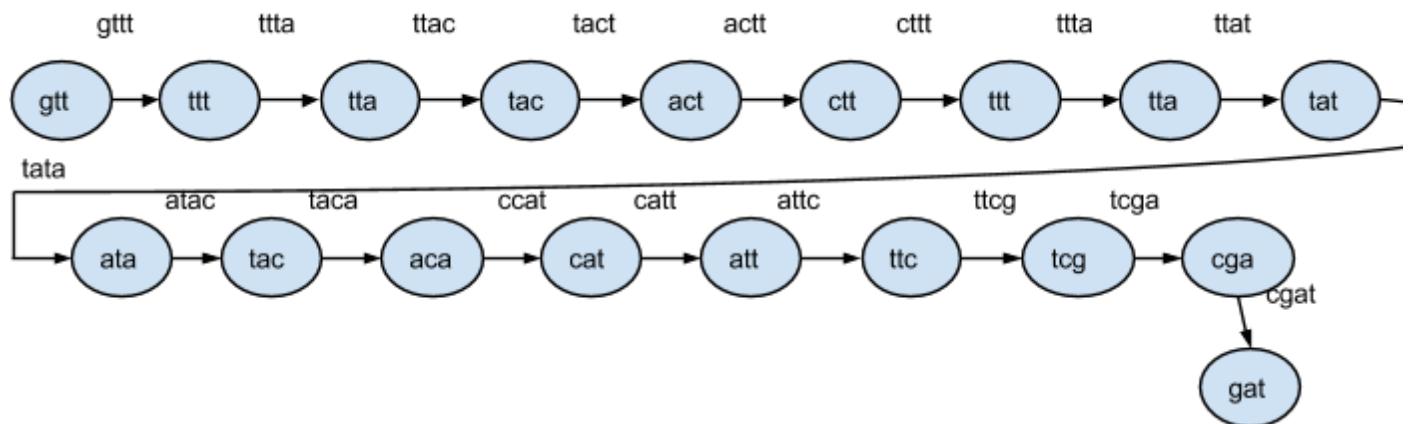


5.1.1

1, začetek GTTT

5.1.2

nesme imeti ciklov



5.1.1 Koliko različnih zaporedij lahko sestavite iz naslednje množice odčitkov ($k=4$)? Pri tem upoštevajte, da vrstni red izpisa odčitkov ni pomemben. (Namig: ali lahko ugotovite, kateri odčitek predstavlja začetek zaporedja)?

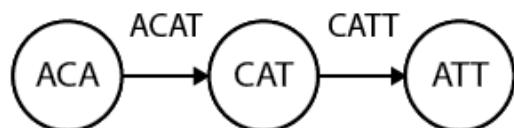
[ACAT, ACTT, ATAC, ATTC, CATT, CGAT, CTTT, GTTT, TACA, TACT, TATA, TCGA, TTAC, TTAT, TTCG, TTTA, TTTA]

Lahko sestavimo 2 zaporedja, če gledamo, da moramo uporabiti vsak odčitek enkrat. Pri sestavljanju gledamo, da se zadnjih $k-1$ znakov odčitka, ujema z drugim odčitkom prvih $k-1$ elementov.

Dobimo zaporedja GTTTACTTATACATTGAT in GTTTATACTTACATTGAT.

Pripona GTT se ne pojavi v nobenem drugem odčitku, zato lahko sklepamo, da je GTTT začetek, enako velja za konec, predpona GAT, se pravtako ne pojavi v nobenem drugem odčitku, zato sklepamo da je CGAT konec.

5.1.2 Pri reševanju problema si lahko pomagamo z de Bruijnovimi grafi. De Bruijnov graf predstavlja odčitek kot usmerjene povezave, kot vozlišča pa njihove pripone in predpone (oboje dolžine $k-1$). Primer: odčitka dolžine $k=4$ za zaporedje ACATT lahko predstavimo z grafom:



Kaj mora veljati za takšen graf, da lahko iz njega enolično sestavimo zaporedje? (Namig: pri sestavljanju moramo vsak odčitek upoštevati natančno enkrat.)

Veljati mora, da lahko vsako vozlišče običemo samo enkrat oz. , da je graf predstavljen kot vrsta. V tem primeru imamo enega prednika in enega naslednjika, graf tudi ne sme imeti ciklov.

**Graf je usmerjen in lahko ima cikle.
Obstajati mora Eulerjeva pot.**

5.1.3. V psevodokodi opišite algoritem za reševanje problema 5.1.2 v poljubnem de Bruijnovem grafom.

Najdemo začetno vozlišče, se sprehodimo do končnega vozlišča. V prvem vozlišču dodamo celoten odčitek, zatem dodajamo samo zadnjo črko odčitka, na koncu dobimo celotno zaporedje.

Naloga 5.1

5.1.1)

Imamo 2 različni zaporedji. Začetek zaporedja predstavlja odčitek GTTT, konec pa CGAT, saj od tam naprej nimamo nobenega odčitka podanega, ki bi se začel z GAT... Zaporedna odčitka se ujemata v k-1 znakih.

1.Zaporedje

```

GTTT
TTTA
TTAC
TACT
ACTT
CTTT
TTTA
TTAT
TATA
ATAC
TACA
ACAT
CATT
ATTC
TTCG
TCGA
CGAT

```

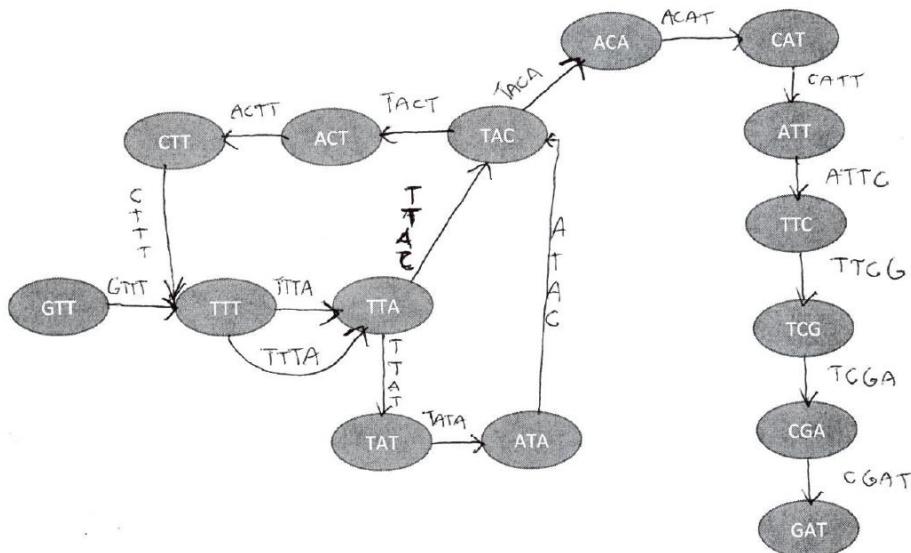
2.Zaporedje

```

GTTT
TTTA
TTAT
TATA
ATAC
TACT
ACTT
CTTT
TTTA
TTAC
TACA
ACAT
CATT
ATTC
TTCG
TCGA
CGAT

```

5.1.2)



Vsek De Bruijn graf ima Eulerjev in Hamiltonov cikel, kar pomeni, da moramo po vsaki povezavi in vsakem vozlišču samo enkrat.

Zadostuje Eulerjeva *pot* od začetnega do ciljnega vozlišča.

5.1.3.)

Algoritme podajajte v psevdokodi!

```
def de_brujin(k, n):      Manjka branje množice odčitkov.
    a = [0] * k * n
    sequence = []
    def db(t, p):
        if t > n:
            if n % p == 0:
                for j in range(1, p + 1):
                    sequence.append(a[j])
            else:
                a[t] = a[t - p]
                db(t + 1, p)
                for j in range(a[t - p] + 1, k):
                    a[t] = j
                    db(t + 1, t)
        db(1, 1)
    return sequence
```

5.1.1)

GT_{TT}A_{CTT}TATACATTCTTCGAT

GT_{TT}

TTA

TTAC

TACT

ACTT

CTTT

TTTA

TTAT

TATA

ATAC

TACA

ACAT

CATT

ATTC

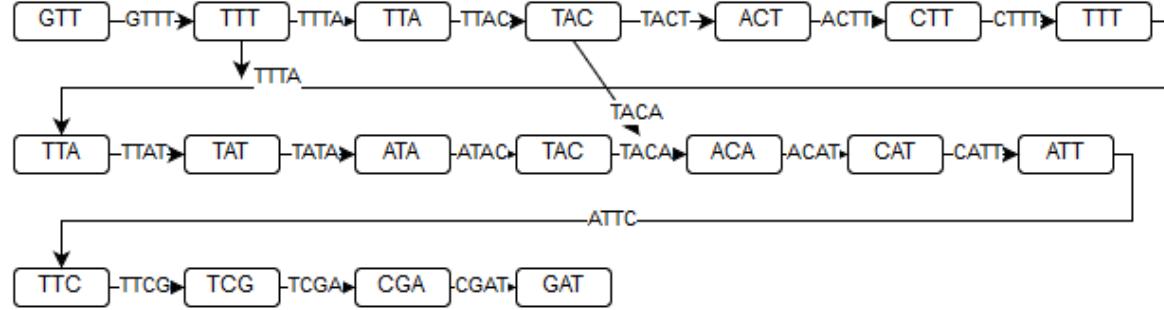
TTCG

TCGA

CGAT

Eno zaporedje, štirič če lahko zgradimo takega ki ne porabi vseh kosov zaporedja.

5.1.2)



naloga 5.1

Teorija grafov pri sestavljanju nizov. Poenostavljeno povedano so DNK molekule so zelooo dolga zaporedja črk iz abecede {A, C, G in T}. Ljudje še nismo izumili stroja, ki bi lahko prebral zaporedje poljubno dolgega zaporedja, ampak znamo prebrati (na najnovejših strojih) tja do par sto znakov dolgo zaporedje. Tako smo se lotili sestavljanja zaporedja nukleotidov posredno. Vzamemo veliko kopij preiskovane DNK in vsako razrežemo na mestih, katerih ne moremo določiti v naprej določiti. Velja zgolj to, da so razrezani koščki dovolj kratki, da lahko v njih določimo zaporedje nukleotidov. Postopku razrezovanja rečemo razrez s šibrovko (angl. *shotgun sequencing*). Delčkom, ki smo jih dobili in jim lahko nato določimo zaporedje nukleotidov, rečemo odčitki (angl. *reads*). V tej nalogi boste sestavljeni odčitki v izvorno DNK.

Dolžina odčitka naj bo k . Odčitki se načeloma lahko začnejo na kateremkoli znaku zaporedja. Primer: za zaporedje ATAGCTAGTA in $k=5$ dobimo naslednje možne odčitke:

ATAGCTAGTA

ATAGC
TAGCT
AGCTA
GCTAG
CTAGT
TAGTA

Zaporedna odčitka se ujemata v $k-1$ zaporednih znakih.

Vprašanja:

5.1.1 Koliko različnih zaporedij lahko sestavite iz naslednje množice odčitkov ($k=4$)? Pri tem upoštevajte, da vrstni red izpisa odčitkov ni pomemben. (Namig: ali lahko ugotovite, kateri odčitek predstavlja začetek zaporedja)?

[ACAT, ACTT, ATAC, ATTC, CATT, CGAT, CTAA, GTTT, TACA, TACT, TATA, TCGA, TTAC, TTAT, TTCG, TTTA, TTTA]

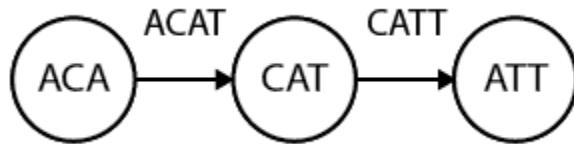
Rešitev:

Začetek zaporedja predstavlja odčitek GTTT. Če narišemo Bruijnov graf, ta odčitek nima vhodne povezave. CGAT nima izhodne povetave, zato predstavlja konec zaporedja. Sestavimo lahko 2 različni zaporedji:

- GTTTACTTATACATTGAT
- GTTTATACTTACATTGAT

Pravilno!

5.1.2 Pri reševanju problema si lahko pomagamo z de Bruijnovimi grafi. De Bruijnov graf predstavlja odčitek kot usmerjene povezave, kot vozlišča pa njihove pripone in predpone (oboje dolžine $k-1$). Primer: odčitka dolžine $k=4$ za zaporedje ACATT lahko predstavimo z grafom:



Kaj mora veljati za takšen graf, da lahko iz njega enolično sestavimo zaporedje? (Namig: pri sestavljanju moramo vsak odčitek upoštevati natančno enkrat.)

Rešitev:

Da iz grafa sestavimo enolično sestavimo zaporedje mora veljati, da začetek in konec grafa ne smeta biti povezana. Vsako vmesno vozlišče mora imeti natanko eno vhodno in eno izhodno povezavo. Graf mora biti povezan. Od začetka do konca grafa, se pot lako razveje, vendar je vsak odčitek možen le enkrat v zaporedju.

5.1.3. V psevodokodi opišite algoritem za reševanje problema 5.1.2 v poljubnem de Bruijnovem grafom.

Rešitev:

zacetek = Z;

Algoritme podajajte v psevdokodi.

fun findNext(Z):

 end = Z.slip[1, Z.length] **Definirajte funkcije, ki jih uporabljate!**

 for read OD and read DO:

 if(OD.split[0,2] == end){

 povezava iz I v OD;

 findNext(OD);

 }

 return

Teorija grafov pri sestavljanju nizov. Poenostavljeno povedano so DNK molekule so zelo dolga zaporedja črk iz abecede {A, C, G in T}. Ljudje še nismo izumili stroja, ki bi lahko prebral zaporedje poljubno dolgega zaporedja, ampak znamo prebrati (na najnovejših strojih) tja do par sto znakov dolgo zaporedje. Tako smo se lotili sestavljanja zaporedja nukleotidov posredno. Vzamemo veliko kopij preiskovane DNK in vsako razrežemo na mestih, katerih ne moremo določiti v naprej določiti. Velja zgolj to, da so razrezani koščki dovolj kratki, da lahko v njih določimo zaporedje nukleotidov. Postopku razrezovanja rečemo razrez s šibrovko (angl. *shotgun sequencing*). Delčkom, ki smo jih dobili in jim lahko nato določimo zaporedje nukleotidov, rečemo odčitki (angl. *reads*). V tej nalogi boste sestavljali odčitke v izvorno DNK.

Dolžina odčitka naj bo k . Odčitki se načeloma lahko začnejo na kateremkoli znaku zaporedja. Primer: za zaporedje ATAGCTAGTA in $k=5$ dobimo naslednje možne odčitke:

ATAGCTAGTA

ATAGC

TAGCT

AGCTA

GCTAG

CTAGT

TAGTA

Zaporedna odčitka se ujemata v $k-1$ zaporednih znakih.

Vprašanja:

5.1.1 Koliko različnih zaporedij lahko sestavite iz naslednje množice odčitkov ($k=4$)? Pri tem upoštevajte, da vrstni red izpisa odčitkov ni pomemben. (Namig: ali lahko ugotovite, kateri odčitek predstavlja začetek zaporedja)?

[ACAT, ACTT, ATAC, ATTC, CATT, CGAT, CTTT, GTTT, TACA, TACT, TATA, TCGA, TTAC, TTAT, TTCA, TTTA]

Vzamemo poljuben odčitek, jaz bom začel kar pri prvem in nato pogledamo če se kakšen od preostalih ujema v prvih ali zadnjih treh črkah.

ACAT

CATT

ATTC

TTCG

TCGA

CGAT

Vidimo da ni več nobenega ki bi lahko bil za CGAT, zato nadaljujemo navzgor.

TTAC

TACA

ACAT

CATT

ATTC

TTCG

TCGA

CGAT

Tu pridemo prvič do tega da sta za naslednji korak navzgor možna dva odčitka, tako vidimo da bosta vsaj dva zaporedja

GTTT

TTTA

TTAT

TATA

ATAC

TACT

ACTT

CTTT

TTTA

TTAC

TACA

ACAT

CATT

ATTC
TTCG
TCGA
CGAT

Drugi:

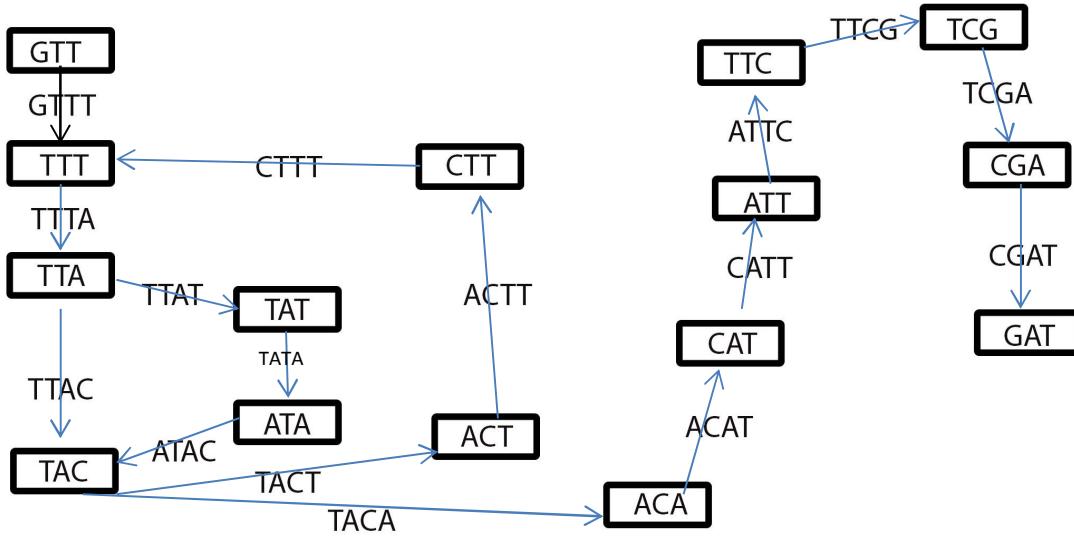
GTTT
TTTA
TTAC
TACT
ACTT
CTTT
TTTA
TTAT
TATA
ATAC
TACA
ACAT
CATT
ATTC
TTCG
TCGA
CGAT

Prvo zaporedje: GTTTACTTTATACATTGAT

Drugo zaporedje: GTTTATACTTTACATTGAT

Začetek zaporedja je GTTT, konec pa CGAT ker ni nobenega odčitka ki bi lahko bil pred oziroma za tem odčitkom.

5.1.2 Pri reševanju problema si lahko pomagamo z de Bruijnovimi grafi. De Bruijnov graf predstavlja odčitek kot usmerjene povezave, kot vozlišča pa njihove pripone in predpone (oboje dolžine $k-1$). Primer: odčitka dolžine $k=4$ za zaporedje ACATT lahko predstavimo z grafom:



Kaj mora veljati za takšen graf, da lahko iz njega enolično sestavimo zaporedje? (Namig: pri sestavljanju moramo vsak odčitek upoštevati natančno enkrat.)

Tak graf mora biti tako Eulerjev kot Hamiltonov.

5.1.3. V psevodokodi opišite algoritem za reševanje problema 5.1.2 v poljubnem de Bruijnovem grafom.

Link: http://en.wikipedia.org/wiki/De_Bruijn_sequence#Algorithm

```
def de_bruijn(k, n):
    """
    De Bruijn sequence for alphabet k
    and subsequences of length n.
    """
    try:
        # Let's see if k can be cast to an integer;
        # if so, make our alphabet a list
        _ = int(k)
        alphabet = list(map(str, range(k)))
    except (ValueError, TypeError):
        alphabet = k
        k = len(k)

    a = [0] * k * n
    sequence = []

    def db(t, p):
        if t > n:
            if n % p == 0:
                sequence.extend(a[1:p + 1])
            else:
                a[t] = a[t - p]
                db(t + 1, p)
                for j in range(a[t - p] + 1, k):
                    a[t] = j
                    db(t + 1, t)
    db(1, 1)
    return "".join(alphabet[i] for i in sequence)

print(de_bruijn(2, 3))
print(de_bruijn("abcd", 2))
```

Ta algoritem ne rešuje problema, ki nas zanima!

5.1

5.1.1

Sestavimo lahko zaporedje GTTTACTTTATACATTGAT:

GT
TTA
TAC
TAC
ACT
CTT
TTA
TTAT
TATA
ATAC
TACA
ACAT
CATT
ATTC
TTCG
TCGA
CGAT

CGAT in GTTT se vzorec zaporedja nikjer ne ponovita v naslednjih odčitkih.

5.1.2

Za takšen graf mora veljati, da je vrsta. Vozlišča imajo lahko samo enega naslednika in enega prednika. Ne sme biti ciklov, kar pomeni da morata biti prvi in zadnji element,

5.1.3

Zaporedje lahko poiščemo tako da najprej poiščemo začetno vozlišče. Dodamo celoten odčitek in nato nadaljujemo, dokler ne pridemo do zadnjega vozlišča. Vsakemu naslednjemu dodamo zadnjo črko odčitka.

Teorija grafov pri sestavljanju nizov. Poenostavljeno povedano so DNK molekule so zelo dolga zaporedja črk iz abecede {A, C, G in T}. Ljudje še nismo izumili stroja, ki bi lahko prebral zaporedje poljubno dolgega zaporedja, ampak znamo prebrati (na najnovejših strojih) tja do par sto znakov dolgo zaporedje. Tako smo se lotili sestavljanja zaporedja nukleotidov posredno. Vzamemo veliko kopij preiskovane DNK in vsako razrežemo na mestih, katerih ne moremo določiti v naprej določiti. Velja zgorj to, da so razrezani koščki dovolj kratki, da lahko v njih določimo zaporedje nukleotidov. Postopku razrezovanja rečemo razrez s šibrovko (angl. *shotgun sequencing*). Delčkom, ki smo jih dobili in jim lahko nato določimo zaporedje nukleotidov, rečemo odčitki (angl. *reads*). V tej nalogi boste sestavljali odčitke v izvorno DNK.

Dolžina odčitka naj bo k . Odčitki se načeloma lahko začnejo na kateremkoli znaku zaporedja. Primer: za zaporedje ATAGCTAGTA in $k=5$ dobimo naslednje možne odčitke:

ATAGCTAGTA

ATAGC

TAGCT

AGCTA

GCTAG

CTAGT

TAGTA

Zaporedna odčitka se ujemata v $k-1$ zaporednih znakih.

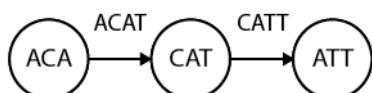
Vprašanja:

5.1.1 Koliko različnih zaporedij lahko sestavite iz naslednje množice odčitkov ($k=4$)? Pri tem upoštevajte, da vrstni red izpisa odčitkov ni pomemben. (Namig: ali lahko ugotovite, kateri odčitek predstavlja začetek zaporedja?)

[ACAT, ACTT, ATAC, ATTC, CATT, CGAT, CTCT, GTTT, TACA, TACT, TATA, TCGA, TTAC, TTAT, TTCA, TTTA]

Iz naslednje množice odčitkov ($k=4$) lahko dobimo 2 zaporedja. Zaporedje ACATTCGAT in GTTTATACCTT. Ko nam uspe sestaviti zaporedje, lahko tudi določimo kateri odčitek predstavlja začetek zaporedja.

5.1.2 Pri reševanju problema si lahko pomagamo z de Bruijnovimi grafi. De Bruijnov graf predstavlja odčitek kot usmerjene povezave, kot vozlišča pa njihove pripone in predpone (oboje dolžine $k-1$). Primer: odčitka dolžine $k=4$ za zaporedje ACATT lahko predstavimo z grafom:



Kaj mora veljati za takšen graf, da lahko iz njega enolično sestavimo zaporedje? (Namig: pri sestavljanju moramo vsak odčitek upoštevati natančno enkrat.)

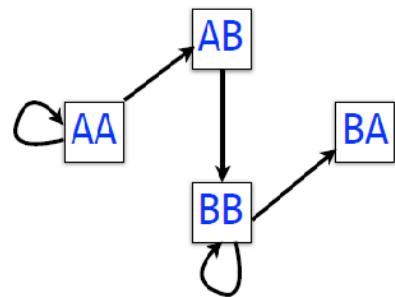
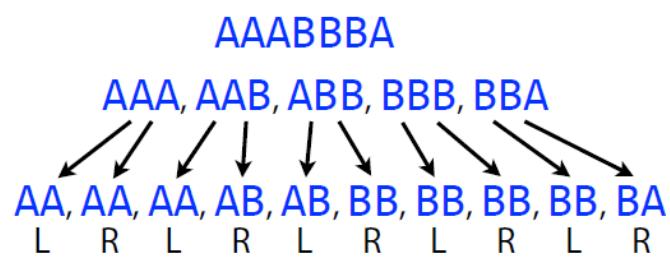
Da lahko iz njega enolično sestavimo zaporedje, mora veljati, da lahko vsako vozlišče obiščemo največ enkrat. De Bruijinov graf vsebuje Hamiltonov in Eulerjev cikel. Tem ciklom se reče De Bruijinovo zaporedje. Prav tako za vsak De Bruijinov graf z 2^n vozlišč velja, da vsebuje natanko $2^{2^{n-1}-1}$ različnih Hamiltonovih ciklov.

5.1.3. V psevodokodi opišite algoritem za reševanje problema 5.1.2 v poljubnem de Bruijninem grafom.

Class DeBruijinGraph:

```
//uporabnik poda besedilo in odčitke dolžine k. Vozlišča (node) v DeBruijinovem grafu so k-1 odčitki,  
//povezave pa povežejo leve in desne odčitke  
  
//metoda, ki razreže string v odčitke  
def chop(st, k):  
    for i in xrange(0,len(st)-(k-1)):  
        yield st[i:i+k]  
  
class Node:  
//vozlišče, ki predstavlja k-1 odčitek v grafu  
    def __init__(self, km1mer):  
        self.km1mer = km1mer  
  
    def __hash__(self):  
        return hash(self.km1mer)  
  
//konstruktor zgradi DeBruijinov graf iz besedila in odčitkov  
  
def __init__(self, strlter, k):  
    self.G = {}  
    self.nodes = {}  
    self.k = k;  
    for st in strlter:  
        for kmer in self.chop(st, k):      //za vsak odčitek, poišči levi in desni k-1 odčitek  
            km1L, km1R = kmer[:-1], kmer[1:]  
            nodeL, nodeR = None, None  
            //ustvari ustrezna vozlišča in povezave za DeBruijinov graf  
            if km1L in self.nodes:  
                nodeL = self.nodes[km1L]  
            else:  
                nodeL = self.nodes[km1L] = self.Node(km1L)  
            if km1R in self.nodes:  
                nodeR = self.nodes[km1R]  
            else:  
                nodeR = self.nodes[km1R] = self.Node(km1R)  
            self.G.setdefault(nodeL, []).append(nodeR)
```

Potrebno je poiskati Eulerjev sprehod.



1

Iz odčitkov [ACAT, ACTT, ATAC, ATTC, CATT, CGAT, CTAA, GTTT, TACA, TACT, TATA, TCGA, TTAC, TTAT, TTCA, TTTA] je možno zgraditi $(n - k)!$ podnizov niz, ki vključuje vse, je GTTTTACACTTACATTGAT in GTTTACTTATACATTGAT. Začne se z GTTT.

2

Da lahko enolično sestavimo zaporedje mora biti graf povezan usmerjen in razen začetnega ter končnega vozlišča, morajo imeti samo eno vstopno in eno izstopno povezavo.

3

Rekurzivno kličemo metodo kateri podamo vse podnize razen trenutnega, poizkušamo dokler obstaja podniz ki ga nismo uporabili.

```
while pripona=input
    if pripona.startsWith(vozlisce.predpona)
        vozlisce=vozlisce.next
    else
        vozlisce=zacetek
```

Ta metoda je požrešna. Kaj nam zagotavlja, da bomo zaporedje res našli?

Naloga 5.1

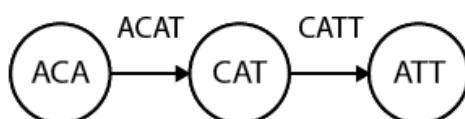
5.1.1 Koliko različnih zaporedij lahko sestavite iz naslednje množice odčitkov (k=4)? Pri tem upoštevajte, da vrstni red izpisa odčitkov ni pomemben.

[ACAT, ACTT, ATAC, ATTC, CATT, CGAT, CTAA, GTTT, TACA, TACT, TATA, TCGA, TTAC, TTAT, TTG, TTTA, TTTA]

Začetek zaporedja predstavlja odčitek »GTTT« saj se noben odčitek ne vsebuje podniza »GTT«. Zadnji odčitek pa je »CGAT« saj se noben odčitek ne začne z »GAT«. Iz zgornjih odčitkov obstajata dve zaporedji:

Zaporedje 1: GTTTATACTTACATTGAT	Zaporedje 2: GTTTACTTATACATTGAT
GTTT TTTA TTAT TATA ATAC TACT ACTT CTTT TTTA TTAC TACA ACAT CATT ATTC TTG TCGA CGAT	GTTT TTTA TTAC TACT ACTT CTTT TTTA TTAT TATA ATAC TACA ACAT CATT ATTC TTG TCGA CGAT

5.1.2 Pri reševanju problema si lahko pomagamo z de Bruijnovimi grafi. De Bruijnov graf predstavlja odčitek kot usmerjene povezave, kot vozlišča pa njihove pripone in predpone (oboje dolžine k-1). Primer: odčitka dolžine k=4 za zaporedje ACATT lahko predstavimo z grafom:



Kaj mora veljati za takšen graf, da lahko iz njega enolično sestavimo zaporedje? (Namig: pri sestavljanju moramo vsak odčitek upoštevati natančno enkrat.)

5.1.3. V psevodokodi opišite algoritom za reševanje problema 5.1.2 v poljubnem de Bruijnovem grafom.

```
def de_bruijn(k, n):
    try:
        _ = int(k)
        alphabet = list(map(str, range(k)))

    except (ValueError, TypeError):
        alphabet = k
        k = len(k)

    a = [0] * k * n
    sequence = []

    def db(t, p):
        if t > n:
            if n % p == 0:
                sequence.extend(a[1:p + 1])
            else:
                a[t] = a[t - p]
                db(t + 1, p)
                for j in range(a[t - p] + 1, k):
                    a[t] = j
                    db(t + 1, t)
        db(1, 1)
    return "".join(alphabet[i] for i in sequence)

print(de_bruijn(2, 3))
print(de_bruijn("abcd", 2))
```

Vir: http://en.wikipedia.org/wiki/De_Bruijn_sequence#Algorithm

Matej Čuček Gerbec 63970034

Naloga 5.1

5.1.1.)

ACAT, ACTT, ATAC, ATTC, CATT, CGAT, CTTT, GTTT, TACA, TACT, TATA, TCGA, TTAC, TTAT, TTCG, TTTA, TTTA

GT
TT
TTA
TTAC
TACT
ACTT
CTTT
TTTA
TTAT
TATA
ATAC
TACA
ACAT
CATT
ATTC
TTCG
TCGA
CGAT

Iz vseh odčitkov lahko sestavimo le niz GTTTACTTATACATTGAT. GTTT vemo da je prvi ker se v nobenem od drugih odčitkov ne pojavi pripona GTTT. In CGAT je zadnji ker se v nobenem od odčitkov ne pojavi predpona GAT.

5.1.2.)

Če hočemo iz de Bruijnovega grafa dobiti enolično zaporedje, mora za graf veljati, da je vrsta. Vsako vozlišče ima lahko le enega prednika in le enega naslednika, in v grafu ne sme biti ciklov (morata obstajat prvi in zadnji element).

5.1.3.)

Dobiti zaporedje iz takega grafa je enostavno. Poишčemo štartno vozlišče ($O(V)$), to pomeni vozlišče, ki nima prednikov in nato nadaljujemo dokler ne pridemo do vozlišča brez naslednikov ($O(V)$). V prvem vozlišču dodamo celoten očitek, v vsakem naslednjem dodamo samo zadnjo črko odčitka.

Naloga 5.1

Teorija grafov pri sestavljanju nizov. Poenostavljeno povedano so DNK molekule so zelo dolga zaporedja črk iz abecede {A, C, G in T}. Ljudje še nismo izumili stroja, ki bi lahko prebral zaporedje poljubno dolgega zaporedja, ampak znamo prebrati (na najnovejših strojih) tja do par sto znakov dolgo zaporedje. Tako smo se lotili sestavljanja zaporedja nukleotidov posredno. Vzamemo veliko kopij preiskovane DNK in vsako razrežemo na mestih, katerih ne moremo določiti v naprej določiti. Velja zgolj to, da so razrezani koščki dovolj kratki, da lahko v njih določimo zaporedje nukleotidov. Postopku razrezovanja rečemo razrez s šibrovko (angl. shotgun sequencing). Delčkom, ki smo jih dobili in jim lahko nato določimo zaporedje nukleotidov, rečemo odčitki (angl. reads). V tej nalogi boste sestavljali odčitke v izvorno DNK.

- 1) Koliko različnih zaporedij lahko sestavite iz naslednje množice odčitkov ($k=4$)? Pri tem upoštevajte, da vrstni red izpisa odčitkov ni pomemben. (Namig: ali lahko ugotovite, kateri odčitek predstavlja začetek zaporedja)?

[ACAT, ACTT, ATAC, ATTC, CATT, CGAT, CTAA, GTTT, TACA, TACT, TATA, TCGA, TTAC, TTAT, TTCG, TTTA, TTTA]

Zaporedje 1: **GTTTATACTTACATTTCGAT**

The diagram shows the DNA sequence **GTTTATACTTACATTTCGAT** at the top. Below it, there are 16 horizontal bars, each representing a 4-base read. The reads are: G T T T, T T T A, T T A T, T A T A, A T A C, T A C T, A C T T, C T T T, T T T A, T T A C, T A C A, A C A T, C A T T, A T T C, T T C G, T C G A, and C G A T.

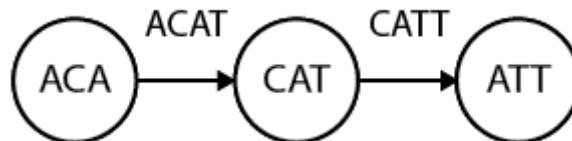
Zaporedje 2: GTTTACTTATACATTGAT

```

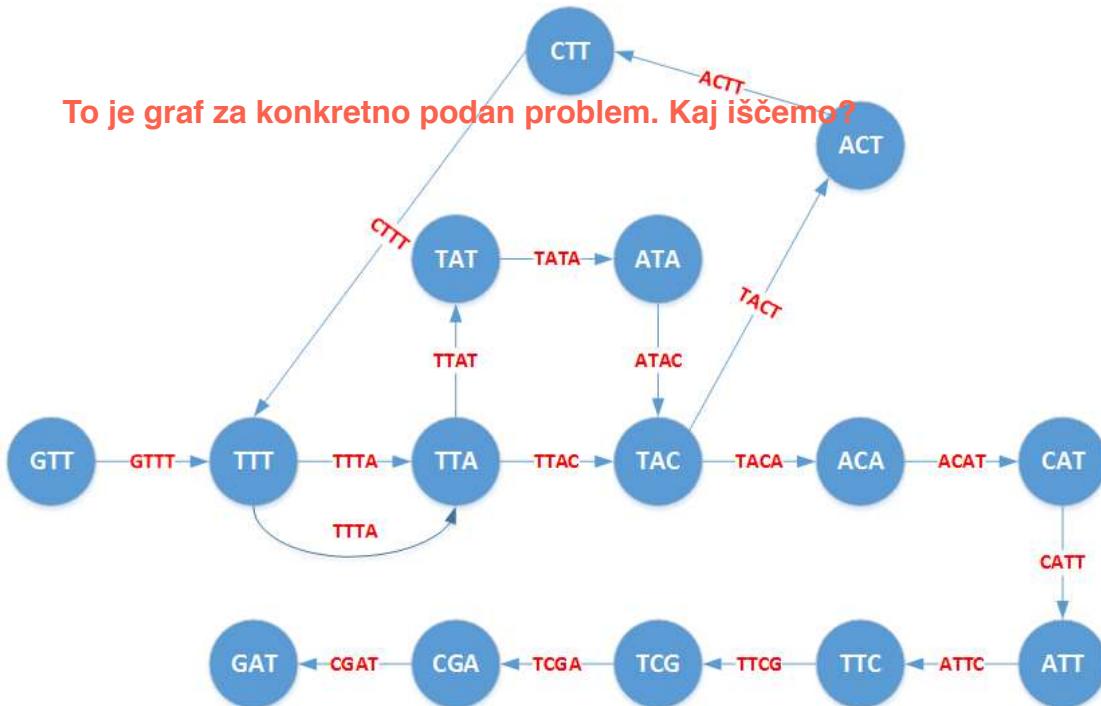
G T T T
T T T A
T T A C
T A C T
A C T T
C T T T
T T T A
T T A T
T A T A
A T A C
T A C A
A C A T
C A T T
A T T C
T T C G
T C G A
C G A T

```

- 2) Pri reševanju problema si lahko pomagamo z de Bruijnovimi grafi. De Bruijnov graf predstavlja odčitek kot usmerjene povezave, kot vozlišča pa njihove pripone in predpone (oboje dolžine $k-1$). Primer: odčitka dolžine $k=4$ za zaporedje ACATT lahko predstavimo z grafom:



Kaj mora veljati za takšen graf, da lahko iz njega enolično sestavimo zaporedje? (Namig: pri sestavljanju moramo vsak odčitek upoštevati natančno enkrat.)



3) V psevodokodi opišite algoritem za reševanje problema 5.1.2 v poljubnem de Bruijnovem grafom.

```
def de_bruijn(k, n):
    try:
        _ = int(k)
        alphabet = list(map(str, range(k)))
    except (ValueError, TypeError):
        alphabet = k
        k = len(k)

    a = [0] * k * n
    sequence = []

    def db(t, p):
        if t > n:
            if n % p == 0:
                sequence.extend(a[1:p + 1])
            else:
                a[t] = a[t - p]
                db(t + 1, p)
            for j in range(a[t - p] + 1, k):
                a[t] = j
                db(t + 1, t)
        db(1, 1)
        return ''.join(alphabet[i] for i in sequence)
```

VIR: http://en.wikipedia.org/wiki/De_Bruijn_sequence

Naloga 5.1

5.1.1

Odčitek, ki predstavlja začetek zaporedja je v našem primeru odčitek GTTT. To je bilo možno ugotoviti zato, ker njegovo predpono GTT ne vsebuje noben drug odčitek kot pripono. To pa pomeni, da je zato odčitek na prvem mestu. Pri iskanju rešitve sem si s tem tudi pomagal. Začel sem z prvim odčitkom in iskal njegove možne naslednike. Če jih je bilo več sem naredil "vejo" nove možne rešitve katero sem obdelal kasneje. Na koncu sem dobil dve možni zaporedji, kjer sem uporabil vse odčitke.

- GTTTATACTTACATTGAT
- GTTTACTTATACATTGAT

5.1.2

Ker v grafu posamezen odčitek predstavlja posamezna povezava mora za graf veljati, da v njem obstaja taka pot, ki se sprehodi po vsaski povezavi natanko enkrat. Torej tak graf je Eulerjev graf. Toliko kolikor je takih različnih poti, toliko je tudi vseh možnih vzorcev.

5.1.3

Algoritem bi začel v začetnem vozlišču sprehod po usmerjenih povezavah in sproti bi sestavljal vzorec. Kadar bi lahko iz določenega vozlišča šel po različnih povezavah bi pomenilo, da je več vzorcev, če bo algoritmu po tej poti uspelo obiskati vse povezave v grafu. Ko se algoritem neha sprehajati, je potrebno pogledati ali so bile obiskane vse povezave. Če niso bile potem vzorec te poti ne obstaja, če pa so bile potem je sestavljeni vzorec možna rešitev za odčitke, ki sestavljajo graf.

Domača naloga 5.1

1. Koliko različnih zaporedij lahko sestavite iz naslednje množice odčitkov ($k=4$)? Pri tem upoštevajte, da vrstni red izpisa odčitkov ni pomemben. (Namig: ali lahko ugotovite, kateri odčitek predstavlja začetek zaporedja)?

Sestavimo lahko 2 različna zaporedja.

Prvo je **GTTTATACTTTACATTCGAT**, ki je sestavljen iz zaporedij: GTTT, TTTA, TTAT, TATA, ATAC, TACT, ACTT, CTTT, TTTA, TTAC, TACA, ACAT, CATT, ATTC, TTCG, TCGA, CGAT.

Drugo je **GTTTACTTTATACATTCGAT**, ki je sestavljen iz zaporedij: GTTT, TTTA, TTAC, TACT, ACTT, CTTT, TTTA, TTAT, TATA, ATAC, TACA, ACAT, CATT, ATTC, TTCG, TCGA, CGAT

2. Pri reševanju problema si lahko pomagamo z de Bruijnovimi grafi. De Bruijnov graf predstavlja odčitek kot usmerjene povezave, kot vozlišča pa njihove pripone in predpone (oboje dolžine $k-1$). aj mora veljati za takšen graf, da lahko iz njega enolično sestavimo zaporedje? (Namig: pri sestavljanju moramo vsak odčitek upoštevati natančno enkrat.)

Tak graf mora biti Eulerjev graf (vsako pot/povezavo, običemo natanko enkrat), hkrati pa mora biti Hamiltonov (vsako vozlišče običemo natanko enkrat). Za graf mora tudi veljati, da ima končno točko.

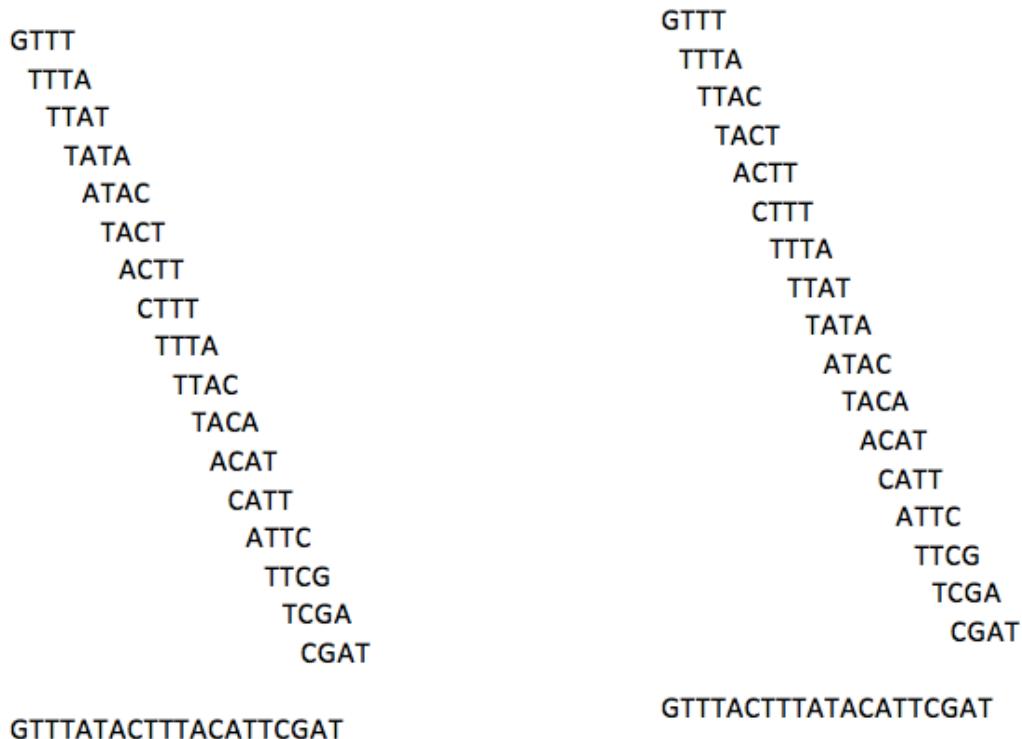
3. V psevodokodi opišite algoritem za reševanje problema 5.1.2 v poljubnem de Bruijnovem grafom.

```
String zaporedje = v[0].text // v zaporedje dodamo črke prvega vozlišča
while(v.e != null) //dokler ima vozlišče povezavo naprej
    v=v.e //naslednje vozlišče
    zaporedje+= v.substring(k/2) //dodamo zadnjo črko vozlišča, kjer k/2 zaokrožimo navzgor
```

Naloga 5.1

5.1.1. Koliko različnih zaporedij:

[ACAT, ACTT, ATAC, ATTC, CATT, CGAT, CTAA, GTTT, TACA, TACT, TATA, TCGA, TTAC, TTAT, TTCG, TTTA, TTTA]



Iz celotne množice odčitkov lahko sestavimo 2 različna zaporedja.

GTTT predstavlja začetek zaporedja, ker se noben drug odčitek na konča s črkami GTT. CGAT predstavlja konec zaporedja, ker se noben drug odčitek ne začne s črkami GAT.

5.1.2. Kaj mora veljati za takšen graf:

Tak graf mora biti Eulerjev in Hamiltonov.

5.1

Sestavil sem 2 različni zaporedji. To sta GTTTACTTATACATTGAT in GTTTATACTTACATTGAT. Ko sestavimo zaporedje, ugotovimo tudi kateri odčitek predstavlja začetek zaporedja.

5.1.2

Če hočemo imeti enolično zaporedje, lahko vsako vozlišče(odčitek) obiščemo samo enkrat. Torej mora biti antisimetričen ter usmerjen. Vsebovati mora Eulerjev in Hamiltonov cikel.

5.1.3

```
Begin procedure algoritem(String beseda, int k){
```

```
    List odcitki;
```

```
    Nodes nodes;
```

```
    odcitki.add(beseda.substring(0,k-1)
```

```
    For(int i = 1; i<beseda.length-k;i++){
```

```
        odcitki.add(beseda.substring(i,i+k-1)
```

```
}
```

```
    Nodes.add(odcitki.get(0));
```

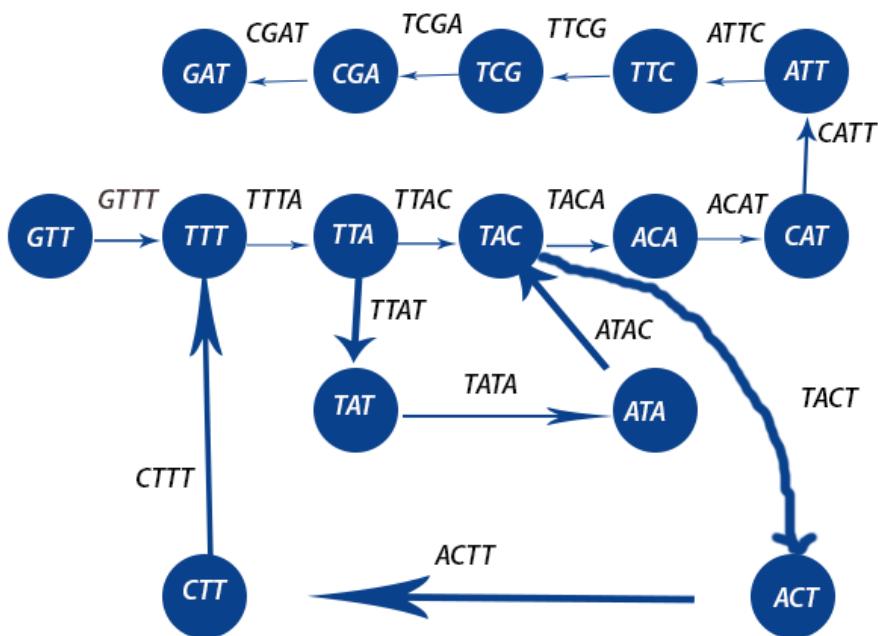
```
    For(int i = 1; i<odcitki.size(); i++){
```

```
        Nodes.addRight(odcitki.get(i));
```

```
}
```


drugo zaporedje je GTTTACTTTATACATTGAT.

5.1.2 Pri reševanju problema si lahko pomagamo z de Bruijnovimi grafi. De Bruijnov graf predstavlja odčitek kot usmerjene povezave, kot vozlišča pa njihove pripone in predpone (oboje dolžine $k-1$). Primer: odčitka dolžine $k=4$ za zaporedje ACATT lahko predstavimo z grafom:



5.1.3. V psevodokodi opišite algoritem za reševanje problema 5.1.2 v poljubnem de Bruijnovem grafom.

```
def de_briujn(k, n):
    """
    De Bruijn sequence for alphabet k
    and subsequences of length n.
    """
    try:
        # let's see if k can be cast to an integer;

```

```

# if so, make our alphabet a list
_ = int(k)
alphabet = list(map(str, range(k)))

except (ValueError, TypeError):
    alphabet = k
    k = len(k)

a = [0] * k * n
sequence = []

def db(t, p):
    if t > n:
        if n % p == 0:
            sequence.extend(a[1:p + 1])
    else:
        a[t] = a[t - p]
        db(t + 1, p)
        for j in range(a[t - p] + 1, k):
            a[t] = j
            db(t + 1, t)
db(1, 1)
return "".join(alphabet[i] for i in sequence)

print(de_briujn(2, 3))
print(de_briujn("abcd", 2))

```

vir: https://en.wikipedia.org/wiki/De_Bruijn_sequence

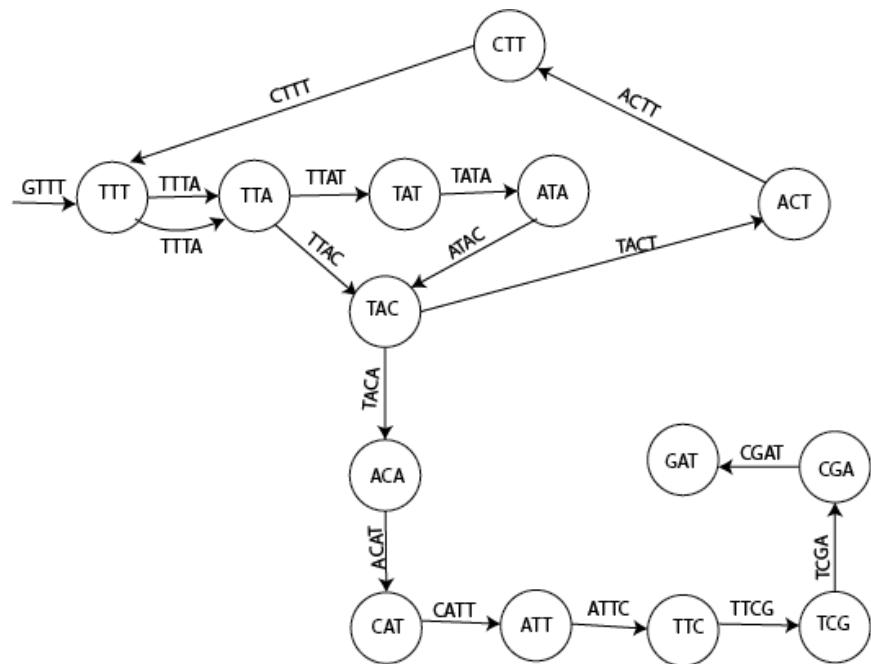
5.1.1

Celotno zaporedje lahko sestavimo na DVA različna načina:

GT_{TTT}
 TTTA
 TTAC
 TACT
 ACTT
 CTTT
 TTTA
 TTAT
 TATA
 ATAC
 TACA
 ACAT
 CATT
 ATTC
 TTCG
 TCGA
 CGAT

GT_{TTT}
 TTTA
 TTAT
 TATA
 ATAC
 TACT
 ACTT
 CTTT
 TTTA
 TTAC
 TACA
 ACAT
 CATT
 ATTC
 TTCG
 TCGA
 CGAT

5.1.2



Da lahko iz grafa enolično sestavimo zaporedje mora veljat, da je graf Hamiltonov in Eulerjev. Kar pomeni, da v grafu obstajata istoimenska obhoda.

naloga 5.1

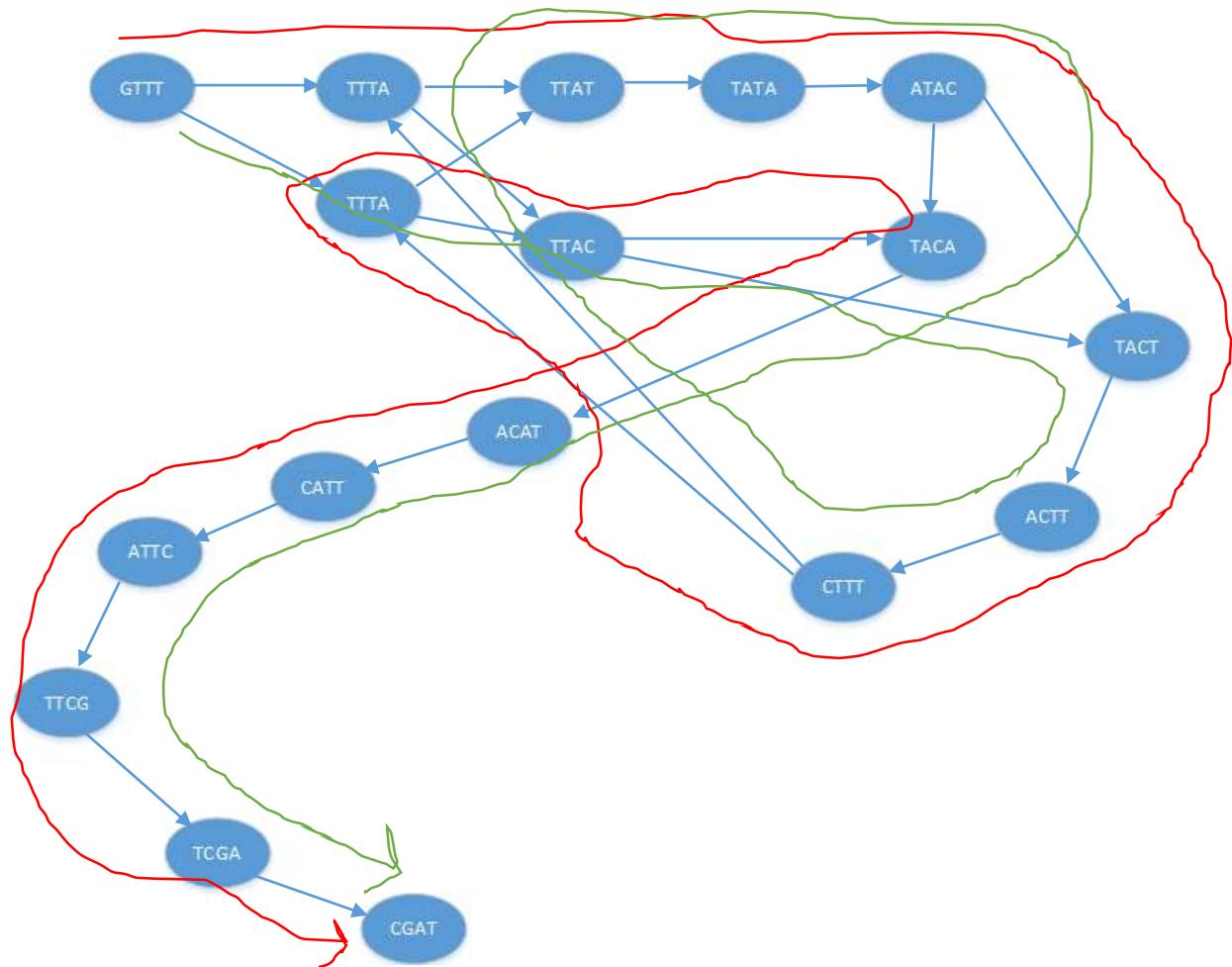
Teorija grafov pri sestavljanju nizov. Poenostavljeni povedano so DNK molekule so zelo dolga zaporedja črk iz abecede {A, C, G in T}. Ljudje še nismo izumili stroja, ki bi lahko prebral zaporedje poljubno dolgega zaporedja, ampak znamo prebrati (na najnovejših strojih) tja do par sto znakov dolgo zaporedje. Tako smo se lotili sestavljanja zaporedja nukleotidov posredno. Vzamemo veliko kopij preiskovane DNK in vsako razrežemo na mestih, katerih ne moremo določiti v naprej določiti. Velja zgolj to, da so rezani koščki dovolj kratki, da lahko v njih določimo zaporedje nukleotidov. Postopku razrezovanja rečemo razrez s šibrovko (angl. shotgun sequencing). Delčkom, ki smo jih dobili in jim lahko nato določimo zaporedje nukleotidov, rečemo odčitki (angl. reads). V tej nalogi boste sestavljali odčitke v izvorno DNK. Dolžina odčitka naj bo k. Odčitki se načeloma lahko začnejo na kateremkoli znaku zaporedja. Primer: za zaporedje ATAGCTAGTA in k=5 dobimo naslednje možne odčitke:

ATAGCTAGTA
ATAGC
TAGCT
AGCTA
GCTAG
CTAGT
TAGTA

Zaporedna odčitka se ujemata v k-1 zaporednih znakih.

5.1.1 Koliko različnih zaporedij lahko sestavite iz naslednje množice odčitkov ($k=4$)? Pri tem upoštevajte, da vrstni red izpisa odčitkov ni pomemben. (Namig: ali lahko ugotovite, kateri odčitek predstavlja začetek zaporedja)?

Da sem ugotovi kje se graf začne in konča sem najprej narisal vsa vozlišča in jih povezal. Ugotovil sem da se zaporedje začne z odčitkom GTTT in konča z odčitkom CGAT.

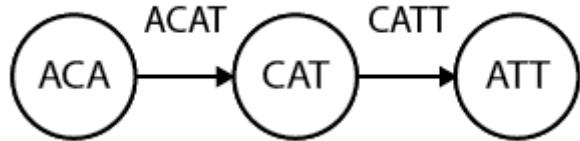


Na podlagi grafa dobimo 2 možna obhoda, se pravi da sta možna 2 niza, ki zajemata vse podane odčitke

GTTTACTTTATACATTGAT - zelen

GTTTATACTTTACATTGAT - rdeč

5.1.2 Pri reševanju problema si lahko pomagamo z de Bruijnovimi grafi. De Bruijnov graf predstavlja odčitek kot usmerjene povezave, kot vozlišča pa njihove pripone in predpone (oboje dolžine $k-1$). Primer: odčitka dolžine $k=4$ za zaporedje ACATT lahko predstavimo z grafom:



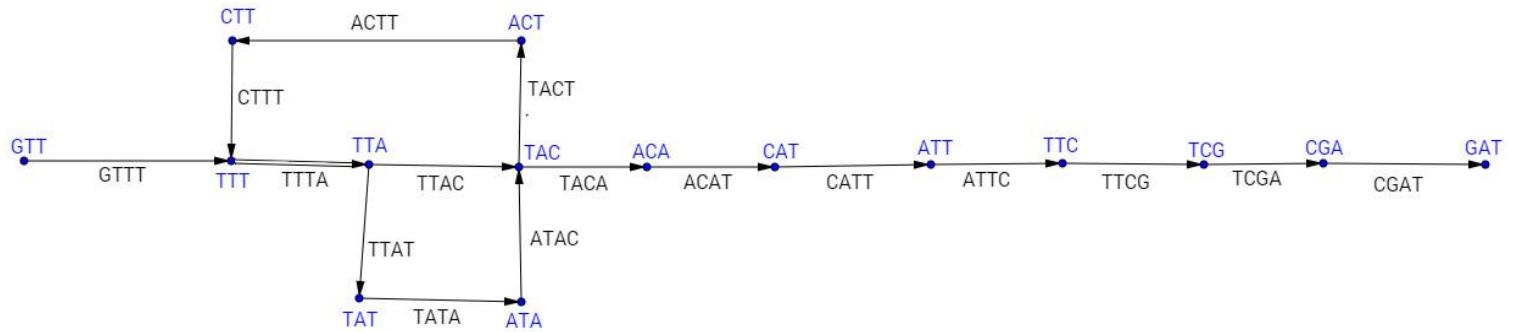
Kaj mora veljati za takšen graf, da lahko iz njega enolično sestavimo zaporedje? (Namig: pri sestavljanju moramo vsak odčitek upoštevati natančno enkrat.)

Za graf morajo veljati nasledne lastnosti :

- Graf mora biti povezan
- Graf mora biti usmerjen
- V vozlišču umamo $k-1$ črk na povezavah pa imamo celotne očitke sepravi k črk koliko je dolg odčitek
- Vsa vozlišča morajo imeti vsaj 1 vhodno in vsaj 1 izhodno povezavo, razen končno in začetno vozlišče ki imata le vhodne oz samo izhodne povezave
- Povezav v grafu mora biti vsaj toliko kot je število vozlišč-1 lahko pa jih je tudi več
- Odčitki se ne smejo ponavljati

Naloga 5.1.1

Po nasvetu z druge točke sem za pomoč pri nalogi naredil še de Bruijnov graf.



Našel sem dve zaporedji:

- GTTTATACTTTACATTGAT
- GTTTACTTTATAACATTGAT

Ker se odčitek TTTA ponovi, je zato povezava iz vozlišča TTT v TTA dvojna (lahko jo odčitamo dvakrat).

Naloga 5.1.2

Graf mora:

- **Imeti izvorno in ponorno vozlišče** (prvo ima le eno povezavo, ki kaže iz vozlišča, drugo pa ima le eno povezavo, ki kaže v vozlišče).
- **Biti Eulerjev** (vsako povezavo lahko obiščemo le enkrat).
- **Imeti vsa vozlišča uravnotežena** (enako število povezav v vozlišče kot povezav iz vozlišča), razen izvornega in ponornega vozlišča.

Naloga 5.1.3

```
edge[] path = {};  
  
function findEuler(node current){  
    if(node.hasEdges())  
        for(edge e : node.edges)  
            if (!edge.isBridge() && !e.visited){  
                e.visited = true;  
                path.add(e);  
                findEuler(e.nextNode);  
            }  
    return;  
}  
  
path = findEuler(startNode);
```

*Psevdokoda je narejena po Fleury-jevem algoritmu, ki v vsakem vozlišču preveri, če ima povezave iz vozlišča. Pot nadaljuje po tisti, ki še ni bila obiskana ter ne deli grafa na dva dela (ne loči povezav ki še niso bile obiskane).

5.1. DOMAČA NALOGA

1. Naloga

GT T TACTTATACATT C GAT	GT T TATACTTACATT C GAT
GT T T	GT T T
TT T A	TT T A
TTA C	TTA T
T T A C	T T A T
T A C T	A T C A
C T T T	T A C T
TT T A	A C T T
TTA T	C T T T
T T A A	T T A T
A T A C	T A C A
T A C A	A C A T
A C A T	C A T T
C A T T	A T T C
A T T C	T T C G
T T C G	T C G A
T C G A	C G A T

2. Naloga

Vsak De Bruijn graf je Eulerjev in Hamiltonov. Torej po vsaki povezavi in v vsakem vozlišču samo enkrat.

Skozi vozlišča gremo lahko večkrat!

5.1.1

Rezultat 1: GTTTACTTATACATTGAT

GT
TTA
TTAC
TACT
ACTT
CTT
TTA
TTAT
TATA
ATAC
TACA
ACAT
CATT
ATTC
TTCG
TCGA
CGAT

Rezultat 2: GTTTATACTTACATTGAT

GT
TTA
TTAT
TATA
ATAC
TACT
ACTT
CTT
TTA
TTAC
TACA
ACAT
CATT
ATTC
TTCG
TCGA
CGAT

V obeh primerih predstavlja začetek GTTT, to pa smo vnaprej vedli zato, ker se noben drug odčitek ne konča z GTT. Ker se noben drug ne konča z GTT, nemoremo odčitka GTTT uporabiti nikjer drugje kot na začetku, kjer nato nadaljujemo niz TTT.

5.1.2

V takem grafu, lahko posamezen odčitek uporabimo samo enkrat. Prav tako rabi veljati, da mora naslednik vozlišča imeti prvih $k-1$ črk enakih kot predhodnik zadnjih $k-1$. Primer: V1 ima zaporedje AGGT, torej se more vozlišče V2 začeti na GGT. Zadnja, k -ta črka, je poljubna.

Da lahko enolično sestavimo zaporedje, rabimo najti Hamiltonov cikel. Torej, da sestavimo neko zaporedje, rabimo obiskati vse odčitke točno enkrat.

5.1.3

- 1.)Vzamemo k -ti odčitek iz seznama
- 2.)Preverimo, če se prve 3 črke katerega drugega odčitka ujemajo z zadnjimi 3 našega. Če se, ga povežemo na prejšnega in izbrišemo iz seznama.
- 3.)Ponovi korak 2 dokler seznam ni prazen
 - 3.1)Če ni novih parov, na novo inicializiramo seznam, povečamo k in se vrnemo na korak 1 in vzamemo nov odčitek.
 - 3.2)Če je seznam prazen, gremo na izbran odčitek iz koraka 1 in se sprehodimo čez vse njegove naslednike.

Naloga 5.1

5.1.1

Začetni odčitek je GTTA saj v njega ni mogoče priti iz drugega odčitka. Možne kombinacije sta dva.

GTTTATACTTTACATTCGAT

GT

TTA

TTAT

TATA

ATAC

TACT

ACTT

CTTT

TTTA

TTAC

TACA

ACAT

CATT

ATTC

TTCG

TCGA

CGAT

GTTTACTTTATACATTCGAT

GT

TTA

TTAC

TACT

ACTT

CTTT

TTTA

TTAT

TATA

ATAC

TACA

ACAT

CATT

ATTC

TTCG

TCGA

CGAT

5.1.2

Graf mora imeti Eulerjev sprehod. Vsa vozlišča mora imeti enako število vhodnih in izhodnih poveza ali pa dve vozlišči od katerih ima prvo vozlišče eno vhodno povezavo več kot izhodno in drugo, ki ima eno vhodno povezavo manj kot izhodno, ostala pa imajo enako vhodnih in izhodnih povezav.

5.1.2

Uporabimo Hierholzerjev algoritem.

Izberemo katero koli začetno vozlišče v in sledimo povezavam iz tega izhodišča dokler se ne vrnemo nazaj v njega. Ker so ostala vozlišča sode stopnje se v njih zagotovo ne bomo ustavili. Dokler so v temu sprehodu vozlišča katerih povezave še niso bile obiskane, si izmed njih izberemo vozlišče u in naredimo nov sprehod, dokler se ne vrnemo v njega. Ko se vrnemo v u nov sprehod združimo z prejšnjim.

Algoritem ima linearno časovno zahtevnost.

**Kaj pa, če je graf vrsta (tak, kot je v navodilih naloge
5.1.2; končamo v ponornem vozlišču)?**