Bioinformatika

27. mart 2018.

Sadržaj

1 Gde u genomu počinje replikacija genoma? 1

1.1 Uvod . . . 1

1.2 Replikacija genoma . . . 2

1.2.1 DNK . . . 2

1.2.2 Replikacija genoma u ćeliji . . . 3

1.2.3 Pronalaženje početnog regiona replikacije . . . 8

1.3 Zadaci sa vežbi . . . 13

1.3.1 FrequentWords . . . 13

1.3.2 Faster FrequentWords . . . 13

1.3.3 Skew Diagram . . . 15

1.3.4 FrequentWords With Mismatches . . . 15

Predgovor

Tekst se sastoji od proširenih beleški sa predavanja na osnovu knjige Pavel A. Pevzner, Phillip Compeau: Bioinformatics Algorithms: An Active Learning Approach.
Tekst su sastavili studenti sa kursa održanog u školskoj 2017/2018 godini:

- Una Stanković 1095/2016
- Marina Nikolić 1055/2017
- Strahinja Milojević 1049/2017

Glava 1

Gde u genomu počinje replikacija genoma?

1.1 Uvod

Na samom početku, želimo da definišemo pojam bioinformatike i da pokušamo da shvatimo koji je njen osnovni cilj. Da bismo to postigli, pogledajmo tri definicije, iz različitih izvora:

- "Bioinformatika je nauka koja se bavi prikupljanjem i analizom kompleksnih bioloških podataka poput genetskih kodova." - Oksfordski rečnik (engl. *Oxford Dictionary*)
- "Bioinformatika predstavlja prikupljanje, klasifikaciju, čuvanje i analizu biohemijskih i bioloških informacija korišćenjem računara, a posebno se primenjuje u molekularnoj genetici i genomici." - Rečnik Meriam-Vebster (engl. *Merriam-Webster Dictionary*)
- "Bioinformatika je interdisciplinarno polje koje radi na razvoju metoda i softverskih alata za razumevanje bioloških podataka." - Vikipedija (engl. *Wikipedia*)

Na osnovu ove tri definicije možemo zaključiti da:
Bioinformatika predstavlja primenu računarskih tehnologija u istraživanjima u oblasti biologije i srodnih nauka.

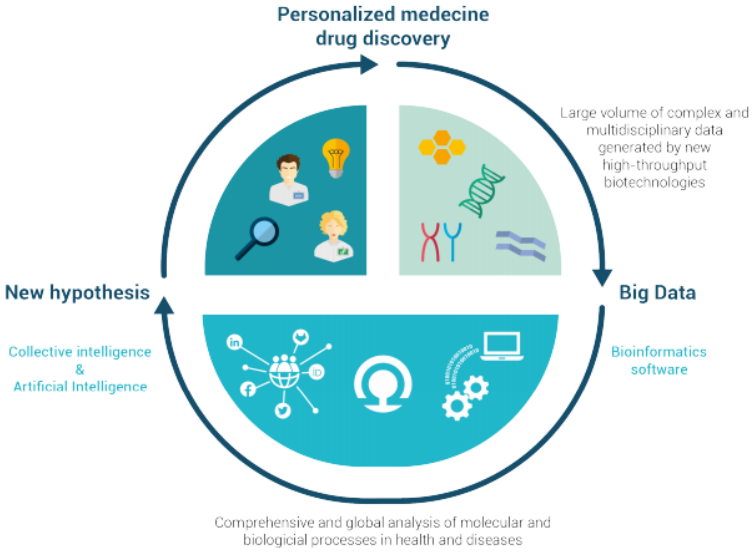
Bioinformatika ima široku primenu i njene primene rastu zajedno sa razvojem discipline. Kao što možemo videti na slici ispod, primena bioinformatike se može sagledati kroz personalizovanu medicinu. Naime, na osnovu prikupljene veće količine podataka i njihove analize, uz pomoć različitih računarskih metoda, na primer metoda veštačke inteligencije, možemo doći do informacija potrebnih da na najbolji način lečimo pacijenta ili mu odredimo terapiju koja će mu na najbolji, najbrži i najbezbolniji način pomoći da prevaziđe određene zdravstvene probleme.

Bioinformatika je spoj više različitih disciplina, kao što su:

- Statistika
- Istraživanje podataka
- Računarstvo
- Računarska biologija
- Biologija
- Biostatistika

Prikaz preklapanja ovih disciplina možemo videti na slici 1.2.

Slika 1.1: Primena bioinformatike



1.2 Replikacija genoma

1.2.1 DNK

Dezoksiribonukleinska kiselina (akronimi DNK ili DNA, od engl. *deoxyribonucleic acid*), nukleinska kiselina koja sadrži uputstva za razvoj i pravilno funkcionisanje svih živih organizama. Zajedno sa RNK i proteinima, DNK je jedan od tri glavna tipa makromolekula koji su esencijalni za sve poznate forme života.

Sva živa bića svoj genetički materijal nose u obliku DNK, sa izuzetkom nekih virusa koji imaju ribonukleinsku kiselinu (RNK). DNK ima veoma važnu ulogu ne samo u prenosu genetičkih informacija sa jedne na drugu generaciju, već sadrži i uputstva za građenje neophodnih ćelijskih organela, proteina i RNK molekula. DNK segment koji sadrži ova važna uputstva se naziva gen.

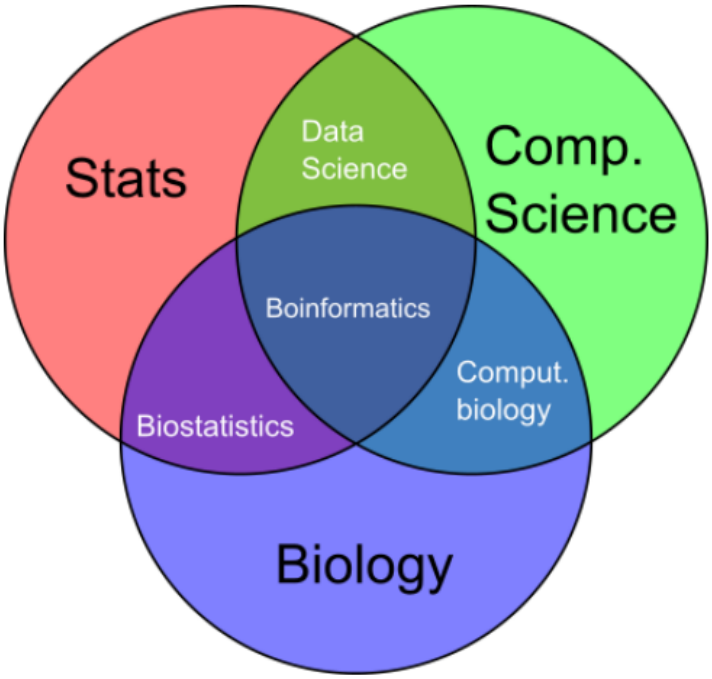
DNK se sastoji iz dva polimerna lanca koji imaju antiparalelnu orijentaciju, i svaki od njih je sastavljen od azotnih baza:

- adenin (A)
- timin (T)
- guanin (G)
- citozin (C)

Lanci DNK su međusobno spojeni i to tako da se veze uspostavljaju isključivo između adenina i citozina ili između guanina i timina. Na osnovu toga, ako nam je poznat sastav jednog lanca, lako možemo zaključiti i sastav drugog lanca, zbog čega se kaže da su DNK lanci **međusobno komplementarni**.

Da bismo lakše manipulisali sa informacijama koje DNK nosi i približili sadržaj računarskoj struci, DNK ćemo posmatrati kao nisku nad azbukom A, C, G, T.

Slika 1.2: Preklapanjem različitih disciplina dobijamo bioinformatiku.



1.2.2 Replikacija genoma u ćeliji

Replikacija genoma je jedan od najvažnijih zadataka ćelije. Pre nego što se podeli, ćelija mora da najpre replicira svoj genom, tako da svaka od ćerki ćelija dobije svoju kopiju.

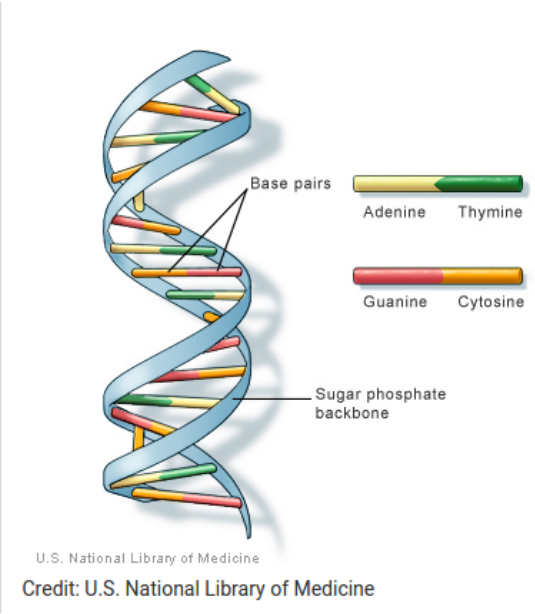
Dzejms Votson (engl. *James Watson*) i Fransis Krik (engl. *Fransis Crick*) su 1953. godine napisali rad u kome su primetili da postoji mehanizam za kopiranje genetskog materijala. Oni su uočili da se lanci roditeljskog DNK molekula odvijaju tokom replikacije i da se, potom, svaki lanac ponaša kao uzorak za sintezu novog lanca (na osnovu toga što se uvek spajaju iste aminokiseline A-C i G-T, rekreiranje lanca je moguće). Kao rezultat ovakvog ponašanja, proces replikacije počinje parom komplementarnih lanca i završava se sa dva para komplementarnih lanaca, kao što se može videti na slici ispod.

Replikacija počinje u regionu genoma koji se naziva **početni region replikacije** (skraćeno *oriC*), izvode je enzimi koje se nazivaju DNK polimeraze, koje predstavljaju mašine za kopiranje na molekularnom nivou.

Nalaženje početnog regiona replikacije predstavlja veoma važan problem, ne samo za razumevanje funkcionisanja kako se ćelije repliciraju, već je koristan i u raznim biomedicinskim problemima. Na primer, neki metodi genskih terapija uključuju genetski izmenjene mini genome, koji se zovu virusni vektori, zbog svoje sposobnosti da prodru kroz ćelijski zid (poput pravih virusa). Virusni vektori u sebi nose veštačke gene koji unapredjuju postojeći genom. Genska terapija je prvi put uspešno izvršena 1990. godine na devojčici koja je bila toliko otporna na infekcije da je bila primorana da živi isključivo u sterilnom okruženju.

Osnovna ideja genske terapije je da se pacijent, koji pati od nedostatka nekog bitnog gena, zarazi viralnim vektorom koji sadrži veštački gen koji enkodira terapeutski protein. Jednom kad

Slika 1.3: Prikaz DNK, slika preuzeta sa https://ghr.nlm.nih.gov/primer/basics/dna



je unutar ćelije, vektor se replicira, što dovodi do lečenja bolesti pacijenta. Da bi moglo da dodje do ovoga, biolozima je neophodno da znaju gde je *oriC*.

Kako ćelija prepoznaje *oriC*?

Pitamo se kako ćelija prepoznaje oriC? Sigurno je da postoji neka niska aminokiselina koja označava oriC, ali kako ga prepoznati?

Ograničimo se na bakterijski genom, koji se sastoji od jednog kružnog hromozoma. Istraživanje je pokazalo da je region, koji predstavlja oriC kod bakterija, dug svega nekoliko stotina nukleotida.

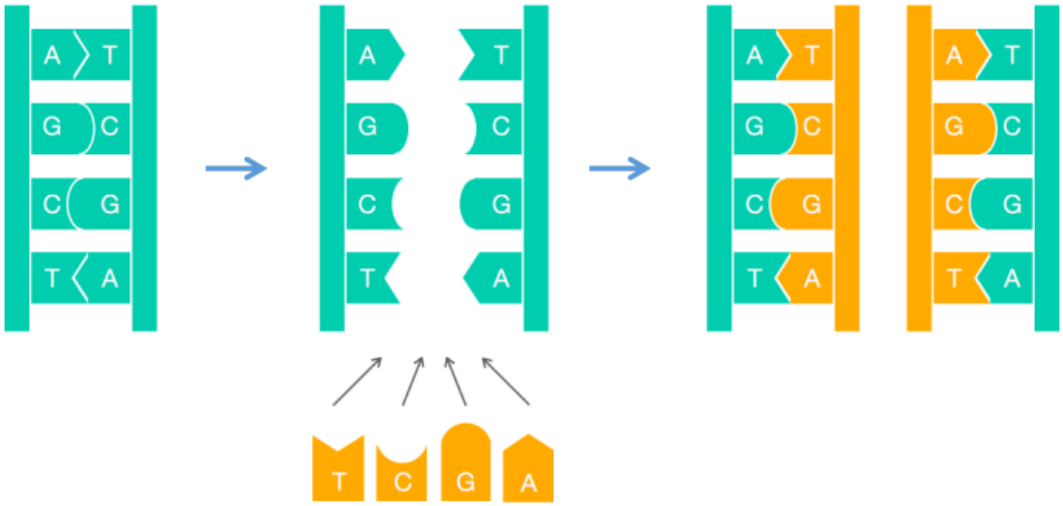
Poznato je da DNKA utiče na početak replikacije. *DNKA* je protein koji se vezuje na kratki segment unutar oriC, poznatiji kao **DNKA boks**. Ona predstavlja poruku unutar sekvence DNK koja govori proteinu DNKA da se veže baš tu. Postavlja se pitanje kao pronaći taj region bez prethodnog poznavanja izgleda DNKA boks?

Da bismo bolje razumeli *problem skrivene poruke* uzmimo za primer priču Edgara Alana Poa - "Zlatni jelenak" (engl. "*The Gold-Bug*"). Naime, u toj priči jedan od likova, Vilijam Legrand (engl. *William Legrand*), treba da dešifruje poruku :

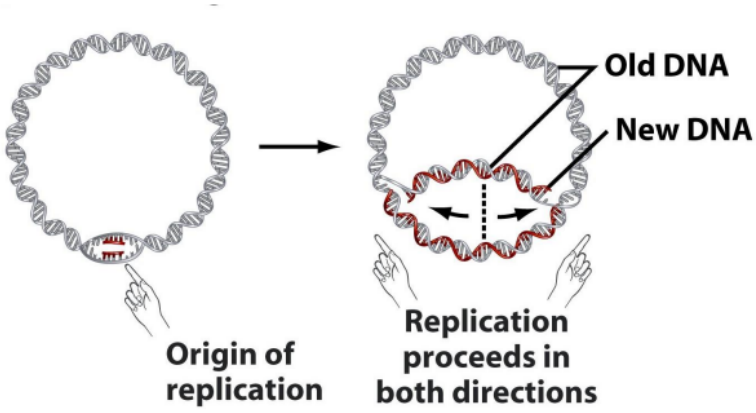
```
53++!305))6*;4826)4+.)4+);806*;48!8‘6 0))85;]8*:+*8!83(88)5*!;46(;88*96*?;8
)*+(;485);5*!2:*+(;4956*2(5*4)8‘8*;40 69285);)6!8)4++;1(+9;48081;8:8+1;48!8 5;4)
485!528806*81(+9;48;(88;4(+?34;48 )4+;161;:188;+?;
```

On uočava da se ";48" pojavljuje veoma često, i da verovatno predstavlja "THE", najčešću reč u engleskom jeziku. Znajući to, zamenjuje karaktere odgovarajućim slovima i postepeno dešifruje celu poruku.

Slika 1.4: Prikaz replikacije



Slika 1.5: Prikaz početka replikacije kod bakterija




```
53++!305))6*THE26)H+.)H+)806*THE !E‘60))E5;]E*:+*E!E3(EE)5*!TH6(T EE*96*?;E)*+
(THE5)T5*!2:*+(TH956 *2(5*H)E‘E*TH0692E5)T)6!E)H++T1( +9THE0E1TE:E+1
THE!E5T4)HE5!52880 6*E1(+9THET(EETH(+?34THE)H+T161T :1EET+?T
```

Želeli bismo da ovaj princip primenimo na naš problem nalaska *oriC*-a. Ideja je da uvidimo da li postoje reči koje se neuobičajeno često pojavljuju. Uvedimo termin k-gram da označimo string dužine k i COUNT(Text, Pattern) da označimo broj puta kojih se k-gram *Pattern* pojavio u tekstu *Text*. Osnovna ideja je da pomeramo prozor, iste dužine kao k-gram *Pattern*, niz tekst, usput proveravajući da li se pojavljuje *Pattern* u nekome od njih.

```
PATTERNCOUNT(Text, Pattern)
    count = 0
    for i = 0 to |Text| - |Pattern|
        if Text(i,|Pattern|) = Pattern
            count = count + 1
    return count
```

Za neki *Pattern* kažemo da je on *najčešći k-gram* u tekstu *Text*, ako je njegov *COUNT* najveći među svim k-gramima. Na primer, **ACTAT** je najčešći 5-gram u tekstu *Text* = ACA**ACTAT**GCA**ACTAT**CGGGACA**ACTAT**CCT, a **ATA** je najčešći 3-gram u *Text* = CG**ATATA**TCC**ATA**G.

Sada, problem pronalaska čestih reči možemo posmatrati kao računarski problem:

Problem čestih reči: Pronaći najčešće k-grame u niski karaktera.
Ulaz: Niska Text i ceo broj k.
Izlaz: Svi najčešći k-grami u niski Text.

Osnovni algoritam za pronalazak čestih k-grama u stringu *Text* proverava sve k-grame koji se pojavljuju u tom stringu (takvih k-grama ima $|Text| - k + 1$) i potom izračunava koliko puta se svaki k-gram pojavljuje. Da bismo implementirali ovaj algoritam, moramo da izgenerišemo niz $COUNT$, gde je $COUNT(i) = COUNT(Text, Pattern)$ za $Pattern = Text(i, k)$.

```
FrequentWords(Text, k)
    FrequentPatterns <- an empty set
    for i = 0 to |Text| - k
        Pattern <- the k-mer Text(i,k)
        COUNT(i) <- PatternCount(Text, Pattern)
    maxCount <- max value in array COUNT
    for i = 0 to |Text| - k
        if COUNT(i) = maxCount
            add Text(i,k) to FrequentPatterns
    remove duplicates from FrequentPatterns
    return FrequentPatterns
```

Pitamo se, sada, kolika je složenost ovakvog pristupa?
Ovaj algoritam, iako uspešno nalazi ono što se od njega traži, nije najefikasniji. S obzirom na to da svaki k-gram zahteva $|Text| - k + 1$ provera, svaki od njih zahteva i do k poređenja, pa je broj koraka izvršavanja funkcije $PatternCount(Text, Pattern)$ zapravo $(|Text| - k + 1) * k$. Osim toga, $FrequentWords$ mora pozvati $PatternCount$ $|Text| - k + 1$ puta (po jednom za svaki k-gram teksta), tako da je ukupan broj koraka $(|Text| - k + 1) * (|Text| - k + 1) * k$.
Iz navedenog, možemo zaključiti da je ukupna cena izvršavanja algoritma $FrequentWords$ $O(|Text|^2 * k)$.

Primer: Pronalazak čestih reči kod bakterije *Vibrio cholerae*

Posmatrajmo, najpre, tablicu najčešćih k-grama u *oriC* regionu bakterije *Vibrio cholerae*. Da li nam se čini da se neki k-grami pojavljuju neuobičajeno često?

Na primer, 9-gram **ATGATCAAG** se pojavljuje tri puta u *oriC* regionu, da li nas to iznenađuje?

Označili smo najčešće 9-grame, umesto nekih drugih k-grama, jer je eksperimentima pokazano da su DNKA boksovi kod bakterija dugi 9 nukleotida. Verovatnoća da postoji 9-gram koji se pojavljuje 3 ili više puta u proizvoljno generisanom DNK stringu dužine 500 je 1/1300. Uočimo da postoje četiri različita 9-grama koji se ponavljaju tri ili više puta u ovom regionu, to su: ATGATCAAG, CTTGATCAT, TCTTGATCA i CTCTTGATC.

Slika 1.6: Tablica najčešćih k-grama u *oriC* regionu bakterije Vibrio cholerae

k	3	4	5	6	7	8	9
count	25	12	8	8	5	4	3
k-mers	tga	atga	gatca	tgatca	atgatca	atgatcaa	atgatcaag
			tgatc				cttgatcat
							tcttgatca
							ctcttgatc

Slika 1.7: Prikaz 9-grama ATGATCAAG i njegovog komplementa u *oriC* regionu Vibrio cholerae

atcaatgatcaacgtaagcttctaagc**ATGATCAAG**gtgctcacacagtttatccacaacctgagtgg
atgacatcaagataggtcgttgtatctccttcctctcgtactctcatgaccacggaaag**ATGATCAAG**
agaggatgatttcttggccatatcgcaatgaatacttgtgacttgtgcttccaattgacatcttcagc
gccatattgcgctggccaaggtgacggagcgggattacgaaagcatgatcatggctgttgttctgttt
atcttgttttgactgagacttgttaggatagacggtttttcatcactgactagccaaagccttactct
gcctgacatcgaccgtaaattgataatgaatttacatgcttccgcgacgatttacct**CTTGATCAT**cg
atccgattgaagatcttcaattgttaattctcttgcctcgactcatagccatgatgagct**CTTGATCA**
Tgtttccttaaccctctatttttacggaaga**ATGATCAAG**ctgctgct**CTTGATCAT**cgtttc

Mala verovatnoća da se neki 9-gram toliko puta pojavi u *oriC*-u kolere, govori nam da neki od četiri 9-grama koje smo pronašli može biti potencijalni DNKA boks, koji započinje replikaciju. Ali, koji?

Podsetimo se da nukleotidi A i T, kao i C i G, su komplementarni. Ako imamo jednu stranu lanca DNK i neke slobodne nukleotide, možemo lako zamisliti sintezu komplementarnog lanca, kao što se vidi na slici ispod.

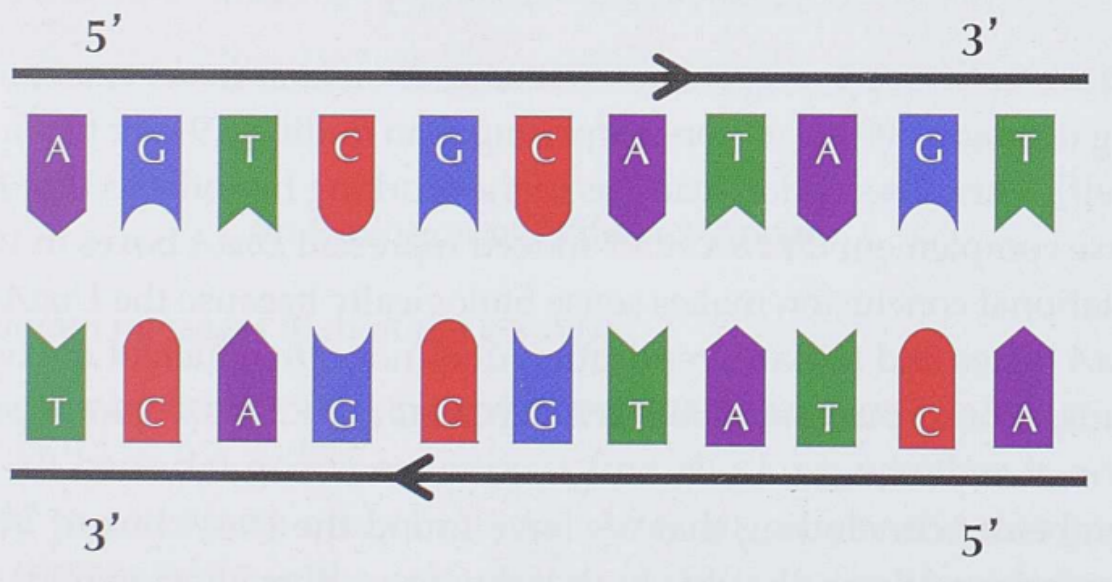
Posmatrajmo ponovo sliku 1.7. Na njoj možemo uočiti 6 pojavljivanja niski ATGATCAAG i CTTGATCAT, koji su zapravo komplementarni. Naći 9-gram koji se pojavljuje 6 puta u DNK nisci dužine 500 nukleotida, je još više iznenadjujuće, nego pronaći 9-gram koji se pojavljuje tri puta. Ovo posmatranje nas dovodi do toga da je ATGATCAAG (zajedno sa svojim komplementom) zaista DNKA boks Vibrio cholerae. Ovaj zaključak ima i smisla biološki, jer DNKA proteinu, koji se vezuje i započinje replikaciju, nije bitno za koji od dva lanca se vezuje.

Primer: Pronalazak čestih reči kod bakterije *Thermotoga petrophila*

Nakon što smo pronašli skrivenu poruku za Vibrio cholerae, ne bi trebalo da odmah zaključimo da je ta poruka ista kod svih bakterija. Najpre bi trebalo da proverimo da li se ona nalazi u *oriC* regionu drugih bakterija, možda različite bakterije, imaju drugačije DNKA boksove. Uzmimo, za primer, *oriC* region bakterije Thermotoga petrophila. Ona predstavlja bakteriju koja obitava u izrazito toplim regionima, na primer u vodi ispod rezervi nafte, gde temperature prelaze 80 stepeni Celzijusa. Pogledajmo kako izgleda *oriC* region ove bakterije.

Možemo lako uočiti da se u ovom regionu nigde ne javljaju niske ATGATCAAG ili CTTGATCAT, iz čega zaključujemo da različite bakterije mogu koristiti različite DNKA boksove, kako bi pružile skrivenu poruku DNKA proteinu. Odnosno, za različite genome imamo različite DNKA boksove.

Slika 1.8: Komplementarni lanci se "kreću" u suprotnim smerovima.



Slika 1.9: Prikaz *oriC* regiona Thermotoga petrophila

```
aactctatacctccttttttgtcgaatttgtgtgatttatagagaaaatcttattaactgaaactaa
aatggtaggtttggtggtaggttttgtgtacattttgtagtatctgatttttaattacataccgta
tattgtattaaattgacgaacaattgcatggaattgaatatatgcaaaacaaacctaccaccaaac
tctgtattgaccatttaggacaacttcagggtggtaggtttctgaagctctcatcaatagactat
tttagtctttacaaacaatattaccgttcagattcaagattctacaacgctgttttaatgggcgtt
gcagaaaacttaccacctaaaatccagtatccaagccgatttcagagaaacctaccacttacctac
cacttacctaccacccgggtggtaagttgcagacattattaaaaacctcatcagaagcttgttcaa
aaatttcaatactcgaaacctaccacctgcgtcccctattatttactactactaataatagcagta
taattgatctgaaaagaggtggtaaaaaa
```

Najčešće reči u ovom *oriC* su:

- AACCTACCA,
- ACCTACCAC,
- GGTAGGTTT,
- TGGTAGGTT,
- AAACCTACC,
- CCTACCACC

Pomoću alata koji se zove Ori-Finder, nalazimo CCTACCACC i njegov komplement GGTGGTAGG kao potencijalne DNKA boksove naše bakterije. Ove dve niske se pojavljuju ukupno 5 puta.

Naučili smo da pronađemo skrivene poruke ako je *oriC* dat, ali ne znamo da pronađemo *oriC* u genomu.

Slika 1.10: Prikaz CCTACCACC i njenog komplementa u *oriC* regionu Thermotoga petrophila

aactctatacctccttttttgtcgaatttgtgtgatttatagagaaaatcttattaactgaaactaa
aatggtaggtttGGTGGTAGGttttgtgtacattttgtagtatctgatttttaattacataccgta
tattgtattaaattgacgaacaattgcatggaattgaatatatgcaaaacaaaCCTACCACCaaac
tctgtattgaccattttaggacaacttcagGGTGGTAGGtttctgaagctctcatcaatagactat
tttagtctttacaaacaatattaccgttcagattcaagattctacaacgctgttttaatgggcgtt
gcagaaaacttaccacctaaaatccagtatccaagccgatttcagagaaacctaccacttacctac
cacttaCCTACCACCcgggtggtaagttgcagacattattaaaaacctcatcagaagcttgttcaa
aaatttcaatactcgaaaCCTACCACCtgcgtcccctattatttactactactaataatagcagta
taattgatctgaaaagaggtggtaaaaaa

1.2.3 Pronalaženje početnog regiona replikacije

Zamislimo da pokušavamo da nađemo *oriC* u novom sekvenciranom genomu bakterije. Ako bismo tražili niske poput ATGATCAAG/CTTGATCAT ili CCTACCACC/GGTGGTAGG to nam verovatno ne bi bilo puno od pomoći, jer novi genom može koristiti potpuno drugačiju skrivenu poruku. Posmatrajmo, zato, drugačiji problem: umesto da tražimo grupe određenog k-grama, pokušajmo da nađemo svaki k-gram koji formira grupu u genomu. Nadajmo se da će nam lokacije ovih grupa u genomu pomoći da odredimo lokaciju *oriC*-a.
Ideja je da pomeramo prozor fiksirane dužine L kroz genom, tražeći region u kome se k-gram pojavljuje više puta uzastopno. Za L ćemo uzeti vrednost 500, koja predstavlja najčešću dužinu *oriC*-a kod bakterija.

Definisali smo k-gram kao *grupu*, ako se pojavljuje više puta unutar kratkog intervala u genomu. Formalno, k-gram Pattern formira (L, t) grupu unutar niske Genome ako postoji interval genoma, dužine L, u kome se k-gram pojavljuje barem t puta.

Problem pronalaženja grupa. Naći k-grame koji formiraju grupe unutar niske karaktera.
Ulaz. Niska Genome i celi brojevi k (dužina podniske), L (dužina prozora) i t (broj podniski u grupi).
Izlaz. Svi k-grami koji formiraju (L, t)-grupe u niski Genome.

U genomu bakterije E.coli postoji 1904 različitih 9- grama koji formiraju (500,3)-grupe. Koji od njih ukazuje na početni region replikacije?

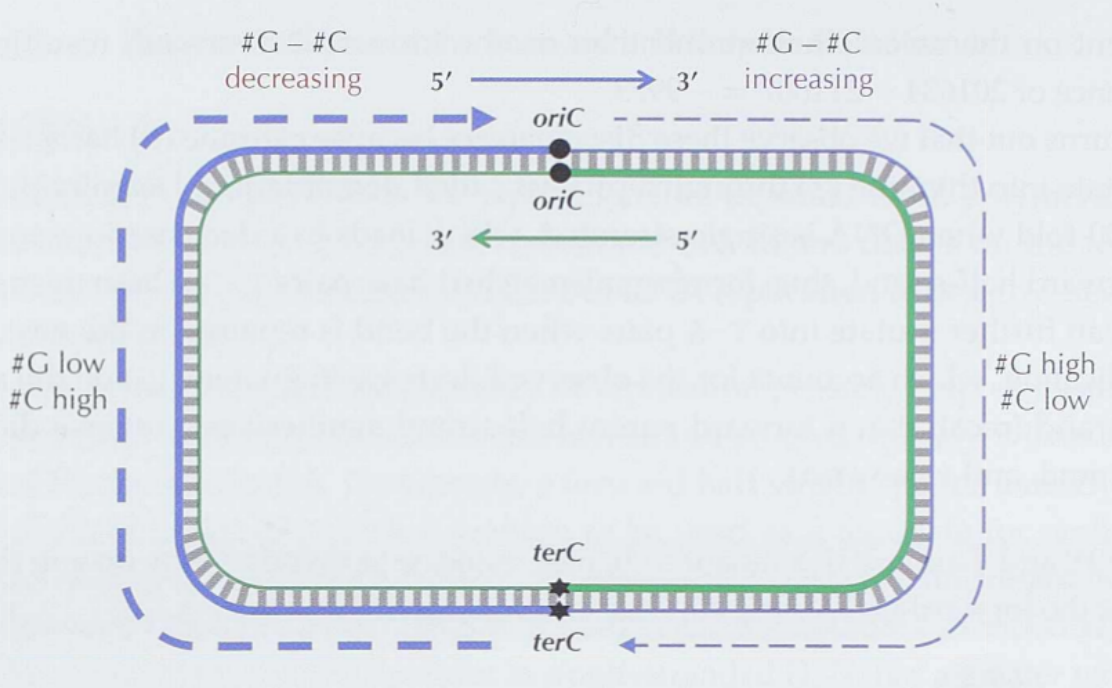
Iskrivljeni dijagrami

S obzirom na to da imamo veliku količinu statističkih podataka, pitamo se kako ih možemo upotrebiti da bismo došli do lokacije *oriC*-a? U tome nam mogu pomoći **iskrivljeni dijagrami**(engl. *skew diagram*). Osnovna ideja je da prođemo kroz genom i da računamo razliku između količine guanina(G) i citozina(C). Ako ova razlika raste, onda možemo pretpostaviti da se krećemo niz polulanac koji ide na desno (u nastavku samo polulanac, smer $5' -> 3'$), a ako razlika počne da se smanjuje, onda pretpostavljamo da smo na obrnutom polulancu ($3' -> 5'$). Zbog procesa koji se naziva deaminacija (gubljenje aminokiselina), svaki polulanac ima manjak citozina u poređenju sa guaninom, a svaki obrnuti polulanac ima manjak guanina u odnosu na citozin.

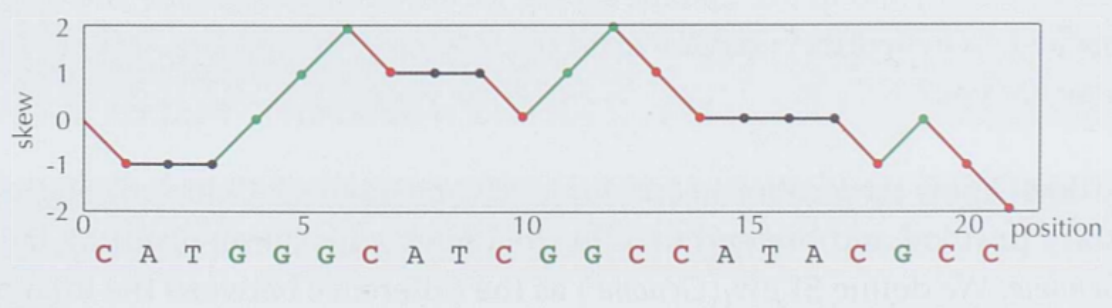
Posmatrajmo iskrivljeni dijagram bakterije Ešerihija Koli. Lako uočavamo minimalnu vrednost skew dijagrama.

Minimalna vrednost iz iskrivljenog dijagrama ukazuje baš na ovaj region:

Slika 1.11: Prikaz kretanja.



Slika 1.12: Iskrivljeni dijagram genoma Genome = CATGGGCATCGGCCATACGCC.

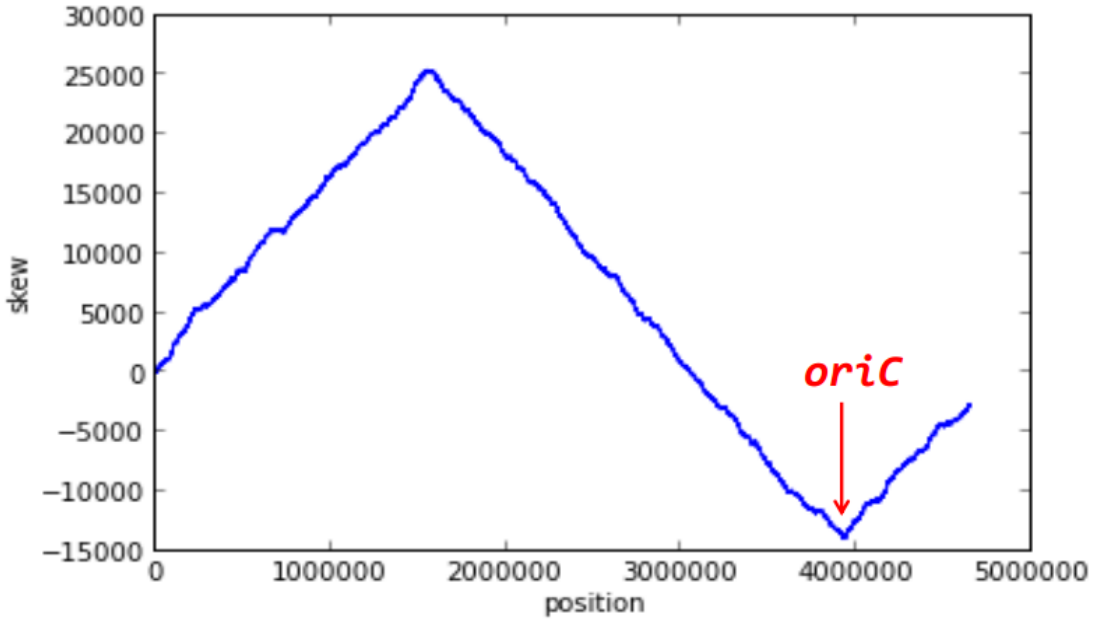


Slika 1.14: Region na koji pokazuje minimalna vrednost iskrivljenog dijagrama Ešerihije koli.

```
aatgatgatgacgtcaaaaggatccggataaaacatggtgattgcctcgcataacgcggta
tgaaaatggattgaagcccgggccgtggattctactcaactttgtcggcttgagaaagacc
tgggatcctgggtattaaaaagaagatctatttatttagagatctgttctattgtgatctc
ttattaggatcgcactgccctgtggataacaaggatccggcttttaagatcaacaacctgg
aaaggatcattaactgtgaatgatcggtgatcctggaccgtataagctgggatcagaatga
ggggttatacacaactcaaaaactgaacaacagttgttctttggataactaccggttgatc
caagcttcctgacagagttatccacagtagatcgcacgatctgtatacttatttgagtaaa
ttaacccacgatcccagccattcttctgccggatcttccggaatgtcgtgatcaagaatgt
tgatcttcagtg
```

Uočimo da u ovom regionu nema čestih 9-grama (koji se pojavljuju 3 ili više puta). Iz toga zaključujemo da, iako smo uspeli da nađemo *oriC* bakterije Ešerihija koli, nismo uspeli da nađemo

Slika 1.13: Iskrivljeni dijagram Ešerihije koli.



DNKA boksove. Međutim, pre nego što odustanemo od potrage, osmotrimo još jednom *oriC* Vibrio colerae, kako bismo pokušali da nađemo način da izmenimo naš algoritam i uspemo da lociramo DNKA boksove u Ešerihiji koli. Veoma brzo, može se uvideti da osim tri pojavljivanja ATGATCAAG i tri pojavljivanja CTTGATCAT, *oriC* Vibrio cholerae sadrži i dodatna pojavljivanja ATGATCAAC i CATGATCAT koji se razlikuju samo u jednom nukleotidu od gornjih niski. Ovo još više povećava šanse da smo naišli na prave DNKA boksove, a ima i biološkog smisla. Naime, DNKA se može vezati i za nesavršene DNKA boksove, one koji se razlikuju u nekoliko nukleotida.

Slika 1.15: Prikaz pojavljivanja nesavršenih niski nukleotida.

atca**ATGATCAAC**gtaagcttctaagc**ATGATCAAG**gtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaag**ATGATCAAG**agaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaag**CATGATCAT**ggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacct**CTTGATCAT**cgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagct**CTTGATCAT**gtt
tccttaaccctctatttttacggaaga**ATGATCAAG**ctgctgct**CTTGATCAT**cgtttc

Cilj nam je da sada izmenimo algoritam čestih reči (FrequentWords) tako da možemo da pronađemo DNKA boksove koji su predstavljeni čestim k-gramima, sa mogućim izmenama na pojedinim nukleotidima. Ovaj problem nazvaćemo problem čestih reči sa propustima.

Problem čestih reči sa propustima. Pronaći najčešće k-grame sa propustima u niski karaktera.
Ulaz: Niska Text i celi brojevi k i d.
Izlaz: Svi najčešći k-grami sa najviše d propusta u niski Text.

Pokušajmo, još jednom, sa pronalaskom DNKA boksova kod Ešerihije koli, tako što ćemo naći najčešće 9-grame sa propustima i komplementima u regionu *oriC* koji nam je predložen minimalnom vrednošću iskrivljenog dijagrama. Pokušaćemo sa malim prozorom koji ili počinje ili se završava ili je centriran na poziciji najmanje iskrivljenosti. Ovakvim izvođenjem pronalazimo TTATCCACA/TGTGGATAA kao najčešći 9-gram. Međutim, ovo nije jedini 9-gram. Za ostale 9-grame još uvek ne znamo čemu služe, ali znamo da nose skrivene informacije, da se grupišu unutar genoma i da većina njih nema veze sa replikacijom.

Slika 1.16: Prikaz pronađenih niski sa propustima i komplementima u *oriC* regionu Ešerihije koli.

aatgatgatgacgtcaaaaggatccggataaaacatggtgattgcctcgcataacgcgg
tatgaaaatggattgaagcccgggccgtggattctactcaactttgtcggcttgagaaa
gacctgggatcctgggtattaaaaagaagatctatttatttagagatctgttctattgt
gatctcttattaggatcgcactgcccTGTGGATAAcaaggatccggcttttaagatcaa
caacctggaaaggatcattaactgtgaatgatcggtgatcctggaccgtataagctggg
atcagaatgaggggTTATACACAactcaaaaactgaacaacagttgttcTTTGGATAAC
taccggttgatccaagcttcctgacagagTTATCCACAgtagatcgcacgatctgtata
cttatttgagtaaattaacccacgatcccagccattcttctgccggatcttccggaatg
tcgtgatcaagaatgttgatcttcagtg

1.3 Zadaci sa vežbi

U nastavku će biti predstavljeni zadaci sa vežbi na kursu rađeni u programskom jeziku Python.

1.3.1 FrequentWords

```
#frequent_words
def pattern_count(text, pattern):
    count = 0
    k = len(pattern)
    for i in range(len(text) - k):
        if text[i:i+k] == pattern:
            count += 1
    return count

def frequent_words(text, k, min_count):
    frequent_patterns = set([])
    count = []
    n = len(text)-k
    for i in range(n):
        # Izvuci podnisku koji pocinje na $i$-toj poziciji i ima $k$ karaktera
        pattern = text[i:i+k]
        count.append(pattern_count(text, pattern))
    max_count = max(count)
    if max_count < min_count:
        return []
    for i in range(n):
        if count[i] == max_count:
            frequent_patterns.add(text[i:i+k])
    return frequent_patterns

def main():
    print(frequent_words('agctagatgctagctagctgatcgagctgatgcaggcagtgctagc', 4,
    ↪ 2))

if __name__ == "__main__":
    main()
```

1.3.2 Faster FrequentWords

```
#faster frequent_words
def pattern_to_number(pattern):
    if len(pattern) == 0:
        return 0
    last = pattern[-1]
    prefix = pattern[:-1]
    return 4 * pattern_to_number(prefix) + symbol_to_number(last)

def number_to_pattern(n, k):
    if k == 1:
        return number_to_symbol(n)
    prefixIndex = n // 4 #celobrojno deljenje
    r = n  %  4
```

```
    symbol = number_to_symbol(r)
    prefix  = number_to_pattern(prefixIndex, k-1)
    return prefix + symbol

def symbol_to_number(c):
    pairs = {
        'a' : 0,
        't' : 1,
        'c' : 2,
        'g' : 3
    }
    return pairs[c]

def number_to_symbol(n):
    pairs = {
        0 : 'a',
        1 : 't',
        2 : 'c',
        3 : 'g'
    }
    return pairs[n]

def pattern_count(text, pattern):
    count = 0
    k = len(pattern)
    for i in range(len(text) - k):
        if text[i:i+k] == pattern:
            count += 1
    return count

def computing_frequencies(text,k):
    frequency_array = [0 for i in range(4**k)] #4 ^ k
    for i in range(len(text) - k):
        pattern = text[i:i+k]
        j = pattern_to_number(pattern)
        frequency_array[j] += 1
    return frequency_array

def faster_frequent_words(text, k, min_count):
    frequent_patterns = set([])
    frequency_array = computing_frequencies(text, k)
    max_count = max(frequency_array)
    if max_count < min_count:
        return []
    for i in range(4**k):
        if frequency_array[i] == max_count:
            pattern = number_to_pattern(i, k)
            frequent_patterns.add(pattern)
    return frequent_patterns

def main():
    print(faster_frequent_words('agctagatgctagctagctgatcgagctgatgcaggcagtgctagc
    ↪ ', 4, 2))
```

```
    #print(number_to_pattern(pattern_to_number('ta'),2))

if __name__ == "__main__":
    main()
```

1.3.3 Skew Diagram

```
#GC-skew
import matplotlib.pyplot as plt

def draw_skew(skew):
    x = [i for i in range(len(skew))]
    ax = plt.subplot()
    ax.plot(x, skew)
    plt.show()

def calculate_skew(text):
    skew = [0 for c in text]
    last = 0
    for i in range(0, len(text)):
        if text[i] == 'g':
            skew[i] = last + 1
        elif text[i] == 'c':
            skew[i] = last - 1
        else:
            skew[i] = last
        last = skew[i]
    return skew


def main():
    text = "catgggcatcggccatacgcc"
    print(calculate_skew(text))
    draw_skew(calculate_skew(text))

if __name__ == "__main__":
    main()
```

1.3.4 FrequentWords With Mismatches

```
# Prevodjenje nukleotida u brojeve
def symbol_to_number(c):
    pairs = {
        'a' : 0,
        't' : 1,
        'c' : 2,
        'g' : 3
    }

    return pairs[c]

# Prevodjenje nukleotida u brojeve
def number_to_symbol(n):
    pairs = {
```

```
        0 : 'a',
        1 : 't',
        2 : 'c',
        3 : 'g'
    }

    return pairs[n]

# Prevodjenje broja u odgovarajucu nukleotidnu sekvencu
def number_to_pattern(n, k):
    if k == 1:
        return number_to_symbol(n)

    prefix_index = n // 4
    r = n %  4
    c = number_to_symbol(r)
    prefix_pattern = number_to_pattern(prefix_index, k - 1)

    return prefix_pattern + c



# Prevodjenje nukleotidne sekvence u odgovarajuci broj
def pattern_to_number(pattern):
    if len(pattern) == 0:
        return 0

    last = pattern[-1:]
    prefix = pattern[:-1]

    return 4 * pattern_to_number(prefix) + symbol_to_number(last)



# Hamingova distanca, broj pozicija karaktera na kojima se tekstovi 1 i 2
    ↪ razlikuju,
# podrazumeva se da je duzina obe niske jednaka
def hamming_distance(text1, text2):
    distance = 0

    for i in range(len(text1)):
        if text1[i] != text2[i]:
            distance += 1

    return distance



# Brojanje pojavljivanja podsekvenci u tekstu koje se od uzorka razlikuju na
    ↪ najvise d pozicija
def approximate_pattern_count(text, pattern, d):
    count = 0
```

```
    for i in range(len(text) - len(pattern)):
        pattern_p = text[i:i+len(pattern)]

        if hamming_distance(pattern, pattern_p) <= d:
            count += 1

    return count


# Pronalazenje svih niski susednih zadatom uzorku sa razlikama na najvise d
    ↪ pozicija
def neighbors(pattern, d):
    if d == 0: # Ako je dozvoljena greska jednaka nuli onda je samo uzorak svoj
    ↪  sused bez gresaka
        return set([pattern])

    # Izlaz iz rekurzije:
    if len(pattern) == 1: # Ako je duzina uzorka jednaka 1, a dozvoljena greska
    ↪  je veca od nule, onda je moguce iskoristiti bilo koji karakter
        return set(['a', 't', 'c', 'g'])

    neighborhood = set([])

    suffix_neighbors = neighbors(pattern[1:], d) # Pronalaze se svi susedi
    ↪ duzine n-1

    for text in suffix_neighbors:
        if hamming_distance(pattern[1:], text) < d: # Ako se sused razlikuje na
    ↪  manje od d pozicija od podniske uzorka bez prvog karaktera
            for x in ['a','t','c','g']:
                neighborhood.add(x + text) # Moguce je dodati bilo koji
    ↪ karakter na pocetak suseda i time dobiti najvise d razlika
        else: # U suprotnom, sused se vec razlikuje na d pozicija od uzorka pa
    ↪ je dozvoljeno samo dodavanje ispravnog karaktera kako se
            neighborhood.add(pattern[0] + text) # razlika ne bi povecala preko
    ↪ d

    return neighborhood



def frequent_words_with_mismatches(text, k, d):
    frequent_patterns = set([])

    close = [0 for i in range(4**k)] # Kandidati za proveru
    frequency_array = [0 for i in range(4**k)]

    # Za svaki uzorak duzine k u tekstu evidentiraju se kandidat susedi cija
    ↪ pojavljivanja treba uzeti u razmatranje (niske koje se razlikuju od
    ↪ uzorka na najvise d pozicija)
    for i in range(len(text) - k):
        neighborhood = neighbors(text[i:i+k], d) # Pronalaze se kandidati
```

```
        for pattern in neighborhood:
            index = pattern_to_number(pattern) # Svaka kandidat sekvenca se
    ↪ prevodi u svoj odgovarajuci indeks
            close[index] = 1 # i evidentira

    # Za svaku kandidat sekvencu (koja je do sada pronadjena u tekstu sa
    ↪ greskom d), broji se pojavljivanje njenih d-suseda
    for i in range(4**k):
        if close[i] == 1:
                pattern = number_to_pattern(i, k)
                frequency_array[i] = approximate_pattern_count(text, pattern, d
    ↪ )

    max_count = max(frequency_array)

    # Pronalaze se one sekvence duzine k cija su pojavljivanja najzastupljenija
    for i in range(4**k):
        if frequency_array[i] == max_count:
            pattern = number_to_pattern(i, k)
            frequent_patterns.add(pattern)

    return frequent_patterns



def main():
    print(frequent_words_with_mismatches('
    ↪ tgactatcatcgtagtatcgatgtgcacacacgtgcgcgcgcgcgccctgtacatgatc', 5, 2))

if __name__ == "__main__":
    main()
```