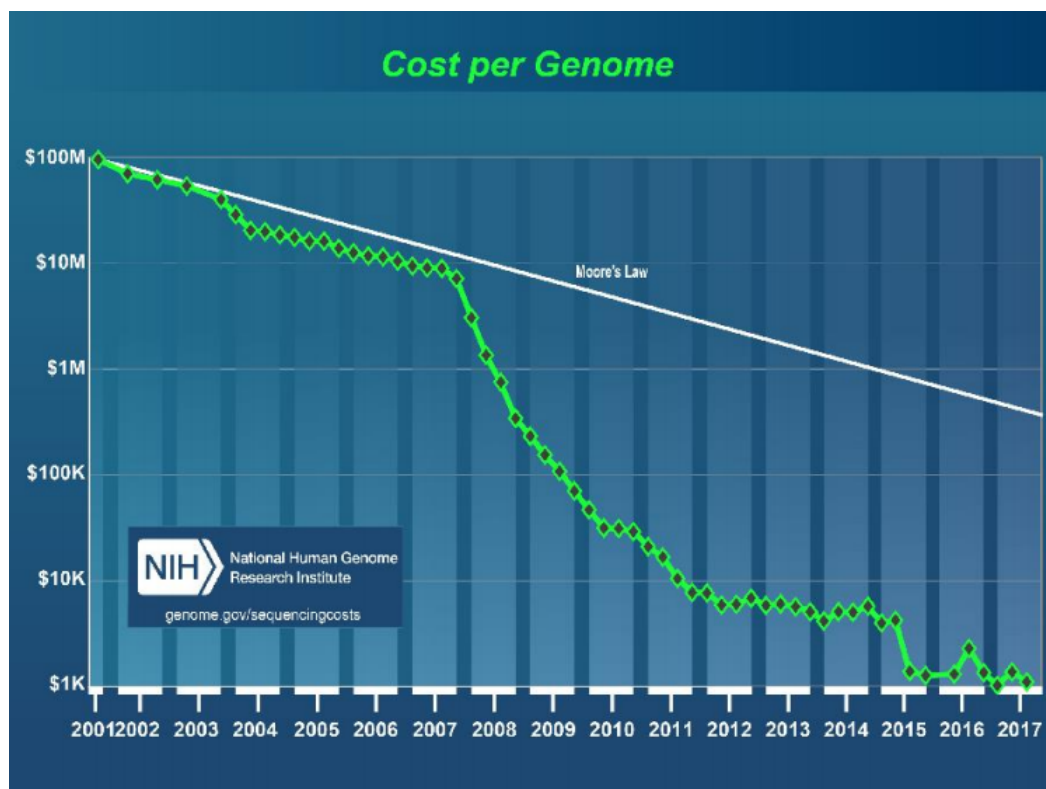


Glava 1

Kako locirati mutacije koje izazivaju bolesti?

1.1 Mapiranje očitavanja

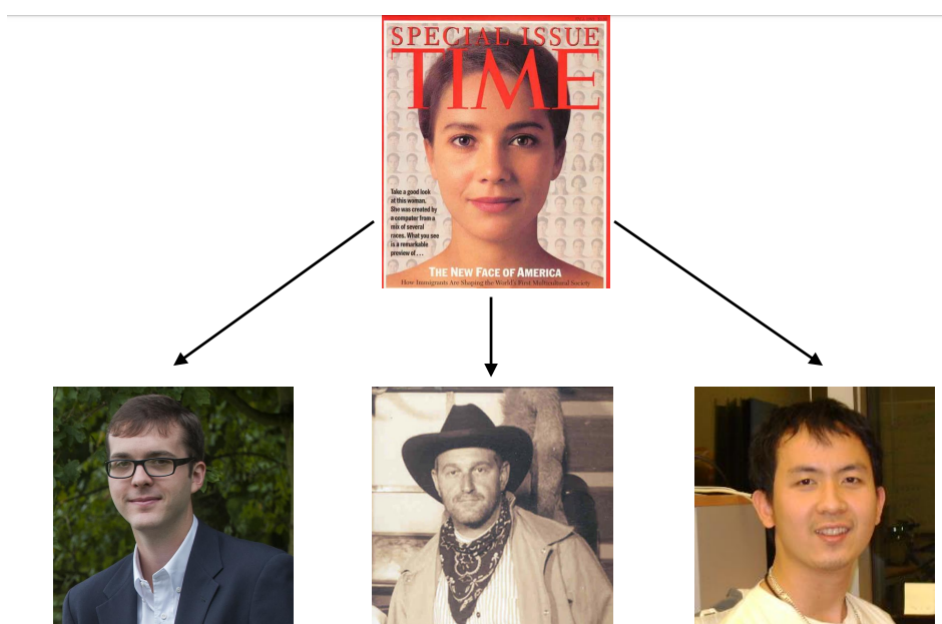
Cena sekvencionisanja genoma je od 2001. u konstantnom padu, i teži se tome da ono postane potpuno pristupačno običnom čoveku, kao i da bude sastavni deo lekarske usluge.



Slika 1.1: Kretanje cene sekvencioniranja u poslednjih 17 godina

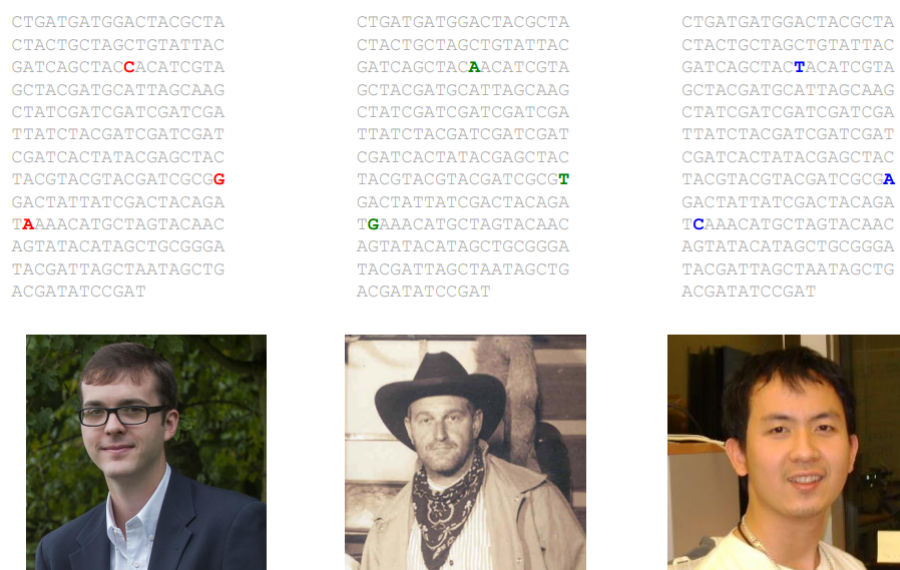
Oko 1% dece rodi se sa mentalnom retardacijom, ali uzroci ove pojave i dan-danas nisu razjašnjeni, jer do nje može dovesti niz različitih genetskih poremećaja. Jedan od njih je i Ohdo

sindrom, koji izaziva bezličan, "maskoliki" izraz lica. Biolozi su 2011. godine uspeli da pronađu niz zajedničkih mutacija kod pacijenata, koje su kasnije iskorišćene za identifikovanje jedinstvene mutacije proteina, odgovorne za nastanak ovog sindroma. Razumevanje suštinskog uzroka Ohdo sindroma samo je jedno od otkrića do kojeg se došlo proučavanjem genetskih poremećaja upotrebom **mapiranja očitavanja**. Kod ove metode, porede se sekvencionisana očitavanja DNK uzeta od pojedinaca sa **referentnim ljudskim genomom**. **Referentni ljudski genomom** zamišljen je da bude "prosečan" ljudski genom izračunat na određenom broju uzoraka. Trenutni referentni genom baziran je na genomima 13 dobrovoljaca iz SAD-a; i dalje se vrši njegovo usavršavanje ispravljanjem grešaka i popunjavanjem rupa (trenutno ih je preko stotinu). U proseku, razlika između individualnog i referentnog genoma je u oko 3 miliona mutacija.

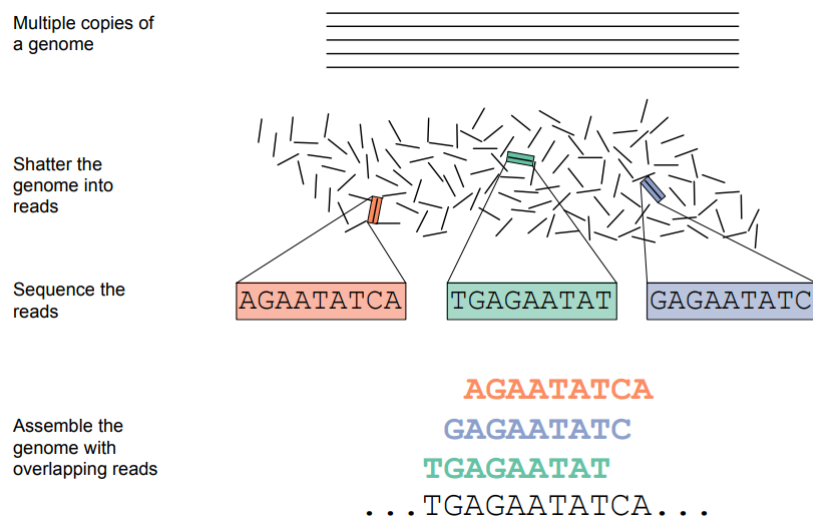


Slika 1.2: Prikaz grananja genoma vrste na više personalnih genoma

Pitanje je kako možemo efikasno sastaviti individualne genome koristeći referentne. Možemo koristiti **asembliranje**, ali konstrukcija de Brojnovog grafa zahteva mnogo memorije. Možemo koristiti postojeću strukturu referentnog genoma kao pomoć u sekvencioniranju genoma pacijenta.



Slika 1.3: Razlike u personalnim genomima



Slika 1.4: Primer asembliranja

Mapiranje očitavanja predstavlja određivanje pozicije u referentnom genomu sa kojima svako očitavanje ima visoku sličnost.

CTGAGGATGGACTACGCTACTACTGATAGCTGTTT
 GAGGA CCACG TGA-A

Slika 1.5: Primer mapiranja očitavanja; gornja niska predstavlja referentni genom, a donje niske očitavanja individualnog genoma

1.1.1 Egzaktno uparivanje šablona

Potrebno je pronaći gde se očitavanja egzaktno poklapaju sa referentnim genomom. Postoji jednostruko i višestruko uparivanje šablona.

Problem jednostrukog uparivanja šablona:

Ulaz: Niske *Pattern* i *Genome*.

Izlaz: Sve pozicije u niski *Genome* gde se niska *Pattern* pojavljuje kao podniska.

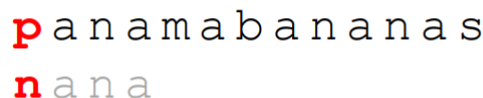
Problem višestrukog uparivanja šablona:

Ulaz: Kolekcija niski *Patterns* i *Genome*.

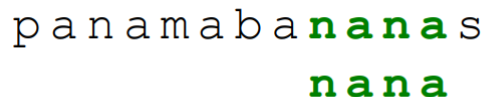
Izlaz: Sve pozicije u niski *Genome* gde se niske iz kolekcije *Patterns* pojavljuju kao podniske.

1.1.2 Moguća rešenja

Rešenje koje nam prvo pada na pamet je rešavanje problema grubom silom. Algoritam se sastoji u tome da se linearno krećemo kroz genom i proveravamo da li se dati šablon poklapa sa podniskom genoma iste dužine, koja počinje na toj poziciji.



Slika 1.6: Uparivanje šablona grubom silom - nepoklapanje



Slika 1.7: Uparivanje šablona grubom silom - poklapanje

Vreme izvršavanja algoritma u slučaju jednostrukog *Pattern*-a je $O(|Genome| * |Pattern|)$, dok je u slučaju višestrukog *Patterns*-a je $O(|Genome| * |Patterns|)$, gde je $|Patterns|$ suma dužina elemenata liste *Patterns*.

Međutim, problem je u tome što genomi mogu biti veoma dugi. U slučaju ljudskog genoma (3 GB), ukupna dužina svih očitavanja može biti veća od 1 TB; kao rezultat toga, algoritam složenosti $O(|Genome| * |Patterns|)$ je previše spor.

1.1.3 Sufiksna stabla

Razlog velike neefikasnosti prethodnog algoritma jeste u tome što paterni prolaze kroz genom nezavisno jedan od drugog. Ako niski *Genome* zamislimo kao put, onda bi izvršavanje algoritma grube sile bilo analogno vožnji svakog paternu po putu u zasebnom automobilu. Ono što želimo jeste da sve paterne smestimo u jedan "autobus", čime bi nam bio dovoljan samo jedan prolazak kroz niski *Genome*.

Zbog toga paterne organizujemo u strukturu podataka nalik na usmereni aciklički graf, koju nazivamo **Trie** i koja ima sledeće osobine :

- Trie ima jedinstven čvor sa ulaznim stepenom nula, koji nazivamo koren
- Svaka grana Trie je obeležena jednim slovom
- Grane koje izlaze iz jednog čvora obeležene su različitim slovima
- Svaki sufiks neke niske dobija se nadovezivanjem slova duž neke putanje grafa, idući od korena naniže
- Svaka putanja stabla od korena do lista, ili do čvora sa izlaznim stepenom 0, predstavlja jedan element iz liste *Patterns*

Najjednostavniji način konstrukcije strukture Trie jeste iterativno dodavanje niski iz niza *Patterns* u rastuću strukturu.

Za date niske *Text* i *Trie(Patterns)* možemo brzo proveriti da li se neki element niza *Patterns* predstavlja prefiks niske *Text*. Dovoljno je da krenemo da spelujemo *Text* i da slovo po slovo prolazimo kroz Trie od korena naniže. Za svako slovo iz *Text*-a gledamo da li iz trenutnog čvora postoji grana obeležena tim slovom; ukoliko postoji, nastavljamo sa pretragom; u suprotnom obustavljamo pretragu i zaključujemo da nijedan element niza *Patterns* nije prefiks niske *Text*. Ukoliko stignemo do lista, onda je pretraga bila uspešna.

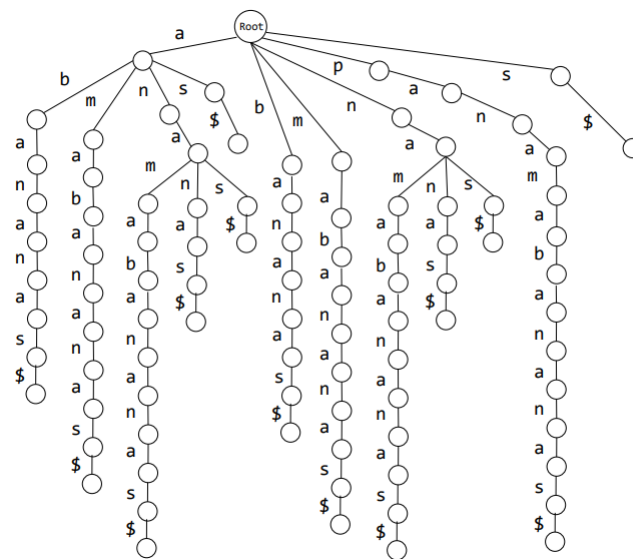
Da bismo pronašli da li se neki patern nalazi u genomu, potrebno je da u $\|Text\|$ iteracija pokrećemo prethodno opisani algoritam, pri čemu u svakoj iteraciji iz niske *Text* izbacujemo početni simbol, sve dok ona ne postane prazna.

Iako je prethodno opisani postupak vremenski efikasan, njegova mana leži u velikom zauzeću memorije. Naime, veličina strukture Trie proporcionalna je ukupnom broju simbola niza *Patterns*, a pošto veličina kolekcije očitavanja ljudskog genoma može dostići 1 TB, memorija potrebna za čuvanje ove strukture je prevelika.

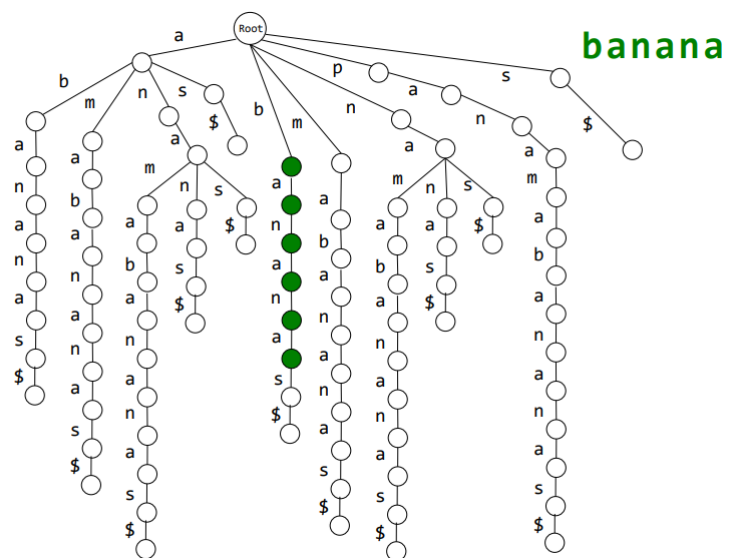
Bolji pristup je da se struktura Trie pravi na osnovu niske *Text*, tj. genoma. Za ovo će nam biti struktura poznata kao **sufiksni trie**. **Sufiksni trie** date niske predstavlja trie formiran na osnovu svih sufiksa te niske. Pre konstrukcije sufiksnog stabla, na kraj niske dodajemo simbol \$, kako bismo kasnije znali kad smo stigli do kraja.

Proveru da li se dati patern nalazi u tekstu vršimo tako što tražimo put u sufiksnom stablu, spelujući slova paternu do kraja. Ukoliko dođe do nepoklapanja, pretraga je neuspešna. U suprotnom, pretraga je uspešna.

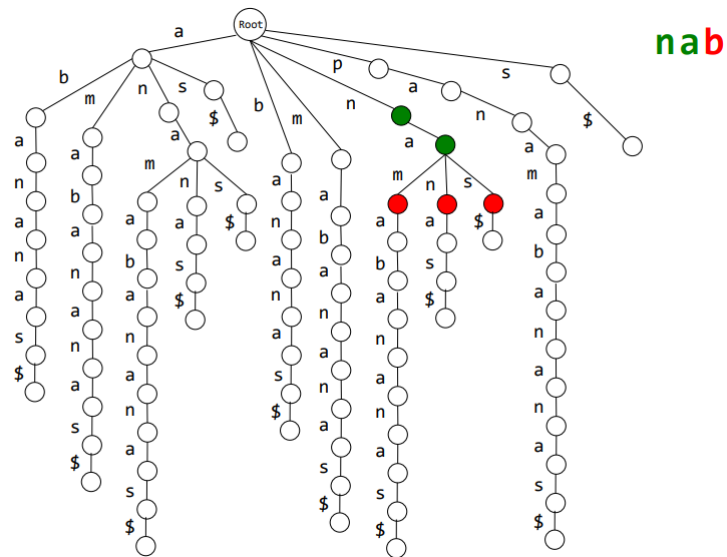
Ovim postupkom možemo utvrditi da li se pattern pojavljuje u genomu, ali ne i na kojoj poziciji. Za to moramo dodati još informacija u stablo. Na svakom listu dodamo početnu poziciju u niski *Genome* sufiksa koji se završava u tom listu.



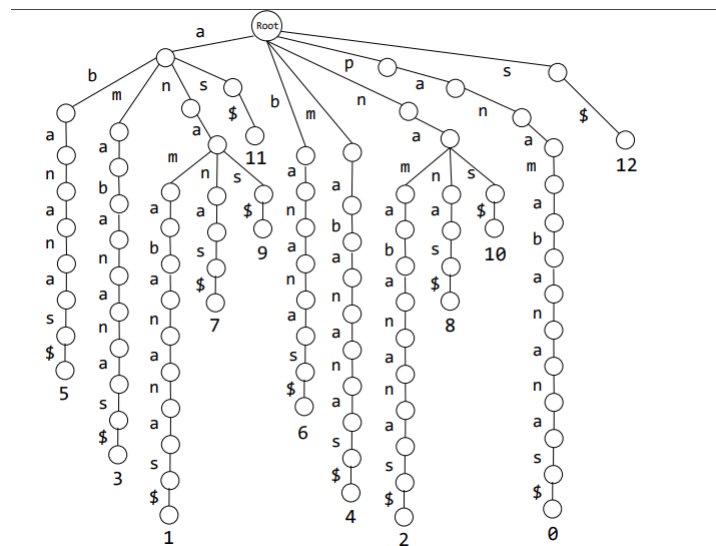
Slika 1.8: Nekompresovano sufiksno stablo



Slika 1.9: Nekompresovano sufiksno stablo - uspešno pronalaženje niske

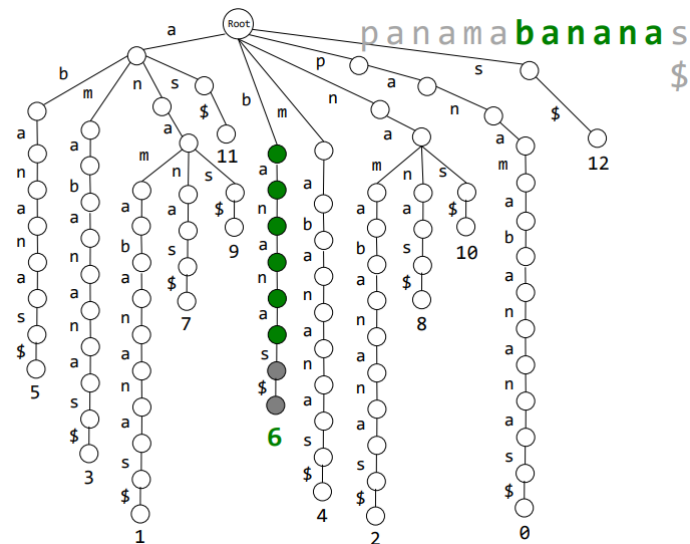


Slika 1.10: Nekompresovano sufiksno stablo - neuspešno pronalaženje niske

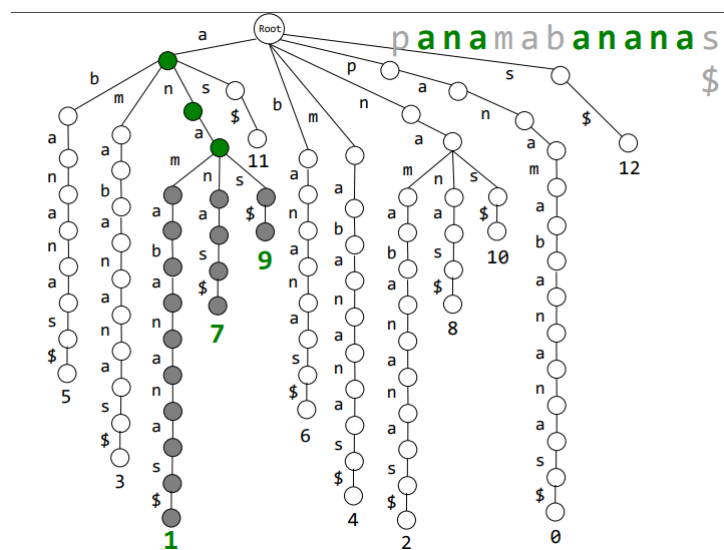


Slika 1.11: Sufiksno stablo sa numerisanim prefiksima

Sad se postavlja pitanje, kada pronađemo uparivanje, kako da znamo na kojoj poziciji se ono nalazi. To je sada lako, kada pronađemo uparivanje, nastavimo sa kretanjem naniže do lista, gde se nalazi pozicija odakle počinje pojavljivanje podniske.



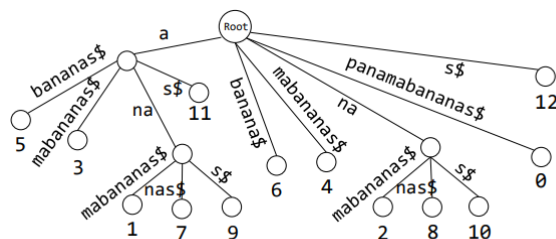
Slika 1.12: Primer nalaženja pozicije nakon uparivanja



Slika 1.13: Primer nalaženja pozicije nakon uparivanja - više poklapanja

Da bismo smanjili prostornu složenost, možemo kompresovati svaku putanju koja se ne grana u jednu granu. Ovakva struktura podataka naziva se **sufiksno stablo**.

Za svaku nisku *Genome* važi da je ukupan broj čvorova manji od dvostruke dužine niske *Genome*, tj. $\#nodes < 2|Genome|$. Ovo važi na osnovu činjenice da je broj listova jednak dužini genoma, tj. $\#leaves = |Genome|$, odnosno da je broj unutrašnjih čvorova manji od dužine genoma umanjene za jedan, tj. $\#internalnodes < |Genome| - 1$.



Slika 1.14: Kompresovano stablo

Prostorna i vremenska složenost

Vremenska složenost:

- $O(|Genome|^2)$ za konstrukciju sufiksnog stabla tako što se prvo konstruiše nekompresovano sufiksno stablo.
- $O(|Patterns|)$ za nalaženje uparivanja.

Prostorna složenost:

- $O(|Genome|^2)$ za konstrukciju sufiksnog stabla tako što se prvo konstruiše nekompresovano sufiksno stablo.
- $O(|Patterns|)$ za čuvanje sufiksnog stabla.

Postoje algoritmi sa linearnom prostornom i vremenskom složenošću. Vremenska složenost:

- $O(|Genome|)$ za konstrukciju sufiksnog stabla direktno.
- $O(|Patterns|)$ za nalaženje uparivanja.
- Ukupno $O(|Genome| + |Patterns|)$

Prostorna složenost:

- $O(|Genome|^2)$ za konstrukciju sufiksnog stabla direktno.
- $O(|Patterns|)$ za čuvanje sufiksnog stabla.
- Ukupno $O(|Genome|)$

1.2 Kompresija niski i Barouz-Vilerova transformacija

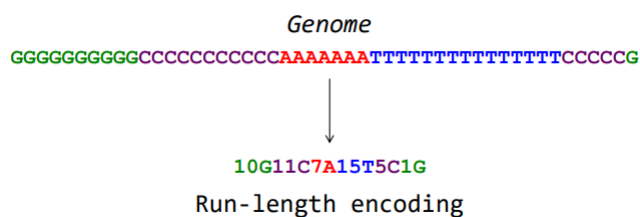
Najveći problem koji se javlja sa prethodnom rešenjem je to što O -notacija ignoriše konstante, a najpoznatija implementacija sufiksni stabala zahteva $20 * |Genome|$ (npr. veličina humanog genoma je $3GB \approx 60 GB$; i dalje unapređenje u odnosu na $1TB$). Postavlja se pitanje da li možemo smanjiti faktor konstante. Odgovor nam daje kompresija genoma.

1.2.1 Kompresija genoma

Glavna ideja ovog rešenja jeste da se smanji količina memorije potrebna za čuvanje niske Genome. Za ovo su nam potrebne metode za kompresiju niske velikih dužina, što je naizgled sasvim drugačiji problem.

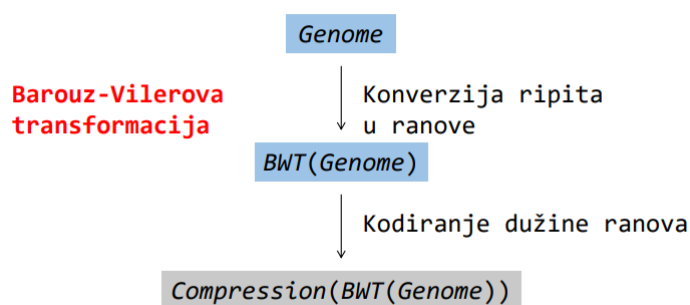
U ovom genomu imamo nekoliko uzastopnih ponavljanja jedne aminokiseline (ranovi, runs): prvo uzastopna ponavljanja aminokiseline G, pa C i tako dalje), a u nekim imamo uzastopna ponavljanja nizova aminokiselina (ripitsi, repeats): prvo uzastopna ponavljanja GAC, pa CATT i tako dalje.

Prva ideja pri rešavanju ovog problema jeste da kodiramo dužine ranova.



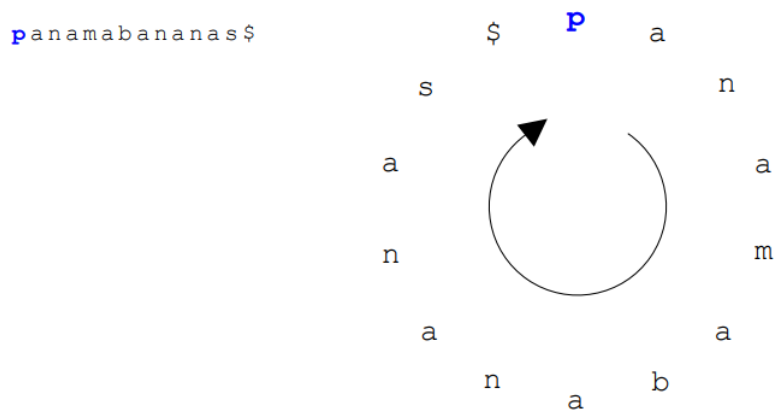
Slika 1.15

Problem kod ovog pristupa jeste to što u genomu nema mnogo ranova. Međutim, ima mnogo ripita. Postavlja se pitanje kako izvesti transformaciju ripita u ranove.



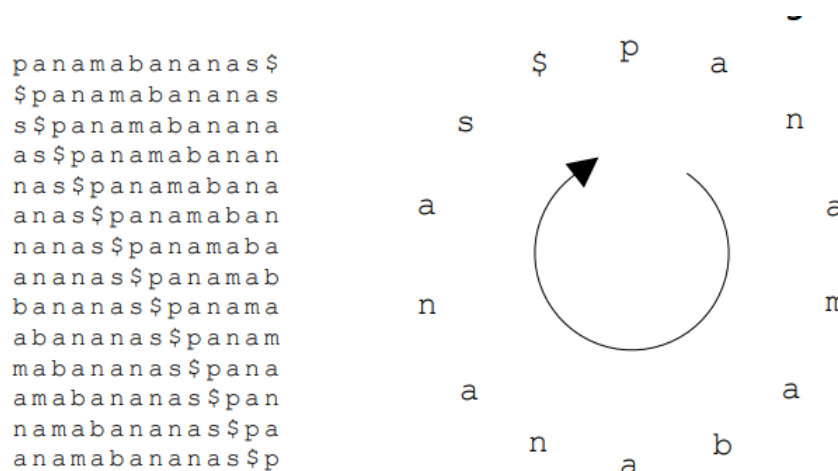
Slika 1.16

Odgovor na ovo pitanje daje nam Varouze-Vilerova transformacija (The Burrows-Wheeler Transform, skraćeno BWT).



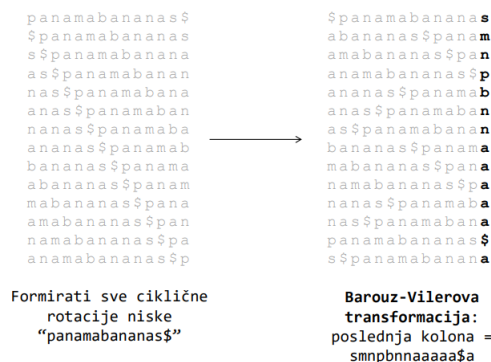
Slika 1.17

Ideja kod ovog algoritma je da se na početku formiraju sve ciklične rotacije date niske.



Slika 1.18: Scenario sa 4 promene

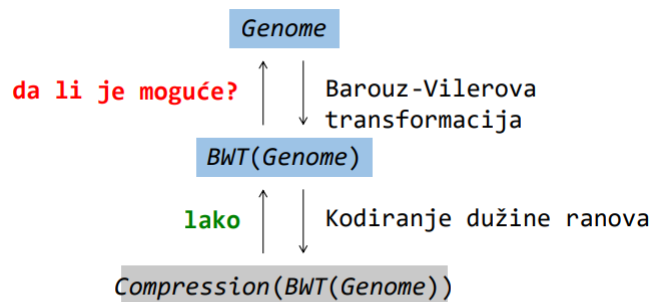
Zatim se vrši sortiranje svih dobijenih niski leksikografski (\$ je na početku).



Slika 1.19: Pohlepno sortiranje

Zatim posmatramo poslednju kolonu. Možemo primetiti da poslednja kolona sadrži veliki broj ranova. Međutim, isti slučaj je i sa prvom kolonom. Prvo ćemo se pozabaviti dekompresijom dobijene niska, pa ćemo se posle vratiti na ovo pitanje.

1.3 Inverzna BWT



Slika 1.20

Pogledajmo primer BWT-a za nisku *banana*.

```
$bananaa
a$banan
ana$bana
anana$b
banana$
na$bana
nana$ba
```

Slika 1.21

Ako sortiramo karaktere poslednje kolone “annb\$aa”, dobićemo prvu kolonu matrice. Na osnovu toga znamo 2-gramski sastav cirkularne niske *banana\$*.

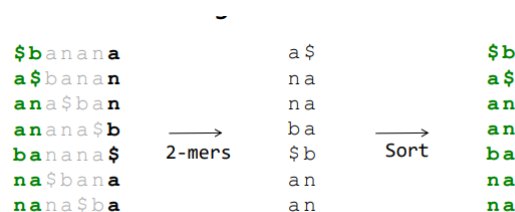
Srtiranjem niski dobijamo prve dve kolone matrice.

| | | |
|-----------|--------|-----|
| \$banana | | a\$ |
| a\$banan | | na |
| ana\$bana | | na |
| anana\$b | → | ba |
| banana\$ | 2-mers | \$b |
| na\$bana | | an |
| nana\$ba | | an |

Slika 1.22

Sada imamo dve kolone cikličnih niski. Zatim ponavljamo postupak - dodamo poslednju koju znamo, itd. Na kraju dobijamo rekonstruisanu celu matricu

Nisku *banana\$* dobijamo tako što uzmemo sve elemente iz prvog reda posle \$.



Slika 1.23

Prostorna složenost:

Rekonstrukcija niske Genome na osnovu BWT(Genome) zahteva čuvanje —Genome— kopija niske Genome, što iznosi $O(|Genome|^2)$. Poboljšanje složenosti je moguće ako primetimo nešto.

First-Last svojstvo: k-to pojavljivanje simbola u FirstColumn i k-to pojavljivanje simbola u LastColumn odgovaraju istoj poziciji simbola u niski Genome.

\$₁panamabananas₁
a₁bananas\$panam₁
a₂mabananas\$pan₁
a₃namabananas\$p₁
a₄nanas\$panamab₁
a₅nas\$panamaban₂
a₆s\$panamaban₃
b₁ananas\$panama₁
m₁abananas\$pana₂
n₁amabananas\$pa₃
n₂anas\$panamaba₄
n₃as\$panamabana₅
p₁anamabananas\$₁
s₁\$panamabanana₆

Slika 1.24: First-Last svojstvo

1.3.1 Efikasnija BWT dekompresija

Krenemo od simbola \$ (prvi u nizu cikličnih niski) u FirstColumn, zatim pogledamo koji simbol je u LastColumn u tom redu, nađemo ga u FirstColumn, onda za taj red nađemo koji simbol je u tom redu u LastColumn, itd. U jednom trenutku ćemo doći do \$ u LastColumn i tada smo okrenuli ceo krug i rekonstruisali celu Genome nisku. Prostorna složenost je $2 \cdot |Genome| = O(|Genome|)$.

1.3.2 Korišćenje BWT za uparivanje šablona

Da se podsetimo, uparivanje šablona korišćenjem sufiksni stabala zahtevalo je vremensku složenost od $O(|Genome| + |Patterns|)$, prostorna $O(|Genome|)$. Problem je bio što je sufiksno stablo tražilo $20 \cdot |Genome|$ prostora.

Poboljšanje možemo dobiti ako umesto sufiksnog stabla koristimo BWT(Genome) kao strukturu podataka.

Postupak se sastoji od toga da krenemo od kraja niske koju tražimo i u FirstColumn nađemo taj karakter. Zatim u odgovarajućem redu u LastColumn tražimo drugi od pozadi karakter od tih kojima je u FirstColumn poslednji iz uzorka.

Zatim nađemo u FirstColumn gde su ti iz LastColumn i gledamo naredni karakter. Ako se poklapa sa trećim od pozadi, nastavljamo dalje, ako ne, nema ga. I tako dok ne pređemo ceo uzorak od kraja ka početku.

```
$1panamabananas1
a1bananas$panam1
a2mabananas$pan1
a3namabananas$pa1
a4nanas$panamab1
a5nas$panamaban2
a6s$panamabanan3
b1ananas$panama1
m1abananas$pana2
n1amabananas$pa3
n2anas$panamaba4
n3as$panamabana5
p1anamabananas$1
s1$panamabanan6
```

Slika 1.25: First-Last svojstvo

1.3.3 Pronalaženje uparenih šablona

```
$1panamabananas1
a1bananas$panam1
a2mabananas$pan1
a3namabananas$pa1
a4nanas$panamab1
a5nas$panamaban2
a6s$panamabanan3
b1ananas$panama1
m1abananas$pana2
n1amabananas$pa3
n2anas$panamaba4
n3as$panamabana5
p1anamabananas$1
s1$panamabanan6
```

Slika 1.26: First-Last svojstvo

1.3.4 Pronalaženje uparenih šablona

Problem višestrukog uparivanja šablona:

Ulaz: Kolekcija niski Patterns i niska Genome.

Izlaz: Sve pozicije u niski Genome gde se niske iz kolekcije Patterns pojavljuju kao podniske.

Treba da nađemo pozicije. BWT ne daje ovaj podatak. Na primer, na gornjem primeru Ana se pojavljuje 3 puta, ali na kojim pozicijama?

Na kojim pozicijama se nalazi odredićemo pomoću sufiksnog niza.

Sufiksni niz je niz koji čuva početnu poziciju za svaki sufiks (niz karaktera u svakom redu matrice do simbola \$).

| | |
|----|--|
| 13 | \$ ₁ panamabananas ₁ |
| 5 | a ₁ bananas\$panam ₁ |
| 3 | a ₂ mabananas\$pan ₁ |
| 1 | a ₃ namabananas\$p ₁ |
| 7 | a ₄ nanas\$panamab ₁ |
| 9 | a ₅ nas\$panamaban ₂ |
| 11 | a ₆ s\$panamabanan ₃ |
| 6 | b ₁ ananas\$panama ₁ |
| 4 | m ₁ abananas\$pana ₂ |
| 2 | n ₁ amabananas\$pa ₃ |
| 8 | n ₂ anas\$panamaba ₄ |
| 10 | n ₃ as\$panamabana ₅ |
| 0 | p ₁ anamabananas\$ ₁ |
| 12 | s ₁ \$panamabanan ₆ |

Slika 1.27: Sufiksni niz

Sa slike vidimo da se **ana** iz prethodnog primera pojavljuje na pozicijama 1, 7 i 9.

Prostorna složenost je $4 * |Genome|$ (ako koristimo 4B za cele brojeve kao elemente niza), što je bolje nego $20 * |Genome|$.

1.4 Približno preklapanje

Ponekad je nophodno pronaći približna uparivanja šablona.

Ulaz: Niska Pattern, niska Genome, ceo broj d (kod višestrukog uparivanja ulaz je kolekcija niski Patterns). **Izlaz:** Sve pozicije niske Genome, gde se niska Pattern pojavljuje kao podniska sa najviše d razlika.

Traženje preklapanja radimo kao i pre, samo što sada prihvatamo i kad imamo različite karaktere (crvena slova na slici ispod), sve dok je broj razlika $j = d$.

| | # Mismatches |
|--|--------------|
| \$ ₁ panamabananas ₁ | |
| a ₁ bananas\$pana m ₁ | 1 |
| a ₂ mabananas\$pan ₁ | 0 |
| a ₃ namabananas\$ p ₁ | 1 |
| a ₄ nanas\$panama b ₁ | 1 |
| a ₅ nas\$panamaban ₂ | 0 |
| a ₆ s\$panamabana n ₃ | 0 |
| b ₁ ananas\$panama ₁ | |
| m ₁ abananas\$pana ₂ | |
| n ₁ amabananas\$pa ₃ | |
| n ₂ anas\$panamaba ₄ | |
| n ₃ as\$panamabana ₅ | |
| p ₁ anamabananas\$ ₁ | |
| s ₁ \$panamabanan ₆ | |

Slika 1.28: Traženje približnog preklapanja za d = 1

Na kraju ovog primera pronašli smo 5 3-grama sa najviše jednim nepoklapanjem (pretpostavili smo da je $d = 1$).

| | Suffix Array |
|---|--------------|
| s_1 panamabananas s_1 | |
| a ₁ b anananas\$panam ₁ | 5 |
| a ₂ m abanananas\$pan ₁ | 3 |
| a ₃ n amabanananas\$p ₁ | 1 |
| a ₄ n anas\$panamab ₁ | 7 |
| a ₅ n as\$panamaban ₂ | 9 |
| a ₆ s\$panamabanan ₃ | |
| b ₁ anananas\$panam ₁ | |
| m ₁ abanananas\$pan ₂ | |
| n ₁ amabanananas\$pa ₃ | |
| n ₂ anas\$panamaba ₄ | |
| n ₃ as\$panamabana ₅ | |
| p ₁ anamabanananas\$ ₁ | |
| s ₁ \$panamabanan ₆ | |

Slika 1.29: Pozicije u genomu gde se javljaju približna preklapanja

1.5 Zadaci sa vezbi

1.5.1 TrieConstruction

1.5.2 SuffixReconstruction

1.5.3 BWT