**Carleton University**

**Department of Systems and Computer Engineering**

**SYSC 2004 – Object Oriented Software Development**

**Project Name**

***Design of DNA Sequence Alignment***

**Team Members:**

**Amer Binmuhana**

**Student IDs:**

**100952424**

Professor: Moayad Aloqaily

TA: Ala'a Al-Habashna

Date:

# Problem Identification and Statement

The alignment of DNA sequence is a common practice within the biology arena. The goal of this project is to apply object-oriented skills such as classes, inheritance, polymorphism, array list, and etc. to solve realistic and crucial problems within the science society. In this project, the exposure to some algorithms like NEEDLEMAN-WUNSCH is expected, thus an opportunity to understand and analyse such algorithms will be a great asset in the future. The alignment is basically analyzing two DNA sequences of two different species to check if they are related. In general DNA sequence consist of both forward strand and reverse strand, however in this project only forward strand is of interest. The procedure of alignment is done by aligning both sequences and compare each character with the corresponding one. The problem I am concerned about in this project is how similar two DNA sequences are and the production of a score that shows the similarity. The initial plan is to construct DNA class that takes ID (the name of the specie) and sequence as fields. The user will be promoted to insert two DNA forward strands and the program will treat both DNA strands as objects of the DNA class. The sequences are considered strings. Since I am using NEEDLEMAN-WUNSCH algorithm thus I will take each character at a time because this is how the algorithm works. When aligning character from sequence 1 with the one from sequence 2, there are two scenarios possible. First, it could be aligned to another character from sequence 2. Second, it could be aligned to a gap. Figure 1 shows an alignment example for certain DNA pieces. ‘|’ indicates both characters are matching.



Figure 1an alignment example

‘\_’ this sign represents a gap in the sequence. Needleman-Wunsch algorithm reorganizes the sequences to conserve the genetic code, maximize the number of matches using score method, minimize the number of gaps through gap penalty method and decrease the number of mismatches using mismatch penalty method. Therefore, the algorithm obtains the optimal global alignment score by summing all matches, mismatches and gap penalties. The traceback is used to identify all possible optimal alignments. I will use a stack to trace all possible paths.

# Gathering of Information and Input/Output Description

## DNA object construction

DNA object will be characterized using an ID (to uniquely identify the DNA object) and DNA sequence (obtained from the user).

## Dynamic programming matrix

The DPM will be used to obtain the optimal alignment (s) of the two sequences. The DPM will be initialized according the gap penalty supplied by the user. The matrix is then will be populated following Needleman-Wunsch. The rightmost cell at the very bottom of the populated matrix represents the optimal score for the alignment.

## Traceback

The trace-back will be implemented to obtain all possible alignments of two sequences. The trace-back works in reverse in order to find the alignment that led to the optimal alignment score.

## Input and Output

The user will be prompt to insert both DNA sequences and the parameters. First, I will prompt the user to enter the first sequence and verify if the sequence inserted is valid by comparing the characters with the known DNA sequence characters C, G, A, T then I will create the first DNA object. Second, , I will prompt the user to enter the second sequence and verify if the sequence inserted is valid by comparing the characters with the known DNA sequence characters C, G, A, T then I will create the second DNA object. After that, I will ask the user to insert the following parameters: match score, mismatch score, gap penalty and they will be assigned to the appropriate variables. After the verification I will print out the following statement:” . . . sequences verified and DNA object constructed . . . “. After that, the user will be asked to choose between 7 options that will be apparent in the console. Options as following:

1. Print both sequences: the program will print both inserted sequences.
2. Print initialized matrix: Print the initial DPM.
3. Result of the Needleman-Wunsch: print the matrix after the algorithm being used.
4. Print one optimal alignment: print only the first optimal alignment without using traceback.
5. Print all optimal alignments: print all optimal alignment that is on the traceback stack.
6. Enter two new sequences: gets two new sequences and the cycle starts again.
7. Exit: the program is terminated.

# Test Cases and Design

## Test Cases

1. Check if one of the sequences inserted is too long or both. If yes, tell the user that the sequence is too long and prompt the user to insert another sequence.
2. Check if one the characters is not one of the following letters: A,C,G,T. If yes, notify the user of an invalid character, point out the invalid character and ask the user to insert the sequence again after the removal of the invalid character.
3. Check if a character other than numbers for match score, mismatch score, and gap penalty. If yes, prompt the user to renter the value as a digit.
4. Check if any of the sequences inserted is empty. If yes, ask the user to renter a non-empty sequences.
5. Check if any lower case character is inserted. If yes convert them to upper case and continue.

## Design

### Introduction

The design basically consist of four classes. The first class is called DNASequenceAlignentSystem. This class contains the main function and get all the inputs from the users such as both sequences through ParseUserInput method. This class also has all the test cases mentioned in 3.1 to verify the validity of the inputs.

The second class called DNA that has two instant variables ID, and sequence. This class has couple of methods such as getters and setters for the two fields as well as toString method that prints both sequences.

The third class is DPM (Dynamic programming Matrix) which has multiple instant variables matrix[][] which is a 2D array, rows , cols (columns) , gapPenalty, matchPenalty,and misMatchPenalty. The last three instant variables for the third class are inputted by the user. It has three methods initilaizematrix, FillMatrix and score method.

The fourth class is called Traceback that has three instant variables dmp which is of type DPM , DNASequence\_1, DNASequence\_2. This class has two methods findAlignment and toString. Please refer to figure 2 that has the UML for the design.

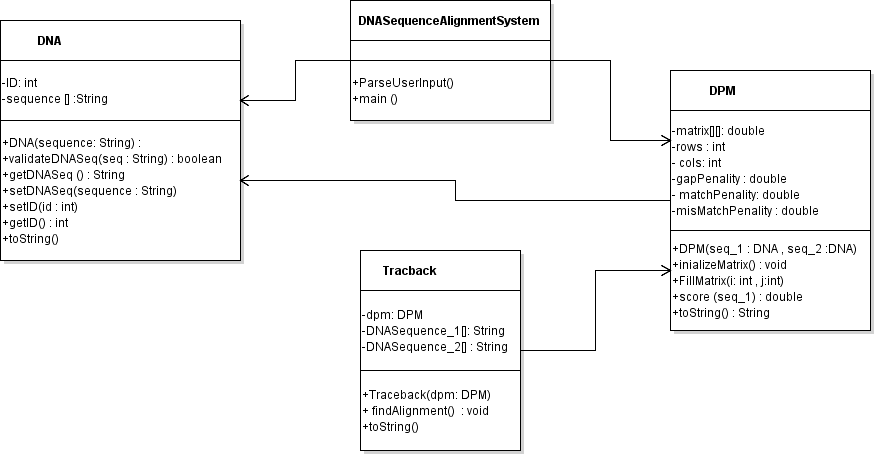


Figure 2 UML of the design

### Method(s) for Interacting with the User

The DNASequenceAlignmentSystem class will be responsible for the interaction with user. It will prompt the user to insert the data through the console. The function system.out.println() will be used to display the instructions to the user.

The Scanner function will be used to collect the data from the user and setters will take the data inserted to the corresponding object if it is already created.

### Functions for Calculations

1. Public double score (char seq\_i, char seq\_j)

This method is implemented in the DPM class to determine the match and mismatch between two characters. This procedure is useful for the traceback.

**Parameters:**

* Seq\_i : the first character to compare from the first sequence
* Seq\_j: the second character to compare from the second sequence.

**Return value:**

This method return a double value which could be the match or mismatch penalty.

**Logic**

This method takes the two characters and compares them. If the characters are the same the match penalty will be returned, otherwise the mismatch penalty will be returned. This simply implemented using if-statements.

1. Public void InitializeMatrix ()

This method is implemented within the DPM class to insert the initial values for the first row and column of the 2D matrix.

**Parameters:**

None.

**Return value:**

None.

**Logic**

It initializes both first row and column using this formula F(i,0) = gap penalty\*i , F(0,j) = gap penalty\*j. this can be done by using for loops to access the rows and columns.

1. Public double FillMatrix ()

This method is implemented within the DPM class to fill the matrix completely using Needleman-Wunsch algorithm.

**Parameters:**

none

**Return value:**

It returns the optimal alignment value.

**Logic**

It initializes both first row and column using this formula F(i,0) = gap penalty\*i , F(0,j) = gap penalty\*j. this can be done by using for loops to access the rows and columns. After the completion of the filling process, the very last value is returned as the optimal alignment score.

1. Public void FindAlignment ()

This method is implemented within the traceback class. It takes the filled matrix from the DPM class and does the traceback procedure to find the possible alignments.

**Parameters:**

none

**Return value:**

none.

**Logic**

It does implement the traceback procedure and fill the two strings DNASequence\_1, and DNASequence\_2 with the possible alignment. I am still working on figuring out how to implement traceback procedure.

### Other Functions

Getters/setters

-These methods are used to allow any class to access a private variable. All the getters have no parameters. Each method has a return type as demonstrated below.

 Public int getID();

Public String getSequence();

Public double getMatchScore();

Public double getMismatchScore();

Public double getGapPenalty();

-setter is a method that has one parameter and sets this parameter to a new field, and it does not have a return type (void).

Public int setID(int ID);

Public void setSequence(String sequence);

Public void setMatchScore(double matchScore);

Public void setMismatchScore(double mismatchScore);

Public void setGapPenalty(double gapPenalty);

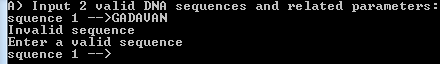
# Implementation

See the Java program in attached file.

# Software Testing and Verification

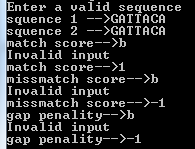
Case1: The program doesn’t check for this case !

Case 2: The program checks if any of the characters inserted is not one of the following: AGCT. The user is notified that the sequence is invalid. The program asks the user to renter a valid sequence. The following figure shows the output of the program when a character other than the four letters is inserted.

****

**Figure 3 Case2**

Case 3: if the value of match score, mismatch score, and gap penalty is not a digit, the program rejects the value and prompts the user to enter a valid input.



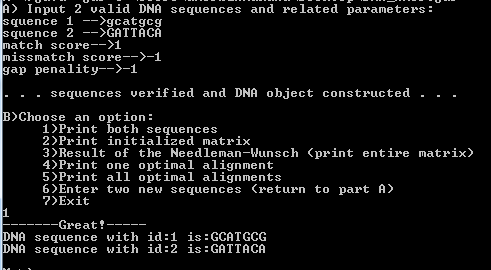
**Figure 4 Case3**

Case 4: if one of the sequences entered is empty, the program asks the user to enter a valid sequence.



**Figure 5 Case4**

Case 5: if any the sequence entered is lower case letters, the program converts them to upper case and continue the same way. Figure 6 shows this case. It can be observed the sequence 1 is lower case characters and when the sequences are printed along with the ID’s, sequence 1 is printed in upper case characters. After that, the sequences in upper case are used to build the Needleman-Wunsch matrix.



**Figure 6 Case 5**