.1 Stvaranje T` iz početnog niza S

S' se tvori iz T' tako da se svaki podniz T' [3i-2, 3i] zamijeni jedinstvenim leksikografskim imenom S'[i] uz uvjet: S'[i] ≤ S'[j], ako i samo ako T'[i, i+2] ≤ T'[j, j+2].

Korištenjem sortiranih sufiksa niza S` se određuje redoslijed sufiksa S``. Potrebno je prvo iz S oblikovati T`` koji čine trojke S[i, i+2], i mod 3 = 0. T`` ispada mississipi$ (Sl 3.1.2). Provodi se sortiranje trojki iz T``.



Slika 3.1.2 Dobivanje T``



Slika 3.1.3 Spajanje S` i S`` i određivanje sufiksnog polja

Sufiksno polje se određuje prema sortiranim sufiksima u T` (tj. S`) i T`` (tj. S``). Sufiksno polje je prikazano na gornjoj slici (Sl 3.1.3).

3.2 Konstrukcija BWT iz dobivenog sufiksnog polja

Konstruirat ćemo niz BWT za niz S = mississippi$ korištenjem SA=[12,11,8,5,2,1,10,9,7,4,6,3].

i	SA[i]	Abecedno poredani sufiksi od S	B[i]
0	12	$	i
1	11	i$	p

2	8	ippi$	s
3	5	issippi$	s
4	2	ississippi$	m
5	1	mississippi$	$
6	10	pi$	p
7	9	ppi$	i
8	7	sippi$	s
9	4	sissippi$	s
10	6	ssippi$	i
11	3	ssissippi$	i

B[i] smo dobili prema pravilu da je B[i] = $ kada je SA[i] = 1, a inače je B[i] = S[SA[i] – 1] [3].

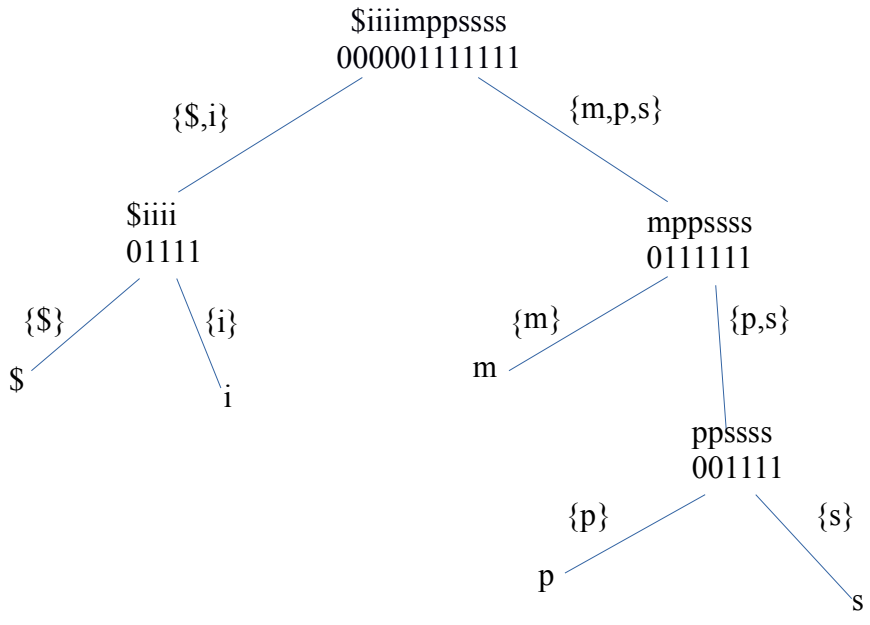
Zadnji stupac, odnosno Sz = ipssm$pissii, jeste BWT. U idućim koracima osim zadnjeg stupca trebat će nam i prvi, Sp = $iiiimppssss.

3.3 Stablo valića

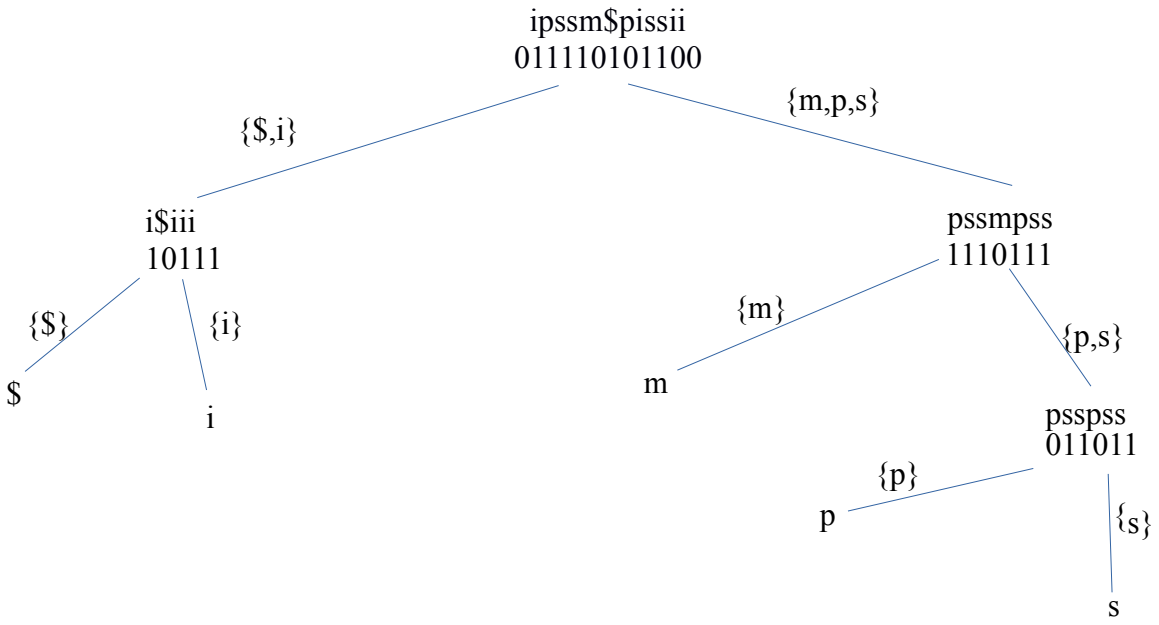
Prvi i zadnji stupac dobivene u prethodnom koraku treba spremiti kao stablo valića (Wavelet Tree).

Krenimo sa prvim stupcem Sp = $iiiimppssss.

Abecedno poredano, niz se sastoji od slova $imps. Podijelimo ih u dvije grupe, grupi [$,i] dajemo vrijednost 0, a grupi [m,p,s] vrijednost 1. Dodijelimo te vrijednosti svim znakovima u nizu, podijelimo one sa vrijednošću 0 u jedan niz, one sa vrijednošću 1 u drugi. Nastavljamo na taj način dok ne dođemo do toga da se svaki niz sastoji od jednog znaka – tada je kreiranje stabla valića završeno.



Isto ćemo napraviti i za zadnji stupac Sz = ipssm$pissii.



3.4 PrefixSum-tablica

U ovoj tablici se za svaki znak u nizu S čuva broj leksikografski manjih znakova.

U nizu S = mississippi$ imamo znakove $,i,m,p,s.

U idućoj tablici je za svaki od tih znakova prikazan broj znakova koji su leksikografski prije tog znaka.

Znak i	Broj znakova koji su leksikografski prije znaka i
$	0
i	1
m	5
p	6
s	8

3.5 Pretraživanje

Neka nam uzorak za pretraživanje bude P = 'iss', a tekst S = '`mississippi`'. Počet ćemo sa *i*=3, *c*=*P*[*i*]='s'. Trenutni znak je označen sa *c*. U idućoj tablici je prikaz prvog koraka, prije no što smo obavili ikakav upit za rang. Početak je označen sa *s,* a kraj sa *e,* te su u prvom retku početak i kraj dodatno označeni sa '>'.

$s = 1$
$e = 12$

	F	BWT	SA	
1 >	$	i	12	$
2	i	p	11	i$
3	i	s	8	ippi$
4	i	s	5	issippi$
5	i	m	2	ississippi$
6	m	$	1	mississippi$
7	p	p	10	pi$
8	p	i	9	ppi$
9	s	s	7	sippi$
10	s	s	4	sissippi$
11	s	i	6	ssippi$
12 >	s	i	3	ssissippi$

Kao što je označeno u tablici iznad, počinjemo sa $s = 1$ i $e = 12$ te c = P[i] = 's', s time da je i = 3. Prva dva upita za rang su:

```
s' = C[c] + rank(0, c) + 1 = 8 + 0 + 1 = 9
e' = C[c] + rank(12, c) = 8 + 4 = 12
```

$s = 9$
$e = 12$

	F	BWT	SA	
1	$	i	12	$
2	i	p	11	i$
3	i	s	8	ippi$
4	i	s	5	issippi$
5	i	m	2	ississippi$
6	m	$	1	mississippi$
7	p	p	10	pi$
8	p	i	9	ppi$
9 >	s	s	7	sippi$
10	s	s	4	sissippi$
11	s	i	6	ssippi$
12 >	s	i	3	ssissippi$

Sada prelazimo na idući korak. Kao što vidimo iz prošlog koraka, u tablici, sva pojavljivanja znaka 's' se nalaze u SA[9..12]. Imamo $s = 9$, $e = 12$, a za $i = 2$ vrijedi $c = P[i] =$'s'. Prema tome, slijede idući

upiti rangova:

```
s'' = C[c] + rank(8, c) + 1 = 8 + 2 + 1 = 11
e'' = C[c] + rank(12, c) = 8 + 4 = 12
```

$s = 11$
$e = 12$

	F	BWT	SA	
1	$	i	12	$
2	i	p	11	i$
3	i	s	8	ippi$
4	i	s	5	issippi$
5	i	m	2	ississippi$
6	m	$	1	mississippi$
7	p	p	10	pi$
8	p	i	9	ppi$
9	s	s	7	sippi$
10	s	s	4	sissippi$
11 >	s	i	6	ssippi$
12 >	s	i	3	ssissippi$

Vidimo da su sva pojavljivanja podniza 'ss' u SA[11..12].

I za zadnje upite, vrijednosti su $s = 11$, $e = 12$, $c = P[i]$ = 'i' gdje je $i = 1$.

```
s''' = C[c] + rank(10, c) + 1 = 1 + 2 + 1 = 4
e''' = C[c] + rank(12, c) = 1 + 4 = 5
```

$s = 4$
$e = 5$

	F	BWT	SA	
1	$	i	12	$
2	i	p	11	i$
3	i	s	8	ippi$
4 >	i	s	5	issippi$
5 >	i	m	2	ississippi$
6	m	$	1	mississippi$
7	p	p	10	pi$
8	p	i	9	ppi$
9	s	s	7	sippi$

10	s	s	4	sissippi$
11	s	i	6	ssippi$
12	s	i	3	ssissippi$

Iz ovog posljednjeg koraka smo dobili sva pojavljivanja traženog podniza 'iss', a nalaze se u SA[4..5].

4. Složenost algoritma

U idućoj tablici prikazana je složenost pojedinih dijelova algoritma. U razmatranju složenosti *n* će označavati duljinu sekvence (odnosno niza znakova), *p* duljinu traženog podniza, a *occ* broj pojavljivanja podniza *p* u originalnom nizu.

Funkcija programa	**Složenost**
Stvaranje sufiksnog polja	O(n)
Stvaranje stabla valića	O(n)
Stvaranje PrefixSum	O(n)
Stvaranje LF tablice	O(n)
Stvaranje FM indeksa	O(n)
Count()	O(p)
Find()	O(p + occ)

5. Testiranje

Za testiranje smo koristili umjetno generirane stringove, a podaci se nalaze u datotekama FASTA formata. Rezultati su prikazani u nastavku, a mogu se naći i na github repozitoriju u folderu 'test'.

Sequence length: 10^1
Memory: 124672B
Index creation time (s): 0.000278

Search time (s)	Count time (s)	Pattern length
5.4e-5	1.8e-5	10^0
8.8e-5	5.9e-5	10^1

Sequence length: 10^2
Memory: 125731B
Index creation time (s): 0.000635

Search time (s)	Count time (s)	Pattern length
6e-5	1.7e-5	10^0
0.000234	0.000212	10^1
0.002647	0.002202	10^2

Sequence length: 10^3
Memory: 129785B
Index creation time (s): 0.005719

Search time (s)	Count time (s)	Pattern length
8.1e-5	1.8e-5	10^0
0.002232	0.002248	10^1
0.01431	0.011487	10^2
0.090894	0.086919	10^3

Sequence length: 10^4
Memory: 172793B
Index creation time (s): 0.025601

Search time (s)	Count time (s)	Pattern length
0.000273	1.7e-5	10^0
0.01103	0.013749	10^1
0.075099	0.073634	10^2
0.736624	0.735301	10^3

Sequence length: 10^5
Memory: 606345B
Index creation time (s): 0.207198

Search time (s)	Count time (s)	Pattern length
0.002635	1.8e-5	10^0
0.083499	0.84742	10^1
0.819831	0.816192	10^2
7.20286	7.18593	10^3

Sequence length: 10^6
Memory: 5024682B
Index creation time (s): 2.19767

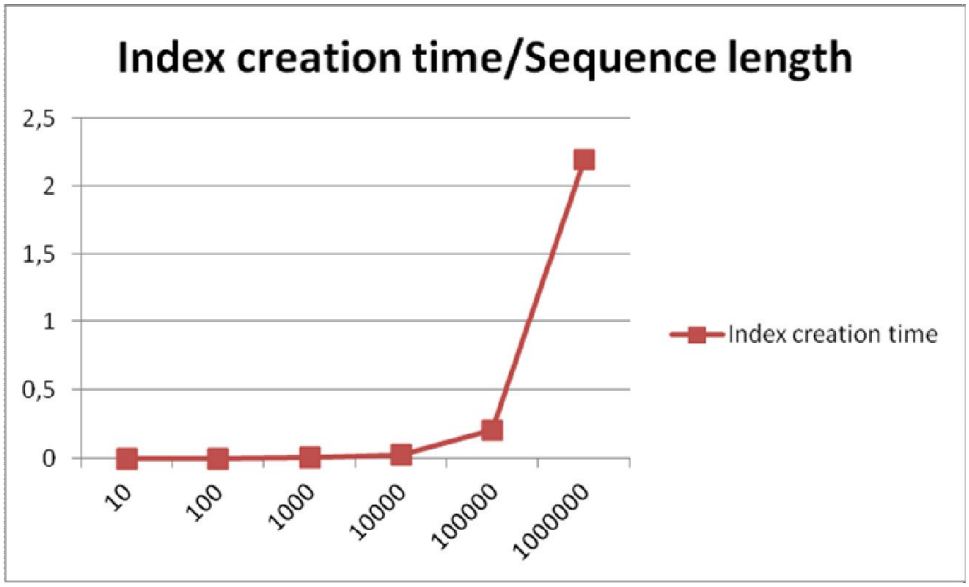
Search time (s)	Count time (s)	Pattern length
0.013445	1.7e-5	10^0
0.68512	0.684397	10^1
6.24546	6.2236	10^2
81.9767	81.9	10^3

Najmanji broj znakova za koji se provelo testiranje je 10^1 i u tom slučaju je memorija 124672B, a vrijeme kreiranja indeksa 0.000278s.

Najveći broj znakova za koji smo proveli testiranje je 10^6 i memorija je tada 5024682B, dok je vrijeme stvaranja indeksa 2.19767s.

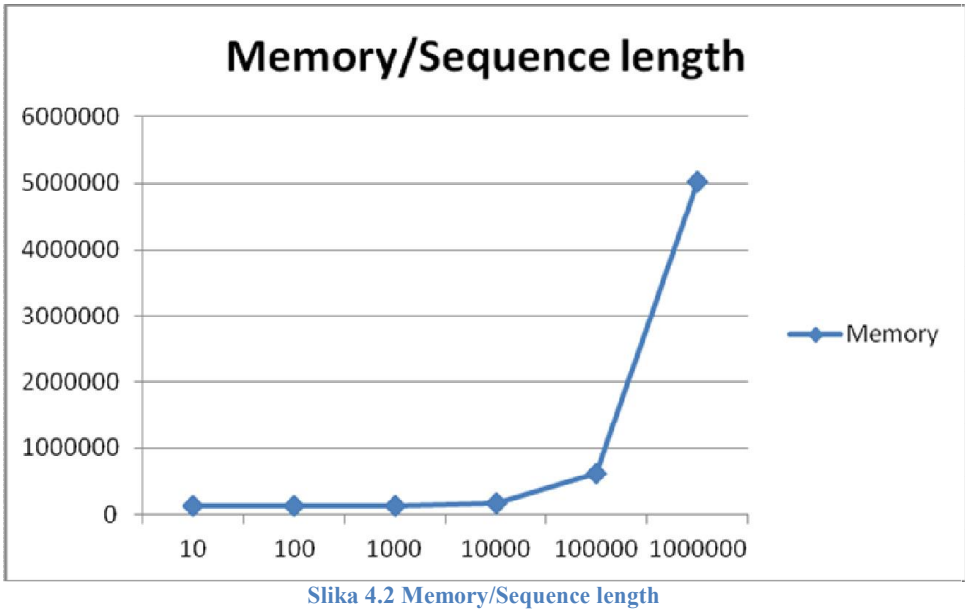
U nastavku su prikazani grafovi temeljeni na gornjim rezultatima testiranja.

Graf na slici ispod prikazuje ovisnost stvaranja indeksa o duljini sekvence (Sl. 4.1). Sve do veoma velikih duljina sekvenci, tj. do duljine 1000 000, vrijednost stvaranja indeksa je blizu nule.



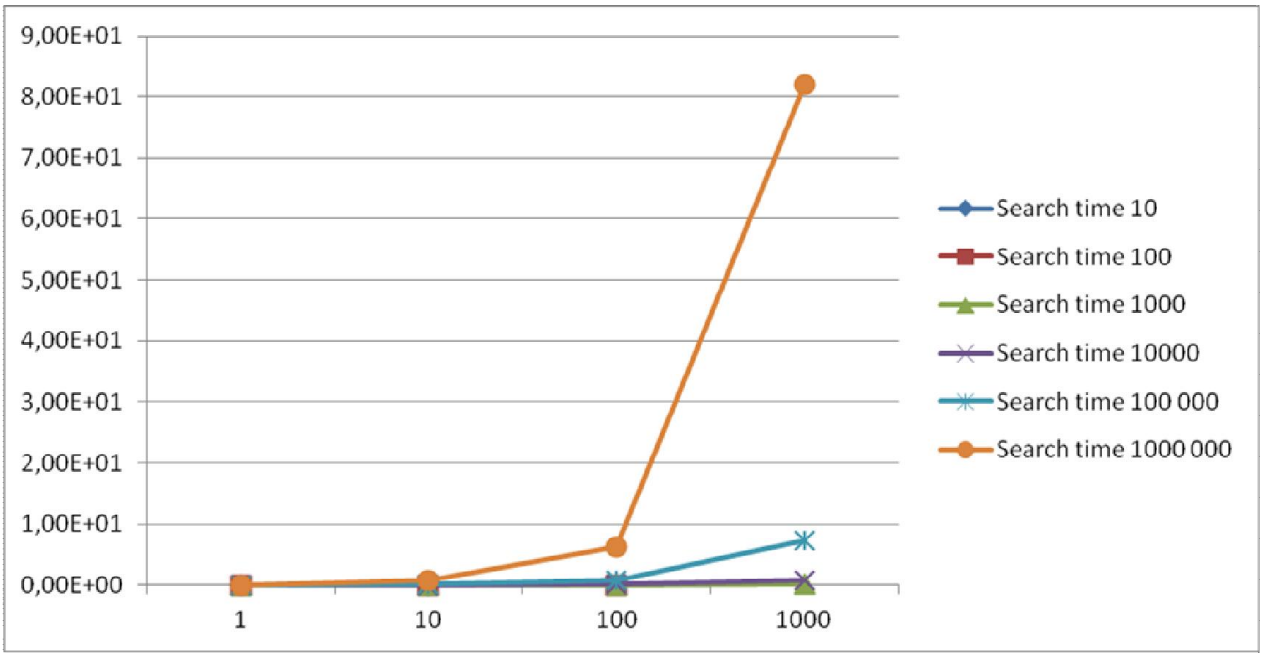
Slika 4.1 Index creation time/Sequence length

Na idućoj slici vidimo kako memorija ovisi o duljini sekvence (Sl. 4.2). I ovdje tek za veće vrijednosti duljine sekvence dolazi do skoka.



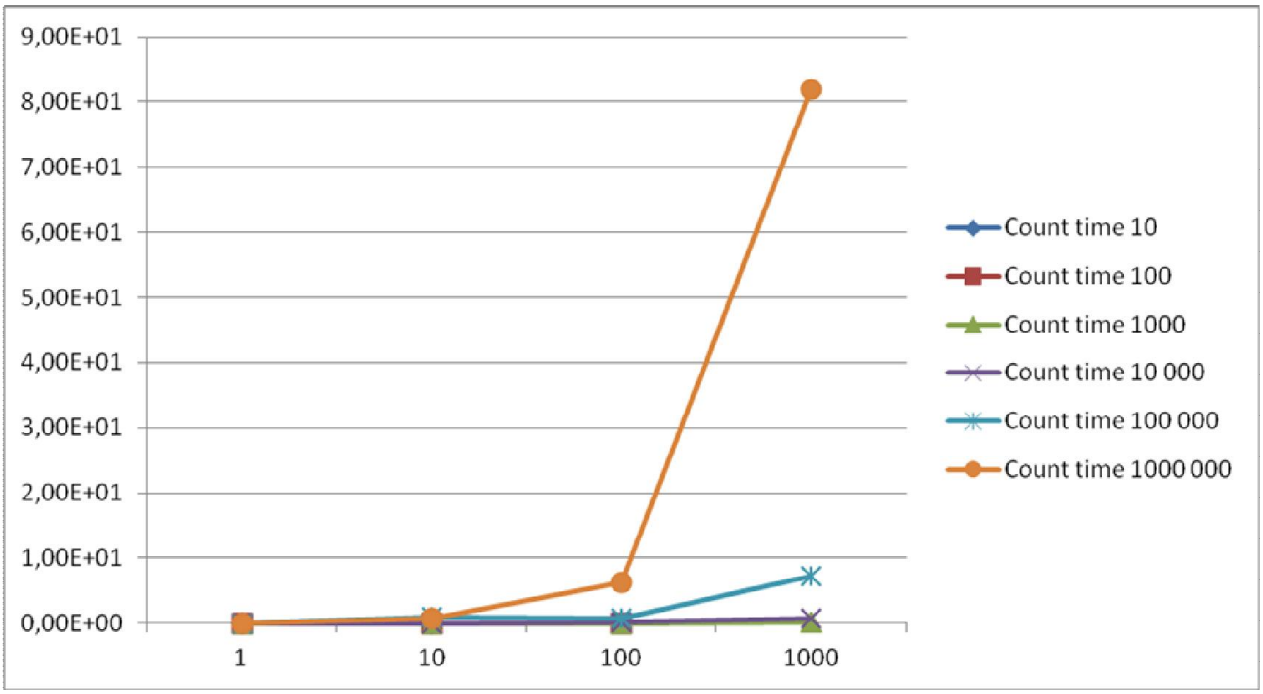
Slika 4.2 Memory/Sequence length

Slijede ovisnosti vremena pretraživanja o duljini traženog podniza (Sl. 4.3). Prikazana su vremena pretraživanja za sekvence duljine 10^1, 10^2, 10^3, 10^4, 10^5, 10^6, a u skladu s time su i nazvane vrijednosti na grafu (Search time 10, Search time 100...).



Slika 4.3 Search time/Pattern length

Posljednji graf prikazuje ovisnost *count time* o duljini traženog podniza (Sl. 4.4). Linija *Count time 10* prikazuje ovisnosti za sekvencu duljine 10^1, *Count time 100* za sekvencu duljine 10^2 itd.



Slika 4.4 Count time/Pattern length

6. Zaključak

Istraživanje u području genoma je prouzročilo stvaranje velikog broja podataka kojeg je trebalo nekako pohraniti te iz toga izvući samo potrebne dijelove. Osmišljen je vrlo djelotvoran postupak koji omogućava upravo pretraživanje informacija od interesa među velikim skupom podataka, odnosno FM-indeks.

Cilj ovoga projekta je bio implementirati FM-indeks te zadovoljiti zahtjeve za efikasno i brzo pretraživanje, kao i korištenje što manje memorije. FM-indeks sadrži sufiksno polje koje kreiramo koristeći Kärkkäinen-Sandersov algoritam koji se pokazao kao dobar odabir budući da prilično brzo obavi posao. FM-indeks još sadrži i LF-tablicu koja se sastoji od Prefix-Sum tablice i stabla valića.

Provedeno je nekoliko testiranja na temelju kojih smo zaključili da smo zadovoljili zahtjeve za brzinom i memorijskom učinkovitošću, a također smo provjerili i ispravnost programa.

7. Literatura

[1] *FM-Indexes and Backwards Search*: http://alexbowe.com/fm-index. Kolovoz 2011.
[2] Šikić, M. Domazet-Lošo, M. *Bioinformatika*: http://www.fer.unizg.hr/_download/repository/bioinformatika_skripta_v1.2.pdf. Zagreb, prosinac 2013.
[3] Domazet-Lošo, M. Samostojni indeksi. http://www.fer.unizg.hr/_download/repository/Bioinformatika_-_7._predavanje_%28Samostojni_indeksi%29.pdf. Zagreb, prosinac 2014.