East West University

Algorithm (CSE246)

Project Title : Sequence Alignment Problem

(Project Report)

Submitted by

Name	ID
Md. Naeemur Rahman	2022-1-60-167
Md. Sadat Ahmed Rafi	2022-1-60-261
Ahsan Aftab	2019-1-60-118
Rajia Afrin Rima	2021-2-60-029

Section: 02

Semester: Fall 2023

Submitted to

Jesan Ahammed Ovi

Senior Lecturer

Department of Computer Science and Engineering

Date of Submission: 24 December 2023

SEQUENCE ALIGNMENT PROBLEM

INTRODUCTION

The Sequence Alignment Project offers a user-friendly tool for aligning biological sequences using the Needleman-Wunsch algorithm. This bioinformatics technique helps researchers compare DNA, RNA, or protein sequences, identifying similarities and differences.

SYSTEM DESIGN

The Needleman-Wunsch algorithm is employed for global sequence alignment. It operates by creating a penalty matrix to assess the cost of aligning each pair of characters in the input sequences. The traceback matrix is then used to backtrack and determine the optimal alignment.

Functions:

minPenalty(a, b, c) : Returns the minimum value among three given integers (a, b, c).

getPenalty(a, b, c) : Calculates the penalty for aligning two characters. Returns 0 for a match and the specified mismatch penalty otherwise.

printAlignmentSeq1(sequence1, i, j) : Recursively prints the aligned sequence for the first input sequence, considering the traceback matrix.

printAlignmentSeq2(sequence2, i, j) : Recursively prints the aligned sequence for the second input sequence, considering the traceback matrix.

calculateAlignment(sequence1, sequence2, gapPenalty, mismatchPenalty) : This function calculates the penalty matrix and traceback matrix for global sequence alignment.

SYSTEM REQUIREMENTS:

Processor: Any modern multi-core processor, preferably dual-core or higher.

RAM: Minimum 4 GB RAM for small to medium-sized instances; additional RAM for larger instances.

Operating System: Compatible with Windows, Linux, macOS.

IDE: Supports standard C++ development, e.g., Visual Studio, Code::Blocks, Dev-C++, Xcode, Visual Studio Code.

CODE IMPLEMENTATION

```
#include<bits/stdc++.h>
using namespace std;

const int MAX_ROWS = 50;
const int MAX_COLS = 50;

int penaltyMatrix[MAX_ROWS][MAX_COLS], gapPenalty, mismatchPenalty, minimumPenalty;
int tracebackMatrix[MAX_ROWS][MAX_COLS];

int minPenalty(int a, int b, int c)
{
    return min(min(a, b), c);
}

int getPenalty(char a, char b, int c)
{
    return (a == b) ? 0 : c;
}

void printAlignmentSeq1(string sequence1, int i, int j)
{
    if (i > 0 && j > 0 && tracebackMatrix[i][j] == 0)
    {
        printAlignmentSeq1(sequence1, i - 1, j - 1);
        cout << sequence1[j - 1];
    }
    else if (i > 0 && tracebackMatrix[i][j] == 1)
    {
        printAlignmentSeq1(sequence1, i - 1, j);
        cout << '_';
    }
    else if (j > 0 && tracebackMatrix[i][j] == 2)
    {
        printAlignmentSeq1(sequence1, i, j - 1);
        cout << sequence1[j - 1];
    }
}

void printAlignmentSeq2(string sequence2, int i, int j)
{
    if (i > 0 && j > 0 && tracebackMatrix[i][j] == 0)
    {
        printAlignmentSeq2(sequence2, i - 1, j - 1);
        cout << sequence2[i - 1];
    }
    else if (i > 0 && tracebackMatrix[i][j] == 1)
    {
        printAlignmentSeq2(sequence2, i - 1, j);
        cout << sequence2[i - 1];
    }
    else if (j > 0 && tracebackMatrix[i][j] == 2)
    {
        printAlignmentSeq1(sequence2, i, j - 1);
        cout << '_';
    }
}

void calculateAlignment(string sequence1, string sequence2, int gapPenalty, int
mismatchPenalty)
{
    for (int i = 0; i <= sequence2.length(); i++)
    {
        for (int j = 0; j <= sequence1.length(); j++)
        {
            if (i == 0 && j == 0)
            {
                penaltyMatrix[i][j] = 0;
            }
            else if (i == 0)
            {
                penaltyMatrix[i][j] = penaltyMatrix[i][j - 1] + gapPenalty;
                tracebackMatrix[i][j] = 2;
            }
            else if (j == 0)
            {
                penaltyMatrix[i][j] = penaltyMatrix[i - 1][j] + gapPenalty;
                tracebackMatrix[i][j] = 1;
            }
            else
            {
                int diagonalPenalty = penaltyMatrix[i - 1][j - 1] +
getPenalty(sequence2[i - 1], sequence1[j - 1], mismatchPenalty);
                int verticalPenalty = penaltyMatrix[i - 1][j] + gapPenalty;
                int horizontalPenalty = penaltyMatrix[i][j - 1] + gapPenalty;

                penaltyMatrix[i][j] = minPenalty(diagonalPenalty, verticalPenalty,
horizontalPenalty);

                if (penaltyMatrix[i][j] == diagonalPenalty)
                {
                    tracebackMatrix[i][j] = 0;
                }
                else if (penaltyMatrix[i][j] == verticalPenalty)
                {
                    tracebackMatrix[i][j] = 1;
                }
                else
                {
                    tracebackMatrix[i][j] = 2;
                }
            }
        }
    }
}
int main()
{
    cout << "\t\t _________________________________________________________________\n";
    cout << "\t\t|\t\t\t\t\t\t\t\t  |\n";
    cout << "\t\t|\t\t Welcome to Sequence Alignment Problem    \t  |\n";
    cout <<
"\t\t|_________________________________________________________________|\n\n";

    string sequence1, sequence2;

    cout << "Enter Sequence X : ";
    cin >> sequence1;
    cout << "Enter Sequence Y : ";
    cin >> sequence2;
    cout << "Enter Gap Penalty , P(gap) : ";
    cin >> gapPenalty;
    cout << "Enter Mismatch Penalty, P(xy) : ";
    cin >> mismatchPenalty;

    calculateAlignment(sequence1, sequence2, gapPenalty, mismatchPenalty);

    cout << endl;
        cout << "+-------------------+\n";
    cout << "|Total Penalty : " <<
penaltyMatrix[sequence2.length()][sequence1.length()] << "  |\n";
```

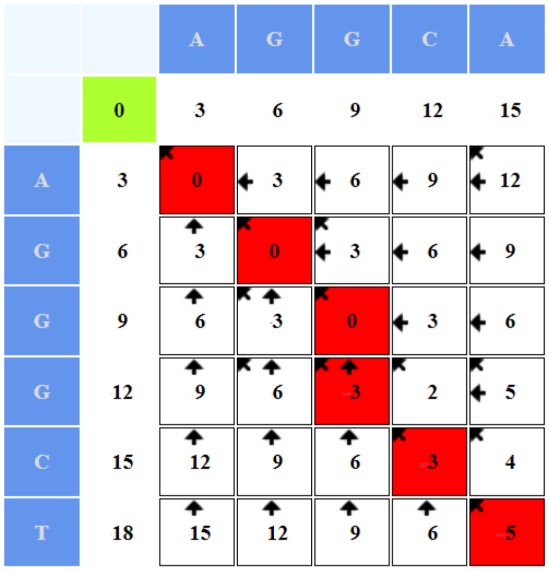
```
        cout << "+-------------------+\n";

    cout << "_____________________________________________\n";

            cout << "+-------------------------------------------+\n";
    cout << "|Alignment for Sequence X: ";
    printAlignmentSeq1(sequence1, sequence2.length(), sequence1.length());
    cout << "\n";
            cout << "+-------------------------------------------+\n";
            cout << "+-------------------------------------------+\n";

    cout << "|Alignment for Sequence Y: ";
    printAlignmentSeq2(sequence2, sequence2.length(), sequence1.length());
   cout << "\n";
            cout << "+-------------------------------------------+\n";
    return 0;
}
```

DEMO TABLE



		A	G	G	C	A
	0	3	6	9	12	15
A	3	↖ 0	← 3	← 6	← 9	↖ ← 12
G	6	↑ 3	↖ 0	↖ ← 3	← 6	← 9
G	9	↑ 6	↖ ↑ 3	↖ 0	← 3	← 6
G	12	↑ 9	↖ ↑ 6	↖ ↑ -3	↖ 2	↖ ← 5
C	15	↑ 12	↑ 9	↑ 6	↖ -3	↖ 4
T	18	↑ 15	↑ 12	↑ 9	↑ 6	↖ -5

TESTING RESULTS

```
 _________________________________________________________________
|                                                                 |
|              Welcome to Sequence Alignment Problem              |
|_________________________________________________________________|

Enter Sequence X : AGGGCT
Enter Sequence Y : AGGCA
Enter Gap Penalty , P(gap) : 3
Enter Mismatch Penalty, P(xy) : 2

+--------------------+
|Total Penalty : 5   |
+--------------------+
_____________________________________________
+---------------------------------------------+
|Alignment for Sequence X: AGGGCT
+---------------------------------------------+
+---------------------------------------------+
|Alignment for Sequence Y: A_GGCA
+---------------------------------------------+
```

LIMITATIONS

The current implementation uses a simple penalty scoring model for mismatches and gaps. It assumes a constant penalty for both, which may not accurately represent biological or real-world scenarios where different types of mismatches or gaps could have varying impact on sequence alignment.

FUTURE SCOPE

For larger sequences, parallelization techniques could be implemented to enhance the efficiency of the algorithm. Parallel computing can significantly reduce the time complexity of sequence alignment, allowing for the alignment of larger datasets or multiple sequences simultaneously.

OPTIMIZATION:

The program employs dynamic programming to optimize the solution to the Sequence Alignment Problem.

The time complexity of this algorithm is O(n*m), where n is the length of sequence1 and m is the length of sequence2. This is because there are two nested loops that iterate through the lengths of both sequences.

CONCLUSION:

In simple terms, the code is effective at aligning sequences, but it has some limitations. The way it scores matches and gaps is basic, and it struggles with certain types of gaps. To improve it, we could use more advanced scoring methods that better capture the complexity of sequences. Additionally, making it work faster for large sets of data is a potential enhancement. Overall, while the code lays a foundation, there's room for refining its accuracy and efficiency.