91.350.202/580.202 HW #2 Vishal Gadiraju 01672049

REPORT

Initialization:
We assign the two input sequences to variables ‘s’ and ‘t’ and calculate their lengths which we store in ‘n’ and ‘m’

Scoring Matrix:
We create a scoring matrix and initialize values as given, we then assign a value to gap penalty.

	T	C	A	G
T	20	10	5	5
C	10	20	5	5
A	5	5	20	10
G	5	5	10	20

Implementation of Dynamic Programming Approach:
We have a recurrence relation based on which we calculate the values in the matrix ‘D‘ by visiting each element.

D[i][j] = max { D[i-1][j-1]+ sim_ma[s[j],t[i]], D[i-1][j] + gap_score, D[i][j-1] + gap_score};

Score can be achieved in 3 possible ways and the maximum value is selected,
We can obtain the value either from the cell horizontal,vertical or diagonal to the cell we desire to calculate.

Trace Back for Creating Sequence Alignment:
We start tracing back from the position in the matrix D where we have the maximum value i.e D[m+1][n+1]. The path that lead to the maximum score has to be constructed. We trace back until we find the first zero that is the zero at D[1][1] by taking the diagonal path if exists, as our priority.

The maximum entry of matrix D is the score of the alignment.

Test Sequence Results/Output:

a) A T G T T A T

A T C G T A G

When Gap penalty = 1 and 2

the output was,

A T C G _ T A G
A T _ G T T A T

b) ATGAATGCGATTTCGGGTGGCC

TTGGCAGGACATGAAGTTCGATACGGAA

When Gap penalty = 2, 5 and 8
the output was,

T T G G C A G G A C A T G A A G T T C G A T A C G G A A
A T G A _ A T G _ C _ _ G _ A T T T C G G _ G T G G C C

The code is implemented in Python and provided below:

Source Code:

```
 #!/usr/bin/python
def bToInt(s):
        if(s == 'A'):
        return 0
        elif s == 'C':
        return 1
        elif s == 'G':
        return 2
        elif s == 'T':
        return 3
#initializing two strings
s = 'ATGTTAT'
t = 'ATCGTAG'
n = len(s)
m = len(t)
print n
print m
D = [[0 for x in range(n+1)] for y in range(m+1)]     # Scoring
matrix initialization
sim_ma = [[0 for x in range(4)] for y in range(4)]
sim_ma[0][0] =20
sim_ma[0][1] =5
sim_ma[0][2] =10
sim_ma[0][3] =5
sim_ma[1][0] =5
sim_ma[1][1] =20
sim_ma[1][2] =5
sim_ma[1][3] =10
sim_ma[2][0] =10
sim_ma[2][1] =5
sim_ma[2][2] =20
sim_ma[2][3] =5
sim_ma[3][0] =5
sim_ma[3][1] =10
sim_ma[3][2] =5
sim_ma[3][3] =20
gap_score = -5              # Gap penalty initialization
for j in range(0,n):
        D[0][j+1] = gap_score*(j+1)
for i in range(0,m):
        D[i+1][0] = gap_score*(i+1)
for i in range(1,m+1):          #dynamic programming
calculation
        for j in range(1,n+1):
#trace  back to create alignment
t_aln = ''
s_aln = ''
while (i>0) and (j>0):
        if D[i][j] - sim_ma[bToInt(s[j-1])][:bToInt(t[i-1])]:
#diagonal  movement
        t_aln = t_aln + t[i-1]        #horizontal right movement
        s_aln = s_aln + s[j-1]        #vertical down movement
        i = i-1
        j = j-1
        elif D[i][j] - gap_score == D[i][j-1]:
        s_aln = s_aln + s[j-1]
        t_aln = t_aln + '_'
        j = j-1
        elif D[i][j] -  gap_score == D[i-1][j]:
        s_aln = s_aln + '_'
        t_aln = t_aln + t[i-1]
        i = i-1
        else :
      print('should not happen')
while (j>0):
        s_aln = s_aln + s[j-1]
        t_aln = t_aln + '_'
        j = j-1
elif i>0:
        while (i>0):
        s_aln = s_aln + '_'
        t_aln = t_aln + t[i-1]
        i = i-1
print s_aln[::-1]
print t_aln[::-1]
```

```
            match = D[i-1][j-1] +
sim_ma[bToInt(s[j-1])][bToInt(t[i-1])]
            gaps = D[i][j-1] + gap_score
            gapt = D[i-1][j] + gap_score
            D[i][j] = max(match,gaps,gapt)
score = D[m][n]
print(score)
```