CSE 4065 – Computational Genomics Assignment II

Sena Altıntaş 150118007

Mehmet Akif Akkaya 150118041

First, we get all the files in the folder whose path contains the inputs at the beginning of the main() method. Then we loop through these files one by one.

```
String directoryPath = "src/Test_Inputs/";
File directory = new File(directoryPath);
File[] files = directory.listFiles();
assert files != null;
Arrays.sort(files, Comparator.comparing(File::getName));

for (File file : files) {
    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
```

Then we run our algorithm separately for each file. First, we assign the first line to sequence1 and the second line to sequence2 as follows.

```
// First line is sequence1.
char[] sequence1 = reader.readLine().toCharArray();

// Second line is sequence1.
char[] sequence2 = reader.readLine().toCharArray();
```

Then we send these two sequences to the align() method. Then we find the aligned versions of the sequences by sending the value returned from this method to the findAlignment() method.

```
// Align two sequences and find optimum scores.
ScoreNode[][] score = align(sequence1, sequence2);
String[] alignment = findAlignment(sequence1, sequence2, score);
```

In the align() method, we create a two-dimensional array of ((sequence1.length + 1) x (sequence2.length + 1)). In this array, the score values will be held starting from 0.0 in the top left. The elements of the array will be objects derived from the ScoreNode class.

```
12 usages  Mehmet Akif AKKAYA
class ScoreNode {
    9 usages
    double i; char c; boolean isOpened;
    6 usages  Mehmet Akif AKKAYA
    public ScoreNode(double i, char c, boolean isOpened) {
        this.i = i; this.c = c; this.isOpened = isOpened;
    }
}
```

These objects contain information about the score value, where it came from last (top or left), and whether there was a gap opening penalty before.

In the align method, we define our constant values.

```
public class SequenceAlignment {
    1 usage
    static final double MATCH_SCORE = 3;
    1 usage
    private static final double MISMATCH_SCORE = -1;
    2 usages
    private static final double GAP_OPENING_PENALTY = -1;
    4 usages
    private static final double GAP_EXTENSION_PENALTY = -0.5;
```

After creating a score array in the align method, we fill the first column and first row of this array.

```
// Initialize the score matrix (start node and first GAP_OPENING_PENALTY)
ScoreNode[][] score = new ScoreNode[m + 1][n + 1];
score[0][0] = new ScoreNode( i: 0.0,  c: '-',  isOpened: false);
score[0][1] = new ScoreNode( i: -1.0, c: 'l',  isOpened: true);
score[1][0] = new ScoreNode( i: -1.0,  c: 'u',  isOpened: true);


// Initialize first column (from up to down)
for (int i = 1; i < m; i++) {
    score[i+1][0] = new ScoreNode( i: score[i][0].i + GAP_EXTENSION_PENALTY,  c: 'u',  isOpened: true);
}

// Initialize first row (from left to right)
for (int j = 1; j < n; j++) {
    score[0][j+1] = new ScoreNode( i: score[0][j].i + GAP_EXTENSION_PENALTY,  c: 'l',  isOpened: true);
}
```

We wrote 'l' on all the nodes in the first row because they are accessed from the left. We wrote 'u' on all the nodes in the first column because they are accessed from the top.

Then, starting from the 1:1st element of the score table, which has only the first row and column filled, we started to fill the scores towards the bottom right. We started this by starting a loop as follows. In this way, we fill all the remaining spaces row by row.

```
// Fill in the score matrix
for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
```

In this loop, we calculate the values that the nodes will have in the delete, insert, mismatch, and match cases. As follows:

```
// Calculate score for match or mismatch
double match = (score[i - 1][j - 1].i + (sequence1[i - 1] == sequence2[j - 1] ? MATCH_SCORE : MISMATCH_SCORE));

// Calculate score for deletion. Select GAP_EXTENSION_PENALTY if there was an GAP_OPENING_PENALTY before.
double delete = (!score[i - 1][j].isOpened) ? score[i-1][j].i + GAP_OPENING_PENALTY : score[i-1][j].i + GAP_EXTENSION_PENALTY;

// Calculate score for insertion. Select GAP_EXTENSION_PENALTY if there was an GAP_OPENING_PENALTY before.
double insert = (!score[i][j - 1].isOpened) ? score[i][j-1].i + GAP_OPENING_PENALTY : score[i][j-1].i + GAP_EXTENSION_PENALTY;
```

Then we select the maximum of them.

```
// Choose the maximum score to be placed on the next node.
double nextScore = Math.max(Math.max(match, delete), insert);
```

If a gap opening penalty has occurred in this movement or in previous movements, isOpened is assigned the true value.

```
// Find if there is GAP_OPENING_PENALTY before.
boolean isOpened = match != nextScore || score[i - 1][j - 1].isOpened;
```

A ScoreNode is created. According to the last movement, 'u', 'l' or 'x' letters are assigned. 'x' means here that it came from the top left.

```
// Movement letter for next node.
char nextMovement = (match == nextScore) ? 'x' : (delete == nextScore) ? 'u' : 'l';

// Create new node and place on the score matrix.
score[i][j] = new ScoreNode(nextScore, nextMovement, isOpened);
```

Then the created ScoreNode is added to the score table. When the entire loop is completed, the score in the bottom right gives the maximum score.

Then, we send the value returned by this method back to the findAlignments() method from the main() method.

```
String[] alignment = findAlignment(sequence1, sequence2, score);
```

The findAlignments() method takes the sequences and the score table. Using these data, it returns a String array consisting of aligned sequences using the backtracking method.

```
int x = sequence1.length;
int y = sequence2.length;

// Root node
ScoreNode node = score[x][y];

StringBuilder align1 = new StringBuilder();
StringBuilder align2 = new StringBuilder();
```

In the code above, the variables x and y are assigned the lengths of the sequences. Then, the ScoreNode at the bottom right of the score table is assigned as the root. From this root value, the sequences are aligned by going backwards through all the sequences.

```
while (node.c != '-'){
    switch (node.c) {
        case 'l' -> {
            y--;
            align1.insert( offset: 0,  str: "-");
            align2.insert( offset: 0, sequence2[y]);
            node = score[x][y];
        }
        case 'u' -> {
            x--;
            align2.insert( offset: 0,  str: "-");
            align1.insert( offset: 0, sequence1[x]);
            node = score[x][y];
        }
        case 'x' -> {
            x--;
            y--;
            align1.insert( offset: 0, sequence1[x]);
            align2.insert( offset: 0, sequence2[y]);
            node = score[x][y];
        }
        default -> {
        }
    }
}
```

With this code, all the nodes are followed backwards. The loop continues until it reaches the starting node.

For each node, the previous move is found by checking where it came from. If the previous move was made from the left, continue left, if it was made from the top, continue up, and if it was made from the top left, continue up left, and update the sequences.

After this loop is completed, the String array is returned.

```
// Align two sequences and find optimum scores.
ScoreNode[][] score = align(sequence1, sequence2);
String[] alignment = findAlignment(sequence1, sequence2, score);

// Alignment Result
System.out.println("\n## "+file.getName()+":");
System.out.println(alignment[0]);
System.out.println(alignment[1]);

// Bottom-right element of the score matrix is the optimal alignment score.
System.out.println("Optimal alignment score: " + score[sequence1.length][sequence2.length].i);
System.out.println("\n-------------");
```

With the above code, the maximum scores and aligned form of the sequences in the sample input file are printed.

OUTPUTS:

```
> test1.seq:

-C-GAGACCGA-C-GAAGAGGTT--TGGC--CCCA-A--CCAGGTTCCCT---GA---TCACGTA--ACT-TACCGGCCAAAAGGACTGGCCTTACTAAG-GCCTTTGTCTACTGC-
G-G--GT-C--CGG-G-GGC-CGTT--GGT-T-TC-GGCAGAACACTTC

CCTG-GACCGAGCTTAA-A--TTGCTAGCAATACAGATGCC-GCTTCCTTGGGGAGGGT-GTGTAGGA-TGTA--GG--TTAACGAAT-G-----C-AAGTTCCGGGGTAT-
C-GCAGAGTCGTGCTACGGCGTGGCAC-TTAGGGTCTCTCGGGAAAAAGAGTAG

Optimal alignment score: 206.0
```

```
> test2.seq:

GTGT-GGTTGCTTGCATCACTCCGTGTACATGTGACAACCGAACGAGTTGATCCAGCTTTGTTAAGTCAGCTTCGAATG--C---G-G-T--AGCTCTCA--AA--
TATGATATGACTCTTGGGGTAGATGCTGGGGACCTATTGCGC-C-CAAAGCGATAT--TCGGGGCA-CCGGTTTAGGGTACCCTATCAAGGCGAT-AC-TT--
CGATGCAGTGTGATGCCGG-A-G-GTGTCG-GTCAGCATGTG--A-GAGCT

G-GTAGGTT-CGTGAAGCACTCC-TGGAC-TCTGACCA-C-AAC-A--TAATCAAGC---G--CAG--AG-TT-GAATGACCAAAGCGCTCAAGCTTTCACGAACGTCTCATAT--
C-C---GCG--G-T-C---GTA-CAAACGCGCTCTTAAAAC-A-ATCGTCTATCCATCCGG----GGATA-CC-A--ATGG-G-TGACATTAAC--TG--G-G-CAGGCCGGCACGCG-
GACGCGACAG-AT-TGAAATG-GAT

Optimal alignment score: 365.5
```

```
> test3.seq:

CG-GGGAAAGA-CGG--A-ATGCATCG-ACCATCGGACAAT-GC-CTCACTGGAAGCGCTG--CTGATTTTTGCGCA-ACGAGCCTCG--GACCTCCCG-C-TCAAA-CT-TACGAA--
A-ATGACTCCACC-----AG-CACTGAA-CCAACTGGCCTCCAG-TGA-AGAT---AGTAT-CT---A-CA--A-A-TC-G--TT---TGC-C-GGGAGAAA-----C--ATT-TG-T---
A-----TGGT--A-G-TCAGCGT-T--CT---G-CACGTCACGT-AATCGTTCACTAGT-TGTGGGGTATACCAGG-TCGTAAT--GAGATGTT-A-G-T-A--TG--A---
ATCGTTTATAA-CGCT-TGTCATAGGAGT-TCCGAATAATCGTC--ACT-A-GGTCAA-TGGC--C--C---C-CTACTTGTAATAACT-ACGTC-TGATTGAGAAACAGAT-
CGTTAGTCATCGGTTAATAGTCC-CGGAAGATAACGGGTT---CTTGC-GT---T-T-----T---TG-CGAATACTACTATCTCATGGCGAT-GGAGCGT-TTGGTCCCATCCAGCCG-
CGCGAGTATC-ACTTGTTCGCC-TGCACCT-GTC-C--GACCTTTCGATGGGGAGCTCCTTTCATTGGTTGTAGGTACTAAGGGTGAGCAATGTCACGTGACCCAA-GGA-
GCCGTGTGTAATTTCCACTTGCTCAGAAAAG--CCTC-GAC---A-AT--CTG-GG-A-CCGACACCTGAGTG-AT-GGCTTACA-GACCAGGGGAG-GG-
GGGCAGGTTCCCTGGCCACAGAATGGCACG-CC-CTGAGGAGGCACCGGCCCAA-CGTCG-CTGG

AGTGAG-AAGACCGGATATATGCAACGAACAATCGCAAAATAGCTC-CA-TGTACGC-CTGTCCCG-TAATACCGCATGCGAGCCTCGAAG-CC-CCCGACGTCAAACCTCAAC-
AAGTATAGGGTTCCACCGTATGAGCCACAGAATTC-AGT-TCCTCC-GCTGATAG-TCCCGGT-TCCTCCAATCAGCAGATTCTGACTTAGGT-
CACTGGAAGAAACCTGGCGTATTGTGATGAAATTTCGTGGTGGACGCT-ACCGTCTAGGTAGAGCCAC-T-A-GTGCATTGTACACT-GTCT-T----T-TTCC-GGCT-ATAATCGGAG-
T-TTCAGGCTGACTTGCCACCCA--GGTCA-AATAG-TATGTCA-CGGTGTAT--G-AT-CTC-TCGGGCTGATAG-CAAGGGGCGGCGTCGGGCACTCCTTG-AAT---TGAC--CAT--
TTG-G---C-G-TGCCTT-GT--T-GCCTCCT--TCCTC--AAG--CA-GGGTTAGGCTT-CAGTAAGTGTGGGGGTGGCTGCCG-A-A--A-GA---C--GG-GCTCGG-G-GTGTT-
G-CCCA---A--CGTC-TG-G--TCGAC---TTC-CCAT-AATCTGGGCACAAGA-C-GT--AT--GGA---CC---C----G-TG-CGCT--TAATGGT--GC-ATTTCACGCGA-
GAAACGGAGGCAG-GTGCCA--TCCGC--G-GC-GAAAAGATCATCAGACATGACATAAC-GAGGAACCCGA-TCCGGAGTGAATACG-TCACATG-CCAGGGG-GCGGTGTGCA-GTT--
CTCGAC-CGGAA----ACGCCCACCCACGACGTACC-GCCAAAGCGTCGCCTGG

Optimal alignment score: 1204.5
```

```
> test4.seq:

AGGC--CGAAAACGTCGCGAAT-T-GACCCTGGCGACGCCGCCGAACGGGACCTCCGTTAGT-G---T-GGGAGGTCATCAATCTCGTTCGCTAGCGGCTGACAC--
C-AATCACTATAAG-TCTGTCATGAC

---CTTC---AA-GT--CAAATATAGATCCTGGC--CG-CTCC--ACGGG--CT---TAAGTCGTTCTCCGAAGGT-A-CGATCTGGTT-G---GATGCT-TC-CGTCTAA--
ACAAGAAGAT-AATC--G--

Optimal alignment score: 174.0
```

```
> test5.seq:

CGG--GTAGTTAACCCT-ACA-GCATAGAGTCGCGAGATAAAGTGCAGGA-GTCTTTCGCGGCAGATTCGTACCTCA---ACCACGTGCTACTT---
TCTGGCATCACGAATCTGCCGCATAGGTCCGTGAGT-CCATATGA

AGGAAGTAGTT-AGCCTAACAGGCATAGAGTCGCGACAT-ATGTG-AAGATGTCATT--CGG--TATTCAAACCTCATGCATCA-TTGC--CTTGAGTC--GC-T--C----
CTGGAGCATA-GTCCCTGAGTGCCATATGA

Optimal alignment score: 257.5
```