MARMARA UNIVERSITY – FACULTY OF ENGINEERING

CSE 4065 ASSIGNMENT #2 REPORT

Name, Surname: Hasan Şenyurt – Melisa Durmuş – Mustafa Kerem Ekinci

Student ID: 150120531 – 150119727 – 150119695

Department: Computer Engineering

IMPLEMENTATION DETAILS

Node Class:

We have node class for creating the 2D grid. Node class has some variables and two functions. Variables are score (node's score), match score, penalty scores, coordinates and Boolean variable which is for gap opening property.

One of the function is __init__ function to define variables. Also, values of initial node of the grid is defined in this function.

```
1  class Node:
2
3      #VALUES
4      score = 0
5      match = 3
6      gapOpeningPenalty = -1
7      gapExtensionPenalty = -0.5
8      misMatchPenalty = -1
9      isOpeningGap = False
10
11     def __init__(self, xCor, yCor, nucleotid):
12         self.xCor = xCor
13         self.yCor = yCor
14         self.nucleotid = nucleotid
15         self.topEdge = 0
16         self.diagonalEdge = 0
17         self.leftEdge = 0
18
19         #Initial nucleotid values.
20         if (self.xCor == 0 and self.yCor == 0) or self.xCor!=0 and self.yCor!=0:
21             self.nucleotid = "X"
22
23         else:
24             self.nucleotid = nucleotid
```

Figure-1: Node Class

Other function is called ‘calculate_score’. This function is written to calculate node’s score by looking left, top and diagonal nodes. It compares scores which comes from left, left-diagonal and top nodes and their edges, then takes maximum value among them. That value becomes score of node that called the function.

```
#Function for score calculation of each node by choosing maximum score which comes from top, diagonal and left nodes, their edges.
    def calculate_score(self,grid):

        #Score of inital node is 0
        if self.xCor == 0 and self.yCor == 0:
            self.score = 0

        #Score of first row is increasing -1 by -1.
        elif self.xCor == 0:
            top_x = self.xCor
            top_y = self.yCor - 1
            self.score = grid[top_x][top_y].score + self.gapOpeningPenalty
            self.topEdge = self.gapOpeningPenalty

        #Score of first column is increasing -1 by -1.
        elif self.yCor == 0:
            left_x = self.xCor -1
            left_y = self.yCor
            self.score = grid[left_x][left_y].score + self.gapOpeningPenalty
            self.leftEdge = self.gapOpeningPenalty

        #Calculating scores of nodes except initial node, first row and column.
        else:
            #Getting coordinates of left, top and diagonal nodes.
            left_x = self.xCor - 1; left_y = self.yCor
            top_x = self.xCor; top_y = self.yCor - 1
            diagonal_x = self.xCor - 1; diagonal_y = self.yCor - 1

            max_left_x = 0; max_left_y = self.yCor
            max_x = self.xCor; max_y = 0

            temp_score=[]

            #CHECKING MATCH!
            if(grid[max_left_x][max_left_y].nucleotid==grid[max_x][max_y].nucleotid):
                self.diagonalEdge = self.match
                temp_score.append(grid[diagonal_x][diagonal_y].score+ self.match)

            #CHECKING MISMATCH!
            else:
                temp_score.append(grid[diagonal_x][diagonal_y].score + self.misMatchPenalty)
                self.diagonalEdge = self.misMatchPenalty

            #assigning penalties to the edges which connects nodes.
            self.topEdge = self.gapOpeningPenalty
            self.leftEdge = self.gapOpeningPenalty

            #Selecting maximum score to put the score value according to dynamic programming rules.
            temp_score.append(grid[left_x][left_y].score+self.leftEdge)#Soldan indel
            temp_score.append(grid[top_x][top_y].score+self.topEdge)#Üstten indel
            self.score=max(temp_score)

```

Figure-2: Score Calculation Function of Node Class

Backtracking Function:

Outside of node class, we have backtracking function to find correct path in grid to make sequence alignment. Main idea of backtracking with gap openings and extension scores is that preferring indel penalties to mismatches in specific situation such as putting indel and making mismatch are available at same point because gap extension penalty is less than mismatch, so if there will be more than one indel, penalty will be less than mismatches. If blocks are designed for the purpose above. We checked indels first to prefer them to mismatches. After checking indels, matches and mismatches are checked. (node's score – node's edge ?= surrounding node's scores).

```python
#backtracking function which backtracks from end of the grid to starting point of grid according to dynamic pairwise alignment rules.
def backTracking(grid,seq1,seq2):

    x = len(seq1)
    y = len(seq2)

    seq1_align = ""
    seq2_align = ""

    gap_extension_scores = 0 # this extension scores will be added to total scores.

    while x+y != 0:

        '''
        Main idea of backtracking with gap openings and extension scores is that preferring indel penalties to mismatches in
        specific situation such as putting indel and making mismatch are available at same point because
        gap extension penalty is less than mismatch, so if there will be more than one indel, penalty will be less than mismatches.
        '''

        #Checking top edge of the node (INDEL)
        if y != 0 and grid[x][y].score - grid[x][y].topEdge == grid[x][y-1].score:
            seq1_align += '-'
            seq2_align += grid[0][y].nucleotid

            #gap opened
            grid[x][y].isOpeningGap = True

            #checking if gap opened before, if yes than it is gap extension.
            if y != len(seq2):
                if grid[x][y+1].isOpeningGap == True:
                    gap_extension_scores += 0.5
            y -= 1

        #Checking left edge of the node (INDEL)
        elif x != 0 and grid[x][y].score - grid[x][y].leftEdge == grid[x-1][y].score:
            seq1_align += grid[x][0].nucleotid
            seq2_align += '-'

            #gap opened
            grid[x][y].isOpeningGap = True

            #checking if gap opened before, if yes than it is gap extension.
            if x != len(seq1):
                if grid[x+1][y].isOpeningGap == True:
                    gap_extension_scores += 0.5
            x -= 1

        #Checking if there is a mismatch or match.
        elif (x != 0 or y != 0) and grid[x][y].score - grid[x][y].diagonalEdge == grid[x-1][y-1].score: #çapraz
            seq1_align += grid[x][0].nucleotid
            seq2_align += grid[0][y].nucleotid
            x -= 1
            y -= 1

    #print the results
    print(seq1_align[::-1])
    print(seq2_align[::-1])
    print("Score:", grid[-1][-1].score + gap_extension_scores)
```

Figure-3: Backtracking Function

Main Function:

In main function, grid is created and scores of nodes are assigned. Scores of nodes which are on first row and column of grid increase -1 by -1. Also, score of initial node which has (0,0) coordinate in grid is 0. Test inputs are defined as well in main function. After all definitions and assignments, backtracking function is called for each test input. Results are printed on the console.

```python
def main():
    test_inputs=["test1.seq","test2.seq","test3.seq","test4.seq","test5.seq"]

    for file in test_inputs:
        with open("Test Inputs/"+file, "r") as f:
            seq1 = f.readline()[:-1]
            seq2 = f.readline()[:-1]

        grid = [ [Node] * (len(seq2)+1) for i in range(len(seq1)+1)]
        grid[0][0] = Node(0, 0, 'X')

        #Calculating initial score of second sequence (-1 indels)
        for i in range(1,len(seq2)+1):
            node = Node(0,i,seq2[i-1])
            grid[0][i] = node
            node.calculate_score(grid)

        #Calculating initial score of first sequence (-1 indels)
        for i in range(1,len(seq1)+1):
            node = Node(i,0,seq1[i-1])
            grid[i][0] = node
            node.calculate_score(grid)

        #putting other nodes to the grid and calculates the scores
        for i in range(1,len(seq1)+1):
            for y in range(1, len(seq2) + 1):
                node = Node(i, y, "X")
                grid[i][y] = node
                node.calculate_score(grid)

        #printing the results
        print("")
        print("Result of "+ file)
        backTracking(grid,seq1,seq2)


main()
```

Figure-4: Main Function

TEST RESULTS

```
Result of test1.seq
C--GAGACCGA-CG-AAGAGGTTTGGCCC-CAAC-CAGGTTCC-CTGATCACGTAACTT---ACCGGCCAAAAGGAC-T-GGCCTTA-CTAAGGCCT-TT--GTC-TACT-GC-G-G--GT-C--CGG-G-GGC-CGTT-GG-T-T-TCGGCAGAAC-
ACTTC
CCTG-GACCGAGCTTAA-A--TT-G-CTAGCAATACAGATGCCGCT--TC-C-T---TGGGGAG-GGTGTGTAGGATGTAGG--TTAACGAATGCAAGTTCCGGGGTA-TCGCAGAGTCGTGCTACGGCGTGGCAC-TTAGGGTCTCTCGGGAAAAAG
AGTAG
Score: 186.5

Result of test2.seq
GTGT-GGTTGCTTGCATCACTCCGTGTACATGTGACAACCGAACGAGTTGATCCAGCTTTGTTAAGTCAGCTTCGAATG-CGGTAGCTCTCAAA-TATGAT-ATGA-C-TCTTGGGGTAGATGCTG-GG--G-ACCTATTGCGCCC--AAAGCGATAT
TCGGGGC-A-CCGGTTTAGGGTACCCTATCAA-GGCGAT-AC-TTCGATGCAGTGTG-ATGCCGG-A-G-GTGTCG-GTCAGCATGTGAGA-GC-T
G-GTAGGTT-CGTGAAGCACTCC-TGGAC-TCTGACCAC--AAC-A-TA-ATCAAGC---GC-A-G--AG-TT-GAATGACCAAAGCGCTCAAGCT-T--TCACGAACGTCTCA---TA--TCC-GCGGTCGTACAAAC-GCGCTCTTAAAAC-A-AT
-CGT--CTATCCA-TCCGGGG-A---TACCAATGG-G-TGACATTA-A--CTG-G-GCAGGCCGGCACGCG-GACGCGACAG-AT-TGAAATGGAT
Score: 337.0

Result of test3.seq
CGGG-GAAAGAC-GGA-AT--GCATCGACCA-TCGGACAAT-GCCTCAC-TGGAAGCGCTG-CT-GATTTTTGCGCAA-CGAGCCTCGGA-CCTCCCG-C-TCAAACTTACGA-AA--AT-GACT-CCACCAGCACTGAACCA-AC--TG--G--CCT
CCAG-TGA-AGATA--G-TATC-T--A--CA--A-AT-C-GT-TT-G--C-CGGGA-GAAACATTT-GTATGGTAG-TCAGCGTTC-TGCACGTC-ACG-TAATCGTTCACTAGTTGTGGGGTATACCAGGTCGTAATGAGAT-GTTAGTA-TGAATC
GTTTATAACG-CTTGTCATAGGAGTT-C---C-GAATAATCGTCACT-AGGTCAA-T-G---GCCCCCTACT-TGTAATAA-CTA-CGT-CTGATTGAGAAACAGATCGTTAGTCATCGGTTAATAGTCCCGGAAGATAACGGGTTCTTGCGT---TT
-TTGCGAATACTACTATC-TCATGGCGATGG--AG-C----GT---T-TGG---TCCCAT-CCAGCC-G-CGCGAGTATCACT-TGTTCGCCTGCA-C--CTG-TCCGACCTTTCGATGG---GGAGCTC----CTTTCATTGGTT--GTAG-GTACT
-AAGGGTGAGCAATGTCACGTGACCCAA-GGAGCC-GTGTGTAATTTCCACTTGCTCAGAAAAGC-C-TC-GACAAT--C-T---G-GGA-CC-GACACCTGAGTGA-TGGCTT-ACA-GACCAGGGGAGGGG-G-GCAGGTTCCCTGGCCACAGAAT
GGCACGCCCTGAGGAGGCACCGGCCCAA-CGTCGC-TGG
AGTGAGAA-GACCGGATATATGCAACGAACAATCGCAAAATAGC-TC-CATGTACGC-CTGTCCCG-TAATACCGCATGCGAGCCTCGAAGCC-CCCGACGTCAAACCT-CAACAAGTATAGGGTTCCACC-GTA-TGAGCCACAGAATTCAGTTCCT
CC-GCTGATAG-TCCCGGT-TCCTCCAATCAGCAGATTCTGACTTAGGTCACTGGAAGAAACCTGGCGTATTGT-GATGAAATTTCGTG---GTGGACGCTAC-CGT-C--TAG--GTAGAG----CCA---C-TAGTGC-ATTGT-AC-ACTG--TC
TTTT-TC-CGGCTA-TAATCGGAGTTTCAGGCTGACT--T-GCCACCCAGGTCAAATAGTATGTC----ACGGTGTA-TGATCTCTCGGGCTGAT--AGCAAGGGG-CG---G-CGTCGGGCACT---CCTTGAAT-TGACCA-TT-TGGCGTGCCTT
GTTGCC--T-C--CT-TCCTCAAG-C-AGGGTTAGGCTTCAGTAAGTGTGGGGGTGGC-TGCC-GAAAGACG-G-GC-TCGGGGTGTT-GCC--CAACGTCTGGTC-GAC-TTCCCATAATCTGG-GCACAAGACGT--AT-GGACCCGT-GCG--CT
TAATGGTG--CATT-TCACGCGAGA-AACGGAGGCAG-GTGCCAT--CCGCG-GC---GAAAAGATCATCAGACA-TGACATAACGAGGAACCCGAT-CCGGAGTGAATA-CGTCACATG-CCAGGGG-GCGGTGTGCAG-TTCTC-GACCGGA-AAC
GCC-CACCC--ACGACGTACCG-CCAAAGCGTCGCCTGG
Score: 1114.5

Result of test4.seq
AGGC--CGAAAACGTCGCGAAT-T-GACCCTGGCGACGCCGCCGAACGGGACCTCCGTTAGT-GTGG---GA-GGTCATCAATCTCGTTCGCTAGCGGCTGAC-ACCAATCACTATAAGTCTGTCATGAC-
---CTTC-AA---GTCA--AATATAGATCCTGGC--CGCT-CC-A-CGGG-C-T---TAAGTCGTTCTCCGAAGGT-A-CGATCTGGTTGGAT-GCTTCCGTCTA--AA-CA--AGAAGA-TA--AT--CG
Score: 160.5

Result of test5.seq
CGG--GTAGTTAACCCTA-CAG-CATAGAGTCGCGAGATAAAGTGCAGGA-GTCTTTCGCGGCAGATTCGTACCTCAACCACGTGCTACTTTC-TGGCA-TCACGAATC-TGCCGCATAGGTCCGTGAGT-CCATATGA
AGGAAGTAGTTAGCC-TAACAGGCATAGAGTCGCGACATAT-GTGAAG-ATGTCATTCG-GT-A--TTCAAACCTCATGCA--T-C-A-TTGCCTTG-AGTCGC---TCCTGGAGCATAG-TCCCTGAGTGCCATATGA
Score: 247.5
```

Figure-5: Results of Five Test Input

Results can be seen above. Our program works properly according to dynamic pairwise sequence alignment algorithm.