SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5133

Algoritam staničenja velike baze podataka genoma

Mateo Stjepanović

Zagreb, lipanj 2017.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Zahvaljujem mentorici doc. dr. sc. Mirjana Domazet-Lošo na podršci.

SADRŽAJ

1. Uvod

Bioinformatika je interdisciplinarna znanost koja se bavi razvojem programa i metoda za interpretiranje bioloških podataka. Spaja matematiku, statistiku, računalnu znanost te druge prirodno matematičke znanosti. U posljednjem desetljeću bioinformatika kao znanost doživljava veliki porast. Ističu se metode analiziranja genoma, te, spajanjem statističkih analiza i algoritama iz područja računalne znanosti, metode određivanja položaja istih u taksonomskom stablu, usporedba genoma različitih vrsta radi proučavanja sličnosti istih, genetika bolesti, tj. proučavanje nasljeđivanja bolesti te pokušaji sprječavanja tih istih bolesti, te mnoge druge. Za potrebe ovog rada zanimljivo je područje određivanja položaja jedinki u taksonomskom stablu,te upravo u tom području postoje mnogi alati koji su razvijeni upravo s ciljem poboljšanja točnosti određivanja taksonomskog stabla novoj, nepoznatoj, jedinki.
Prilikom pokušaja klasificiranja jedinki iz nekog uzorka, moguća su dva slučaja. Prvi je da je jedinka potpuno nepoznata, te ne postoji nikakva sličnost s do sad poznatim vrstama. U tom slučaju se ona označava kao nova, nepoznata, vrsta te se pokušaj klasifikacije tu zaustavlja. Drugi slučaj je da postoje sličnosti s drugim vrstama te je potrebno pronaći upravo te sličnosti. Za to služe razni algoritmi poravnanja. Poravnanje predstavlja preraspored dijelova genoma tako da se poklapaju dijelovi koji su isti kod oba genoma. To se radi zbog određivanja evolucijske sličnosti između dva genoma. Jedan od alata koji koristi poravnanje je BLAST (Lobo (2008)). Postoje druge metode i programi koji pospješuju učinkovitost BLAST-a uvođenjem raznih metoda. MEGAN (Huson H. et al. (2007)) pretražuje pomoću BLAST algoritma preko više baza podataka, te korištenjem najmanjeg zajedničkog pretka klasificira podatak. Nadalje poboljšanje pokušavaju postići i PhymmBL (Brady i Salzberg.S (2009)) , NBC(*Naive Bayes Clasificator*) (Rosen L. et al. (2011))uvođenjem metoda strojnog učenja te algoritama umjetne inteligencije pokušava poboljštati rezultate, i dr.. Iako bolje preciznosti ti alati imaju jednu ozbiljnu manu. Njihova brzina izvođenja manja od one kod BLAST-a. Tada se pojavljuje alat po imenu Kraken (E. Wood i L.Salzberg (2014)), koji uspijeva uskladiti preciznost izvođenja BLAST-a te poboljšati brzinu.

Za razliku od alata koji su pokušali poboljšati preciznost BLAST algoritma, te time izgubili na brzini, Kraken je jedan od rijetkih alata koji postiže točnost koja premašuje onu u BLAST algoritmu, s time da ne gubi na brzini izvođenja (E. Wood i L.Salzberg (2014)). Rad Krakena se sastoji od toga da se ulazni podatak "razbija" na k-mere (Marçais i Kingsford (2011)), te se minimizer algoritmom (Roberts et al. (2004)) traže oni k-meri koji imaju istu vrijednost *minimzer-a*.
Minimizer predstavljaju mali podskup jedinstvenih uzoraka iz svih mogućih k-mera s ciljem poboljšanja brzine pretrage i izvođenja (Roberts et al. (2004)).
Tada se taj podatak uspoređuje s podacima u Kraken-ovoj bazi podataka, s tim da je baza sortirana tako da su podaci s istim *minimizerom* smješteni jedan pored drugog, tako da se pretraživanje pospješuje i ubrzava.
Autori alata su shvatili da Kraken može naići na problem prilikom izvođenja na računalu s ograničenim resursima, točnije na računalu s RAM-om ispod 70GB. Iz tog razloga je razvijena MiniKraken baza podataka koja je s prvotnih 70GB podataka smanjena na 4GB.

	HiSeq		MiSeq		simBA-5	
Klasifikator	Preciznost	Osjetljivost	Preciznost	Osjetljivost	Preciznost	Osjetljivost
NBC	82.33	82.33	77.78	77.78	97.64	97.64
PhymmBL	79.14	79.14	76.21	76.21	96.11	96.11
PhymmBL65	99.13	73.95	92.47	73.03	99.08	95.45
Kraken	99.20	77.15	94.71	73.46	99.90	91.25
Kraken-Q	99.12	76.31	94.69	70.41	99.92	89.54
MiniKraken	99.44	66.12	97.41	67.95	99.95	65.87
MiniKraken-Q	99.36	65.67	97.32	65.84	99.98	65.31
Kraken-GB	99.51	93.75	98.48	86.23	99.48	91.13

Tablica 1.1: Klasifikacija tri metagenoma (E. Wood i L.Salzberg (2014))

Iz tablice 1.1, koja predstavlja rezultate izvođenja na 3 metagenoma,koji su predstavljeni s tri vrste podataka (HiSeq, MiSeq i simBA-5), se može vidjeti kako Kraken prednjači u preciznosti svoje klasifikacije. Iako je osjetljivost nešto manja nego u Megablast-a, program koji radi na implementaciji BLAST algoritma za poravnanje, ta razlike je nezamjetna. Tablica prikazuje i neke druge alate, te Kraken alat s raznim opcijama i na raznim bazama podataka. Kraken-Q predstavlja izvedbu Krakena s uključenom opcijom *quick*, koja doprinosi brzini izvođenja. MiniKraken je Kraken program koji se izvodi na MiniKraken bazi podataka. Ona je stvorena iz originalne baze podataka tako da se uklanja svaki k-ti element iz nje.

	HiSeq		MiSeq		simBA-5	
Klasifikator	Preciznost	Osjetljivost	Preciznost	Osjetljivost	Preciznost	Osjetljivost
Kraken	99.20	77.15	94.71	73.46	99.90	91.25
MiniKraken	99.44	66.12	97.41	67.95	99.95	65.87

Tablica 1.2: Usporedba kvalitete izvođenja Kraken-a i Minikraken-a. (E. Wood i L.Salzberg (2014))

Iz tablice 1.2 se može vidjeti razlika u kvaliteti izvođenja između Kraken i MiniKraken baze podataka. Iako je alat osposobljen za izvođenje vidimo da osjetljivost jako opada, što nam baš i ne odgovara.
Osjetljivost predstavlja mjeru koja određuje postotak podataka klasificiranih kao točni, koji su uistinu točni. Npr. postotak ljudi koji su klasificirani kao bolesni, a to ustvari i jesu. Osjetljivost je predstavljena formulom (1.1) (Wikipedia (2017)), gdje *TPR* označava postotak podataka klasificiranih kao točni. *TP* predstavlja broj podataka koji su klasificirani kao točni te to uistinu i jesu, a *FN* predstavlja broj podataka koji su klasificirani kao negativni, ali to nisu.

$$TPR = TP/(TP + FN) \tag{1.1}$$

S druge strane preciznost predstavlja mogućnost točnosti određene kvalifikacije podatka. Npr. osobi su nalazima utvrđene neke abnormalnosti, koja je vjerojatnost da osoba uistinu ima određenu bolest. Preciznost je određena formulom (1.2) (Wikipedia (2017)), gdje *PPV* označava preciznost. *TP* predstavlja broj podataka koji su klasificirani kao točni, te to ujedno i jesu, a *FP* predstavlja broj podataka koji su klasificirani kao točni, ali to nisu.

$$PPV = TP/(TP + FP) \tag{1.2}$$

Rješenju problema smanjene kvalitete nakon smanjenja baze podataka pristupa ovaj rad. Pokušava se riješiti problem iskoristivosti resursa na velikoj bazi podataka. Iako će se brzina izvođenja očekivano smanjiti, za potrebe ovog rada je potrebno postići pretragu svih 70GB baze podataka u puno manjem RAM-u. Ideja rješenja je da se algoritmima sličnim algoritmima straničenja, te samim algoritmima straničenja, ospособi baza podataka tako da se u RAM učitavaju samo oni dijelovi za koje smo odredili da mogu sadržavati podatak koji nama treba. Originalnu bazu podataka je potrebno prvotno razdijeliti na manje baze podataka, zatim su moguća dva pristupa. Prvi je

učitavanje slijedno bazu po bazu podataka te pretraživanje svake, kako bi se postiglo straničenje. Drugi slučaj je određivanje one baze podataka koja može sadržavati podatke koji nama trebaju. To se postiže binarnim pretraživanjem po indeksima baze podataka. Indeksiranje ,koje će nam poslužiti za određivanje dijela baze podataka koji nam odgovara, bazirat će se na već pripremljenim "spremnicima" koji sadrže one podatke koji imaju iste *minimizer* vrijednosti.

2. Strukture podataka i algoritmi u Kraken-u

2.1. Baza podataka

Kreiranje baze podataka

Nastanak baze podataka se odvija u nekoliko koraka. Za početak se sa NCBI-ove stranice (NCBI (2017)) preuzimaju biblioteke koje sadrže genome. Naravno nije potrebno preuzeti sve podatke, nego se može preuzeti samo jedno carstvo ili samo jedan podskup podataka od kojih će se graditi baza podataka. Budući da se pretraživanje baze podataka u Kraken-u temelji na k-merima potrebno je odrediti k-mere za svaki podatak u bazi podataka. Za to služi alat naziva Jellyfish (Marçais i Kingsford (2011)) koji računa k-mere zadane duljine 31 (korisnik također može zadati duljinu k-mera).
Jellyfish je alat za brzo i memorijski povoljno prebrojavanje podnizova u danom nizu. Za potrebe prilagodbe svim resursima za Jellyfish postoje opcije za brzo izračunavanje koje je memorijski zahtjevnije, te ono sporije ali memorijski ne toliko zahtjevno. U kontekstu bioinformatike Jellyfish se koristi za prebrojavanje k-mera u zadanom metagenomu. Sam rad se bazira na tablici raspršenog adresiranja koja je osposobljena za paralelan rad preko više CPU-a. Tablica raspršenog adresiranja se sastoji od para (ključ,vrijednost), gdje je ključ zadani k-mer a vrijednost je broj njegovog ponavljanja u metagenomu. (Marçais i Kingsford (2011))

Tablica 2.1: Prikaz niza te njegovih k-mera (k = 4)

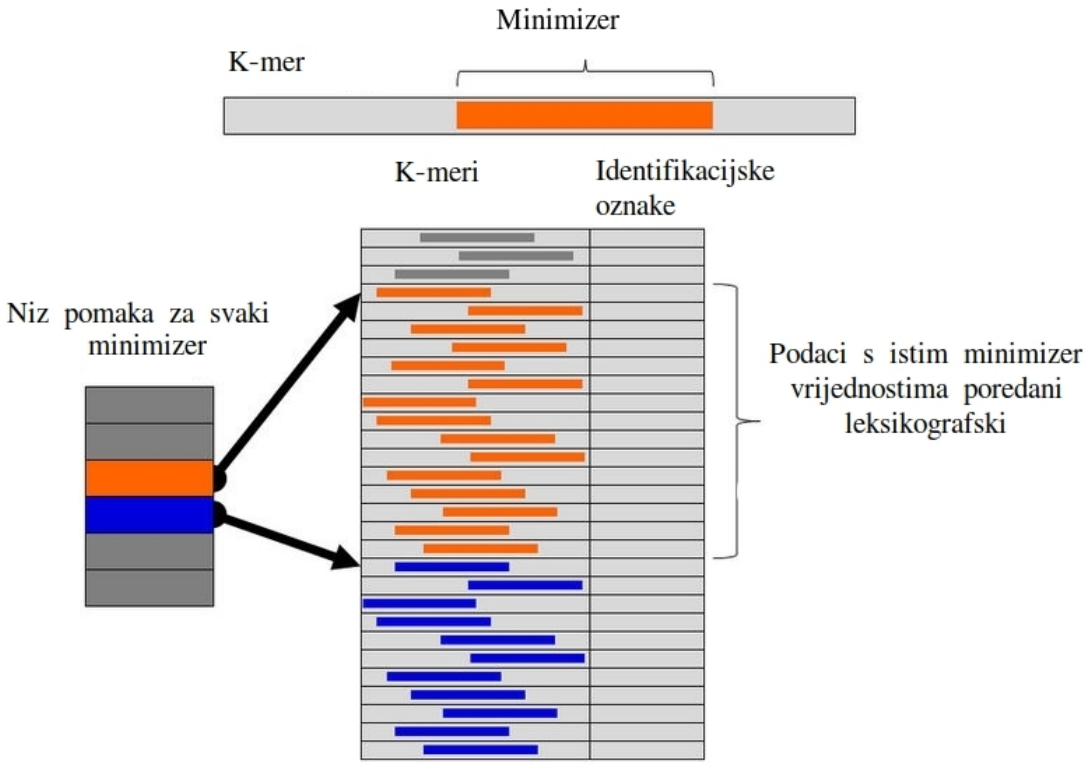
Ulaz:	AGATCGAGTG						
4-mer:	AGAT	GATC	ATCG	TCGA	CGAG	GAGT	AGTG

Nakon rada Jellyfish-a u bazu se spremaju vrijednosti k-mera te se kreira identifikacijska oznaka za svaki podatak. Tada program kreiranja baze podataka ponovno prolazi kroz bazu podatka, te za svaki podatak izračunava njegov LCA - Lower common

ancestor(*Najmanji zajednički predak*)(El Moufatich (2008); A. Bender et al. (2005); Aho et al. (1973)), te ga sprema u bazu podataka zajedno s identifikatorom i njegovim k-merima.

2.1.1. Strukture baze podataka

Gore navedeni proces klasificiranja ulaznog genoma pomoću baze podataka se, kao što je već navedeno, bazira na pretraživanju k-mera. Baza podataka se sastoji od identifikacijskih oznaka svakog podatka u njoj, njihovog izračunatog najmanjeg zajedničkog pretka, te k-mera koje on tvori.



Slika 2.1: Prikaze strukture baze podataka (E. Wood i L.Salzberg (2014))

Na slici 2.1 je prikazana struktura baze podataka koja se temelji na razvrstavanju i grupiranju podataka koji imaju istu *minimizer* vrijednost. Tada se binarnim pretraživanje nalazi onaj spremnim u kojem se nalaze podaci s istom *minimizer* vrijednošću kao i ulazni niz.

S ciljem poboljšanja brzine pretraživanja, uvodi se algoritam indeksiranja i raspoređivanja poznat pod imenom Minimizer (Roberts et al. (2004)). Računanje indeksa

spremnika se odvija kroz dvije petlje. U prvoj petlji se iterira po broju spremnika te se za svaki od njih dohvaća njihov jedinstveni k-mer. Na osnovu njega se računa ključ te se popunjava niz brojem podataka koji ulaze u taj indeks. Nakon toga se računa pomak svih podataka te se zbrajaju s onim podacima koji se nalaze u nizu indeksa. Na kraju se iz varijable ofset kreira indeks vanjskom funkcijom.

```
ptr <- pokazivač_na_k-mer;
k_ct <- brojač kontejnera;
vrijednosti <- 1ull [a] « (nt * 2);
while brojač je različit od broja kontejnera do
    brojač++;
    dohvati_k-mer(ptr);
    ključ <- izračunaj_ključ();
    b_brojač[ključ]++;
end
b_pomak[vrijednosti +1];
for i < vrijednosti do
    b_pomak = b_pomak[i-1]+b_brojač[i-1];
end
indeks(b_pomak);
```

Algoritam 1: Računanje indeksa spremnika (E. Wood i L.Salzberg (2014))

[a] *Unsigned long long 1*

Algoritam 1 prikazuje načina računanja indeksa spremnika realiziran u izvornom kodu Kraken-a. Kroz petlju se dohvaćaju svi k-meri te se računaju ključevi za tablicu raspršenog adresiranja, te se povećava brojač koji označava koliko podataka se nalazi u jednom spremniku s jednim ključem. Tada se realizira pomak te se za vodi evidencija koliki je pomak potreban za svaki spremnik, tj. za koliko se mora pomaknuti pokazivač da bi došao do spremnika sa drugim ključem.

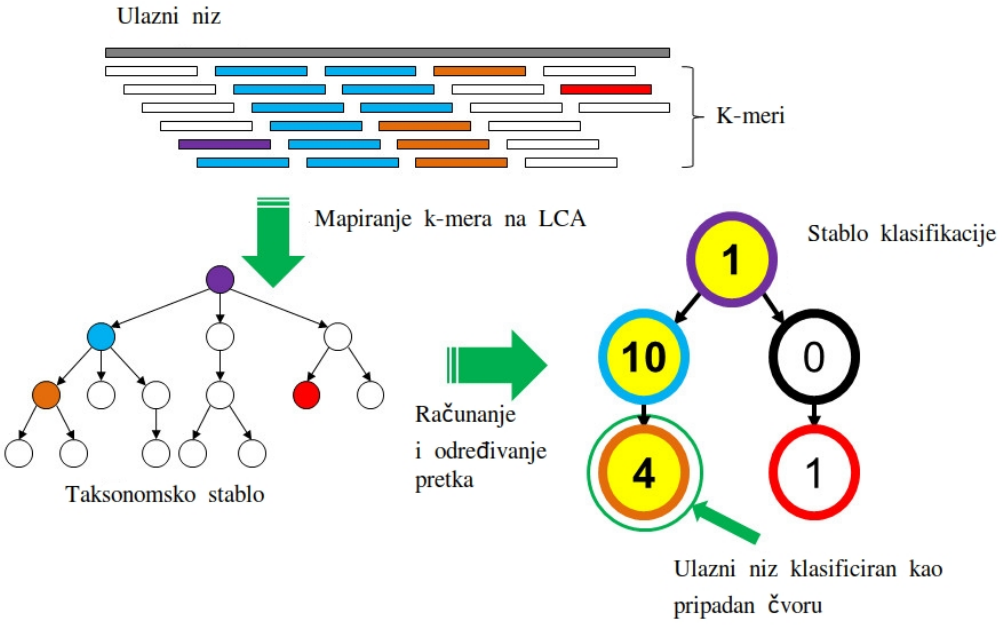
Prilikom sortiranja i indeksiranja se stvaraju svojevrsni spremnici u bazu podataka slijedno sadržavaju one podatke čiji k-meri imaju iste *minimizer* vrijednosti. Podaci u svakom spremniku su leksikografski poredani kako bi se isti osposobio za daljnje binarno pretraživanje. Stvaranjem zasebnih spremnika se poboljšava pretraživanje na način da se za svaki ulazni podatak računaju njegovi k-meri te njegove *minimizer* vrijednosti, te se pretragom liste indeksa dohvaća samo onaj spremnik koji sadrži istu *minimizer* vrijednost. Prilikom dolaska novog podataka *minimizer* vrijednost se ne ra-

čuna odmah, nego se prvo vrši pretraga na spremniku na kojem smo već pozicionirani. Ako se podatak ne nalazi u zadanom spremniku, tada se prelazi na računanje *minimizer* vrijednosti. Ako je *minimizer* vrijednost ista kao i kod spremnika tada se podatak svrstava u neklasificirane te se svaka daljnja klasifikacija zaustavlja. U slučaju dobivanja *minimizer* vrijednosti različite od trenutne pozicionira se na indeks izračunatog spremnika te se isti pretražuje. Način pretrage i podjele baze podataka prikazan je slici 2.1.

2.2. Algoritam klasifikacije podataka

Klasificiranje ulaznog podatka se odvija na više razina. Za početak se za ulazni genom računaju svi njegovi k-meri (k može biti zadan inicijalno ili ga korisnik može zadati). Zatim se za svaki k-mer kreiraju njegove *LCA* vrijednosti, tj. traži se onaj podatak, u taksonomskom stablu, koji predstavlja prvog pretka tog k-mera. Od svih k-mera se tada kreira stablo koje predstavlja sve moguće klasifikacije za dani genom, a koje su dobivene upravo navedenim algoritmom. Svaka vrijednost u stablu ima izračunatu vrijednost koja predstavlja koliko k-mera se podudara sa tim podatkom. Te vrijednosti se nazivaju težinama čvora stabla. Ako za neki k-mer ne postoji poznati LCA , tada se on neće ni stavljati u navedeno stablo. Način rada koji uvodi težine za svaki čvor u stablu omogućuje veliku točnost u radu Krakena-a, tako da ako postoji neki put u stablu koji je pogrešan a težine su jako male, on će se zanemariti. Ako postoje dva puta u stablu koja imaju iste težine tada se samo za te podatke ponovno računa LCA te se pokušava odrediti koji put zapravo predstavlja točnu klasifikaciju. (E. Wood i L.Salzberg (2014))

Iz slike 2.2 se može vidjeti način rada algoritma. Nakon izračuna svih k-mera ulaznog podatka, isti se pokušavaju klasificirati i izgraditi taksonomsko stablo pomoću vrijednosti najmanjeg zajedničkog pretka (LCA). To se izvodi pomoću predefinirane baze podataka. Upravo opisani dio rada prikazan je na slici 2.2 u gornjem dijelu slike. Nakon toga se računaju težine za svaki čvor u stablu. Težine predstavljaju broj k-mera koji se podudaraju sa tim istim čvorom. Težine su predstavljene na slici 2.2 u desnom kutu. Ako postoji više različitih puteva koji imaju iste težine, tada se za iste ponovno računa najmanji zajednički predak te se pokušava dokučiti koji je pravi put, tj. koji od čvorova je pravi predak između listova koje smo tražili. Nadalje ako postoji više puteva ali oni nisu istih težina, tada se put do zajedničkog pretka uzima onaj put koji ima najveću težinu, tj. kojem odgovara najviše k-mera. To je prikazano na slici 2.2 u desnom kutu, gdje možemo vidjeti da se od dva različita puta bira onaj lijevi upravo



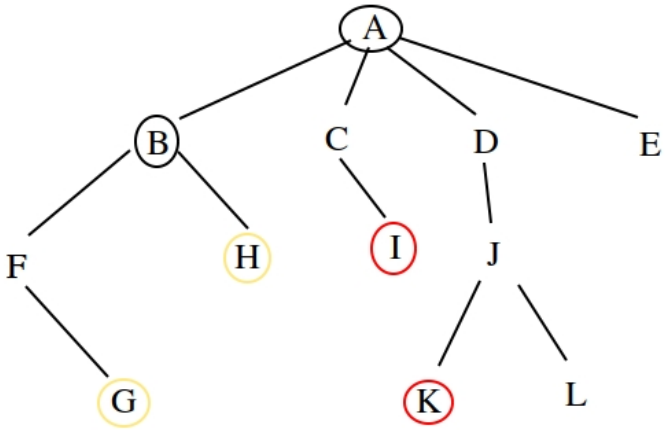
Slika 2.2: Prikaz klasifikacije podataka (E. Wood i L.Salzberg (2014))

zbog toga jer je njegov zbroj težina veći od desnog. S time se zaključuje da su težine proporcionalne vjerojatnosti točnosti zadanog puta. (E. Wood i L.Salzberg (2014))

2.3. LCA(*Lower Common Ancestor*)

LCA(*Lower common ancestor*) je jedan od osnovnih algoritamskih problema u strukturama stabala.A označava pronalazak čvora u stablu koji je najudaljeniji od korijena stabla, te je zajednički čvorovima *u* i *v*. Prvu ideju i definiciju LCA dali su Alfred Aho, John Hopcroft i Jeffrey Ullman 1973. godine u svom radu *On finding lowest common ancestors in trees*. Zbog svoje složenosti te raznolike primjenjivosti ovaj problem je još uvijek aktualan, te se i danas objavljuje mnogo članaka i radova na tu temu. U uvodu svog rada autori navode glavni problem koji LCA pokušava riješiti. Problem se navodi kao primanje informacija o novim osobama u određenim vremenskim intervalima, te je pomoći zadanog algoritma potrebno točno odrediti sve veze između svakog novog čvora koji predstavlja danu informaciju. (El Moufatich (2008))

Na slici 2.3 se nalazi prikaz strukture stabla te su bojama označeni listovi za koje želimo naći najmanjeg zajedničkog pretka (LCA). Crno obojeni čvorovi su čvorovi



Slika 2.3: Prikaz stabla i LCA vrijednosti

koji su preci tim dvama listovima. Tako je za listove H i G najmanji zajednički predak čvor B, dok je za listove I i K najmanji zajednički predak A. Ta ideja je predstavljena u radu *On finding lowest common ancestors in trees* 1973. godine, te je predstavljen sljedeći pseudokod za određivanje najmanjeg zajedničkog pretka. (El Moufatich (2008); A. Bender et al. (2005); Aho et al. (1973))

```
procedure getancestor(u,i):
if ancestor(u,i-1) = undefined then; then
    ancestor(u,i) <- getancestor(getancestor(u,i-1),i-1);
end
resultis ancestor(u,i);
```

Algoritam 2: Dohvaćanje pretka zadanog čvora El Moufatich (2008)

Algoritam 2 predstavlja metodu getancestor(u,i), koja prima čvor *u* te dubinu *i*. Funkcija dohvaća pretka koji je na 2^i-toj poziciji te ga stavlja u niz koji sadrži pretke zadanog čvora. *Getancestor* se poziva tijekom traženja najmanjeg zajedničkog pretka. Ova metoda predstavlja očuvanje svih informacija o precima za jedan čvor.

```
procedure find(u,v,i,d):
if i = 0 then
    resultis ancestor(u,0);
else
    if getancestor(u,i-1)=getancestor(v,i-1) then
        resultis find(u,v,i-1,d);
    else
        resultis find(getancestor(u,i-1),getancestor(v,i-1),min(i-1,|_log(d-
          2^(i-1))_|),d-2^(i-1));
    end
end
```

Algoritam 3: Pronalazak najmanjeg zajedničkog pretka između dva čvora (El Moufatich (2008))

Algoritam 3 predstavlja metodu koja pronalazi najmanjeg zajedničkog pretka za ulazne podatke u i v koji su na dubini d. Početne pretpostavke su da je $d > 2^{(i-1)}$ te su im 2^i-ti preci jednaki ili ne postoje. Zadana metoda radi na način da u svakom novom koraku smanjuje radni prostor koji pretražuje , tako da je put do najmanjeg zajedničkog pretka uvijek u zadanom prostoru. Gore navedene dvije metode omogućuju stvaranje metode koja osigurava pronalazak najmanjeg zajedničkog pretka uz pravilno biranje početnih uvjeta. Budući da se zadnji uvijek računa logaritamskom funkcijom, algoritam je optimiziran tako da se podaci predstavljaju kao binarne znamenke, te se u svakom koraku petlje pomiče za jedan bit dok ne dođe do kraja.

3. Algoritmi indeksiranja i pretrage

3.1. Minimizer - algoritam za reduciranje podataka

Usporedba podataka je osnova moderne bioinformatike. Ona se danas temelji na metodi *seed-and-extend* (pronađi uzorak). Za svaki ulazni podatak se uzima samo jedan uzorak, te se on uspoređuje u bazi podataka. Ako se nađe poklapanje tada se taj uzorak proširuje na cijeli podatak te se ponovno uspoređuje. Da bi sve to funkcioniralo potrebno je uzeti uzorak za svaki podskup u jednom genomu, što , za kompleksnije genome, zauzima jako puno memorije.Na ideju rješenja su došli Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount i James A. York u svom radu *Reducing storage requirements for biological sequence comparison* (Roberts et al. (2004)). Koristi se isti način rada, tj.*seed-and-extend*, uz razliku da se ne uzimaju svi uzorci, nego samo mali podskup jedinstvenih, nazvanih *minimizer-i*.(Roberts et al. (2004))
Navedeni algoritam sažimanja se koristi i u Kraken bazi podataka, te ju uvelike smanjuje i pospješuje brzinu izvođenja. Prilikom kreiranja MiniKraken baze podataka se koristi isti način rada, uz razliku da se odbacuje samo određeni postotak prvih i zadnjih vrijednosti. Da bi spremnici u kojima su sadržani podaci s istim *minimizer* vrijednostima zadržao svojstvo kolekcije, potrebno je da su ti isti podaci normalno distribuirani, što se postiže brisanjem pola bitova prilikom ispitivanja jednakosti. S ciljem pospješenja brzine izvođenja, jer svako računanje *minimizer* vrijednosti je relativno skupo, prilikom klasifikacije podatka koristi se onaj spremnik koji je korišten prije, te ako k-meri nisu nađeni u njemu, računa se *minimizer* vrijednost. Ako je *minimizer* vrijednost jednaka kao u spremniku zaključuje se da se podatak ne može klasificirati, inače se pozicionira u dani spremnik te se ponovno vrši pretraga. (E. Wood i L.Salzberg (2014))

Postoje dvije vrste *minimizer* vrijednosti: unutarnje i vanjske. Svakoj od njih je srž računanja isti, tj. za ulazni niz se računaju svi njegovi k-meri za zadani k, te se iz njih tada bira onaj k-mer koji je leksikografski najmanji.

Slika 3.1 prikazuje način odabira *minimizera* za ulazni niz. Ulazni niz je podijeljen

Pozicija:	1	2	3	4	5	6	7
Ulazni niz:	2	3	1	0	3	4	3
k-meri	2	3	1				
		3	1	0			
s			1	0	3		
podebljanim				**0**	**3**	**4**	
minimizerom					3	4	3

Slika 3.1: K-meri ulaznog niza s minimizer-om (Roberts et al. (2004))

na 3-mere , a za računanje *minimizer-a* se koristi prozor od 5 k-mera. Korištenje prozora znači da se samo u 5 k-mera odjednom traži *minimizer* vrijednost. Budući da je *034* leksikografski najmanji podniz on se uzima kao zadani *minimizer*. Algoritam *minimizer* nam garantira da će dva niza koja su ista sadržavati barem jednu istu *minimizer* vrijednost.

3.2. Binarno pretraživanje

Za pretraživanje svakog spremnika se koristi binarno pretraživanje. Binarno pretraživanje je jedan od osnovnih algoritama u računalnoj znanosti. Glavni zadatak algoritma je pronalazak vrijednosti u sortiranom nizu. Osnovna i glavna pretpostavka jest da je niz sortiran, u suprotnom sam algoritam nema smisla.

```
binarno_pretraživanje(niz,cilj):
lo = 1, hi = size(niz);
while lo <= hi do
    mid = lo + (hi - lo) / 2;
    if niz[mid] == cilj then
        return mid;
    else
        if niz[mid] < cilj then
            lo = mid + 1;
        else
            hi = mid + 1;
        end
    end
end
```

Algoritam 4: Binarno pretraživanje

Inicijalno je radni prostor binarnog pretraživanja cijeli ulazni niz. Tada se pozicionira na sredinu tog niza, te se ispituje je li vrijednost na sredini jednaka onoj koju mi tražimo. Ako je rezultat potvrdan izlazimo iz petlje te vraćamo poziciju te vrijednosti. U protivnom se ispituje odnos tražene vrijednosti i one koja je na sredini. Ako je vrijednost na sredini manja od tražene radni prostor postavljamo na onu polovicu koja se nalazi desno od indeksa, tj. prelazimo u polovicu koja sadrži veće vrijednosti, u suprotnom se pozicioniramo na manju polovicu. Složenost binarnog pretraživanja je O(logN), gdje je N broj znakova u nizu. Logaritam je sporo rastuća funkcija, stoga je binarno pretraživanje jako efikasan algoritam. Kao primjer za pretragu imenika od milijun imena, pomoću binarnog pretraživanja bismo pronašli traženu vrijednost u najviše 21 korak.

4. Pseudokod i razrada algoritma

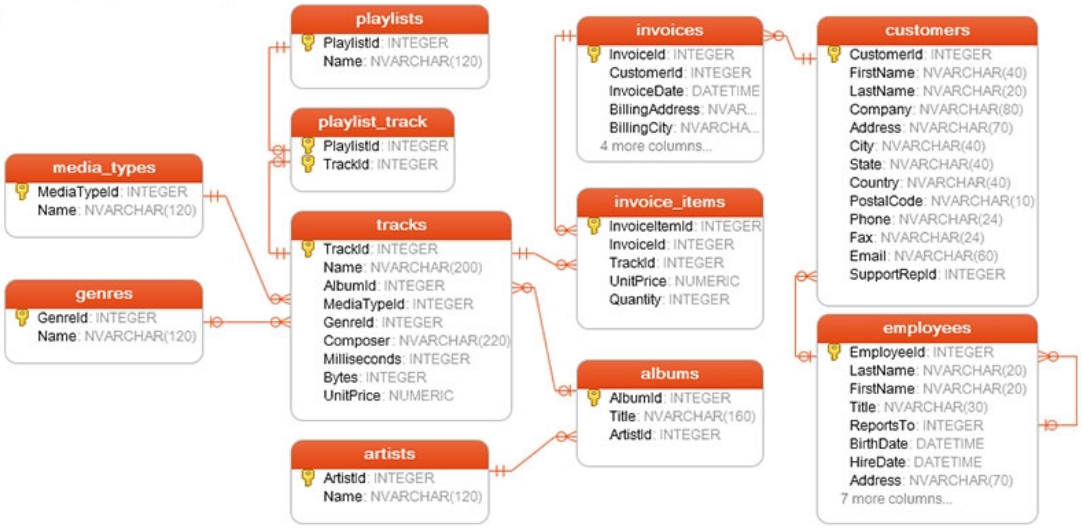
Rješenja navedenog problema o korištenju baze podataka na ograničenim resursima se sastoji iz nekoliko dijelova. Za početak je potrebno podijeliti bazu podataka na manje cjeline, zatim je potrebno ostvariti straničenje na tim cjelinama.

4.1. Problemi kod korištenja Kraken baze podataka

Iako je prvotna ideja bila koristiti Kraken bazu podataka, te je algoritam i osmišljen upravo za nju, zbog tehničkih problema se moralo odustati od te ideje. Pod tehničke probleme misli se na nemogućnost pristupa bazi podataka, a bez pristupa se nije mogao provesti nijedan algoritam nad njom. Sama baza podataka je u nepodržanom formatu za bilo koji preglednik baze podataka koji sam uspio naći, a proučavanjem izvornog koda Kraken alata se nije moglo puno zaključiti. Kod je slabo zakomentiran te je veliki problem snaći se u takvom okruženju. Stoga se za daljnji rad koristi jednostavna baza podataka od par stotina kilobyte-a.(SQlite Tutorial (2017))

4.2. Podjela baze podataka

Iz gore navedenih razloga odlučeno je prezentirati rad algoritma, te samu raspodjelu baze podataka na uzorku baze podataka. (SQlite Tutorial (2017)) Baza podataka je sqlite3 tipa, te se u izvornom kodu programa koristi paket sqlite3.
Podjela baze podataka na cjeline je ostvarena bash skriptom, koja pomoću sqlite3 naredbi iz zadane baze podataka kreira datoteku koja sadrži sql naredbe koje stvaraju kostur baze podataka, tj. stvara zadani broj (trenutno je to 5 baza podataka) identičnih baza podataka kao originalna ali bez ijednog podatka u sebi.
Budući da je već gotovu bazu podataka gotovo nemoguće podijeliti na funkcionalne baze, morao se koristiti pristup gdje se ista pretvara u niz sql upita.



Slika 4.1: Prikaz korištene baze podataka (SQlite Tutorial (2017))

if *postoji schema.sql* **then**
    izbriši schema.sql;
end
if *postoji dump.sql* **then**
    izbriši dump.sql;
end
sqlite3 imebaze .schenm -> schema.sql;
sqlite3 imebaze .dump -> dump.sql;

Algoritam 5: Podjela baze podataka

Gore navedenim dijelom koda se stvaraju datoteke schema.sql i dump.sql. Schema.sql sadrži SQL naredbe koji stvaraju strukturu originalne baze podataka, a dump.sql se sastoji od SQL naredbi koji pune bazu podataka podacima iz originalne baze podataka.

Budući da želimo bazu podataka podijeliti na cjeline potrebno je datoteku dump.sql podijeliti na onoliko datoteka koliko želimo da imamo baza podataka. U svrhe demonstracije rješenja dump.sql je podijeljen u 5 cjelina koje su slijedno razdvojene.

Za Kraken bazu podataka se može koristiti datoteka koja sadrži indekse svakog spremnika, te možemo dobiti baze podataka koje sadrže samo one podatke čiji k-meri imaju jednake minimizer vrijednosti. Iza tog problema se krije ne jako zahtjevna logika koju relativno lako možemo realizirati u bash skripti.

Time ne završava podjela baze podataka na manje dijelove, naime potrebno je kreirati svaku bazu podataka te unijeti podatke u svaku zasebno. Iz dump.sql datoteke je do-

biveno 6 dump.split.000x.sql datoteka , koje će svaka za sebe graditi bazu podataka. Kako smo datoteku podijelili pomoću jednostavne *split* naredbe u bash-u dobivene datoteke nisu pravilno konfigurirane za kreiranje. Stoga je potrebno , ako ne postoji, na početku svake datoteke upisati SQL naredbu *BEGIN TRANSACTION*, te na kraju iste , opet ako ne postoji, upisati naredbu *COMMIT*. To je izvedeno sljedećim dijelom bash skripte.

```
for svaku datoteku u direktoriju do
    if "BEGIN TRANSACTION" se ne nalazi na početku datoteke then
        dodaj_BEGIN_TRANSACTION_na_početak;
    end
    if "COMMIT" se ne nalazi na kraju datoteke then
        dodaj_COMMIT_na_kraj;
    end
end
```

Algoritam 6: Osposobljavanje dump datoteka

Naposljetku je potrebno kreirati bazu podataka. To se radi na način da se za svaku dump.split.000x.sql datoteku kreira baza podataka koja sadrži samo kostur originalne baze, te se u nju tada pomoću navedene datoteke ubacuju podaci. Na kraju izvedbe zadanog programa smo dobili 6 baza podataka koje zajedno imaju sve podatke koje je zadržavala originalna baza podataka.

4.3. Straničenje

Za prezentaciju algoritma straničenja koristit će se jednostavni SQL upit koji želi dohvatiti sve podatke za neku tablicu. Algoritam straničenja je prvotno bio osmišljen u verziji koja bi bila dosta brža, ali za ovaj rad je obrađeno samo standardno straničenje.

```
for ima podataka u listi do
    for i < broj_baza_podataka do
        otvori bazu podataka();
        pozovi SQL upit nad bazom podataka();
        zatvori bazu podataka();
    end
end
```

Algoritam 7: Algoritam straničenja

Gore navedeni pseudokod predstavlja algoritam koji je implementiran za potrebe ovoga rada. Poboljšana verzija bi se sastojala od toga da se za ulazni podataka izračuna

indeks te se otvori i pretraži samo baza podataka koja može sadržavati zadani indeks. Taj algoritam je predstavljen sljedećim pseudokodom.

```
for ima podataka u listi do
    izračunaj indeks za ulazni podatak(); if otvorena baza je istog indeksa ko
     podatak then
        pozovi SQL upit nad bazom podataka();
    else
        zatvori bazu podataka();
        pozovi SQL upit nad bazom podataka();
        otvori bazu podataka();
    end
end
```

Algoritam 8: Poboljšani algoritam straničenja

Svi podaci su ostali sačuvani, te je za tablicu imena *albums*, SQL upitom SELECT * FROM albums rezultat prikazao sve podatke koji se nalaze u originalnoj bazi.

5. Analiza učinkovitosti rješenja

Zbog nemogućnosti korištenja Kraken i MiniKraken baze podataka testovi koje je moguće provesti za zadani algoritam su ograničeni. Prvi provedeni test je izveden naredbom SELECT * FROM playlists te je utvrđeno da su svi podaci ostali sačuvani, te da *callback* metoda, koja je zaslužena za ispisivanje svih dobivenih podataka, nema nikakvih problema s njihovom interpretacijom.

```
1 static int callback(void *data, int argc, char **argv,char **↩
    azColName){
2    int i;
3    fprintf(stderr, "%s:", (const char*)data);
4    for(i=0; i<argc; i++){
5       printf("%s=%s\n", azColName[i],argv[i] ? argv[i] : "NULL");
6    }
7    printf("\n");
8    return 0;
9 }
```

Algoritam 8 predstavlja funkciju koja za prima rezultat sql naredbe te ga parsira u format koji je prikladan za čitanje.
S obzirom na vrijeme izvođenja u samom početku se moglo zaključiti da će program raditi sporije, pogotovo na većim bazama koje će se morati učitati u RAM. Međutim cilj ovog rješenja je osposobljavanje na rad, ali i spremnost na nešto sporiji rad. Naravno sve se još da optimizirati korištenjem indeksa za interpretaciju i dohvaćanje raznih baza podataka. Jedan od glavnih problema ovog pristupa je bio kako uspješno već gotovu bazu podataka raspodijeliti u manje a da podaci ostanu kontinuirani i indeksi u svim tablicama ostanu očuvani, što je uspješno odrađeno. Iako neobičan, pristup problemu tako da se cijela baza podataka pretvori u niz SQL upita se pokazao kao jako dobar, te će se zasigurno koristiti u mom daljnjem istraživanju i pokušavanju rješenja

navedenog problema. Budući da je baza podataka jako mala nije se mogao provesti smislen test potrošnje vremena koji će potkrijepiti gore navedenu teoriju.

5.1. Za budućnost

U budućem radu ću se više osloniti na implementaciju rješenja konkretno na Kraken bazu podataka, koja je do sada predstavljala veliki problem. Pokušat kontaktirati autore samog alata se u dogovoru s njim raditi na rješavanju, po mom mišljenju, jednog jako zanimljivog problema koji može poboljšati Kraken . Budući rad podrazumijeva dolaženje do podataka koji se nalaze u Kraken bazi podataka, te podjela iste baze na manje cjeline. Razlika od dosadašnjeg rada je da se baza podataka neće dijeliti na bilo koje cjeline, nego će jednu novu bazu podataka činiti jedan spremnik jedinstvenog *minimizera*. Tada u izradu nastupa implementiranje straničenja podataka te poboljšanje samog straničenja korištenjem liste indeksa.

6. Zaključak

Bioinformatika je interdisciplinarna znanost koja spaja više prirodno-matematičkih znanosti. U zadnjim desetljećima je doživjela veliki porast. Upravo zbog svog razvitka se razvila potreba za programima koji mogu jako brzo i precizno klasificirati ulazni metagenom. Jedan od takvih programa je i Kraken. Svojom izrazitom preciznošću i brzinom izvođenja u vrhu je alata da klasificiranje genoma, i zbog toga je odabran za analizu u ovom radu. Uspješno spaja nekoliko algoritama za indeksiranje, pretragu te klasifikaciju podataka. Za klasificiranje podataka se koristi LCA algoritam, te za indeksiranje i pretragu baze podataka se koristi minimizer algoritam. Problem kojim se bavi ovaj rad se odnosi upravo na bazu podataka koja je prevelika za učitavanje u RAM na većini osobnih računala.
Kako bazu od nekoliko desetak GB ubaciti u RAM koji je samo nekoliko GB?
Idejom raspodjele baze podataka te straničenjem tih baza se pokušalo približiti rješenju, te se ovim radom to i uspjelo. Iako se poboljšao utrošak memorije, vremenska složenost pretrage je porasla, što je bilo očekivano. Rana testiranja, koja se nisu uspjela odviti na Kraken bazi podataka, su pokazala jednu od najbitnijih značajki koja se morala zadržati, a to je očuvanje svih podataka i njihovih indeksa, te jednakost rezultata pretrage na originalnoj bazi podataka i manjim bazama. U daljnjem radu na ovom problemu pokušat će se sve to implementirati na prvotno namijenjenu bazu podataka te ispitati ponašanje u tom slučaju.

LITERATURA

Michael A. Bender, Martín Farach-Colton, Giridhar Pemmasani, Steven Skiena, i Pavel Sumazin. Lowest common ancestor in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75 – 94, 2005.

A. V. Aho, J. E. Hopcroft, i J.D. Ullman. On finding lowest common ancestors in trees. *STOC '73 Proceedings of the fifth annual ACM symposium on Theory of computing*, stranice 253 – 265, 1973.

Arthur Brady i Steven Salzberg.S. Phymm and PhymmBL: metagenomic phylogenetic classification with interpolated Markov models. *Nat Methods*, 6:673 – 676, 2009.

Derrick E. Wood i Steven L.Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology 15:R46*, 2014.

Fayssal El Moufatich. Lowest Common Ancestor(LCA), 2008.

Daniel Huson H., Alexander Auch F., Ji Qi, i Stephen Schuster.C. MEGAN analysis of metagenomic data. *Genome Res*, 17:377 – 386, 2007.

Ingrid Lobo. Basic Local Alignment Search Tool (BLAST). *Nature Education 1(1):215*, 2008.

Guillaume Marçais i Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurances of k-mers. *Bioinformatics*, 27(6):764 – 770, 2011.

NCBI. Nacional center for biotechnology information, 2017. URL `https://www.ncbi.nlm.nih.gov/`. Pristupio: 24.06.2017.

Michael Roberts, Wayne Hayes, Brian R.Hunt, Stephen M.Mount, i James A.Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363 – 3369, 2004.

Gail Rosen L., Erin Reichenberger L., i Aaron Rosenfeld M. NBC: the Naïve Bayes Classification tool webserver for taxonomic classification of metagenomic reads. *Bioinformatics 27(1):127 - 129.*, 2011.

SQlite Tutorial. Sqlite sample database, 2017. URL `http://www.sqlitetutorial.net/sqlite-sample-database`. Pristupio: 24.05.2017.

Wikipedia. Sensitivity and specificity — wikipedia, the free encyclopedia, 2017. URL `https://en.wikipedia.org/w/index.php?title=Sensitivity_and_specificity&oldid=786237802`. Pristupio: 15.05.2017.

Algoritam staničenja velike baze podataka genoma

Sažetak

Straničenje velike baze podataka se odvija u dva koraka. Prvi je podjela baze podataka na manje cjeline(baze podataka manjeg memorijskog kapaciteta). On je ostvaren pomoću bash skripte koja, uz sqlite3 naredbe, uspijeva napraviti kostur originalne baze te SQL upite za punjenje baze podacima.
Drugi korak je samo straničenje koje se iterira po svim bazama te traži podatak koji želimo obraditi. Za nastavak i poboljšanje straničenja se može uvesti lista indeksa baza podataka te se jednostavnim pretraživanjem liste može utvrditi samo jedna baza podataka koju trebamo učitati.

Ključne riječi: straničenje, baza podataka, Kraken, metagenom, k-mer, LCA, minimizer

Algorithm for Paging Large-Scale Genome Database

Abstract

Paging of a large-scale database is made from two steps. The first one is splitting a database into smaller units(databases that contains less memory). It is done by the bash script that manages to create skeleton of original database, and to fill the databases with data.
The second step is paging itself that iterates over all databases and searches the data that we want to process. For of paging one can use index list and, by simple search of that list, can determine only one database that needs to be loaded .

Keywords: paging, database, Kraken, metagenom, k-mer, LCA, minimizer

A. Upute za korištenje programske potpore

Upute za korištenje programske potpore:

1. Za koristiti programsku podršku potrebno je raditi na nekoj od Linux distribucija, te ista mora sadržavati instalirane sqlite3 i sqlite3-devel pakete.

2. Za koristiti programsku podršku potrebno je raditi na nekoj od Linux distribucija, te ista mora sadržavati instalirane sqlite3 i sqlite3-devel pakete.

3. Za rad sa izvršnim programom preuzeti direktorij koji sadrži sve zadane komponente.

4. Pozicionirati se u direktorij Database.

5. Pokrenuti skriptu splitScript.sh sa argumentima imena baze podataka te brojem baza koje želimo napraviti.

6. Dump.sql i schema.sql će biti spremljeni u trenutni direktorij.

7. Baze podataka će biti spremljene u direktorij Databases dok će raspodijeljena dump datoteka biti u Splitted direktoriju.

8. Pokrenuti Makefile.

9. Pokrenuti izvršni program sa argumentom broja baza koje želimo koristiti.