SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Projekt iz predmeta Bioinformatika, 2017/2018

Računanje udaljenosti uređivanja algoritmom 4 Russians

Marko Čepo, Tomislav Lokotar
Voditelj: *Mile Šikić*

Zagreb, siječanj, 2018

Sadržaj

1.Uvod

Metoda Four Russians (hrv. Četiri Rusa) je metoda ubrzavanja algoritama koji koriste dinamičku matrice za izračun razlika između dva tekstualna niza. Four Russians metoda prvi puta se spominje u radu Arlazarova, Dinica, Kronroda, Faradzeva 1974. godine [1], a pretpostavlja se kako je metoda dobila ime po nacionalnoj pripadnosti četiriju autora. U ovom radu, primijenili smo spomenutu metodu na problem poravnavanja sekvenci. Poravnavanje sekvenci je postupak pronalaženja jednakih regija dvaju ulaznih nizova, obično nukleotidnih baza. Poravnavanje sekvenci ima široku primjenu ima u bioinformatici, računalnoj obradi prirodnoga jezika, kao i ostalim interdisciplinarnim znanostima koje uključuju računarstvo i tekstualnu reprezentaciju problema.

2. Four Russians metoda

2.1 Općenito o metodi

Općenito, Four Russians metoda može se primijeniti na veće matrice čiji elementi poprimaju konačan broj vrijednosti. Ideja je manje blokove (podmatrice) unaprijed izračunati i pohraniti u kolekciju za koju pristup elementima ne zahtijeva mnogo računalnih resursa, vremenski idealno O(1). Često korištena metoda pohranjivanja je raspršenim adresiranjem (engl. Hashing, HashMap, Dictionary). Kako je broj mogućih vrijednosti elemenata matrice konačan, samim time je konačan i ukupan broj različitih blokova. Veličinu bloka potrebno je odrediti prije samog kalkuliranja svih mogućih blokova. Vrijeme potrebno za kalkuliranje blokova ovisi najviše o dimenzijama bloka. Također, prostor potreban za pohranu kalkuliranih blokova, u slučaju 2D matrica, raste s kvadratom dimenzije bloka, i s potencijom broja mogućih vrijednosti matrice.

U slučaju 3D matrica ili višedimenzijskih matrica, broj kombinacija u odnosu na veličinu bloka koji se pohranjuju dolazi još više do izražaja. Vrijeme izvođenja algoritama koji koriste navedene matrice također ovise o veličini bloka. Većom dimenzijom bloka postiže se veće ubrzanje, ako ne računamo vrijeme potrebno za generiranje svih mogućih kombinacija. Ukratko, veličina bloka negativno utječe na memorijske zahtjeve, a pozitivno na vrijeme izvođenja. Samim time, metoda uvodi kompromis između dostupnih resursa, pa je potrebno prilagoditi parametre algoritma računalnom sustavu na kojem se izvodi, a odnosi se ponajprije na parametar veličine bloka, u nastavku označen kao T.

2.2 Udaljenost uređivanja

Udaljenost uređivanja je minimalan broj operacija potrebnih da bi se jedna sekvenca poistovjetila s drugom, odnosno da bi se obje sekvence bile jednake. Udaljenost uređivanja je dio postupka koji nazivamo poravnavanje sekvenci.

Često korišten algoritam u svrhu poravnavanje sekvenci je onaj predstavljen u radu Needleman&Wunsch 1970. godine. Za provedbu algoritma, Needleman i Wunsch koriste metodu dinamičkog programiranja, konstruiranjem tablice (matrice) koju nazivamo tablicom dinamičkog programiranja. Primjer provedbe takvog algoritma ilustriran je na jednostavnom primjeru, slika 1. Rezultat (broj operacija potrebnih za poravnavanje dviju sekvenci) provedbe algoritma nalazi se u donjem desnom elementu tablice dinamičkog programiranja.

		A	C	G	T

	0	1	2	3	4
A	1	0	1	2	3
C	2	1	0	1	2
G	3	2	1	0	1

Slika 1, provedba poravnavanja dviju jednostavnih sekvenci,
potrebna je jedna operacija

Iz primjera je lako zaključiti kako vrijednosti pojedinih ćelija tablice dinamičkog programiranja ovise o dužini ulaznih stringova. Samim time, broj vrijednosti nije konačan, ako se ne želimo ograničiti na duljinu sekvence.

Dung [3] je pojednostavio matricu tako da su vrijednosti matrice popunjavali relativnim odmacima vrijednosti u odnosu na ćeliju prije (gornju i ljeviju). Primjer matrice dinamičkog programiranja kodirane pomacima prikazan je na slici 2, iz koje je lako vidjeti odnos klasične matrice korištene algoritmom NW i verzije prikladne za ubrzanje postupkom Four Russians.

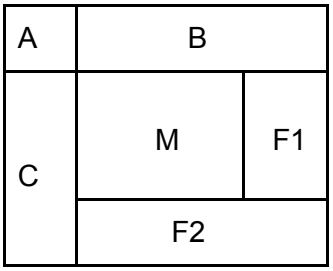
		A	C	G
	0	1	2	3
A	1	0	1	2
C	2	1	0	1
G	3	2	1	0

		A	C	G
	0	+1	+1	+1
A	+1	m	m	-1
C	+1	m	m	-1
G	-1	-1	-1	-1

Slika 2: lijevo: tablica dinamičkog programiranja, desno: tablica kodirana posmacima

2.3 Blokovi Four Russians metode

Blokovi koji se kalkuliraju podijeljeni su u 7 dijelova, kao na slici 3. X i Y su dijelovi sekvenci koje poravnavamo, B i C su vrijednosti poravnavanja za blok iznad, odnosno lijevo. Rezultat dobiven kalkuliranjem je pohranjen kao dva vektora, F1 i F2, u kojima se nalaze rezultati poravnavanja za trenutni blok. Interni dio bloka M računa se standardnim NW postupkom, uz to da se prethodne vrijednosti kodirane pomakom (pozitivan, nula i negativan) kodiraju na standardan način, a izlazne vrijednosti se ponovo kodiraju kao pomaci.



slika 3: blok

Blokovi se u memoriju pohranjuju kao uređeni par: vektor F1 i vektor F2. Interno izračunate vrijednosti, na slici prikazane kao M, se nepovratno gube, jer nisu potrebne za izračunavanje poravnavanja. Struktura (kolekcija) u koju se pohranjuju blokovi je običan niz.

Uobičajena struktura za pohranu blokova bila bi Hash mapa, ali kako je funkcija preslikavanja blokova bijektivna, odnosno funkcija koja ne producira kolizije između dvaju različitih blokova. Funkcija preslikavanja blokova, odnosno informacija B, C, X i Y, koje su dovoljne za preslikavanje opisane su sljedećom formulom:

blockIndex = append(B, C, X, Y)

Faktor T je dimenzija bloka koji se pohranjuje u memoriju. U radu su isključivo razmatrani kvadratni blokovi.

2.4 Predkalkuliranje blokova

Postupak kalkuliranja svih mogućih blokova je prva stvar koju implementacija, ostvarena na ovom projektu, radi. To radi tako da se za sve kombinacije ulaznih parametara koji jedinstveno definiraju blok, izračunaju vektori F1 i F2, te pohrane u niz pod indeksom izračunatim kao što je prikazano na formuli 1.

2.5 Izračun udaljenosti uređivanja

Minimalan broj operacija za poravnavanje dvije sekvence, ili općenitije između dvaju nizova znakova, nazivamo udaljenošću uređivanja. Udaljenost uređivanja još se naziva i Levenshteinova udaljenost[4]. funkcija bodovanja prikazana je sljedećom funkcijom:

$$(1)\ f(baza1,\ baza2) = \{0\ ako\ baza1 == baza2,\ 1\ inače\}$$

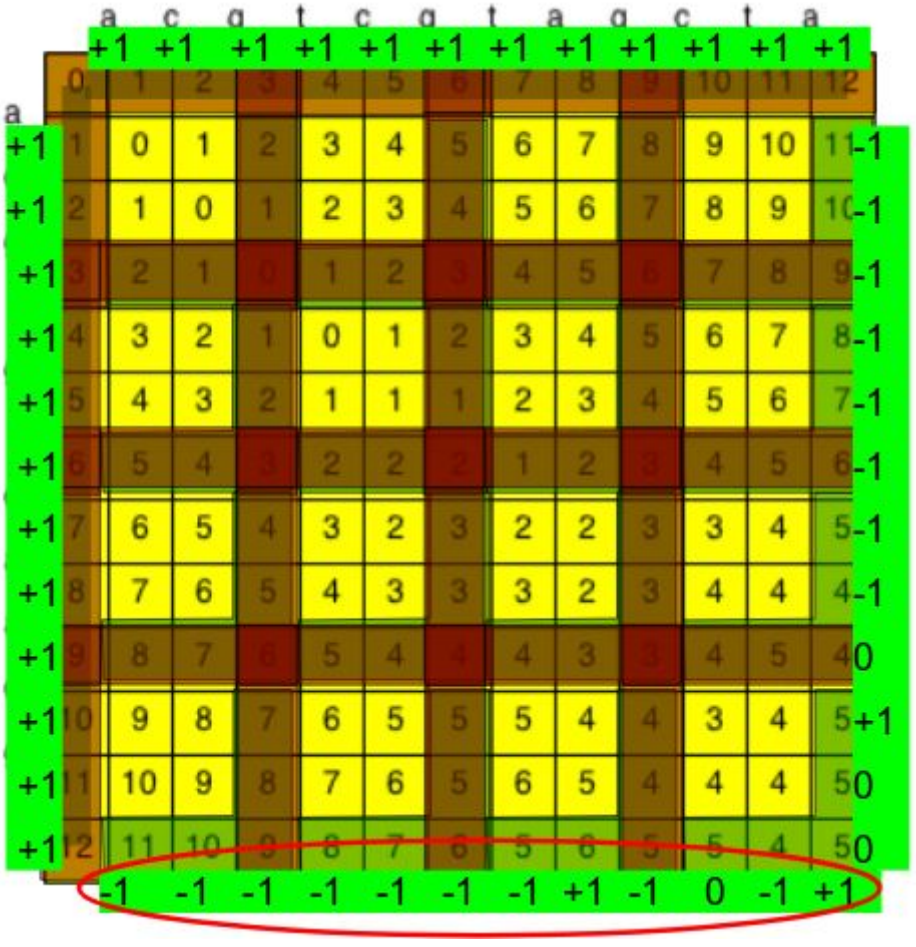
Ako je udaljenost uređivanja dvaju nizova jednaka nuli, to znači da su nizovi jednaki. Općenito se u bioinformatici koriste i drugi sustavi bodovanja operacija potrebnih za

poravnavanje, koji mogu rezultirati pronalaskom drugih regija poklapanja. Primjer jedne takve funkcije je:

$$(2)\ f(baza1,\ baza2) = \{+1\ ako\ baza1 == baza2,\ +2\ inače\}$$

Uporište takvog sustava jest u tome da ne mora biti jednaka vjerojatnost prelaska jedne baze u drugu, kao nepostojanja ili brisanja.

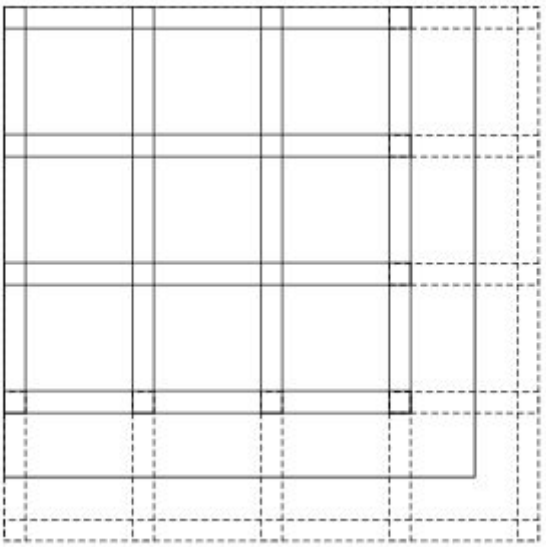
Postupak izračuna udaljenosti pomoću blokova ilustriran je na slici 4. Za prvi redak i stupac se inicijaliziraju vrijednosti X, Y, B i C. Potom, postupak iterativno, počevši od gornjeg lijevog prema desno, računa koji blok mu je potreban. Informacija koja mu je za to potrebna jest prethodno izračunati blok. Po završetku retka blokova, prelazi se na sljedeći redak, za koji je potreban prethodni redak, i tako sve do krajnje desnog donjeg bloka. Izračun udaljenosti potom se vrši zbrajanjem prvog stupca i zadnjeg retka.



slika 4: postupak izračunavanja udaljenosti uređivanja, slika iz javno dostupne prezentacije[8] na http://cs.au.dk

Izračun udaljenosti uređivanja kad duljina sekvenci nije dijeljiva s dimenzijom bloka radi se nadopunjavanjem sekvenci znakovima, samo da bi se ona proširila do djeljivosti. Po izračunu udaljnosti, uzima se u obzir koliki je broj znakova u kojoj sekvenci dodan, te se

ponovno izračunavaju vrijednosti zadnjeg bloka u kojemu se nalazi rješenje. Postupak je prikazan na slici 5.



slika 5: Iz rada Dung My Hoa, prikaz računanja udaljenosti s popunjavanjem do dijeljivosti s veličinom bloka

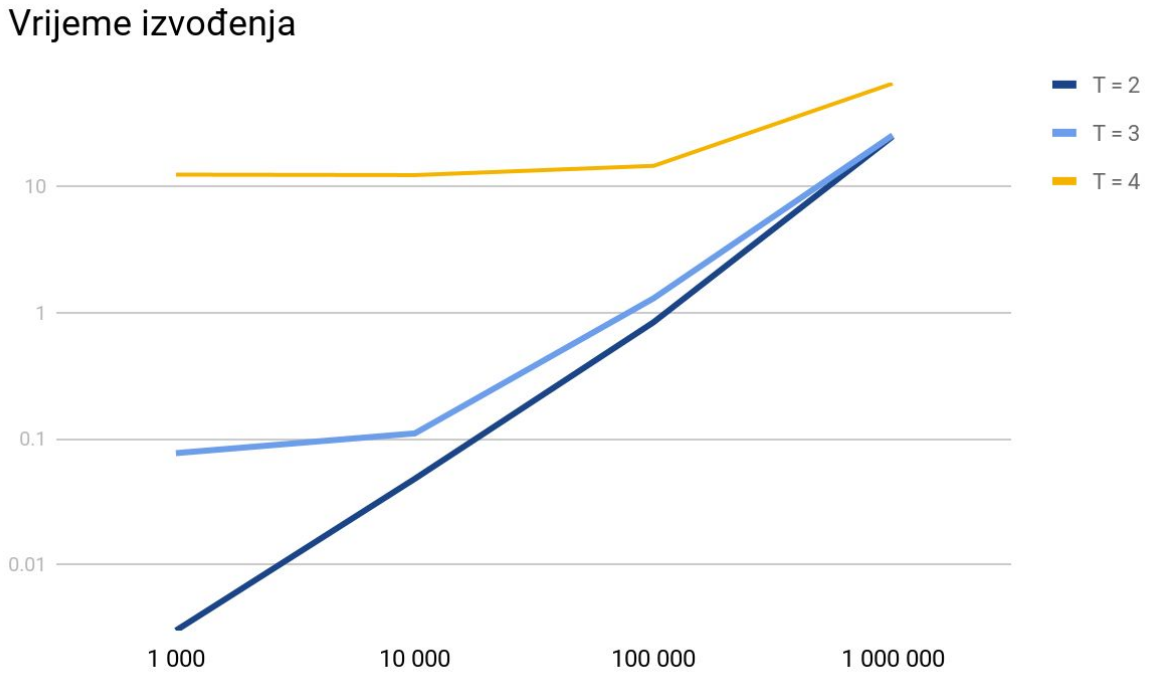
2.6 Izračun puta uređivanja

Izračun puta uređivanja vrši se na način analogan onome kako su ga opisali NW. Postupak se sastoji od pronalaska puta od krajnjeg desnog donjeg elementa (rezultata) do početka, pritom zapisujući put poravnavanja. Kako je za ovakav način potrebno cijelu tablicu dinamičkog programiranja držati u memoriji, a ne samo jedan redak blokova koji je potreban za izračun udaljenosti uređivanja, memorijski zahtjevi višestruko rastu. Blokove nije potrebno držati u punom obliku, već je u memoriji potrebno držati tablicu indeksa blokova. Blokovi se potom ponovno evaluiraju, odnosno izračunavaju se njihove vrijednosti M. Iako su ti blokovi u nekom trenutku već bili izračunati te njihove vrijednosti M obrisane, utjecaj na performanse je zanemariv kako će se ponovo izračunavati broj blokova u linearnoj zavisnosti od duljina sekvenci, dok sam postupak izračunavanja tablice dinamičkog programiranja ovisi o kvadratu duljina sekvenci.

$$(3)\ O(n + m) << O(n * m/(T * T))$$

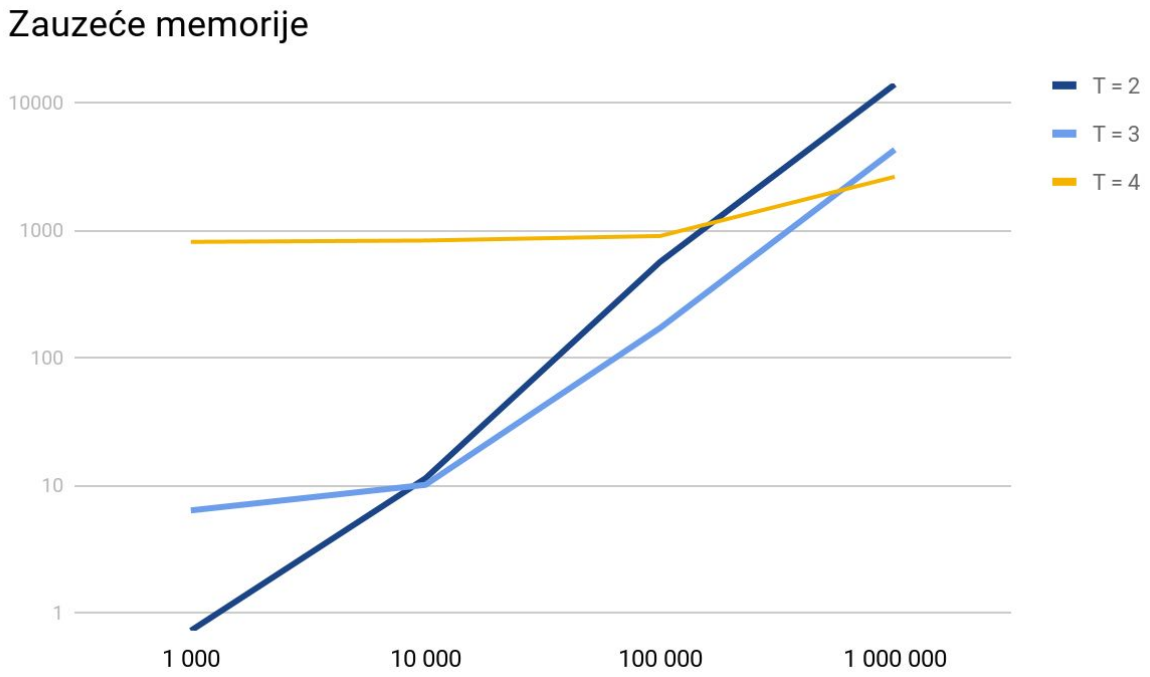
3. Performanse i složenost

3.1 Performanse



Graf 1. Vrijeme izvođenja[sec] u odnosu na veličinu bloka i duljinu ulaznih nizova

Graf 1 prikazuje vrijeme izvođenja algoritma u odnosu na veličinu ulaznih nizova i veličinu blokova. Iz grafa možemo vidjeti da što su manji ulazni nizovi potrebno je koristiti manje veličine blokova kako bi se dobile bolje performanse. Razlog tomu je prvenstveno vrijeme potrebno za kalkuliranje blokova. Kako veličina ulaznog niza raste tako veći blokovi dolaze do izražaja i daju bolje performanse. Isto tako možemo vidjeti kako vrijeme izvođenja raste eksponencijalno sa ulaznim nizovima. Razlog tomu je što je vremenska složenost *O(m*n/T*T)*. Pri čemu su m i n dužine ulaznih nizova.



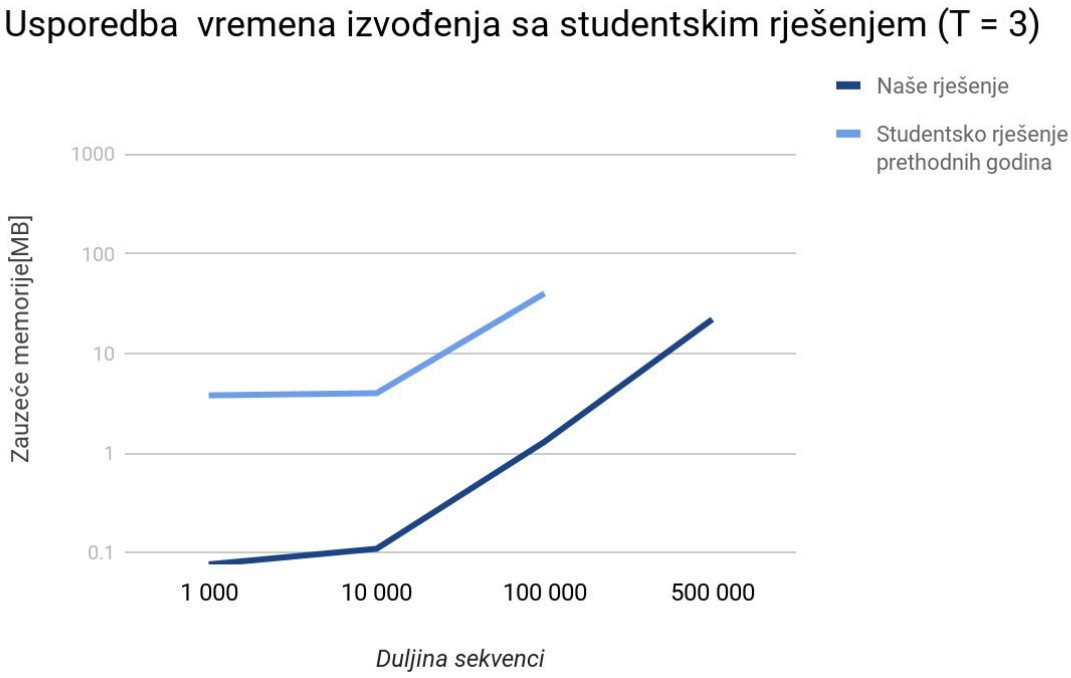
Graf 2. Zauzeće memorije[MB] prilikom izvođenja u odnosu na veličinu bloka i duljinu ulaznih nizova

Graf 2 prikazuje zauzeće memorije prilikom izvođenja algoritma. Količina zauzete memorije se sporije povećava za veći T. Razlog tomu je što matrica izračuna manje za veći T, jer se dužina stringa dijeli u manje blokova. Isto tako možemo vidjeti da iako veći T sporije raste, početna vrijednost zauzete memorije je veća. Razlog tomu je što se treba pohraniti više kalkuliranih blokova. Kad sve to uzmemo u obzir dolazimo do zaključka kao i kod vremenske složenosti. Bolje je koristiti veći T za veće dužine ulaznih nizova, a manji T za manje nizove. Formula za izračun optimalne veličine bloka u odnosu na ulazni niz dana je u nastavku.

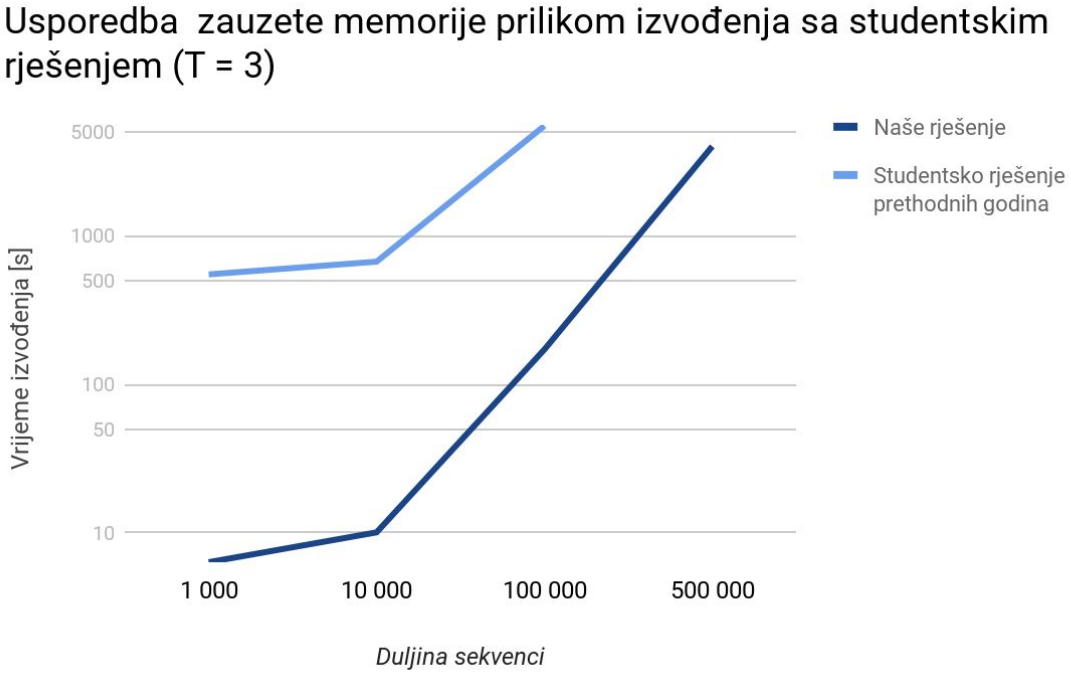
$$(4)\ T_{opt} = \frac{log(3*|\Sigma|n)}{2}$$

Sam program je implementiran tako da ako se za veličinu bloka stavi vrijednost 0. Program će koristiti optimalnu veličinu bloka.

3.2 Usporedba



Graf 3.



Graf 4.

4. Budući rad na projektu

4.1 Izračun puta poravnavanja u linearnom prostoru

Jedna od mogućih nadogradnji na izgrađeno programsko rješenje je implementacija računanja puta poravnavanja kako je to predložio Hirschberg [5]. Spomenuti postupak omogućava računanje puta poravnavanja s prostornom složenosti O(m). Dung je u svome radu uspješno implementirao Hirschberg metodu za pronalazak puta poravnavanja uz korištenje FourRussians postupka ubrzavanja.

4.2 Pohrana blokova u trajnu memoriju

Kako svaki put prilikom pokretanja, trenutna implementacija iznova računa sve moguće blokove, nameće se pitanje pohranjivanja niza blokova u trajnu memoriju, te učitavanje iz trajne memorije prilikom pokretanja. Pohrana nizova u memoriju za veličine bloka T=1 do 4 zauzima oko 900MB diskovnog prostora. Moderni SSD sustavi za trajnu pohranu podatka su u mogućnosti u vrlo kratkom vremenu učitati sadržaj ovakve veličine u radnu memoriju. Ovakav sustav bi, posljedično, bio vrlo osjetljiv i na operacije čitanja iz trajne memorije. Kao moguće rješenje predlažemo tri metode, koje ne isključuju jedna drugu:

1) Paralelno učitavanje podataka u memoriju i generiranje svih blokova. Ovako bi se donekle doskočilo nekonzistentnošću, kako bi se uvijek uzeo rezultat one dretve koja je prije gotova.

2) Korištenjem kompresije podataka moguće je pohraniti generirane nizove u diskovni prostor kao arhivu, iz koje bi se izravno raspakiralo u radnu memoriju. Moderna programska rješenja[6] zahtijevaju vrlo malo resursa da se datoteka otpakira, daleko manje od generiranja svih blokova. Pokusno arhiviranje navedene datoteke od 900MB rezultiralo je arhivom veličine između 2MB i 10MB, ovisno o razini kompresije. Iz toga je ujedno lako zaključit kako podaci sadrže redundantne informacije.

3) Smanjenje prostora pretraživanja blokova, tako da se dodatno optimizira funkcija preslikavanja blokova u indeks niza, . Pokusom je uspješno reducirana veličina niza za nešto više od 10 puta. Samim time je i broj operacija za izračun indeksa nešto veći, pa se opet nameće pitanje kompromisa između procesorske zahtjevnosti i utroška memorije.

4.3 Paralelizacija

Za velike sekvence, postavlja se pitanje paralelizacije generiranja redaka s rezultatima, kao i sa samim kalkuliranjem. Na ovom projektu je tek šturo proučena mogućnost paralelizacije, ali je bilo dovoljno da se pokaže korisnim. Potrebno bi bilo napraviti i procjenu treba li za trenutnu sekvencu raditi paralelizaciju, kako sam proces pokretanja nove dretve iziskuje nešto vremena.

5. Zaključak

Po završetku projekta, otvarala su se mnoga pitanja nadogradnje i prostora za napredak. Mnogo puta su se postavljala pitanja kompromisa između resursa, urednosti, nadogradivosti, i vremena izvođenja. Uvid u problematiku je vrlo koristan bilo kojem inženjeru na području računalnih znanosti, kako lijepo prikazuje odluke koje je potrebno donijeti u situacijama gdje je vrijeme izvođenja vrlo bitno, računalni resursi ograničeni, a primjena mnogo. Izvorni kod implementacije javno je dostupan uz MIT licencu. Kroz dva mjeseca implementacije, autori ovog projekta naučili su mnogo u implementacijama algoritama na niskoj razini, kao i o utjecaju detalja na samo rješenje.

7. Literatura

[1] Arlazarov, V;Dinic, E.; Kronrod, M.; Faradzev, I., "On economical construction of the transitive closure of a directed graph", 1970. *Dokl. Akad. Nauk SSSR*

[2] Needleman, Saul B. & Wunsch, Christian D., "A general method applicable to the search for similarities in the amino acid sequence of two proteins", 1970

[3] My Hoa D., "Speeding-up dynamic programming in sequence alignment", 2010.

[4] Levenshtein, V. I., "Binary Codes Capable of Correcting Deletions, Insertions and Reversals", 1966. *Soviet Physics Doklady*

[5] Daniel S. Hirschberg. “A linear space algorithm for computing maximal common subsequences. Commun.” ACM, 18(6):341–343, 1975.1

[6] https://github.com/sebastiandev/zipper, pristupano 17. Siječnja 2018.

[7] https://github.com/nuzelac/Bioinformatika-4Russians, pristupano 17. Siječnja 2018.

[8] Speeding up dynamic programming The Four Russians, http://cs.au.dk/~cstorm/courses/AiBS_e12/slides/FourRussians.pdf

8. Sažetak

U ovom radu izučuje se ubrzanje postupka poravnavanja sekvenci korištenjem tablice dinamičkog programiranja pomoću metode Four Russians. Four Russians metoda, općenito, ubrzava algoritme koji koriste velike matrice, na način da dijelove matrica (blokove) pohranjuje u memoriju i dohvaća po potrebi. Napredak u odnosu na studentska rješenjima prethodnih godina dovoljno govori o koliko parametara ovise performanse postupka. U poglavlju 2 ukratko se pojašnjavaju detalji postupka. U poglavlju 3 prikazani su rezultati i usporedba sa studentskim rješenjem prethodnih godina. U poglavlju 4 predložene su buduće nadogradnje na programsko rješenje. Izvorni kod ovog projekta javno je dostupan na adresi https://github.com/mcepo/bioinfo-project pod MIT licencom.