SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

BIOINFORMATIKA - PROJEKT

essaMEM: finding maximal exact matches using enhanced sparse suffix arrays

Daniel Guja, Antun Razum, Petra Rebernjak
Voditelj: *doc. dr. sc. Mirjana Domazet-Lošo*

Zagreb, siječanj 2017.

SADRŽAJ

1. Uvod

Maksimalno točno slaganje (engl. *Maximal Exact Matches, MEM*) je točno slaganje između dva niza koje se ne može produžiti lijevo niti desno bez da se dogodi neslaganje između ta dva niza. Za pronalaženje MEM-ova potrebna je pogodna struktura podataka. Jedna od njih je sufiksno stablo. To je osnovna struktura podataka koja omogućuje brzu analizu znakova.

Sufiksno stablo ima široku primjenu u bioinformatici. Mnoštvo primjena opisano je posebno početkom 21. stoljeća. Kako je potreba za korištenjem sufiksnih stabala rasla, bilo je potrebno pronaći novu strukturu podataka koja će unaprijediti već postojeću. Tako se došlo do alternativne strukture podataka, sufiksnog polja. Sufiksno polje je zamijelo sufiksno stablo. Glavni razlog tome je memorijski otisak sufiksnog stabla. Sufiksno stablo za ulazni niz od n znakova zahtijeva najmanje $10n$ okteta memorijskog prostora u praksi, a često i $15n-20n$ okteta. Sufiksno polje teorijski zauzima $O(nlogn)$ bita, a u praksi obično $4n$ okteta.

Uobičajeno se koriste sufiksna stabla ili unaprijeđena sufiksna polja (engl. *Enhanced Suffix Arrays, ESA*) za pronalaženje MEM-ova. ESA se sastoji od četiri polja: sufiksnog polja, polja najvećih zajedničkih prefiksa (engl. *Longest Common Prefix, LCP*), child polja i polja sufiksnih poveznica (engl. *suffix link arrays*) koja sadrže dijelove informacija iz sufiksnog stabla te zajedno dostižu potpunu ekspresivnost sufiksnih stabala. Dakle, svaki problem koji se može riješiti korištenjem sufiksnih stabala, može se riješiti i korištenjem ESA s istom asimptotskom složenošću.

Khan *et. al.* u [2] savjetuje korištenje rijetkih sufiksnih polja (engl. *Sparse Suffix Arrays, SSA*). Kod SSA, indeksira se svaki *K*-ti sufiks. Parametar *K* naziva se parametar rijetkosti, *sparsness factor*. Njihov algoritam za pronalaženje MEM-ova koji se temelji na SSA pronalazi MEM-ove puno brže od prethodnih metoda te koristi manje memorije. Posljedično, uporabom SSA moguće je indeksirati znatno veće genome.

Vyverman *et. al.* u [3] želi poboljšati metodu predstavljenu u [2] implementiranjem rijetkih child polja za velike parametre rijetkosti. Dodatno, korištenjem sufiksnih poveznica (engl. *suffix links*) i child polja gradi se unaprijeđeno rijetko sufiksno polje

(engl. *Enhanced Sparse Suffix Array, ESSA*) koje ima istu ekspresivnost kao i sufiksno stablo za podnizove veće od *K*. Memorijski učinkoviti algoritmi za pronalaženje MEM-ova mogu se podijeliti u dvije kategorije: *online* i *index* metode. essaMEM pripada kategoriji indeksnih metoda algoritama za pronalaženje MEM-ova. Ova metoda uspoređuje niz Q veličine m niz s indeksnom strukturom izgrađenom nad referentnim nizom S veličine n. Prednost ovog pristupa je ta što se konstruirana indeksna struktura može ponovno iskoristiti, odnosno za sljedeću usporedbu nizova nije potrebno ponovno indeksirati niz.

Cilj ovog projektnog zadatka je izgraditi rijetko child polje.

2. Child polje

MEM-ovi se pronalaze obilaskom sufiksnog stabla odozgo prema dolje. Koristeći SA i LCP polje ovaj problem moguće je riješiti u složenosti $O(m + logn)$. Ukoliko se SA proširi dodatnim child poljem problem je moguće riješiti u složenosti $O(m)$.

Dohvat djece unutarnjih čvorova konceptualnog sufiksnog stabla ostvarenog pomoću SA i LCP polja moguće je obaviti u konstantnom vremenu koristeći child polje. Za izgradnju child polja koristi se informacija o LCP intervalima, definiranima u [1]. Bitno je naglasiti da su LCP intervali samo konceptualni te se nikada ne konstruiraju niti čuvaju u memoriji.

Child polje je duljine n/K, a svaki element u polju sadrži vrijednosti *up*, *down* i *nextLIndex*. Te vrijednosti definirane su u nastavku:

$childtab[i].up = min\{q \in [0..i-1] \mid lcptab[q] > lcptab[i]$ i $\forall k \in [q+1..i-1] : lcptab[k] \geq lcptab[q]\}$

$childtab[i].down = max\{q \in [i+1..n] \mid lcptab[q] > lcptab[i]$ i $\forall k \in [i+1..q-1] : lcptab[k] > lcptab[q]\}$

$childtab[i].nextLIndex = min\{q \in [i+1..n] \mid lcptab[q] = lcptab[i]$ i $\forall k \in [i+1..q-1] : lcptab[k] > lcptab[i]\}$

Tablica 2.1 prikazuje vrijednosti sufiks i LCP polja te sve tri vrijednosti child polja na primjeru *S = mississippi*.

Tablica 2.1: SA niza *S = mississippi* proširen s LCP i child tablicom.

i	sa[i]	lcp[i]	up[i]	down[i]	nextLIndex[i]	s[sa[i]]
0	10	0		1	2	i$
1	4	1				issippi$
2	0	0	1		3	mississippi$
3	8	0			4	ppi$
4	6	0		5		sippi$
5	2	1				ssissippi$

Child polje nosi informaciju o odnosu roditelj-dijete u LCP intervalima. Za LCP interval [i..j] s *l*-indeksima $i_1 < i_2 < ... < i_k$, $childtab[i].down$ ili $childtab[j+1].up$ vrijednost koristi se za dobivanje prvog *l*-indeksa i_1. Ostali *l*-indeksi i_2, ..., i_k dobivaju se redom iz $childtab[i_1].nextLIndex$, ..., $childtab[i_{k-1}].nextLIndex$. U [1] je dokazano da su sada child intervali LCP intervala [i..j] redom $[i..i_1-1]$, $[i_1..i_2-1]$, ..., $[i_k..j]$.

3. Konstrukcija child polja

Algoritam 3.1 prikazuje izgradnju rijetkog child polja. Originalni algoritam dokazan je u [1], a autori u [3] modificiraju ga za rad s rijetkim sufiksnim poljem. U poglavlju 2 pokazano je da je za izgradnju child polja potrebno imati samo informaciju sadržanu u LCP intervalima. Obzirom da uvođenje parametra rijetkosti K ne utječe na definiciju LCP intervala, moguće je izgraditi rijetko child polje.

U prvom linearnom prolazu LCP polja računaju se *up* i *down* vrijednosti. Trenutni indeks stavlja se na vrh stoga ukoliko je njegova LCP-vrijednost veća ili jednaka LCP-vrijednosti s vrha stoga. U suprotnom, elementi se skidaju sa stoga dok su njihove LCP-vrijednosti veće od LCP-vrijednosti trenutnog indeksa. Usporedbom LCP-vrijednosti trenutnog indeksa i indeksa s vrha stoga, *up* i *down* vrijednostima se pridjeljuju elemenati skinutih sa stoga tijekom prolaska. **TODO: dodati opis čišćenja stoga**

U drugom linearnom prolazu LCP polja računaju se *nextLIndex* vrijednosti. Izračun *nextLIndex* vrijednosti je znatno intuitivniji. Potrebno je usporediti LCP vrijednost trenutnog indeksa s LCP vrijednošću indeksa s vrha stoga. Ukoliko su navedene vrijednosti jednake, *nextLValue* u child polju na indeksu s vrha stoga postaje trenutni index.

Algoritam 3.1: Algoritam za konstrukciju child polja

```
1  function constructChildArray(LCP) begin
2      lastIndex = −1;
3      ST.push(0);
4      for i = 1 to n/K − 1 do
5          while LCP[i] < LCP[ST.top] do
6              lastIndex = ST.pop;
7              if LCP[i] ≤ LCP[ST.top] and LCP[ST.top] ≠ LCP[lastIndex]
                then
8                  CHILD[ST.top].down = lastIndex;
9          if lastIndex ≠ −1 then
10             CHILD[i].up = lastIndex;
11             lastIndex = −1;
12         ST.push(i);
13     while 0 < LCP[ST.top] do
14         lastIndex = ST.pop;
15         if 0 ≤ LCP[ST.top] and LCP[ST.top] ≠ LCP[lastIndex] then
16             CHILD[ST.top].down = lastIndex
17     for i = 1 to n/K − 1 do
18         while LCP[i] < LCP[ST.top] do
19             ST.pop;
20         if LCP[i] = LCP[ST.top] then
21             lastIndex = ST.pop;
22             CHILD[lastIndex].nextLIndex = i;
23         ST.push(i);
```

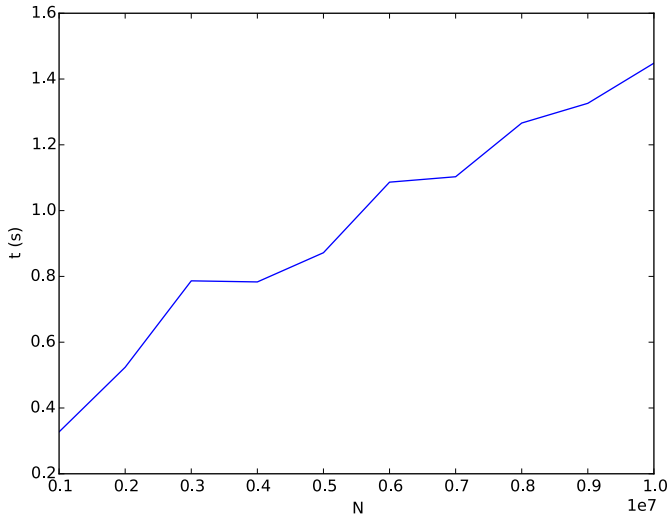
4. Rezultati

Tablica 4.1: Usporedba vremena izvršavanja i utrošene memorije izvedene implementacije i referentnog rješenja za različite primjere i vrijednosti parametra k

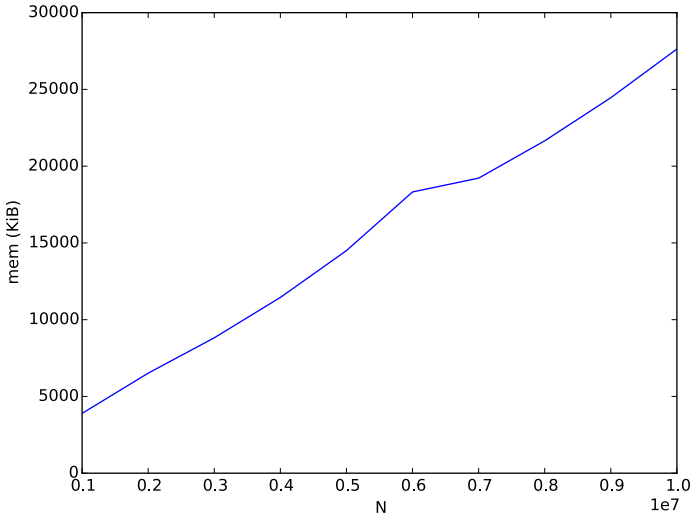
Primjer	Broj baza	K	t_{impl}	t_{ref}	mem_{impl}	mem_{ref}
E. coli #1	5676107	5	0.709326	0.9197	26580	29720
E. coli #1	5676107	20	5.16589	5.30045	12352	13584
E. coli #1	5676107	100	8.82316	8.95225	9616	13580
E. coli #2	4393089	5	0.553883	0.701784	20488	25180
E. coli #2	4393089	20	4.12148	4.0905	10916	11676
E. coli #2	4393089	100	7.17556	6.75088	10600	11388
Gen #1	100	5	0	0.004481	1476	9808
Gen #2	1000	5	0	0	1480	9816
Gen #2	1000	20	0.005451	0.004367	1512	1648
Gen #3	10000	5	0.00917	0.001312	1504	9840
Gen #3	10000	20	0.015292	0.018068	1524	1664
Gen #3	10000	100	0.027169	0.024442	1636	1752
Gen #4	100000	5	0.012961	0.014353	1884	10160
Gen #4	100000	20	0.124132	0.117844	1668	1832
Gen #4	100000	100	0.178385	0.174271	1736	1848
Gen #5	1000000	5	0.100779	0.0977	5296	13208
Gen #5	1000000	20	1.13231	1.03717	3100	3432
Gen #5	1000000	100	1.59657	1.71334	2576	3436

TODO: pojasniti tablicu 4.1

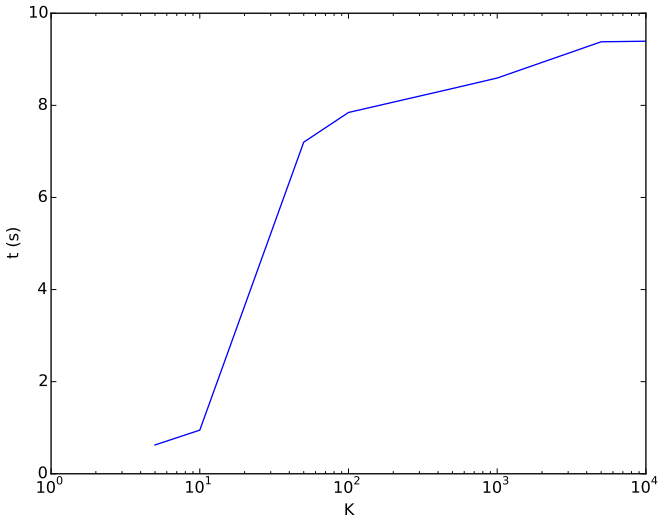
TODO: komentirati grafove



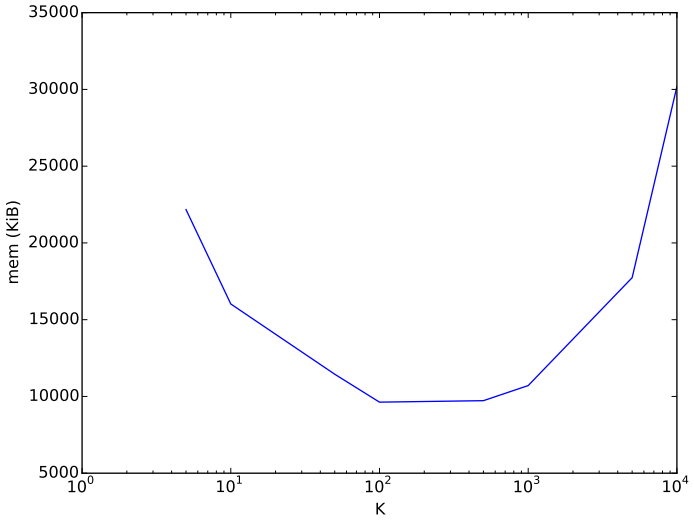
Slika 4.1: Ovisnost vremena izvršavanja o broju baza ($K = 10$)



Slika 4.2: Ovisnost utrošene memorije o broju baza ($K = 10$)



Slika 4.3: Ovisnost vremena izvršavanja o parametru K ($N = 5 \cdot 10^6$)



Slika 4.4: Ovisnost utrošene memorije o parametru K ($N = 5 \cdot 10^6$)

5. Zaključak

TODO: napisati...

6. Literatura

[1] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, i Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.

[2] Zia Khan, Joshua S Bloom, Leonid Kruglyak, i Mona Singh. A practical algorithm for finding maximal exact matches in large sequence datasets using sparse suffix arrays. *Bioinformatics*, 25(13):1609–1616, 2009.

[3] Michaël Vyverman, Bernard De Baets, Veerle Fack, i Peter Dawyndt. essamem: finding maximal exact matches using enhanced sparse suffix arrays. *Bioinformatics*, 29(6):802–804, 2013.

essaMEM: finding maximal exact matches using enhanced sparse suffix arrays

Sažetak

TODO: napisati...

Ključne riječi: MEM, enhanced sparse suffix array, child array