SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**SEMINAR**

**Algoritam za traženje maksimalnih egzaktnih preklapanja u dugačkim sekvencama koristeći rijetka sufiksna polja**

*Josip Domšić, univ. bacc. ing. comp*

*Dino Kovač, univ. bacc. ing. comp*

*Marko Obrvan, univ. bacc. ing. comp*

Voditelj: *Mile Šikić, doc. dr. sc*

Zagreb, Listopad, 2014.

Table of Contents

[**1. Uvod 2**](#_Toc408609824)

[**2. Opis problema 3**](#_Toc408609825)

[**3. FASTA format 4**](#_Toc408609826)

[**4. Sufiksna polja 6**](#_Toc408609827)

[**4.1. Izgradnja sufiksnog polja 6**](#_Toc408609828)

[**4.2. Implementacija izgradnje sufiksnog polja IS-SA 7**](#_Toc408609829)

[**5. Rijetka sufiksna polja 10**](#_Toc408609830)

[**6. Biološka usporedba nizova 10**](#_Toc408609831)

[**7. Maksimalna egzaktna preklapanja MEMS 10**](#_Toc408609832)

[**9. Implementacija algoritma za maksimalna preklapanja 10**](#_Toc408609833)

[**10. Literatura 10**](#_Toc408609834)

[**11. Sažetak 11**](#_Toc408609835)

# Uvod

Bioinformatika je znanstvena grana koja se bavi razvojem programskih alata za proučavanje i razumijevanje bioloških podataka i procesa. Osnova joj je korištenje kračunala na području biologije u svrhu lakšeg rukovanja i obrade bioloških informacija. Računala se koriste da skupljaju, pohranjuju, analiziraju i integriraju biološke i genetke informacije koje potom mogu biti korištene za genetsko bazirano dobivanje lijekova i njihov razvoj.

Svoj nagli razvoj doživjela je nakon eksplozije javno dostupnih informacija o genetici koja su proizašla iz “Human Genom Project”. Krajnji cilj projekta je bio određivanje sekvence cijelog ljudskog genoma, koji sadrži otprilike tri bilijuna baznih parova. Unutar projekta nastalo je prvo mješanje kompjuterske znanosti s biologijom, a njegova važnost ležala je u otkrivanju genetskih informacija vezanih za razne bolesti, te molekula koje će biti novi cilj za stvaranje raznih ljekova.

Danas su mnoga sveučilišta, državne institucije i farmaceutske tvrtke diljem svijeta oformila svoje bionformatičke grupe. Cilj takvih grupa je otkrivanje i prevođenje nevjerojatno velikih količine informacija generiranih sljedovima unutar genoma, u svrhu boljeg razumjevanja gentike i njenom manipulacijom.

# Opis problema

Osnovni problem s kojim se susrećemo su dimenzije genoma nad kojima vršimo obradu. Oni mogu biti dimenzija 2-3Mb pa do nekoliko gigabajta čistih tekstualnih podataka. Iz samih dimenzija genoma vidljivo je da se naivnim pretraživanjem rezultati neće dobiti u razumnom vremenu već da treba priskočiti nekom algoritmu ili strukturi koji će ubrzati proces pretraživanja genoma.

Koncept sufiksnih polja je prvi put upotrebljen od strane Mamber i Myersa u SODA’90[1] i SICOMP’93[2]. Bila je riječ o prostorno efikasnom algoritmu, kao alternativi sufiksnim stablima. Od tada je bila dobro organizirana i osnovna podatkovna struktura korisna za širok spektar primjena, indeksiranja podataka, pohranjivanje, dohvat i obradu.

Za niz od *n* znakova, nad abecedom Σ={ $, *A*, *C*, *G*, *T*}, koje označavamo sa *S*, sufiksno polje, označeno sa SA(*S*) je je polje indeksa koji pokazuju sve sufikse od *S* poredane ulaznim ili silaznim abecednim redom. Sufiksno polje od S tako zahtjeva *n⌈log n⌉* bitova podatkovnog prostora. Iako izgleda prihvatljivo, genomi su sastavljeni od nekoliko bilijuna parova, što nas navodi da će vrijeme pretraživanja biti dugo, a prostor potreba za pohranjivanje sljeda ovakvim algoritmom velik. Dolazi se do potrebe usavršavanja postojećeg ili izrada novog algoritma za pohranu i indeksiranje genoma. Algoritmi koji se bave rješavanjem navedenog problema zovu se SACA (eng. Suffix Array Construction Algorithms). Danas se problem uspješno rješava u linearnom vremenu. Za sada postoje dva algoritma s linearnim vremenom izvođenja sa zadovoljavajućim mogućnostima prostornog i vremenskog izvršavanja, SA-IS i SA-DS. Oba algoritma koriste *podijeli pa vladaj* metodu da rekurzivno izračunaju suifiksno polje i linearnom vremenu. U praksi SA-IS je brži od SA-DS ali koristi puno više prostora u najgorem mogućem scenariju. Za rješavanje našeg problema koristili smo SA-IS algoritam.

# FASTA format

Za spremanje podataka koristi se FASTA format datoteke. U bioinformatici to je tekstualno bazirani format kojim se prikazuju nukleoidi ili amino kiseline u kojoj su nukleoidi ili amino kiseline prezentirane kao jedno slovo abecede. Osim mogućnosti spremanja nukleoida ili amino kiselina, u datoteci se također mogu nalaziti i komentari.

Sekvenca u FASTA-i započinje opisnikom, nakon čega sljede linije sekvenci. Opisnik sekvenci razlikuje se od same sekvence tako da započinje sa specijalnim znakom “>” i proteže se do kraja reda. Pravilo je da ne postoji razmak između oznake opisnika prvog simbola. Nakon oznake opisnika sljedi indetifikator sekvence, a po njegovom završetku ostatak teksta je opcionalan. Također postoji nepisano pravilo da ni jedna linija nebi smjela biti duža od 80 znakova. Time se osigurava lakša programska obrada jer programer ne mora paziti na mogućnost dohvata predugih linija i povećava se preglednost samog zapisa.

|  |
| --- |
| >gi|71025128|ref|NC\_007194.1| Aspergillus fumigatus Af293 chromosome 1, whole genome shotgun sequence  CCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTA  ACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTTTAGGCAACTGCAGCTT  CAAACCAGATTTGGATGGGCCACACGCGTGCTAGGTTTCCTGGTTCTTGGAACGACATTGTTCTCACCTA |

**Tablica 3.1** primjer FASTA zapisa

| **Kod nukleinske kiseline** | **Značenje** |
| --- | --- |
| A | A |
| C | C |
| G | G |
| T | T |
| U | U |
| R | A or G |
| Y | C, T or U |
| K | G, T or U |
| M | A or C |
| S | C or G |
| W | A, T or U |
| B | not A (i.e. C, G, T or U) |
| D | not C (i.e. A, G, T or U) |
| H | not G (i.e., A, C, T or U) |
| V | not T nor U (i.e. A, C or G) |
| N | A C G T U |
| X | maska |
| - | jaz neodređene duljine |

**Tablica 3.2** Reprezentacija sekvenci

# Sufiksna polja

Sufiksna polja *SA* (end. **S**uffix **A**rray) su alternativa sufiksnim stablima. Prednost sufiksnih polja na sufiksnim stablima je manji utrošak memorije. Sufiksna polja su polje ne-negativnih cijelih brojeva, koje predstavljaju početne pozicije abecedno porednih sukisa ulaznog niza. Ako imamo tekst *T* njegovo sufiksno polje je leksikograpski uređeno polje sastavljeno od svih sufiksa od *T[0...n]*. Preciznije, sufiksno polje je polje *SA[0...n]* gdje je polje cijelobrojnih vrijednosti *n* koje sadrže permutacije od *0..n* tako da vrijdi *TSA[0]< TSA[1]< TSA[2]<...< TSA[n]*.

Pri izgradnji *SA* najčešće se istovremeno gradi njegovo srodno inverzno sufiksno polje *ISA* (eng.**I**nverse **S**uffix **A**rray) koje je inverzna permutacija SA. Za *SA-1* vrijedi *SA­­-1[SA[i]] = i* . Drugim riječima *SA-1* je leksikografska vrijednost sufiksa *Ti*. Uobičajeno je da se teks *T* dodatno proširi terminirajućim, kranjim simbolom, *„$“*. Sam simbol nema nikakav utjecaj na niz i njegova vrijednost u polju se smatra najmanjom ili čak negativnom, svrha njegova dodavanja je oznaka kraja teksa.

## Izgradnja sufiksnog polja

Sufiksno polje i njegov inverz grade se u tri prolaza. U prom prolazu potrebno je od teksta *T* napraviti sufikse. Broj sufiksa jednak je broju znakova u tekstu i iznosi *n*. Nakon što su se izlučili svi sufiksi oni se sortiraju silazno abecedno počevši od *A*. Specijalni simbol *$* pri tome ima najveću vrijednost i njegova indeks je *0* i prvi je na listi. Nakon sortiranja se računa *SA-1* koji nije ništa drugo nego indeks sufiksa u abecedno sortiranom nizu. U trećem prolazu se računa najduži zajednički prefiks *LCP* (eng. **L**ongest **C**ommon **P**refix). *LCP* nam u obliku cjelobrojne vrijednosti opisuje koliko se prefiks pododara s njegovim prethodnikom u sortiranom polju. Izgradnja sufiksnog polja prikazana je na sljedećim tablicama redom 4.1., 4.2. i 4.3., na primjeru niza teksta *T=“banana“*.

| **Index** | **SA[i]** | **Sufiks teksa** |
| --- | --- | --- |
| 0 | 0 | banana$ |
| 1 | 1 | anana$ |
| 2 | 2 | nana$ |
| 3 | 3 | ana$ |
| 4 | 4 | na$ |
| 5 | 5 | a$ |
| 6 | 6 | $ |

**Tablica 4.1.** Izrada sufiksa

| **SA[i]** | **SA-1[i]** | **Sufiks teksa** |
| --- | --- | --- |
| 6 | 6 | $ |
| 5 | 0 | a$ |
| 3 | 1 | ana$ |
| 1 | 2 | anana$ |
| 0 | 3 | banana$ |
| 4 | 4 | na$ |
| 2 | 5 | nana$ |

**Tablica 4,2.** sortiranje sufiksa abecedno

| **SA[i]** | **SA-1[i]** | **Sufiks teksa** | **LCP** |
| --- | --- | --- | --- |
| 6 | 6 | $ | -1 |
| 5 | 1 | a$ | 0 |
| 3 | 2 | ana$ | 1 |
| 1 | 3 | anana$ | 3 |
| 0 | 4 | banana$ | 0 |
| 4 | 5 | na$ | 0 |
| 2 | 6 | nana$ | 2 |

**Tablica 4.3.** Potpuna tablica s izračunatim LCP

## Rijetka sufiksna polja

Za razliku od sufiksnih polja, kod rijetkih sufiksnih polja spremamo samo svaki *K-*ti sufiks. Iako algoritmi i ideja postoji već desetljeće, tek je nedavno napravljen algoritam koji radi unutar prihvatljivog vremena. Najveći problem stvaranja sufiksnog polja je sortiranje sufiksa. Dugački nizovi mnogu imati tisuće i sortiranje sufiksa nastalih iz takvog niza vremenski je izazov. Nakon sortiranja potrebno je odrediti koji se sufiksi odbacuju, na način da se svaki sufiks može rekonstruirati iz zadržanih sufiksa, a da se pri tome maksimalno smanji broj zadržanih sufiksa. Rekonstrukcja svi sufiksa iz preostalih vremenski je izazov korištenja rijetkih *SA* pri traženju maksimalnih istih preklapanja *MEMS* (eng. **M**aksimal **E**xact **M**atche**s)** Opće prihvaćeni algoritam za izgradnju sufiksnog stabla odvija se u četiri koraka:

* Koristeći „*IS-SA“* prvo se sortira sufiks. Iako se mogu koristiti i drugi načini sortiranja, navedini se eksperimentalno pokazao kao jedan od najboljih.
* Konstruirati novi referentni tekst dužine *n/K*, gdje je *K* broj po kojem se sortirao sufiks, a *n* ukupan broj sufiksa.
* Konstruiraj nove indekse intSA
* Pomnoži sve indekse sa *K*

## Implementacija izgradnje sufiksnog polja IS-SA

Na malenim nizovima postupak izgradnje sufiksnog polja je triviljala, mešutim nizovi koji se ptratražju mogu biti veliki preveliki da bi koristili naivna sortiranja. Potrebno je implementirati algoritme za sortiranja s malom složenošću O(n), koji će vratiti rezultate u malom vremenu. Algoritam koji pri tome koristi zove se SA-IS algoritam opisan u Ge Nong [3.]. Princip rada temelji se na rekurzivnom dijeljenju niza te njegovo sklapanje u povratku iz rekurzije. Z objašnjenje rada implementiranog algoritma potrebno je prvo objasniti sljedeće notacije:

* *S* je nis od *n* znakova i završava specijalnim znakom *$*
* *Σ(S)* je abeceda nad *S*
* Podnis od *S* koji se kreće od *i* do *j* označavamo s *S[i..j]*
* *suf(S,i)* je sufiks od S*,* koji započinje na poziciji *i* i završava na *$*
* Sufiks  *suf(S,i)* je L-tipa ako vrijedi *suf(S,i)< suf(S,i+1)* u suprotnome je S tipa, sufiks koji je sastavljen samo od *$* je S-tipa
* Dali je sufiks S ili L tipa bilježi se u bool polju *t[i]* veličine *n*.
* Kod traženja S ili L tipa bitno je primjetiti sljedeće:
  + *S[i]* je S-tip ako :
    - *S[i]* *< S[i + 1]*
    - *S[i]* = *S[i + 1]* i *suf(S,i + 1)* je S-tip;
  + *S*[*i*] je L-tip ako:
    - *S[i]* *> S[i + 1]*
    - *S[i]* = *S[i + 1]* i *suf(S,i + 1)* je L-tip
* LMS (eng. **L**eft **M**ost **S**uffix) je znak *S[i]* ako je S-tipa, a *S[i-1]* L-tipa

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | S | S | L | **S** | S | L | L | L | L |

* LMS-podniz je *S[i..j]* tako da su *S[i]* i *S[j]* LSM i između njih ne postoji ni jedan drugi LMS

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | L | **S** | S | S | S | L | **S** | L | L |

Iz navedenog je vidljivo da se pri samo jednom prolazu može odrediti tip svih sufiksa ako ga prolazimo s desna na lijevo. Sam algoritam kako radi SA-IS algoritam prikazan je na u narednom odsječku.

|  |
| --- |
| SA-IS ( *S, S A* )  *S* je ulazni niz znakova  *SA* izlaz oblika sufiksnog polja od *S*  *t*: polje [0 *..n −* 1] boolean elemenata;  *P*1, *S*1: polje [0 *..n* 1 *−* 1] integera  *B*: polje [0 *..* ||Σ( *S*)∥ *−* 1] integera;  Napravi prolaz po *S* i označi sve znakove kao S ili L tip I pribilježi to u *t*  Iteriraj kroz *t* dok ne pronađeš sve LMS-podnizove u *S* into *P*1;  Induced sort all the LMS-substrings using *P*1 and *B*;  Name each LMS-substring in *S* by its bucket  index to get a new shortened string *S*1;  **if** Each character in *S*1 is unique  **then**  Directly compute *SA* 1 from *S*1;  **else**  SA-IS(*S*1*, SA* 1); *✄* Fire a recursive call  Induce *SA* from *SA* 1;  **return** |

# Maksimalna egzaktna preklapanja MEMS

Maksimalna egzaktna preklapanja su podudaranja između dva niza znakova koja se ne mogu proširiti ni u smjeru početka ni u smjeru kraja bez da postoji pogreška u podudaranju. Nizovi nad kojima se traže preklapanja u kontekstu bioinformatike su vrlo veliki. Klasični pristup traženju maksimalnih egzaktnih preklapanja uključuje učitavanje oba niza u radnu memoriju računala te izgradnja sufiksnog stabla. Nedostatak takvog pristupa je velika količina radne memorije potrebna za rad takvog algoritma, kao i vremenska neefikasnost. U novije vrijeme traže se novi pristupi koji troše manje radne memorije, pa tako jedan od pristupa uključuje korištenje sufiksnih polja kojima se značajno smanjuje potrošnja memorije. Kako bi se još smanjila potrošnja memorije i omogućilo nalaženje maksimalnih egzaktnih preklapanja za veće nizove, a istovremeno ostalo u okvirima prihvatljivog vremena izvršavanja, u implementaciji algoritma koju opisuje ovaj dokument koriste se ‘rijetka’ sufiksna polja.

## Implementacija algoritma za maksimalna preklapanja

# Literatura

[1]  U. Manber and G. Myers, “Suffix arrays: A new method for online string searches,” in Proceedings of SODA, 1990, pp. 319–327.

[2]  ——, “Suffix arrays: A new method for on-line string searches,”SIAM Journal on Computing, vol. 22, no. 5, pp. 935–948, 1993.

[3] Ge Nong, Sen Zhang, and Wai Hong Chan, “Two efficient algorithms for linear time suffix array construction” IEE Transactions, (Volume:60, Issue: 10)

# Sažetak

Unazad nekoliko godina poćelo se mnogo pažnje posvečivati na izučavanje prirode, a posebno veliki napradek napravljen je na razini genetike. Svakim danom istražuju se sve kompleksnije sakvence, mapiraju se i pretražuju. Da bi ta istraživanja mogla nastaviti svoj pozitivan trend, prijeko je potrebno napraviti algoritme koji će izvršavati potrebne akcije nad genomima i razumnom vremenu s korištenjem jako malo sredstava računala. Iako danas već postoje računala koja mogu obrađivati gigabajtske podatke samo u djelićima sekunde, ona nisu ni jeftina ni dostupna svima.

Algoritmi implementirani u sklopu projekta jedni su od najnovijih i najefikasnijih algoritama koji se bave navedenom problematikom. Prilikom pisanja algoritama posebna se pažnja pridodavala uskim grlima unutar same implementacije, kao i jeziku u kojem je pisana, sve u svrhu maksimalnog iskorištenja memorije i vremenske optimizacije. Krajnji rezultat je primjenjivi program koji koristeći SA-IS algoritam pronalazi MEMS unutar programskog jezika C++, koji je odabran zbog svoje prenosivosti i brzine izvođenja.