40 Marks

RYERSON UNIVERSITY

Faculty of Engineering, Architecture and Science

Department of Electrical and Computer Engineering

Program: Biomedical Engineering

Course Number	**BME 808**
Course Title	**Computations in Genetic Engineering**
Semester/Year	Winter 2021
Instructor	**P Siddavaatam**

Tut Report No.	3

Report Title	**Dynamic Programming: Needleman-Wunsch in Python**

Section No.	8
Group No.	n/a
Submission Date	03/05/2021
Due Date	03/06/2021

Name	Student ID	Signature*
Andrew Mullen	500787631	

(Note: remove the first 4 digits from your student ID)

* By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your Academic record under the Student Code of Academic Conduct, which can be found online at: www.ryerson.ca/senate/current/pol60.pdf.

BME 808 – Tutorial 3

A. Needleman Wunsch Algorithm Function

```
def needleAlg(s1, s2, match, gap, mismatch) :
    N = len(s1)
    M = len(s2)

    mx = np.zeros((N+1, M+1))

    for i in range(1,N+1) :
        mx[i,0] = mx[i-1,0] + gap

    for j in range(1,M+1) :
        mx[0,j] = mx[0,j-1] + gap

    for i in range(1,N+1) :
        for j in range(1,M+1) :
            if s1[i-1] == s2[j-1] :
                score1 = mx[i-1,j-1] + match
            else :
                score1 = mx[i-1,j-1] + mismatch

            score2 = mx[i,j-1] + gap
            score3 = mx[i-1,j] + gap

            mx[i,j] = max(score1, score2, score3)

    return mx, N, M
```

Build Directional String Function

```
def buildDirectionalString(mx, N, M, gap) :
    dstring = ""

    currentRow = N
    currentCol = M

    while(currentRow != 0 and currentCol != 0) :

        if currentRow == 0 :
            dstring += 'H'
            currentCol -= 1

        elif currentCol == 0 :
            dstring += 'V'
            currentRow -= 1

        elif mx[currentRow, currentCol-1] + gap == mx[currentRow, currentCol] :
            dstring += 'H'
            currentCol -= 1

        elif mx[currentRow-1, currentCol] + gap == mx[currentRow, currentCol] :
            dstring += 'V'
            currentRow -= 1

        else :
            dstring += 'D'
            currentRow -= 1
            currentCol -= 1

    return dstring
```

Build Alignment Function

```
def buildAlignment(dstring, s1, s2, N, M) :
    s1pos = N-1
    s2pos = M-1
    dirpos = 0
    s1Align = ''
    s2Align = ''
    matchLine = ''
    dstringLen = len(dstring)

    while dirpos < dstringLen :
        if dstring[dirpos] == 'D' :
            if s1[s1pos] == s2[s2pos] :
                matchLine += '|'
            else :
                matchLine += ' '
            s1Align += s1[s1pos]
            s2Align += s2[s2pos]
            s1pos -= 1
            s2pos -= 1
            dirpos += 1

        elif dstring[dirpos] == 'V' :
            matchLine += ' '
            s1Align += s1[s1pos]
            s2Align += '-'
            s1pos -= 1
            dirpos += 1

        else :
            matchLine += ' '
            s1Align += '-'
            s2Align += s2[s2pos]
            s2pos -= 1
            dirpos += 1

    percentId = matchLine.count('|')/dstringLen
    percentGap = (s1Align.count('-')+s2Align.count('-'))/dstringLen

    return s1Align[::-1], matchLine[::-1], s2Align[::-1], percentId*100, percentGap*100
```

B.

```
# Initialize Variables
s1 = 'GTTGCG'
s2 = 'ATCGACG'
match = 1
gap = -1
mismatch = 0

# Call Functions
matrix, N, M = needleAlg(s1, s2, match, gap, mismatch)
dstring = buildDirectionalString(matrix, N, M, gap)
alignment = buildAlignment(dstring, s1, s2, N, M)

# Print Results
print('Needleman Wunsch Alignment Matrix:')
print(matrix)
print('\nDirectional String:')
print(dstring[::-1])
print('\nAlignment:')
print(alignment[0])
print(alignment[1])
print(alignment[2])
print('\nPercent ID:', alignment[3], '\nPercent Gap:', alignment[4])
```

```
Needleman Wunsch Alignment Matrix:
[[ 0. -1. -2. -3. -4. -5. -6. -7.]
 [-1.  0. -1. -2. -2. -3. -4. -5.]
 [-2. -1.  1.  0. -1. -2. -3. -4.]
 [-3. -2.  0.  1.  0. -1. -2. -3.]
 [-4. -3. -1.  0.  2.  1.  0. -1.]
 [-5. -4. -2.  0.  1.  2.  2.  1.]
 [-6. -5. -3. -1.  1.  1.  2.  3.]]

Directional String:
DDDDHDD

Alignment:
GTTG-CG
 | | ||
ATCGACG

Percent ID: 57.14285714285714
Percent Gap: 14.285714285714285
```

C.

```
# Read in sequences
sHFile = open('humanSeq.txt')
file = sHFile.readline()
H1 = sHFile.read()
H1 = H1.replace('\n', '')

sMFile = open('mouseSeq.txt')
file = sMFile.readline()
M1 = sMFile.read()
M1 = M1.replace('\n', '')

# Divide mouse and human sequences
M2 = M1[1900:2045]
H2 = H1[21:166]

# Save to files
M2file = open("M2seq.txt", "w+")
M2file.write(M2)
M2file.close()

H2file = open("H2seq.txt", "w+")
H2file.write(H2)
H2file.close()
```

D.

```python
# Initialize Values
gap = -1
match = 1
mismatch = 0

# Call Functions
matrix, N, M = needleAlg(M2, H1, match, gap, mismatch)
dstring = buildDirectionalString(matrix, N, M, gap)
alignment = buildAlignment(dstring, M2, H1, N, M)

# Print Results
print('Needleman Wunsch Alignment Matrix:')
print(matrix)
print('\nDirectional String:')
print(dstring[::-1])
print('\nAlignment:')
print(alignment[0][0:114])
print(alignment[1][0:114])
print(alignment[2][0:114])
print('\nAlignment Continued:')
print(alignment[0][114:])
print(alignment[1][114:])
print(alignment[2][114:])
print('\nPercent ID:', alignment[3], '\nPercent Gap:', alignment[4])
```

```
Needleman Wunsch Alignment Matrix:
[[   0.   -1.   -2. ... -180. -181. -182.]
 [  -1.    0.    0. ... -178. -179. -180.]
 [  -2.    0.    0. ... -176. -177. -178.]
 ...
 [-143. -141. -139. ...   92.   91.   91.]
 [-144. -142. -140. ...   94.   93.   92.]
 [-145. -143. -141. ...   95.   95.   94.]]

Directional String:
DDHHHHDDHHDHDHHHHHHHDDHHHDHDDDHHDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDHHHDHHDDHHHHHHDHDHHHDH

Alignment:
CT----AG--A-T-------CT---T-TCC--AGTGGCCGTCTCTATTGGGTTGATTCCAAACTCCACTCTATCTCCAGCATCGATGTCAATGGGGGCAATCGGAAAACCATTT
||    |   | |       ||   | |||  |||||||| ||||| |||||||| |||||||| ||||| ||||| ||||||||||||||||||||||| ||||| ||||| |
CTCCTTATCCACTTGTGTGTCTAGATCTCCTCAGTGGCCGCCTCTACTGGGTTGACTCCAAACTTCACTCCATCTCAAGCATCGATGTCAATGGGGGCAACCGGAAGACCATCT

Alignment Continued:
TGGAGGATGAGAACCGGCTGGCCCACCCCTTCTCCTTGGCCATCT---A--TG------A-G---G-
|||||||||| ||  |||||||||||||||||||||||||| |||   |  ||      | |   |
TGGAGGATGAAAAGAGGCTGGCCCACCCCTTCTCCTTGGCCGTCTTTGAGGTGTGGCTTACGTACGA

Percent ID: 72.37569060773481
Percent Gap: 19.88950276243094
```

E.

```python
# Initialize Values
gap = -1
match = 1
mismatch = 0

# Call Functions
matrix, N, M = needleAlg(M2, H2, match, gap, mismatch)
dstring = buildDirectionalString(matrix, N, M, gap)
alignment = buildAlignment(dstring, M2, H2, N, M)

# Print Results
print('Needleman Wunsch Alignment Matrix:')
print(matrix)
print('\nDirectional String:')
print(dstring[::-1])
print('\nAlignment:')
print(alignment[0][0:114])
print(alignment[1][0:114])
print(alignment[2][0:114])
print('\nAlignment Continued:')
print(alignment[0][114:])
print(alignment[1][114:])
print(alignment[2][114:])
print('\nPercent ID:', alignment[3], '\nPercent Gap:', alignment[4])
```

```
Needleman Wunsch Alignment Matrix:
[[   0.   -1.   -2. ... -143. -144. -145.]
 [  -1.    1.    0. ... -141. -142. -143.]
 [  -2.    0.    2. ... -139. -140. -141.]
 ...
 [-143. -141. -139. ...  126.  125.  124.]
 [-144. -142. -140. ...  125.  127.  126.]
 [-145. -143. -141. ...  124.  126.  128.]]

Directional String:
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD

Alignment:
CTAGATCTTTCCAGTGGCCGTCTCTATTGGGTTGATTCCAAACTCCACTCTATCTCCAGCATCGATGTCAATGGGGGCAATCGGAAAACCATTTTGGAGGATGAGAACCGGCTG
||||||||   ||||||||| ||||| |||||||| |||||||| ||||| ||||| ||||||||||||||||||||||| ||||| ||||| ||||||||||| ||  |||||
CTAGATCTCCTCAGTGGCCGCCTCTACTGGGTTGACTCCAAACTTCACTCCATCTCAAGCATCGATGTCAATGGGGGCAACCGGAAGACCATCTTGGAGGATGAAAAGAGGCTG

Alignment Continued:
GCCCACCCCTTCTCCTTGGCCATCTATGAGG
||||||||||||||||||||| ||| |||||
GCCCACCCCTTCTCCTTGGCCGTCTTTGAGG

Percent ID: 88.27586206896552
Percent Gap: 0.0
```

F. The alignment in part E has a higher percent identity and a lower percent gap which means that the sequences are more similar, and the algorithm didn't have to insert gaps to make them align optimally. This means that the alignment in part E is more biologically relevant than part D.

G. A better alignment program might be changing up the gap penalties to make them less likely to introduce gaps. This could result in longer uninterrupted nucleotide sequences and they would be more biologically relevant.