Scientific Programmming 13/02/2019

Before you start

Please write one single python script for the lab part and one text file with the answers to the theoretical questions.
IMPORTANT: Add your name and ID (matricola) on top of the .py and text files!

Theory

Please write the solution in a text file.

Exercise 1:

Let L a list of size n, and i and j two indeces. Return the computational complexity of the function fun() with respect to n.

```
def fun(L, i, j):
     # j-i+1 is the number of elements
     # between index i and index j (both included)
     if j-i+1 <= 3:
         # Compute their minimum
         return min(L[i:j+1])
     else:
        onethird = (j-i+1)//3
        res1 = fun(L,i, i+onethird)
        res2 = fun(L,i+onethird+1, i+2*onethird)
        res3 = fun(L,i+2*onethird+1, j)
        return min(res1,res2,res3)
```

Practical part

Exercise 1 (Part A):

The file test_genes.tsv is a tab separated file containing the following information regarding genes:

```
ScaffoldID    feature    StartPos    EndPos    GeneID
s159888    gene    48555    49286    gene00001
s159888    mRNA    48555    49286    gene00001
s159888    start_codon    48555    48557    gene00001
s159888    CDS    48555    48653    gene00001
...
```

As the header says, the first column is the scaffold where the gene is located, feature represents the information specified in the line (e.g. gene, mRNA, start_codon,...) , StartPos and EndPos are the starting and ending position of the feature and GeneID is the identifier of the gene.
The file test_seqs.fasta is a fasta formatted file with the sequence of some of the scaffolds contained in the test_genes.tsv file.

Read the file in input and store it in a suitable data structure (hint: **pandas**?) then write the following methods:

1. `computestats(data)` gets the data structure loaded and prints the total number of scaffolds, geneIDs and features. The method should also print the first ten geneIDs. For each feature (i.e. second column) print the average length of the feature (with 3 decimal places of precision);

2. `scaffHistogram(data, scaffold, start, end, kind=”bar”)` given the input data, a scaffold, a start position and an end position, prints how many features are **entirely** contained in the closed interval [start,end] and plots the counts of the different number of features in the interval. An optional parameter `kind` can be specified to control the type of plot (possible values: bar, pie). The default of this optional parameter is “bar”;

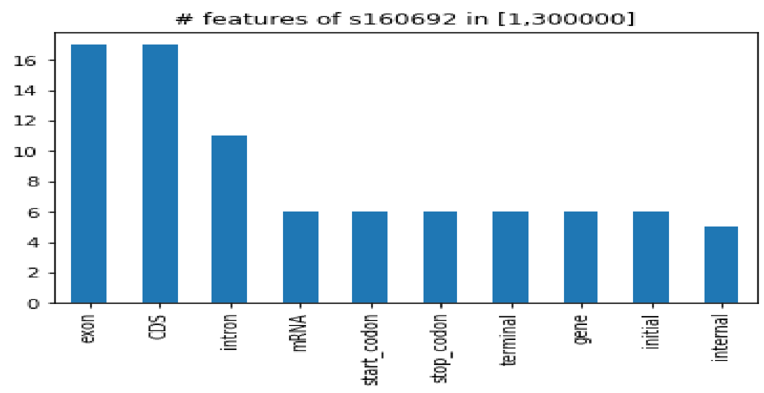
3. `getInfo(data, geneID, sequenceFile)` given the input data, a geneID and a fasta file containing sequences, the function prints all the features belonging to geneID (if geneID exists, an error message otherwise) and the sequence of the **start_codon**, all **exons** and **end_codon** (if the corresponding scaffold is in the sequenceFile, otherwise ‘UNAVAILABLE’). **Note that positions in the test_genes.tsv file are 1-based, which means that the first base is considered in position 1**). **Hint: use biophyton.** **<u>See the examples below to see how the output should look like.</u>**

After loading the data, calling

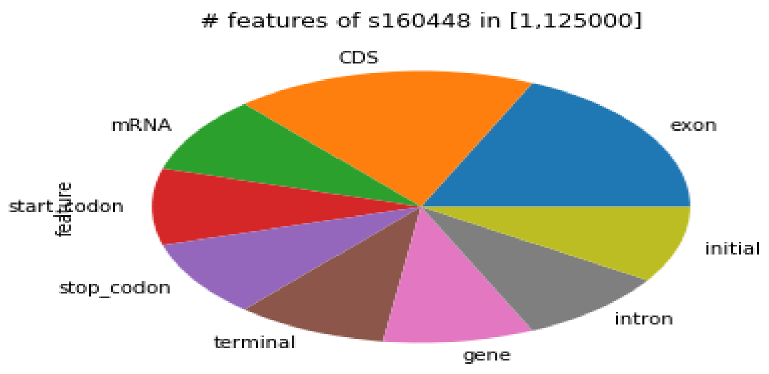
```
dataFile = "test_genes.tsv"
seqFile = "test_seqs.fasta"

data = pd.read_csv(dataFile,
                   sep="\t",  header = 0)

computestats(data)
print("\n")
scaffInfo(data, "s160692",1,300000)
scaffInfo(data, "s160448",1,125000, "pie")
scaffInfo(data, "s160551",1,50000)
scaffInfo(data, "s163228", 1,10000)
print("\n")
getInfo(data, "gene00001" ,seqFile)
print("\n")
getInfo(data, "gene00607" ,seqFile)
print("\n")
getInfo(data, "gene72" ,seqFile)
```

should return

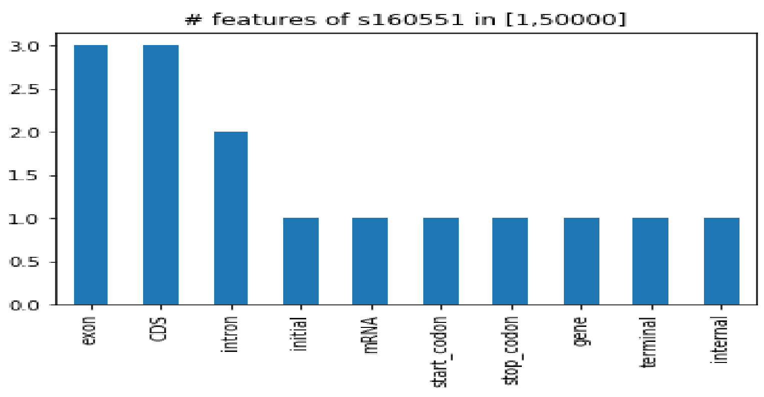
```
The file contains 9982 features
Belonging to 481 scaffolds and 622 genes.
First 10 genes are:
      gene00044
      gene00556
      gene00512
      gene00323
      gene00124
      gene00541
      gene00553
      gene00430
      gene00588
      gene00382
Overall average feature length: 1193.051 bps

Feature CDS has avg. length 338.342 bps
Feature exon has avg. length 338.342 bps
Feature gene has avg. length 5644.326 bps
Feature initial has avg. length 357.402 bps
Feature internal has avg. length 222.450 bps
Feature intron has avg. length 2007.303 bps
Feature mRNA has avg. length 5644.326 bps
Feature start_codon has avg. length 3.000 bps
Feature stop_codon has avg. length 3.000 bps
Feature terminal has avg. length 465.545 bps

86 total features are in s160692 in [1,300000]
```



```
11 total features are in s160448 in [1,125000]
```



```
15 total features are in s160551 in [1,50000]
```



```
0 total features are in s163228 in [1,10000]
Nothing to plot for s163228:[1,10000]

Features of gene "gene00001"
gene on s159888 in position [48555,49286]
mRNA on s159888 in position [48555,49286]
start_codon on s159888 in position [48555,48557]
    Sequence: ATG
CDS on s159888 in position [48555,48653]
exon on s159888 in position [48555,48653]
    Sequence:
ATGGAGAAGGGCGATACTATATCTACATACCTGAACAAGCTCACCACCTGTATAGATGAGCTTAGTAGTGTGGGAATAACTACT
GTTGATGATGATATG
initial on s159888 in position [48555,48653]
intron on s159888 in position [48654,48764]
CDS on s159888 in position [48765,49286]
exon on s159888 in position [48765,49286]
    Sequence:
GAGGATATCAGGAGGAGCACACGAGATGGTTCTTCATCCAAGAATGATGATGAAGAAAATCTGGCCTTAGCAAGTAAGGCAAGG
AAAGGGAAAGGTAAGGCTTCCCATTCCAAATCGAATTCTTCTCATGGAGGGAAGAAGGTTGACAAGTCTAAAGTGAGATGCTTT
AATTGTCACGACATGGGACATTATGCAACTAACTTCCCATTGAAGAAGTCCAAGAAGGGATCCTCGAAAGGATCACAAGGTGAG
GCATTAGCCTCTCAGTTTGAAATGGACTTTTCCCTCATCGCATGCATGGTTTCATCGATGGTGGGTTGTGTTTGGTACCTTGAC
AGCGGAGCCTCATTCCACATGACCGGCAATAAAAGTTTATTCAGTACCTTGGAGGAGAAAGACCTTAAGATGCACATAGAAATG
GGTGACAACAGAAAGTACAGTGTTTCAAGAGTGGGCACAGTTGCTTTTCAGAGGGAACACAGATCTCCTCATACCTTGATAGAT
GTGAAGTATGTGCCTTGA
terminal on s159888 in position [48765,49286]
stop_codon on s159888 in position [49284,49286]
    Sequence: TGA


Features of gene "gene00607"
gene on s163228 in position [77187,90573]
mRNA on s163228 in position [77187,90573]
start_codon on s163228 in position [77187,77189]
 - Sequence information UNAVAILABLE
CDS on s163228 in position [77187,77465]
exon on s163228 in position [77187,77465]
 - Sequence information UNAVAILABLE
initial on s163228 in position [77187,77465]
intron on s163228 in position [77466,89710]
CDS on s163228 in position [89711,89768]
exon on s163228 in position [89711,89768]
 - Sequence information UNAVAILABLE
internal on s163228 in position [89711,89768]
intron on s163228 in position [89769,90073]
CDS on s163228 in position [90074,90573]
exon on s163228 in position [90074,90573]
 - Sequence information UNAVAILABLE
terminal on s163228 in position [90074,90573]
stop_codon on s163228 in position [90571,90573]
 - Sequence information UNAVAILABLE

No information available for gene "gene72"
```

Exercise 2 (Part B):

Recall directed graphs as adjacency matrices (implemented in the included `DiGraphAsAdjacencyMatrix class`).

1. Write a method `getInConnected(self, mynode)` that returns the list of nodes that outlink to the node `mynode` if `mynode` exists in the set of nodes, it should print an error message and return None otherwise.

2. Write a method `getTopConnected(self, connectionType)` that returns **the node with the highest number of connections of the specified type (and the corresponding connections count)**. Connection type can be "**`incoming`**" to get the node with the highest number of incoming edges, "**`outgoing`**" to get the node with the highest number of outgoing edges or "**`both`**" to get the node that has the highest number of total connections (outgoing plus incoming).

3. Write a **recursive method** `countPathsRec(self, startNode, endNode, K)` that given a starting node `startNode` and a ending node endNode counts how many paths of length exactly equal to K connect the two (following the orientation of edges).

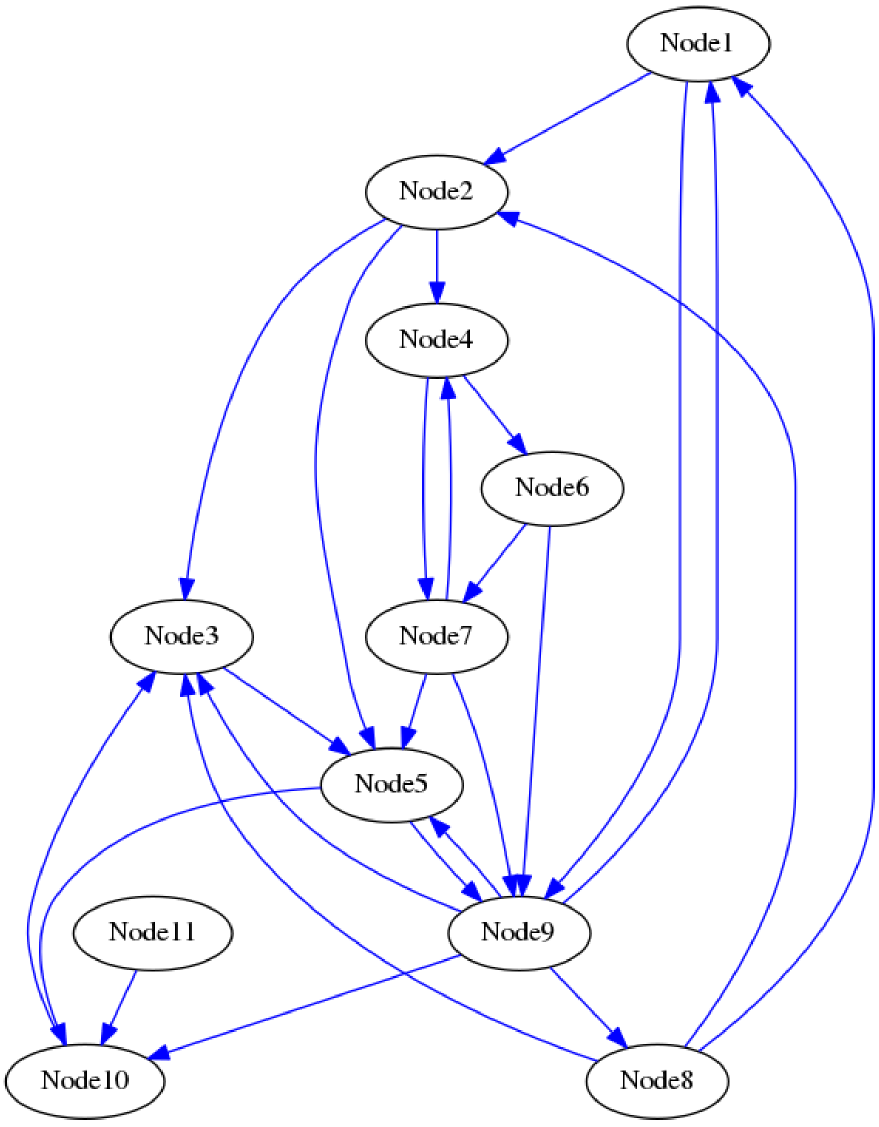
   To solve this problem consider that the base cases are:
   a. if startNode == endNode and K = 0 there is one path connecting the two;
   b. if startNode outlinks to endNode and K = 1, then there is one path;
   c. if K <0 there are no possible paths;
   The recursive method should move from start node to all its adjacent nodes and recursively count how many paths of length K-1 exist from that point on towards endNode (summing all of them).

4. Modify the previous method to return also all the found paths. Hint: add a parameter `path` to the countPathsRec method.

Test the code with the graph that can be implemented with:

```
G = DiGraphAsAdjacencyMatrix()
    for i in range(11):
        n = "Node{}".format(i+1)
        G.insertNode(n)

G.insertEdge("Node1","Node2", 1)
G.insertEdge("Node2", "Node3", 1)
G.insertEdge("Node2","Node4", 1)
G.insertEdge("Node2" ,"Node5", 1)
G.insertEdge("Node3" ,"Node5", 1)
G.insertEdge("Node9" ,"Node5", 1)
G.insertEdge("Node4","Node6", 1)
G.insertEdge("Node6","Node7", 1)
G.insertEdge("Node7","Node5", 1)
G.insertEdge("Node7" ,"Node4", 1)
G.insertEdge("Node4" ,"Node7", 1)
G.insertEdge("Node8" ,"Node3", 1)
G.insertEdge("Node8" ,"Node2", 1)
G.insertEdge("Node1" ,"Node9", 1)
G.insertEdge("Node9" ,"Node1", 1)
G.insertEdge("Node9" ,"Node10", 1)
G.insertEdge("Node5" ,"Node10", 1)
```

```
G.insertEdge("Node5" ,"Node9", 1)
G.insertEdge("Node6" ,"Node9", 1)
G.insertEdge("Node7" ,"Node9", 1)
G.insertEdge("Node9" ,"Node8", 1)
G.insertEdge("Node9" ,"Node3", 1)
G.insertEdge("Node8" ,"Node1", 1)
G.insertEdge("Node10" ,"Node3", 1)
G.insertEdge("Node11" ,"Node10", 1)
```



Calling

```
n="Node3"
print("{} has incoming links from: {}".format(n, G.getInConnected(n)))
n="Node7"
print("{} has incoming links from: {}".format(n, G.getInConnected(n)))
n="Node9"
print("{} has incoming links from: {}".format(n, G.getInConnected(n)))
n="Node472"
print("{} has incoming links from: {}".format(n, G.getInConnected(n)))
n="Node11"
print("{} has incoming links from: {}".format(n, G.getInConnected(n)))
print("\n")
```

```
TI = G.getTopConnected("incoming")
print("Top connected node(s) (incoming) is: {} ({} tot
connections)".format(TI[0],TI[1]))
TO = G.getTopConnected("outgoing")
print("Top connected node(s) (outgoing) is: {} ({} tot
connections)".format(TO[0],TO[1]))
TB = G.getTopConnected("both")
print("Top connected node(s) is: {} ({} tot connections)".format(TB[0],TB[1]))

print("\n")
n1 = "Node1"
n2 = "Node4"
n = 4
print("{} paths of length {} between node {} and
{}".format(G.countPathsRec(n1,n2,n),n, n1,n2))

n1 = "Node3"
n2 = "Node9"
n = 7
print("{} paths of length {} between node {} and
{}".format(G.countPathsRec(n1,n2,n), n, n1,n2))
```

should return:

```
Node3 has incoming links from: ['Node2', 'Node8', 'Node9', 'Node10']
Node7 has incoming links from: ['Node4', 'Node6']
Node9 has incoming links from: ['Node1', 'Node5', 'Node6', 'Node7']
Error. Node "Node472" does not exist
Node472 has incoming links from: None
Node11 has incoming links from: []


Top connected node(s) (incoming) is: Node3,Node5,Node9 (4 tot connections)
Top connected node(s) (outgoing) is: Node9 (5 tot connections)
Top connected node(s) is: Node9 (9 tot connections)


3 paths of length 4 between node Node1 and Node4
18 paths of length 7 between node Node3 and Node9
12 paths of length 5 between node Node4 and Node10
```

with the paths (exercise 4):

```
PATH: Node1 --> Node2 --> Node4 --> Node7 --> Node4
PATH: Node1 --> Node9 --> Node1 --> Node2 --> Node4
PATH: Node1 --> Node9 --> Node8 --> Node2 --> Node4
3 paths of length 4 between node Node1 and Node4
PATH: Node3 --> Node5 --> Node9 --> Node1 --> Node2 --> Node3 --> Node5 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node1 --> Node2 --> Node4 --> Node6 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node1 --> Node2 --> Node4 --> Node7 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node1 --> Node9 --> Node3 --> Node5 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node1 --> Node9 --> Node8 --> Node1 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node3 --> Node5 --> Node9 --> Node1 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node3 --> Node5 --> Node9 --> Node5 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node5 --> Node9 --> Node3 --> Node5 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node5 --> Node9 --> Node8 --> Node1 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node5 --> Node10 --> Node3 --> Node5 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node8 --> Node1 --> Node2 --> Node5 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node8 --> Node1 --> Node9 --> Node1 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node8 --> Node1 --> Node9 --> Node5 --> Node9
```

```
PATH: Node3 --> Node5 --> Node9 --> Node8 --> Node2 --> Node3 --> Node5 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node8 --> Node2 --> Node4 --> Node6 --> Node9
PATH: Node3 --> Node5 --> Node9 --> Node8 --> Node2 --> Node4 --> Node7 --> Node9
PATH: Node3 --> Node5 --> Node10 --> Node3 --> Node5 --> Node9 --> Node1 --> Node9
PATH: Node3 --> Node5 --> Node10 --> Node3 --> Node5 --> Node9 --> Node5 --> Node9
18 paths of length 7 between node Node3 and Node9
PATH: Node4 --> Node6 --> Node7 --> Node5 --> Node9 --> Node10
PATH: Node4 --> Node6 --> Node7 --> Node9 --> Node5 --> Node10
PATH: Node4 --> Node6 --> Node9 --> Node1 --> Node9 --> Node10
PATH: Node4 --> Node6 --> Node9 --> Node3 --> Node5 --> Node10
PATH: Node4 --> Node6 --> Node9 --> Node5 --> Node9 --> Node10
PATH: Node4 --> Node7 --> Node4 --> Node6 --> Node9 --> Node10
PATH: Node4 --> Node7 --> Node4 --> Node7 --> Node5 --> Node10
PATH: Node4 --> Node7 --> Node4 --> Node7 --> Node9 --> Node10
PATH: Node4 --> Node7 --> Node5 --> Node9 --> Node5 --> Node10
PATH: Node4 --> Node7 --> Node9 --> Node1 --> Node9 --> Node10
PATH: Node4 --> Node7 --> Node9 --> Node3 --> Node5 --> Node10
PATH: Node4 --> Node7 --> Node9 --> Node5 --> Node9 --> Node10
12 paths of length 5 between node Node4 and Node10
```