SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKT IZ BIOINFORMATIKE

Računanje najduljeg zajedničkog prefiksa temeljeno na BWT

Autori
ZVONIMIR JURELINAC, TOMISLAV ŽIVEC, TONKO ČUPIĆ
Voditelj
doc.dr.sc MIRJANA DOMAZET- LOŠO

Zagreb, prosinac 2017.

Sadržaj

1 Uvod

Bioinformatika je grana znanosti koja usko povezuje biologiju i računarstvo, a ubrzano se razvijala zadnja dva desetljeća. Pojeftinjenje i sve veća dostupnost tehnologije sekvenciranja rezultirale su stvaranjem velikih skupova bioloških podataka. Često se kao zadatak u bioinformatici nameće analiza sekvence genoma. Pošto su te sekvence predugačke za uobičajenu pohranu i analizu, potrebno je koristiti posebna, prilagođene strukture podataka kao što su sufiksna polja i polja najdužih zajedničkih prefiksa.

Cilj projekta je bio čim učinkovitije implementirati algoritme 1 i 2 iz rada Beller et al. (2013), koristeći pritom gotovu knjižnicu za izgradnju sufiksnog polja, te potom ostvarenu implementaciju usporediti s originalnom, ali i s rezultatima prošlogodišnjeg studentskog tima, koji su opisani u njihovu radu Mrčela et al. (2017). Za usporedbu rezultata korišteni su sintetski podaci različitih duljina, kao i sekvenca genoma bakterije E. coli.

2 Algoritmi

TODO: Correct! U problemima analiza sekvenci vrlo često se javlja potreba za izračunom najduljeg zajedničkog prefiksa (LCP). U tu svrhu koriste sufiksni nizovi koji spremaju u listu spremaju sve moguće sufikse sekvence, od duljine 1 do najdulje. Iz sufiksnog niza dobiva se LCP polje u linearnom vremenu. Veliki resursni zahtjevi analize senkvenci DNK nameću zahtjeve za korištenjem podatkovnih struktura koje koriste manje memorijskog prostora. Iz te potrebe razvijeno je stablo valića niza transformiranog Burrows- Wheelerovom transformacijom (BWT). Metoda je sljedeća: sekvenca se prvo podliježe Burrows- Wheelerovoj transformaciji, potom se transformirana sekvenca sprema u stablo valića. Stablo valića podržava pretraživanje unatrag po originalnom nizu pa tako dobivamo tražene sufikse. Opisani algoritam ima složenost $O(n \log \sigma)$, gdje je σ veličina abecede.

2.1 Podatkovne strukture

2.1.1 Sufiksno polje

TODO: Correct! Sufiksno polje sadrži sve sufikse od teksta S. Zamijenila je u toj zadaći sufiksno stablo jer koristi manje memorije od sufiksnog stabla za istu zadaću. Sufiksno polje se pokazalo kao vrlo važan alat u raspoznavanju i analizi teksta, bioinformatici i drugim primjenama. Sufiksno polje SA od niza znakova S je cjelobrojno polje u intervalu od 1 do n koje određuje leksikografski poredak svih n sufiksa niza znakova S. Preciznije, sufiksno polje zadovoljava $S_{SA[1]} < S_{SA[2]} < ... < S_{SA[n]}$, gdje S_i označava i-ti sufiks niza znakova S, te sadrži znakove S[i..n].

2.1.2 Burrows-Wheelerova transformacija (BWT)

Burrows- Wheelerova transformacija pretvara niz znakova u sličan niz znakova BWT[1..n] sa istom abecedom. Koristi se za kompresiju podataka. Transformacija je reverzibilna bez dodatnih resursnih zahtjeva. Elemente transformiranog niza računamo formulom:

$$BWT[i] = \begin{cases} S[SA[i]-1], & \text{ako } SA[i] \neq 1 \\ \$, & \text{inače.} \end{cases}$$

Tablica 1: Pridruživanje indeksa sufiksima niza S, počevši od najduljeg.

i	S_{SA}[i]
1	abrakadabra$
2	brakadabra$
3	rakadabra$
4	akadabra$
5	kadabra$
6	adabra$
7	dabra$
8	abra$
9	bra$
10	ra$
11	a$
12	$

Tablica 2: Sufiksi su poredani leksikografski, a njihovi indeksi čine sufiksno polje SA.

i	SA[i]	S_{SA}[i]	BWT[i]
1	12	$	a
2	11	a$	r
3	8	abra$	d
4	1	abrakadabras$	$
5	6	adabra$	k
6	4	akadabra$	r
7	9	bra$	a
8	2	brakadabra$	a
9	7	dabra$	a
10	5	kadabra$	a
11	10	ra$	b
12	3	rakadabra$	b

2.1.3 Stablo valića

Stablo valića je struktura podataka koja rekurzivno particionira tok u 2 dijela sve dok su u svakom dijelu preostali homogeni podatci. Ime stabla je analogno valnoj transformaciji signala koji rekurzivno dekompresira signal prema frenkvencijama komponenti.

Stablo valića može efikasno raditi upite rank i select nad nizovima proizvoljnih abeceda. To nam omogućuje pretraživanje unatrag u vremenskoj složenosti od $O(\log \sigma)$ po koraku. Prvo definiramo uzlazno poredanu abecedu η kao niz znakova veličine σ. Zatim definiramo interval [1..r] kao podinterval abecede, r <= σ Za interval [1..r], niz znakova BWT[1..r] dobijemo tako da iz transformiranog niza znakova od S uklonimo sve znakove iz B-W transformacije koji ne pripadaju segmentu abecede [1..r]. Stablo valića niza znakova BWT izgrađeno nad abecedom η[1..$sigma$] je balansirano binarno stablo pretraživanja. Stablo se sastoji od čvorova, a svaki čvor v odgovara nizu znakova $BWT^{[l..r]}$, gdje je [l..r] abecedni interval. Korijen stabla odgovara nizu znakova BWT = $BWT^{[1..\sigma]}$. Ako je l=r, onda taj čvor v nema djece. Inače, svaki čvor v ima dvoje djece: lijevo dijete koje odgovara nizu znakova $BWT^{[l..m]}$, i desno dijete odgovara nizu znakova $BWT^{[m+1..r]}$, gdje je $\left\lceil \frac{l+r}{2} \right\rceil$. U ovom slučaju v sadrži bit vektor $B^{[l..r]}$, čiji je i-ti element 0 ako je i-ti znak u $BWT^{[l..r]}]$ iz pod- abecede η[1..m], a 1 ako je iz pod abecede η[m+1..r]. Drugim riječima, ako traženi znak pripada lijevom podstablu, element bit vektora je 0, a ako pripada desnom podstablu, onda je element bit vektora 1. Za svaki vektor B potrebno je napraviti predprocesiranje na način da se upiti $rank_0(B, i)$ i $rank_1(B, i)$ mogu odgovoriti u konstantnom vremenu, gdje $rank_b(B, i)$ predstavlja broj pojavljivanja bita b u B[1..i]. Stablo valića ima visinu $O(log\sigma)$. S obzirom da je u implementaciji dovoljno spremiti samo bit vektore, stablo valića zahtijeva samo nlog σ bitova prostora i $O(n * log\sigma)$ bitova za podatkovne strukture koje podržavaju rang upite u konstantnom vremenu.

3 Primjer rada algoritma

U nastavku dokumenta dan je primjer rada algoritma na stringu S = mississippi.

3.1 Izgradnja sufiksnog polja

1. Na kraj ulaznog niza dodaje se znak $ te je sada S = mississippi$. U daljnjem tekstu vrijedi pretpostavka da je znak $ abecedno manji od svih ostalih znakova od kojih je S izrađen.

2. Svakom sufiksu niza S pridružuju se indeksi od 1 do n, počevši od najduljeg. Ovo je prikazano u **tablici 3**

Tablica 3: Pridruživanje indeksa sufiksima niza S

i	S_{SA}[i]
1	mississippi$
2	ississippi$
3	ssissippi$
4	sissippi$
5	issippi$
6	ssippi$
7	sippi$
8	ippi$
9	ppi$
10	pi$
11	i$
12	$

3. Sufiksno polje SA dobivamo soritiranjem dobivenih sufiksa leksikografski od najmanjeg prema najvećem. Ovo je prikazano u **tablici ??**.

   Dobiveno je sufiksno polje SA = [12, 11, 8, 5, 2, 1, 10, 9, 7, 4, 6, 3]

3.2 Burrows-Wheelerova transformacija

Izračunava se Burrows-Wheelerova transformacija BWT[1..n] za dobiveno sufiksno polje SA prema formuli navedenoj u **??** . Npr.

$$BWT[4] = S[SA[4] - 1] = S[5 - 1] = S[4] = s.$$

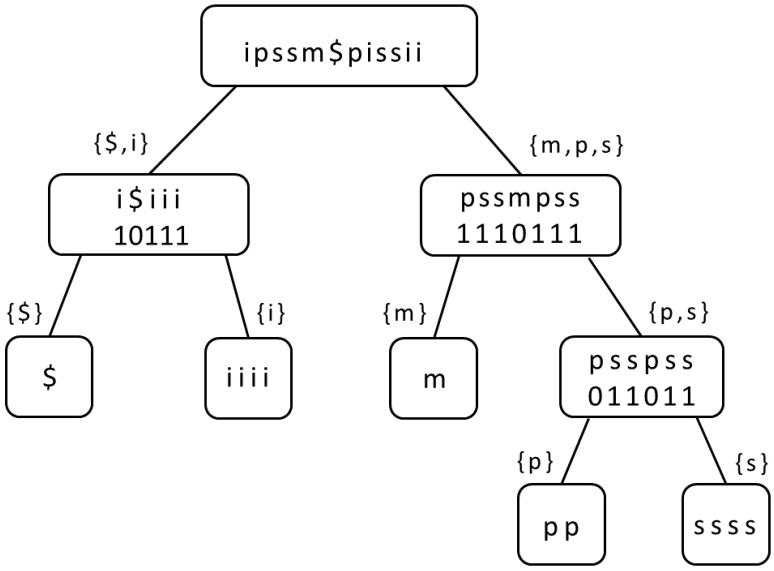
Rezultat je prikazan u tablici **tablice 5**.

Tablica 4: Sufiksi su poredani leksikografski, a njihovi indeksi čine sufiksno polje SA.

i	SA[i]	S_{SA}[i]
1	12	$
2	11	i$
3	8	ippi$
4	5	issippi$
5	2	ississippi$
6	1	misssissippi$
7	10	pi$
8	9	ppi$
9	7	sippi$
10	4	sissippi$
11	6	ssippi$
12	3	ssissippi$

Tablica 5: Burrows-Wheelerova tranformacija

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



Slika 1: Stablo valića za dani primjer

3.3 Izgradnja LCP niza

1. Iz dobivene Burrows-Wheelerove transformacije gradi se stablo valića opisanom u **??**. Prvo se stvara sortirana abeceda ulaznog niza koja je u ovom slučaju veličine 5 znakova (Σ[1..5]=$imps). Korijen stabla čini bit vektor dobiven kodiranjem niza Burrows-Wheelerove transformacije dobivene u prethodnom koraku. Abeceda se potom dijeli na pola te dobivamo dva podniza, ovom slučaju: Σ[1..2]=$i i Σ[3..5]=mps. Znakovi prve polovice u svakoj skupini kodiraju se vrijednošću 0, a ostali 1. Postupak se ponavlja sve dok se u čvoru ne nalaze samo jednaki znakovi koji čine listove stabla. Potpuno izgrađeno stablo za dani primjer prikazano je na slici 1.

2. Pomoću Algoritama 1 i 2 iz rada Beller et al. (2013) gradi se polje najdužih zajedničkih prefiksa (LCP) u nekoliko koraka:

   (a) Inicijalizacija LCP polja i reda Q. Vrijednosti polja LCP se postavljaju na nevažeću ($\perp$) vrijednost, osim LCP[1] i LCP[n+1] koji se postavljaju na -1. U red Q stavljamo strukturu koja sadrži početni interval $I = [i..j] = [1..12]$ i broj $l = 0$:

$$LCP = [-1, \perp, \perp, \perp, \perp, \perp, \perp, \perp, \perp, \perp, \perp, \perp, \perp, -1]$$

$$Q = [< [1..12], 0 >]$$

(b) Izračunavanje cω-intervala za interval dobiven uzimanjem elementa iz reda Q (FIFO) unutar funkcije *getIntervals* iz Algoritma 1 Beller et al. (2013).

- c - znak
- *C[c]* - zbroj rangova svih elemenata leksikografski sortirane abecede koji su manji od c
- i - početak intervala
- j - kraj intervala
- Funkcija *rang(a,k)* vraća broj pojavljivanja znaka a do k-tog indeksa u polju.
- Znakovi abecede iz intervala I se sortiraju od najmanjeg prema najvećem. Ima onoliko cω-intervala koliko je i jedinstvenih znakova. Kako je abeceda za dani primjer $\Sigma[1..5] = \$imps$, traži se 5 c$\omega$-intervala.
- $C = [0, 1, 5, 6, 8]$
- Indeks početka intervala dobiva se po formuli *rang*(c,i-1)+C[c]+1 Indeks kraja intervala dobiva se po formuli C[c]+*rang*(c,j).
- U intervalu I se nalaze svi znakovi abecede ($,i,m,p,s). Za svaki od znakova računa se njegov interval prema formuli navedenoj iznad.

  $[rang('\$', 0) + C['\$'] + 1..C['\$'] + rang('\$', 12)] = [0+0+1..+0+1] = [1..1]$

  $[rang('i', 0) + C['i'] + 1..C['i'] + rang('i', 12)] = [0+1+1..+1+4] = [2..5]$

  $[rang('m', 0) + C['m'] + 1..C['m'] + rang('m', 12)] = [0+5+1..+5+1] = [6..6]$

  $[rang('p', 0) + C['p'] + 1..C['p'] + rang('p', 12)] = [0+6+1..+6+2] = [7..8]$

  $[rang('s', 0) + C['s'] + 1..C['s'] + rang('s', 12)] = [0+8+1..+8+4] = [9..12]$
- Povratna vrijednost *getIntervals*, prema tome, je lista cω-intervala: [[1..1],[2..5],[6..6],[7..8],[9..12]]
- Za svaki od dobivenih intervala [*lb .. rb*] se potom provjerava vrijednost LCP[*rb*+1]. Ako je vrijednost tog polja **NULL** u red stavljamo strukturu [<[*lb..rb*],*l*+1>], a na index *rb*+1 vrijednost *l*. Navedeni postupak se ponavlja sve dok u redu ima elemenata, a prva dva koraka prikazana su u nastavku.

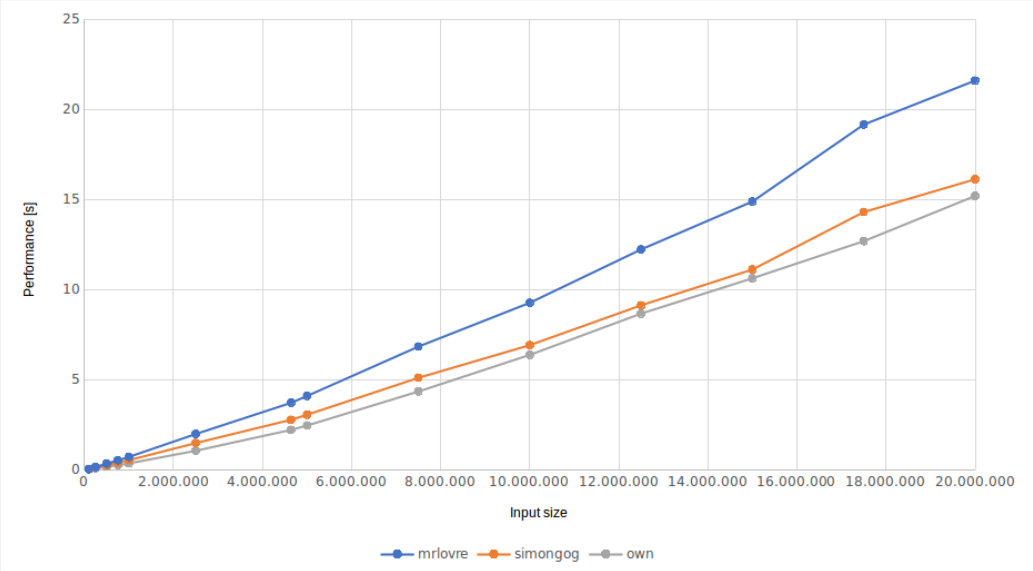
i. LCP = [-1,⊥, ⊥, ⊥, ⊥, ⊥, ⊥, ⊥, ⊥, ⊥, ⊥, ⊥, -1] Q = [<[1..1],0>]
[*lb* .. *rb*] = [1..1], *l* = 0
LCP[*rb*+1]=LCP[2]=⊥ -> u red Q stavljamo strukturu
[<[*lb*..*rb*],*l*+1>] = [<[*1*..*1*],1>], a LCP[*rb*+1]=LCP[2]=0.

ii. LCP = [-1,0,⊥, ⊥, ⊥, ⊥, ⊥, ⊥, ⊥, ⊥, ⊥, ⊥, -1] Q = [<[2..5],0>,<[1..1],1>]
[*lb* .. *rb*] = [2..5], *l* = 0
LCP[*rb*+1]=LCP[6]=⊥ -> u red Q stavljamo strukturu
[<[*lb*..*rb*],*l*+1>] = [<[*2*..*5*],1>], a LCP[*rb*+1]=LCP[6]=0.

iii. LCP = [-1,0,⊥, ⊥, ⊥,0, ⊥, ⊥, ⊥, ⊥, ⊥, ⊥, -1] Q = [<[6..6],0>,<[2..5],1>,<[1..1],1>
[*lb* .. *rb*] = [6..6], *l* = 0
(...)

(c) Vrijednost LCP polja koju dobijemo kao rezultat izvršavanja algoritma je: LCP = [-1,0,1,1,4,0,0,1,0,2,1,3,-1].

4 Rezultati



Slika 2: Grafički prikaz usporedbe vremena izvođenja različitih implementacija algoritama, prema podacima iz tablice 6. Sivom bojom prikazana je ovdje opisana implementacija.

Tablica 6: Rezultati usporedbe vremena izvođenja algoritama

Duljina ulaza [znak]	Prošlogodišnja implementacija [s]	Originalna implementacija* [s]	Naša implementacija [s]
100.000	0,04	0,03	0,03
250.000	0,16	0,12	0,09
500.000	0,35	0,26	0,17
750.000	0,53	0,40	0,26
1.000.000	0,72	0,54	0,36
2.500.000	1,99	1,49	1,06
4.639.675	3,72	2,78	2,22
5.000.000	4,1	3,06	2,46
7.500.000	6,84	5,11	4,35
10.000.000	9,27	6,92	6,38
12.500.000	12,23	9,13	8,67
15.000.000	14,89	11,12	10,63
17.500.000	19,16	14,30	12,69
20.000.000	21,6	16,13	15,2

** Vrijednosti interpolirane na temelju rezultata prošlogodišnjeg rada*

5 Zaključak

lalala

Tablica 7: Rezultati usporedbe vremena izvođenja algoritama

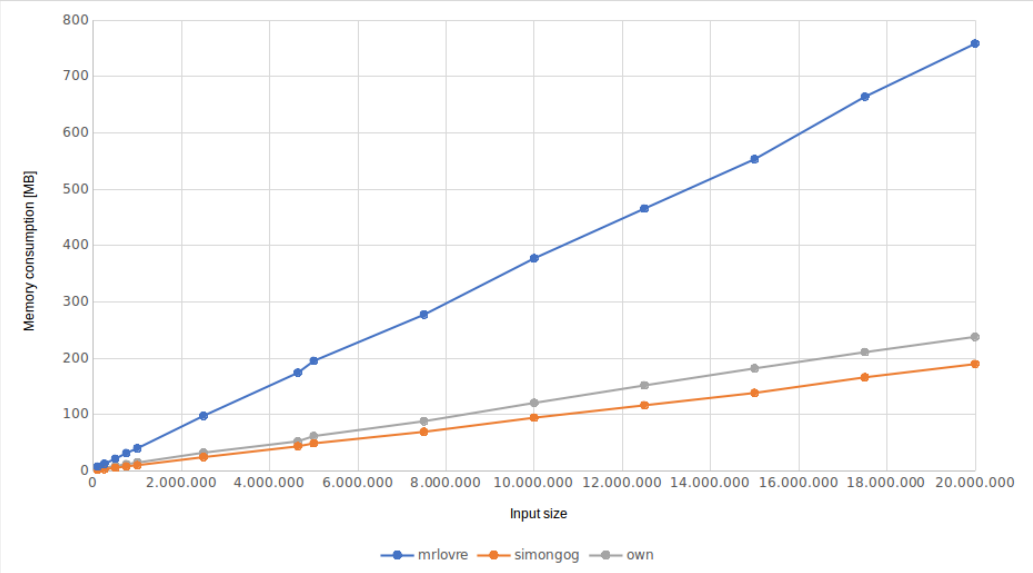
Duljina ulaza [znak]	Prošlogodišnja implementacija [MB]	Originalna implementacija* [MB]	Naša implementacija [MB]
100.000	7,32	1,83	4,328
250.000	12,428	3,107	5,908
500.000	21,388	5,347	8,56
750.000	31,592	7,898	11,92
1.000.000	40,024	10,006	14,656
2.500.000	97,68	24,42	32,304
4.639.675	174,224	43,556	52,372
5.000.000	195,212	48,803	61,772
7.500.000	277,296	69,324	88,064
10.000.000	377,372	94,343	120,644
12.500.000	465,616	116,404	151,74
15.000.000	553,476	138,369	181,98
17.500.000	664,256	166,064	210,7
20.000.000	758,572	189,643	238,004

** Vrijednosti interpolirane na temelju rezultata prošlogodišnjeg rada*

6 Literatura

T. Beller, S. Gog, E. Ohlebusch, i T. Schnattinger. *Computing the longest common prefix array based on the Burrows-Wheeler transform*. Elsevier B.V., 2013.

L. Mrčela, A. Škaro, i A. Žužul. *Računanje najduljeg zajedničkog prefiksa temeljeno na BWT*. 2017.



Slika 3: Grafički prikaz usporedbe memorijskog zauzeća različitih implementacija algoritama, prema podacima iz tablice 7. Sivom bojom prikazana je ovdje opisana implementacija.