Bioinformatika - sekvencioniranje genoma

Petrović, Josip

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku

Josip Petrović

Bioinformatika - sekvencioniranje genoma

Završni rad

Osijek, 2019.

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku

Josip Petrović

Bioinformatika - sekvencioniranje genoma

Završni rad

Mentor: izv. prof. dr. sc. Domagoj Matijević

Osijek, 2019.

Sažetak

U ovom radu ćemo se upoznati sa osnovama sekvencioniranja genoma i ulogom bioinformatike u samom procesu sekvencioniranja. Rad je podijeljen na dva poglavlja. U prvom poglavlju navodimo pojmove iz teorije grafova koji su nam potrebni za razumijevanje bioinformatičkih algoritama za sekvencioniranje. Drugo poglavlje se odnosi na sam postupak sastavljanja genoma iz danih očitanja. Na kraju rada ćemo pokazati primjer slaganja sekvence u programskom jeziku Python.

Ključne riječi

bioinformatika, genom, DNA, graf, Eulerova tura, k-torke, (k,d)-torke, preklapajući graf, de Bruijnov graf

Abstract

In this paper, we will introduce the basics of genome sequencing and the role of bioinformatics in the sequencing process itself. The paper is divided into two chapters. In the first section, we outline the concepts in graph theory we need to understand bioinformatics sequencing algorithms. The second chapter deals with the genome assembly process itself from the given reads. At the end of the paper we will show an example of sequence assembly in Python programming language.

Key words

bioinformatics, genome, DNA, graph, Eulerian cycle, k-mers, (k,d)-mers, overlap graph, de Bruijn graph

Sadržaj

Uvod

Genomika je grana genetike koja primjenjuje metode DNA sekvenciranja i bionformatike u cilju sekvenciranja, sastavljanja i analize funkcija i strukture genoma. Sekvencioniranje općenito je metoda kojom utvrđujemo sastav i redoslijed elemenata u nekoj molekuli, a sekvencioniranje genoma nekog organizma je proces određivanja cjelokupne DNA tog organizma, pri čemu mislimo na određivanje točnog slijeda svih nukleotida te DNA. Nukleotidi su osnovne strukturne jedinice DNA i RNA, a razlikujemo ih prema nukleinskim bazama koje su dio njihovog sastava: adenin (A), gvanin (G), citozin (C) i timin (T) , odnosno uracil (U) ako je u pitanju RNA.

Prvi ljudski genom sastavljen je 2003. godine, a taj projekt su zajednički pokrenuli Ministarstvo energetike i Nacionalni institut za zdravlje vlade SAD-a 1990. godine. Tim projektom je pokazano da ljudski genom sadrži oko 3 milijarde parova baza (A, C, G i T). Cijena projekta je iznosila preko 3 milijarde američkih dolara te je potakla razvoj metoda za sekvencioniranje čije su cijene znatno manje od spomenute.

Obzirom da je sva genetska informacija sadržana u DNA (ili u RNA kod nekih virusa), imamo brojne koristi od određivanja njenog slijeda: razumijevanje nasljednih bolesti, infekcija, tumora itd. Također pomaže u sintezi novih lijekova, utvrđivanju srodnosti i evoluciji vrsta.

Moderne metode očitavaju ograničen broj nukleotida. Zbog toga treba lance DNA podijeliti na manje prije samog očitavanja. Najpopularnija metoda je tzv. *shotgun* sekvencioniranje. Tom metodom se DNA lomi na manje fragmente na slučajan način koji se zatim povezuju i nastaju očitanja (eng. reads). Na kraju se, uz pomoć računalnih programa, preklapajući krajevi tih očitanja koriste da bi ih sastavili u veće cjeline.

U ovom radu ćemo se primarno baviti računalnim dijelom sastavljanja genoma. U prvom poglavlju navodimo osnovne pojmove iz teorije grafova, jer su grafovi temeljne strukture na kojima djeluju algoritmi za rekonstrukciju genoma. Drugo poglavlje se bavi rastavljanjem genoma na očitanja, njegovim ponovnim sastavljanjem iz tih očitanja i objašnjavanjem algoritama koji se koriste u samom procesu. U posljednjem poglavlju ćemo primjeniti spomenute metode i algoritme sekvencioniranja na genomu bakterije *Carsonelle ruddii* i prikazati neke zanimljive rezultate dobivene tim projektom. Teorijski dio rada je napravljen po uzoru na [1], iz kojeg je preuzeta i većina slika drugog poglavlja ovog rada.

1. Grafovi

1.1. Osnovni pojmovi iz teorije grafova

Grafovi su jedna od osnovnih matematičkih struktura. Mogu se pojaviti u raznim oblicima. Međutim, nama nije bitan sâm prikaz grafa, nego njegovi vrhovi i bridovi kojima su ti vrhovi povezani. Grafovi mogu predstavljati razne pojave, npr. vrhovi u grafu mogu predstavljati ključne točke u nekom gradu, dok su bridovi putevi koji ih povezuju.

Moderni algoritmi za sastavljanje genoma temelje se na algoritmima nad grafovima. Zbog toga ćemo navesti nekoliko važnih pojmova iz teorije grafova koji će nam biti neophodni za nastavak priče o sekvencioniranju genoma.

Definicija. *Graf je uređen par $G = (V, E)$, gdje je $\emptyset \neq V = V(G)$ skup vrhova, $E = E(G)$ skup bridova disjunktnih s V. Svaki brid $e \in E$ spaja dva vrha u, $v \in V$ koji se zovu krajevi od e. Tada su vrhovi u i v incidentni s e, a vrhovi u i v susjedni i pišemo e=uv.*

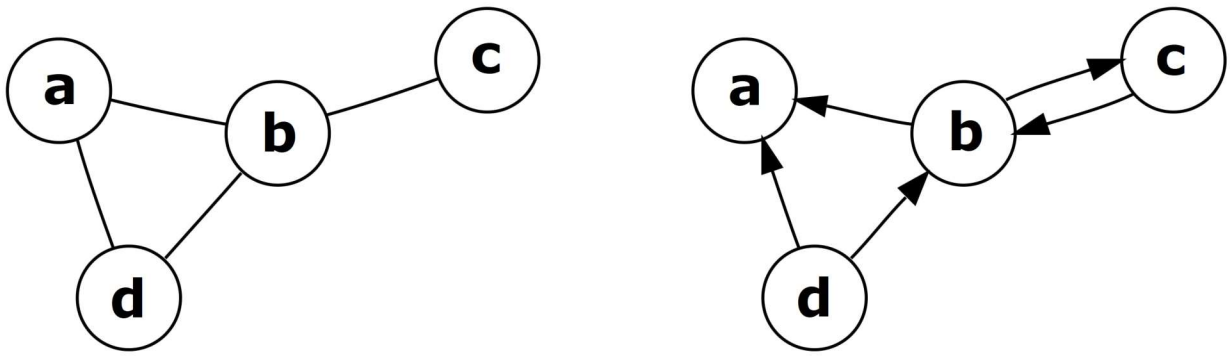
Brid čiji se krajevi podudaraju zove se *petlja*, a ako su krajevi različiti naziva se pravi brid.

Stupanj vrha je broj bridova incidentan s tim vrhom, pri čemu petlju smatramo kao dva brida incidentna s tim vrhom.

Graf je *jednostavan* ako nema petlji ni višestrukih bridova.

Jednostavan graf u kojem je svaki par vrhova spojen bridom naziva se *potpun graf*.

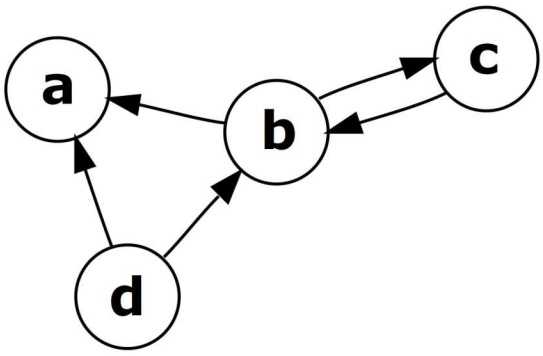
Usmjereni graf D je graf G u kojem svaki brid ima smjer od početka prema kraju. D se još zove orijentacija od G i pišemo $D = \overrightarrow{G}$. Brid s početkom u u, a krajem u v je uređen par (u, v) i katkad pišemo $u \to v$.



Neusmjereni graf Usmjereni graf

Slika 1: Primjeri neusmjerenog i usmjerenog grafa sa 4 vrha.

Graf je najjednostavnije prikazati matricom susjedstva. U toj matrici redovi i stupci predstavljaju vrhove grafa. Kod neusmjerenog grafa, križanjem retka i i stupca j dobijemo polje koje predstavlja broj bridova između vrhova i i j, dok kod usmjerenog grafa dobijemo polje koje predstavlja broj usmjerenih bridova čiji je početak u vrhu i, a kraj u vrhu j.



	a	b	c	d
a	0	0	0	0
b	1	0	1	0
c	0	1	0	0
d	1	1	0	0

Slika 2: Usmjereni graf sa pripadnom matricom susjedstva.

1.2. Šetnje i putovi u grafu

Definicija. *Šetnja u grafu G je niz $W := v_0, e_1, v_1, e_2, \ldots, e_k, v_k$, čiji članovi su naizmjence vrhovi v_i i bridovi e_i, tako da su krajevi od e_i vrhovi v_{i-1} i v_i, $1 \leq i \leq k$.*

U jednostavnom je grafu *šetnja* potpuno određena samo nizom svojih vrhova $v_0, v_1, \ldots, v_k$. Kažemo da je v_0 početak, a v_k kraj šetnje W. Vrhovi $v_1, v_2, \ldots, v_{k-1}$ su unutarnji vrhovi šetnje, a broj k se zove duljina šetnje W.

Ako su svi bridovi $e_1, \ldots, e_k$ šetnje W medusobno različiti, onda se W zove *staza.*

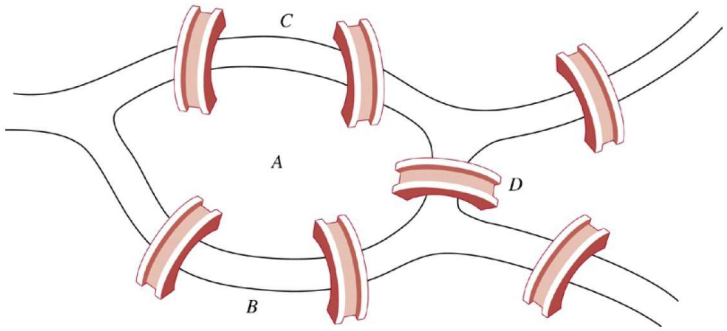
Ako su na stazi W svi vrhovi $v_1, \ldots, v_k$ medusobno različiti, šetnja se zove *put.*

Zatvorena staza pozitivne duljine čiji su vrhovi medusobno različiti (osim krajeva) zove se *ciklus.*

Dva vrha u, v grafa G su povezana ako postoji (u, v)-put u G. Udaljenost $d_G(u, v)$ dvaju vrhova u i v u grafu G je duljina najkraćeg (u, v)-puta u G, pri čemu duljinom puta zovemo duljinu odgovarajuće šetnje.

1.3. Eulerova tura

Jedan od utemeljitelja teorije grafova je švicarski matematičar Leonhard Euler (1707.-1783.). On je prvi rješio problem Königsberških mostova dokazavši da se gradom ne može prošetati na način da se svaki od sedam mostova prijeđe samo jednom. Zbog toga su po njemu nazvane neke specijalne šetnje po grafu.



Slika 3: Sedam mostova Königsberga

Definicija. *Neka je G graf. Eulerova staza od G je staza koja sadrži sve bridove od G. Eulerova tura je zatvorena Eulerova staza.*

Za graf G kažemo da je *Eulerov* ako sadrži Eulerovu turu.

U nastavku priče o sekvencioniranju genoma od ključne će nam važnosti biti algoritmi nad usmjerenim grafovima. Pri tome će nam biti važno prepoznati kada će graf imati Eulerovu turu. Iz već navedene tvrdnje, svaki Eulerov graf ima Eulerovu turu. Pitanje koje slijedi je: Kada je graf Eulerov?

Prije nego odgovorimo na to pitanje, bitno je navesti još dvije klasifikacije usmjerenih grafova. Za vrh v definiramo ulazni i izlazni stupanj, a to su brojevi bridova s krajem, odnosno početkom u v.

Za vrh v kažemo da je *uravnotežen* ako mu je izlazni stupanj jednak ulaznome. Za usmjereni graf kažemo da je *uravnotežen graf* ako su mu svi vrhovi uravnoteženi.

Za usmjereni graf kažemo da je *povezan* ako za njega vrijedi da se iz svakog čvora može doći u svaki drugi čvor.

Teorem (Eulerov teorem za usmjerene grafove)**.** *Svaki uravnotežen i povezan usmjeren graf je Eulerov graf.*

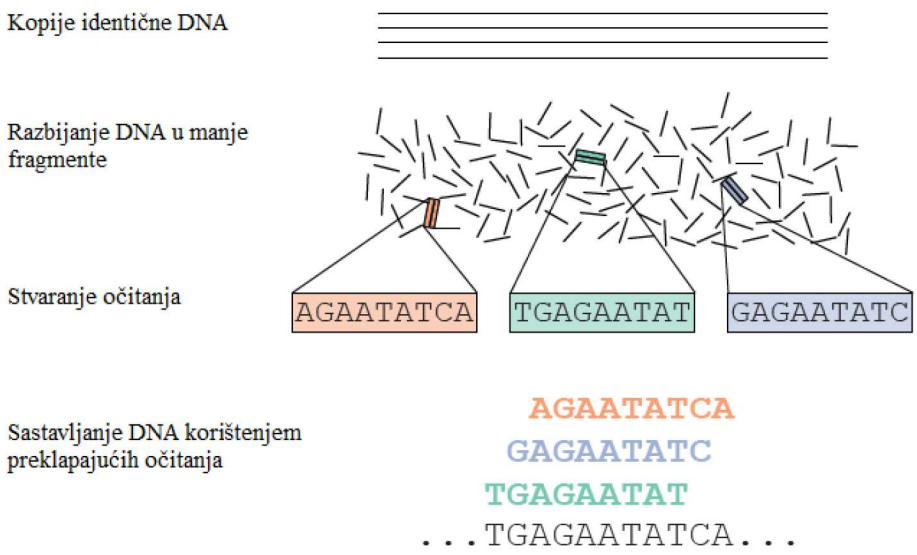
U drugom poglavlju ćemo pokazati kako u svakom uravnoteženom i povezanom usmjerenom grafu možemo konstruirati Eulerovu turu, što povlači da je graf Eulerov, a to će nam poslužiti i kao svojevrsni dokaz teorema (str. 13).

Sve ostale tvrdnje i pojmove vezane za grafove koji će se pojavljivati u nastavku teksta ćemo zbog praktičnosti definirati u samome tekstu.

2. Sastavljanje genoma

2.1. Problemi sastavljanja

Ljudski genom se sastoji od 3 milijarde nukleotida (A, C, T, G), dok neki organizmi imaju i višestruko veće genome. Biolozi danas nemaju tehnologiju s kojom bi mogli očitati cijeli genom od početka do kraja, nego lome DNA na manje fragmente i tako nastaju *očitanja*. Međutim, kao i u problemu s novinama, znanstvenici nemaju samo jednu kopiju DNA, već se u stanicama uzorka (npr. iz krvi) se nalaze milijuni kopija identične DNA. Kad se oni razlome i nastanu očitanja, ne znamo iz kojeg dijela genoma dolaze ta očitanja, te moramo koristiti njihove preklapajuće dijelove da bi rekonstruirali DNA (slika 4).



Slika 4: Primjer *shotgun* sekvencioniranja

Postoje još neki problemi u postupku sekvencioniranja. Neki dijelovi genoma nisu uopće pokriveni, tj. ne postoje očitanja u tim regijama genoma. Nadalje, tehnologija za sekvencioniranje nije savršena pa očitanja koja dobijemo nerijetko sadrže pogreške. Mi ćemo pretpostaviti da nemamo ovakvih problema radi lakšeg razumijevanja priče o sekvencioniranju.

2.2. Rekonstrukcija genoma

Očitanja koja dobivamo modernim strojevima za sekvencioniranje su uglavnom jednake duljine, stoga možemo pretpostaviti da su sva očitanja nekakve k-torke, za neki $k \in \mathbb{N}$. Uzmimo, na primjer, slijed nukleotida TATGGGGTGC. Obzirom na to da ćemo se baviti sekvencioniranjem uz pomoć računalnih tehnika, TATGGGGTGC možemo gledati kao *string* u bilo kojem programskom jeziku.

Prvi zadatak je jednostavan. Trebamo iz danog stringa pokupiti sve podstringove duljine k. Uzmimo da je $k = 3$. Tada ćemo, na primjeru TATGGGGTGC, dobiti sljedeće podstringove duljine 3: ATG, GGG, GGG, GGT, GTG, TAT, TGC i TGG. Primjetite da smo dobivene 3-torke poredali leksikografski. Naime, kad uz pomoć uređaja za sekvencioniranje dobijemo očitanja, ona neće biti poredana onako kako trebaju biti u samom genomu.

Pogledajmo sada obratni problem, gdje iz dobivenih k-torki trebamo rekonstruirati string. Uzmimo za primjer sljedeće 3-torke: AAT, ATG, GTT, TAA i TGT. Prirodno je pokušati povezati dvije k-torke na način da usporedimo njihovih $k-1$ simbola. Na primjer, lako je primjetiti da će rješenje započeti s 3-torkom TAA. To je zato jer ne postoji 3-torka koja završava sa TA. Nadalje, iduća 3-torka u nizu će biti AAT, jer se zadnja 2 simbola od TAA poklapaju sa prva dva simbola od AAT i tako nastane 4-torka TAAT. Postupak nastavljamo na isti način i na kraju dobivamo string koji glasi TAATGTT:

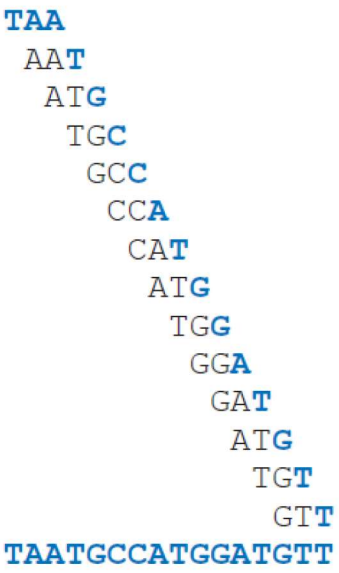
```
TAA
 AAT
  ATG
   TGT
    GTT
TAATGTT
```

Ovaj zadatak se ne čini previše teškim pa ćemo zato pogledati drugi primjer. Uzmimo sada sljedeće 3-torke:

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

Ponovno započinjemo sa TAA, za kojim slijedi AAT, a iza njega je ATG. Sljedeća 3-torka može biti: TGC, TGG ili TGT. Slučajim odabirom izabrali smo TGT. Njega može slijediti jedino GTT, ali nijedna 3-torka ne može slijediti GTT. Možemo se probati izvući iz ove situacije tako da probamo proširiti string na lijevu stranu, ali nemamo 3-torku koja završava sa TA.

Ako se vratimo korak unazad i umjesto TGT odaberemo TGC, možemon nastaviti konstrukciju stringa i na kraju dobivamo:



Međutim, ovo rješenje je također netočno jer smo iskoristili 14 od 15 zadanih 3-torki (nedostaje nam GGG) te nam je dobiveni string, ili genom, prekratak.

U ovom primjeru, problem nam stvaraju ponovljena 3-torka, a to je ATG. ATG se pojavljuje 3 puta, a nju slijede TGC, TGG ili TGT. Ovo nije velik problem kad je u pitanju skup od 15 3-torki, ali kad imamo milijune očitanja, onda je teško predvidjeti gdje će se ponovljene k-torke morati smjestiti.

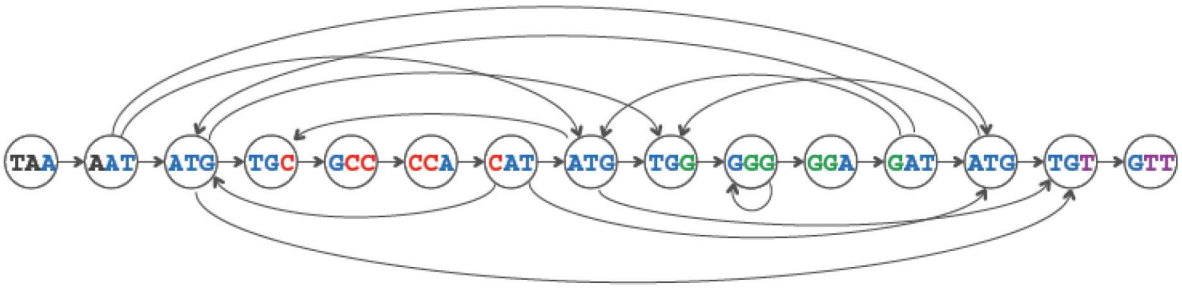
2.3. Prikaz genoma uz pomoć grafa

Kada uz pravilno preslagivanje 3-torki konačno riješimo prethodni zadatak, dobivamo slijed nukleotida **TA**ATGCCATGGGATGTT. Primjetimo da smo identične 3-torke **ATG** obojili jednako, a preostale dijelove (i međusobno) različitim bojama.

Sada ćemo uzastopne 3-torke u navedenom stringu povezati tako da tvore put u genomu:

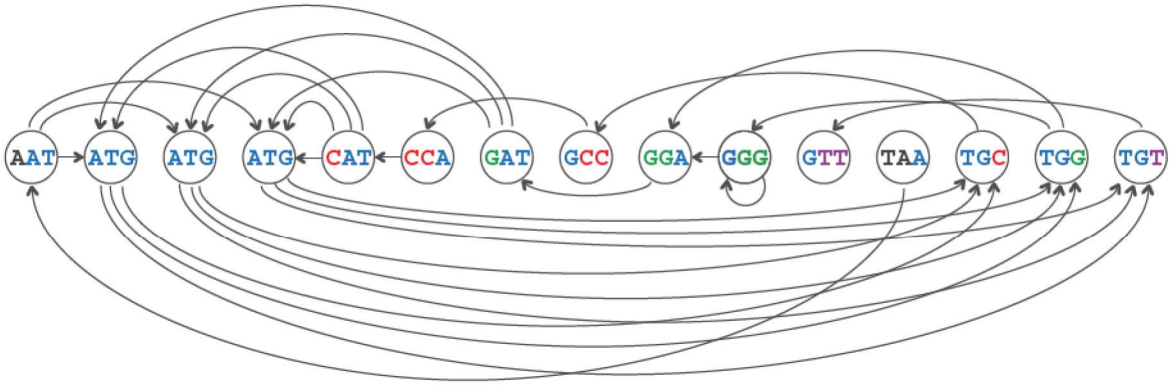
Iz ovako dobivenog puta u genomu, vrlo je lako rekonstruirati genom. Međutim, to od nas zahtjeva da mi već znamo genom unaprijed, što u stvarnosti nije moguće.

Ono što možemo primjetiti u ovom primjeru je to da su dvije 3-torke uzastopne kada se druga 2 elementa prve 3-torke preklapaju sa prva 2 elementima druge 3-torke. Te 2-torke unutar 3-torki ćemo zvati *sufiks*, ako su to druga dva elementa, i *prefiks*, ako su to prva dva elementa dane 3-torke. Na primjer, *sufiks*(**TA**A) = **A**A i *prefiks*(**TA**A) = **TA**. Konačno, **TA**A i **A**AT su uzastopne 3-torke jer vrijedi: *sufiks*(**TA**A) = *prefiks*(**A**AT) = **A**A. Sada ćemo uvesti pravilo da su dvije 3-torke povezane ako je sufiks prve jednak prefiksu druge 3-torke. U našem primjeru genoma **TA**ATGCCATGGGATGTT rastavljenom na uzastopne 3-torke ovo pravilo vrijedi samo djelomično. Naime, ako ispoštujemo pravilo u potpunosti, sve bi 3-torke naziva **ATG** trebale biti povezane sa **TGC**, **TGG** i **TGT**. Napravimo li to za sve ostale 3-torke dobit ćemo nama vrlo poznatu strukturu, usmjereni graf (slika 5).



Slika 5: Usmjereni graf sa 15 vrhova i 25 bridova

Na slici 5 možemo primjetiti da se genom još uvijek vrlo lako da pročitati, i to tako da jednostavno prošetamo od **TA**A do G**TT** slijeva na desno. Međutim, mi nećemo unaprijed znati redoslijed danih 3-torki. Najbolje što možemo jest poredati ih leksikografski, a zatim na isti način povezati 3-torke. Tako dobiveni graf zovemo *preklapajući graf* (slika 6).



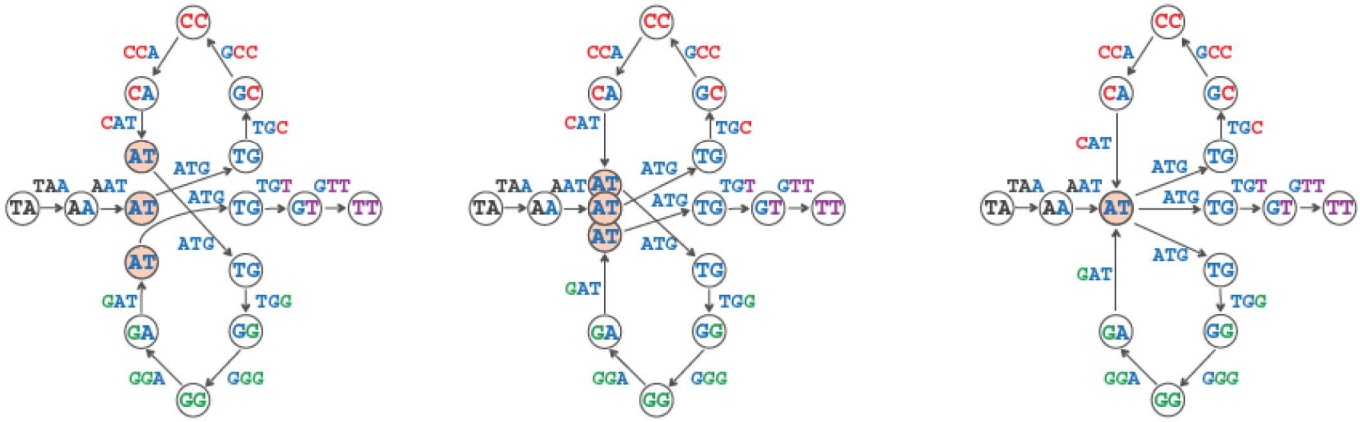
Slika 6: Preklapajući graf

Više nije tako jednostavno vidjeti konačan genom, ali on je još uvijek prisutan. Da bi ga pročitali, moramo prošetati grafom tako prođemo svakim vrhom točno jednom. Takva šetnja se naziva *Hamiltonov put*, prema matematičaru Williamu Hamiltonu (1805.-1865.). Problem koji nam se sada javlja je poprilično velik. Za mnogo veće i kompleksnije grafove od spomenutog, ni u današnje vrijeme ne postoji način da znamo postoji li uopće Hamiltonov put ili ne. To je jedan od važnijih neriješenih problema u teoriji grafova i računalnoj znanosti. Stoga ćemo krenuti drukčijim putem.

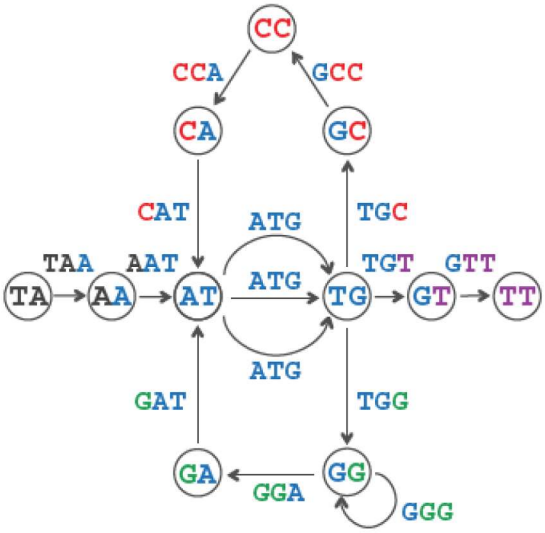
2.4. De Bruijnov graf

Vratimo se ponovno genomu **TAATGCCATGGGATGTT**. Složit ćemo novi graf, u kojemu ćemo 3-torke dodijeliti bridovima grafa, a vrhovima ćemo dodijeliti 2 nukleotida u kojima se uzastopne 3-torke preklapaju. Na primjer, vrh sa ulaznim bridom **CCA** i izlaznim bridom **CAT** ćemo nazvati **CA**. Sada **TAATGCCATGGGATGTT** prikazujemo na sljedeći način:

Ovdje nismo napravili ništa posebno drukčije. Međudim, sada ćemo početi *lijepiti* istoimene vrhove. Postupak je slijedeći: U grafu postoje 3 vrha sa parom nukleotida **AT**. Te vrhove ćemo međusobno približavati dok ne postanu jedan vrh (slika 7). Isti postupak ćemo ponoviti sa trojkom vrhova **TG** i parom **GG**. U konačnici, dobivamo zapetljanu strukturu koju zovemo *de Bruijnov graf* (slika 8), nazvanu prema nizozemskom matematičaru Nicolaasu de Brujinu (1918.-2012.).



Slika 7: Lijepljenje vrhova AT u jedan vrh



Slika 8: De Bruijnov graf

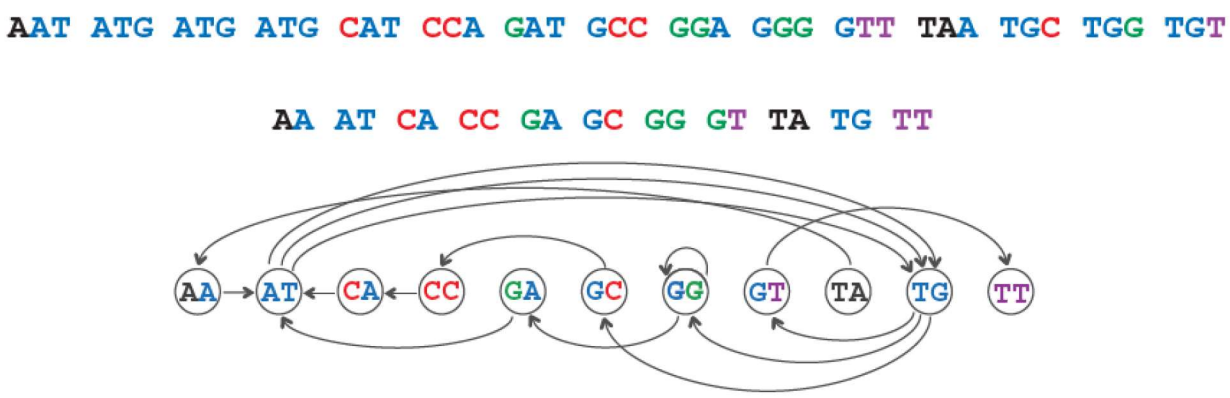
Primjetimo da smo graf koji sadrži 16 vrhova "pretvorili" u graf sa 11 vrhova, pri čemu je broj bridova ostao jednak. Vrhovi **AT** i **TG** su povezani sa 3 brida, što predstavlja ponavljanje 3-torke **ATG** u genomu 3 puta, a vrh **GG** sadrži petlju, što predstavlja pojavljivanje 3-torke **GGG** u genomu. Pogledajmo kako je to utjecalo na matricu susjedstva početnog grafa:

	TA	AA	AT_1	TG_1	GC	CC	CA	AT_2	TG_2	GG_1	GG_2	GA	AT_3	TG_3	GT	TT
TA	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AA	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
AT_1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
TG_1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
GC	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
CC	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
CA	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
AT_2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
TG_2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
GG_1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
GG_2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
GA	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
AT_3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
TG_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
GT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
TT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	TA	AA	AT	TG	GC	CC	CA	GG	GA	GT	TT
TA	0	1	0	0	0	0	0	0	0	0	0
AA	0	0	1	0	0	0	0	0	0	0	0
AT	0	0	0	3	0	0	0	0	0	0	0
TG	0	0	0	0	1	0	0	1	0	1	0
GC	0	0	0	0	0	1	0	0	0	0	0
CC	0	0	0	0	0	0	1	0	0	0	0
CA	0	0	1	0	0	0	0	0	0	0	0
GG	0	0	0	0	0	0	0	1	1	0	0
GA	0	0	1	0	0	0	0	0	0	0	0
GT	0	0	0	0	0	0	0	0	0	0	1
TT	0	0	0	0	0	0	0	0	0	0	0

Slika 9: Matrica susjedstva početnog grafa (gore) i de Bruijnovog grafa (dolje)

Opet nam se javlja početni problem, a to je da ne znamo unaprijed kako genom izgleda. Ono što imamo je skup k-torki (u našem primjeru 3-torki) koje trebamo složiti u genom. Iz tih k-torki ćemo uzeti prefikse i sufikse te ćemo ih grupirati tako da se svaka (k-1)-torka pojavljuje samo jednom. Te (k-1)-torke će predstavljati vrhove u grafu kojeg ćemo upravo složiti. Na kraju ćemo još jednom proći kroz skup k-torki i za svaku od njih stvoriti usmjereni brid u grafu čiji će početni vrh biti prefiks, a krajnji sufiks dane k-torke. Ovim postupkom smo ponovno stvorili de Bruijnov graf (slika 10).

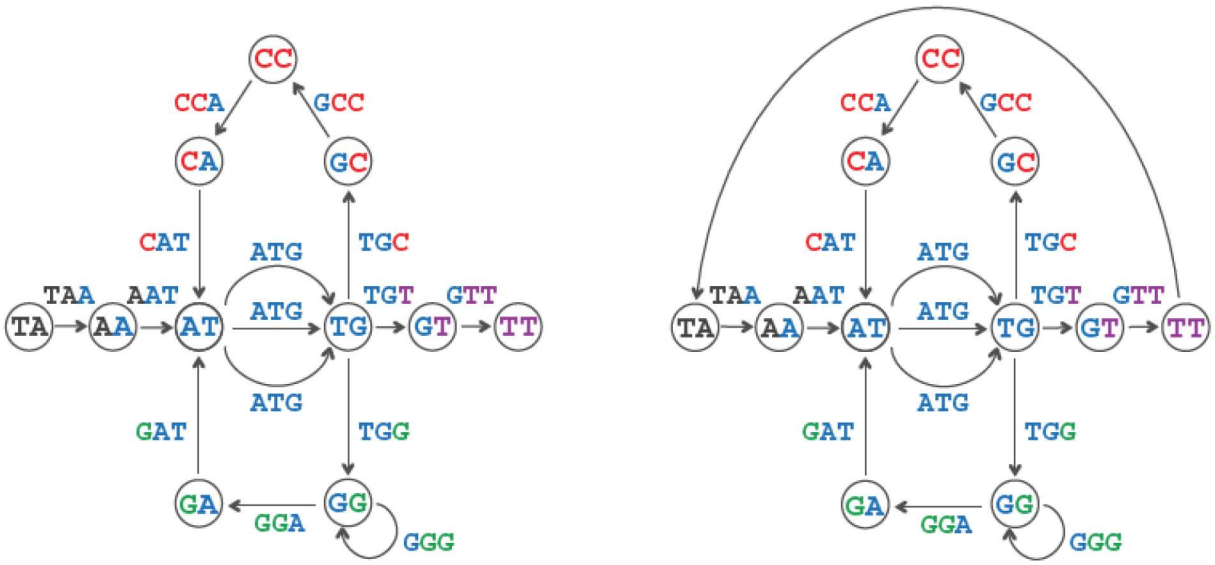


Slika 10: Skup 3-torki (gore). Skup jedinstvenih 2-torki nastalih uzimanjem prefiksa i sufiksa 3-torki (sredina). De Bruijnov graf nastao postupkom opisanim u tekstu (dolje).

2.5. Eulerova tura u de Bruijnovom grafu

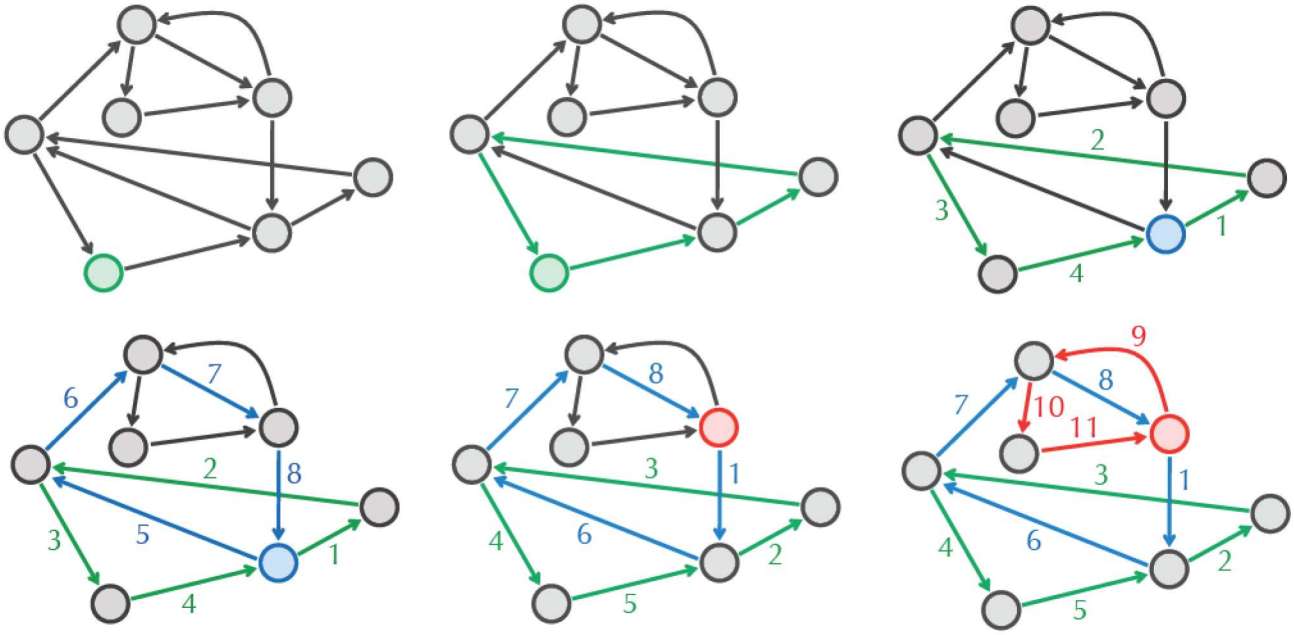
Nakon što smo sastavili de Bruijnov graf, želimo pronaći način kako iz njega iščitati genom. Ovdje zasigurno nećemo moći koristiti Hamiltonov put, jer da bi prošli svim bridovima (tj. iskoristili sve dane k-torke) moramo kroz neke vrhove proći više puta. Zato ćemo se ovdje poslužiti Eulerovom stazom (ili turom). Podsjetimo se, Eulerova staza je staza koja prolazi svakim bridom grafa točno jednom.

Najprije moramo provjeriti postoji li takva staza uopće. U prethodnom poglavlju smo naveli teorem koji nam uvelike olakšava posao (str. 4). Međutim, važno je napomenuti da za provjeru postoji li Eulerova staza ne moraju nužno svi vrhovi u grafu biti uravnoteženi. Ako postoji točno jedan vrh u kojemu je broj izlaznih bridova za jedan veći od broja ulaznih bridova i točno jedan vrh v kojemu je broj ulaznih bridova za jedan veći od broja izlaznih bridova, tada će graf sadržavati Eulerovu stazu (ali ne i turu) u kojoj će u biti početni, a v krajnji vrh staze. Ovaj graf lako možemo pretvoriti u Eulerov graf i stvoriti Eulerovu turu tako da stvorimo usmjereni brid iz vrha v u vrh u (slika 11).



Slika 11: Dodavanjem brida s kranjeg na početni vrh dobivamo uravnotežen graf.

Algoritam za pronalaženje Eulerove ture je sljedeći: Izaberemo početni vrh šetnje i nazovemo ga u_0. Nasumično šetamo grafom tako da nijedan brid ne prođemo dvaput, sve dok u jednom trenutku ne dođemo do vrha koji nema izlaznih bridova kojima možemo proći. Tako dobiveni ciklus ćemo nazvati $ciklus_0$. Jedini vrh u kojemu možemo tako zapeti je početni vrh u_0. Ako smo iskoristili sve bridove, gotovi smo, ali vrlo su male šanse da ćemo uspjeti u prvom pokušaju. U slučaju da nismo iskoristili sve bridove, pronađemo jedan vrh iz prvog ciklusa koji je incidentan sa bridom kojim nismo prošli. Taj vrh nazovemo u_1 i on će nam postati novi početni vrh. Sada ćemo, krenuvši od u_1, ponovo proći ciklusom $ciklus_0$. Kada se vratimo u u_1, iz njega ćemo izaći neiskorištenim bridom i ponovno nasumično šetati grafom. Novonastali ciklus nazovemo $ciklus_1$ i za njega sigurno znamo da je veći od $ciklus_0$. Jedini vrh u kojemu smo sad mogli zapeti je u_1. Ako ni $ciklus_1$ nije Eulerova tura, postupak ponavljamo dok ne pronađemo $ciklus_m$ koji će iskoristiti sve bridove grafa i konačno postati Eulerova tura (slika 12). Valjalo bi napomenuti da je korisno provjeriti postoji li petlja (ili više njih) u svakom vrhu te proći tim bridom prije nastavka nasumične šetnje.



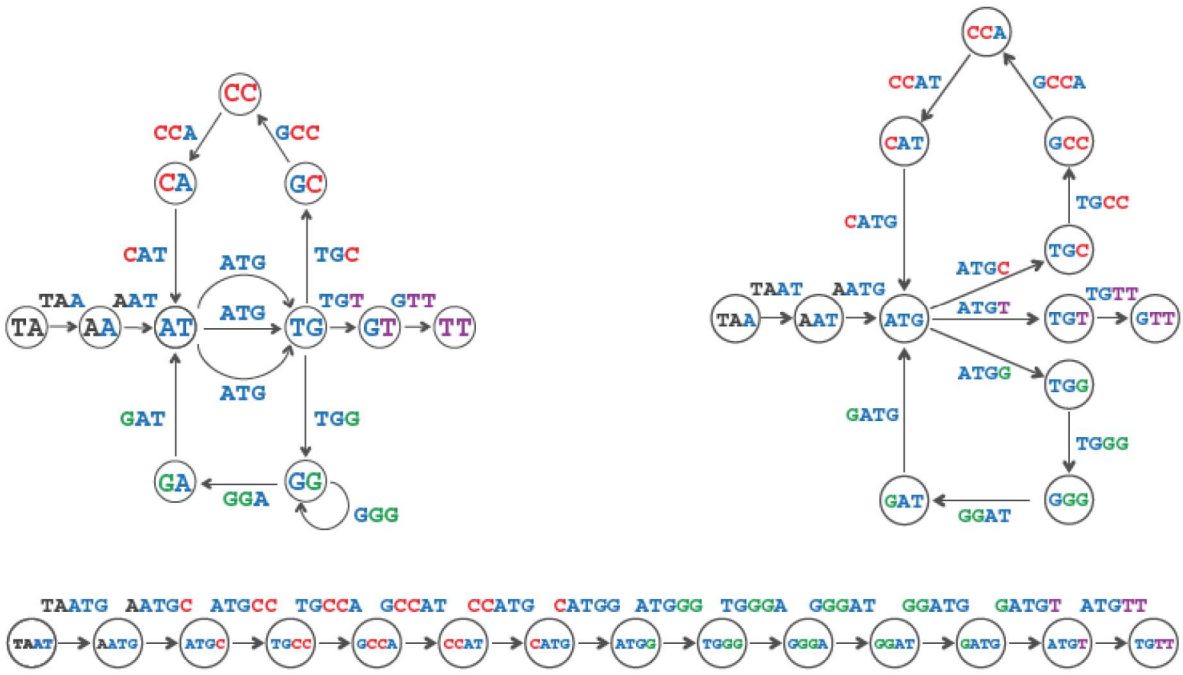
Slika 12: Stvaranje Eulerove ture postupkom opisanim u tekstu.

U slučaju da smo imali graf koji nije uravnotežen i pretvorili ga u uravnotežen dodavanjem brida, taj brid moramo kasnije ukloniti. Idealno, to je zadnji brid kojim prolazimo i uklanjanjem njega ćemo dobiti Eulerovu stazu s početkom u vrhu koji ima izlazni brid viška, dok će kraj biti vrh sa ulaznim bridom viška. Vjerojatnost da se to neće dogoditi je velika te ćemo izgubiti Eulerovu turu pri čemu nećemo dobiti Eulerovu stazu. Obzirom da dobivena Eulerova tura može početi u bilo kojem vrhu, morat ćemo je "pomaknuti" tako da počinje od nama željenog vrha.

Valjalo bi spomenuti još jedan problem u algoritmu za sekvencioniranje. Primjetite da se iz našeg primjera de Bruijnovog grafa (slika 8) genom **TAATGCCATGGGATGTT** neće dobiti kao jedinstveno rješenje. Ovdje također možemo konstruirati Eulerovu stazu koja će prikazati genom **TAATGGGATGCCATGTT**. Ne radi se o računalnoj pogrešci, nego nam nedostaju dodatne informacije o danim 3-torkama koje smo koristili za rekonstrukciju genoma.

2.6. Parovi k-torki

Kada bismo produljili dane 3-torke u 4-torke ili čak 5-torke, promijenili bi izgled de Bruijnovog grafa. On bi postajao sve raspetljaniji jer bi se manje (k-1)-torki, koje predstavljaju vrhove, poklapalo. Ako duljina (k-1)-torki postane tolika da nemamo više istoimenih vrhova, graf se pretvori u jednostavan put (slika 13).



Slika 13: Stvaranje de Bruijnovog grafa iz 3-torki (lijevo), 4-torki (desno) i 5-torki (dolje).

Ovo zvuči praktično, međutim, moderni uređaji za sekvencioniranje nam daju očitanja ograničene duljine (oko 300 nukleotida). Biolozi su ponudili rješenje u obliku *parova očitanja* (eng. read-pairs) s kojima možemo na indirektan način dobiti očitanja veće duljine. Parovi očitanja su 2 očitanja duljine k međusobno udaljeni fiksnom duljinom $d \in \mathbb{N}$ (slika 14). Njih možemo shvatiti kao jedno veliko očitanje duljine $k + d + k$ sa poznatim k-torkama na početku i na kraju i sa nepoznatom d-torkom u sredini. Ako saznamo koji su to nukleotidi koji se nalaze između dviju k-torki dobit ćemo očitanje duljine $2 \cdot k + d$. Time bismo postigli produljavanje početnih k-torki i stvaranje raspetljanijeg de Bruijnovog grafa koji preciznije rekonstruira genom.

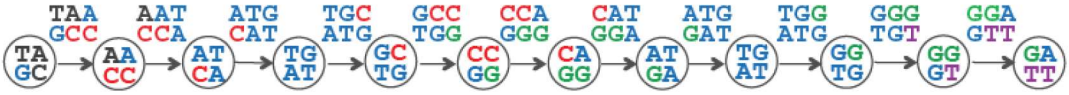
Slika 14: Parovi očitanja dobiveni iz **TA**ATGCCATGGGATGTT predstavljeni kao parovi 3-torki udaljeni za $d = 1$.

U nastavku teksta ćemo parove k-torki udaljenim za d nazivati (k,d)-torke, a te parove ćemo označavati sa (k-torka$_1$ | k-torka$_2$). Na primjer, (ATG | GGG) u genomu **TA**ATGCCATGGGATGTT predstavlja (3,4)-torku. Ako iz tog genoma izvučemo sve moguće (3,1)-torke dobivamo:

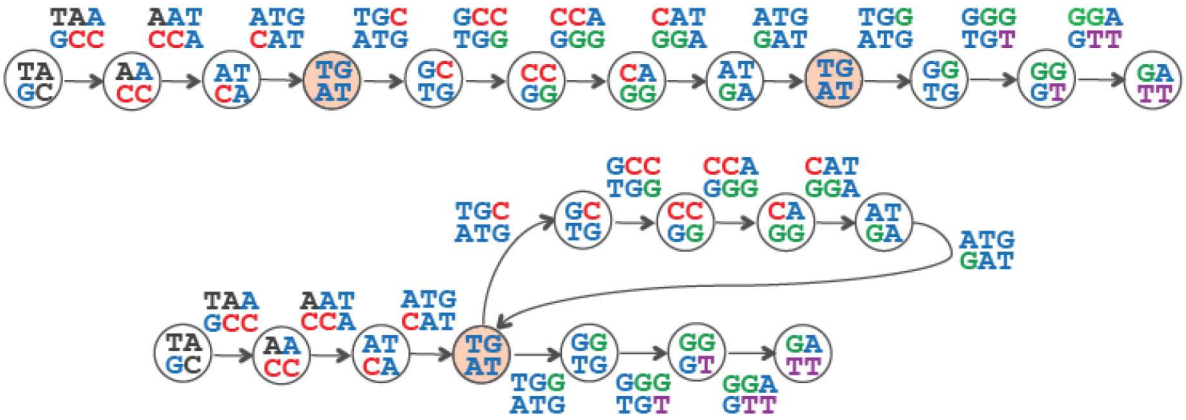
(**AA**T | CCA) (ATG | CAT) (ATG | GAT) (CAT | GGA) (CCA | GGG) (GCC | TGG)
(GGA | GTT) (GGG | TGT) (**TA**A | GCC) (TGC | ATG) (TGG | ATG)

Naravno, (3,1)-torke smo poredali leksikografski jer u stvarnom sekvencioniranju nećemo znati njihov točan redoslijed. Primjetimo da u ovom skupu ne postoje dvije identične (3,1)-torke, za razliku od samih 3-torki od kojih su se neke višestruko ponavljale.

Ponovno ćemo u priču uvesti pojmove sufiks i prefiks pri čemu će se uzimati prefiksi i sufiksi obje k-torke u (k,d)-torki. Na primjer, *prefiks*((TGC | AGA))=(TG | AG), dok je *sufiks*((TGC | AGA))=(GC | GA). Kao i prije, dvije (k,d)-torke će biti uzastopne ako je sufiks prve jednak prefiksu druge. Prikažimo **TA**ATGCCATGGGATGTT u obliku grafa čiji su bridovi (3,1)-torke spomenutog genoma, a početni i krajnji vrhovi tih bridova pripadni prefiksi, odnosno sufiksi (3,1)-torki:



Zalijepimo li istoimene vrhove ovog grafa vraćamo se de Bruijnovom grafu, ali s ponešto drukčijim izgledom nego prije (slika 15). Naime, ovdje su samo 2 vrha identična pa će de Bruijnov graf biti minimalno zapetljan. Pokušamo li pronaći Eulerovu stazu u ovakvom grafu, ona će biti jedinstvena i dat će nam genom **TA**ATGCCATGGGATGTT. To ne mora uvijek biti slučaj, ali činjenica je da će u grafu složenom uz pomoć (k,d)-torki biti manje Eulerovih staza. Problem koji nam se tu javlja je taj što neke Eulerove staze uopće neće dati legitimnu sekvencu nukleotida. Međutim, taj problem možemo riješiti.



Slika 15: De Bruijnov graf dobiven iz (3,1)-torki.

Nakon što smo dobili Eulerovu stazu u de Bruijnovom grafu sastavljenom iz (k,d)-torki, rastavit ćemo te (k,d)-torke u početne k-torke. Zatim ćemo posebno grupirati prve k-torke, a posebno druge k-torke iz parova (k-torka$_1$ | k-torka$_2$), pazeći na redoslijed. Složit ćemo 2 stringa jednake duljine, svaki iz jedne grupe k-torki. Na kraju samo trebamo provjeriti podudaraju li se ti stringovi od $(k+d)$-tog nukleotida prvog stringa do njegovog završetka sa početkom drugog stringa do njegovog $[l-(k+d)]$-tog nukleotida, pri čemu l označava duljinu tih stringova. Ako imamo više Eulerovih staza (a u praksi uvijek imamo), potrebno ih je sve iščitati i zatim gore navedenim postupkom provjeriti koja od njih daje pravo rješenje.

Iako je ova priča o sekvencioniranju genoma do neke mjere idealizirana, ona svejedno daje dobar uvid u metode kojima se bioinformatičari koriste u sekvencioniranju. Na kraju ovog rada ćemo pokazati primjer rekonstrukcije *stvarnog* genoma koristeći metode i algoritme iz teksta.

3. Primjer rekonstrukcije genoma

3.1. Rekonstrukcija uz pomoć *k*-torki

Candidatus Carsonella ruddii je bakterija koja živi endosimbiotski unutar nekih kukaca. Takav način života omogućio joj je da reducira svoj genom na svega 173904 parova nukleotida, što predstavlja najmanji genom među svim bakterijama. Iz nacionalnog centra za biotehnološke informacije (NCBI) preuzeli smo sekvencu nukleotida jednog lanca njezine DNA[4]. U ovom poglavlju ćemo rastavljati sekvencu na k-torke različitih duljina te je pokušati sastaviti metodama navedenim u prethodnom poglavlju. Zatim ćemo usporediti rezultate takvih sastavljanja i na temelju njih donijeti neke zaključke o kvaliteti bioinformatičkih algoritama koji se koriste.

Simulacija rekonstrukcije je implementirana u programskom jeziku *Python*. Na raspolaganju su nam dva modula. Prvi modul na ulazu prima $k \in \mathbb{N}$ kojim ćemo odrediti broj nukleotida na koji rastavljamo genom (k-torke). Program zatim lomi sekvencu na spomenute k-torke, stvara de Bruijnov graf sa vrhovima koji predstavljaju odgovarajuće (k-1)-torke te traži Eulerovu stazu u grafu. Na kraju će iz dobivene staze sastaviti genom. Kao što je spomenuto u prethodnom poglavlju, preciznost rekonstrukcije će ovisiti o parametru k (slika 13). Pogledajmo djelovanje programa na primjer sekvence TAATGCCATGGGATGTT. Ako za k uzmemo 3, program će rastaviti TAATGCCATGGGATGTT na $(l-k+1)$ 3-torki (TAA, AAT, ATG, TGC, GCC, CCA, CAT, ATG, TGG, GGG, GGA, GAT, ATG, TGT, GTT). De Bruijnov graf će biti predstavljen strukturom *dictionary* u Pythonu koji za ključeve uzima sve prefikse i sufikse (2-torke) navedenih 3-torki, a njima pripadne vrijednosti će biti one 2-torke koje sa ključevima čine par prefiks-sufiks u pripadnim 3-torkama. Na primjer, ključ 'GC' će imati jednu pripadnu vrijednost 'CC' (zbog GCC), dok će ključ 'TG' imati 3 pripadne vrijednosti: 'GC', 'GG' i 'GT' (zbog TGC, TGG i TGT). Na taj način smo izgubili početni poredak 3-torki i ne možemo više jasno vidjeti originalnu sekvencu. Sada postupkom navedenim u prethodnom poglavlju tražimo Eulerovu stazu u dobivenom grafu. Uz nešto sreće, dobivena staza će izgledati ovako: ['TA', 'AA', 'AT', 'TG', 'GC', 'CC', 'CA', 'AT', 'TG', 'GG', 'GG' , 'AT', 'TG', 'GT', 'TT']. Iz nje sada jednostavno možemo iščitati sekvencu TAATGCCATGGGATGTT. Ali, kao što smo već prije napomenuli, ovo rješenje nije jedinstveno pa možemo dobiti i sekvencu TAATGGGATGCCATGTT. Ako bismo za ulazni broj k uzeli 5, tada postoji jedinstvena Eulerova staza koja daje jedinstveno rješenje. Međutim, mi ćemo se baviti rekonstrukcijom mnogo većeg genoma te je malo vjerojatno da ćemo za tako male brojeve dobiti jedinstvenu Eulerovu stazu.

U programu se nalazi funkcija (compareStrings) koja uspoređuje originalni genom sa dobivenim na način da za svaki nukleotid u kojem se razlikuju funkcija u ukupnu sumu dodaje 1 te na izlazu vraća tu sumu. Na spomenutom primjeru pogrešne rekonstrukcije genoma TAATGCCATGGGATGTT funkcija bi nam vratila razliku stringova 4, jer se 'GG' nalazi na mjestu 'CC', i obratno. U slučaju da smo pravilno rekonstruirali genom, funkcija vraća 0. Obzirom da se Eulerova staza u de Bruijnovom grafu generira slučajno, mi ćemo za svaki ulazni k pokušati rekonstruirati genom 100 puta te izračunati prosječnu pogrešku u rekonstrukciji. Također ćemo za svaki k navesti koliko puta je genom savršeno rekonstruiran, koliko se različitih pogrešnih sekvenci pojavilo i kolika je najveća razlika između originalnog genoma i genoma dobivenih pogrešnim rekonstrukcijama. Na primjer, za $k = 3$ prosječna pogreška u rekonstrukciji iznosi 108660.42, maksimalna 109132, a genom nije niti jednom uspješno rekonstruiran. Isto ćemo učiniti za k =10, 20, 35, 50, 100, 150, 151 i 200. Rezultate ćemo prikazati u tablici (tablica 1).

k	prosječna razlika	maksimalna razlika	uspješne rekonstrukcije
3	108660.42	109132	0
10	108615.07	109116	0
20	108616.66	108972	0
35	99810.1	108635	3
50	84570.72	108424	22
100	76119.12	105721	28
150	44102.92	93836	53
151	0	0	100
200	0	0	100

Tablica 1: Rezultati u 100 pokušaja rekonstrukcije genoma *Carsonelle ruddii* pomoću k-torki.

Primjetimo da ni sa rekonstrukcijama pomoću 10 i 20-torki nismo dobili originalni genom. Eulerova staza u de Bruijnovim grafovima konstruiranim uz pomoć tih k-torki postoji, ali je u 100 pokušaja rekonstrukcije nismo uspjeli pronaći. Prosječna razlika kod tih pokušaja se također bitno ne mijenja. Prvi pomak se pojavio za $k = 35$, gdje smo uspjeli pronaći barem 3 ispravne Eulerove staze. Te staze ne moraju biti identične, jer su neki vrhovi u de Bruijnovom grafu međusobno povezani sa više bridova. Međutim, one nam generiraju istu sekvencu. Također se može primjetiti i lagani pad prosječne razlike. U narednim k-torkama uspjevamo dobiti sve više ispravnih sekvenci, prosječna razlika i dalje pada, a kod sekvence uz pomoć 151-torki očito postoji jedinstveno rješenje koje predstavlja originalni genom. Daljnjim testiranjem možemo otkriti da je to najmanji k za koji ćemo dobiti jedinstveno rješenje, a dobit ćemo ga i za svaki $k > 151$. Ono što je još zanimljivo u ovim rezultatima je to da se maksimalna razlika između originalnog genoma i dobivenih genoma bitno ne mijenja za sve k za koje postoji *pogrešna* Eulerova staza. To je zato što se dobiveni genomi (za velike k) razlikuju od originalnog po tome što su neki veći komadi genoma zamijenili mjesta. To je jedan od standardnih problema koji se pojavljuju u sekvencioniranju genoma mnogo složenijih organizama.

3.2. Rekonstrukcija uz pomoć (k,d)-torki

Drugi Python modul koji ćemo koristi se od prvoga razlikuje u samo nekoliko elemenata. Na ulazu prima $k, d \in \mathbb{N}$ koji označavju parove k-torki udaljene za d nepoznatih nukleotida. Ponovno lomimo genom, prikupljamo (k,d)-torke, sastavljamo de Bruijnov graf i tražimo Eulerovu stazu u grafu. Primjenimo li program na genom TAATGCCATGGGATGTT, uz pomoć (3,1)-torki ćemo ga uspješno rekonstruirati jer će postojati samo jedna Eulerova staza. Prirodno je onda pretpostaviti da će rekonstrukcija uz pomoć (k,d)-torki biti uspješnija od one uz pomoć k-torki za isti k, ali još ništa ne znamo o zahtjevu za parametar d. U prethodnom poglavlju smo već napomenuli da neće svaka Eulerova staza u grafu generiranom iz (k,d)-torki dati ispravan genom. Kao što smo već rekli, to možemo provjeriti tako da iz dobivene Eulerove staze složimo 2 stringa i provjerimo njihovo preklapanje. Program koji koristimo će bez obzira na preklapanje složiti genom na način da dio u kojem se stringovi trebaju preklopiti uzme iz prvog stringa. To ćemo učini tako jer nas zapravo opet zanima razlika između originalnog i dobivenog genoma, ali ćemo i navesti koliko se neispravnih genoma generiralo.

Za početak ćemo za parametar d uzeti 1, dok ćemo za k uzimati 3, 10, 25, 50, 74 i 75. Opet pratimo prosječnu pogrešku i broj uspješnih rekonstrukcija, ali ćemo umjesto maksimalne pogreške brojati pravilno generirane genome. Rezultat ćemo prikazati u novoj tablici (tablica 2).

k	**d**	**prosječna razlika**	**pravilne rekonstrukcije**	**uspješne rekonstrukcije**
3	1	108654.41	0	0
10		108619.91	0	0
25		85429.6	21	21
50		81345.0	25	25
74		48726.72	52	52
75		0	100	100

Tablica 2: Rezultati pokušaja rekonstrukcije pomoću (k,1)-torki.

Rezultati koje smo dobili su donekle očekivani. Naime, iako u svakom paru k-torki ne poznamo 1 nukleotid koji se između njih nalazi, možemo ga u procesu rekonstrukcije saznati iz drugih parova i tako iz (k,1)-torki dobiti rekonstrukciju sličnu onoj sa $(2 \cdot k + 1)$-torkama u prethodnim testiranjima. Na primjer, rezultati dobiveni sa (25,1)-torkama su slični onima sa 50-torkama, dok sa (75,1)-torkama već dobivamo jedinstveno rješenje. Pretpostavljamo da se to dogodilo jer smo iz njih stvorili $2 \cdot 75 + 1 = 151$-torke. Treba još primjetiti da nismo naišli na pravilne Eulerove staze koje daju krive rekonstrukcije. Ovaj postupak ćemo ponoviti za različite vrijednosti parametra d, a rezultate ćemo prikazati u tablici tako da za kombinaciju parametara k i d prikažemo prosječnu razliku dobivenih genoma od originalnog (gore) i broj uspješnih konstrukcija (dolje) uz koji se u zagradi nalazi i broj pravilnih rekonstrukcija (tablica 3).

d \ **k**	**3**	**10**	**25**	**50**	**75**
10	108674.71	108610.04	87723.0	54400.5	0
	0(0)	0(0)	19(19)	50(50)	100(100)
51	108691.06	108590.95	84611.2	0	0
	0(0)	0(0)	28(28)	100(100)	100(100)
101	108691.81	108612.94	0	0	0
	0(0)	0(0)	100(100)	100(100)	100(100)

Tablica 3: Rezultati pokušaja rekonstrukcije pomoću (k,d)-torki.

Ono što odmah možemo zaključiti iz tablice je da u ovom genomu nema pravilnih Eulerovih staza koje konstruiraju pogrešan genom. To znači da bi pronalaskom svih mogućih staza trebali u svakoj od njih provjeriti preklapanja 2 stringa spomenuta ranije i, u slučaju da odgovaraju, dobivamo uspješno rekonstruiran genom. Pronalaženje svih Eulerovih staza je vremenski složen proces čak i za manje genome kao što je naš pa se mi nećemo time baviti u ovom radu. Nadalje, primjećujemo da povećavanjem parametra k ponovno sve uspješnije rekonstruiramo genom, međutim, za d to nije uvijek slučaj. Generalno, povećanjem oba parametra povećava se i broj $2 \cdot k + d$, ali za manje vrijednosti k taj broj ne dolazi do izražaja. Na primjer, rekonstrukcija uz pomoć 100-torki je bolja od rekonstrukcije pomoću (10,101)-torki iako je $2 \cdot 10 + 101 = 121$.

U svakom slučaju, iz dobivenih rezultata možemo sigurno zaključiti da će rekonstrukcija pomoću (k,d)-torki biti uspješnija od one sa k-torkama. Naravno, mi se ovdje bavimo idealnim slučajem gdje imamo savršenu pokrivenost relativno malenog genoma Carsonelle ruddii, kao fiksnu vrijednost parametra d. Biolozi moraju vršiti mnoštvo testiranja da bi došli do takvih podataka pri čemu se često javljaju pogreške u očitanjima. To je razlog zašto genome većine organizama još ni danas nismo sekvencionirali.

Programe koje smo koristili u rekonstrukciji kao i cijeli genom Carsonelle ruddii možete pronaći na `https://github.com/jpetrovic91/GenomeSequencing/tree/master`

Literatura

[1] P. Compeau, P. Pevzner, *Bioinformatics Algorithms: An Active Learning Approach*, Active Learning Publishers, LLC, 2015.

[2] M. Domazet-Lošo, M. Šikić, *Bioinformatika*, Zagreb, 2013.
https://www.fer.unizg.hr/_download/repository/bioinformatika_skripta_v1.2.pdf

[3] R. Diestel, *Graph theory*, Electronic Edition, 2000.
http://www.esi2.us.es/~mbilbao/pdffiles/DiestelGT.pdf

[4] https://www.ncbi.nlm.nih.gov/nuccore/CP024798.1?report=fasta