Eksamensopgave i Bioinformatik og Programmering 2017

Denne eksamensopgave består af to dele, som vægtes lige i bedømmelsen:

1. Et sæt programmeringsopgaver.
2. Et sæt bioinformatikopgaver.

Filer til brug ved eksamen

Udover denne PDF fil som indeholder eksamensopgaverne har du også downloaded tre andre filer fra Digital Eksamen som du skal bruge til at løse eksamensopgaven:

- `progexam.py`: Det er i denne fil du skal skrive de Python funktioner der bedes om i eksamensopgavens programmeringsdel.
- `test_progexam.py`: Det er denne fil du kan bruge til at teste de funktion du skriver i `progexam.py`.
- `bioinfexam.py`: Det er i denne fil du skriver svarene på eksamensopgavens bioinformatikdel.

Sådan løser du programmeringsopgaverne

Som i programmeringsprojekterne fra kurset skriver du din kode i `progexam.py` og kører koden sådan her:

```
python progexam.py
```

Som i programmeringsprojekterne i kurset kan du kan taste din kode sådan her:

```
python test_progexam.py
```

Test scriptet er tilgængeligt som en hjælp til at teste din kode, men du har selv det fulde ansvar for rigtigheden af din kode.

Følgende er *afgørende* for at din eksamensbesvarelse kan evalueres korrekt:

1. Hver funktion skal navngives *præcis* som angivet i opgaven. Funktioner der ikke er navngivet korrekt regnes som ikke besvarede.
2. Funktioner der ikke fuldstænding opfylder opgavens beskrivelse regnes som ikke besvarede (så hellere lave halvdelen af opgaverne helt end alle opgaverne halvt).
3. Du må ikke importere kode fra andre filer du har skrevet eller installeret (e.g., biopython). Det vil sige at du ikke må bruge `import` statements i din fil.
4. Når du afleverer `progexam.py` må den *kun* indeholde definitioner af de funktioner der er beskrevet i eksamensopgaven. Al kode udenfor funktionsdefinitioner skal slettes inden du afleverer, så tjek i god tid inden aflevering om dine funktioner stadig virker når du sletter denne ekstra kode.

Sådan løser du bioinformatikopgaverne

Bioinformatikdelen af eksamensopgaven består af et sæt af "multiple-choice" opgaver der hver dækker et emne. Hver opgave indeholder fire udsagn (1, 2, 3, 4), hvor hvert udsagn enten er sandt (`True`) eller falskt (`False`). Filen `bioinfexam.py` er en Python fil og indeholder en variabel for hvert sådant udsagn. For eksempel: den variabel der hører til udsagn tre i emne syv hedder `emne_7_udsagn_3` og har som default værdien `None`:

```
emne_7_udsagn_3 = None
```

Du besvarer hver opgave ved at udskifte `None` med enten `True` eller `False`. Det er *afgørende* at du bruger `True` og `False`. Du må altså hverken skrive true, false, sandt falskt, 1, 0, `"True"`, `"False"` eller noget andet skørt.

Følgende er *afgørende* for at din eksamensbesvarelse kan evalueres korrekt:

1. Udsagn der ikke er besvaret med `True` eller `False` betragtes som forkert besvarede.
2. Du skal svare på *alle* udsagn. Udsagn der ikke er besvaret betragtes som *forkert* besvarede, så du er bedst tjent med at gætte fremfor ikke at svare.

I nogle af opgaveemnerne refererer statements til en vist illustration. Alt efter størrelsen på din skærm kan du være nødt til at "zoome ind" på illustrationerne i det program du bruger til at vise denne PDF, ellers kan der være detaljer du ikke kan se.

Sådan afleverer du din eksamensopgave i Digital Ekasamen

Inden du afleverer skal du tjekke at `progexam.py` kun indeholder definitioner af de funktioner der er beskrevet i eksamensopgaven. Hvis du har skrevet yderligere kode for at teste dine funktioner skal du slette den inden du oplader din fil.

Du afleverer din eksamensbesvarelse ved at uploade disse to filer til Digital Examen:

- `progexam.py` skal afleveres som *hoveddokument*.
- `bioinfexam.py` skal afleveres som *bilag*.

Eksamensopgaverne starter på næste side

Programming assignments

This assignment is in English to make it most like you what you are used to from the programming exercises in the course.

In the assignment, Python code will look like this:

```
print("hello world")
```

Whenever I refer to Python code inside a sentence it will be styled like `this`.

In this exam assignment, you will work with DNA sequences. For this exam assignment, we will assume that DNA sequences are always upper-case and that they can only contain the characters A, T, G, and C.

The actual assignment starts below.

Happy coding.

Problem 1

You would like to be able to determine how long a DNA sequence is. I.e., how many bases it has.

Write a function, `get_number_of_bases`, which takes one argument:

1. A string, which is a DNA sequence.

The function must return:

- An integer, which represents the number of bases in the sequence.

Example usage: If the function is called like this:

```
get_number_of_bases("ATGA")
```

then it should return:

```
4
```

Problem 2

You would like to be able to determine how many times each base occur in the sequence. You can assume that the only characters in the string are A, T, G, and C.

Write a function, `count_bases`, which takes one argument:

1. A string, which is a DNA sequence.

The function must return:

- A dictionary, which in which keys are strings that represent bases and values are integers that represent the number of occurrences of each base. If a base is not found in the sequence, its count must be zero.

Example usage: If the function is called like this:

```
count_bases("ATGG")
```

then it should return (not necessarily with key-value pairs in the same order):

```
{"A": 1, "C": 0, "G": 2, "T": 1}
```

Problem 3

You would like to be able to get the reverse complement of a DNA string. The reverse complement of a DNA string is one where the sequence of bases is first reversed, and then each base is replaced with its Watson-Crick complementary base.

Write a function, `reverse_complement`, which takes one argument:

1. A string, which is a DNA sequence.

The function must return:

- A string, which represents the reverse complement of the DNA string given as argument.

Example usage: If the function is called like this:

```
reverse_complement("AATTC")
```

then it should return:

```
"GAATT"
```

Problem 4

You would like to be able to count how many times the dinucleotide "CG" (so-called CpG sites) occurs in the sequence.

Write a function, `count_cpg`, which takes one argument:

1. A string, which is a DNA sequence.

The function must return:

- An integer, which represents the number of CpG sites in the sequence.

Example usage: If the function is called like this:

```
count_cpg("ATCGTCGT")
```

then it should return:

```
2
```

Problem 5

You would like to be able to calculate the melting temperature of a piece of double-stranded DNA. If the DNA string has less than 14 bases the formula for calculating melting temperature (Temp) is this (where A, T, C, and G represent the numbers of A, T, C and G in the DNA string):

$$Temp = (A + T) * 2 + (G + C) * 4$$

and if the DNA string has more than 13 bases it is calculated like this:

$$Temp = 64.9 + 41 * (G + C - 16.4)/(A + T + G + C)$$

You must implement a function that applies these two formulas appropriately.

Write a function, `melting_temp`, which takes one argument:

1. A string, which is a DNA sequence.

The function must return:

- A float, which represents the melting temperature of the double-stranded DNA corresponding to the DNA string given as argument.

Example usage: If the function is called like this:

```
melting_temp("ATG")
```

then it should return:

```
8
```

Problem 6

You would like to be able to count all occurrences of kmers (subsequences of length k) in a DNA sequence.

Write a function, `count_kmers`, which takes two arguments:

1. A string, which is a DNA sequence.
2. An integer, which represents the length of kmers to count.

The function must return:

- A dictionary, in which keys are sequences representing kmers found in the sequence, and values are integers representing the number of times each kmer occurs in the sequence.

Example usage: If the function is called like this:

```
count_kmers("ATATAT", 2)
```

then it should return (not necessarily with key-value pairs in the same order):

```
{"TA": 2, "AT": 3}
```

Problem 7

You would like to be able to compute a kmer *profile* of a DNA sequence. It should count the kmers in the DNA sequence for a range of kmer lengths: from a length of 2 to some specified maximum length.

Write a function, `kmer_profile`, which takes two arguments:

1. A string, which is a DNA sequence.
2. An integer, which represents the longest kmer to count.

The function must return:

- A dictionary in which keys are integers representing kmer lengths from 2 up to the number given as the second argument. The value associated with each kmer length must be a dictionary with counts of each kmer of that length found the sequence. The keys in these inner dictionaries must be sequences representing kmers found in the sequence, and values must be integers that represent the number of times each kmer occurs in the sequence. If no kmers are found for some kmer length, then the corresponding inner dictionary should be empty.

Example usage: If the function is called like this:

```
kmer_profile("ATATA", 6)
```

then it should return (not necessarily with key-value pairs in the same order):

```
{2: {"AT": 2, "TA": 2}, 3: {"ATA": 2, "TAT": 1},
4: {"TATA": 1, "ATAT": 1}, 5: {"ATATA": 1}, 6: {}}
```

Problem 8

You would like to be able to determine if a DNA string can fold to form a hairpin with some specified minimum number of consequtive base pairs. We assume that hairpin loops are always at least four bases long and that base pairs in the hairpin can only be Watson-Crick basepairs. Here is an example of a hairpin with five basepairs and a loop of four bases:

```
  C C
C     C
 A - T
 T - A
 A - T
 T - A
 A - T
```

To test if a sequence can form a hairpin with at least four consequtive base pairs, you need to test if the sequence contains any subsequence of length four whose reverse complement is identical to another nonoverlapping subsequence. To take into account that the hairpin loop is at least four bases long, any such two subsequences must be separated by at least four bases.

Write a function, `has_hairpin`, which takes two arguments:

1. A string, which is a DNA sequence.
2. An integer, which represents the minimum number of consequtive base pairs in the hairpins to search for.

The function must return:

- `True` if the sequence contains a hairpin of at least the specified length and `False` otherwise.

Example usage: If the function is called like this:

```
print(has_hairpin("ATATACCCCTATAT", 4))
```

then it should return:

```
True
```

Problem 9

You would like to count the number of substitutions (mutations) between two sequences that are transitions and transversions. Transitions are substitutions between A and G or between C and T. All other substitutions are transversions.

Write a function, `substitutions`, which takes two arguments:

1. A string, which is a DNA sequence.
2. A string, which is a DNA sequence of the same length as argument one.

The function must return:

- A list of two integers. The first integer must represent the number of transitions. The second integer must represent the number of transversions.

Example usage: If the function is called like this:

```
substitutions("AAAAAAAA", "AGAAATTA")
```

then it should return:

```
[1, 2]
```

Problem 10

To identify repeats in a DNA sequence a dot plot can be made. A dot plot is graphical representation of an n by n matrix (n is the sequence length) where a dot is drawn at position i,j if the bases at positions i and j are identical. You would like to do something similar by making a list of lists to represent such an n by n matrix. In that matrix, you can have a value of `1` represent identical bases, and a value of `0` represent different bases.

Write a function, `self_comparison`, which takes one argument:

1. A string, which is a DNA sequence.

The function must return:

- A list of lists of integers. The integers must be either `0` or `1`. The length of the returned list must be the same as the length of the sequence argument. The lists in the big list must also have this length. That way it represents a square matrix. The position at column index `i` and row index `j` should be `1` if the base at sequence index `i` is identical to that at sequence index `j`. It should be `0` otherwise.

Example usage: If the function is called like this:

```
self_comparison("AAATGGG")
```

then it should return:

```
[[1, 1, 1, 0, 0, 0, 0], [1, 1, 1, 0, 0, 0, 0], [1, 1, 1, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 1, 1], [0, 0, 0, 0, 1, 1, 1],
[0, 0, 0, 0, 1, 1, 1]]
```

or if written nicely:

```
[[1, 1, 1, 0, 0, 0, 0],
 [1, 1, 1, 0, 0, 0, 0],
 [1, 1, 1, 0, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 1],
 [0, 0, 0, 0, 1, 1, 1],
 [0, 0, 0, 0, 1, 1, 1]]
```

Problem 11

When making dot plot, you can often choose to not show dots for all identical bases but only for positions where a small subsequence match a subsequence somewhere else in the sequence. To achieve something similar, you can write an improved version of the function you made in the previous problem. This function should not only compare individual bases, but also a specified number of flanking bases, and should only add a `1` to the matrix if both the two individual bases *and* their flanking bases are identical.

Write a function, `better_self_comparison`, which takes two arguments:

1. A string, which is a DNA sequence.
2. An integer, which represents the number of flanking bases at each position that should also match.

The function must return:

- A list of lists of integers. The integers must be either `0` or `1`. The length of the returned list must be the same as the length of the sequence argument. The lists in the big list must also have this length. That way it represents a square matrix. If we call the integer argument `k`, then the difference from the previous problem is the following: the position at column index `i` and row index `j` should only be `1` if the base at sequence index `i` and its `k` flanking bases on both sides are identical to that at sequence index `j` its `k` flanking bases. Note that at the ends of the sequence, where flanking bases will extend past the ends of the sequence, it is not possible to make a valid comparison. When this happens you must put `0` in the matrix.

Example usage: If the function is called like below, then subsequences of length three are compared (the center base plus one flanking base at each side):

```
better_self_comparison("AAATAAA", 1)
```

and it should return:

```
[[0, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 1, 0], [0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 1, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0]]
```

or if written nicely:

```
[[0, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 1, 0],
 [0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 0, 1, 0],
 [0, 0, 0, 0, 0, 0, 0]]
```

Bioinformatikopgaver

Emne 1: Databaser

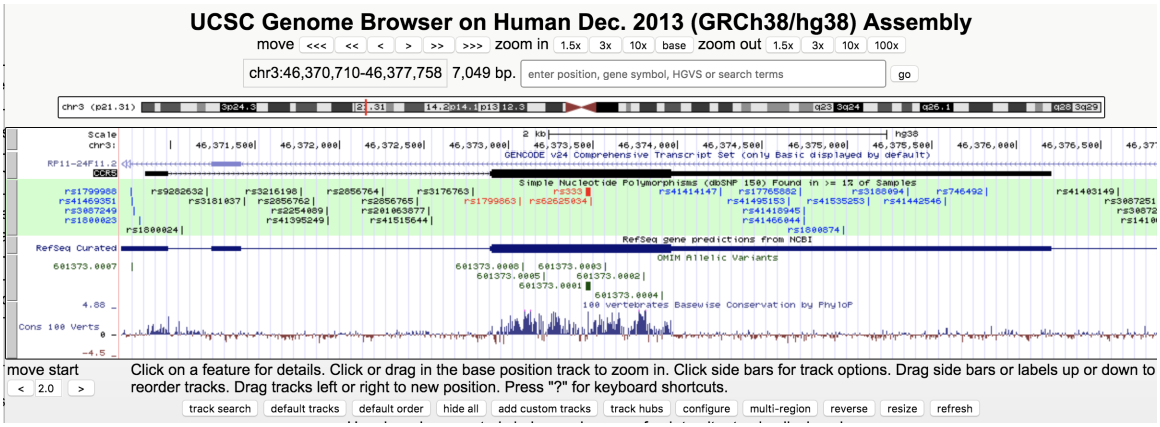
Online databaser giver adgang til et væld af forskellig information. Nogen indeholder rå sekvens information. Andre samler forskellige typer af information relevant for f.eks. gener, proteiner, sygdomme, eller pathways.

Statements:

1. Pfam er en database for protein familier og domæner.
2. Genbank er en database, der indeholder alle offentligt tilgængelige DNA sekvenser.
3. PDB er en database, der indeholder alle offengligt tilgængelige plasmider.
4. OMIM er en database der sammenfatter information om sygdomme der ikke er arvelige.

Emne 2: UCSC genome browser

Figuren nedenfor viser et screenshot fra UCSC genome browseren (du kan være nødt til at zoome ind for at se alle detaljer):



CCR5-delta32 er en variant af CCR5 genet med en deletion af 32 fortløbende nukleotider. Varianten optræder i databaser om rs333.

Statements:

1. CCR5 genet ligger på kromosom tre.
2. SNPen med id rs333 ligger i den protein-kodende region af genet.
3. SNPen med id rs333 er associeret med en sygdom beskrevet i OMIM databasen.
4. SNPen med id rs333 er en frame-shift mutation.

Emne 3: Genetiske sygdomme

Et ægtepar, Yvonne og Mogens, vil gerne have et barn, men er bekymret for om et sådant barn vil få den genetiske sygdom Huntingtons disease. Huntingtons disease skyldes et enkelt gen, HTT, og sygdomsvarianten af dette gen er *dominant*. Både Yvonne og Mogens er helt raske, men både Yvonnes mor og Mogens far havde Huntingtons disease.

Statements:

1. Sandsynligheden for at enten Mogens eller Yvonne er bærer af en sygdomsvariant af HTT (uden at være syg) er mindst 50%.
2. Sandsynligheden for at Mogens og Yvonnes barn er bærer af en sygdomsvariant af HTT (uden at være syg) er mindst 50%.
3. Sandsynligheden for at Mogens og Yvonnes barn får Huntingtons disease er mindst 25%.
4. Det kan udelukkes at Mogens og Yvonnes børnebørn får Huntingtons disease.

Emne 4: Needleman-Wunch og Smith-Waterman

Needleman-Wunch og Smith-Waterman algoritmerne er begge algoritmer til henholdsvis globalt og lokalt alignment af to sekvenser.

Statements:

1. Needleman-Wunch algoritmen identificerer mindst ét parvist alignemnt, der er optimalt givet en scoringsmatrix og en gap score.
2. Der kan være mere end ét optimalt lokalt parwist alignment for en given scoringsmatrix og en gap score.
3. Tiden det kræver at beregne et globalt alignment ved hjælp af Needleman-Wunch algoritmen er proportional med summen af de to sekvensers længde.
4. Smith-Waterman algoritmen vil i nogen tilfælde producere et globalt alignment.

Emne 5: Globalt alignment af to korte sekvenser

Nedenfor ser du to korte DNA sekvenser:

```
sekvens 1: TAAC
sekvens 2: TGAA
```

Lav et globalt alignment af de to sekvenser ved at bruge en match score på 2, en mismatch score på 0 og et gap score på -1 . Du kan gøre dette ved at udfylde en tabel vha. dynamisk programmering.

Statements:

1. Der findes et optimalt alignemnt, der ikke har nogen gaps.
2. Der findes et optimalt alignment der har to gaps.
3. Der er tre optimale alignments.
4. Alle optimale alignments at de to sekvenser har det samme antal alignmentkolonner hvor de to baser er ens.

Emne 6: Lokalt alignment af to korte sekvenser

Nedenfor ser du en dynamisk programmerings matrix for Smith-Waterman alignment af følgende to sekvenser:

```
sekvens 1: AATCG
sekvens 2: AACG
```

S		A_1	A_2	C_3	G_4
	0	0	0	0	0
A_1	0	1	1	0	0
A_2	0	1	2	0	0
T_3	0	0	0	1	0
C_4	0	0	0	1	0
G_5	0	0	0	0	2

Der er brugt en match score på 1, en mismatch score på -1, og en gap score på -2.

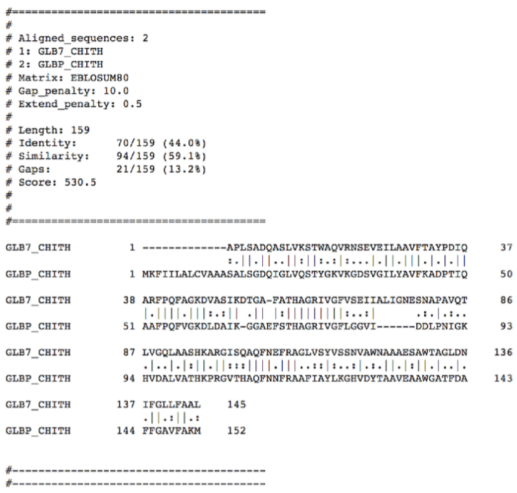
Statements:

1. Tabellen viser at der er præcis ét optimalt alignment.
2. Den optimale aligmentscore er 2.
3. Der findes et optimalt alignment der spænder tre baser i sekvens 1 og to baser i sekvens 2.
4. Der findes et optimalt alignment baserne AA fra sekvens 1 med baserne AA fra sekvens 2.

Emne 7: Parvist alignment ved hjælp af EBIs web tools.

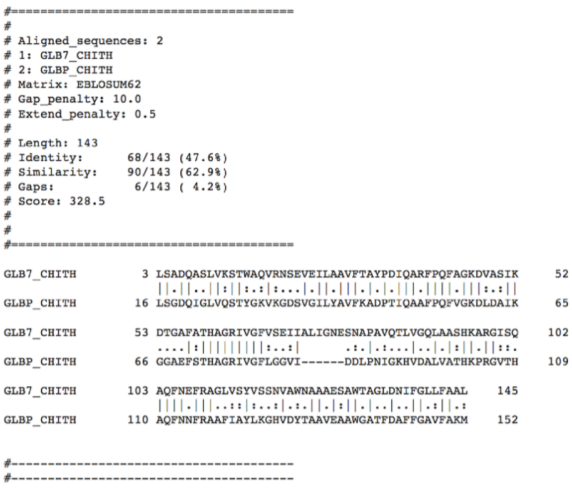
Nedenfor ser du to screen shots med output fra to parvise alignments foretaget ved hjælp af EBIs webservice.

Alignment A



```
#=======================================
#
# Aligned_sequences: 2
# 1: GLB7_CHITH
# 2: GLBP_CHITH
# Matrix: EBLOSUM80
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 159
# Identity:      70/159 (44.0%)
# Similarity:    94/159 (59.1%)
# Gaps:          21/159 (13.2%)
# Score: 530.5
#
#
#=======================================

GLB7_CHITH         1 -------------APLSADQASLVKSTWAQVRNSEVEILAAVFTAYPDIQ     37
                                  :.||.||..||:||:.:|:...:.||.|||.|.|.||
GLBP_CHITH         1 MKFIILALCVAAASALSGDQIGLVQSTYGKVKGDSVGILYAVFKADPTIQ     50

GLB7_CHITH        38 ARFPQFAGKDVASIKDTGA-FATHAGRIVGFVSEIIALIGNESNAPAVQT     86
                     |.||||.|||:.:|| .|| |:|||||||||:..:|       .:.|.:..
GLBP_CHITH        51 AAFPQFVGKDLDAIK-GGAEFSTHAGRIVGFLGGVI------DDLPNIGK     93

GLB7_CHITH        87 LVGQLAASHKARGISQAQFNEFRAGLVSYVSSNVAWNAAAESAWTAGLDN    136
                     .|..|.|:||.|||:::||||.:||...:|:..:|.:.||.|:||.|..|.
GLBP_CHITH        94 HVDALVATHKPRGVTHAQFNNFRAAFIAYLKGHVDYTAAVEAAWGATFDA    143

GLB7_CHITH       137 IFGLLFAAL    145
                        .||.:||.:
GLBP_CHITH       144 FFGAVFAKM    152


#---------------------------------------
#---------------------------------------
```

Alignment B



```
#=======================================
#
# Aligned_sequences: 2
# 1: GLB7_CHITH
# 2: GLBP_CHITH
# Matrix: EBLOSUM62
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 143
# Identity:      68/143 (47.6%)
# Similarity:    90/143 (62.9%)
# Gaps:           6/143 ( 4.2%)
# Score: 328.5
#
#
#=======================================

GLB7_CHITH         3 LSADQASLVKSTWAQVRNSEVEILAAVFTAYPDIQARFPQFAGKDVASIK     52
                     ||.||..||:||:.:|:...:.||.|||.|.|.||||.|||:.:||
GLBP_CHITH        16 LSGDQIGLVQSTYGKVKGDSVGILYAVFKADPTIQAAFPQFVGKDLDAIK     65

GLB7_CHITH        53 DTGAFATHAGRIVGFVSEIIALIGNESNAPAVQTLVGQLAASHKARGISQ    102
                     ....|:|||||||||:..:|       .:.|.:...|..|.:||.||::.
GLBP_CHITH        66 GGAEFSTHAGRIVGFLGGVI------DDLPNIGKHVDALVATHKPRGVTH    109

GLB7_CHITH       103 AQFNEFRAGLVSYVSSNVAWNAAAESAWTAGLDNIFGLLFAAL    145
                     ||||.|||..::|:..:|.:.||.:||.|..|..||.:||.:
GLBP_CHITH       110 AQFNNFRAAFIAYLKGHVDYTAAVEAAWGATFDAFFGAVFAKM    152


#---------------------------------------
#---------------------------------------
```

Statements:

1. Alignment A er et lokalt alignment.
2. Alignment B er et globalt alignment.
3. I alignment A vil et gap med længde seks bidrage til den totale score med en score på -60.
4. Til at producere alignment B er der brugt en substitutionsmatrix, der er bedre egnet til nært beslægtede sekvenser end den, der er brugt til at producere alignment A.

Emne 8: Multipelt alignment

Der findes flere forskellige programmer til multipelt alignment. Et af dem er Clustal Omega, men der er mange andre.

Statements:

1. Det er muligt at identificere et optimalt multipelt alignmet for en given scoringsmatrix og gap score.
2. Clustal Omega er garanteret at producere et optimalt alignment for en given scoringsmatrix og gap score.
3. Multiplet alignment kan identificere homologi mellem fjernt beslægtede sekvenser som ikke kan identificeres ved hjælp af parvist alignment.
4. Forskellige programmer til multipelt alignment producerer samme alignment hvis man bruger den samme scoringsmatrix og gap cost.

Emne 9: Blast

BLAST programmet implementerer en algoritme til at søge med en sekvens i en stor database af sekvenser. NCBI giver online adgang til BLAST mod sekvenserne i Genbank, men man kan også lave sin egen lille database på sin egen computer og så søge mod den.

Statements:

1. BLAST er garanteret at finde det bedste alignment mellem en sekvens i databasen og den sekvens man søger med.
2. Den score, der angives for hver identificeret databasesekvens, representerer en alignment score for et globalt alignment af en sekvens i databasen og den sekvens man søger med.
3. E-værdien angiver sandsynligheden for at finde et match i databasen med samme eller bedre score ved rent tilfælde.
4. Den rapporterede E-værdi kan være større end 1 selvom den sekvens man søger med er helt identisk med det match BLAST finder i databasen.

Emne 10: Modeller for DNA evolution

Jukes-Cantor og Kimura 2-parameter modellerne er modeller for DNA evolution. Begge modeller kan bruges til at udregne den evolutionære afstand mellem to sekvenser ud fra andelen af forskelle mellem dem.

Statements:

1. Jukes-Cantor modellen lader os korrigere for at den samme positition i sekvensen har muteret flere gange.
2. Kimura 2-parameter modellen korrigerer for at adenine og thymine muterer oftere end guanine og cytocine.
3. Både Jukes-Cantor modellen og Kimura 2-parameter modellen antager at alle fire baser optræder lige hyppigt i DNA sekvenser.
4. Hvis 50% af baserne i to alignede sekvenser er forskellige så vil den Jukes-Cantor-korrigerede afstand være udendeligt stor.

Emne 11: PAM og BLOSUM matricer

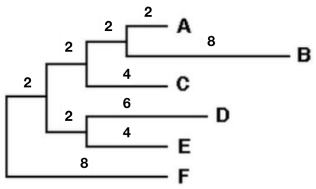
Når man konstruerer substitutionmatricer for proteiner (PAM og BLOSUM matricer), så beregnes hver score ud fra information fra et sæt alignments.

Statements:

1. PAM250 repræsenterer en mindre evolutionær afstand end PAM120
2. PAM62 og BLOSUM62 repræsenterer den samme evolutionære afstand.
3. BLOSUM62 er konstrueret ved ekstrapolation fra en BLOSUM1 matrix
4. Størrelsen på hver indgang i en PAM1 scoringsmatrix repræsenterer hvor ofte vi forventer at se en aminosyre blive erstattet med en anden, når man tager frekvensen af forskellige aminosyrer i betragtning.

Emne 12: Afstande på et træ

Nedenfor ser du et træ for seks sekvenser: A, B, C, D, E og F. Hver grens længde er angivet.



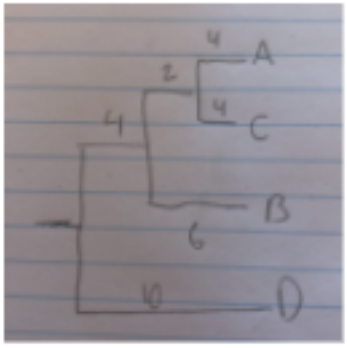
Statements:

1. Afstanden mellem B og C er 14.
2. A og D er tættere beslægtet end E og B.
3. Træets totale grenlængde er 40.
4. Afstandene på træet indikerer at substitutionsraten er den samme i hele træet.

Emne 13: UPGMA

Tabellen nedenfor viser de parvise afstande mellem fire sekvenser: A, B, C og D.

	A	B	C
A			
B	12		
C	8	12	
D	20	20	20



De fire sekvensers indbyrdes afstande kan repræsentres some et træ ved hjælp af clustering algoritmen UPGMA. Det konstruerede træ har ingen rod.

Statements:

1. UPGMA vil begynde med at clustre A og B.
2. Hvis vi byggede et træ med kun sekvenserne A, B og C, så ville vi kunne bruge sekvens D til at sætte rod på dette træ.
3. Den korteste gren i træet har længde 2.
4. Den længste gren i træet har længde 20.

Emne 14: Fylogeni

Metoder til konstruktion af fylogenier kan opdeles i karakterbaserede metoder og metoder der clustrer sekvenser baseret på forudberegnede afstande mellem par af sekvenser.

Statements:

1. Neighbor-Joining er en karakterbaseret metode til konstruktion af fylogeni.
2. PhyML er en karakterbaseret metode til konstruktion af fylogeni.
3. Den konstruerede fylogeni afhænger af den benyttede model for DNA evolution.
4. En fylogenis likelihood er sandsynligheden for den benyttede model for DNA evolution givet det benyttede sekvensdata.

Emne 15: Genome assembly

Den bioinformatiske udfordring at samle sekvens reads til et samlet genom kaldes genome assembly.

Statements:

1. Repeats i en sekvens kan kun samles hvis længden af reads er større end længden af det enkelte repeat.
2. Genome contigs er sat sammen af flere scaffolds.
3. N50 er en statistik der repræsenterer hvor mange contigs der skal til for at repræsentere mindst halvdelen af alle baser i et assembly.
4. Et "draft assembly" er et assembly der repræsenterer fulde chromosomer.

Emne 16: Fil og data formater

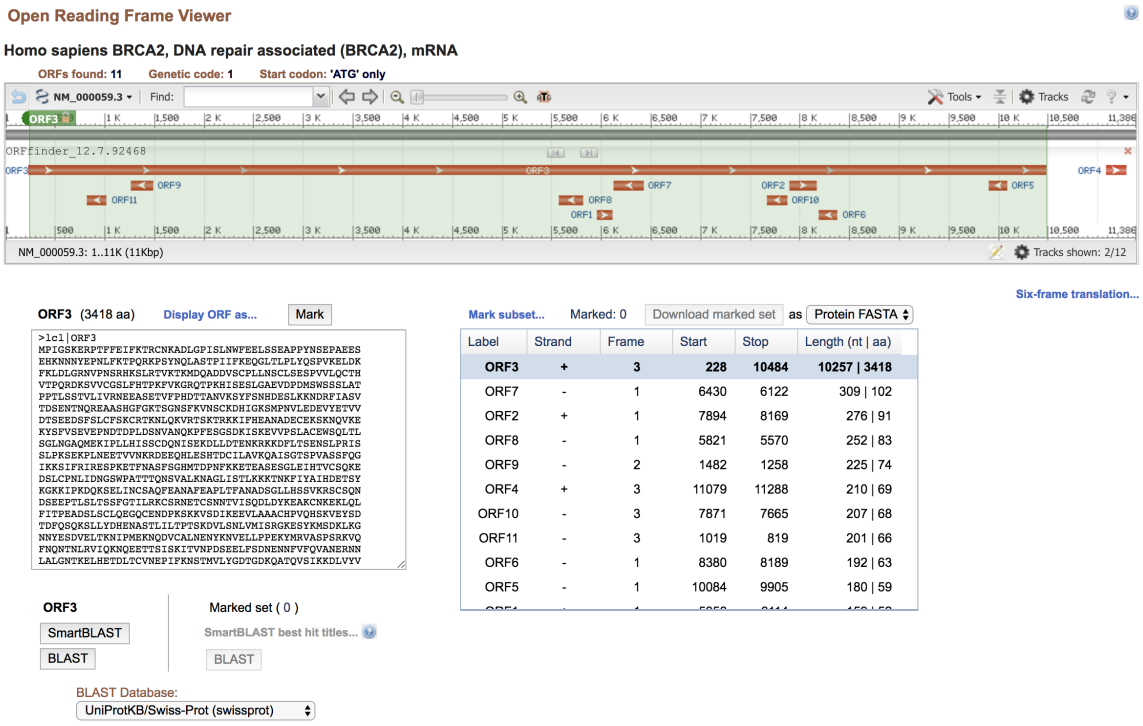
Bioinformatisk data findes i mange former og formater. Omformatering af et format til et andet er i nogen tilfælde ikke muligt. I tilfælde hvor det er muligt kan det indebære et tab af information fordi et format indeholder en type information som et andet format ikke indeholder.

Statements:

1. En Fasta fil kan formateres til en Fastq fil.
2. En Fastq fil kan formateres til en Fasta fil.
3. En Fasta fil kan formateres til en Genbank fil uden tab af information.
4. Data i Newick format repræsenterer RNA sekundær struktur.

Emne 17: Åbne læserammer

Figuren nedenfor er et screenshot fra NCBIs ORFfinder, hvor der søgt efter åbne læserammer (ORFs) i en mRNA fra BRCA2 genet.



Statements:

1. ORFfinder er er baseret på en skjult Markov model.
2. Der er identificeret 10 åbne læse rammer.
3. Der søges efter åbne læserammer i alle seks mulige læserammer.
4. Den tredie aminosyre i den længste læseramme er isoleucine (I).

Emne 18: Sammenligning af genforudsigelser

Nedenfor ser du et output EasyGene, hvor der søgt efter gener i et stykke DNA fra Bacillus subtilis.

```
##gff-version 2
##source-version easygene-1.2b
##date 2017-12-14
##Type DNA
# model:  BS03 Bacillus subtilis
# seqname   model feature start end   score       +/-  ?  startc  odds
# ------------------------------------------------------------------------
AB010576.1  BS03  CDS     67    324   0.0271875   +    0  #ATG    20.1861
AB010576.1  BS03  CDS     1129  1269  0.0190622   +    0  #ATG    15.7102
AB010576.1  BS03  CDS     1370  2314  2.13273e-12 +    0  #ATG    74.7815
AB010576.1  BS03  CDS     2327  2491  0.0167943   +    0  #ATG    17.2951
AB010576.1  BS03  CDS     300   668   1.43511     -    0  #ATG    10.6215
# ------------------------------------------------------------------------
```

Figuren nedenfor er et screenshot fra NEBcutters "orfsummary" hvor der er lavet en forudsigelse på den samme stykke DNA.

Gene	Product	aa seq.	Coordinates	Protein ID	Closest enzyme at 5' end	Closest enzyme at 3' end	Show flanking enzymes
-	-	314 aa	1370..2314	-	BfuAI	MmeI	show
-	-	122 aa	compl(300..668)	-	MboII	#AlwI	show
-	-	89 aa	55..324	-	*Fnu4HI	*AciI	show
-	-	61 aa	compl(782..967)	-	*AciI	AvaII	show
-	-	54 aa	2327..2491	-	MmeI	CviQI	show
-	-	46 aa	1129..1269	-	BccI	EcoP15I	show
-	-	42 aa	compl(1204..1332)	-	HinfI	*BcgI	show
-	-	41 aa	976..1101	-	AvaII	BccI	show
-	-	40 aa	compl(922..1044)	-	MluCI	#DpnII	show
-	-	34 aa	725..829	-	DpnI	SfcI	show
-	-	30 aa	6..98	-	AluI	MluCI	show
-	-	30 aa	1061..1153	-	CviKI-1	MboII	show
-	-	28 aa	667..753	-	#LpnPI	*AciI	show
-	-	27 aa	1333..1416	-	EcoP15I	HinfI	show

Statements:

1. NEBcutter er et værktøj der både finder åbne læserammer og kløvningssteder for restriktionsenzymer.
2. EasyGene er baseret på en skjult Markov model.
3. Kun tre af de åbne læserammer der identificeres med EasyGene identificeres også med NEBcutter.
4. Den åbne læseramme som EasyGene angiver har størst sandsynlighed er et rigtigt gen (og ikke bare er en ikke-kodende åben læse ramme) ligger på den negative streng.

Emne 19: Generelt om skjulte Markov modeller

Skjulte Markov modeller (hidden Markov models) er en klasse af matematiske modeller der ofte benyttes i bioinformatiske sammenhænge.

Statements:

1. En skjult Markov models emissionssandsynligheder angiver hvor sandsynligt det er at gå fra en skjult tilstand (state) til en anden.
2. Skjulte Markov modeller benyttes til at inddele en sekvens i delsekvenser, der repræsenterer forskellige kategorier.
3. Hvis man bruger en skjult Markov model til at lave en forudsigelse (prediction) på en sekvens, så vil antallet af forudsagte skjulte tilstande altid være det samme som antallet af positioner i sekvensen.
4. For at bruge en skjult Markov model til lave en forudsigelse (prediction) skal modellens parametre først estimeres.

Emne 20: Neurale netværk

Neurale netværk er en klasse af matematiske modeller der benyttes i bioinformatik.

Statements:

1. Neurale netværk er en særlig type skjult Markov model.
2. Neurale netværk kan anvendes til klassifikation af sekvenser.
3. Det er nødvendigt at estimere et neuralt netværks parametre før det kan bruges til forudsigelse.
4. PSIPRED er et program der bruger neurale netværk til forudsigelse af proteiners sekundær struktur.

Emne 21: Chou-Fasman

Der er mange forskellige måder at forudsige protein sekundær struktur. En af de mest simple metoder er Chou-Fasman algoritmen.

Statements:

1. Chou-Fasman algoritmen udnytter at phi of psi vinklerne i peptid kæden er anderledes i alpha-helices end i beta-strands.
2. Chou-Fasman algoritmen udnytter at forskellige aminosyrer optræder med forskellige hyppigheder i alpha-helices og beta-strands.
3. Chou-Fasman algoritmen udnytter at fordelingen af afstande mellem C-atomer i peptidkæden er anderledes i alpha-helices end i beta-strands.
4. Chou-Fasman algoritmen udnytter at forskellige aminosyrer optræder med forskellige hyppigheder i hairpin turns mellem beta-strands.

Emne 22: Protein foldning

Statements:

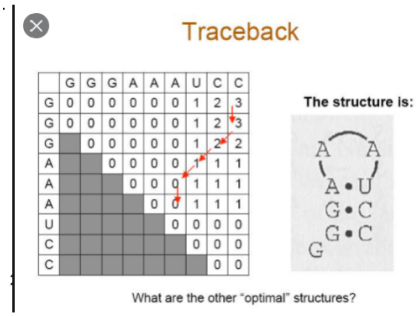
1. RMSD (root mean square deviation) er et mål for hvor ens to homologe proteiners 3D strutuktur er.
2. En PDB fil indeholder de relative positioner af atomer i et foldet protein.
3. Jmol er et program det tillader 3D visualisering af protein strukturer.
4. Formålet med homologimodellering er at beregne hvor ens to kendte proteinfoldninger er.

Emne 23: Nussinov algoritmen

Nussinov algoritmen finder den RNA sekundær struktur, der giver det største antal base parringer. Både Watson-Crick base par og GU basepar regnes som komplementære. Det antages enviddere at loops skal være på mindst to baser.

Nedenfor ser du en dynamisk programmerings table for sekvensen: CAUGACCUUAU

D		C_1	A_2	U_3	G_4	A_5	C_6	C_7	U_8	U_9	A_{10}	U_{11}
C_1	0	0	0	0	1	1	1	1	2	2	3	4
A_2		0	0	0	0	0	0	1	2	2	3	4
U_3			0	0	0	0	0	1	1	2	3	3
G_4				0	0	0	0	1	1	2	2	2
A_5					0	0	0	0	1	1	1	1
C_6						0	0	0	0	0	0	0
C_7							0	0	0	0	0	0
U_8								0	0	0	0	0
U_9									0	0	0	0
A_{10}										0	0	0
U_{11}											0	0



Statements:

1. Den optimale struktur har et loop på tre baser.
2. Den optimale struktur har en hair-pin med fire parrede baser.
3. I den optimale struktur indgår den første base (C) i en baseparring.
4. Der er mere end en optimal struktur.