Assignment 2

Bioinformatics (CIS 455)

Javier Arechalde

February 26, 2018

Problem 2-1 Jones & Pevzner, Problem 4.1

In this problem, we will be given a set X, and we will need to calculate the multiset ΔX,

Pseudocode

In our implementation, we will iterate over the list with two index, the first one to indicate the position we are calculating the distances from, and the second one to indicate the position we want to calculate the distance with. We suppose that the given list comes presorted, so we won't.

Algorithm 1 title

```
for $i$ in range $1 \to length(X)$ do
    for $j$ in range $j \to length(X)$ do
        $distance = X[j] - X[i]$
        We add the calculated distance to the $\Delta X$ array
    end for
end for
```

Implementation

Here is the implementation of the pseudocode listed above.

```
#First we declare the X
X =[1,2,4,6,9,14]

#We will store DX in this list
dx =[]

#Now we will iterate over the list to calculate all the distances
for i in range(0,len(X)):
 for j in range(i+1,len(X)):
  deltax = X[j]-X[i]
  dx.append(deltax)

#Print the results
print('X: ')
print(X)
print('')
print('Delta X: ')
print(dx)
```

After running the code over a sample dataset, we obtained the following results.

```
X:
[1, 2, 4, 6, 9, 14]

Delta X:
[1, 3, 5, 8, 13, 2, 4, 7, 12, 2, 5, 10, 3, 8, 5]
```

Problem 2-2 Jones & Pevzner, Problem 4.2

Following $ANOTHERBRUTEFORCEPDP(L, n)$

Here is the implementation code for $ANOTHERBRUTEFORCEPDP$, note that for finding all the possible sets of length $n-2$, we decided to use *itertools*, to make the implementation easier.

```
#Importing numpy module
import numpy as np
from itertools import permutations

#We are given the following partial digest
L = [1,1,1,2,2,3,3,3,4,4,5,5,6,6,6,9,9,10,11,12,15]

def anotherbruteforcepdp(L):
 M = max(L)
  Ll = len(L)

 #Calculating n
 #n^2-n-2*L = 0
 n = (1+np.sqrt(1+8*Ll))/2

  if (int(n) != n):
   print('Error')
   return

 n = int(n)

  print('Maximum: %i Length: %i' %(M,n))

 #First we need to get the unique integers
 UL = []
  for number in L:
```

```
  if UL == []:
   UL.append(number)
  else:
   if number != UL[len(UL)-1] and number<M and number>0:
        UL.append(number)
   else:
        continue

#List of good sets
XList = []

#Valid X
Xret = []

#Now we will build all the different possible sets,
#using the itertools package
for p in permutations(UL, 5):
 valid = 1

 #We check if the set is valid
 for i in range(0,len(p)-1):
        if p[i]>=p[i+1]:
         valid = 0
 #If the set is valid, we create X
 if valid == 1:
        p = list(p)
        p.insert(0,0)
        p.append(M)
        XList.append(p) #Adding set to the list of sets

#For each set, we check if deltaX equals L
for X in XList:

 dx =[]

 for i in range(0,len(X)):
  for j in range(i+1,len(X)):
   deltax = X[j]-X[i]
   dx.append(deltax)

 #Sorting the Delta X list
 dx.sort()

 if dx == L:
        Xret.append(X)
```

```
    #If we finish iterating the for loop without finding any solution we
    #return that there is no solution

    if Xret == []:
     print('No solution')
     return

    else:
     return Xret

#We call the function and store the results
X = anotherbruteforcepdp(L)

print('')
print('X Found:')
print(X)
```

After running the code over the given partial digest, we obtained the following results.

```
Maximum: 15 Length: 7

X Found:
[[0, 3, 4, 5, 6, 9, 15], [0, 6, 9, 10, 11, 12, 15]]
```

Problem 2-3 Jones & Pevzner, Problem 4.5

Two sets A and B are said to be homometric if $\Delta A = \Delta B$.

We will try to prove that

$$U \oplus V = \{u + v : u \in U, v \in V\}$$

and

$$U \ominus V = \{u - v : u \in U, v \in V\}$$

are homometric.

In this problem we have two sets:

$$U = \{u_1, u_2, ..., u_n\}$$
$$V = \{v_1, v_2, ..., v_n\}$$

Then,

$$A = U \oplus V = \{u_1 + v_1, u_1 + v_2, ..., u_1 + v_n, ..., u_n + v_1, u_n + v_2, ...u_n + v_n\}$$

$$B = U \ominus V = \{u_1 - v_1, u_1 - v_2, ...u_1 - v_n, ..., u_n - v_1, u_n - v_2, ...u_n - v_n\}$$

Now we calculate ΔA and ΔB:

$$\Delta A = \{u_1 + v_1 - (u_1 + v_2), ...\} = \{v_1 - v_2, ...\}$$
$$\Delta B = \{u_1 - v_1 - (u_1 - v_2), ...\} = \{v_2 - v_1, ...\}$$

As we can see, the elements from U disappear when we are calculating ΔA and ΔB, and then, we are only calculating the distances within the set U. And as the distance is an absolute value of the difference of the two points, $v_1 - v_2 = v_2 - v_1$, and this is true for each one of the elements int ΔA and ΔB.

Then, we proven that $\Delta A = \Delta B$ and thus, sets $U \oplus V$ and $U \ominus V$ are *homometric.*

Problem 2-4 Jones & Pevzner, Problem 4.9

Pseudocode

In this problem we are given three sets of fragment lengths A, B and $A+B$. The goal is to reconstruct the original restriction map for each one of the enzymes, A and B.

For our brute force implementation, we will use all the possible orderings of the different fragments from A, B, and $A + B$ and see if they "match".

Algorithm 2 DDP Brute Force

```
1: for Every possible ordering of $A + B$ do
2:     for Every possible ordering of $A$ do
3:         for Every possible ordering of $B$ do
4:             for fragment $dX(i)$ in $dX$ do
5:                 if $dX(i) == A[j]$ then
6:                     Append to the A position $A[j] = sum(dX[0 : i])$
7:                     j++
8:                 else if $dxi == B[k]$ then
9:                     Append to the A position $B[k] = sum(dX[0 : i])$
10:                    k++
11:                elseIt doesnt fit any of the fragments
12:                    Continue to the next iteration
13:                end if
14:            end for
15:        end for
16:    end for
17: end for
```

Branch and Bound

A *Branch and Bound* implementation for this algorithm could list all the possible combinations of $A + B$ as a tree, having as the root nodes only the nodes that are equal or smaller to the first fragments of A and B. While we explored the tree, we could check if the sequence that we are building matches the other fragment lengths in A and B, and if it doesnt, we could skip that branch, and go to the next one.

Problem 2-5 Jones & Pevzner, Problem 4.12

In this problem, we have a long text string T, and a shorter pattern string s, we are trying to find the first occurence of s in T, if there is any.

We suppose that the length of T is m, and the length of s is n.

Pseudocode

Algorithm 3 Pseudocode for finding pattern

```
function FindPattern(T,s,m,n)
    while $i < (m - n)$ do
        if $s[i] == T[i]$ then
            for $j$ in range $i \to (i + n)$ do
                if $s[j] == T[j]$ then
                    if $j! = (i + n)$ then
                        $j + +$
                    end if
                    if $j == (i + n)$ then
                        Return $s[i : (i + n)]$
                    end if
                end if
                if $s[j]! = T[j]$ then
                    $i = j$
                    Exit for loop
                end if
            end for
        end if
        if $s[i]! = T[i]$ then
            $i + +$
        end if
    end while
end function
```

Time Complexity

In our implementation, we will iterate over the long string with our pattern. If we find that the first letter of our pattern is equal to the position we are checking in the longer text string, we will continue to check the following positions of our pattern to see if they match the ones on the long string.

Then in worst case scenario, we will have to check if the pattern is in the long string, on every position possible of the long string, without success. Then the time complexity of our algorithm will be $O((m - n) * n)$.

Problem 2-6 Jones & Pevzner, Problem 4.16

The $YETANOTHERMOTIFSEARCH$ algorithm is similar to $BRANCHANDBOUNDMOTIFSEARC$
as the algorithm, tries to find the best score by trying only the motifs that may give us a better score than the current best score.

The main difference between the two algorithms, is that for building all the possible motifs, $YETANOTHERMOTIFSEARCH$ uses recursive calls to the $FIND$ function, while $BRANCHANDBOUNDMOTIFSEARCH$, models the problem as a tree, and uses branch and bound for the same purpose.

Problem 2-7 Rosalind

This problem was solved entirely on **Rosalind**.

Username: *jarechalde*