Name: Nigel Nicholas

SID:1155088791

CSCI3220-Assignment 1 (Non-Programming Question)

*s*

*r*

G

T

A

C

C

C

T

C

C

C

Ø

-2

Ø

-2

0

2

-4

-6

-8

-10

-4

-6

-8

-10

4

0

0

-2

-4

-6

-2

-4

-6

2

0

-2

2

0

-2

3

4

2

1

-1

2

3

1

0

Optimal Alignment Score is: 1

Alignment 1:

r = \_GTACC

s = CCT\_CC

Alignment 2:

r = G\_TACC

s = CCT\_CC

Alignment 3:

r = GTACC

s = CCTCC



*s*

*r*

G

T

A

C

C

C

T

C

C

C

Ø

0

Ø

0

0

2

0

0

0

0

0

0

0

0

4

2

2

0

0

0

0

2

2

4

2

2

2

0

0

3

4

2

1

2

2

0

2

1

Local Alignment Best Score: 4

Alignment 1:

r = CC r[4..5]

s = CC s[1..2]

Alignment 2:

r = CC r[4..5]

s = CC s[4..5]

Alignment 3:

r = TACC r[2..5]

s = T\_CC s[3..5]

1. ( Programming Question ) , attached later in other file.
2. (a) With this scoring scheme, the penalty given to indel is 2 times higher than mismatch. And therefore if we visualize in a table, you would rather take a diagonal approach which is leading towards mismatch rather than indel. What is needed for each box, is that from that box, there should be a possibility of getting >= -5 through its path. So in a 6x6 box with index starting at 1, the boxes with row and col (4,1), (5,1), (6,1) , (5,2), (6,2), (6,3), (1,4), (1,5) , (2, 5) , (1,6) , (2,6), (3,6). If you plot these in the table, it is a triangle on bottom left and top right.

(b) The simplest algorithm would just simply use an if-condition for all these table entries given its row and column. Once you see the combination of row and column above, set the score to -1000 and return immediately to terminate the recursion.

1. (a)

|  |  |
| --- | --- |
| 2-mers subsequence of r | Position |
| AC | 1 |
| CA | 7 |
| CG | 2,4 |
| GC | 3 |
| GT | 5 |
| TC | 6 |

(b) r[2..3] & s[2..3]

r[1..2] & s[7..8]

r[3..4] & s[5..6]

r[4..6] & s[2..4]

r[7..8] & s[6..7]

(c) Algorithm:

function FASTA( s ):

for i=1 to length(s):

index= explore(i)

if( index – i >=1 ) {

y = lookup(s[i..i+1] )

putList( r[y..y+index] , s[i..i+index] )

i= index

}

function explore(i):

if i = length(s)

return length(s)

m = lookup( s[i..i+1] )

if m Not found:

return i

n = lookup( s[i+1..i+2])

if n Not found :

return i+1

if( n = m+1 ) : // if the position is next to it keep exploring til you cant find

j=explore(i+1)

else

return i+1

return j