# Source Code Documentation
# Jin Tao 11474660

## Class Summary:

My source code includes three classes:
1. Cell Class: This class implements one element in the alignment matrix.
2. Blosum62Matrix Class: This class implements how to get the corresponding BLOSUM62 score for specific amino acid pairs from the BLOSUM62 scoring matrix.
3. GlobalSequenceAlignmentTable Class: This class implements the Global Sequence Alignment Dynamic Programming Algorithm.

## Class Description:

### 1. Cell Class:

Each object in the Cell class represents one element in the alignment matrix. According to the Global Sequence Alignment Dynamic Programming Algorithm, each element has a score and a pointer for final tracking back.

Instance Variable Description:
The Cell class includes four instance variables:
Instance variable *row* and *col* are used to specify the position of each Cell object in the alignment matrix.
Instance variable *score* represents the score of each Cell object.
Instance variable *prevCell* represents the tracking back pointer of each Cell object.

Constructor Description:
*public Cell(int i, int j)* creates a Cell object according to the position in the alignment matrix, with *0* as the initial score and *null* as the initial tracking back pointer.

Method Description:
*public void setScore(int Score):* This setter method sets the score for each Cell object.
*public void setPrevCell(Cell Pointer):* This setter method sets the arrow inside each Cell object for back tracking.
*public int getScore():* This getter method gets the score of each Cell object.
*public int getRow():* This getter method gets the row number of each Cell object in the alignment matrix.
*public int getCol():* This getter method gets the column number of each Cell object in the alignment matrix.
*public Cell getPrevCell():*This getter method gets the arrow inside each Cell object in the alignment matrix.

### 2. Blosum62Matrix Class:

This class implements how to get the corresponding BLOSUM62 score for specific amino acid pairs from the BLOSUM62 scoring matrix. Initially, I define a constant class variable *matrix*, which is the BLOSUM62 scoring matrix.

Method Description:
*private static int getIndex(char input):* This method returns the column or row index for input character representing specific amino acid in the BLOSUM62 scoring matrix.
*private static int getScore(int row, int col):* This method returns the corresponding BLOSUM62 score given the position in the BLOSUM62 scoring matrix.
*public static int getDiagScore(char a1, char a2):* This methods returns the corresponding BLOSUM62 score for specific amino acid pairs from the BLOSUM62 scoring matrix. As a class method, in the global sequence alignment we directly use Blosum62Matrix.getDiagScore(char a1, char a2) to get the corresponding BLOSUM62 score for specific amino acid pairs.

## 3. GlobalSequenceAlignmentTable Class:

This class implements the Global Sequence Alignment Dynamic Programming Algorithm for DNA sequence and protein sequence. The essential frameworks for two kinds of sequence alignment are the same except that we follow the BLOSUM62 scoring matrix to calculate the score of matching amino acid pairs in aligning protein sequences while we use the constant score of matching/mismatching, namely, 1 and -1, in aligning DNA sequences.

Method Description:
*protected static void initializeScores(Cell[][] scoreTable):* This method initializes the scores in all cells in the first row and the first column in the alignment matrix.
*protected static void initializePointers(Cell[][] scoreTable):* This method initializes the arrows in all cells in the first row and the first column in the alignment matrix.
*protected static void initialize(Cell[][] scoreTable):* This method builds each cell inside the alignment matrix and initialize the whole alignment matrix.
*protected static void fillInCellDNASeq(String sequence1,String sequence2, Cell currentCell,Cell cellAbove, Cell cellToLeft, Cell cellAboveLeft):* This method fills in one Cell object in the alignment matrix for the DNA sequence alignment. For DNA sequence alignment, we use 1 and -1 as the score of matching/mismatching; use -2 as the gap penalty.
*protected static void fillInCellProSeq(String sequence1,String sequence2, Cell currentCell,Cell cellAbove, Cell cellToLeft, Cell cellAboveLeft):* This method fills in one Cell object in the alignment matrix for the protein sequence alignment. For protein sequence alignment, we follow the BLOSUM62 scoring matrix to calculate the score of matching one amino acid with another; use -2 as the gap penalty.
*protected static Cell[][] fillInTable(String str1, String str2,int choice):* This method fills in the whole alignment matrix depending on the input choice on DNA/protein sequence alignment.
*protected static String[] trackBack(String str1, String str2,Cell[][] scoreTable):* This method constructs the alignment from tracing back path. Tracing back begins at the element in the bottom right hand corner of the matrix and then it follows pointers that gave maximum score for each element. Tracing back continues until the top left corner of the matrix is reached.

*protected static void repeatedProcess(String strt1, String strt2,int choice):* This method represents the process of aligning two sequences based on the input choice parameter. It first fills in the alignment matrix and then traces back to get the final pairwise alignment. When choice=0, it aligns the DNA sequences; when choice=1, it aligns the protein sequences.
*protected static void TestingMode():* This method runs three testing cases respectively for DNA sequence alignment and protein sequence alignment.

**Main Method:**
*public static void main(String[] args) throws IOException:*
Firstly, the main function runs the six testing cases. Afterwards, the user can test whatever they like based on their preference on whether aligning DNA sequences or protein sequences by changing the value for the parameter *choice*. The program won't end until the user chooses to exit.

**Testing Cases:**

```
     DNA sequence alignment testing:
1.
     strt1="ACCGTA";
     strt2="ACGTT";
2.
     strt3="CGGATTGCATTACG";
     strt4="CGATTCATG";
3.
     strt5="TAGGCTGAGCGACCGCGTA";
     strt6="TGGCCTAGGCCGA";

     Protein sequence alignment testing:
1.
     str1="MEKVNEERDAVF";
     str2="EDHIGDRRRSV";
2.
     str3="RSLLEEA";
     str4="FADEMEKT";
3.
     str5="SYDVEVADTPQPHIPIRFRHPPIA";
     str6="MKRGQAVDFCHWVSHLIATEIDEK";
```