Asembliranje sekvenci

Seminarski rad u okviru kursa
Uvod u bioinformatiku
Matematički fakultet

Aleksandra Karadžić, Dragutin Ilić
karadzic.matf@gmail.com, dragutin_ilic@yahoo.com

18. maj 2017.

Sažetak

U ovom tekstu je ukratko prikazana osnovna forma seminarskog rada. Obratite pažnju da je pored ove .pdf datoteke, u prilogu i odgovarajuća .tex datoteka, kao i .bib datoteka korišćena za generisanje literature. Na prvoj strani seminarskog rada su naslov, apstrakt i sadržaj, i to sve mora da stane na prvu stranu! Kako bi Vaš seminarski zadovoljio standarde i očekivanja, koristite uputstva i materijale sa predavanja na temu pisanja seminarskih radova. Ovo je samo šablon koji se odnosi na fizički izgled seminarskog rada (šablon koji *morate* da ispoštujete!) kao i par tehničkih pomoćnih uputstava. Molim Vas da kada budete predavali seminarski rad, imenujete datoteke tako da sadrže temu seminarskog rada, kao i imena i prezimena članova grupe (ili samo temu i prezimena, ukoliko je sa imenima predugačko). Predaja seminarskih radova biće isključivo preko web forme, a NE slanjem mejla.

Sadržaj

1 Uvod

Sekvenciranje genoma, odnosno određivanje rasporeda nukleotida (nt) u genomu, je jedan od fundamentalnih zadataka u bioinformatici. Dužina genoma varira u zavisnosti od organizma (dužina ljudskog genoma je oko 3 milijarde nt, a jedan od najdužih genoma pripada amorfnom jednoćelijskom organizmu *Ameoba dubia* koja je oko 200 puta duži). Najveća prepreka je zapravo činjenica da još uvek nisu razvijene tehnologije koje omogućavaju čitanje nukleotida u genomu od početka do kraja (slično kao čitanje stranica knjige sleva udesno). Trenutno zastupljeno rešenje je sekvenciranje manjih fragmenata DNK koji se nazivaju **ridovi (*eng. reads*)**. Uzima se mali uzorak tkiva ili krvi koji sadrži milione kopija DNK. Biohemijskim procesima se DNK razbija na fragmente, čijim sekvenciranje se dobijaju ridovi. Ne zna se iz kog dela genoma je dobijen određeni rid, pa se koristi tehnika preklapanja ridova da bi se rekontruisao genom. Ovaj ceo proces se naziva i **asembliranje genoma (*eng. genome assembly*)**.

Prvo sekvenciranje genoma je odrađeno 1977. godine od strane Frederika Sangera (*eng. Frederick Sanger*). U Sangerovom metodu dužina ridova je bila između 500 i 1000 ridova. Većina programa zasnovana na ovoj tehnici (*eng. de novo assembly programs*) se bazira na strategiji "preklapanje-raspored-konsenzus"(*eng. overlap-layout-consensus strategy*), u kojoj se preklapanja između ridova izračunavaju brzim tehinikama upoređivanja, raspored kontiga (*eng. contigs*) se generiše pomoću preklapanja u opadajućem rasporedu kvaliteta, a konsenzus sekvenci kontiga se dobija brzim metodama višestrukog poravnavanja. Glavni razlog uspešnosti ovih programa jeste ta da je veličina ridova dovoljna za ustanovljavanje razlika između pravih i lažnih preklapanja.

Prilikom sekvenciranja genoma nailazimo na otežavajuće situacije.Prvo, DNK se sastoji od dve niti, pa ne možemo znati iz koje od njih je rid izveden (nemamo inforamciju da li da koristimo dobijen rid ili njegov obrnuti komplement prilikom asembliranja određene niti u genomu). Drugo, tehnologije koje se koriste nisu savršene, pa dobijeni ridovi često sadrže greške (čime se otežava preklapanje ridova). Treće, neki regioni genoma mogu da ostanu nepokriveni ridovima, čime je onemogućena rekonstrukcija celog genoma. Za otklanjanje ovih problema se koriste razne tehnike kao na primer: pristup razbijanja ridova (*eng. read breaking approach*) se koristi za problem nepokrivenosti genoma ridovima, asembliranje kontiga (neprkidnih delova genoma), a ne čitavih hromozoma, se vrši zbog navedenih problema,tehnika uklanjanja mehurova (*eng. bubble removal*) kojom se rešavaju ridovi sa greškom (lažni ridovi) itd.

Sa ulaskom u drugu genereciju (masovno paralelnih) sekvencijalnih tehnologija, počelo se sa proizvodnjom na milijarde kratkih ridova dužine od 50 do 150 nukleotida. Ogromni skupovi podataka vremenski i prostorno usložnjavaju izračunavanje preklapanja dinamičkim algoritmima. Ridovi male dužine otežavaju otkrivanje tačnih od lažnih preklapanja prilikom generisanja rasporeda kontiga. Kako bi se ovo izbeglo, razvijeni su algoritmi za asembliranje zasnovani na detekciji preklapanja kao tačnih pogodataka fiksne dužine. Na osnovu ovakvih preklapanja formira se *de Bruijin*-ov graf, u kome svaki jedinstevni string dužine k (k-mer) koji se pojavljuje u ridu, predstavlja granu koja spaja čvorove označene sa k-1-mer prefiksom i k-1-mer sufiksom. Pomoću ovog grafa i distribucije frekvencije stringova u ridovima razlikujemo stringove koji imaju grešku od onih koji nemaju. Stringovi sa greškama se ili ne koriste ili se prepravljaju i zatim koriste pri asembliranju.

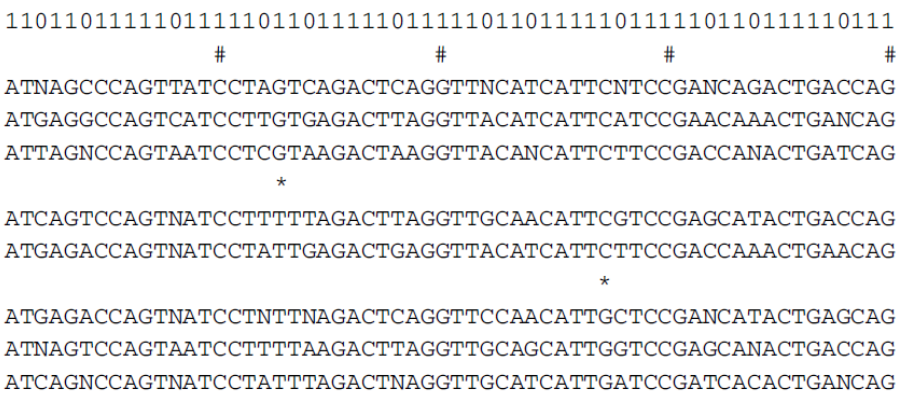
Kako bi koristili sekvenciranje ridova direktno, u ovom radu je predstavljen efikasan metod za izračunjavanje preklapanja za kratke ridove. Metod dozvoljava promašaje u preklapanjima, ali ne dodavanja i brisanja, koji se inače i javljaju ređe od promašaja. U grafu preklapanja, svaki čvor predstavlja rid, dok grana predstavlja preklapanje. U ovom grafu algoritam pronalazi jedinstvenu putanju ridova koja reprezentuje kontigu. Konsenzus sekvence svake kontige je dobijen izračunavanjem poravnanja višestrukih ridova koji nisu razdvojeni nukleotidima koji oni ne sadrže.

U nastavku će biti predstavljni detalji algoritma kao i opis programa PCAP.Solexa u kome je algoritam implementiran.

2 Opis algoritma

Na početku ćemo opisati metod koji se koristi za izračunavanje preklapanja izmedju ridova. Za realizaciju ovog metoda je korišćena struktura podataka koju ćemo zvati **"niz super-reč"** (*eng. superword array*), koja je ime dobila na sličan način kao i druga struktura podataka zvana "sufiksni niz" (*eng. sufix array*). Reč dužine w je string od w karaktera, a super-reč sa v reči je string sa $v*w$ karaktera dobijena konkatenacijom tih reču po redu u kome se javljaju. Reči u super-reči ćemo indeksirati pozicijama od 1 do v. Pozicije u rečima (indeksiranje kreće od jedinice) se dele u dve grupe: čekirane i nečekirane poicije. Za dve reči dužine w kažemo da se podudaraju ukoliko imaju identične nukleotide na svakoj čekiranoj poziciji tih reči. Na primer ako imamo dve reči dužine 15, **ACCATACCATAGCAC** i **ACTATTCCATAACAC**, i ako pretpostavimo da su nečekirane pozicije 3,6 i 12 tada za ove dve reči možemo reći da se podudaraju. Dužina reči w se često namešta da bude 12 ili veća, kako bi broj čekiranih pozicija u reči bio 12, čime se osigurava da lookup tabela za sve stringove ove dužine (ako je reč duža od 12 brišemo sve nukleotide iz nečekiranih pozicija) može stati u glavnu memoriju. Broj reči v se bira tako da super-reč dužine $v*w$ bude manja od dužine svakog rida (dužinu rida ćemo označavati sa r). Dužina super-reči se još naziva i minimalna dužina preklapanja. Adekvatan primer za ridove dužine 150 bi bio da postavimo da w na 15 i v na 6 čime dobjamo super-reči dužine 90. Rid dužine r ima r - $w*v$ + 1 super-reči na pozicijama 1,2, ...,r - $w*v$ + 1. U našem primeru bi rid dužine 150 imao 61 super-reč na pozicijama 1,2, ...,61. Za dve super-reči kažemo da su identične ako im se sve reči podudaraju, odnosno svi nukleotidi na čekiranim pozcijama su im identični. Za jednu super-reč kažemo da je manja (veća) od druge ako string nukleotida na svakoj čekiranoj poziciji prve super-reči, u leksikografskom poredku, dolazi pre (posle) stringa nukleotida na čekiranim pozicijama druge super-reči. Svaki rid dobija jedinstveni nenegativni ceo broj koji se zove indeks rida. Pomoću njega i početne pozicije super-reči u ridu se izračunava jedinstveni indeks svake super-reči u svakom ridu. U algoritmu se koristi i funkcija koja pomoću datog indeksa super-reči efikasno računa poziciju te super-reči u indeksu i indeks rida.Sada imamo sve podatke kako bi definisali "niz super-reč" strukturu podataka. **"Niz super-reč"** za skup ridova je niz svih indeksa super-reči gde su super-reči sortirane od manje ka većoj.

Na slici 2 prikazano je osam super-reči u sortiranom poredku. Svaka super-reč se sastoji od četiri reči (v=4) dužine w = 15, gde je svaka čekirana pozicija označena bitom 1 , a nečekirana 0. Poslednja pozicija svake reči je označena taraba znakom. Super-reči su grupisane tako



Slika 1: Prikaz grupisanja i sortiranja super-reči

da se u svakom bloku nalaze identične super-reči (prve tri, sledeće dve i poslednje tri). Vidimo da nukleotidi na nečekiranim pozicijama mogu biti ili različiti ili nedefinisani (označeni sa **N**) u istom bloku. Pvi blok dolazi pre drugog zato što na poziciji označenoj sa zvezdicom se u prvom bloku nalazi **G** nukleotid a u drugom **T** (**G** leksikografski dolazi pre **T**). Slično zvezdica izmedju drugog i trećeg bloka označava zašto drugi dolazi pre trećeg.

2.1 Konstrukcija niza super-reč

Niz super-reč se konstriše pomoću bucket sorta u v krugova sortiranja (podsetimo se da se svaka super-reč sastoji od v reči od kojih svaka ima tačno 12 čekiranih pozicija). Najpre se niz inicializuje indeksima super-reči u rastućem poretku po vrednostima indeksa. Zatim se iz svake super-reči uzima reč na poziciji p gde p ide od v do 1 (krećemo se sdesna u levo u svakoj super-reči) i niz sortiramo po leksikografskom poretku tih reči. Kako idemo sdesna u levo nakon v koraka bucket sort garantuje da će "niz super-reč" biti sortiran. Za sortiranje se koriste dve pomocne strukture podataka, lookup tabela i niz lokacija , koji služe kao kofe (*eng. bucket*) u koje smeštamo reči. Niz lokacija je niz celih brojeva koji je indeksiran od 0 do najvećeg indeksa super-reči. Lookup tabela se inicijalizuje negativnim brojevima što označava praznu kofu, a ima onoliko polja koliko ima kodova za sve stringove dužine 12. Bucket sortiranje se obavlja tako što se elementi "niza super-reč" sleva na desno čitaju ,smeštaju u kofe i zatim se kopiraju iz kofa nazad u "niz super-reč" sdesna u levo. Detaljnije, trenutni element, indeks super-reč,i se smešta u kofu iz "niza super-reč" na sledeći način. Indeks super-reči se koristi kako bi se našla reč na poziciji p. Zatim tu reč koja je dužine w transformišemo u string dužine 12 korišćenjem bitovskih operacija kako bi uklonili nukleotide na nečekiranim pozicijama. Znamo da postoji indeks u lookup tabeli koji predstavlja kod tog stringa dužine 12, pa vrednost lookup tabele na tom indeksu čuvamo u nizu lokacija na poziciji koja ima isti indeks kao i indeks super-reči, a u lookup tabelu upisujemo sam indeks super-reči. Kada smo sve elemente "niza super-reč" ubacili u kofe, čitamo ih počevši od najveće

kofe i kopiramo ih u "niz super-reč" sdesna ulevo.

2.2 Izračunavanje preklapanja koristeći niz super-reč

Nakon konstrukcije "niza super-reč" delimo ga u sekcije tako da se u svakoj sekciji nalaze identične super-reči. Ove sekcije se obradjuju kako bi se generisala preklapanja izmedju ridova. To radimo na sledeći način. Posmatraćemo trenutnu sekciju koja ima s super-reči. Svaka dva rida koja imaju super-reč u ovoj sekciji mogu da se poklapaju. Tako da imamao $(s * (s-1))/2$ potencijalnih preklapanja u ovoj sekciji. Kako bi uprostili problem mi nećemo računati sva ova preklapanja, već ćemo računati njegov podskup (koji ćemo dobiti za linearno vreme), a ostala preklapanja će moći da budu zaključena iz ovog podskupa. Ovo se može uraditi , tako što sortiramo super-reči u sekciji po dužini njihovih prefiksa u ridu i selektovanjem susednih parova super-reči kako bi izračunali preklapanja. Ako super-reči pripadaju različitim ridovima (podsetimo se da ovo možemo proveriti koristeći funkciju koja od indeksa super-reči računa poziciju super-reči u ridu i indeks rida kome ta super-reč pripada) onda računamo preklapanje izmedju njih. Ukoliko je broj nukleotida koji se razlikuju u preklapanju ograničen odsecanjem, preklapanje čuvamo.

2.3 Ubrzanje paralelizacijom

Veliki skupovi ridova se dele u manje podskupove (obeleženih brojevima od 0 do j-1). Svaki rid iz istog podskupa ima super-reč kod koje ostatak pri deljenju koda reči na poziciji p (p je izmedju 1 i v) sa j daje oznaku tog podskupa. Naglasimo da rid može pripadati u više podskupova. Ovi podskupovi se dodeljuju j procesorima kako bi se paralelno izračunala preklapanja. Svaki procesor računa preklapanja u podskupu konstruišući niz super-reč za svaki podskup. Ovim postupkom dobijamo različite fajlove preklapanja za svaki podskup. Ovi fajlovi se potom skeniraju kako bi se uklonila redudantna preklapanja. Na taj način dobijamo skup fajlova neredudantnih preklapanja.

2.4 Konstrukcija grafa preklapanja

Fajlovi preklapanja se čitaju redom jadan po jedan i svako preklapanje sa dobrim procentom preklapanja (broj nukleotida koji su se razlikovali u preklapanju je manji od granice) čuvamo u glavnoj memoriji. Graf preklapanja se konstruiše da čvorovi budu ridova, a grane izmedju čvorova su preklapanja izmedju dva rida koju povezuje ta grana. Graf se potom ispituje kako bi se našle dugačke putanje preklapanja neponavljajućih čvorova. Čvor je ponavljajući ako postoje dve dugačke putanje koje se završavaju u čvoru, ali nemaju ponavljanja izmedju njihovih prefiksnih putanja. Ovo se izvodi na principu dijekstrinog algoritma. Svaka dugačka putanja neponavljajućih čvorova se predstavlja kao kontiga ridova. Stvaranje rasporeda svake kontige se obavlja na jednom procesoru sa velikom količinom glavne memorije. Na kraju ovog koraka se dobijaju fajlovi koji predstavljaju raspored kontiga.

2.5 Generisanje fajla konsenzus sekvenci kontiga i ocene kvaliteta

Fajlovi rasporeda kontiga se paralelno procesiraju. Ridovi se rasporedjuju kako bi formirali višestruka poravnanja ridova. Ova poravnanja se koriste za generisanje konsenzus sekvenci za kontige. Za svaki fajl rasporeda kontiga se generiše fajl konsenzus sekvenci kontiga kao i fajl ocene kvaliteta baza konsenzus kontiga. Na osnovu ovih fajlova se generiše jedan fajl konsenzus sekvenci kontiga i jedan fajl ocene kvaliteta baze konsenzus kontiga.

3 Korišćenje programa PCAP.Solexa

Algoritam opisan u prethodnom poglavlju je implementiran u programu PCAP.Solexa. U ovom ovom poglavlju će biti opisan način rada, kao i prateće komande koje je potrebno znati za ovaj program.
PCAP.Solexa se pokreće u komandnoj liniji i može se pokrenuti na 64-bitnim Linux/Unix/MacOSX sistemima. Program je dosta memorijski zahteva, pa za asembliranje genoma gljivice je potrebno 100Gb glavne memorije i 500Gb memorije hard diska. Takođe, potrebno je napomenuti da program nije poželjno pokretati za asembliranje genoma velikih biljaka i životinja.

Nakon skidanja direktorijuma PCAP.Solexa, koji je dostupan na `http://seq.cs.iastate.edu`, može se naći više izvršnih fajlova. Ovi izvršni fajlovi se pokreću po odgovarajućem poretku, koji je zadat u skripti PCAP.Solexa.perl. Ova skripta pre samog pokretanja mora da se modifikuje kako bi stavili do znanja lokaciju PCAP.Solexa direktorijuma i postavili odgovarajuće parametre. U promenljivoj $CodeDirPath je potrebno upisati apsolutnu putanju do direktorijuma, kako program mogao da se izvršava. Na primer:

```
$CodeDirPath = "/home/aleksandra/Downloads/pcap.solexa"
```

Ostali parametri su postavljene na podrazumevane vrednosti, opis parametara i vrednosti su dati kao komentari u skripti. U programu je dat jedan primer u poddirektorijumu zvanom test, koji će se koristiti za demonstraciju rada PCAP.Solex programa. On sadrži dva fajla **s_4_1_sequence.fastq** i **s_4_2_sequence.fastq** u kojima se nalazi po 7572 rida gde svaki sadrži po 151 nukleotida dobijeni Ilumina sistemom. Poddirektorijum takokđe sadrži odgovarajuće fofn, fofn.con i fofn.lib fajlove. Fajl fofn sadrži imena fajlova iz kojih se čita s_4_1_sequence.fastq s_4_2_sequence.fastq, svaki u posebnom redu. Fofn.con fajl ima sledeći sadržaj:

```
s_4_1_sequence.fastq s_4_2_sequence.fastq 100 700 tmp clone
```

Prva dva argumenti označavaju fajlove iz kojih se čitaju ridovi. Treći i četvrti označavaju donju i gornju granicu nukleotida koje ćemo unositi, argument clone predstavlja ime biblioteku koja će se koristiti, dok je parametar tmp čuvar mesta (*eng. placeholder*). Sadržaj fajla fofn.lib je:

```
clone 300 50
```

Drugi i treći argument predstavljaju matematičko očekivanje i standardnu devijaciju za biblioteku koja se koristi i koja se nalazi u clone argumentu. Pokretanje programa je moguće na dva načina. Prvi način je komandom

```
pcap.solexa.perl fofn &
```

čime smo u pozadini pokrenuli rad datotteka s_4_1_sequence.fastq i s_4_2_sequence.fastq. Drugi način je pokretanje skripte *cs551_test_script*, koja automatski smešta u direktorijum PCAP.Solexa programa i pokreće predhodno navedenu komandu. Pre pokretanja ove skripte je potrebno promeniti apsolutnu putanju do programa PCAP.Solexa. Nakon uspešnog izvršavanja programa, statistike i rezultati asembliranja se nalaze u sledećim fajlovima: *contigs.bases*, *contigs.quals*, *reads.placed*, *fofn.pcap.scaffold*.ace*, *readpairs.contigs*, *fofn.con.pcap.results*, *z.n50* i *fofn.pcap.contigs*.snp*. Fajl contigs.bases sadrži sekvence u fasta formatu od 4 kontiga: Contig0.1, Contig1.1, Contig2.1 i Contig3.1. Brojevi u nazivu kontiga označavaju koja je kontiga u kom skefoldu, recimo Congtig0.1 je prva kontiga u nultom skelfodu. Fajl contigs.quals sadrži Pred ocene kvaliteta sekvenci (*Phred quality score sequence*) u fasta formatu za svaku kontigu. Pred ocena kvaliteta je mera identifikacije nukleobaze generisanje automatskim DNK sekvenciranje. Dužine kontiga su date u fajlu readpairs.contigs. Sadržaj ovog fajla je sledeći:

```
C0.1 1 23144
C1.1 1 491
C2.1 1 210
C3.1 1 200
```

Prva kolona je skraćeno ime kontiga, druga označava orijentaciju kontiga, a treća dužinu. Dužina kontiga N50 predstavlja za skup kontiga je definisan kao najveći broj L takav da kontigi dužine barem L imaju ukupn sumu veću od polovine sume svih dužina kontiga. Broj kontiga N50 za skup kontiga je definisan kao najmanji broj N takav da N-ti najduži kontig ima ukupnu dužinu veću od polovine sume dužina svih kontiga. Dužina i broj najvećeg kontiga N50 su slično definisani uključivanjem samo kontiga čija je dužina najmanje 1Kb. Statistike kontiga N50 se nalaze u fajlu z.n50. U fajlu z.n50 koji je dobijen pokretanjem primera se nalaze statistike za N50 kontig.

```
Ctg N50 length: 23134, Ctg N50 number: 1
Major ctg N50 length: 23134, Major ctg N50 number: 1
```

U fajlu fofn.con.pcap.results se nalaze informacije o svakom paru ridova da li se nalazi u skafildu ili da li je kontig u odgovarajućoj orijentaciji, kao i veličina ulaza u zadatom rangu, svaki par u jednoj liniji. Svaka linija ima barem 6 kolona. Prva i druga kolona predstavljaju indekse dva rida u paru, u ovom primeru indeksi ridova iz fajla s_4_1_sequence.fastq idu od 0 do 7571, a ridovi iz fajla s_4_2_sequence.fastq od 7572 do 15413. Treća i četvrta kolona označavaju donju i gornju granicu ulaza. Kolona pet je čuvar prostora. Šesta kolona je ključna reč koja označava status para rida: ključna reč šingltonžnači da barem jedan od ridova u paru nije uključen u asembliranju, "redundantan"označava da je par isključen jer je bio sličan nekom paru u orijentaciji i rasporedu ridova u asembliranju, "kratak"označava da je jedan od ridova u paru uključen u skafoldu ili je pri kraju skafolda. Broj koji se nalazi u šestoj koloni označava da su dva rida uključeni u skalfoldu ili u kontigu u suprotnoj orijenticiji sa broj koji indikuje razdaljinu između ridova. Ako je razdvojenost u rangu ulaznih podataka, onda se u sedmoj koloni nalazi opis ridova žadovoljen u kontigu"ili žadovoljen u skafoldu". Na kraju fajla se nalaze brojeve koji označavaju koliko je ridova u svakoj kategoriji pročitano. U fajlovima fofn.pcap.contigs*.snp se nalaze izveštaji o kandidatima za jedan nukleotid polimorfizma (*single nucleotide polymorphisms (SNP)*

4 Drugi naslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

5 n-ti naslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

5.1 ... podnaslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

5.2 ... podnaslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

6 Poslednji naslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

7 Zaključak

Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak.

A Dodatak

Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe.