SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Implementacija FM-indeks algoritma

Ivan Borko, Sofia Čolaković, Florijan Stamenković
Voditelj: *doc. dr. sc. Mirjana Domazet Lošo*

Zagreb, siječanj 2015.

SADRŽAJ

1. Uvod i problematika

Pretraživanje teksta česta je praktična potreba mnogih informacijskih sustava. Pod terminom "pretraživanje teksta" podrazumijevamo pronalazak svih pojavljivanja nekog niza znakova *Q* (engl. *query*) unutar drugog niza znakova *S* (engl. *string*). Tipično se rezultat pretraživanja *R* formulira kao niz indeksa (rednog broja znaka) unutar niza *S* na kojem počinje pojavljivanje niza *Q*. Primjerice, za niz *S = "Žuti pas je opasan kad je opasan remenom oko pasa"* i niz *Q = "pas"* rezultati pretraživanja su *R = {6, 14, 28, 46}*.

U području bioinformatike pretraživanje teksta koristi se u za pronalazak specifičnih sekvenci unutar zadanog genoma. Definicija pretraživanja je jednaka. Specifičnost bioinformatičkog pretraživanja jest da su nizovi koji se pretražuju iznimno velike duljine. Primjerice, ljudski genom tipično sadrži oko 3.3×10^9 znakova, što bi otisnuto na A4 stranice fontom veličine 10pt rezultiralo s otprilike milijun stranica.

Postoje mnogi algoritmi pretraživanja teksta koji na jednostavan način ispunjavaju definirane zahtjeve. Iz perspektive računalne složenosti algoritama, jednostavno je implementirati pretraživanje teksta koje radi u linearnom vremenu[1]. Nažalost, za nizove vrlo velike duljine linearno vrijeme znači praktično predugo trajanje pretraživanja. Otud potreba, pogotovo u području bioinformatike, za vremenski sub-linearnim algoritmima pretraživanja.

U ovom projektu razmatramo implementaciju pretraživanja teksta koja se bazira na konceptu FM-indeksa. Konkretna implementacija bazira se na binarnim stablima valića (engl. *wavelet-trees*). Teoretsko razmatranje i praktično testiranje pokazuju da ovakva implementacija pretraživanja ima vremenski sub-linearnu složenost.

[1] Ako nije drukčije navedeno pri razmatranju složenosti pretraživanja uvijek govorimo o složenosti s obzirom na duljinu pretraživanog niza *S*.

2. FM-indeks algoritam

"Indeksiranje" teksta označava generiranje struktura podataka koje su podrška efikasnom pretraživanju. Za velike tekstove poželjno je da indeks bude memorijski efikasan. FM-indeks [1] pristup je indeksiranju koji ispunjava zahtjeve memorijske efikasnosti i sub-linearnog vremena pretraživanja. Prije nego definiramo FM-indeks, potrebno je razmotriti podatkovne strukture i algoritme koji ga sačinjavaju.

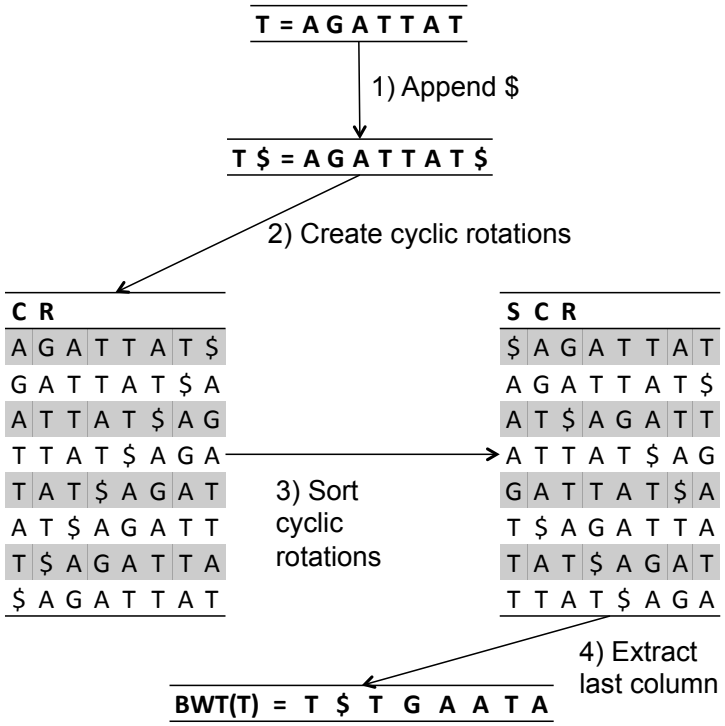
2.1. Burrows-Wheeler transformacija (BWT)

Burrows-Wheeler transformacija [2] transformira niz znakova na način koji će omogućiti efikasnu pohranu i brzo ptreživanje. BWT transformirani niz originalnog teksta T označavati ćemo sa T^{BTW}. Transformacija se provodi sljedećim koracima:

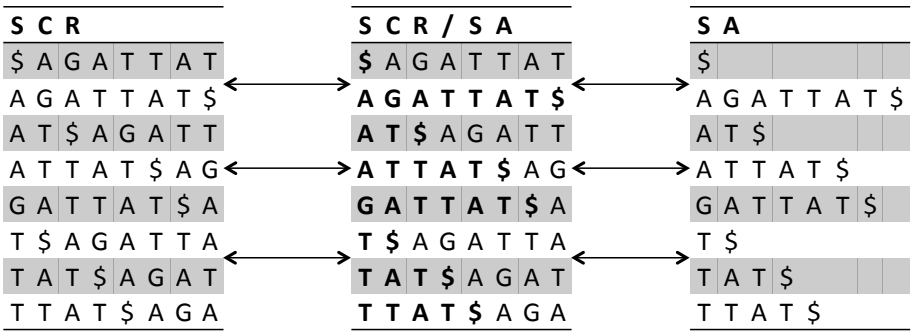
1. Poseban znak '$' koji je leksikografski manji od svih ostalih znakova se dodaje na kraj niza T
2. Cikličkim rotacijama niza T dobiva sa se skup nizova koji čini tablicu CR (engl. *cyclic rotation*)
3. Tablica CR se leksikografski sortira u tablicu SCR
4. Posljednji stupac tablice SCR se ektrahira u rezultat T^{BTW}

Opisani postupak ilustriran je za niz T = *"AGATTAT"* na slici 2.1, preuzetoj iz rada [3].

Bitno je primjetiti kako je BWT transformacija niza srodna sufiksnoj listi SA (engl. *suffix array*). Sufiksna lista je struktura podataka koja se često koristi u algoritmima nad tekstom. Njenu formulaciju nećemo detaljno objašnjavati, materijali na temu su široko dostupni. Sličnost između BWT transformacije i SCR tablice korištene u BWT transformaciji ilustrirana je slikom 2.2.



Slika 2.1: Primjer algoritma BWT transormacije niza $T = AGATTAT$



Slika 2.2: SCR tablica BWT tranfsformacije i sufiksna lista za niz $T = AGATTAT$

2.1.1. LF-mapiranje

LF-mapiranje (engl. *last-to-first mapping*) opisuje relaciju između posljednjeg stupca *SCR* tablice (označenog *L*) i prvog stupca te iste tablice (označenog *F*)[1]. LF-mapiranje postulira da *i*-to pojavljivanje znaka *c* unutar stupca *L* korespondira *i*-tom pojavljivanju tog istog znaka unutar stupca *F*. Pri tome "korespondira" znači da se radi o istom znaku unutar originalnog niza *T*. Jednostavan dokaz ovog iskaza moguće je naći u [3].

Brza implementacija LF-mapiranja (konstantne vremenske složenosti) može se ostvariti implementacijom dviju pomoćnih tablica. Tablica prefiksnih suma *C* (engl. *prefix-*

[1] Primjetimo da je *L* isti stupac koji definira BWT transformaciju.

sum table) niza T za svaki znak c pohranjuje broj znakova u T koji su manji od c. Tablica pojavljivanja Occ (engl. *occurrence table*) pohranjuje informaciju koliko puta se neki znak c pojavio u nizu T do pozicije i (isključujući znak točno na poziciji i). Korištenjem tablica C i Occ LF mapiranje za T^{BWT} (koji odgovara stupcu L) računa se na sljedeći način, za znak c na poziciji i:

1. Pronađi broj pojavljivanja c u T^{BWT} do pozicije i unutar tablice Occ
2. Pronađi broj znakova manjih od c u T^{BWT} unutar tablice C
3. Zbroj pronađenih vrijednosti je indeks korespondirajućeg znaka u stupcu L

2.1.2. Rekonstrukcija originala

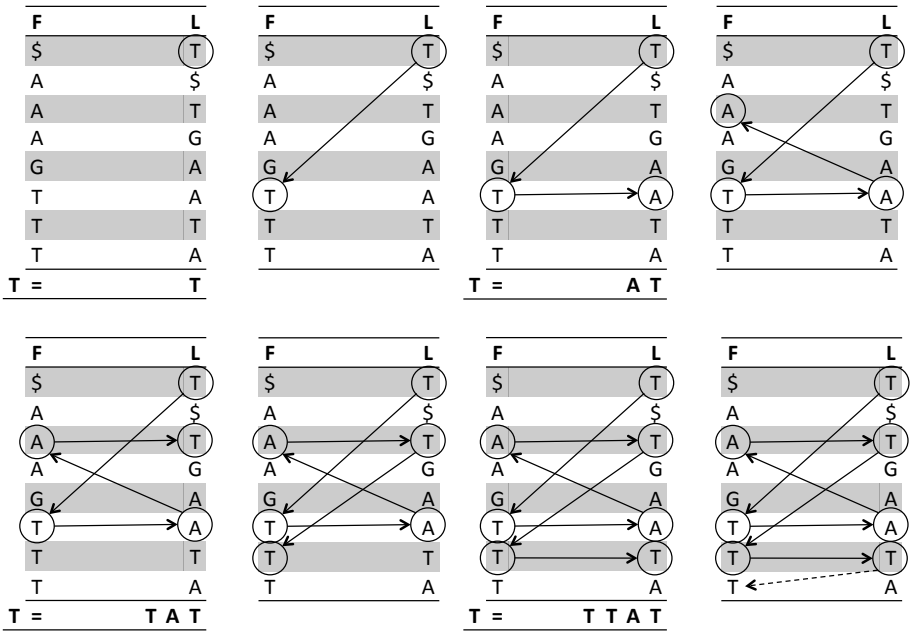
Na temelju transformiranog niza T^{BWT} moguće je rekonstruirati originalni niz T korištenjem LF-transformacije. Postupak je jednostavan, ako imamo na umu definiciju LF-transformacije i činjenicu da je prvi znak u T^{BWT} zasigurno posljednji znak niza T (ovo proizlazi iz činjenice da smo pri postupku BWT transormacije na početak niza umetnuli znak '$').

Rekonstrukcija niza T obavlja se unatrag, od posljednjeg znaka prema prvom. Postupak je sljedeći:

1. Prvi znak iz T^{BWT} je posljednji znak iz T, zabilježimo ga
2. LF-transformacijom pronađimo F-indeks posljednjeg zabilježenog znaka [2]
3. Dobiveni F-indeks u T^{BWT} nizu ukazuje na znak koji u nizu T prethodi posljednjem zabilježenom znaku (posljedica rotacije pri konstrukciji SCR)
4. Zabilježimo znak na F-indeks poziciji niza T^{BWT} u rekonstrukciju
5. Ako posljednji zabilježeni znak nije '$', vraćamo se na korak 2.

Opisani postupak vizualiziran je na slici 2.3. Bitno je primjetiti kako pri rekonstrukciji originala nismo koristili ništa osim transformiranog niza T^{BWT} i LF-mapiranja (koje se ostvaruje tablicama C i Occ).

[2] U ovom trenutku nemamo tablicu SCR, zanima nas samo indeks



Slika 2.3: Primjer rekonstrukcije originala iz transformiranog niza T^{BWT}

2.1.3. Pretraživanje

Transformirani niz T^{BWT} može se koristiti i za pretraživanje originalnog teksta T. Algoritam pretraživanja vrlo je sličan rekonstrukciji originala. Bazira se zapravo na poznatom načinu pretraživanja sufiksnih polja, kao što je već spomenuto, BWT transformacija i sufiksno polje nekog niza su skoro ekvivalentni.

Algoritam pretraživanja pojavljivanja niza Q unutar niza T na temelju BWT transforacije T^{BWT} bazira se na sljedećim konceptima:

- Pretraživanje se vrši po znakovima Q unatrag (od zadnjeg prema prvom)
- Prate se početni i krajnji indeks sufiksa (*SCR* tablice) unutar kojih su moguće podudarnosti
- U svakom koraju (procesiranom znaku iz P) se područje mogućih podudarnosti smanjuje

Ako početni indeks označimo *ps* (engl. *pointer start*), a krajnji indeks *pe* (engl. *pointer end*), tada se algoritam izvršava sljedećim koracima:

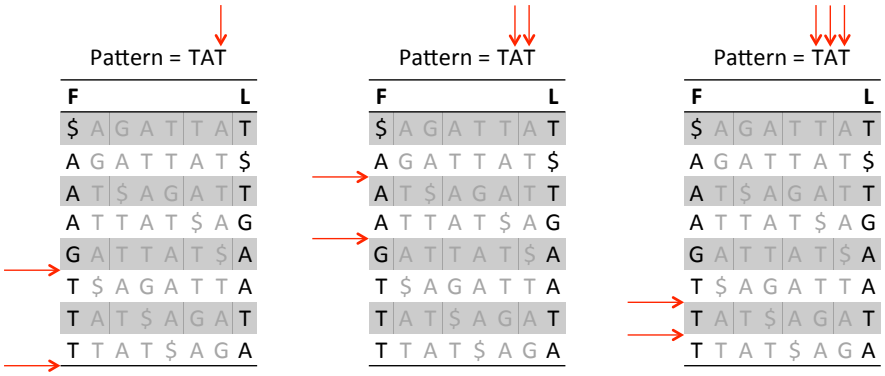
1. Inicijaliziraj indekse *ps* i *pe* tako da obuhvaćaju cijelu tablicu *SCR*
2. Odaberi prvi neobrađeni znak c iz niza Q, počevši od kraja
3. Unutar područja među indeksima nađi prvo i posljednje pojavljivanje znaka c [3]

[3] Učinkovita implementacija pretraživanja ne izvršava ovaj korak, navodimo ga samo radi opisa rada

4. Izračunaj L-indekse nađenih pojavljivanja koristeći LF-mapiranje

5. Ako postoje neobrađeni znakovi u Q, vrati se na korak 2.

Navedeni koraci algoritma formulirani su kako bi bili što jasniji. Opis prikladniji za izravnu računalnu implementaciju može se naći u radu [3]. Slika 2.4 prikazuje korake pretraživanja za $Q = TAT$.



Slika 2.4: Primjer pretraživanja teksta korištenjem transformiranog niza T^{BWT}

2.2. Analiza složenosti algoritma

Glavni razlog za razvoj FM-indeksa je povećanje efikasnosti pretraživanja dugih nizova (poput genotipa). U ovom poglavlju će se razmotriti složenost pretraživanja korištenjem FM-indeksa, sa teoretskog stanovišta. Pri tome nas zanima složenost korištenja izgrađenog indeksa za neki niz, a ne složenost stvaranja indeksa.

2.2.1. Vremenska složenost

Razmotrimo vrijeme pronalaska pojavljivanja niza Q unutar niza T, na način opisan u poglavlju 2.1.3. Iz algoritma je vidljivo da je potreban jedan korak za svaki znak u Q, dakle pretraživanje ima linearnu složenost s obzirom na duljinu Q. Svaki od tih koraka svodi se na dvije operacije LF-mapiranja, koje ima konstantnu vremensku složenost (ne ovisi o duljinama nizova T i Q).

Važna napomena je da prolazak kroz znakove niza Q može biti prekinut u slučaju da se utvrdi da nema pojavljivanja Q unutar T, što se može desiti u bilo kojem trenutku prolaska kroz Q.

algoritma.

Dakle, vremenska složenost pretraživanja je generalno linearna s obzirom na duljinu niza Q.

2.2.2. Memorijska složenost

Memorijska (prostorna složenost) je razmatranje utroška memorije na potporne strukture podataka FM-indeksa. Podatci koji se koriste su transformirani niz T^{BWT} (odnosno sufiksno polje) te tablice C i Occ korištene za LF-mapiranje.

Sufiksno polje ima jednak broj elemenata kao i originalni niz T. U tom smislu je memorijska složenost polja linearna s obzirom na duljinu T. Isto vrijedi i za transformirani niz T^{BWT}. Prostorni utrošak sufiksnog polja može se smanjiti na više načina, a niz T^{BWT} je često oblika pogodnog za komprimiranje, ali obje tehnike izlaze izvan okvira ovog rada.

Tablica prefiksnih suma C pohranjuje po jedan cijeli broj za svaki element abecede niza T. Iako je u teoriji taj broj ograničen samo duljinom niza T, u praksi je zapravo vrlo malen te tablica C ne predstavlja problem u kontekstu zauzeća memorije.

Tablica pojavljivanja Occ (u naivnoj implementaciji) pohranjuje po jedan cijeli broj za svaku poziciju niza T, za svaki element abecede T. Primjećujemo linearnu složenost s obzirom na duljinu niza T. S obzirom na potencijalno ogromne duljine nizova koje želimo moći pretraživati, ovo je problem. Postoji više pristupa "kodiranja" tablice Occ. Unutar ovog rada, zadatak je implementirati FM-indeks korištenjem stabla valića.

3. Stablo valića

4. Implementacija i testiranje

5. Zaključak

Zaključak.

6. Literatura

[1] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. pages 390–398, 2000.

[2] M. Burrows, D. J. Wheeler, M. Burrows, and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, 1994.

[3] J. Singer. A wawelet tree based fm-index for biological sequences in seqan. Master's thesis, 2012.

7. Sažetak

Sažetak.