PRIMENJENA BIOINFORMATIKA ZADACI

Napomena: Ako neki podatak neće da se otvori pod zadatim imenom, searchovati neki keyword iz imena u projektu i videti pravi path, ponekad izostave da je u podfolderu. Fajlovi u Jupyteru mogu da se nađu skroz levo u drugom horizontalnom tabu.

Rosalind

Precizne formulacije zadataka na: http://rosalind.info/problems/list-view/

Counting DNA Nucleotides

Prebrojati broj pojavljivanja 'A', 'C', 'G', i 'T' u nukleotidu zadadtom kao string dužine najviše 1000. Ispisati kao četiri integera odvojenih razmakom.

```
s = input()
a, g, c, t = 0, 0, 0, 0
for ch in s:
    if ch=='A':
        a+=1
    elif ch=='C':
        c+=1
    elif ch=='G':
        g+=1
    elif ch=='T':
        t+=1
print(a, c, g, t)
```

Transcribing DNA into RNA

Transkrajbuj DNK u RNK tj. zameni 'T' sa 'U' u datom stringu.

```
s = input()
res = ''
for c in s:
    if c=='T':
        res+='U'
    else:
        res+=c
print(res)
```

Complement a Strand of DNA

Komplementiraj DNK tako što zameniš 'A' i 'T' jedno s drugim i 'C' i 'G' jedno s drugim gde god da se pojave u stringu, a onda obrneš ceo string naopačke.

```
s = input()
res = ''
```

```
for c in s:
    if c=='T':
        res+='A'
    elif c=='A':
        res+='T'
    elif c=='C':
        res+='G'
    elif c=='G':
        res+='C'
print(res[::-1])

'''Wtf is this -1?
array[range_start : range_end : step],
where if step=2 for example it would print
every second value, while when it is negative
it prints them backwards, reversing the list'''
```

Week 3

```
import pysam
```

pysam je biblioteka koju koristimo da bismo čitali fasta, fastq, bam, i ostale tipove fajlova u kojima se često čuvaju informacije o genomu. Importovaćemo je u praktično svaki zadatak od sad.

Zadatak 1

Naći fasta fajl na lokaciji */sbgenomics/project-files/example_human_reference.fasta* i odraditi sledeće:

- kreirati pysam Fastafile parser
- outputovati imena sekvenci
- outputovati ukupan broj sekvenci
- fetchovati cele sekvencu, ispisati koliko je dugačka, i ispisati prvih 100 baza

```
import pysam

# Pravimo FastaFile parser, koji je objekat iz pysam biblioteke, sa linkom datim u zadatku
fasta = pysam.FastaFile('/sbgenomics/project-files/example_human_reference.fasta')

# Ispisujemo imena svih sekvenci u referentnom genomu
for sequence_name in fasta.references:
    print(sequence_name)

# Ispisujemo broj sekvenci
print(len(fasta.references))

# Ispisujemo cele sve sekvence, fetch vraća string koji sadrži očitanu sekvencu genoma
# Pošto imamo samo jednu sekvencu, samo ona se ispisuje
for sequence_name in fasta.references:
    print(fasta.fetch(sequence_name))

# Ispisujemo dužinu sekvenci (ovo je tuple koji ima onoliko elemenata koliko sekvenci
imamo,
```

```
# u ovom slučaju 1)
print(fasta.lengths)

# Ispisujemo prvih 100 baza u našoj sekvenci
# fetch kao druga dva argumenta može da primi početak i kraj intervala i onda uzima samo taj deo sekvence
# interval je [a,b) tj. u ovom slučaju prvih 100 baza, pošto krećemo od 0
for sequence_name in fasta.references:
    print(fasta.fetch(sequence_name,0,100))
```

Zadatak 2

Naći fasta fajl na lokaciji */sbgenomics/project-files/Homo_sapiens_assembly38.fasta* i odraditi sledeće:

- kreirati pysam Fastafile parser
- outputovati imena sekvenci (za svaki contig - read, iščitavanje, sekvenca, bitno za zapamtiti jer može da zbuni u tekstu zadatka)
- Koliko je dugačak hromozom 5?
- fetchovati sekciju hromozoma 17:43044295-43125370
    - Koji je "CG content" ovog regiona (procenat C i procenat G baza)?
    - Koji je najčešći trimer?
- fetchovati bazu u hromozomu 1:248755121 i ispisati koja je
- fetchovati region hromozoma 1:50000-50100 i ispisati koji je "base composition"

```
import pysam

fasta = pysam.Fastafile('/sbgenomics/project-files/Homo_sapiens_assembly38.fasta')

for sequence_name in fasta.references:
    print(sequence_name)

# Ovo je samo kraći zapis za len(fasta.references), jer se ta vrednost već čuva u atributu nreferences
print(fasta.nreferences)

# Koliko je dugačak hromozom 5?
print(fasta.get_reference_length('chr5'))
# Napomena: Radi i print(len(fasta.fetch('chr5'))), samo je sporije

# Fetchujemo hromozom 17, druga dva argumenta su zadati interval
region=fasta.fetch('chr17',43044295,43125370)
print(region)

# CG content je zbir procenata G i C baza u regionu
print((region.count('G')+region.count('C'))/len(region))

# Koji je najčešći trimer?
trimer_dict={}

# Idemo do dužine regiona -3 jer nema trimera koji počinju od poslednje 2 pozicije
```

```
# (jer nema još 3 baze na kraju), a pošto niz počinje od 0 oduzima se 3, a ne 2
for i in range(len(region)-3):
    try: # trimer se pojavio još jednom, uvećavamo count
        trimer_dict[region[i:i+3]]+=1
    except KeyError: # ako nije pronađen key u dictionaryju, dodaj taj key
        trimer_dict[region[i:i+3]]=1

# (key, value) par za svaki trimer među ključevima
trimer_list=[(trimer_dict[trimer],trimer) for trimer in trimer_dict.keys()]

# Kad sortiramo listu broja pojavljivanja trimera, pošto je automatski ascending, posljednji će biti najčešći,
# pa zato -1, a onda pristupamo prvom elementu te liste pošto smo napravili [key, value] listu, pa je value
# na poziciji 1
print(sorted(trimer_list)[-1][1])
```

Week 3: Spisak bitnih funkcija, biblioteka, i atributa

import pysam

pysam.Fastafile

fasta.references

fasta.nreferences

fasta.fetch(sequence_name, region_start, region_end)

fasta.lengths

fasta.get_reference_length('chr5')

region.count('G')

try:

dict[key] += 1

except KeyError:

dict[key] = 1

sorted(lista)

Week 4

kmerize

Napiši funkciju *kmerize*, koja uzima string *read* i dužinu k-mera *k*, a vraća listu svih jedinstvenih k-mera koji postoje u stringu.

```
def kmerize(read, k):
    kmers = []
    for i in range(len(read) - k + 1):
        kmer = read[i:i+k]
        if kmer not in kmers:
            kmers.append(kmer)
    return kmers
```

kmer_count

Napiši funkciju *kmer_count* koja prima string *read* i dužinu k-mera *k*, a vraća dictionary koji sadrži sve jedinstvene k-mere i broj njihovih pojavljivanja.

```
def kmer_count(read, k):
    kmers = {}
    for i in range(len(read) - k + 1):
        kmer = read[i:i+k]
        if kmer not in kmers:
            kmers[kmer] = 1
        else:
            kmers[kmer] = kmers[kmer] + 1
    return kmers
```

de_brujin_ize

Napiši funkciju *de_bruijn_ize* koja prima string *read* i dužinu k-mera *k*, a vraća listu koja za svaki k-mer sadrži njegov levi k-1-mer i desni k-1-mer kao par.

```
def de_bruijn_ize(read, k):
    edges = []
    nodes = set()
    for i in range(len(read) - k + 1):
        edges.append((read[i:i+k-1], read[i+1:i+k]))
        nodes.add(read[i:i+k-1])
        nodes.add(read[i+1:i+k])
    return nodes, edges

# Ne kapiram ovo sasvim, potencijalno postoji greška
```

Week 5

Exercise 1 - a simple seeding aligner step

Podaci koje ćemo koristiti:

- Read: 'ACATACACATGTCCTGTTTTGATGTCCTATAATTAATTTTCTCTCCGTTTTTAACTTTTATCTATCTTATTAATGT'
- *example_reference.fasta*

Seed faza algoritma za align:

- implementirati aligner baziran na hash tabeli

    - može da se koristi dictionary da se napravi index
    - napravi index za svaki k-mer u sekvenci
- Da bismo mapirali read, treba da nađemo lokacije svakog k-mera u tom readu
    - Naći region na koji je mapirano najviše k-mera
    - Obrati pažnju na offset između referentnog genoma i reada!

Objašnjenje:

Pravimo hash mapu (dictionary) u kojoj su keys k-meri iz referentnog genoma, a values liste indeksa na kojima se nalazi isti taj k-mer tj. gde smo uspeli da matchujemo taj k-mer. Iz ovog dictionaryja kad ga napravimo možemo pročitati broj jedinstvenih k-mera, kao i broj k-mera koji imaju više od jednog matcha (tj. za koje postoji kolizija).

Koristimo tu hash mapu da napravimo seed funkciju. Poenta seed funkcije je da alignuje zadati read na referentni genom na osnovu matchovanih k-mera. Ako prođemo kroz ceo read i u našoj hash tabeli za svaki k-mer iz reada vidimo gde se taj k-mer nalazi u referentnom genomu, možemo da oduzmemo offset od početka reada i vidimo gde bi read počinjao ako bismo ga alignovali na osnovu tog k-mera.

Npr. ako imamo referentni genom ATTGCATGCTT, k=3, i imamo read TGCT.

k-meri u readu: TGC, GCT

hash tabela za referentni genom:

trimeri	**pozicije**
ATT	[0]
TTG	[1]
TGC	[2, 6]
GCA	[3]
CAT	[4]
ATG	[5]
GCT	[7]
CTT	[8]

Sad gledamo gde smo dobili match sa k-merima iz reada. TGC u readu počinje na poziciji 0, a GCT na poziciji 1.

1. TGC: 2-0 = 2
2. TGC: 6-0 = 6
3. GCT: 7-1 = 6

To znači da se read može alignovati na jedan od sledeća dva načina:

`ATTGCATGCTT`

`--TGCT-----` - na osnovu 1.

`------TGCT-` - na osnovu 2. i 3.

Kao što vidimo, read nije potpuno matchovan na osnovu 1. ali je jako blizu, pa se ovakvo alignovanje može iskoristiti da se nađu bliske sekvence i otkriju mutacije ili greške u čitanju.

```
import pysam
# videćemo na dnu koda šta će nam ove dve biblioteke, just roll with it
from collections import Counter
from itertools import chain

fasta = pysam.FastaFile("/sbgenomics/project-
files/example_human_reference.fasta").fetch('20')
read = 'ACATACACATGTCCTGTTTTGATGTCCTATAATTAATTTTCTCTCCGTTTTTAACTTTTATCTATCTTATTAATGT'
k = 10

# Kreiramo hash tabelu sa indeksima
def create_index(fasta, k):
    kmers = {}
    for i, x in enumerate(fasta[:-k]):
        kmer = fasta[i:i+k]
        if kmer not in kmers:
            kmers[kmer] = []
        kmers[kmer].append(i)
    return kmers

index = create_index(fasta, k)
print("Number of unique k-mers:", len(index))
print("Number of k-mers with collisions:", len([k for k,v in (index).items() if len(v)>1]))

# Napomena: Na času smo radili 3 različite implementacije za seed, ali navodim ovu jer je
najrazumljivija,
# a koga zanimaju kompaktnije ili efikasnije implementacije neka gleda sa časa

def seed_read2(index, k, read):
    result = []

    # gledamo svaki k-mer u readu
    for i in range(len(read)):
        # ako smo prošli poslednji indeks gde ima dovoljno do kraja da stane još jedan k-
mer, stajemo
        if i+k-1 == len(read):
            break

        kmer_results = []

        # za svako pojavljivanje k-mera iz reada u referentnom genomu
        for kmer_position in index[read[i:i+k]]:
            # vraćamo poziciju na koju bi se read alignovao na osnovu tog k-mera na
referentni genom
            kmer_results.append(kmer_position-i)

        result.append(kmer_results)

    # chain.from_iterable(result) pretvara result, koji je lista manjih listi, u jednu
    # listu čiji su elementi elementi manjih listi redom
```

```
    # napomena: return type nije prava lista nego generator, što znači da se može pročitati
samo jednom
    # Counter je klasa koja služi za brojanje stvari, sadrži dictionary u kom je key
element liste, a
    # value broj pojavljivanja tog elementa u listi. Ovime ćemo izbrojati koliko puta je
read alignovan
    # na koju poziciju tj. koliko k-mera iz reada se mapiralo počevši od te pozicije na
referencu
    return Counter(chain.from_iterable(result))
```

Exercise 2 - Smith Waterman by hand

Smith Waterman je DP (dinamičko programiranje) algoritam koji služi da se proceni najbolje mesto da se alignuju dve niske. Pošto je algoritam kvadratne složenosti i morao bi skroz ispočetka da se pokreće za svaki novi read, obično se primenjuje za lokalni alignment, a ne na celom referentnom genomu. Algoritam radi tako što dodaje neki predefinisani broj za svaki match dva slova, a oduzima neki za mismatch (tačkasta mutacija tj. jedna mutirana baza) ili preskakanje (kod delecije i insercije, što su vrste mutacije kada se jedna baza obriše ili doda u genom gde ne treba).

Pravimo tabelu u kojoj su kolone i redovi slova iz niski koje alignujemo, sa dodatom kolonom i redom na početku za obe niske. Prazna kolona i red se popunjavaju nulama na početku.

Tabelu za reči *a* i *b* popunjavamo po formuli:

$$v_{i,j} = max(0, \quad v_{i-1,j} + gap\_score, \quad v_{i,j-1} + gap\_score, \quad \left\{ \begin{array}{c} v_{i-1,j-1} + match\_score, \ ako\ je\ a[i] = b[j] \\ v_{i-1,j-1} + mismatch\_score, \ ako\ je\ a[i] \neq b[j] \end{array} \right\})$$

gap_score se još često zove i *indel_score* (bitno je da bi se razumela formulacija zadatka.

Vrednosti *match_score*, *mismatch_score*, i *indel_score* biće unapred zadate, i obično je *match_score* pozitivan, a druga dva negativna. U tabeli match i mismatch predstavljaju dijagonalno kretanje, a gap horizontalno ili vertikalno.

Tabela se popunjava po redovima redom, gledaju se susedna tri polja (gore levo, gore, i levo) i upisuje se maksimum na osnovu gore navedene formule.

Primer za reči ATTCAGCT i ATCAGTCT, kada je:

- *match_score* = +2
- *mismatch_score* = -1
- *indel_score* = -2

		A	T	T	C	A	G	C	T
	0	0	0	0	0	0	0	0	0
A	0	2	0	0	0	2	0	0	0
T	0	0	4	2	0	0	1	0	2
C	0	0	2	3	4	2	0	3	1
A	0	2	0	1	2	6	4	2	2
G	0	0	1	0	0	4	8	6	4
T	0	0	2	3	1	2	6	7	8
C	0	0	0	1	5	3	4	8	6
T	0	0	2	2	3	4	2	6	10

Kada smo napravili ovu tabelicu radimo traceback od maksimuma iz tabele do nule iz koje je taj maksimum krenuo. Prvo slovo nakon te nule je slovo od kog je optimalno matchovati ta dva stringa. Napomena: Nismo kodirali traceback na časovima, samo popunjavanje tabele.

Exercise 3 - Smith Waterman simplified implementation

Algoritam je objašnjen kod Excercise Exercise 2 - Smith Waterman by hand.

```
# dodajemo numpy da nam popuni matricu nulama
import numpy

# zadajemo inicijalne vrednosti
sequence1 = "atcagtct"
sequence2 = "attcagct"

indel_score = -2
match_score = 2
mismatch_score = -1

sequence1 = "-" + sequence1
sequence2 = "-" + sequence2

score_table = numpy.zeros((len(sequence1), len(sequence2)))

def calculate_possibilities(score_table, i, j, sequence1, sequence2):
    values = []
    # Dijagonalan prilaz tj. match ili mismatch
    if sequence1[i] == sequence2[j]:
        values.append(score_table[i-1][j-1] + match_score)
    else:
        values.append(score_table[i-1][j-1] + mismatch_score)
    # Odozgo ili odozdo tj. gap
    values.append(score_table[i][j-1] + indel_score)
    values.append(score_table[i-1][j] + indel_score)
```

```
    # Dodajemo 0, koja je max u slučaju da sve ostalo ide u negativno
    values.append(0)

    return values

for i in range(1,len(sequence1)):
    for j in range(1,len(sequence2)):
        score_table[i][j] = max(calculate_possibilities(score_table, i, j, sequence1,
sequence2))
```

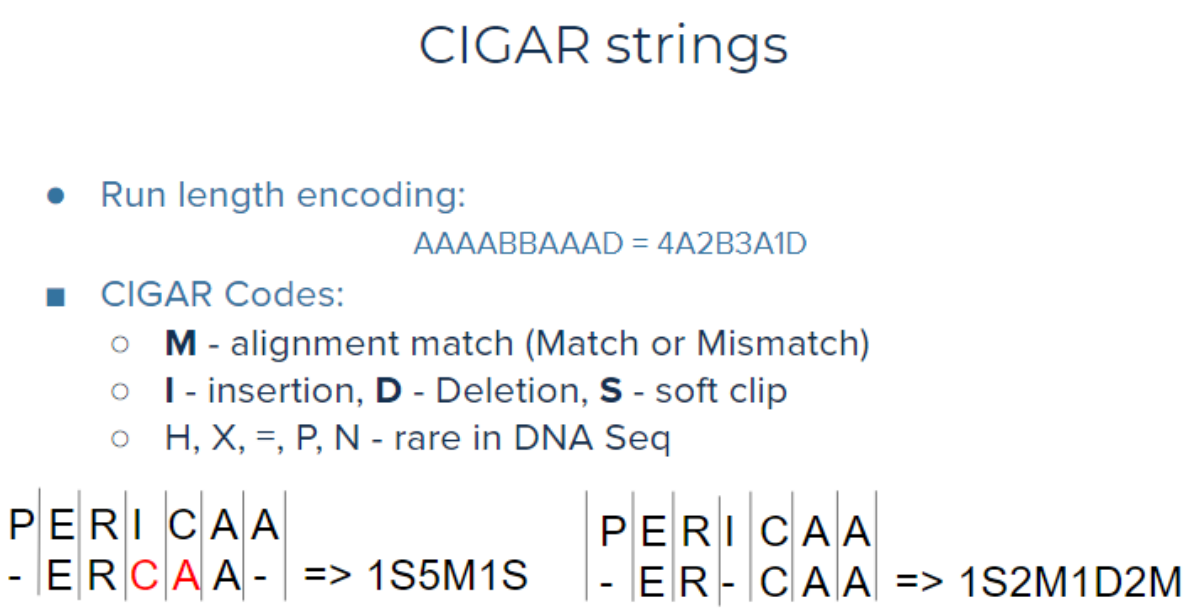
Exercise 4 - Genome browser

nije rađen na času

Exercise 5 - Pysam

.bam tip fajla se koristi, BAM sadrži:

- pročitanu sekvencu i kvalitet (preciznost) iščitavanja za svaku bazu
- poziciju (hromozom i prva baza koja se poklapa) za svaki read
- CIGAR string :



- flagove (read has a pair, read is aligned)
- read pair position??
- druge opcionalne stvari

```
import pysam

bam = pysam.AlignmentFile('/sbgenomics/project-files/merged-tumor.bam')

# Ispis podataka o prvom readu
for x in bam:
```

```
    print(x)
    break

# Broj nemapiranih readova
unmapped_count = 0

for x in bam.fetch():
    if x.is_unmapped:
        unmapped_count +=1

print("Number of unmapped reads: ", unmapped_count)



# Ispis raznih informacija

print("Total number of reads:", len([x.mapping_quality for x in bam.fetch()]))

print("Number of reads with mapping quality 0:", len([x.mapping_quality for x in
bam.fetch() if x.mapping_quality == 0]))

print("Average mapping quality for all the reads:", sum([x.mapping_quality for x in
bam.fetch()])/len([x.mapping_quality for x in bam.fetch()]))

print("Average mapping quality if reads with 0 mapp quality are filtered out:",
sum([x.mapping_quality for x in bam.fetch() if x.mapping_quality >
0])/len([x.mapping_quality for x in bam.fetch() if x.mapping_quality > 0]))
```

Week 5: Spisak bitnih funkcija, biblioteka, i atributa

import pysam

from collections import Counter from itertools import chain

import numpy

bam = pysam.AlignmentFile('/sbgenomics/project-files/merged-tumor.bam')

for x in bam.fetch(): if x.is_unmapped: unmapped_count +=1

print(len([x.mapping_quality for x in bam.fetch()]))

Week 6

Practice - Code Mini Variant Caller

Neki pakao, preskočen za sad

Week 7

Strukturne varijacije (SV) su velike mutacije (preko 50 baznih parova), kao što su delecije, insercije, duplikacije, inverzije, translokacije.

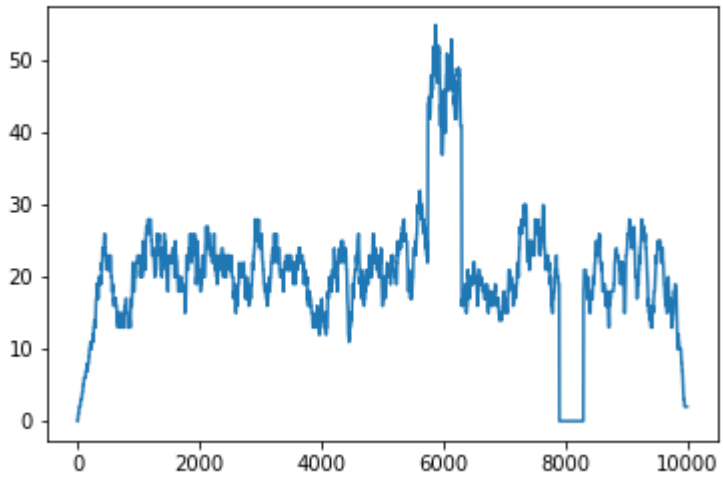
- balansirane - bez promene dužine genoma

    - inverzije
    - translokacije
- nebalansirane - sa promenom dužine genoma
    - insercije
    - CNV (copy number variation) - delecije i duplikacije

Detektovanje delecija:

- read pair - smanjen razmak između baznih parova
- read depth - značajno manji broj readova mapiran na deo referentnog genoma od proseka
- split read - kada se read delimično mapira na dve lokacije tako da je razmak između te dve lokacije manji od dužine reada, i ta dva mapirana dela mogu da se spoje da čine relativno precizan ceo taj read
- assembly - kada se assembluje sekvenca fali neki deo u odnosu na referentni genom

```
import pysam
# matplotlib.pyplot nam treba da štampamo grafove
import matplotlib.pyplot as plt

# Učitavanje BAM fajla
# Napomena: mode može biti 'r' ili 'w' za read/write, 'b' za kompresovane, 'u' za
nekompresovane fajlove
# Kada mode nije eksplicitno naveden, pysam će pokušati da "pogodi" šta nam treba
alignment = pysam.AlignmentFile("/sbgenomics/project-files/simulated_somatic.bam", "rb")

# Računamo read depth za graf, gde je read_depth broj sekvenci mapiranih na neki deo
referentnog genoma
# intervals će biti x osa, a read_depth y osa
interval_length = 5
reference_length = alignment.lengths[0]
# intervals = [0, 5, 10, 15, ...] itd. do dužine reference
intervals = [i*interval_length for i in range(round(reference_length / interval_length))]

read_depth = [
    len(list(alignment.fetch('20', start, end)))
    for start, end in zip(intervals[1:-1], intervals[2:])
]
# zip(intervals[1:-1], intervals[2:])=[(5,10), (10, 15), (15, 20), ...],
# a start i end uzimaju respectively vrednosti iz jednog po jednog tuple-a
# Dakle, proveravamo koliko se readova mapiralo na hromozom '20' za jedan po jedan region



# iscrtavamo graf
plt.plot(intervals[1:-1], read_depth)
plt.show()
```



Na grafu se može videti da negde oko 6000 imamo nagli porast readova, što ukazuje na pojavu duplikacije.

Oko 8000 broj readova značajno opada, što ukazuje na pojavu delecije.

Kako bismo ovo pronašli u kodu?

```
average_coverage = sum(read_depth)/len(read_depth)
previous_depth = read_depth[0] # ovo ćemo posle ažurirati u koraku petlje
deletion_start = 0 # false, ne nalazimo se na početku delecije
duplication_start = 0 # false, ne nalazimo se na početku duplikacije

# Structural Variants
SVs = {
    'DEL': [],
    'DUP': []
}

# Dato nam je da je neophodna promena u read deapth-u od bar 30% average coverage-a
# da bi se nešto smatralo strukturalnom varijacijom
threshold = 0.3 * average_coverage

for curr_bin, depth in enumerate(read_depth):
    #trenutni naglo opadne u odnosu na prethodni -> pocetak delecije ili kraj duplikacije
(pogledaj graf)
    if previous_depth - depth >= threshold and not(deletion_start):
        if duplication_start:
            SVs['DUP'].append((duplication_start * interval_length, curr_bin *
interval_length))
            duplication_start = 0
        else:
            deletion_start = curr_bin
    #trenutni naglo skoci u odnosu na prethodni -> pocetak duplikacije ili kraj delecije
(pogledaj graf)
    if previous_depth - depth < -threshold and not(duplication_start):
        if deletion_start:
            SVs['DEL'].append((deletion_start * interval_length, curr_bin *
interval_length))
```

```
            deletion_start = 0
        else:
            duplication_start = curr_bin
    # ažuriramo prošli depth da bismo mogli da proverimo nagle promene
    previous_depth = depth



# Ispisujemo strukturalne varijacije
print(SVs)
# {'DEL': [(7895, 8290)], 'DUP': [(5740, 6295)]}
```

Strukturalne varijacije se mogu drugačije detektovati i uz pomoć ranije pomenutog CIGAR stringa, ali ovaj način nam ne daje informacije o tome da li je određena tačka početak ili kraj anomalije, niti koja je SV u pitanju. Algoritam, ipak, sledi:

```
alignments = alignment.fetch('20')
breakpoints = []
for read in alignments:
    # 'S' = soft clip
    if 'S' in read.cigarstring and not read.is_secondary:
        cigar = read.cigarstring
        # početak gde se read mapirao na referenci
        start = read.reference_start
        # 'M' = match/mismatch
        if cigar.find('M') < cigar.find('S'):
            # broj matcheva pre clipa (takav je format CIGAR-a, npr 25M1S2A
            # cigar.split('M') = ['25', '1S2A']
            # cigar.split('M')[0] = '25'
            # int('25') = 25 )
            location = int(cigar.split('M')[0])
            breakpoints.append(start + location)
        elif cigar.find('S') < cigar.find('M'):
            breakpoints.append(start + 1)

# Ovakav pristup nam ne govori da li se breakend (mesto pucanja hromozoma)
# odnosi na pocetak ili kraj varijante, ni koji je tip varijante u pitanju.
# Za odgovor na ova pitanja bilo bi potrebno malo izmeniti algoritam, ili
# ga kombinovati sa drugim algoritmima detekcije strukturnih varijanti
print(set(breakpoints))
#{8300, 6300, 5750, 6297, 7900}
```

Week 7: Spisak bitnih funkcija, biblioteka, i atributa

import matplotlib.pyplot as plt

read_depth = [len(list(alignment.fetch('20', start, end))) for start, end in zip(intervals[1:-1], intervals[2:])]

plt.plot(intervals[1:-1], read_depth) plt.show()

if 'S' in read.cigarstring and not read.is_secondary:

cigar = read.cigarstring

cigar.find('M')

location = int(cigar.split('M')[0])

Week 8

Week 9

Spisak svih bitnih funkcija, biblioteka, i atributa

Biblioteke i kako se importuju

import pysam

import pysam

from collections import Counter from itertools import chain

import numpy

import matplotlib.pyplot as plt

Fasta fajlovi

fasta = pysam.Fastafile('/sbgenomics/project-files/Homo_sapiens_assembly38.fasta')

fasta.references

fasta.nreferences

fasta.fetch(sequence_name, region_start, region_end)

fasta.lengths

fasta.get_reference_length('chr5')

BAM fajlovi

bam = pysam.AlignmentFile('/sbgenomics/project-files/merged-tumor.bam')

for x in bam.fetch(): if x.is_unmapped: unmapped_count +=1

print(len([x.mapping_quality for x in bam.fetch()]))

CIGAR string

if 'S' in read.cigarstring and not read.is_secondary:

cigar = read.cigarstring

cigar.find('M')

location = int(cigar.split('M')[0])

Code Snippets

region.count('G')

```
try:
dict[key] += 1
except KeyError:
dict[key] = 1

sorted(lista)

read_depth = [ len(list(alignment.fetch('20', start, end))) for start, end in zip(intervals[1:-1], intervals[2:]) ]

plt.plot(intervals[1:-1], read_depth) plt.show()
```