SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKT IZ KOLEGIJA BIOINFORMATIKA

Four Russians algorithm

Domagoj Magaš, Nino Uzelac, Marko Sertić

Voditelj: *Mile Šikić*

Zagreb, siječanj, 2016.

Sadržaj

1. Uvod

Udaljenost uređivanja između dva niza znakova možemo definirati kao minimalan trošak slijeda operacija uređivanja koje pretvaraju jedan niz znakova u drugi. Operacije koje sačinjavaju slijed su: brisanje znakova, umetanje znakova, i zamjena znakova. Moguće je da svaka od navedenih operacija ima različitu težinu kojom pridonosi ukupnom trošku. Algoritam za izračun udaljenosti uređivanja između dva niza n i m, gdje je $n \geq m$, čija je složenost $O(n * max(1, m/logn))$, mora imati poznate težine pojedine operacije(brisanje, umetanje, zamjena) te također mora biti poznata konačna abeceda iz koje će nizovi n i m biti izgrađeni. Zadovoljavanje tih uvjeta je nužno da bi ostvarili gore navedenu vremensku složenost. Navedeni problem može se riješiti primjenom nekoliko algoritama: Wagner-Fischer, Needleman-Wunsch. Zadatak ovog projekta jest implementacija algoritma Four Russians, metode koja rješava navedeni problem, ali brže nego predloženi algoritmi, te prikaz da za dulje readove radi barem jednako brzo kao Needelamn-Wunsch. U nastavku će biti dan opis samog algoritma, s naglaskom na složenost, kao i prikaz provedbe algoritma na jednostavnom primjeru, kao i rezultati koje algoritam ostvaruje nad nizovima različitih duljina [2].

2. Opći opis algoritma Four Russians

U nastavku će biti opisana sama ideja koji stoji iza algoritma Four Russians. Ali će na samom početku biti dan i kratak osvrt na algoritam Needeman-Wunsch, jer sama ideja rada jest implementirati algoritam Four Russians ali uz uvjet da za velike nizove radi barem jednako dobro kao Needeman-Wunsch, kao što je već ranije rečeno.

2.1 Needelman-Wunsch algoritam

U nastavku je dan vrlo kratki opis algoritma Needelman-Wunch, ako je potreban detaljniji opis, u priloženoj literaturi se može naći opširnija izlaganja o navedenom algoritmu.

Način na koji Needelman-Wunsch algoritam radi, jest da prema relaciji (Relacija 2.1.1), popuni matricu(čije su dimenzije dane duljinama nizova n i m nad kojima se provodi algoritam), te na osnovu popunjene matrice vrati udaljenost uređivanja između dva niza.

$$D(i,j) = \begin{cases} 0 & i = 0 \; and \; j = 0 \\ d * i & j = 0 \\ d * j & i = 0 \\ max \begin{cases} D(i-1, j-1) + w(s_j + t_j) \\ D(i-1, j) + d \\ D(i, j-1) + d \end{cases} & inače \end{cases}$$

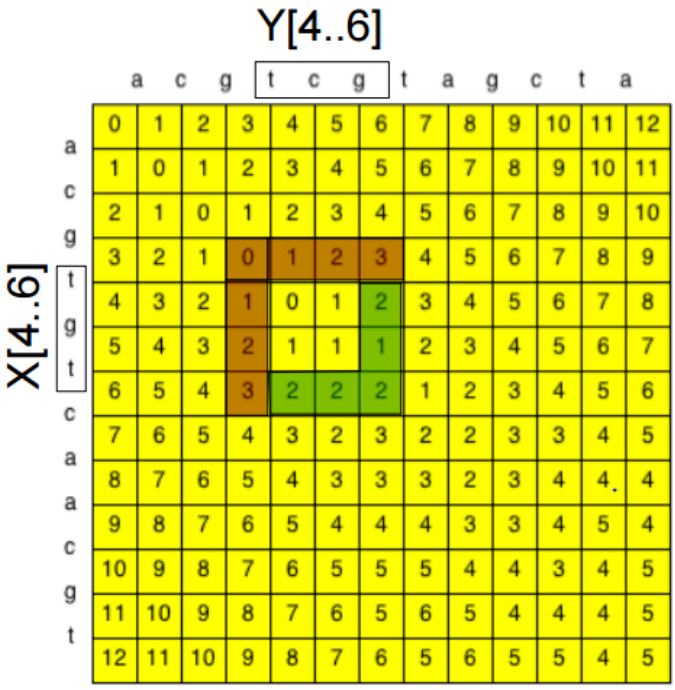
Realacija 2.1.1 Needelman-Wunsch algoritam

Gdje je d kretanje desno i dolje po matrici (umetanje i brisanje), a w je cijena zamijene dvaju znakova odnosno, kretanje po dijagonali. Njegova vremenska i memorijska složenost iznosi $O(n * m)$, te ako se računa samo sličnost ne i poravnavanje onda mu je memorijska složenost linearna [1].

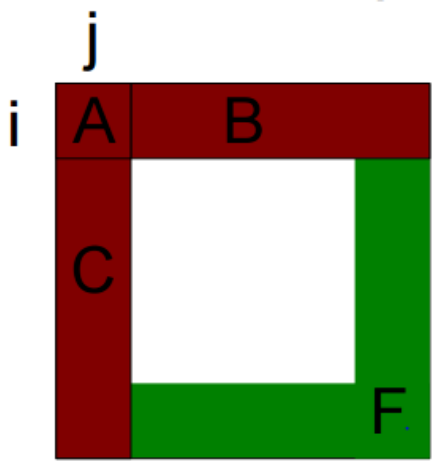
Postavlja se pitanja da li je zadani posao moguće napraviti u manjoj složenosti. Odgovor je potvrdan, i to ako našu matricu podijelimo u podmatrice koje su sačinjene od malog broja redaka, i izračunamo vrijednosti svih mogućih elemenata u podmatricama koje su nam potrebne za nastavak provedbe algoritma, prije nego što sami algoritam i pokrenemo (u nastavku detaljnije opisano). Upravo takav način izvedbe algoritma se naziva „The Four Russians Algorithm“ i u nastavku će biti detaljnije opisan način na koji radi.

2.2 The Four Russians Algorithm

Glavna ideja algoritam je prije same provedbe algoritma izračunati dijelove (pod matrice) koje sudjeluju u popunjavanju tablice (Slika 2.2.1). Na slici se može vidjeti podmatrica dimenzija 4x4 koja je izračunata prije samog pokretanja algoritma. Matrica se nalazi na koordinatama (3,3), a sama širina je označena s t = 3, duljina podmatrice bez elementa koji se nalazi u samom ishodištu (3,3).



Slika 2.2.1. Primjer matrice 4x4



Slika 2.2.2. Izgleda bloka

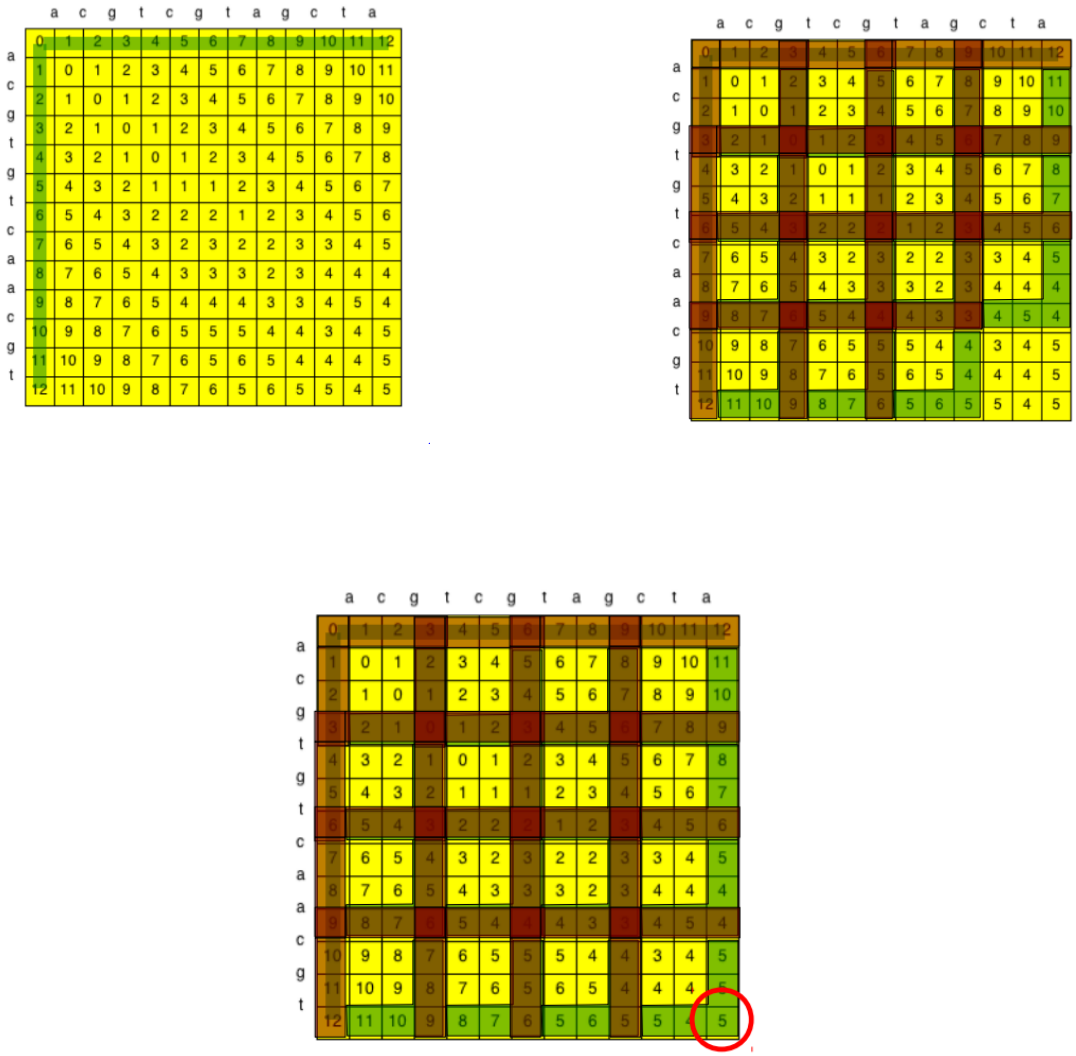
Ako se podmatrica pobliže promotri, može se primijetiti da ona ima oblik prikaz na slici ispod (Slika 2.2.2.). Ono što mi želimo jest iskonstruirati funkciju, koja će nam dati za izlaz F dio podmatrice, tj. prije samog pokretanja algoritma mi želimo u memoriju pohraniti sve moguće kombinacije A,B,C dijelova podmatrice, te X[i+1 ... i+t] i Y[j+1 ... j+t] pod nizova n i m, kako bi za bilo koju kombinaciju A,B,C, X[i+1 ... i+t], Y[j+1 ... j+t] dobili odgovarajući F.

$$b(A,B,C, X[\ i+1 \ldots i+t\ ],Y[\ j+1 \ldots j+t\ ]) = F$$

Relacija 2.2.1 Prikaz ovisnosti izlaza F o ulazu

Proces rješavanja problema mogao bi se opisati kroz sljedeće korake, potrebno je obavit izračun svih blokova (blok je dan prema relaciji 2.2.1), nakon toga je potrebno inicijalizirati prvi redak i stupac matrice, popuniti matricu .korištenjem već izračunatih blokova, te za udaljenost uređivanja vratiti element na poziciji D[n,m].

Na slici u nastavku dan je djelomičan prikaz spomenutih koraka. Za pretpostavku je uzeto da su svi blokovi koji se koriste za popunjavanje tablice već izračunati i poznati.



Slika 2.2.2. Ilustracija provedbe algoritma

Postavlja se pitanje koliko postoji blokova koje je potrebno izračunati prije provedbe samog algoritma. Broj mogućih elemenata koji se mogu pojaviti na mjestu:

1. A = n+1
2. B = $(n+1)^t$
3. C = $(n+1)^t$
4. Broj elemenata u x-u je : $|\Sigma|^t$ gdje Σ označava abecedu iz koje su nizovi izgrađeni
5. Broj elemenata u y-u je : $|\Sigma|^t$ gdje Σ označava abecedu iz koje su nizovi izgrađeni

   Ukupno imamo $(n+1)^{2t+1} + |\Sigma|^{2t}$

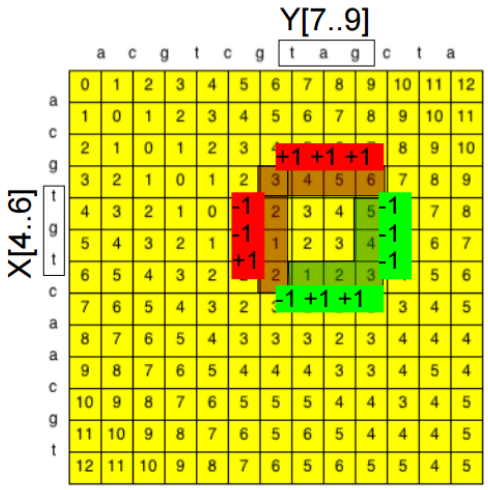
Ono što nas zanima jest da li se taj broj može nekako smanjiti. Odgovor je potvrdan ako se primijeti dvije činjenice:

1. Susjedne ćelije u našoj matrici su udaljene najviše za jedan, pa se stoga B i C mogu zapisati kao pomak (-1,0,1) od A (slika 2.2.3.).
   A = 3
   B = (4,5,6)
   C = (2,1,2)

   A = 3
   B' = (+1,+1,+1)
   C' = (-1,-1,+1)

2. Neka b' bude označimo relaciju koja će za ulaz primiti vrijednosti A,B',C' a za izlaz će dati vrijednost F', koja će biti kodirana s vrijednostima (-1,0,+1) Slika 2.2.3., Onda se može primijetiti da takvu relaciju ne moramo računati za sve vrijednosti varijable A, već slijedi b'(A,B',C',x,y) = b'(0,B',C',x,y).

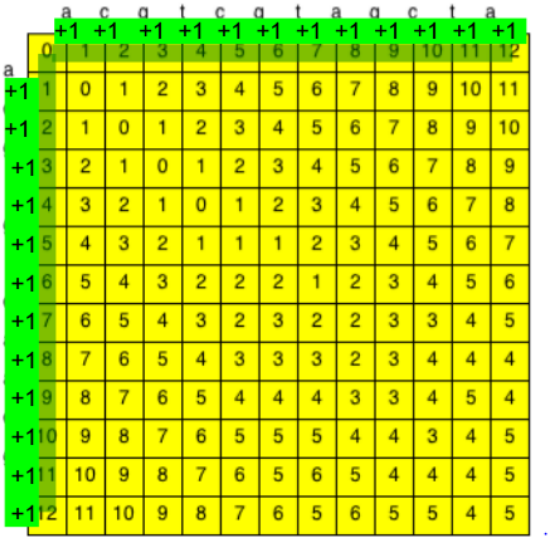
Dokaze za navedene tvrdnje možemo pronaći u literaturi [3][4].

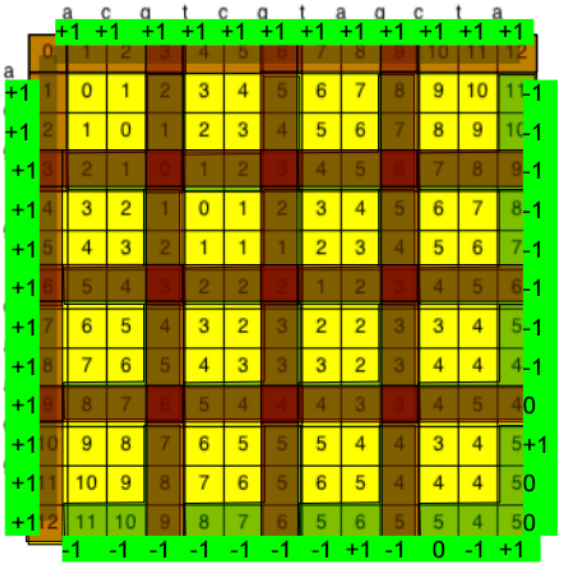


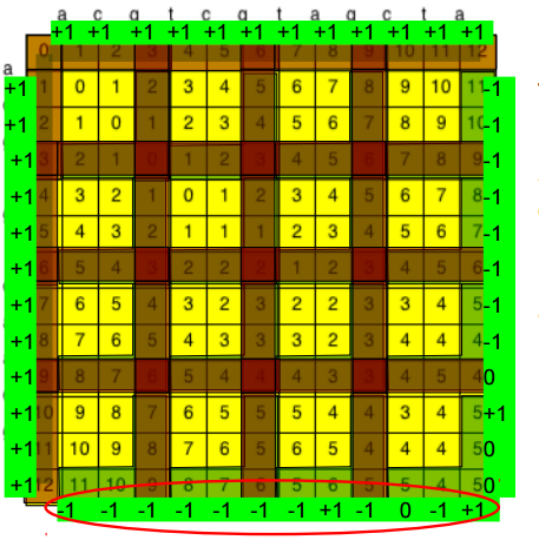
Slika 2.2.3. Primjer bloka korištenjem kodiranja pomaka

Algoritam sada možemo promatrati kroz slijedeće korake:

1. Prvi redak i stupac tablice popuni s pomakom od +1
2. Popuni tablicu uz pomoć već izračunatih blokova (sada je pomak korišten u stvaranju blokova)
3. Vrati kao rezultat sumu pomaka prvog stupca i zadnjeg retka







Slika 2.2.4 Prikaz rezultata korištenjem kodiranja pomaka

Nakon svega postavlja se pitanje kako odabrati veličinu bloka (t vrijednost). Veličinu bloka možemo odabrati prema relaciji danoj u nastavku [3][4]:

$$t = log_{3|\Sigma|}/2$$

Da bi bilo moguće odrediti samo poravnavanje između dva niza za u samoj fazi pred računa nije dovoljno izračunati vrijednost F, nego izračunati vrijednosti svih elemenata koje se pojavljuju u danoj podmatrici. U sljedećem poglavlju bit će dan primjer koji opisuje način na koji algoritam radi kad je riječ o izračunu udaljenosti uređivanja, kao i kako radi kada mora odrediti poravnavanje između dva niza, pa će na primjeru biti jasnije što se je mislilo s prethodnom rečenicom.

3. Primjer provedbe algoritma

U nastavku je dana prikaz provedbe algoritma, uzeto je da su blokovi koji su dobiveni u fazi pretprocesiranja poznati, i da ih je moguće dohvatiti korištenjem relacije:

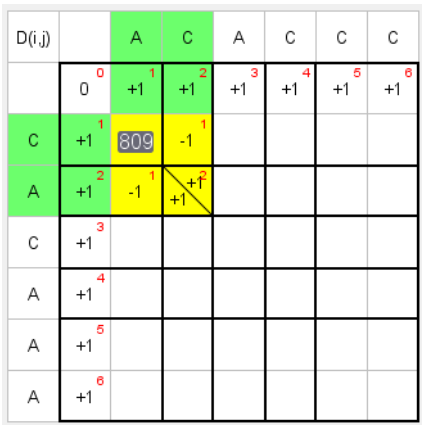
$$b'(0,B',C',x,y) = F$$

Dan je prikaz kako tablica izgleda u svakom koraku kada se u nju blok doda, a sa strana je prikazan sadržaj bloka:

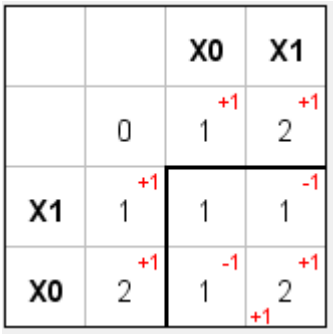
Inicijalno popunjavanje tablice [5]:

D(i,j)		A	C	A	C	C	C
	0 [0]	+1 [1]	+1 [2]	+1 [3]	+1 [4]	+1 [5]	+1 [6]
C	+1 [1]						
A	+1 [2]						
C	+1 [3]						
A	+1 [4]						
A	+1 [5]						
A	+1 [6]						

Prvi korak:

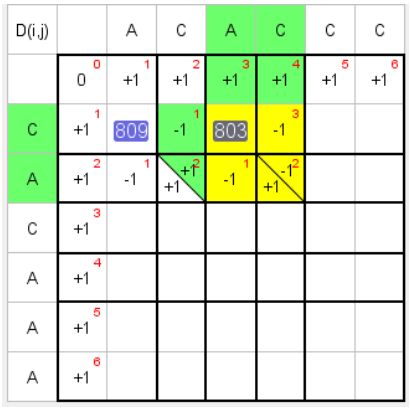


D(i,j)		A	C	A	C	C	C
	0	+1	+1	+1	+1	+1	+1
C	+1	809	-1				
A	+1	-1	+1 +1				
C	+1						
A	+1						
A	+1						
A	+1						

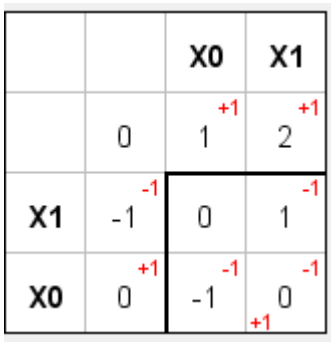


		X0	X1
	0	1 +1	2 +1
X1	1 +1	1	1 -1
X0	2 +1	1 -1	2 +1 +1

Drugi korak:

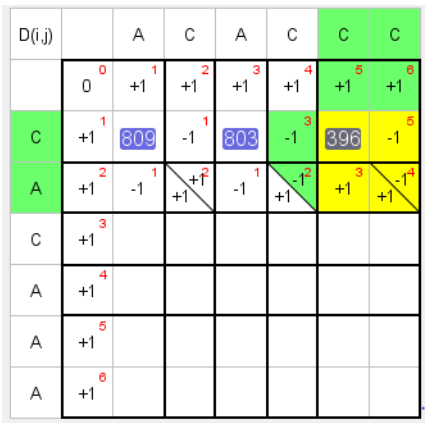


D(i,j)		A	C	A	C	C	C
	0	+1	+1	+1	+1	+1	+1
C	+1	809	-1	803	-1		
A	+1	-1	+1 +1	-1	-1 +1		
C	+1						
A	+1						
A	+1						
A	+1						

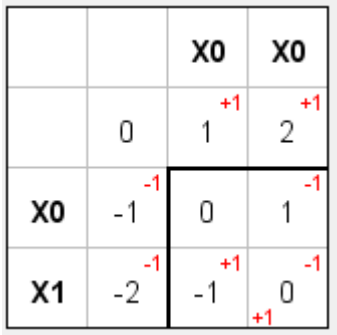


		X0	X1
	0	1 +1	2 +1
X1	-1 -1	0	1 -1
X0	0 +1	-1 -1	0 -1 +1

Treći korak:



D(i,j)		A	C	A	C	C	C
	0	+1	+1	+1	+1	+1	+1
C	+1	809	-1	803	-1	396	-1
A	+1	-1	+1 +1	-1	-1 +1	+1	-1 +1
C	+1						
A	+1						
A	+1						
A	+1						



		X0	X0
	0	1 +1	2 +1
X0	-1 -1	0	1 -1
X1	-2 -1	-1 +1	0 -1 +1

Četvrti korak:

D(i,j)		A	C	A	C	C	C
	0	+1	+1	+1	+1	+1	+1
C	+1	809	-1	803	-1	396	-1
A	+1	-1	+1	-1	-1	+1	-1
C	+1	755	-1				
A	+1	-1	-1				
A	+1						
A	+1						

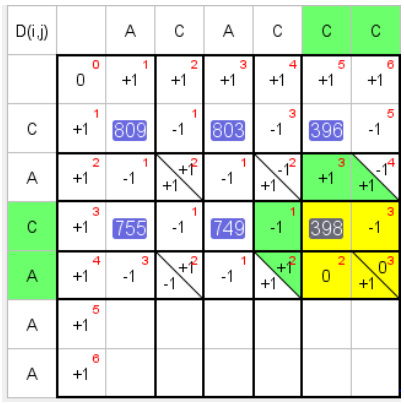
		X0	X1
	0	-1	0
X1	1	0	-1
X0	2	1	0

Peti korak:

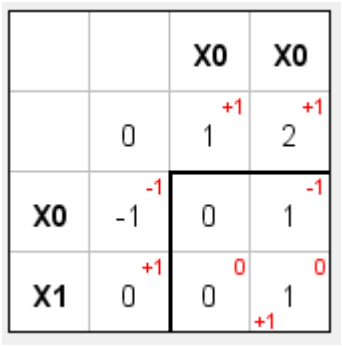
D(i,j)		A	C	A	C	C	C
	0	+1	+1	+1	+1	+1	+1
C	+1	809	-1	803	-1	396	-1
A	+1	-1	+1	-1	-1	+1	-1
C	+1	755	-1	749	-1		
A	+1	-1	-1	-1	+1		
A	+1						
A	+1						

		X0	X1
	0	-1	0
X1	-1	0	-1
X0	0	-1	0

Šesti korak:

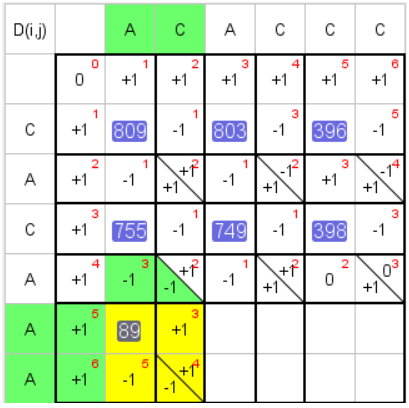


D(i,j)		A	C	A	C	C	C
	0	+1	+1	+1	+1	+1	+1
C	+1	809	-1	803	-1	396	-1
A	+1	-1	+1 / +1	-1	-1 / +1	+1	-1 / +1
C	+1	755	-1	749	-1	398	-1
A	+1	-1	+1 / -1	-1	+1 / +1	0	0 / +1
A	+1						
A	+1						

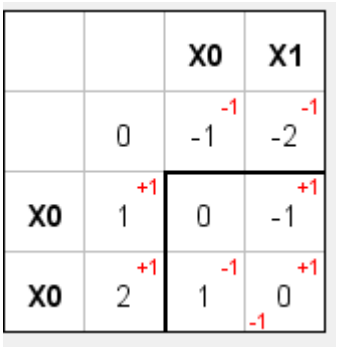


		X0	X0
	0	1	2
X0	-1	0	1
X1	0	0	1

Sedmi korak:

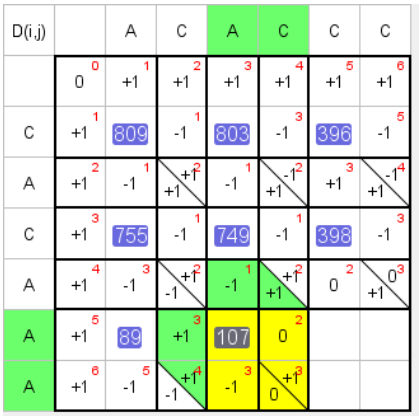


D(i,j)		A	C	A	C	C	C
	0	+1	+1	+1	+1	+1	+1
C	+1	809	-1	803	-1	396	-1
A	+1	-1	+1 / +1	-1	-1 / +1	+1	-1 / +1
C	+1	755	-1	749	-1	398	-1
A	+1	-1	+1 / -1	-1	+1 / +1	0	0 / +1
A	+1	89	+1				
A	+1	-1	+1 / -1				

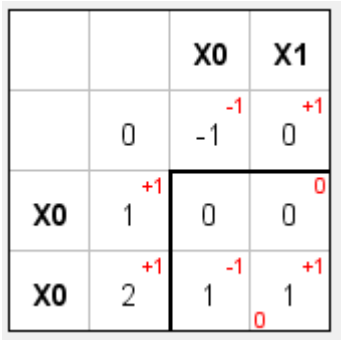


		X0	X1
	0	-1	-2
X0	1	0	-1
X0	2	1	0

Osmi korak:

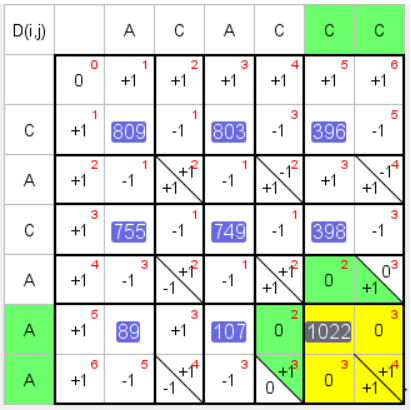


D(i,j)		A	C	A	C	C	C
	0	+1	+1	+1	+1	+1	+1
C	+1	809	-1	803	-1	396	-1
A	+1	-1	+1 / +1	-1	-1 / +1	+1	-1 / +1
C	+1	755	-1	749	-1	398	-1
A	+1	-1	+1 / -1	-1	+1 / +1	0	0 / +1
A	+1	89	+1	107	0		
A	+1	-1	+1 / -1	-1	+1 / 0		

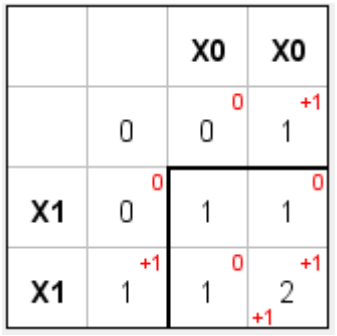


		X0	X1
	0	-1	0
X0	1	0	0
X0	2	1	1

Deveti korak:

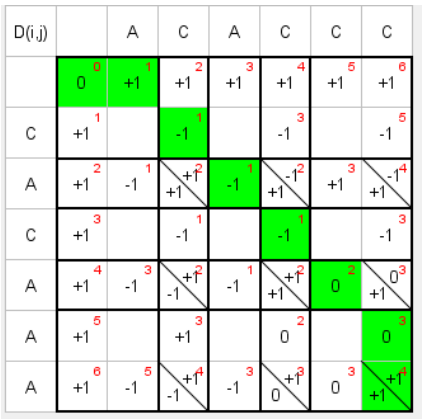


D(i,j)		A	C	A	C	C	C
	0	+1	+1	+1	+1	+1	+1
C	+1	809	-1	803	-1	396	-1
A	+1	-1	+1	-1	-1	+1	-1
C	+1	755	-1	749	-1	398	-1
A	+1	-1	-1	-1	+1	0	+1
A	+1	89	+1	107	0	1022	0
A	+1	-1	-1	-1	0	0	+1



		X0	X0
	0	0	1
X1	0	1	1
X1	1	1	2

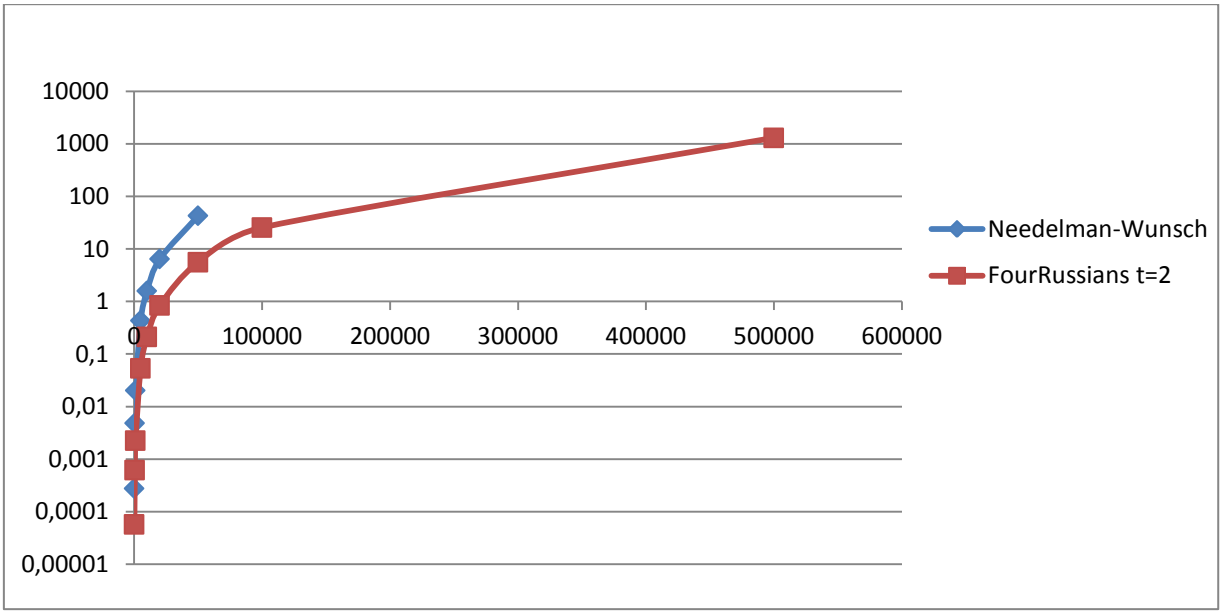
Edit script dobivena standardnim algoritmom:



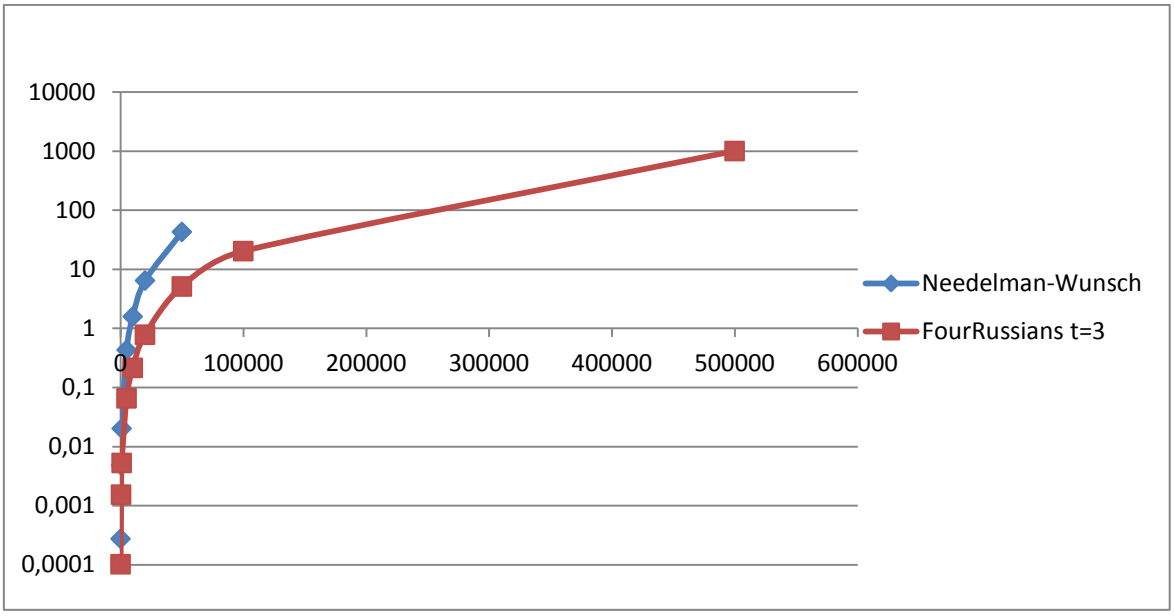
D(i,j)		A	C	A	C	C	C
	0	+1	+1	+1	+1	+1	+1
C	+1		-1		-1		-1
A	+1	-1	+1	-1	-1	+1	-1
C	+1		-1		-1		-1
A	+1	-1	-1	-1	+1	0	0
A	+1		+1		0		0
A	+1	-1	-1	-1	0	0	+1

4. Rezultati

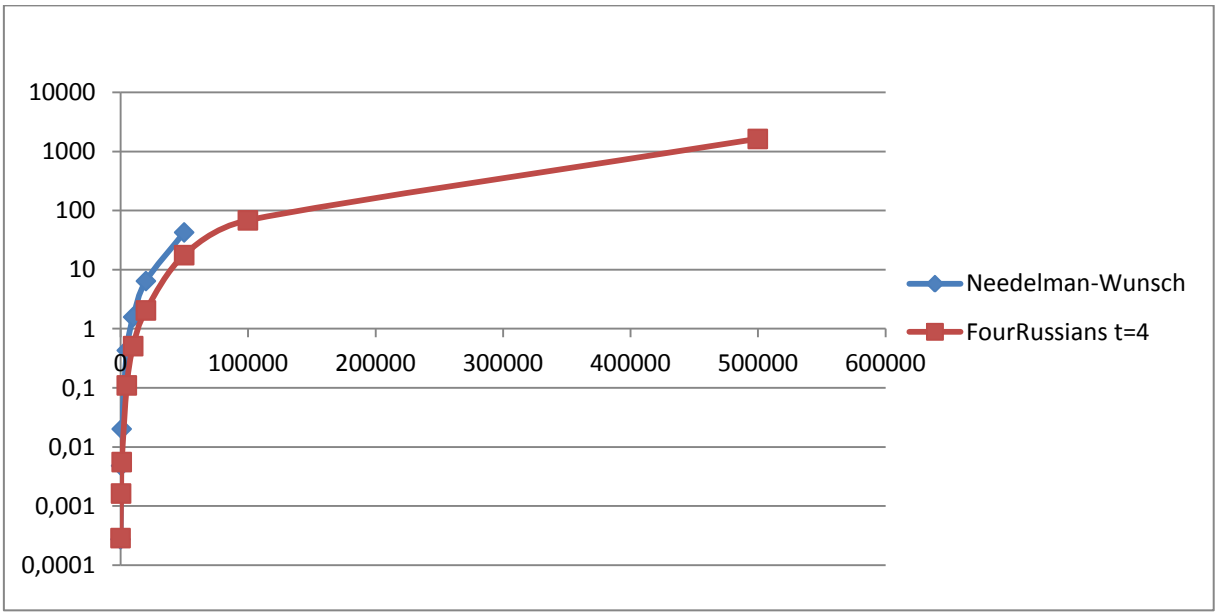
U nastavku će biti dan prikaz rezultata usporedbe Four Russian metode i Needelman Wunsch algoritma:



Slika 4.1. Usporedba izvođenja Needelman-Wunsch algoritma i Four Russians metode (x os – duljina stringa , y-os vrijeme izvođenja (logaritamska skala)) t=2



Slika 4.2. Usporedba izvođenja Needelman-Wunsch algoritma i Four Russians metode (x os – duljina stringa , y-os vrijeme izvođenja (logaritamska skala)) t=3



Slika 4.3. Usporedba izvođenja Needelman-Wunsch algoritma i Four Russians metode (x os – duljina stringa , y-os vrijeme izvođenja (logaritamska skala)) t=4

Duljina niza	Memorija t=2	Memorija t=3	Memorija t=4
50000	2389MB	2885MB	-
100000	9575MB	6065MB	-
500000	234GB	105GB	180GB

Tablica 4.1. Prikaz zauzeća memorije prilikom izvedbe Four Russian algoritma

Iz mjerenja se može zaključiti da brzina Four Russian metode primjenjene na riješavanje danog problema rezultira ubrzanjem, i pobjeđuje Needelman-Wunsch algoritam. Također bitno je napomenuti da se je algoritam Needelman-Wunsch srušio na nizovima čija je duljina bila veća od 50000 znakova. Također može se primjetiti da u gornjoj tablici nedostaju određenja mjerenja to je ili iz razloga što se je algoritam prebrzo izvršio te stoga očitanje memorije u zadanom alatu nije bilo moguće. Mjerenja koja nedostaju u tablici za t=4, nisu završila prebrzo no nisu očitana sa servera, a duljina izvođenja predprocesiranja za t=4 traje predugo da bi se ista ponovila.

Također bitno je napomenuti da se rezultati izračunate duljine uređivanja poklapaju, izlaz Needelman-Wunscha jednak je izlazu Four Russian algoritma, stoga možemo zaključiti da rezultat implementiranog riješenja jest točan.

5. Zaključak

Problem pronalaženje udaljenosti uređivanja dvaju nizova, kao i sam problem poravnavanja dvaju nizova, ne spada u računalno ne zahtjevne operacije. Iako za nizove manjih duljina nije potrebno poduzeti velike korake optimizacije, već je moguće uzeti dobro provjerene algoritme čija težina razumijevanja i implementacije nije prevelika, s povećanjem duljine nizova takvi algoritmi nisu dostatni već je potrebno pribjeći koracima optimizacije i prilagodbe algoritama. U sklopu rada napravljena je implementacija metode „Four Russian“ čija je zadaća upravo ubrzati izvođenje algoritama koji se koriste za rješavanje gore navedenih problema. Sam cilj projekta je postignut, vidljivo je da implementirani algoritam, koji sadržava navedenu metodu, radi brže nego implementacija dobro poznatog algoritma Needlman-Wunsch. Kroz projekt upoznali smo se s procesom optimizacije algoritma, sam proces ne spada u domenu trivijalnih operacija, no bez odgovarajućih optimizacije moguće je da bi imali algoritam koji je neupotrebljiv za određenu veličinu danog problema. Stoga možemo zaključiti da je sam proces optimizacije jedna od važnijih stavaka kod izrade algoritama.

6. Literatura

[1] M.Šikić , M. Domazet-Lošo. Skripta, Kolegij: Bioinformatika. Zagreb, 2013.

[2] W.J.Masek, M.S.Paterson. A faster algorithm computing string edit distances, Journal of Computer and System Sciences. Volume 20 / Issue 1, February 1980, Pages 18–31

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

[4] Carl Kingsford, Four Russians'Speedup,
http://www.cs.cmu.edu/~ckingsf/class/02-714/Lec09-4russians.pdf

[5] http://baba.sourceforge.net/docs/BABA_report.pdf

7. Sažetak

Sama dokumentacija načinjena je u svrhu pojašnjena kako metoda „Four Russians“ funkcionira u teoriji, koji su njezini koraci, kada ju se može upotrijebiti i na koje vrste problema se primjenjuje, te koji su rezultati primjene metode, dobre strane i loše strane. U uvodu je dano kratko upoznavanje s problemom koji je bio osnova rada, kao i samo razmatranje koje su mana određenih jednostavnijih rješenja i zašto je opravdano navedenu metodu iskoristiti pri rješavanju zadanog problema. Poglavlje „The Four Russians Algorithm“ daje objašnjenje metode, ali s vrlo apstraktnog nivoa. Više je dan pogled s matematičke strane, početna ideja, dobre i loše strane, te kako početnu ideju poboljšati, kao i određeni matematički zaključci koji potvrđuju sva navedena razmatranja. Nakon toga dan je vrlo jednostavan primjer problema i objašnjeno je korak po korak kako metoda radi u implementiranom rješenju. U poglavlju „Rezultati“ moguće je vidjeti prikaz rezultata koje metoda ostvaruje primjenom na stvarnim problemima, kao i usporedba s jednostavnijim rješenjima danog problema. Za sam kraj dan je kratak osvrt na cijeli projekt, te osvrt na uspješnost samog projekta kao i na poteškoće koje su uočene prilikom izrade istog.