Tutorial 2

February 23, 2019

1 Tutorial 2

1.1 Week 3-5

1.1.1 1) Write a program to implement Needleman-Wunsch for proteins

Importing blosum50 scoring matrix

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Importing blosum50 scoring matrix, adding column and row headers
header = ['A', 'R', 'N', 'D', 'C', 'Q','E', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S
df=pd.read_csv('blosum50.txt', delim_whitespace = True, names = header, index_col = Fa
df.rename(index = dict(zip(range(len(header)),header)),inplace =True)
df.head(21)
```

Out[2]:

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-3	15	2	-3
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5

In [2]:
```
m = 11
n = 8

# Creating a table to run HEAGAWGHEE versus PAWHEAE
h = [' ', 'H', 'E', 'A', 'G', 'A', 'W','G', 'H', 'E', 'E']
header2 = [' ','P', 'A', 'W', 'H', 'E', 'A','E']
df1 = pd.DataFrame(data=[[np.zeros(3)]*m]*n)
df1.rename(index = dict(zip(range(len(header2)),header2)),inplace =True)
df1.rename(columns=dict(zip(range(len(h)),h)), inplace = True)

df1
```

Out[2]:

		H	E	A	\
	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
P	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
A	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
W	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
H	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
E	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
A	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
E	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	

		G	A	W	G	\
	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]		
P	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]		
A	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]		
W	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]		
H	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]		
E	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]		
A	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]		
E	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]		

		H	E	E
	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
P	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
A	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
W	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
H	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
E	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
A	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
E	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	

In [3]:
```
# Filling the Global Allignment Table (array in row-colum pair)
#Forward Pass

cost = -8 # Cost of insertion or delition

# Calculating price of substitution when arrows moves diagonally
```

```
def blos(r, c):
    cname = df1.columns[c]
    rname = df1.index[r]
    if r == 0 or c == 0:
        return np.nan # Handling nan error when filling first row/column
    else:
        return int(df[rname][cname]) + int(df1.iloc[r-1,c-1][0])

arrow =[]

# Loop to fill the table, deciding between if the upelement, leftelement or the diagel
for r in range(0,8):
    for c in range(0,11):
        upelement = int(df1.iloc[r-1,c][0]) + cost if r >= 1 else np.nan # Handling na
        leftelement = int(df1.iloc[r,c-1][0]) + cost if c >= 1 else np.nan # Handling
        diagelement = blos(r, c)
        if r == 0 and c == 0:
            upelement = 0
        list1 = np.array([diagelement, upelement,leftelement])
        re = np.nanargmax(list1)
        # Storing the location of the selected max element to find arrow direction dur
        if list1[re] == diagelement:
            vec = [r-1,c-1]
            arrow.extend([r-1,c-1, r,c])
        elif list1[re] == upelement:
            vec = [r-1,c]
            arrow.extend([r-1,c, r,c])
        elif list1[re] == leftelement:
            vec = [r,c-1]
            arrow.extend([r,c-1, r,c])
        # Storing value, row and column coordinates in an array for each row-column pa
        df1.iloc[r,c] = np.array([list1[re], vec[0], vec[1]])

df1
```

Out[3]:

		H	E	A	\
	[0.0, -1.0, 0.0]	[-8.0, 0.0, 0.0]	[-16.0, 0.0, 1.0]	[-24.0, 0.0, 2.0]	
P	[-8.0, 0.0, 0.0]	[-2, 0, 0]	[-9, 0, 1]	[-17, 0, 2]	
A	[-16.0, 1.0, 0.0]	[-10, 1, 0]	[-3, 1, 1]	[-4, 1, 2]	
W	[-24.0, 2.0, 0.0]	[-18, 2, 1]	[-11, 2, 2]	[-6, 2, 2]	
H	[-32.0, 3.0, 0.0]	[-14, 3, 0]	[-18, 3, 1]	[-13, 3, 2]	
E	[-40.0, 4.0, 0.0]	[-22, 4, 1]	[-8, 4, 1]	[-16, 5, 2]	
A	[-48.0, 5.0, 0.0]	[-30, 5, 1]	[-16, 5, 2]	[-3, 5, 2]	
E	[-56.0, 6.0, 0.0]	[-38, 6, 1]	[-24, 6, 1]	[-11, 6, 3]	

	G	A	W	G	\
	[-32.0, 0.0, 3.0]	[-40.0, 0.0, 4.0]	[-48.0, 0.0, 5.0]	[-56.0, 0.0, 6.0]	
P	[-25, 1, 3]	[-33, 0, 4]	[-41, 1, 5]	[-49, 1, 6]	

```
A        [-12, 2, 3]        [-20, 1, 4]        [-28, 2, 5]        [-36, 2, 6]
W         [-7, 2, 3]        [-15, 2, 4]         [-5, 2, 5]        [-13, 3, 6]
H         [-8, 3, 3]         [-9, 3, 4]        [-13, 3, 6]         [-7, 3, 6]
E        [-16, 4, 3]         [-9, 4, 4]        [-12, 4, 5]        [-15, 4, 7]
A        [-11, 6, 3]        [-11, 5, 4]        [-12, 5, 5]        [-12, 5, 6]
E         [-6, 6, 3]        [-12, 6, 4]        [-14, 6, 5]        [-15, 6, 6]

                   H                  E                  E
   [-64.0, 0.0, 7.0]  [-72.0, 0.0, 8.0]  [-80.0, 0.0, 9.0]
P        [-57, 1, 7]        [-65, 0, 8]        [-73, 0, 9]
A        [-44, 2, 7]        [-52, 2, 8]        [-60, 2, 9]
W        [-21, 3, 7]        [-29, 3, 8]        [-37, 3, 9]
H         [-3, 3, 7]        [-11, 4, 8]        [-19, 4, 9]
E         [-7, 4, 7]          [3, 4, 8]         [-5, 4, 9]
A        [-15, 5, 8]         [-5, 5, 9]          [2, 5, 9]
E        [-12, 6, 7]         [-9, 6, 8]          [1, 6, 9]
```

In [4]:
```python
# Filling the Global Allignment Table (integer in row-column pair)
#Forward Pass

cost = -8 # Cost of insertion or delition

# Calculating price of substitution when arrows moves diagonally
def blos(r, c):
    cname = df1.columns[c]
    rname = df1.index[r]
    if r == 0 or c == 0:
        return np.nan # Handling nan error when filling first row/column
    else:
        return int(df[rname][cname]) + int(df1.iloc[r-1,c-1])

arrow =[]

# Loop to fill the table, deciding between if the upelement, leftelement or the diagel
for r in range(0,8):
    for c in range(0,11):
        upelement = int(df1.iloc[r-1,c]) + cost if r >= 1 else np.nan # Handling nan e
        leftelement = int(df1.iloc[r,c-1]) + cost if c >= 1 else np.nan # Handling nan
        diagelement = blos(r, c)
        if r == 0 and c == 0:
            upelement = 0
        list1 = np.array([diagelement, upelement,leftelement])
        # Storing the max integer in row-column pair to fill table
        re = np.nanargmax(list1)
        df1.iloc[r,c] = list1[re]
        # Storing the location of the selected max element and of the corrisponding ad
        # to find arrow direction during the backward pass
        if list1[re] == diagelement:
```

```
                arrow.extend([r-1,c-1, r,c])
            elif list1[re] == upelement:
                arrow.extend([r-1,c, r,c])
            elif list1[re] == leftelement:
                arrow.extend([r,c-1, r,c])
```

```
df1
```

Out[4]:

		H	E	A	G	A	W	G	H	E	E
	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80
P	-8	-2	-9	-17	-25	-33	-41	-49	-57	-65	-73
A	-16	-10	-3	-4	-12	-20	-28	-36	-44	-52	-60
W	-24	-18	-11	-6	-7	-15	-5	-13	-21	-29	-37
H	-32	-14	-18	-13	-8	-9	-13	-7	-3	-11	-19
E	-40	-22	-8	-16	-16	-9	-12	-15	-7	3	-5
A	-48	-30	-16	-3	-11	-11	-12	-12	-15	-5	2
E	-56	-38	-24	-11	-6	-12	-14	-15	-12	-9	1

In [5]:
```
# Backward pass

# Making empty lists where to append the backward pass elements and the two strings
back = []
letter1 = []
letter2 = []

# Starting from the last column-row pair in the table
K = arrow[len(arrow)-1] # Storing it's column coordinate
M = arrow[len(arrow)-1-1] # Storing it's row coordinate
h1 = arrow[len(arrow)-1-2] # Storing the column coordinate where the arrow in the bloc
h2 = arrow[len(arrow)-1-3] # Storing the row coordinate where the arrow in the block e
back.append(df1.iloc[r,c]) # Storing the element corrisponding to the M,K coordinates
letter1.append(df1.index[r]) # Storing the letter corrisponding to the M row to form t
letter2.append(df1.columns[c]) # Storing the letter corrisponding to the K column to f

i = len(arrow) -5 # Looping throgh all the arrow array apart from all the initializati

while i > 0:
    if ((h1 <= arrow[i-2]) and (h2 <= arrow[i-3])): # If there are duplicates or coord
                                                    # in the arrow array, skip them
        i -=2
    elif ((arrow[i] == h1) and (arrow[i-1] == h2)): # If while going through the array
                                                    # are in is pointing at, store its
                                                    # coorrdinates this new element is
                                                    # corrispondent row/column letter
                                                    # respect to the previous element
                                                    # add - in the first string instea
                                                    # previous element the column numb
                                                    # the second string instead of the
```

```
                    K = arrow[i]
                    M = arrow[i-1]
                    h1 = arrow[i-2]
                    h2 = arrow[i-3]
                    back.append(df1.iloc[M,K])
                    if i>5:                                  # If we are too close to the end of
                                                             # are stored each time one iteratio

                        if((K==h1) and (M!=h2)):
                            letter2.append('-')
                            letter1.append(df1.index[M])
                        elif((K!=h1) and (M==h2)):
                            letter1.append('-')
                            letter2.append(df1.columns[K])
                        else:
                            letter1.append(df1.index[M])
                            letter2.append(df1.columns[K])
                    i -=4                                    # Once we store these 4 erray eleme
                                                             # iteration to avoid duplicates
            else:
                i -=1 # If no corrispondence if found go ahead to the next array element

        back = np.asarray(back)
        print(letter2[::-1]) # Reversing the string letter order
        print(letter1[::-1]) # Reversing the string letter order
        print(back)
```

```
['H', 'E', 'A', 'G', 'A', 'W', 'G', 'H', 'E', '-', 'E']
['-', '-', 'P', '-', 'A', 'W', '-', 'H', 'E', 'A', 'E']
[  1.  -5.   3.  -3. -13.  -5. -20. -25. -17. -16.  -8.   0.]
```

In [6]:
```
# Creating a table to Match the protein sequence SALPQPTTPVSSFTSGSMLGRTDTALTNTYSAL wit

m = 34
n = 35

# adding '' and , automatically to string
# import re
# st = 'S A L P Q P T T P V S S F T S G S M L G R T D T A L T N T Y S A L'
# print(re.findall(r"[\w']+", st))

# adding column and row headers
h = [' ', 'S', 'A', 'L', 'P', 'Q', 'P', 'T', 'T', 'P', 'V', 'S', 'S', 'F', 'T', 'S', '
         'D', 'T', 'A', 'L', 'T', 'N', 'T', 'Y', 'S', 'A', 'L']
header2 = [' ','P', 'S', 'P', 'T', 'M', 'E', 'A', 'V', 'T', 'S', 'V', 'E', 'A', 'S', '
           'P', 'H', 'S', 'T', 'S', 'S', 'Y', 'F', 'A', 'T', 'T', 'Y', 'Y', 'H', 'L',
df2 = pd.DataFrame(data=[[np.zeros(1)]*m]*n)
df2.rename(index = dict(zip(range(len(header2)),header2)),inplace =True)
```

```
df2.rename(columns=dict(zip(range(len(h)),h)), inplace = True)

df2
```

Out[6]:

	S	A	L	P	Q	P	T	T	P	\
	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
P	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
S	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
P	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
T	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
M	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
E	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
A	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
V	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
T	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
S	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
V	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
E	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
A	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
S	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
T	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
A	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
S	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
H	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
P	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
H	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
S	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
T	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
S	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
S	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
Y	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
F	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
A	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
T	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
T	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
Y	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
Y	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
H	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
L	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
Y	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	

	...	T	A	L	T	N	T	Y	S	A	L
	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
P	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
S	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
P	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
T	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
M	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]

```
E  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
A  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
V  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
T  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
S  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
V  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
E  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
A  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
S  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
T  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
A  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
S  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
H  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
P  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
H  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
S  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
T  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
S  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
S  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
Y  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
F  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
A  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
T  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
T  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
Y  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
Y  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
H  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
L  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]
Y  ...    [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]  [0.0]

[35 rows x 34 columns]
```

In [7]:
```python
# Same as:
# Filling the Global Allignment Table (array in row-colum pair)
#Forward Pass

cost = -8

def blos(r, c):
    cname = df2.columns[c]
    rname = df2.index[r]
    if r == 0 or c == 0:
        return np.nan
    else:
        return int(df[rname][cname]) + int(df2.iloc[r-1,c-1][0])

arrow =[]
```

```
for r in range(0,35):
    for c in range(0,34):
        upelement = int(df2.iloc[r-1,c][0]) + cost if r >= 1 else np.nan
        leftelement = int(df2.iloc[r,c-1][0]) + cost if c >= 1 else np.nan
        diagelement = blos(r, c)
        if r == 0 and c == 0:
            upelement = 0
        list1 = np.array([diagelement, upelement,leftelement])
        re = np.nanargmax(list1)
        #df1.iloc[r,c] = list1[re]
        if list1[re] == diagelement:
            vec = [r-1,c-1]
            arrow.extend([r-1,c-1, r,c])
        elif list1[re] == upelement:
            vec = [r-1,c]
            arrow.extend([r-1,c, r,c])
        elif list1[re] == leftelement:
            vec = [r,c-1]
            arrow.extend([r,c-1, r,c])
        df2.iloc[r,c] = np.array([list1[re], vec[0], vec[1]])

df2
```

Out[7]:

		S	A	\
	[0.0, -1.0, 0.0]	[-8.0, 0.0, 0.0]	[-16.0, 0.0, 1.0]	
P	[-8.0, 0.0, 0.0]	[-1, 0, 0]	[-9, 0, 1]	
S	[-16.0, 1.0, 0.0]	[-3, 1, 0]	[0, 1, 1]	
P	[-24.0, 2.0, 0.0]	[-11, 2, 1]	[-4, 2, 1]	
T	[-32.0, 3.0, 0.0]	[-19, 3, 1]	[-11, 3, 1]	
M	[-40.0, 4.0, 0.0]	[-27, 4, 1]	[-19, 4, 2]	
E	[-48.0, 5.0, 0.0]	[-35, 5, 1]	[-27, 5, 2]	
A	[-56.0, 6.0, 0.0]	[-43, 6, 1]	[-30, 6, 1]	
V	[-64.0, 7.0, 0.0]	[-51, 7, 1]	[-38, 7, 2]	
T	[-72.0, 8.0, 0.0]	[-59, 8, 1]	[-46, 8, 2]	
S	[-80.0, 9.0, 0.0]	[-67, 9, 0]	[-54, 9, 2]	
V	[-88.0, 10.0, 0.0]	[-75, 10, 1]	[-62, 10, 2]	
E	[-96.0, 11.0, 0.0]	[-83, 11, 1]	[-70, 11, 2]	
A	[-104.0, 12.0, 0.0]	[-91, 12, 1]	[-78, 12, 1]	
S	[-112.0, 13.0, 0.0]	[-99, 13, 0]	[-86, 13, 2]	
T	[-120.0, 14.0, 0.0]	[-107, 14, 1]	[-94, 14, 2]	
A	[-128.0, 15.0, 0.0]	[-115, 15, 1]	[-102, 15, 1]	
S	[-136.0, 16.0, 0.0]	[-123, 16, 0]	[-110, 16, 2]	
H	[-144.0, 17.0, 0.0]	[-131, 17, 1]	[-118, 17, 2]	
P	[-152.0, 18.0, 0.0]	[-139, 18, 1]	[-126, 18, 2]	
H	[-160.0, 19.0, 0.0]	[-147, 19, 1]	[-134, 19, 2]	
S	[-168.0, 20.0, 0.0]	[-155, 20, 0]	[-142, 20, 2]	
T	[-176.0, 21.0, 0.0]	[-163, 21, 1]	[-150, 21, 2]	
S	[-184.0, 22.0, 0.0]	[-171, 22, 0]	[-158, 22, 2]	

S	[-192.0, 23.0, 0.0]	[-179, 23, 0]	[-166, 23, 2]
Y	[-200.0, 24.0, 0.0]	[-187, 24, 1]	[-174, 24, 2]
F	[-208.0, 25.0, 0.0]	[-195, 25, 1]	[-182, 25, 2]
A	[-216.0, 26.0, 0.0]	[-203, 26, 1]	[-190, 26, 1]
T	[-224.0, 27.0, 0.0]	[-211, 27, 1]	[-198, 27, 2]
T	[-232.0, 28.0, 0.0]	[-219, 28, 1]	[-206, 28, 2]
Y	[-240.0, 29.0, 0.0]	[-227, 29, 1]	[-214, 29, 2]
Y	[-248.0, 30.0, 0.0]	[-235, 30, 1]	[-222, 30, 2]
H	[-256.0, 31.0, 0.0]	[-243, 31, 1]	[-230, 31, 2]
L	[-264.0, 32.0, 0.0]	[-251, 32, 1]	[-238, 32, 2]
Y	[-272.0, 33.0, 0.0]	[-259, 33, 1]	[-246, 33, 2]

	L	P	Q	P \
	[-24.0, 0.0, 2.0]	[-32.0, 0.0, 3.0]	[-40.0, 0.0, 4.0]	[-48.0, 0.0, 5.0]
P	[-17, 1, 2]	[-14, 0, 3]	[-22, 1, 4]	[-30, 0, 5]
S	[-8, 2, 2]	[-16, 2, 3]	[-14, 1, 4]	[-22, 2, 5]
P	[-4, 2, 2]	[2, 2, 3]	[-6, 3, 4]	[-4, 2, 5]
T	[-5, 3, 2]	[-5, 3, 3]	[1, 3, 4]	[-7, 3, 5]
M	[-8, 4, 2]	[-8, 4, 3]	[-5, 4, 4]	[-2, 4, 5]
E	[-16, 5, 3]	[-9, 5, 3]	[-6, 5, 4]	[-6, 5, 5]
A	[-24, 6, 3]	[-17, 6, 3]	[-10, 6, 4]	[-7, 6, 5]
V	[-29, 7, 2]	[-25, 7, 4]	[-18, 7, 5]	[-13, 7, 5]
T	[-37, 8, 3]	[-30, 8, 3]	[-26, 8, 4]	[-19, 8, 5]
S	[-45, 9, 3]	[-38, 9, 3]	[-30, 9, 4]	[-27, 9, 5]
V	[-53, 10, 2]	[-46, 10, 4]	[-38, 10, 5]	[-33, 10, 5]
E	[-61, 11, 3]	[-54, 11, 3]	[-44, 11, 4]	[-39, 11, 5]
A	[-69, 12, 3]	[-62, 12, 3]	[-52, 12, 5]	[-45, 12, 5]
S	[-77, 13, 3]	[-70, 13, 3]	[-60, 13, 5]	[-53, 13, 5]
T	[-85, 14, 3]	[-78, 14, 3]	[-68, 14, 5]	[-61, 14, 5]
A	[-93, 15, 3]	[-86, 15, 3]	[-76, 15, 5]	[-69, 15, 5]
S	[-101, 16, 3]	[-94, 16, 3]	[-84, 16, 5]	[-77, 16, 5]
H	[-109, 17, 3]	[-102, 17, 4]	[-92, 17, 5]	[-85, 17, 6]
P	[-117, 18, 3]	[-99, 18, 3]	[-100, 18, 5]	[-82, 18, 5]
H	[-125, 19, 3]	[-107, 19, 4]	[-98, 19, 4]	[-90, 19, 6]
S	[-133, 20, 3]	[-115, 20, 4]	[-106, 20, 5]	[-98, 20, 6]
T	[-141, 21, 3]	[-123, 21, 4]	[-114, 21, 5]	[-106, 21, 6]
S	[-149, 22, 3]	[-131, 22, 4]	[-122, 22, 5]	[-114, 22, 6]
S	[-157, 23, 3]	[-139, 23, 4]	[-130, 23, 5]	[-122, 23, 6]
Y	[-165, 24, 3]	[-147, 24, 4]	[-138, 24, 5]	[-130, 24, 6]
F	[-173, 25, 2]	[-155, 25, 4]	[-146, 25, 5]	[-138, 25, 6]
A	[-181, 26, 3]	[-163, 26, 4]	[-154, 26, 5]	[-146, 26, 6]
T	[-189, 27, 3]	[-171, 27, 4]	[-162, 27, 5]	[-154, 27, 6]
T	[-197, 28, 3]	[-179, 28, 4]	[-170, 28, 5]	[-162, 28, 6]
Y	[-205, 29, 3]	[-187, 29, 4]	[-178, 29, 5]	[-170, 29, 6]
Y	[-213, 30, 3]	[-195, 30, 4]	[-186, 30, 5]	[-178, 30, 6]
H	[-221, 31, 3]	[-203, 31, 4]	[-194, 31, 4]	[-186, 31, 6]
L	[-225, 32, 2]	[-211, 32, 4]	[-202, 32, 5]	[-194, 32, 6]
Y	[-233, 33, 3]	[-219, 33, 4]	[-210, 33, 5]	[-202, 33, 6]

	T	T	P	\
	[-56.0, 0.0, 6.0]	[-64.0, 0.0, 7.0]	[-72.0, 0.0, 8.0]	
P	[-38, 1, 6]	[-46, 1, 7]	[-54, 0, 8]	
S	[-28, 1, 6]	[-36, 1, 7]	[-44, 2, 8]	
P	[-12, 3, 6]	[-20, 3, 7]	[-26, 2, 8]	
T	[1, 3, 6]	[-7, 3, 7]	[-15, 4, 8]	
M	[-7, 4, 7]	[0, 4, 7]	[-8, 5, 8]	
E	[-3, 5, 6]	[-8, 5, 7]	[-1, 5, 8]	
A	[-6, 6, 6]	[-3, 6, 7]	[-9, 6, 8]	
V	[-7, 7, 6]	[-6, 7, 7]	[-6, 7, 8]	
T	[-8, 8, 6]	[-2, 8, 7]	[-7, 8, 8]	
S	[-16, 9, 7]	[-6, 9, 7]	[-3, 9, 8]	
V	[-24, 10, 7]	[-14, 10, 8]	[-9, 10, 8]	
E	[-32, 11, 7]	[-22, 11, 8]	[-15, 11, 8]	
A	[-39, 12, 6]	[-30, 12, 8]	[-23, 12, 8]	
S	[-43, 13, 6]	[-37, 13, 7]	[-31, 13, 8]	
T	[-48, 14, 6]	[-38, 14, 7]	[-38, 14, 8]	
A	[-56, 15, 7]	[-46, 15, 8]	[-39, 15, 8]	
S	[-64, 16, 7]	[-54, 16, 7]	[-47, 16, 8]	
H	[-72, 17, 7]	[-62, 17, 8]	[-55, 17, 9]	
P	[-80, 18, 7]	[-70, 18, 8]	[-52, 18, 8]	
H	[-84, 19, 6]	[-78, 19, 8]	[-60, 19, 9]	
S	[-88, 20, 6]	[-82, 20, 7]	[-68, 20, 9]	
T	[-93, 21, 6]	[-83, 21, 7]	[-76, 21, 9]	
S	[-101, 22, 7]	[-91, 22, 7]	[-84, 22, 8]	
S	[-109, 23, 7]	[-99, 23, 7]	[-92, 23, 8]	
Y	[-117, 24, 7]	[-107, 24, 8]	[-100, 24, 9]	
F	[-125, 25, 7]	[-115, 25, 8]	[-108, 25, 9]	
A	[-133, 26, 7]	[-123, 26, 8]	[-116, 26, 8]	
T	[-141, 27, 6]	[-128, 27, 7]	[-124, 27, 8]	
T	[-149, 28, 6]	[-136, 28, 7]	[-129, 28, 8]	
Y	[-157, 29, 7]	[-144, 29, 8]	[-137, 29, 9]	
Y	[-165, 30, 7]	[-152, 30, 8]	[-145, 30, 9]	
H	[-173, 31, 7]	[-160, 31, 8]	[-153, 31, 9]	
L	[-181, 32, 7]	[-168, 32, 8]	[-161, 32, 9]	
Y	[-189, 33, 7]	[-176, 33, 8]	[-169, 33, 9]	

	...	T	A	\
	...	[-192.0, 0.0, 23.0]	[-200.0, 0.0, 24.0]	
P	...	[-174, 1, 23]	[-182, 1, 24]	
S	...	[-161, 2, 23]	[-169, 2, 24]	
P	...	[-146, 3, 23]	[-154, 3, 24]	
T	...	[-133, 3, 23]	[-141, 4, 24]	
M	...	[-118, 5, 23]	[-126, 5, 24]	
E	...	[-108, 6, 23]	[-116, 6, 24]	
A	...	[-100, 6, 23]	[-103, 6, 24]	
V	...	[-94, 7, 23]	[-100, 7, 24]	

T	...	[-84, 8, 23]	[-92, 9, 24]
S	...	[-77, 9, 23]	[-83, 9, 24]
V	...	[-68, 11, 23]	[-76, 11, 24]
E	...	[-58, 12, 23]	[-66, 12, 24]
A	...	[-50, 12, 23]	[-53, 12, 24]
S	...	[-42, 13, 23]	[-49, 13, 24]
T	...	[-31, 14, 23]	[-39, 15, 24]
A	...	[-27, 15, 23]	[-26, 15, 24]
S	...	[-19, 16, 23]	[-26, 16, 24]
H	...	[-22, 17, 23]	[-21, 17, 24]
P	...	[-17, 18, 23]	[-23, 18, 24]
H	...	[-13, 20, 23]	[-19, 19, 24]
S	...	[-3, 20, 23]	[-11, 21, 24]
T	...	[-2, 21, 23]	[-3, 21, 24]
S	...	[-5, 22, 23]	[-1, 22, 24]
S	...	[2, 23, 23]	[-4, 23, 24]
Y	...	[-3, 24, 23]	[0, 24, 24]
F	...	[-10, 25, 23]	[-6, 25, 24]
A	...	[-16, 26, 23]	[-5, 26, 24]
T	...	[-14, 27, 23]	[-13, 27, 25]
T	...	[-19, 28, 23]	[-14, 28, 24]
Y	...	[-27, 29, 24]	[-21, 29, 24]
Y	...	[-28, 30, 23]	[-29, 30, 24]
H	...	[-36, 31, 23]	[-30, 31, 24]
L	...	[-41, 32, 23]	[-38, 32, 24]
Y	...	[-49, 33, 24]	[-43, 33, 24]

	L	T	N \
	[-208.0, 0.0, 25.0]	[-216.0, 0.0, 26.0]	[-224.0, 0.0, 27.0]
P	[-190, 1, 25]	[-198, 1, 26]	[-206, 1, 27]
S	[-177, 2, 25]	[-185, 2, 26]	[-193, 2, 27]
P	[-162, 3, 25]	[-170, 3, 26]	[-178, 3, 27]
T	[-149, 4, 25]	[-157, 3, 26]	[-165, 4, 27]
M	[-134, 5, 25]	[-142, 5, 26]	[-150, 5, 27]
E	[-124, 6, 25]	[-132, 6, 26]	[-140, 6, 27]
A	[-111, 7, 25]	[-119, 7, 26]	[-127, 7, 27]
V	[-102, 7, 25]	[-110, 8, 26]	[-118, 8, 27]
T	[-100, 9, 25]	[-97, 8, 26]	[-105, 9, 27]
S	[-91, 10, 25]	[-98, 9, 26]	[-96, 9, 27]
V	[-82, 10, 25]	[-90, 11, 26]	[-98, 11, 27]
E	[-74, 12, 25]	[-82, 12, 26]	[-90, 11, 27]
A	[-61, 13, 25]	[-69, 13, 26]	[-77, 13, 27]
S	[-56, 13, 25]	[-59, 13, 26]	[-67, 14, 27]
T	[-47, 15, 25]	[-51, 14, 26]	[-59, 14, 27]
A	[-34, 16, 25]	[-42, 16, 26]	[-50, 16, 27]
S	[-29, 16, 25]	[-32, 16, 26]	[-40, 17, 27]
H	[-29, 17, 25]	[-31, 17, 26]	[-31, 17, 27]
P	[-25, 18, 25]	[-30, 18, 26]	[-33, 18, 27]

H	[-26, 19, 25]	[-27, 19, 26]	[-29, 19, 27]
S	[-19, 21, 25]	[-24, 20, 26]	[-26, 20, 27]
T	[-11, 22, 25]	[-14, 21, 26]	[-22, 22, 27]
S	[-6, 22, 25]	[-9, 22, 26]	[-13, 22, 27]
S	[-4, 23, 25]	[-4, 23, 26]	[-8, 23, 27]
Y	[-5, 24, 25]	[-6, 24, 26]	[-6, 24, 27]
F	[1, 25, 25]	[-7, 25, 26]	[-10, 25, 27]
A	[-7, 26, 26]	[1, 26, 26]	[-7, 27, 27]
T	[-6, 27, 25]	[-2, 27, 26]	[1, 27, 27]
T	[-14, 28, 25]	[-1, 28, 26]	[-2, 28, 27]
Y	[-15, 29, 25]	[-9, 29, 27]	[-3, 29, 27]
Y	[-22, 30, 25]	[-17, 30, 26]	[-11, 30, 27]
H	[-30, 31, 26]	[-24, 31, 26]	[-16, 31, 27]
L	[-25, 32, 25]	[-31, 32, 26]	[-24, 32, 28]
Y	[-33, 33, 26]	[-27, 33, 26]	[-32, 33, 28]

	T	Y	S \
	[-232.0, 0.0, 28.0]	[-240.0, 0.0, 29.0]	[-248.0, 0.0, 30.0]
P	[-214, 1, 28]	[-222, 1, 29]	[-230, 1, 30]
S	[-201, 2, 28]	[-209, 2, 29]	[-217, 1, 30]
P	[-186, 3, 28]	[-194, 3, 29]	[-202, 3, 30]
T	[-173, 3, 28]	[-181, 4, 29]	[-189, 4, 30]
M	[-158, 5, 28]	[-166, 5, 29]	[-174, 5, 30]
E	[-148, 6, 28]	[-156, 6, 29]	[-164, 6, 30]
A	[-135, 7, 28]	[-143, 7, 29]	[-151, 7, 30]
V	[-126, 8, 28]	[-134, 8, 29]	[-142, 8, 30]
T	[-113, 8, 28]	[-121, 9, 29]	[-129, 9, 30]
S	[-103, 9, 28]	[-111, 10, 29]	[-116, 9, 30]
V	[-96, 10, 28]	[-104, 10, 29]	[-112, 11, 30]
E	[-98, 12, 28]	[-98, 11, 29]	[-105, 11, 30]
A	[-85, 13, 28]	[-93, 13, 29]	[-97, 12, 30]
S	[-75, 13, 28]	[-83, 14, 29]	[-88, 13, 30]
T	[-62, 14, 28]	[-70, 15, 29]	[-78, 15, 30]
A	[-58, 16, 28]	[-64, 15, 29]	[-69, 15, 30]
S	[-48, 16, 28]	[-56, 17, 29]	[-59, 16, 30]
H	[-39, 18, 28]	[-46, 17, 29]	[-54, 18, 30]
P	[-32, 18, 28]	[-40, 19, 29]	[-47, 18, 30]
H	[-35, 19, 28]	[-30, 19, 29]	[-38, 20, 30]
S	[-27, 20, 28]	[-35, 21, 29]	[-25, 20, 30]
T	[-21, 21, 28]	[-29, 21, 29]	[-33, 21, 30]
S	[-20, 22, 28]	[-23, 22, 29]	[-24, 22, 30]
S	[-11, 23, 28]	[-19, 24, 29]	[-18, 23, 30]
Y	[-10, 24, 28]	[-3, 24, 29]	[-11, 25, 30]
F	[-8, 25, 28]	[-6, 25, 29]	[-6, 25, 30]
A	[-10, 26, 28]	[-10, 26, 29]	[-5, 26, 30]
T	[-2, 27, 28]	[-10, 28, 29]	[-8, 27, 30]
T	[6, 28, 28]	[-2, 29, 29]	[-8, 28, 30]
Y	[-2, 29, 29]	[14, 29, 29]	[6, 30, 30]

```
Y         [-5, 30, 28]          [6, 30, 29]         [12, 30, 30]
H        [-13, 31, 28]         [-2, 31, 30]          [5, 31, 30]
L        [-17, 32, 28]        [-10, 32, 30]         [-3, 32, 31]
Y        [-25, 33, 29]         [-9, 33, 29]        [-11, 33, 31]
```

	A	L
	[-256.0, 0.0, 31.0]	[-264.0, 0.0, 32.0]
P	[-238, 1, 31]	[-246, 1, 32]
S	[-225, 2, 31]	[-233, 2, 32]
P	[-210, 3, 31]	[-218, 3, 32]
T	[-197, 4, 31]	[-205, 4, 32]
M	[-182, 5, 31]	[-190, 5, 32]
E	[-172, 6, 31]	[-180, 6, 32]
A	[-159, 6, 31]	[-167, 7, 32]
V	[-150, 8, 31]	[-158, 7, 32]
T	[-137, 9, 31]	[-145, 9, 32]
S	[-124, 10, 31]	[-132, 10, 32]
V	[-116, 10, 31]	[-123, 10, 32]
E	[-113, 11, 31]	[-119, 11, 32]
A	[-100, 12, 31]	[-108, 13, 32]
S	[-96, 13, 31]	[-103, 13, 32]
T	[-86, 15, 31]	[-94, 15, 32]
A	[-73, 15, 31]	[-81, 16, 32]
S	[-67, 17, 31]	[-75, 17, 32]
H	[-61, 17, 31]	[-69, 18, 32]
P	[-55, 18, 31]	[-63, 19, 32]
H	[-46, 20, 31]	[-54, 20, 32]
S	[-33, 21, 31]	[-41, 21, 32]
T	[-25, 21, 31]	[-33, 22, 32]
S	[-32, 22, 31]	[-28, 22, 32]
S	[-23, 23, 31]	[-31, 24, 32]
Y	[-19, 25, 31]	[-24, 24, 32]
F	[-14, 25, 31]	[-18, 25, 32]
A	[-1, 26, 31]	[-9, 27, 32]
T	[-5, 27, 31]	[-2, 27, 32]
T	[-8, 28, 31]	[-6, 28, 32]
Y	[-2, 30, 31]	[-9, 29, 32]
Y	[4, 30, 31]	[-3, 30, 32]
H	[10, 31, 31]	[2, 32, 32]
L	[3, 32, 31]	[15, 32, 32]
Y	[-5, 33, 31]	[7, 33, 33]

[35 rows x 34 columns]

In [8]:
```
arrow2 =[]
letter1 = []
letter2 = []
```

```
# Setting initial conditions, starting from the last element of the table
r =34
c =33
nextr = 34
nextc = 33
# Same as:
# Backward pass
if((c==nextc) and (r==nextr)):
    letter2.append('-')
    letter1.append(df2.index[r])
elif((c!=nextc) and (r==nextr)):
    letter1.append('-')
    letter2.append(df2.columns[c])
elif((c!=nextc) and (r!=nextr)):
    letter1.append(df2.index[r])
    letter2.append(df2.columns[c])

# Looping through the table to find the backward pass elements and their corrisponding
for r in range(34, 0,-1):
    for c in range(33,-1,-1):
        if r == nextr and c == nextc: # Similar implementation as in Backward pass, th
                                      # direction has been stored in an array direnctl
                                      # in a separate list
            arrow2.append(df2.iloc[r,c][0])
            nextr = df2.iloc[r,c][1]
            nextc = df2.iloc[r,c][2]
            if c == 33 and r==34:  # Handling starting condition
                letter2.append(df2.columns[c])
                letter1.append(df2.index[r-1])
            if c != 33 and r!=34:
                if(((c==nextc) and (r==nextr)) or (nextr==0 and nextc==0)):
                    letter2.append('-')
                    letter1.append(df2.index[r])
                elif((c!=nextc) and (r==nextr)):
                    letter1.append('-')
                    letter2.append(df2.columns[c])
                else:
                    letter1.append(df2.index[r])
                    letter2.append(df2.columns[c])

arrow2.append(0)
arrow2 = np.asarray(arrow2)
print(letter2[::-1])
print(letter1[::-1])
print(arrow2)
# -SALPQPTTPVSSFTSGSMLGRTDTALTNTYSAL-
# PSPTMEAVTSVEA-STASHPHSTSSYFATTYYHLY
```

```
['-', 'S', 'A', 'L', 'P', 'Q', 'P', 'T', 'T', 'P', 'V', 'S', 'S', 'F', 'T', 'S', 'G', 'S', 'M'
['P', 'S', 'P', 'T', 'M', 'E', 'A', 'V', 'T', 'S', 'V', 'E', 'A', '-', 'S', 'T', 'A', 'S', 'H'
[ 7. 15. 10. 12. 14.  6.  1.  1.  1.  0.  2.  0.  0. -5. -4. -2.  2.  3.
 -2. -2. -4. -6.  2.  1.  2. -3. -2. -7. -7. -6. -8. -5. -4. -3. -8.  0.]
```

1.1.2 2) Modify your program to implement the Smith-Waterman algorithm

Again run this on HEAGAWGHEE versus PAWHEAE

```
In [9]: m = 11
        n = 8

        # Creating a table to run HEAGAWGHEE versus PAWHEAE
        h = [' ', 'H', 'E', 'A', 'G', 'A', 'W','G', 'H', 'E', 'E']
        header2 = [' ','P', 'A', 'W', 'H', 'E', 'A','E']
        df3 = pd.DataFrame(data=[[np.zeros(3)]*m]*n)
        df3.rename(index = dict(zip(range(len(header2)),header2)),inplace =True)
        df3.rename(columns=dict(zip(range(len(h)),h)), inplace = True)

        # Filling the Global Allignment Table (array in row-colum pair)
        #Forward Pass

        cost = -8 # Cost of insertion or delition

        # Calculating price of substitution when arrows moves diagonally
        def blos(r, c):
            cname = df3.columns[c]
            rname = df3.index[r]
            if r == 0 or c == 0:
                return np.nan # Handling nan error when filling first row/column
            else:
                return int(df[rname][cname]) + int(df3.iloc[r-1,c-1][0])

        arrow =[]
        maxf = []

        # Loop to fill the table, deciding between if the upelement, leftelement or the diagel
        for r in range(0,8):
            for c in range(0,11):
                zero = 0
                upelement = int(df3.iloc[r-1,c][0]) + cost if r >= 1 else np.nan # Handling na
                leftelement = int(df3.iloc[r,c-1][0]) + cost if c >= 1 else np.nan # Handling
                diagelement = blos(r, c)
                if r == 0 and c == 0:
                    upelement = 0
                list1 = np.array([zero, diagelement, upelement,leftelement])
                re = np.nanargmax(list1)
```

```
            # Storing the location of the selected max element to find arrow direction dur
            if list1[re] == zero:   # Added condition of 0 with priority 1 between the opt
                vec = [0,0]
                vec2 = [r,c]  # Added vec2 to find the index of the max element in the tab
                arrow.extend([0,0, r,c])
            elif list1[re] == diagelement:
                vec = [r-1,c-1]
                vec2 = [r,c]
                arrow.extend([r-1,c-1, r,c])
            elif list1[re] == upelement:
                vec = [r-1,c]
                vec2 = [r,c]
                arrow.extend([r-1,c, r,c])
            elif list1[re] == leftelement:
                vec = [r,c-1]
                arrow.extend([r,c-1, r,c])
            # Storing value, row and column coordinates in an array for each row-column pa
            df3.iloc[r,c] = np.array([list1[re], vec[0], vec[1]])
            maxf.append([list1[re],vec[0], vec[1], vec2[0], vec2[1]]) # Array for later fi
                                                                      # table
    df3
```

Out[9]:

		H	E	A	\
	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
P	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
A	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[5, 1, 2]	
W	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
H	[0.0, 0.0, 0.0]	[10, 3, 0]	[2, 4, 1]	[0, 0, 0]	
E	[0.0, 0.0, 0.0]	[2, 4, 1]	[16, 4, 1]	[8, 5, 2]	
A	[0.0, 0.0, 0.0]	[0, 0, 0]	[8, 5, 2]	[21, 5, 2]	
E	[0.0, 0.0, 0.0]	[0, 0, 0]	[6, 6, 1]	[13, 6, 3]	

	G	A	W	G	\
	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
P	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
A	[0, 0, 0]	[5, 1, 4]	[0, 0, 0]	[0, 0, 0]	
W	[2, 2, 3]	[0, 0, 0]	[20, 2, 5]	[12, 3, 6]	
H	[0, 0, 0]	[0, 0, 0]	[12, 3, 6]	[18, 3, 6]	
E	[0, 0, 0]	[0, 0, 0]	[4, 4, 6]	[10, 4, 7]	
A	[13, 6, 3]	[5, 5, 4]	[0, 0, 0]	[4, 5, 6]	
E	[18, 6, 3]	[12, 6, 4]	[4, 7, 5]	[0, 0, 0]	

	H	E	E
	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]
P	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]
A	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]
W	[4, 3, 7]	[0, 0, 0]	[0, 0, 0]
H	[22, 3, 7]	[14, 4, 8]	[6, 4, 9]

```
E       [18, 4, 7]       [28, 4, 8]       [20, 4, 9]
A       [10, 5, 8]       [20, 5, 9]       [27, 5, 9]
E        [4, 6, 7]       [16, 6, 8]       [26, 6, 9]
```

In [10]:
```
arrow2 =[]
letter1 = []
letter2 = []

print("Maximum element in matrix, it's coordinates in the table, coordinates of where
is pointing from this position, index of max element in the array ="
      ,max([(v,i) for i,v in enumerate(maxf)]))

# Setting initial conditions, starting from the last element of the table
r =5
c =9
nextr = 4
nextc = 8
arrow2.append(df3.iloc[5,9][0])
# Same as:
# Backward pass
if((c==nextc) and (r==nextr)):
    letter2.append('-')
    letter1.append(df3.index[r])
elif((c!=nextc) and (r==nextr)):
    letter1.append('-')
    letter2.append(df3.columns[c])
elif((c!=nextc) and (r!=nextr)):
    letter1.append(df3.index[r])
    letter2.append(df3.columns[c])

# Looping through the table to find the backward pass elements and their corrisponding
for r in range(7, 1,-1):
    for c in range(10,-1,-1):
        if r == nextr and c == nextc: # Similar implementation as in Backward pass, t
                                      # direction has been stored in an array direnct
                                      # in a separate list
            arrow2.append(df3.iloc[r,c][0])
            nextr = df3.iloc[r,c][1]
            nextc = df3.iloc[r,c][2]
            if c == 9 and r==5:  # Handling starting condition
                letter.append(df3.columns[c])
                letter1.append(df3.index[r-1])
            if c != 9 and r!=5:
                if(((c==nextc) and (r==nextr))):
                    letter2.append('-')
                    letter1.append(df3.index[r])
                elif((c!=nextc) and (r==nextr)):
                    letter1.append('-')
```

```
                letter2.append(df3.columns[c])
            else:
                letter1.append(df3.index[r])
                letter2.append(df3.columns[c])

    arrow2.append(0)
    arrow2 = np.asarray(arrow2)
    print(letter2[::-1])
    print(letter1[::-1])
    print(arrow2)
```

```
Maximum element in matrix, it's coordinates in the table, coordinates of where the arrow is po
['A', 'W', 'G', 'H', 'E']
['A', 'W', '-', 'H', 'E']
[28 22 12 20  5  0]
```

Find the best local match between:

MQNSHSGVNQLGGVFVNGRPLPDSTRQKIVELAHSGARPCDISRILQVSNGCVSKILGRY and TDDECHSGVNQLGGVFVGGRPLPDSTRQKIVELAHSGARPCDISRI

In [11]:
```
# Creating a table to Match the protein sequence SALPQPTTPVSSFTSGSMLGRTDTALTNTYSAL wi

m = 61
n = 47

# adding '' and , automatically to string
# import re
# st = 'T D D E C H S G V N Q L G G V F V G G R P L P D S T R Q K I V E L A H S G A R
# print(re.findall(r"[\w']+", st))

# adding column and row headers
h = [' ', 'M', 'Q', 'N', 'S', 'H', 'S', 'G', 'V', 'N', 'Q', 'L', 'G', 'G', 'V', 'F',
     'D', 'S', 'T', 'R', 'Q', 'K', 'I', 'V', 'E', 'L', 'A', 'H', 'S', 'G', 'A', 'R',
     'L', 'Q', 'V', 'S', 'N', 'G', 'C', 'V', 'S', 'K', 'I', 'L', 'G', 'R', 'Y']
header2 = [' ','T', 'D', 'D', 'E', 'C', 'H', 'S', 'G', 'V', 'N', 'Q', 'L', 'G', 'G',
           'L', 'P', 'D', 'S', 'T', 'R', 'Q', 'K', 'I', 'V', 'E', 'L', 'A', 'H', 'S',
           'S', 'R', 'I']
df4 = pd.DataFrame(data=[[np.zeros(1)]*m]*n)
df4.rename(index = dict(zip(range(len(header2)),header2)),inplace =True)
df4.rename(columns=dict(zip(range(len(h)),h)), inplace = True)

df4
```

Out[11]:

	M	Q	N	S	H	S	G	V	N	\
[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	
T	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]

D	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
D	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
E	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
C	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
H	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
S	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
G	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
V	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
N	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
Q	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
L	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
G	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
G	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
V	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
F	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
V	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
G	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
G	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
R	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
P	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
L	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
P	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
D	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
S	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
T	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
R	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
Q	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
K	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
I	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
V	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
E	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
L	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
A	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
H	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
S	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
G	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
A	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
R	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
P	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
C	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
D	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
I	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
S	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
R	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
I	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]

...	G	C	V	S	K	I	L	G	R	Y
...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]

T	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
D	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
D	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
E	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
C	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
H	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
S	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
G	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
V	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
N	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
Q	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
L	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
G	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
G	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
V	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
F	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
V	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
G	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
G	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
R	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
P	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
L	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
P	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
D	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
S	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
T	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
R	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
Q	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
K	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
I	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
V	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
E	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
L	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
A	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
H	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
S	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
G	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
A	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
R	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
P	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
C	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
D	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
I	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
S	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
R	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]
I	...	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]	[0.0]

[47 rows x 61 columns]

In [12]:
```
# Filling the Global Allignment Table (array in row-colum pair)
#Forward Pass

cost = -8 # Cost of insertion or delition

# Calculating price of substitution when arrows moves diagonally
def blos(r, c):
    cname = df4.columns[c]
    rname = df4.index[r]
    if r == 0 or c == 0:
        return np.nan # Handling nan error when filling first row/column
    else:
        return int(df[rname][cname]) + int(df4.iloc[r-1,c-1][0])

arrow =[]
maxf = []

# Loop to fill the table, deciding between if the upelement, leftelement or the diage
for r in range(0,47):
    for c in range(0,61):
        zero = 0
        upelement = int(df4.iloc[r-1,c][0]) + cost if r >= 1 else np.nan # Handling n
        leftelement = int(df4.iloc[r,c-1][0]) + cost if c >= 1 else np.nan # Handling
        diagelement = blos(r, c)
        if r == 0 and c == 0:
            upelement = 0
        list1 = np.array([zero, diagelement, upelement,leftelement])
        re = np.nanargmax(list1)
        # Storing the location of the selected max element to find arrow direction du
        if list1[re] == zero:  # Added condition of 0 with priority 1 between the opt
            vec = [0,0] # Added vec2 to find the index of the max element in the tabl
            vec2 = [r,c]
            arrow.extend([0,0, r,c])
        elif list1[re] == diagelement:
            vec = [r-1,c-1]
            vec2 = [r,c]
            arrow.extend([r-1,c-1, r,c])
        elif list1[re] == upelement:
            vec = [r-1,c]
            vec2 = [r,c]
            arrow.extend([r-1,c, r,c])
        elif list1[re] == leftelement:
            vec = [r,c-1]
            arrow.extend([r,c-1, r,c])
        # Storing value, row and column coordinates in an array for each row-column p
        df4.iloc[r,c] = np.array([list1[re], vec[0], vec[1]])
        maxf.append([list1[re],vec[0], vec[1], vec2[0], vec2[1]]) # Array for later f
                                                                  # table
```

```
df4
```

Out[12]:

		M	Q	N	\
	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
T	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
D	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[2, 1, 2]	
D	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[2, 2, 2]	
E	[0.0, 0.0, 0.0]	[0, 0, 0]	[2, 3, 1]	[0, 0, 0]	
C	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
H	[0.0, 0.0, 0.0]	[0, 0, 0]	[1, 5, 1]	[1, 5, 2]	
S	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[2, 6, 2]	
G	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
V	[0.0, 0.0, 0.0]	[1, 8, 0]	[0, 0, 0]	[0, 0, 0]	
N	[0.0, 0.0, 0.0]	[0, 0, 0]	[1, 9, 1]	[7, 9, 2]	
Q	[0.0, 0.0, 0.0]	[0, 0, 0]	[7, 10, 1]	[1, 10, 2]	
L	[0.0, 0.0, 0.0]	[3, 11, 0]	[0, 0, 0]	[3, 11, 2]	
G	[0.0, 0.0, 0.0]	[0, 0, 0]	[1, 12, 1]	[0, 0, 0]	
G	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[1, 13, 2]	
V	[0.0, 0.0, 0.0]	[1, 14, 0]	[0, 0, 0]	[0, 0, 0]	
F	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
V	[0.0, 0.0, 0.0]	[1, 16, 0]	[0, 0, 0]	[0, 0, 0]	
G	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
G	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
R	[0.0, 0.0, 0.0]	[0, 0, 0]	[1, 19, 1]	[0, 0, 0]	
P	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
L	[0.0, 0.0, 0.0]	[3, 21, 0]	[0, 0, 0]	[0, 0, 0]	
P	[0.0, 0.0, 0.0]	[0, 0, 0]	[2, 22, 1]	[0, 0, 0]	
D	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[4, 23, 2]	
S	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[1, 24, 2]	
T	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
R	[0.0, 0.0, 0.0]	[0, 0, 0]	[1, 26, 1]	[0, 0, 0]	
Q	[0.0, 0.0, 0.0]	[0, 0, 0]	[7, 27, 1]	[1, 27, 2]	
K	[0.0, 0.0, 0.0]	[0, 0, 0]	[2, 28, 1]	[7, 28, 2]	
I	[0.0, 0.0, 0.0]	[2, 29, 0]	[0, 0, 0]	[0, 0, 0]	
V	[0.0, 0.0, 0.0]	[1, 30, 0]	[0, 0, 0]	[0, 0, 0]	
E	[0.0, 0.0, 0.0]	[0, 0, 0]	[3, 31, 1]	[0, 0, 0]	
L	[0.0, 0.0, 0.0]	[3, 32, 0]	[0, 0, 0]	[0, 0, 0]	
A	[0.0, 0.0, 0.0]	[0, 0, 0]	[2, 33, 1]	[0, 0, 0]	
H	[0.0, 0.0, 0.0]	[0, 0, 0]	[1, 34, 1]	[3, 34, 2]	
S	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[2, 35, 2]	
G	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
A	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
R	[0.0, 0.0, 0.0]	[0, 0, 0]	[1, 38, 1]	[0, 0, 0]	
P	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
C	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
D	[0.0, 0.0, 0.0]	[0, 0, 0]	[0, 0, 0]	[2, 41, 2]	
I	[0.0, 0.0, 0.0]	[2, 42, 0]	[0, 0, 0]	[0, 0, 0]	

```
S  [0.0, 0.0, 0.0]        [0, 0, 0]       [2, 43, 1]       [1, 43, 2]
R  [0.0, 0.0, 0.0]        [0, 0, 0]       [1, 44, 1]       [1, 44, 2]
I  [0.0, 0.0, 0.0]       [2, 45, 0]        [0, 0, 0]        [0, 0, 0]

                  S                 H                 S                 G  \
   [0.0, 0.0, 0.0]  [0.0, 0.0, 0.0]  [0.0, 0.0, 0.0]  [0.0, 0.0, 0.0]
T        [2, 0, 3]        [0, 0, 0]        [2, 0, 5]        [0, 0, 0]
D        [0, 0, 0]        [1, 1, 4]        [0, 0, 0]        [1, 1, 6]
D        [2, 2, 3]        [0, 0, 0]        [1, 2, 5]        [0, 0, 0]
E        [1, 3, 3]        [2, 3, 4]        [0, 0, 0]        [0, 0, 0]
C        [0, 0, 0]        [0, 0, 0]        [1, 4, 5]        [0, 0, 0]
H        [0, 0, 0]       [10, 5, 4]        [2, 6, 5]        [0, 0, 0]
S        [6, 6, 3]        [2, 6, 5]       [15, 6, 5]        [7, 7, 6]
G        [2, 7, 3]        [4, 7, 4]        [7, 7, 6]       [23, 7, 6]
V        [0, 0, 0]        [0, 0, 0]        [2, 8, 5]       [15, 8, 7]
N        [1, 9, 3]        [1, 9, 4]        [1, 9, 5]        [7, 9, 7]
Q       [7, 10, 3]       [2, 10, 4]       [1, 10, 5]        [0, 0, 0]
L        [0, 0, 0]       [4, 11, 4]        [0, 0, 0]        [0, 0, 0]
G       [3, 12, 3]        [0, 0, 0]       [4, 12, 5]       [8, 12, 6]
G        [0, 0, 0]       [1, 13, 4]        [0, 0, 0]      [12, 13, 6]
V        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]       [4, 14, 7]
F        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
V        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
G        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]       [8, 17, 6]
G        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]       [8, 18, 6]
R        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
P        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
L        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
P        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
D        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
S       [9, 24, 3]       [1, 25, 4]       [5, 24, 5]        [0, 0, 0]
T       [3, 25, 3]       [7, 25, 4]       [3, 25, 5]       [3, 25, 6]
R        [0, 0, 0]       [3, 26, 4]       [6, 26, 5]        [0, 0, 0]
Q        [0, 0, 0]       [1, 27, 4]       [3, 27, 5]       [4, 27, 6]
K       [1, 28, 3]        [0, 0, 0]       [1, 28, 5]       [1, 28, 6]
I       [4, 29, 3]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
V        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
E        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
L        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
A       [1, 33, 3]        [0, 0, 0]       [1, 33, 5]        [0, 0, 0]
H        [0, 0, 0]      [11, 34, 4]       [3, 35, 5]        [0, 0, 0]
S       [8, 35, 3]       [3, 35, 5]      [16, 35, 5]       [8, 36, 6]
G       [2, 36, 3]       [6, 36, 4]       [8, 36, 6]      [24, 36, 6]
A       [1, 37, 3]        [0, 0, 0]       [7, 37, 5]      [16, 37, 7]
R        [0, 0, 0]       [1, 38, 4]        [0, 0, 0]       [8, 38, 7]
P        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
C        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
D        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]        [0, 0, 0]
```

I	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]
S	[5, 43, 3]	[0, 0, 0]	[5, 43, 5]	[0, 0, 0]
R	[0, 0, 0]	[5, 44, 4]	[0, 0, 0]	[2, 44, 6]
I	[0, 0, 0]	[0, 0, 0]	[2, 45, 5]	[0, 0, 0]

	V	N	...	G	\
	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	...	[0.0, 0.0, 0.0]	
T	[0, 0, 0]	[0, 0, 0]	...	[0, 0, 0]	
D	[0, 0, 0]	[2, 1, 8]	...	[0, 0, 0]	
D	[0, 0, 0]	[2, 2, 8]	...	[3, 2, 50]	
E	[0, 0, 0]	[0, 0, 0]	...	[0, 0, 0]	
C	[0, 0, 0]	[0, 0, 0]	...	[0, 0, 0]	
H	[0, 0, 0]	[1, 5, 8]	...	[0, 0, 0]	
S	[0, 0, 0]	[1, 6, 8]	...	[1, 6, 50]	
G	[15, 8, 7]	[7, 8, 8]	...	[9, 7, 50]	
V	[28, 8, 7]	[20, 9, 8]	...	[1, 8, 50]	
N	[20, 9, 8]	[35, 9, 8]	...	[0, 0, 0]	
Q	[12, 10, 8]	[27, 10, 9]	...	[5, 10, 50]	
L	[4, 11, 8]	[19, 11, 9]	...	[2, 11, 50]	
G	[0, 0, 0]	[11, 12, 9]	...	[8, 12, 50]	
G	[4, 13, 7]	[3, 13, 9]	...	[14, 13, 50]	
V	[17, 14, 7]	[9, 15, 8]	...	[10, 14, 50]	
F	[9, 15, 8]	[13, 15, 8]	...	[2, 15, 50]	
V	[5, 16, 7]	[6, 16, 8]	...	[0, 0, 0]	
G	[0, 0, 0]	[5, 17, 8]	...	[9, 17, 50]	
G	[4, 18, 7]	[0, 0, 0]	...	[8, 18, 50]	
R	[5, 19, 7]	[3, 19, 8]	...	[4, 19, 50]	
P	[0, 0, 0]	[3, 20, 8]	...	[0, 0, 0]	
L	[1, 21, 7]	[0, 0, 0]	...	[0, 0, 0]	
P	[0, 0, 0]	[0, 0, 0]	...	[0, 0, 0]	
D	[0, 0, 0]	[2, 23, 8]	...	[0, 0, 0]	
S	[0, 0, 0]	[1, 24, 8]	...	[2, 24, 50]	
T	[0, 0, 0]	[0, 0, 0]	...	[1, 26, 50]	
R	[0, 0, 0]	[0, 0, 0]	...	[6, 26, 50]	
Q	[0, 0, 0]	[0, 0, 0]	...	[0, 0, 0]	
K	[1, 28, 7]	[0, 0, 0]	...	[0, 0, 0]	
I	[5, 29, 7]	[0, 0, 0]	...	[1, 30, 50]	
V	[5, 30, 7]	[2, 30, 8]	...	[7, 31, 50]	
E	[0, 0, 0]	[5, 31, 8]	...	[15, 32, 50]	
L	[1, 32, 7]	[0, 0, 0]	...	[28, 33, 50]	
A	[0, 0, 0]	[0, 0, 0]	...	[41, 34, 50]	
H	[0, 0, 0]	[1, 34, 8]	...	[59, 35, 50]	
S	[0, 0, 0]	[1, 35, 8]	...	[72, 36, 50]	
G	[16, 37, 7]	[8, 37, 8]	...	[88, 36, 50]	
A	[24, 37, 7]	[16, 38, 8]	...	[101, 38, 50]	
R	[16, 38, 8]	[23, 38, 8]	...	[116, 39, 50]	
P	[8, 39, 8]	[15, 39, 9]	...	[134, 40, 50]	
C	[0, 0, 0]	[7, 40, 9]	...	[155, 41, 50]	

D	[0, 0, 0]	[2, 41, 8]	...	[171, 42, 50]
I	[4, 42, 7]	[0, 0, 0]	...	[184, 43, 50]
S	[0, 0, 0]	[5, 43, 8]	...	[197, 44, 50]
R	[0, 0, 0]	[0, 0, 0]	...	[212, 45, 50]
I	[6, 45, 7]	[0, 0, 0]	...	[225, 46, 50]

	C	V	S	K	\
	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	
T	[0, 0, 0]	[0, 0, 0]	[2, 0, 53]	[0, 0, 0]	
D	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[1, 1, 54]	
D	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
E	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[1, 3, 54]	
C	[13, 4, 51]	[5, 5, 52]	[0, 0, 0]	[0, 0, 0]	
H	[5, 5, 52]	[9, 5, 52]	[4, 5, 53]	[0, 0, 0]	
S	[0, 0, 0]	[3, 6, 52]	[14, 6, 53]	[6, 7, 54]	
G	[1, 8, 51]	[0, 0, 0]	[6, 7, 54]	[12, 7, 54]	
V	[8, 8, 51]	[6, 8, 52]	[0, 0, 0]	[4, 8, 55]	
N	[0, 0, 0]	[5, 9, 52]	[7, 9, 53]	[0, 0, 0]	
Q	[0, 0, 0]	[0, 0, 0]	[5, 10, 53]	[9, 10, 54]	
L	[3, 11, 51]	[1, 11, 52]	[0, 0, 0]	[2, 11, 54]	
G	[0, 0, 0]	[0, 0, 0]	[1, 12, 53]	[0, 0, 0]	
G	[6, 14, 51]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
V	[13, 14, 51]	[11, 14, 52]	[3, 15, 53]	[0, 0, 0]	
F	[8, 15, 51]	[12, 15, 52]	[8, 15, 53]	[0, 0, 0]	
V	[1, 16, 51]	[13, 16, 52]	[10, 16, 53]	[5, 16, 54]	
G	[1, 18, 51]	[5, 17, 53]	[13, 17, 53]	[8, 17, 54]	
G	[6, 18, 51]	[0, 0, 0]	[5, 18, 53]	[11, 18, 54]	
R	[4, 19, 51]	[3, 19, 52]	[0, 0, 0]	[8, 19, 54]	
P	[0, 0, 0]	[1, 20, 52]	[2, 20, 53]	[0, 0, 0]	
L	[0, 0, 0]	[1, 21, 52]	[0, 0, 0]	[0, 0, 0]	
P	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
D	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	
S	[0, 0, 0]	[0, 0, 0]	[5, 24, 53]	[0, 0, 0]	
T	[1, 25, 51]	[0, 0, 0]	[2, 25, 53]	[4, 25, 54]	
R	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[5, 26, 54]	
Q	[3, 27, 51]	[0, 0, 0]	[0, 0, 0]	[2, 27, 54]	
K	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[6, 28, 54]	
I	[0, 0, 0]	[4, 29, 52]	[0, 0, 0]	[0, 0, 0]	
V	[0, 0, 0]	[5, 30, 52]	[2, 30, 53]	[0, 0, 0]	
E	[7, 32, 51]	[0, 0, 0]	[4, 31, 53]	[3, 31, 54]	
L	[20, 33, 51]	[12, 33, 52]	[4, 33, 53]	[1, 32, 54]	
A	[33, 34, 51]	[25, 34, 52]	[17, 34, 53]	[9, 34, 54]	
H	[51, 35, 51]	[43, 35, 52]	[35, 35, 53]	[27, 35, 54]	
S	[64, 36, 51]	[56, 36, 52]	[48, 35, 53]	[40, 36, 54]	
G	[80, 37, 51]	[72, 37, 52]	[64, 37, 53]	[56, 37, 54]	
A	[93, 38, 51]	[85, 38, 52]	[77, 38, 53]	[69, 38, 54]	
R	[108, 39, 51]	[100, 39, 52]	[92, 39, 53]	[84, 39, 54]	
P	[126, 40, 51]	[118, 40, 52]	[110, 40, 53]	[102, 40, 54]	

C	[147, 40, 51]	[139, 41, 52]	[131, 41, 53]	[123, 41, 54]
D	[163, 42, 51]	[155, 42, 52]	[147, 42, 53]	[139, 42, 54]
I	[176, 43, 51]	[168, 43, 52]	[160, 43, 53]	[152, 43, 54]
S	[189, 44, 51]	[181, 44, 52]	[173, 43, 53]	[165, 44, 54]
R	[204, 45, 51]	[196, 45, 52]	[188, 45, 53]	[180, 45, 54]
I	[217, 46, 51]	[209, 46, 52]	[201, 46, 53]	[193, 46, 54]

	I	L	G	R \
	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]	[0.0, 0.0, 0.0]
T	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]
D	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]
D	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]
E	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]
C	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]
H	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]
S	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]
G	[4, 8, 55]	[0, 0, 0]	[8, 7, 57]	[0, 0, 0]
V	[16, 8, 55]	[8, 9, 56]	[0, 0, 0]	[5, 8, 58]
N	[8, 9, 56]	[12, 9, 56]	[8, 9, 57]	[0, 0, 0]
Q	[1, 11, 55]	[6, 10, 56]	[10, 10, 57]	[9, 10, 58]
L	[11, 11, 55]	[6, 11, 56]	[2, 11, 57]	[7, 11, 58]
G	[3, 12, 56]	[7, 12, 56]	[14, 12, 57]	[6, 13, 58]
G	[0, 0, 0]	[0, 0, 0]	[15, 13, 57]	[11, 13, 58]
V	[4, 14, 55]	[1, 14, 56]	[7, 14, 58]	[12, 14, 58]
F	[0, 0, 0]	[5, 15, 56]	[0, 0, 0]	[4, 15, 58]
V	[4, 16, 55]	[1, 16, 56]	[1, 16, 57]	[0, 0, 0]
G	[1, 17, 55]	[0, 0, 0]	[9, 17, 57]	[1, 18, 58]
G	[4, 18, 55]	[0, 0, 0]	[8, 18, 57]	[6, 18, 58]
R	[7, 19, 55]	[1, 19, 56]	[0, 0, 0]	[15, 19, 58]
P	[5, 20, 55]	[3, 20, 56]	[0, 0, 0]	[7, 20, 59]
L	[2, 21, 55]	[10, 21, 56]	[2, 22, 57]	[0, 0, 0]
P	[0, 0, 0]	[2, 22, 57]	[8, 22, 57]	[0, 0, 0]
D	[0, 0, 0]	[0, 0, 0]	[1, 23, 57]	[6, 23, 58]
S	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]
T	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]
R	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[7, 26, 58]
Q	[2, 27, 55]	[0, 0, 0]	[0, 0, 0]	[1, 27, 58]
K	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	[3, 28, 58]
I	[11, 29, 55]	[3, 30, 56]	[0, 0, 0]	[0, 0, 0]
V	[4, 30, 55]	[12, 30, 56]	[4, 31, 57]	[0, 0, 0]
E	[0, 0, 0]	[4, 31, 57]	[9, 31, 57]	[4, 31, 58]
L	[5, 32, 55]	[5, 32, 56]	[1, 32, 58]	[6, 32, 58]
A	[1, 34, 55]	[3, 33, 56]	[5, 33, 57]	[0, 0, 0]
H	[19, 35, 55]	[11, 35, 56]	[3, 35, 57]	[5, 34, 58]
S	[32, 36, 55]	[24, 36, 56]	[16, 36, 57]	[8, 36, 58]
G	[48, 37, 55]	[40, 37, 56]	[32, 36, 57]	[24, 37, 58]
A	[61, 38, 55]	[53, 38, 56]	[45, 38, 57]	[37, 38, 58]
R	[76, 39, 55]	[68, 39, 56]	[60, 39, 57]	[52, 38, 58]

```
P     [94, 40, 55]     [86, 40, 56]     [78, 40, 57]     [70, 40, 58]
C    [115, 41, 55]    [107, 41, 56]     [99, 41, 57]     [91, 41, 58]
D    [131, 42, 55]    [123, 42, 56]    [115, 42, 57]    [107, 42, 58]
I    [144, 42, 55]    [136, 43, 56]    [128, 43, 57]    [120, 43, 58]
S    [157, 44, 55]    [149, 44, 56]    [141, 44, 57]    [133, 44, 58]
R    [172, 45, 55]    [164, 45, 56]    [156, 45, 57]    [148, 44, 58]
I    [185, 45, 55]    [177, 46, 56]    [169, 46, 57]    [161, 46, 58]
```

```
                Y
   [0.0, 0.0, 0.0]
T        [0, 0, 0]
D        [0, 0, 0]
D        [0, 0, 0]
E        [0, 0, 0]
C        [0, 0, 0]
H       [2, 5, 59]
S        [0, 0, 0]
G        [0, 0, 0]
V        [0, 0, 0]
N       [3, 9, 59]
Q      [1, 11, 59]
L      [8, 11, 59]
G      [4, 12, 59]
G      [3, 13, 59]
V     [10, 14, 59]
F     [16, 15, 59]
V      [8, 16, 60]
G        [0, 0, 0]
G        [0, 0, 0]
R      [7, 20, 59]
P     [12, 20, 59]
L      [6, 21, 59]
P        [0, 0, 0]
D        [0, 0, 0]
S      [4, 24, 59]
T        [0, 0, 0]
R        [0, 0, 0]
Q      [6, 27, 59]
K        [0, 0, 0]
I      [2, 29, 59]
V        [0, 0, 0]
E        [0, 0, 0]
L      [3, 32, 59]
A      [4, 33, 59]
H      [2, 34, 59]
S      [3, 35, 59]
G     [16, 37, 59]
A     [29, 38, 59]
```

```
R     [44, 39, 59]
P     [62, 40, 59]
C     [83, 41, 59]
D     [99, 42, 59]
I    [112, 43, 59]
S    [125, 44, 59]
R    [140, 45, 59]
I    [153, 46, 59]

[47 rows x 61 columns]
```

In [13]:
```
arrow2 =[]
letter1 = []
letter2 = []

print("Maximum element in matrix, it's coordinates in the table, coordinates of where
is pointing from this position, index of max element in the array ="
      ,max([(v,i) for i,v in enumerate(maxf)]))

# Setting initial conditions, starting from the last element of the table
r =46
c =45
nextr = 45
nextc = 44
#arrow2.append(df4.iloc[r,c][0])
# Same as:
# Backward pass
if((c==nextc) and (r==nextr)):
    letter2.append('-')
    letter1.append(df4.index[r])
elif((c!=nextc) and (r==nextr)):
    letter1.append('-')
    letter2.append(df4.columns[c])
elif((c!=nextc) and (r!=nextr)):
    letter1.append(df4.index[r])
    letter2.append(df4.columns[c])

# Looping through the table to find the backward pass elements and their corrisponding
for r in range(46, 0,-1):
    for c in range(60,4,-1): # Skipping the first 4 elements because we already analy
        if r == nextr and c == nextc: # Similar implementation as in Backward pass, t
                                      # direction has been stored in an array direnct
                                      # in a separate list
            arrow2.append(df4.iloc[r,c][0])
            nextr = df4.iloc[r,c][1]
            nextc = df4.iloc[r,c][2]
            if c == 45 and r==46:  # Handling starting condition
                letter.append(df4.columns[c])
```

```
                    letter1.append(df4.index[r-1])
                if c != 45 and r!=46:
                    if(((c==nextc) and (r==nextr))or (nextr==-1 and nextc==0)):
                        letter2.append('-')
                        letter1.append(df4.index[r])
                    elif((c!=nextc) and (r==nextr)):
                        letter1.append('-')
                        letter2.append(df4.columns[c])
                    else:
                        letter1.append(df4.index[r])
                        letter2.append(df4.columns[c])

        arrow2.append(0)
        arrow2 = np.asarray(arrow2)
        print(letter2[::-1])
        print(letter1[::-1])
        print(arrow2)
        # HSGVNQLGGVFVNGRPLPDSTRQKIVELAHSGARPCDISRI
        # HSGVNQLGGVFVGGRPLPDSTRQKIVELAHSGARPCDISRI
```

```
Maximum element in matrix, it's coordinates in the table, coordinates of where the arrow is po
['H', 'S', 'G', 'V', 'N', 'Q', 'L', 'G', 'G', 'V', 'F', 'V', 'N', 'G', 'R', 'P', 'L', 'P', 'D'
['H', 'S', 'G', 'V', 'N', 'Q', 'L', 'G', 'G', 'V', 'F', 'V', 'G', 'G', 'R', 'P', 'L', 'P', 'D'
[268 261 256 251 243 230 220 213 208 200 195 185 180 175 169 164 159 153
 146 139 134 129 121 111 106  96  89  81  81  76  68  63  55  47  42  35
  28  23  15  10   0]
```

1.1.3 3) BLAST algorithm test

```
In [11]: #Pax6 protein for the mouse

        # MQNSHSGVNQLGGVFVNGRPLPDSTRQKIVELAHSGARPCDISRILQVSNGCVSKILGRY
        # YETGSIRPRAIGGSKPRVATPEVVSKIAQYKRECPSIFAWEIRDRLLSEGVCTNDNIPSV
        # SSINRVLRNLASEKQQMGADGMYDKLRMLNGQTGSWGTRPGWYPGTSVPGQPTQDGCQQQ
        # EGGGENTNSISSNGEDSDEAQMRLQLKRKLQRNRTSFTQEQIEALEKEFERTHYPDVFAR
        # ERLAAKIDLPEARIQVWFSNRRAKWRREEKLRNQRRQASNTPSHIPISSSFSTSVYQPIP
        # QPTTPVSSFTSGSMLGRTDTALTNTYSALPPMPSFTMANNLPMQPPVPSQTSSYSCMLPT
        # SPSVNGRSYDTYTPPHMQTHMNSQPMGTSGTTSTGLISPGVSVPVQVPGSEPDMSQYWPR
        # LQ

        # eyeless protein for the fruit fly

        # NVIAMRNLPCLGTAGGSGLGGIAGKPSPTMEAVEASTASHPHSTSSYFATTYYHLTDDEC
        # HSGVNQLGGVFVGGRPLPDSTRQKIVELAHSGARPCDISRI

        # Comparison result: https://blast.ncbi.nlm.nih.gov/Blast.cgi
```

```
         df5=pd.read_csv('6P13GW4F114-Alignment-HitTable.csv')
         df5.head()
```

Out[11]:

	Query_124327	Query_124329	97.561	41	1	0	5	45	61	101	1.72e-26	\
0	Query_124327	Query_124329	26.667	30	22	0	300	329	26	55	2.1	
1	Query_124327	Query_124329	40.000	20	11	1	279	297	35	54	2.7	
2	Query_124327	Query_124329	31.250	16	11	0	361	376	27	42	5.2	

	88.2	97.56
0	13.9	36.67
1	13.5	65.00
2	12.7	50.00

```
In [12]: f = open('6P13GW4F114-Alignment.txt', 'r')
         file_contents = f.read()
         print (file_contents)
         f.close()
```

```
BLASTP 2.8.1+
Reference: Stephen F. Altschul, Thomas L. Madden, Alejandro
A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and
David J. Lipman (1997), "Gapped BLAST and PSI-BLAST: a new
generation of protein database search programs", Nucleic
Acids Res. 25:3389-3402.


Reference for compositional score matrix adjustment: Stephen
F. Altschul, John C. Wootton, E. Michael Gertz, Richa
Agarwala, Aleksandr Morgulis, Alejandro A. Schaffer, and
Yi-Kuo Yu (2005) "Protein database searches using
compositionally adjusted substitution matrices", FEBS J.
272:5101-5109.


RID: 6P13GW4F114


Query=
Length=422


                                                                   Score        E     Max
Sequences producing significant alignments:                       (Bits)     Value  ident

Query_124329 unnamed protein product                               88.2       2e-26  98%

ALIGNMENTS
>Query_124329 unnamed protein product
```

```
Length=101

 Score = 88.2 bits (217),  Expect = 2e-26, Method: Compositional matrix adjust.
 Identities = 40/41 (98%), Positives = 40/41 (98%), Gaps = 0/41 (0%)

Query  5    HSGVNQLGGVFVNGRPLPDSTRQKIVELAHSGARPCDISRI  45
            HSGVNQLGGVFV GRPLPDSTRQKIVELAHSGARPCDISRI
Sbjct  61   HSGVNQLGGVFVGGRPLPDSTRQKIVELAHSGARPCDISRI  101


 Score = 13.9 bits (24),  Expect = 2.1, Method: Compositional matrix adjust.
 Identities = 8/30 (27%), Positives = 11/30 (37%), Gaps = 0/30 (0%)

Query  300  PQPTTPVSSFTSGSMLGRTDTALTNTYSAL  329
            P PT      ++ S    T +    TY  L
Sbjct  26   PSPTMEAVEASTASHPHSTSSYFATTYYHL  55


 Score = 13.5 bits (23),  Expect = 2.7, Method: Compositional matrix adjust.
 Identities = 8/20 (40%), Positives = 13/20 (65%), Gaps = 1/20 (5%)

Query  279  SNTPSHIPISSS-FSTSVYQ  297
            ++T SH   +SS F+T+ Y
Sbjct  35   ASTASHPHSTSSYFATTYYH  54


 Score = 12.7 bits (21),  Expect = 5.2, Method: Compositional matrix adjust.
 Identities = 5/16 (31%), Positives = 8/16 (50%), Gaps = 0/16 (0%)

Query  361  SPSVNGRSYDTYTPPH  376
            SP++      T + PH
Sbjct  27   SPTMEAVEASTASHPH  42
```

1.1.4 4) HMM for detecting CG regions

```
In [167]: import random

          def roll(sides, bias_list):
              assert len(bias_list) == sides
              number = random.uniform(0, sum(bias_list))
              current = 0
              for i, bias in enumerate(bias_list):
```

```
        current += bias
        if number <= current:
            return i + 1

prob1 = [0.2698, 0.3237, 0.2080, 0.1985]
stayleave1 = [0.9998, 0.0002]
# stayleave1 = [0.7, 0.3]
prob2 = [0.2459, 0.2079, 0.2478, 0.2984]
stayleave2 = [0.9997, 0.0003]
# stayleave2 = [0.4, 6]

initialstate = [0.5,0.5]

selectstate = roll(2, initialstate)
if selectstate == 1:
    print('Starting in state 1 \n')
    state1=1
    state2=0
else:
    print("Starting in state 2 \n")
    state1=0
    state2=1

numlist = []
letlist = []

for i in range(0,500):
    if state1==1:
        numlist.append(roll(4, prob1))
        letlist.append('AT')
        switchstate = roll(2, stayleave1)
        if switchstate == 1:
            state1=1
            state2=0
        else:
            print("Switching to state 2 \n")
            state1=0
            state2=1
    if state2==1:
        numlist.append(roll(4, prob2))
        letlist.append('CG')
        switchstate = roll(2, stayleave2)
        if switchstate == 1:
            state1=0
            state2=1
        else:
            print("Switching to state 1 \n")
            state1=1
```

```
                state2=0
        # print(len(prob1))

        letlist = ''.join(letlist)
        numlist = int(''.join(str(i) for i in numlist))

        print(letlist)
        print(numlist)
```

```
Starting in state 2

Switching to state 1

Switching to state 2

CGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCG
21241321114142324441111234132432444231433411424233221213114414143343231424114423422444211424322
```