Needleman - Wunsch Algorithm

Data Structure & Algorithm II

CSE 2218 Sec: D
Group-1

Abdullah Al Jobair
Lecturer, Department of CSE
United International University

Iftekhar Mahmud	011182073
Fahad Al Islam	011183070

Needleman-Wunsch Algorithm

The Needleman Wunsch Algorithm is used in Bioinformatics to align protein or nucleotide sequences.

- It is one of the first applications of **DP** to compare biological sequences.
- It was developed by Saul B. Needleman and Christian D. Wunsch and published in 1970.

Needleman-Wunsch Algorithm

So in short,

- It is a **Dynamic Programming** Algorithm.
- It efficiently computes the **Optimal Alignment** between two sequences based on a scoring scheme.
- Proved to be one for the most fundamental tools in comparing biological sequences

Alignment Approaches

There are 2 types of alignment approaches:

- Local Alignment: Smith-Waterman
- Global Alignment: Needleman-Wunsch

NWA guarantees finding the Globally Optimal Alignment by considering all possible alignments and choosing the one with the highest score.

Algorithm Steps

Step: 1 Initialization:

- The algorithm creates a matrix (known as Scoring Matrix/ DP matrix)
- Each cell stores the score of a possible alignment between subsequences of the input sequences.

	-	A	G	C	G	A
-	0					
A						
C						
G						
A						
A						

Algorithm Steps

Step: 2 Filling the Matrix:

The matrix is filled iteratively by considering three possible operations:

Match/Mismatch: Compare the characters in the current positions of the sequences. Assign a score based on whether they match or mismatch.

Gap Opening: Introduce a gap in one of the sequences, penalizing the gap opening with a certain score.

Gap Extension: Extend an existing gap in one of the sequences, adding to the gap penalty.

$$V_{i,j} = \max \begin{cases} V_{i-1,j} + \delta(s_i, -) \\ V_{i,j-1} + \delta(-, t_j) \\ V_{i-1,j-1} + \delta(s_i, t_j) \end{cases}$$

Algorithm Steps

		i=0	i=1	i=2	i=3	i=4	i=5
		-	A	G	C	G	A
j=0	-	0					
j=1	A						
j=2	C						
j=3	G						
j=4	A						
j=5	A						

$$V_{i,j} = \max \begin{cases} V_{i-1,j} + \delta(s_i, -) \\ V_{i,j-1} + \delta(-, t_j) \\ V_{i-1,j-1} + \delta(s_i, t_j) \end{cases}$$

1. V (i,j) = V(i-1,j) + σ()

   V (4,3) = V(3,3) + σ()

2. V (i,j) = V(i,j-1) + σ()

   V (4,3) = V(4,2) + σ()

3. V (i,j) = V(i-1,j-1) + σ()

   V (4,3) = V(3,2) + σ()

Algorithm Steps

Scoring Scheme:

The **NWA** relies on a scoring scheme that assigns values to matches, mismatches, gap openings, and gap extensions.

Match: For every match we will give **+1**

Mismatch: For every mismatch we will give **-1**

Gap: For every gap we will give **-2**

Algorithm Steps

Match = +1
Mismatch = -1
Gap = -2

		i=0	i=1	i=2	i=3	i=4	i=5
		-	A	G	C	G	A
j=0	-	0	-2	-4	-6	-8	-10
j=1	A	-2	-1	-3	-5	-7	-9
j=2	C	-4	-1	-2	-4	-4	-6
j=3	G	-6	-3	-2	-3	-5	-5
j=4	A	-8	-5	-2	-1	-3	-4
j=5	A	-10	-7	-4	-3	0	-2

$$T_{(I,j)} = \max \begin{cases} T_{(i-1,\,j-1)} + \sigma\,(S1_{(i)},\, S2_{(j)}) \\ T_{(i-1,j)} + \text{gap penalty} \\ T_{(I,j-1)} + \text{gap penalty} \end{cases}$$

T(1,0)

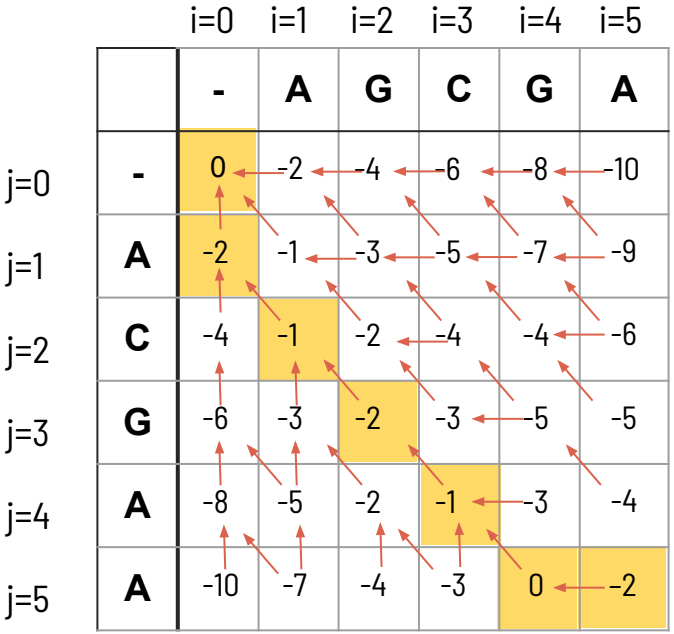
T (1-1,0-1) = T(0,-1) + σ(S1(i),S2(j)) = x

T (1-1,0) = T(0,0) + σ(S1(i),S2(j)) = 0 + (-2) = -2

T (1,0-1) = T(1,-1) + σ(S1(i),S2(j)) = x

Backtracking for Alignment:

Once the matrix is filled, the algorithm backtracks from the bottom-right corner to the top-left corner to reconstruct the optimal alignment. This provides insight into which characters are matched, mismatched, or aligned with gaps.




```
- T G G T G    S1
  |   | |
A T C G T -    S2
```

The Pseudocode

First to construct a 2D matrix F :

```
d ← Gap penalty score
for i = 0 to length(A)
    F(i,0) ← d * i
for j = 0 to length(B)
    F(0,j) ← d * j
for i = 1 to length(A)
    for j = 1 to length(B)
    {
        Match ← F(i−1, j−1) + S(Ai, Bj)
        Delete ← F(i−1, j) + d
        Insert ← F(i, j−1) + d
        F(i,j) ← max(Match, Insert, Delete)
    }
```

Once a matrix F is completed, The entry F_{nm} gives maximum score among all possible alignments. Now to complete the alignment that actually gives the score :

```
AlignmentA ← "Place your sequence A"
AlignmentB ← "Place your sequence B"
i ← length(A)
j ← length(B)
while (i > 0 or j > 0)
{
   if (i > 0 and j > 0 and F(i, j) == F(i−1, j−1) + S(Ai, Bj))
   {
      AlignmentA ← Ai + AlignmentA
      AlignmentB ← Bj + AlignmentB
      i ← i − 1
      j ← j − 1
   }
   else if (i > 0 and F(i, j) == F(i−1, j) + d)
   {
      AlignmentA ← Ai + AlignmentA
      AlignmentB ← "−" + AlignmentB
      i ← i − 1
   }
   else
   {
      AlignmentA ← "−" + AlignmentA
      AlignmentB ← Bj + AlignmentB
      j ← j − 1
   }
}
```

Full code in cpp and explanation :
https://github.com/iftekhar-mahmud/Needleman-Wunsch/tree/main

A problem solved with Needleman-Wunsch Algorithm

Question : Using the Needleman and Wunsch dynamic programming method outlined below, construct the alignment score matrix for the following two sequences, using the following scoring parameters: match score = +1, mismatch score = -1, gap penalty = -2.
AGTACGCA and TATGC

Simulation :

		A	G	T	A	C	G	C	A
	0	-2	-4	-6	-8	-10	-12	-14	-16
T	-2	-1	-3	-3	-5	-7	-9	-11	-13
A	-4	-1	-2	-4	-2	-4	-6	-8	-10
T	-6	-3	-2	-1	-3	-3	-5	-7	-9
G	-8	-5	-2	-3	-2	-4	-2	-4	-6
C	-10	-7	-4	-3	-4	-1	-3	-1	-3

Result :

A	G	T	A	C	G	C	A
-	-	T	A	T	G	C	-

Results and Application of Needleman-Wunsch Algorithm

Needleman-Wunsch algorithm tries to achieve the best global alignment. It is widely used for optimal global alignment when quality of the alignment is of the utmost importance. But this algorithm is expensive in **Time and Space complexity** which is **O(mn)**. It is not suitable for long sequences.
This algorithm is mostly used in the field of bioinformatics and computational biology for sequential alignment. Some of the real world applications for this algorithm are :

1. **DNA and RNA Sequence Comparison**: The algorithm is used to compare DNA and RNA sequences, which is crucial for understanding genetic variations, identifying mutations, and detecting functional elements within the sequences.
2. **Phylogenetic Analysis**: The Needleman-Wunsch algorithm is used in constructing phylogenetic trees, which depict evolutionary relationships between different organisms or species. Sequence alignment helps identify similarities and differences in genetic material, which in turn aids in inferring the evolutionary history of different species.
3. **Drug Discovery**: Sequence alignment is employed in drug discovery to identify potential drug targets and design molecules that can interact with specific proteins. By aligning protein sequences and identifying conserved regions, researchers can pinpoint potential drug-binding sites.
4. **Functional Annotation**: Researchers use sequence alignment to predict the functions of genes and proteins. If a new sequence shares high similarity with a known sequence, it's likely that they have similar functions.
5. **Metagenomics**: Metagenomics involves analyzing the genetic material in environmental samples. Sequence alignment helps identify and classify the genetic content of diverse microorganisms present in a sample.
6. **MicroRNA Target Prediction**: MicroRNAs are small RNA molecules that regulate gene expression by binding to target sequences in messenger RNAs. The Needleman-Wunsch algorithm can be used to predict potential microRNA target sites within mRNA sequences.

In essence, the Needleman-Wunsch algorithm plays a fundamental role in bioinformatics and molecular biology, enabling researchers to uncover insights into genetic, evolutionary, and functional aspects of biological sequences.

```
//Coded by Iftekhar Mahmud
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

int match_score = 1;
int mismatch_score = -1;
int gap_penalty = -2;

int max3(int a, int b, int c) {
    return max(a, max(b, c));
}
//max3 is a helper function that returns the maximum of three integers a, b, and c using the max function from the <algorithm> library.
void needleman_wunsch(const string& sequence1, const string& sequence2) {
    int rows = sequence1.length() + 1;
    int cols = sequence2.length() + 1;

    vector<vector<int>> matrix(rows, vector<int>(cols, 0));
//needleman_wunsch is the main function implementing the Needleman-Wunsch algorithm. It takes two input sequences as strings. The function initializes the number of rows and columns for the matrix based on the lengths of the sequences.
    //A 2D vector named matrix is created with dimensions (rows, cols) and initialized with zeros. This matrix will store the scores for each alignment position.
    // Initialize the first row and column
    for (int i = 1; i < rows; ++i) {
        matrix[i][0] = matrix[i - 1][0] + gap_penalty;
    }
    for (int j = 1; j < cols; ++j) {
        matrix[0][j] = matrix[0][j - 1] + gap_penalty;
    }
    //These loops initialize the first row and first column of the matrix using gap penalties. The scores are calculated based on the scores from adjacent cells in the matrix.
```

```
// Fill in the matrix
   for (int i = 1; i < rows; ++i) {
      for (int j = 1; j < cols; ++j) {
         int match = matrix[i - 1][j - 1] + (sequence1[i - 1] == sequence2[j - 1] ? match_score : mismatch_score);
         int deleteOp = matrix[i - 1][j] + gap_penalty;
         int insertOp = matrix[i][j - 1] + gap_penalty;
         matrix[i][j] = max3(match, deleteOp, insertOp);
      }
   }
  //These nested loops fill in the rest of the matrix using the Needleman-Wunsch dynamic programming approach. For each cell (i, j), three possible operations are considered: match/mismatch, delete (gap in sequence1), and insert (gap in sequence2).
  //The match score is calculated based on the previous diagonal cell score, either adding match_score for a match or mismatch_score for a mismatch.
  //The deleteOp score is based on the score above the current cell, adding the gap penalty.
  //The insertOp score is based on the score left of the current cell, adding the gap penalty.
  //The maximum of these three scores is taken using the max3 function and stored in the current cell (i, j).
    // Backtrack to find the optimal alignment
    string alignment1 = "";
    string alignment2 = "";
    int i = rows - 1;
    int j = cols - 1;
    while (i > 0 && j > 0) {
       if (matrix[i][j] == matrix[i - 1][j - 1] + (sequence1[i - 1] == sequence2[j - 1] ? match_score : mismatch_score)) {
          alignment1 = sequence1[i - 1] + alignment1;
          alignment2 = sequence2[j - 1] + alignment2;
          --i; --j;
       } else if (matrix[i][j] == matrix[i - 1][j] + gap_penalty) {
          alignment1 = sequence1[i - 1] + alignment1;
          alignment2 = "-" + alignment2;
          --i;
       } else {
          alignment1 = "-" + alignment1;
          alignment2 = sequence2[j - 1] + alignment2;
          --j;
       }
    }
```

```
//This loop performs the traceback step to reconstruct the optimal alignment by backtracking through the matrix from the bottom-right corner (rows-1, cols-1) to the top-left corner (0, 0).
//Depending on the maximum score in the current cell, the algorithm determines whether the optimal path involved a match/mismatch, a gap in sequence1, or a gap in sequence2. The corresponding characters are added to alignment1 and alignment2.

    while (i > 0) {
        alignment1 = sequence1[i - 1] + alignment1;
        alignment2 = "-" + alignment2;
        --i;
    }
    while (j > 0) {
        alignment1 = "-" + alignment1;
        alignment2 = sequence2[j - 1] + alignment2;
        --j;
    }

    cout << "Alignment 1: " << alignment1 << endl;
    cout << "Alignment 2: " << alignment2 << endl;
}
//After the traceback, any remaining characters in either sequence are added to the alignments, representing gaps.
//The final alignments are printed to the console.
int main() {
    string sequence1 = "AGTACGCA";
    string sequence2 = "TATGC";
    needleman_wunsch(sequence1, sequence2);

    return 0;
}
//The main function initializes the input sequences and calls the needleman_wunsch function with them.
//It returns 0 to indicate successful execution.
```

Thank you