Spring
2016

Assignment 6

109561770

MOHAMAD MAHDI ZIAEE

Contents

Description

In this document, there two different implementation of Smith Waterman algorithm using Affine Gap and both source codes are included. These two applications can be executed independently and they only share POJOs. In order to execute any of the applications, place the input files into `Assignment6/io` folder. The output file will be in the same folder.

The two applications can be simply executed by running these files:

- `/Assignment6/src/com/bio/main/MainApp.java`
- `/Assignment6/src/com/bio/main/CabiosApp.java`

Application specifications

Programming language: JAVA

Other required installations: JRE 1.8 to run the application,

JDK 1.8 to compile the application.

Version Control: Git (GitHub) - https://github.com/momazia/Bioinformatic/tree/master/Assignment6

IDE: Eclipse (Mars 4.5.1)

Build automation tool: Maven (Refer to `/Assignment6/pom.xml` for the list of the libraries used)

Main application

com.bio.main.MainApp

```
package com.bio.main;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

import com.bio.pojo.AffineResult;
import com.bio.pojo.Sequence;
import com.bio.util.FileUtils;
import com.bio.util.SmithWatermanUtils;

/**
 * This is the main program to be executed in order to run Smith Waterman logic using Affine gap, on a query file and a
set of sequences. Both these
 * files must be placed under /Assignment6/io folder. The output file will be placed under the same folder.
 *
 * @author Mohamad Mahdi Ziaee
 *
 */
public class MainApp {

     /**
      * Main method to be executed to run the application.
      *
      * @param args
      */
     public static void main(String[] args) {
          try {
               // Reading the query file
               Sequence query = FileUtils.getInstance().readQuery(FileUtils.E_COLI_QUERY1_FA);
               // Reading the sequences from the file
               List<Sequence> seqs = FileUtils.getInstance().readSequences(FileUtils.SWISSPROT_100_FA);
               PrintWriter out = FileUtils.getInstance().writeHeader(query, FileUtils.OUTPUT_TXT);
               // Looping through each of the sequences and running SmithWaterman and printing the output into a file.
               for (Sequence sequence : seqs) {
                    AffineResult affineResult = SmithWatermanUtils.getInstance().run(sequence.getStr(), query.getStr());
                    SmithWatermanUtils.getInstance().backTrace(sequence.getStr(), query.getStr(), affineResult);
                    FileUtils.getInstance().write(out, affineResult, sequence.getName(), sequence.getStr().length());
               }
               out.close();
          } catch (IOException e) {
               e.printStackTrace();
          }
     }
}
```

com.bio.util.SmithWatermanUtils

```java
package com.bio.util;

import org.apache.commons.lang3.StringUtils;

import com.bio.pojo.AffineResult;
import com.bio.pojo.Cell;
import com.bio.pojo.Direction;

/**
 * This class is the utility class for all the operations related to Smith Waterman.
 * 
 * @author Mohamad Mahdi Ziaee
 * 
 */
public class SmithWatermanUtils {

    public static final char CHAR_DASH = '-';
    public static final String SPACE = " ";
    private static final int SCORE_GAP_EXT = -1;
    private static final int SCORE_GAP_OPEN = -11;
    private static SmithWatermanUtils instance = null;
    private static char[] ALPHABETS = new char[] { 'A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I', 'L', 'K', 'M', 'F',
'P', 'S', 'T', 'W', 'Y', 'V', 'B', 'J', 'Z', 'X', '*' };
    public static int[][] SCORE_MATRIX = new int[][] { //
            { 4, -1, -2, -2, 0, -1, -1, 0, -2, -1, -1, -1, -1, -2, -1, 1, 0, -3, -2, 0, -2, -1, -1, -1, -4 }, //
            { -1, 5, 0, -2, -3, 1, 0, -2, 0, -3, -2, 2, -1, -3, -2, -1, -1, -3, -2, -3, -1, -2, 0, -1, -4 }, //
            { -2, 0, 6, 1, -3, 0, 0, 0, 1, -3, -3, 0, -2, -3, -2, 1, 0, -4, -2, -3, 4, -3, 0, -1, -4 }, //
            { -2, -2, 1, 6, -3, 0, 2, -1, -1, -3, -4, -1, -3, -3, -1, 0, -1, -4, -3, -3, 4, -3, 1, -1, -4 }, //
            { 0, -3, -3, -3, 9, -3, -4, -3, -3, -1, -1, -3, -1, -2, -3, -1, -1, -2, -2, -1, -3, -1, -3, -1, -4 }, //
            { -1, 1, 0, 0, -3, 5, 2, -2, 0, -3, -2, 1, 0, -3, -1, 0, -1, -2, -1, -2, 0, -2, 4, -1, -4 }, //
            { -1, 0, 0, 2, -4, 2, 5, -2, 0, -3, -3, 1, -2, -3, -1, 0, -1, -3, -2, -2, 1, -3, 4, -1, -4 }, //
            { 0, -2, 0, -1, -3, -2, -2, 6, -2, -4, -4, -2, -3, -3, -2, 0, -2, -2, -3, -3, -1, -4, -2, -1, -4 }, //
            { -2, 0, 1, -1, -3, 0, 0, -2, 8, -3, -3, -1, -2, -1, -2, -1, -2, -2, 2, -3, 0, -3, 0, -1, -4 }, //
            { -1, -3, -3, -3, -1, -3, -3, -4, -3, 4, 2, -3, 1, 0, -3, -2, -1, -3, -1, 3, -3, 3, -3, -1, -4 }, //
            { -1, -2, -3, -4, -1, -2, -3, -4, -3, 2, 4, -2, 2, 0, -3, -2, -1, -2, -1, 1, -4, 3, -3, -1, -4 }, //
            { -1, 2, 0, -1, -3, 1, 1, -2, -1, -3, -2, 5, -1, -3, -1, 0, -1, -3, -2, -2, 0, -3, 1, -1, -4 }, //
            { -1, -1, -2, -3, -1, 0, -2, -3, -2, 1, 2, -1, 5, 0, -2, -1, -1, -1, -1, 1, -3, 2, -1, -1, -4 }, //
            { -2, -3, -3, -3, -2, -3, -3, -3, -1, 0, 0, -3, 0, 6, -4, -2, -2, 1, 3, -1, -3, 0, -3, -1, -4 }, //
            { -1, -2, -2, -1, -3, -1, -1, -2, -2, -3, -3, -1, -2, -4, 7, -1, -1, -4, -3, -2, -2, -3, -1, -1, -4 }, //
            { 1, -1, 1, 0, -1, 0, 0, 0, -1, -2, -2, 0, -1, -2, -1, 4, 1, -3, -2, -2, 0, -2, 0, -1, -4 }, //
            { 0, -1, 0, -1, -1, -1, -1, -2, -2, -1, -1, -1, -1, -2, -1, 1, 5, -2, -2, 0, -1, -1, -1, -1, -4 }, //
            { -3, -3, -4, -4, -2, -2, -3, -2, -2, -3, -2, -3, -1, 1, -4, -3, -2, 11, 2, -3, -4, -2, -2, -1, -4 }, //
            { -2, -2, -2, -3, -2, -1, -2, -3, 2, -1, -1, -2, -1, 3, -3, -2, -2, 2, 7, -1, -3, -1, -2, -1, -4 }, //
            { 0, -3, -3, -3, -1, -2, -2, -3, -3, 3, 1, -2, 1, -1, -2, -2, 0, -3, -1, 4, -3, 2, -2, -1, -4 }, //
            { -2, -1, 4, 4, -3, 0, 1, -1, 0, -3, -4, 0, -3, -3, -2, 0, -1, -4, -3, -3, 4, -3, 0, -1, -4 }, //
            { -1, -2, -3, -3, -1, -2, -3, -4, -3, 3, 3, -3, 2, 0, -3, -2, -1, -2, -1, 2, -3, 3, -3, -1, -4 }, //
            { -1, 0, 0, 1, -3, 4, 4, -2, 0, -3, -3, 1, -1, -3, -1, 0, -1, -2, -2, -2, 0, -3, 4, -1, -4 }, //
            { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -4 }, //
            { -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, 1 } //
    };

    /**
     * Private constructor to avoid this class being initialized by the client.
     */
    private SmithWatermanUtils() {
        super();
    }

    /**
     * Gets the static instance of this class.
     * 
     * @return
     */
    public static SmithWatermanUtils getInstance() {
        if (instance == null) {
            instance = new SmithWatermanUtils();
        }
        return instance;
    }

    /**
```

```
      * The main method to be invoked to run Smith Waterman logic for the given sequence and query strings. If the scores
are passed, they will be
      * used, otherwise it will use Affine gap scoring system.
      *
      * @param sequence
      * @param query
      * @param gapScore
      * @param matchScore
      * @param misMatchScore
      * @return
      */
     public AffineResult run(String sequence, String query, Integer gapScore, Integer matchScore, Integer misMatchScore) {
          sequence = addSpacePrefix(sequence);
          query = addSpacePrefix(query);
          char[] seqChrs = sequence.toCharArray();
          char[] queryChrs = query.toCharArray();
          Cell[][] table = createEmptyTable(seqChrs, queryChrs);
          int maxScore = 0;
          int iIndex = 0;
          int jIndex = 0;
          for (int i = 1; i < queryChrs.length; i++) {
               for (int j = 1; j < seqChrs.length; j++) {
                    int diagScore = table[i - 1][j - 1].getScore() + matchMisMatchScore(seqChrs[j], queryChrs[i],
matchScore, misMatchScore);
                    int horScore = table[i][j - 1].getScore() + getScoreGap(table, i, j, Direction.LEFT, gapScore);
                    int verScore = table[i - 1][j].getScore() + getScoreGap(table, i, j, Direction.TOP, gapScore);
                    table[i][j] = populateCell(diagScore, horScore, verScore);
                    if (maxScore <= table[i][j].getScore()) {
                         iIndex = i;
                         jIndex = j;
                         maxScore = table[i][j].getScore();
                    }
               }
          }
          return new AffineResult(table, maxScore, iIndex - 1, jIndex - 1); // Reducing the indexes by 1 because of the
space added to the strings
     }

     /**
      * Gets the gap score for the position of i and j in table 2D array. If the gapScore is given, that will be used,
otherwise it will check to see
      * if the gap was opened before or not. If the gap was opened before, it will just return the score gap, otherwise it
will add gap opening score
      * together with a single gap score.
      *
      * @param table
      * @param i
      * @param j
      * @param left
      * @param gapScore
      * @return
      */
     private int getScoreGap(Cell[][] table, int i, int j, Direction direction, Integer gapScore) {
          if (gapScore != null) {
               return gapScore;
          }
          switch (direction) {
          case LEFT:
               return direction.equals(table[i][j - 1].getDirection()) ? SCORE_GAP_EXT : SCORE_GAP_OPEN + SCORE_GAP_EXT;
          case TOP:
               return direction.equals(table[i - 1][j].getDirection()) ? SCORE_GAP_EXT : SCORE_GAP_OPEN + SCORE_GAP_EXT;
          default:
               break;
          }
          return 0; // Cannot happen
     }

     /**
      * For 2 given characters, it will find the match/mismatch score in the scoring matrix.
      *
      * @param ch1
      * @param ch2
      * @return
      */
     public int getScore(char ch1, char ch2) {
          return SCORE_MATRIX[getAlphabetPosition(ch1)][getAlphabetPosition(ch2)];
     }

     /**
      * Finds the position of the given character in alphabet array. If the character is not found, it will return the
position of * (last index).
      *
      * @param ch
      * @return
      */
     private int getAlphabetPosition(char ch) {
          for (int i = 0; i < ALPHABETS.length; i++) {
               if (ALPHABETS[i] == ch) {
                    return i;
               }
          }
          return ALPHABETS.length - 1; // It will return the position of * character
     }

     /**
      * Based on the maximum score coming from 3 different directions, it creates a cell which stores the direction the
score was originated and the
      * value of the score. It will look into Diagonal, Top and lastly Left scores in order.
      *
      * @param diagScore
      * @param horScore
      * @param verScore
      * @return
      */
     public Cell populateCell(int diagScore, int horScore, int verScore) {
          int maxScore = 0;
          Direction dir = null;
          if (maxScore <= diagScore) {
               maxScore = diagScore;
               dir = Direction.DIAGONAL;
          }
          if (maxScore <= verScore) {
               maxScore = verScore;
               dir = Direction.TOP;
          }
          if (maxScore <= horScore) {
               maxScore = horScore;
               dir = Direction.LEFT;
          }
          return new Cell(maxScore, dir);
     }

     /**
      * The method instantiates empty cells with zero score for the table.
      *
      * @param seqChrs
      * @param queryChrs
      * @return
      */
     public Cell[][] createEmptyTable(char[] seqChrs, char[] queryChrs) {
          Cell[][] result = new Cell[queryChrs.length][seqChrs.length];
          for (Cell[] cells : result) {
               for (int j = 0; j < cells.length; j++) {
                    cells[j] = new Cell();
               }
          }
          return result;
     }

     /**
      * If both match and mismatch scores are given, they will be used to determine the score. If they are not provided
(in case of Affine), it will
      * get the score from the scoring table.
      *
      * @param seq
      * @param query
```

```
	 * @param matchScore
	 * @param misMatchScore
	 * @return
	 */
	private int matchMisMatchScore(char seq, char query, Integer matchScore, Integer misMatchScore) {
		if (matchScore != null && misMatchScore != null) {
			if (seq == query) {
				return matchScore;
			}
			return misMatchScore;
		}
		return getScore(seq, query);
	}

	/**
	 * Adds an empty character to the beginning of the string passed.
	 *
	 * @param str
	 * @return
	 */
	private String addSpacePrefix(String str) {
		return SPACE + str;
	}

	/**
	 * The main method to be called for Smith Waterman if Affine gap is to be used.
	 *
	 * @param sequence
	 * @param query
	 * @return
	 */
	public AffineResult run(String sequence, String query) {
		return run(sequence, query, null, null, null);
	}

	/**
	 * Back traces the final result by looking at maximum score i and j indexes. At the end, it will reverse the strings.
	 *
	 * @param seq
	 * @param query
	 * @param affineResult
	 */
	public void backTrace(String seq, String query, AffineResult affineResult) {
		StringBuffer seqStr = new StringBuffer();
		query = addSpacePrefix(query);
		seq = addSpacePrefix(seq);
		StringBuffer queryStr = new StringBuffer();
		trace(query.toCharArray(), seq.toCharArray(), affineResult.getTable(), affineResult.getiIndex()+1,
affineResult.getjIndex()+1, seqStr, queryStr);
		affineResult.setSeqStr(StringUtils.reverse(seqStr.toString()));
		affineResult.setQueryStr(StringUtils.reverse(queryStr.toString()));
	}

	/**
	 * The main tracing back method which must be called recursively. The order of finding the single final result is
diagonal, top and left. In case
	 * of insertion or deletion, dash is added.
	 *
	 * @param queryChrs
	 * @param seqChrs
	 * @param table
	 * @param i
	 * @param j
	 * @param seqStr
	 * @param queryStr
	 */
	private void trace(char[] queryChrs, char[] seqChrs, Cell[][] table, int i, int j, StringBuffer seqStr, StringBuffer
queryStr) {
		if (table[i][j].getScore() == 0) {
			return;
		}
		if (table[i][j].getDirection() == Direction.DIAGONAL) {
			queryStr.append(queryChrs[i - 1]);
```

```
                seqStr.append(seqChrs[j - 1]);
                trace(queryChrs, seqChrs, table, i - 1, j - 1, seqStr, queryStr);
            } else if (table[i][j].getDirection() == Direction.TOP) {
                seqStr.append(CHAR_DASH);
                queryStr.append(queryChrs[i - 1]);
                trace(queryChrs, seqChrs, table, i - 1, j, seqStr, queryStr);
            } else if (table[i][j].getDirection() == Direction.LEFT) {
                seqStr.append(seqChrs[j - 1]);
                queryStr.append(CHAR_DASH);
                trace(queryChrs, seqChrs, table, i, j - 1, seqStr, queryStr);
            }
        }
}
```

CabiosApp

com.bio.main.CabiosApp

```
package com.bio.main;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

import com.bio.pojo.AffineResult;
import com.bio.pojo.Sequence;
import com.bio.util.CabiosUtils;
import com.bio.util.FileUtils;

/**
 * This is the main program to be executed in order to run Smith Waterman logic using Affine gap, on a query file and a
set of sequences. Both these
 * files must be placed under /Assignment6/io folder. The output file will be placed under the same folder.
 *
 * @author Mohamad Mahdi Ziaee
 *
 */
public class CabiosApp {

     public static void main(String[] args) {
          try {
               // Reading the query file
               Sequence query = FileUtils.getInstance().readQuery(FileUtils.E_COLI_QUERY1_FA);
               // Reading the sequences from the file
               List<Sequence> seqs = FileUtils.getInstance().readSequences(FileUtils.SWISSPROT_100_FA);
               PrintWriter out = FileUtils.getInstance().writeHeader(query, FileUtils.OUTPUT_TXT);
               // Looping through each of the sequences and running SmithWaterman and printing the output into a file.
               for (Sequence sequence : seqs) {
                    AffineResult affineResult = CabiosUtils.getInstance().sw(sequence.getStr().toCharArray(),
sequence.getStr().length(), query.getStr().toCharArray(), query.getStr().length(), 11, 1);
                    CabiosUtils.getInstance().backTrace(sequence.getStr(), query.getStr(), affineResult);
                    FileUtils.getInstance().write(out, affineResult, sequence.getName(), sequence.getStr().length());
               }
               out.close();
          } catch (IOException e) {
               e.printStackTrace();
          }
     }
}
```

com.bio.util.CabiosUtils

```
package com.bio.util;

import org.apache.commons.lang3.StringUtils;
import org.apache.commons.lang3.math.NumberUtils;

import com.bio.pojo.AffineResult;
import com.bio.pojo.Cell;
import com.bio.pojo.Direction;
```

```java
/**
 * A utility class for Cabios version of Smith Waterman.
 *
 * @author Mohamad Mahdi Ziaee
 *
 */
public class CabiosUtils {

     private static CabiosUtils instance = null;
     private static final char CHAR_DASH = '-';
     private static char[] ALPHABETS = new char[] { 'A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I', 'L', 'K', 'M', 'F',
'P', 'S', 'T', 'W', 'Y', 'V', 'B', 'J', 'Z', 'X', '*' };
     public static int[][] SCORE_MATRIX = new int[][] { //
               { 4, -1, -2, -2, 0, -1, -1, 0, -2, -1, -1, -1, -1, -2, -1, 1, 0, -3, -2, 0, -2, -1, -1, -1, -4 }, //
               { -1, 5, 0, -2, -3, 1, 0, -2, 0, -3, -2, 2, -1, -3, -2, -1, -1, -3, -2, -3, -1, -2, 0, -1, -4 }, //
               { -2, 0, 6, 1, -3, 0, 0, 0, 1, -3, -3, 0, -2, -3, -2, 1, 0, -4, -2, -3, 4, -3, 0, -1, -4 }, //
               { -2, -2, 1, 6, -3, 0, 2, -1, -1, -3, -4, -1, -3, -3, -1, 0, -1, -4, -3, -3, 4, -3, 1, -1, -4 }, //
               { 0, -3, -3, -3, 9, -3, -4, -3, -3, -1, -1, -3, -1, -2, -3, -1, -1, -2, -2, -1, -3, -1, -3, -1, -4 }, //
               { -1, 1, 0, 0, -3, 5, 2, -2, 0, -3, -2, 1, 0, -3, -1, 0, -1, -2, -1, -2, 0, -2, 4, -1, -4 }, //
               { -1, 0, 0, 2, -4, 2, 5, -2, 0, -3, -3, 1, -2, -3, -1, 0, -1, -3, -2, -2, 1, -3, 4, -1, -4 }, //
               { 0, -2, 0, -1, -3, -2, -2, 6, -2, -4, -4, -2, -3, -3, -2, 0, -2, -2, -3, -3, -1, -4, -2, -1, -4 }, //
               { -2, 0, 1, -1, -3, 0, 0, -2, 8, -3, -3, -1, -2, -1, -2, -1, -2, -2, 2, -3, 0, -3, 0, -1, -4 }, //
               { -1, -3, -3, -3, -1, -3, -3, -4, -3, 4, 2, -3, 1, 0, -3, -2, -1, -3, -1, 3, -3, 3, -3, -1, -4 }, //
               { -1, -2, -3, -4, -1, -2, -3, -4, -3, 2, 4, -2, 2, 0, -3, -2, -1, -2, -1, 1, -4, 3, -3, -1, -4 }, //
               { -1, 2, 0, -1, -3, 1, 1, -2, -1, -3, -2, 5, -1, -3, -1, 0, -1, -3, -2, -2, 0, -3, 1, -1, -4 }, //
               { -1, -1, -2, -3, -1, 0, -2, -3, -2, 1, 2, -1, 5, 0, -2, -1, -1, -1, -1, 1, -3, 2, -1, -1, -4 }, //
               { -2, -3, -3, -3, -2, -3, -3, -3, -1, 0, 0, -3, 0, 6, -4, -2, -2, 1, 3, -1, -3, 0, -3, -1, -4 }, //
               { -1, -2, -2, -1, -3, -1, -1, -2, -2, -3, -3, -1, -2, -4, 7, -1, -1, -4, -3, -2, -2, -3, -1, -1, -4 }, //
               { 1, -1, 1, 0, -1, 0, 0, 0, -1, -2, -2, 0, -1, -2, -1, 4, 1, -3, -2, -2, 0, -2, 0, -1, -4 }, //
               { 0, -1, 0, -1, -1, -1, -1, -2, -2, -1, -1, -1, -1, -2, -1, 1, 5, -2, -2, 0, -1, -1, -1, -1, -4 }, //
               { -3, -3, -4, -4, -2, -2, -3, -2, -2, -3, -2, -3, -1, 1, -4, -3, -2, 11, 2, -3, -4, -2, -2, -1, -4 }, //
               { -2, -2, -2, -3, -2, -1, -2, -3, 2, -1, -1, -2, -1, 3, -3, -2, -2, 2, 7, -1, -3, -1, -2, -1, -4 }, //
               { 0, -3, -3, -3, -1, -2, -2, -3, -3, 3, 1, -2, 1, -1, -2, -2, 0, -3, -1, 4, -3, 2, -2, -1, -4 }, //
               { -2, -1, 4, 4, -3, 0, 1, -1, 0, -3, -4, 0, -3, -3, -2, 0, -1, -4, -3, -3, 4, -3, 0, -1, -4 }, //
               { -1, -2, -3, -3, -1, -2, -3, -4, -3, 3, 3, -3, 2, 0, -3, -2, -1, -2, -1, 2, -3, 3, -3, -1, -4 }, //
               { -1, 0, 0, 1, -3, 4, 4, -2, 0, -3, -3, 1, -1, -3, -1, 0, -1, -2, -2, -2, 0, -3, 4, -1, -4 }, //
               { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -4 }, //
               { -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, 1 } //
     };

     /**
      * Private constructor for Singleton design pattern purpose. Declared private so it is not accessible from outside.
      */
     private CabiosUtils() {
     }

     /**
      * Gets instance of this class. It will instantiate it if it is not done yet, once.
      *
      * @return
      */
     public static CabiosUtils getInstance() {
          if (instance == null) {
               instance = new CabiosUtils();
          }
          return instance;
     }

     /**
      * Back traces the final result by looking at maximum score i and j indexes. At the end, it will reverse the strings.
      *
      * @param seq
      * @param query
      * @param affineResult
      */
     public void backTrace(String seq, String query, AffineResult affineResult) {
          StringBuffer seqStr = new StringBuffer();
          StringBuffer queryStr = new StringBuffer();
          trace(query.toCharArray(), seq.toCharArray(), affineResult.getTable(), affineResult.getiIndex(),
affineResult.getjIndex(), seqStr, queryStr);
          affineResult.setSeqStr(StringUtils.reverse(seqStr.toString()));
          affineResult.setQueryStr(StringUtils.reverse(queryStr.toString()));
     }

     /**
      * The main tracing back method which must be called recursively. The order of finding the single final result is
diagonal, top and left. In case
      * of insertion or deletion, dash is added.
      *
      * @param queryChrs
      * @param seqChrs
      * @param table
      * @param i
      * @param j
      * @param seqStr
      * @param queryStr
      */
     private void trace(char[] queryChrs, char[] seqChrs, Cell[][] table, int i, int j, StringBuffer seqStr, StringBuffer
queryStr) {
          if (table[i][j].getScore() == 0 || i <= 0 || j <= 0) {
               return;
          }
          if (table[i][j].getDirection() == Direction.DIAGONAL) {
               queryStr.append(queryChrs[i - 1]);
               seqStr.append(seqChrs[j - 1]);
               trace(queryChrs, seqChrs, table, i - 1, j - 1, seqStr, queryStr);
          } else if (table[i][j].getDirection() == Direction.TOP) {
               seqStr.append(CHAR_DASH);
               queryStr.append(queryChrs[i - 1]);
               trace(queryChrs, seqChrs, table, i - 1, j, seqStr, queryStr);
          } else if (table[i][j].getDirection() == Direction.LEFT) {
               seqStr.append(seqChrs[j - 1]);
               queryStr.append(CHAR_DASH);
               trace(queryChrs, seqChrs, table, i, j - 1, seqStr, queryStr);
          }
     }

     /**
      * Main method to be executed in order to run Smith Waterman logic.
      *
      * @param seqA
      * @param lena
      * @param seqB
      * @param lenb
      * @param gap_open
      * @param gap_ext
      * @return
      */
     public AffineResult sw(char[] seqA, int lena, char[] seqB, int lenb, int gap_open, int gap_ext) {
          int my_i = 0;
          int my_j = 0;
          int[] nogap = new int[lena];
          int[] b_gap = new int[lena];
          int last_nogap, prev_nogap;
          int score = 0;
          init_vect(lena, nogap, 0);
          init_vect(lena, b_gap, -gap_open);
          Cell[][] table = createEmptyTable(seqA, seqB);
          for (int i = 0; i < lenb; i++) {
               int a_gap;
               last_nogap = prev_nogap = 0;
               a_gap = -gap_open;
               for (int j = 0; j < lena; j++) {
                    Direction dir = null;
                    a_gap = NumberUtils.max((last_nogap - gap_open - gap_ext), (a_gap - gap_ext));
                    b_gap[j] = NumberUtils.max((nogap[j] - gap_open - gap_ext), (b_gap[j] - gap_ext));
                    int diagScore = (prev_nogap + getScore(seqA[j], seqB[i]));
                    last_nogap = NumberUtils.max(diagScore, 0);
                    if (diagScore > 0) {
                         dir = Direction.DIAGONAL;
                    }
                    int horScore = a_gap;
                    if (last_nogap < horScore) {
                         dir = Direction.LEFT;
                    }
                    last_nogap = NumberUtils.max(last_nogap, a_gap);
                    int verScore = b_gap[j];
                    if (last_nogap < verScore) {
                         dir = Direction.TOP;
                    }
                    last_nogap = NumberUtils.max(last_nogap, b_gap[j]);
                    prev_nogap = nogap[j];
                    nogap[j] = last_nogap;
                    if (score <= last_nogap) {
                         my_i = i;
                         my_j = j;
                    }
                    score = NumberUtils.max(score, last_nogap);
                    table[i][j] = new Cell(score, dir);
               }
          }
          return new AffineResult(table, score, my_i, my_j);
     }
     /**
      * For 2 given characters, it will find the match/mismatch score in the scoring matrix.
      *
      * @param ch1
      * @param ch2
      * @return
      */
     public int getScore(char ch1, char ch2) {
          return SCORE_MATRIX[getAlphabetPosition(ch1)][getAlphabetPosition(ch2)];
     }
     /**
      * Finds the position of the given character in alphabet array. If the character is not found, it will return the
position of * (last index).
      *
      * @param ch
      * @return
      */
     private int getAlphabetPosition(char ch) {
          for (int i = 0; i < ALPHABETS.length; i++) {
               if (ALPHABETS[i] == ch) {
                    return i;
               }
          }
          return ALPHABETS.length - 1; // It will return the position of * character
     }
     /**
      * The method instantiates empty cells with zero score for the table.
      *
      * @param seqChrs
      * @param queryChrs
      * @return
      */
     private Cell[][] createEmptyTable(char[] seqChrs, char[] queryChrs) {
          Cell[][] result = new Cell[queryChrs.length][seqChrs.length];
          for (Cell[] cells : result) {
               for (int j = 0; j < cells.length; j++) {
                    cells[j] = new Cell();
               }
          }
          return result;
     }
     /**
      * Puts the value given into all the elements of the array.
      *
      * @param lena
      * @param array
      * @param val
      */
     private void init_vect(int lena, int[] array, int val) {
          for (int i = 0; i < lena; i++) {
               array[i] = val;
          }
     }
}
```

Common classes

com.bio.pojo.AffineResult

```
package com.bio.pojo;

/**
 * The main POJO to hold the output of the program.
 *
 * @author Mohamad Mahdi Ziaee
 *
 */
public class AffineResult {

     private Cell[][] table;
     private int maxScore = 0;
     private int iIndex;
     private int jIndex;
     private String queryStr;
     private String seqStr;

     /**
      * Constructor which sets the values in the POJO.
      *
      * @param table
      * @param maxScore
      * @param iIndex
      * @param jIndex
      */
     public AffineResult(Cell[][] table, int maxScore, int iIndex, int jIndex) {
          this.table = table;
          this.maxScore = maxScore;
          this.iIndex = iIndex;
          this.jIndex = jIndex;
     }

     public Cell[][] getTable() {
          return table;
     }

     public void setTable(Cell[][] table) {
          this.table = table;
     }

     public int getMaxScore() {
          return maxScore;
     }

     public void setMaxScore(int maxScore) {
          this.maxScore = maxScore;
     }

     public int getiIndex() {
          return iIndex;
     }

     public void setiIndex(int iIndex) {
          this.iIndex = iIndex;
     }

     public int getjIndex() {
          return jIndex;
     }

     public void setjIndex(int jIndex) {
          this.jIndex = jIndex;
     }

     public String getQueryStr() {
          return queryStr;
     }

     public void setQueryStr(String queryStr) {
          this.queryStr = queryStr;
     }

     public String getSeqStr() {
          return seqStr;
     }

     public void setSeqStr(String seqStr) {
          this.seqStr = seqStr;
     }

     /**
      * Overriding equals method to compare two affineResults together by checking their i, j indexes and the max score.
This is used in Unit test to
      * figure out how MainApp's result is different from CabiosApp. It can be ignored.
      *
      * @param obj
      */
     @Override
     public boolean equals(Object obj) {
          if (obj instanceof AffineResult) {
               return ((AffineResult) obj).getiIndex() == getiIndex() && ((AffineResult) obj).getjIndex() == getjIndex() &&
((AffineResult) obj).getMaxScore() == getMaxScore();
          }
          return false;
     }

     /**
      * Overriding toString to pretty print the content of Affine result class.
      */
     @Override
     public String toString() {
          return String.format("MaxScore: %s, i: %s, j: %s", getMaxScore(), getiIndex(), getjIndex());
     }
}
```

com.bio.pojo.Cell

```
package com.bio.pojo;

/**
 * Cell object represents each of the cells in Smith Waterman algorithm table. It holds a score and the direction in which
the result of that score
 * came from. The direction is later used for back tracing.
 *
 * @author Mohamad Mahdi Ziaee
 *
 */
public class Cell {

    private int score;
    private Direction direction;

    /**
     * Constructor to set the values into the POJO.
     *
     * @param score
     * @param dir
     */
    public Cell(int score, Direction dir) {
        this.score = score;
        this.direction = dir;
    }

    public Cell() {
        this.score = 0;
        this.direction = null;
    }

    public int getScore() {
        return score;
    }
	public void setScore(int score) {
		this.score = score;
	}

	public Direction getDirection() {
		return direction;
	}

	public void setDirection(Direction direction) {
		this.direction = direction;
	}

}
```

com.bio.pojo.Direction

```
package com.bio.pojo;

/**
 * An enumeration to represent the directions available in Smith Waterman algorithm.
 * 
 * @author Mohamad Mahdi Ziaee
 * 
 */
public enum Direction {
	LEFT, DIAGONAL, TOP
}
```

com.bio.pojo.Sequence

```
package com.bio.pojo;

/**
 * A POJO to hold the sequence name and its string value.
 * 
 * @author Mohamad Mahdi Ziaee
 * 
 */
public class Sequence {

	private String name;
	private String str;

	/**
	 * Default constructor for Sequence object.
	 * 
	 * @param name
	 * @param str
	 */
	public Sequence(String name, String str) {
		this.name = name;
		this.str = str;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getStr() {
		return str;
	}

	public void setStr(String str) {
		this.str = str;
	}
}
```

com.bio.util.FileUtils

```
package com.bio.util;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;

import org.apache.commons.lang3.StringUtils;

import com.bio.pojo.AffineResult;
import com.bio.pojo.Sequence;

/**
 * This utility class is in charge of all the IO related operations needed.
 *
 * @author Mohamad Mahdi Ziaee
 *
 */
public class FileUtils {
     private static final String PLUS = "+";
     private static final String SPACE = " ";
     public static final String SWISSPROT_100_FA = "swissprot-100.fa";
     public static final String E_COLI_QUERY1_FA = "EColi-query1.fa";
     public static final String IO_PATH = "../Assignment6/io/";
     public static final String OUTPUT_TXT = "output.txt";
     private static FileUtils instance = null;
     private static final DecimalFormat df = new DecimalFormat("#");

     /**
      * Private constructor for Singleton design pattern purpose. Declared private so it is not accessible from outside.
      */
     private FileUtils() {
     }

     /**
      * Gets instance of this class. It will instantiate it if it is not done yet, once.
      *
      * @return
      */
     public static FileUtils getInstance() {
          if (instance == null) {
               instance = new FileUtils();
          }
          return instance;
     }

     /**
      * Read the query file and returns the query string by putting the whole sequence into one single line.
      *
      * @param fileName
      * @return
      * @throws IOException
      */
     public Sequence readQuery(String fileName) throws IOException {
          StringBuffer str = new StringBuffer();
          List<String> lines = readFile(fileName);
          for (int i = 1; i < lines.size(); i++) {
               String line = lines.get(i);
               if (StringUtils.isNotBlank(line)) {
                    str.append(line);
               }
          }
          return new Sequence(lines.get(0), str.toString());
     }

     /**
      * Reads the file name given in io folder and returns a list of strings representing each line in the file.
      *
      * @param fileName
      * @return
      * @throws IOException
      */
     private List<String> readFile(String fileName) throws IOException {
          return Files.readAllLines(Paths.get(FileUtils.IO_PATH + fileName));
     }

     /**
      * Reads the sequences from the sequence file name given.
      *
      * @param seqFileName
      * @return
      * @throws IOException
      */
     public List<Sequence> readSequences(String seqFileName) throws IOException {
          List<Sequence> results = new ArrayList<>();
          String header = null;
          List<String> lines = readFile(seqFileName);
          for (int i = 0; i < lines.size(); i++) {
               if (i % 2 != 0) {
                    results.add(new Sequence(header, lines.get(i)));
               } else {
                    header = lines.get(i);
               }
          }
          return results;
     }

     /**
      * Deletes the file placed in io folder if it already exists.
      *
      * @param fileName
      * @throws IOException
      */
     public void deleteIfExists(String fileName) throws IOException {
          Files.deleteIfExists(Paths.get(FileUtils.IO_PATH + fileName));
     }

     /**
      * Writes the final result to the PrintWriter object passed.
      *
      * @param out
      * @param affineResult
      * @param name
      * @param seqLength
      */
     public void write(PrintWriter out, AffineResult affineResult, String name, int seqLength) {
          out.print(formatOutput(affineResult, name, seqLength));
     }

     /**
      * Formats the output to look like BlastN output.
      *
      * @param affineResult
      * @param name
      * @param seqLength
      * @return
      */
     public String formatOutput(AffineResult affineResult, String name, int seqLength) {
          String queryStr = affineResult.getQueryStr();
          int resultLength = queryStr.length();
          int iIndex = affineResult.getiIndex();
          int jIndex = affineResult.getjIndex();
          String seqStr = affineResult.getSeqStr();
          StringBuffer str = new StringBuffer();
          str.append(name + " (len=" + seqLength + ")\n");
          str.append("Score = " + affineResult.getMaxScore() + " (i=" + iIndex + ", j=" + jIndex + ")\n");
          String similarityString = getSimilarityString(queryStr.toCharArray(), seqStr.toCharArray());
          int identity = getIdentity(similarityString);
          int positives = getPositive(similarityString);
          int queryStrLength = queryStr.length();
```

```
        int gaps = getGaps(queryStr, seqStr);
        str.append(String.format("Identities = %s/%s (%s%%), Positives = %s/%s (%s%%), Gaps = %s/%s (%s%%)\n", identity,
queryStrLength, getPerc(identity, queryStrLength), positives, queryStrLength, getPerc(positives, queryStrLength), gaps,
                queryStrLength, getPerc(gaps, queryStrLength)));
        str.append(String.format("Query: %5s %s %s\n", iIndex - resultLength + 1, queryStr, iIndex));
        str.append(String.format("\t\t\t %s\n", similarityString));
        str.append(String.format("Sbjct: %5s %s %s\n\n", jIndex - resultLength + 1, seqStr, jIndex));
        return str.toString();
    }

    /**
     * Counts number of gaps in both query and DB sequence
     *
     * @param queryStr
     * @param seqStr
     * @return
     */
    private int getGaps(String queryStr, String seqStr) {
        int counter = 0;
        for (int i = 0; i < queryStr.length(); i++) {
            if (queryStr.charAt(i) == '-') {
                counter++;
            }
        }
        for (int i = 0; i < seqStr.length(); i++) {
            if (seqStr.charAt(i) == '-') {
                counter++;
            }
        }
        return counter;
    }

    /**
     * Returns number of characters in similarity String which are not spaces.
     *
     * @param str
     * @return
     */
    private int getPositive(String str) {
        int counter = 0;
        for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) != ' ') {
                counter++;
            }
        }
        return counter;
    }

    /**
     * Calculates percentage by following formula: (val1/val2) * 100
     *
     * @param val1
     * @param val2
     * @return
     */
    private String getPerc(int val1, int val2) {
        return df.format((val1 / Double.valueOf(val2)) * 100.0);
    }

    /**
     * Returns identity by looking at the similarity String and counting all the characters that are not space or plus.
     *
     * @param str
     * @return
     */
    private int getIdentity(String str) {
        int counter = 0;
        for (int i = 0; i < str.length(); i++) {
            if (!(str.charAt(i) == '+' || str.charAt(i) == ' ')) {
                counter++;
            }
        }
        return counter;
     }

     /**
      * Generates similarity string. If the characters are the same, it will be printed as it is. Otherwise, it looks up
into scoring matrix and if the
      * score is zero or a positive number, it will print +, otherwise a space will be used.
      *
      * @param chr1s
      * @param chr2s
      * @return
      */
     private String getSimilarityString(char[] chr1s, char[] chr2s) {
          StringBuffer str = new StringBuffer();
          for (int i = 0; i < chr1s.length; i++) {
               if (chr1s[i] == chr2s[i]) {
                    str.append(chr1s[i]);
               } else {
                    str.append(CabiosUtils.getInstance().getScore(chr1s[i], chr2s[i]) >= 0 ? PLUS : SPACE);
               }
          }
          return str.toString();
     }

     /**
      * Writes the header by printing the query name and string into the given output file name. If the file already
exists, it will delete it first.
      * The method returns PrintWriter pointing the output file.
      *
      * @param headerSequence
      * @param outputFileName
      * @return
      * @throws IOException
      */
     public PrintWriter writeHeader(Sequence headerSequence, String outputFileName) throws IOException {
          // Deleting the output file if it already exists
          deleteIfExists(outputFileName);
          PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(FileUtils.IO_PATH + outputFileName, true)));
          out.println(headerSequence.getName());
          out.println(headerSequence.getStr());
          out.println();
          return out;
     }
}
```