Scientific Programming 02/07/2019

Before you start

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 L1 and L2 be two lists containing n lists, each of them of size n. Compute the computational complexity of function fun() with respect to n.

```
def fun(L1,L2):
     for r1 in L1:
         for val in r1:
             for r2 in L2:
                 if val = sum(r2):
                     print(val)
```

Practical part

Exercise 1:

The file `ctg_data.tsv` is a tab separated file containing the following information regarding assembled sequences:

```
CtgID   LG      Start   End   Size
seqctg_1        LG7     1       7950836 7950836
seqctg_2        LG15    1       2550565 2550565
seqctg_3        LG12    1       4134320 4134320
seqctg_4        LG7     7958328 13580696        5622369
...
```

As the header says, the first column is the contig identifier, the second is the linkage group (i.e. chromosome) the sequence has been assembled into, the third is the size of the sequence, fourth and fifth are the start and end point of the sequence within the LG. **Note that there can be gaps between contigs of each LG (unknown bases between the end of a contig and the beginning of the next one)**.

Write the following methods

1.loadTsv(filename): gets the filename of the tsv file and returns a pandas dataframe with all the loaded information;

2.computestats(data) gets the data structure loaded and prints the total number of sequences, the total number of LGs, and for each LG the total number of contigs, total size and total sum of gaps (i.e. total size minus the sum of all the contig sizes).

Finally, the total size of the genome should be printed (i.e. the sum of all the total sizes);

3. computeN50(data, lg) given the input data, and an LG (**if unspecified all data should be considered** i.e. specify a default value of lg to None and check for it) returns the N50. If the LG does not exist a warning message should be returned. The N50 is computed as in this picture:

1a. Contigs, sorted according to their lengths.

1b. Calculation of N50 using sorted contigs.

Calling

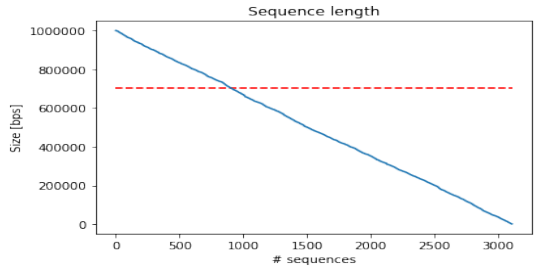
```
dataFile = "ctg_data.tsv"
data = loadTSV(dataFile)

computestats(data)

n50 = computeN50(data)
print("\nN50 of all data: {}".format(n50))

n50 = computeN50(data, "LG1")
print("N50 of LG1: {}".format(n50))

plotN50(data)

plotN50(data, "LG2")

plotN50(data, "LG7")

plotN50(data, "LG9")

computeN50(data, "LG16")
```

should return

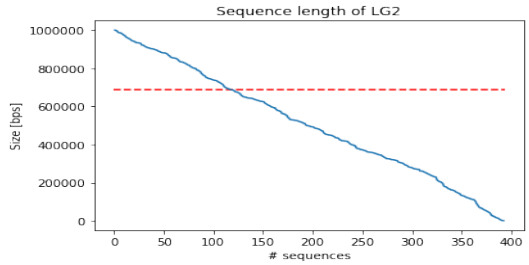
```
Total number of sequences: 3106
Total number of LGs: 11

LG0 has 814 seq. (TOT size 409,573,145 bps gaps: 4,144,890 bps)
LG1 has 561 seq. (TOT size 278,217,470 bps gaps: 2,998,726 bps)
LG10 has 31 seq. (TOT size 17,167,851 bps gaps: 160,390 bps)
LG2 has 392 seq. (TOT size 201,407,626 bps gaps: 1,959,628 bps)
LG3 has 356 seq. (TOT size 173,744,420 bps gaps: 1,712,060 bps)
LG4 has 286 seq. (TOT size 142,642,805 bps gaps: 1,469,537 bps)
LG5 has 223 seq. (TOT size 111,530,236 bps gaps: 1,075,199 bps)
LG6 has 166 seq. (TOT size 84,417,104 bps gaps: 776,918 bps)
LG7 has 130 seq. (TOT size 70,044,770 bps gaps: 13,813,134 bps)
LG8 has 88 seq. (TOT size 49,315,759 bps gaps: 476,756 bps)
LG9 has 59 seq. (TOT size 28,274,943 bps gaps: 293,993 bps)

The total size of the genome is 1,566,336,129 bps

N50 of all data: 702139
```

N50 of LG1: 714265

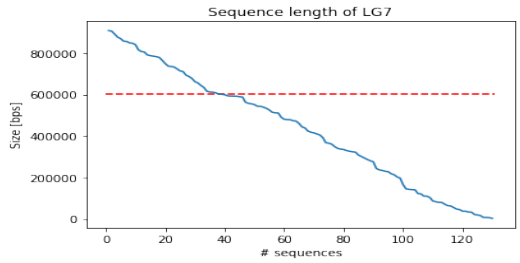
N50: 702139



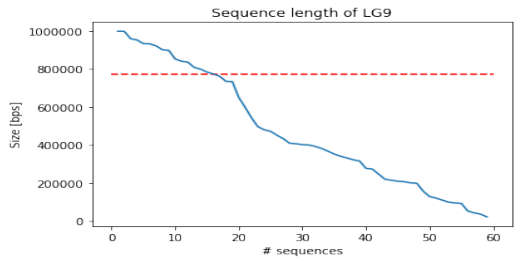
N50: 687296



N50: 602267



N50: 773762



Warning cannot find information on LG16

Exercise 2:

1. Given the following recurrence equation:

$$F(n) = \begin{cases} 0 & if\, n < 0 \\ n & if\, n < 3 \\ 2 * [F(n-1) + F(n-2)] - F(n-3) & if\, n >= 3 \end{cases}$$

where n is an integer, write a time and possibly space efficient python function that computes F(n).

Test it with the following code:

```
for i in range(0,12):
    print("F({}) = {:.3f}".format(i,F(i)))

print("F(1250) = {}".format(F(1250)))
```

which should return:

```
F(0) = 0
F(1) = 1
F(2) = 2
F(3) = 6
F(4) = 15
F(5) = 40
F(6) = 104
F(7) = 273
F(8) = 714
F(9) = 1870
F(10) = 4895
F(11) = 12816
F(1250) =
95305565117374330360175383388422251346300689327102823182767242585807390008733354293089871205329980609242561585302413249290548494851100500587299498247237180976514353511854213984817155603442807369719289043837539441468269184903281345761080469416662470127427234727110808503048824766869695786387279311281977880130238601779898552134236954469815644703463821278714256899382488224320497936697837784700198102914321928640500810713620525040719000434534564045722943146378220046907665074728019917130535012838089810503113523316457882125 0
```

2. Recall queues (i.e. first-in first-out data structures that allow access only to the first inserted element). Extend the (slow) `myStack` class seen in the lecture and provided in the `MyQueue.py` file creating a new class `ReversibleQueue`(edit the provided ReversibleQueue.py file) which implements the following methods:

1. `printContent(self)`that prints the minimum and maximum value of the queue USING ONLY THE METHODS ENQUEUE, DEQUEUE, TOP AND ISEMPTY (**note: the queue must not be changed at the end of this method**). **Hint: remove elements and add them back in**. See below for the expected output.
2. `reverseK(self, k)` that reverses the first k elements inserted in the queue (modifying the queue). If the queue Q contains the following elements:

```
Q : 1 2 3 4 5 6 7 8 9 10
```

reversing the first 4 elements of `Q` with `Q.reverseK(4)` should change `Q` in the following way: `Q : 4 3 2 1 5 6 7 8 9 10`

while `Q.reverseK(0)`, `Q.reverseK(-10)` etc. should not reverse anything and `Q.reverseK(1000)` for any `K >= len(Q)` should completely reverse `Q`.

Calling

```
    Q = ReversibleQueue()
    data = [ 72, 11, 2, 13, 87, 27, 44, 3, 1, -10, 4]
    for el in data:
        Q.enqueue(el)
    print("First in queue: {}".format(Q.top()))
    Q.printContent()

    print("Empty? {}".format(Q.isEmpty()))
    print("First in queue: {}".format(Q.top()))


    Q2 = ReversibleQueue()
    dataStr = ["APPLE", "BLUEBERRY", "PINEAPPLE"]
    for el in dataStr:
        Q2.enqueue(el)

    Q.reverseK(0)
    print("\nQ now (rev(0) unchanged):")
    Q.printContent()
    Q.reverseK(-5)
    print("\nQ now (rev(-5) unchanged):")
    Q.printContent()
    Q.reverseK(5)
    print("\nQ now (first 5 rev):")
    Q.printContent()

    print("\nQ2:")
    Q2.printContent()
    Q2.reverseK(0)
    print("\nQ2 now (rev(0) unchanged):")
    Q2.printContent()
    print("\nQ2 now (rev(rev(2))unchanged):")
    Q2.reverseK(2)
    Q2.reverseK(2)
    Q2.printContent()
    Q2.reverseK(10)
    print("\nQ2 now (rev(10):")
    Q2.printContent()
```

should return:

```
First in queue: 72
[0]: 72
[1]: 11
[2]: 2
[3]: 13
[4]: 87
[5]: 27
[6]: 44
[7]: 3
[8]: 1
[9]: -10
[10]: 4
Min: -10 Max:87
Empty? False
First in queue: 72

Q now (rev(0) unchanged):
[0]: 72
[1]: 11
[2]: 2
[3]: 13
[4]: 87
[5]: 27
```

```
[6]: 44
[7]: 3
[8]: 1
[9]: -10
[10]: 4
Min: -10 Max:87

Q now (rev(-5) unchanged):
[0]: 72
[1]: 11
[2]: 2
[3]: 13
[4]: 87
[5]: 27
[6]: 44
[7]: 3
[8]: 1
[9]: -10
[10]: 4
Min: -10 Max:87

Q now (first 5 rev):
[0]: 87
[1]: 13
[2]: 2
[3]: 11
[4]: 72
[5]: 27
[6]: 44
[7]: 3
[8]: 1
[9]: -10
[10]: 4
Min: -10 Max:87

Q2:
[0]: APPLE
[1]: BLUEBERRY
[2]: PINEAPPLE
Min: APPLE Max:PINEAPPLE

Q2 now (rev(0) unchanged):
[0]: APPLE
[1]: BLUEBERRY
[2]: PINEAPPLE
Min: APPLE Max:PINEAPPLE

Q2 now (rev(rev(2))unchanged):
[0]: APPLE
[1]: BLUEBERRY
[2]: PINEAPPLE
Min: APPLE Max:PINEAPPLE

Q2 now (rev(10):
[0]: PINEAPPLE
[1]: BLUEBERRY
[2]: APPLE
Min: APPLE Max:PINEAPPLE
```