Algorithm in Computational Biology :Assignment 1

Name : Preethi Raksha Thorali Gobinath
ID: 1229592977

Question 2

Since we need to check whether sequence X and Y come from the same circular sequence, we will need to create a sequence which has all possible single rotations, this can be done by adding the sequence to itself one more time.

Let's take an example,

- X= “GTACA” and Y= “CAGTA”,
  We will consider Pattern = X = “GTACA”

- We will create a text string with all possible single rotations, Text= Y+Y = CAGTA +CAGTA = “**CAGTACAGTA**”. This string contains all possible rotations of CAGTA in the form of substring, which are CAGTA, AGTAC, **GTACA**, TACAG, ACAGT, CAGTA. Of all the rotations, our pattern X matches the substring 3.

- Given Pattern = “GTACA” and Text = “CAGTACAGTA” , we can visibly notice that our text contains the pattern at index starting from 2 till index 7.

- Now for the z algorithm the input pattern is “GTACA” and the input text is “CAGTACAGTA”, let's add the “$” symbol between them to process the algorithm “GTACA**$**CAGTACAGTA”.

- Now if we run the algorithm to calculate the z values we get z[]= [x,0,0,0,0,0,0,0,5,0,0,0,0,3].

- After which we check in the z array, is there a value with length of the input pattern, which is equal to 5 in our example. If the length of the input pattern is present in the z array, then we output “both the sequences are from the same circular sequence” else we output they are different sequences.

Question 3

Given:

P = ATCATCT and S = TCATCATGATGATCATCT

Let :

x be the pattern Index iterator
y be the Text index iterator

i = 1 2 3 4 5 6 7
P[i] = A T C A T C T
lps[i] = 0 0 0 1 2 3 0
lps'[i]= 0 0 0 0 0 3 0
X = 0 1 2 3 4 5 6

S = T C A T C A T G A T G A T C A T C T
Y = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

KMP Algo:

```
M = len(P)
N = len(S)

y = 0  # index for txt
x = 0  # index for pat

while (N -y) >= (M - x):
        if pat[x] == txt[y]:
                x += 1
                y += 1
        if x == M:
                result.append(y - x + 1)
                x = lps[x - 1]
        elif y < N and pat[x] != txt[y]:
                if x != 0:
                        x = lps[x - 1]
                else:
                        y += 1
```

Manually running lps:

Index x = 0 1 2 3 4 5 6
lps[x] = 0 0 0 1 2 3 0
P[x] = A T C A T C T

Steps	Pattern	Text	Inference
1	x=0 A	y=0 T	No equal
2	x=0 A	y=1 C	No equal
3	x=0 A	y=2 A	Equal
4	x=1 T	y=3 T	Equal
5	x=2 C	y=4 C	Equal
6	X=3 A	y=5 A	Equal
7	x=4 T	y=6 T	Equal
8	x=5 C	y=7 G	Not Equal
9	x=lps[5-1]=lps[4]=2 C	y=7 G	Not Equal
10	x=lps[2-1]=lps[1]=0 A	y=7 G	Not Equal
11	x=0 A	y=8 A	Equal
12	x=1 T	y=9 T	Equal
13	x=2 C	y=10 G	Not Equal
14	x=lps[2-1]=lps[1]=0 A	y=10 G	Not Equal
15	x=0 A	y=11 A	Equal
16	x=1 T	y=12 T	Equal
17	x=2 C	y=13 C	Equal
18	X=3 A	y=14 A	Equal
19	x=4 T	y=15 T	Equal
20	x=5 C	y=16 C	Equal
21	x=6 T	y=17 T	Equal
Stop	x=7=len(P)		Print

Manually running lps’:

Index x = 0 1 2 3 4 5 6
lps'[x]   = 0 0 0 0 0 3 0
P[x]   = A T C A T C T

Steps	**Pattern**	**Text**	**Inference**
1	x=0 A	y=0 T	No equal
2	x=0 A	y=1 C	No equal
3	x=0 A	y=2 A	Equal
4	x=1 T	y=3 T	Equal
5	x=2 C	y=4 C	Equal
6	X=3 A	y=5 A	Equal
7	x=4 T	y=6 T	Equal
8	x=5 C	y=7 G	Not Equal
9	x=lps[5-1]=lps[4]=0 A	y=7 G	Not Equal
10	x=0 A	y=8 A	Equal
11	x=1 T	y=9 T	Equal
12	x=2 C	y=10 G	Not Equal
13	x=lps[2-1]=lps[1]=0 A	y=10 G	Not Equal
14	x=0 A	y=11 A	Equal
15	x=1 T	y=12 T	Equal
16	x=2 C	y=13 C	Equal
17	X=3 A	y=14 A	Equal
18	x=4 T	y=15 T	Equal
19	x=5 C	y=16 C	Equal
20	x=6 T	y=17 T	Equal
Stop	x=7=len(P)		Print

From the above calculation I can see that **lps takes 21 steps** which is greater than **lps' which takes only 20 steps**. For the above example I can confirm that Dr.wiz solution is valid for this example, but however one example cannot be used to assure the validity of the algorithm in all cases. A valid proof would be required to prove that it works for all cases or an example which breaks the algorithm would be required to disprove it.

Question 4

<u>Let us take a example 1 Input = **CAGTACAGTA**</u>

i = 1 2 3 4 5 6 7 8 9 10
input [i] = C A G T A C A G T A

- Lets calculate the z values: which store the length of the longest substring starting from input[i] if it is the prefix of input[1..n-1].

  z[]= [x,0,0,0,0,5,0,0,0,0]

- Lets calculate the lps values: hich stores length of the longest non trivial suffix of input[1..i] that matches a prefix of input.

  lps[i]= [0,0,0,0,0,1,2,3,4,5]

- Lets calculate the lps' values: which stores length of the longest non trivial suffix of input[1..i] that matches a prefix of input such that input[lps[i] + 1] ! = input[i + 1].

  lps[i]= [0,0,0,0,0,0,0,0,0,5]

<u>Let us take a example 2 Input = **CAGTCCAGTC**</u>

i = 1 2 3 4 5 6 7 8 9 10
input [i] = C A G T C C A G T C

- Lets calculate the z values: which store the length of the longest substring starting from input[i] if it is the prefix of input[1..n-1].

  z[]= [x,0,0,0,1,5,0,0,0,1]

- Lets calculate the lps values: Which stores length of the longest non trivial suffix of input[1..i] that matches a prefix of input.

  lps[i]= [0,0,0,0,1,1,2,3,4,5]

- Lets calculate the lps' values: which stores length of the longest non trivial suffix of input[1..i] that matches a prefix of input such that input[lps[i] + 1] ! = input[i + 1].

lps[i]= [0,0,0,0,1,0,0,0,0,5]

From both the above examples, we can notice that the z values store the length of the longest substring at the **start index** of the match, whereas lps' stores at the **last index** of the match.

So we can calculate the lps's values for each index of z values by assigning:

lps' [current z index + current z value -1] => current z value, given lps' is in initial state (0)

Code:

```
Def calculateLpsDash (zvalues):

        # Initialize lps' with 0 for length of zValues array
        lpsDash=[0] * len(zValues)

        # Iterate over the zValues array from index 1, as first value is x
        For i in range(1,len(zvalues)):

                # If zValue contains value & lps' is in the initial state (0) than assign the value to lps'
                 If zValues[i]>0 and lpsDash[i + zValues[i] -1] == 0 :
                         lpsDash[i + zValues[i] -1]= zValues[i]

Main function:

        # Input
         S= "CAGTACAGTA"
         zValues=z[]= [x,0,0,0,0,5,0,0,0,0]

        # Call the function
         result = calculateLpsDash ( zValues)

        # Display the function
         Print (result)
```