

**CSE246: Algorithms (Section 2)**

**Spring 2024**

**Project Report**

**Sequence Alignment Problem**

**Submitted by:**

|  |  |  |
| --- | --- | --- |
| **Student ID** | **Student Name** | **Contribution Percentage** |
| 2022-3-60-109 | Sheikh Sarafat Hossain | 34% |
| 2022-3-60-192 | Rijia Parveen Raya | 33% |
| 2021-3-60-191 | Mashfia Zaheen | 33% |

**Problem Statement**

The project focuses on the problem of sequence alignment, a fundamental task in bioinformatics and computational biology. Sequence alignment aims to arrange two sequences of characters (commonly DNA, RNA, or protein sequences) in a way that highlights their similarities and differences. This alignment helps in identifying regions of similarity that may indicate functional, structural, or evolutionary relationships between the sequences. The objective of this project is to align two given sequences in a manner that minimizes the alignment penalty, which accounts for gaps and mismatches between the sequences.

**System Requirements**

**Processor:** Any modern processor such as Intel Core i3 or higher, or AMD equivalent, which ensures efficient computation.

**RAM:** A minimum of 4 GB RAM is recommended to handle the dynamic programming table used for sequence alignment.

**Operating System:** The project can be run on various operating systems including Windows, macOS, and Linux, providing flexibility in the development and execution environment.

**IDE:** Any Integrated Development Environment (IDE) that supports C++ can be used. Examples include Visual Studio, Code::Blocks, and CLion. These IDEs offer tools and features that assist in writing, debugging, and testing the code efficiently.

**System Design**

The design of the sequence alignment system is based on dynamic programming, a method that efficiently computes the alignment by breaking down the problem into simpler subproblems. The main components of the system design include:

**1. Initialization:** A dynamic programming (DP) table is created to store the penalties for different alignments. The table is initialized with gap penalties.

**2. DP Table Filling:** The DP table is filled based on penalties for matches, mismatches, and gaps, using predefined gap (‘Pgap = 2’) and mismatch (‘Pxy = 3’) penalties.

**3. Traceback:** After filling the DP table, a traceback procedure is used to reconstruct the optimal alignment by following the path of minimum penalties from the bottom-right corner of the DP table to the top-left corner.

**Implementation**

The implementation of the sequence alignment algorithm in C++ involves several key steps:

**1. Defining Penalties:**

A close up of a text

Description automatically generated

**2. DP Table Initialization:**

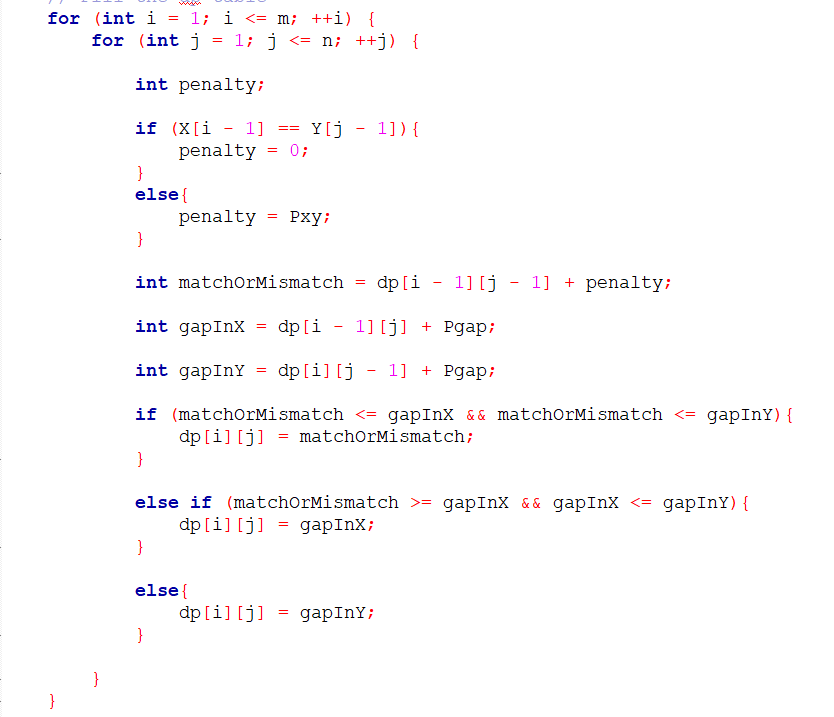
The DP table is initialized to account for the penalties incurred by aligning a sequence with gaps.

A white background with black and red text

Description automatically generated

**3. DP Table Filling:**

The DP table is filled by calculating the minimum penalty for aligning each prefix of the sequences.



**4. Traceback to Find Aligned Sequences:**

The traceback process reconstructs the optimal alignment by tracing back through the DP table.

A screenshot of a computer code

Description automatically generated

A computer screen shot of a code

Description automatically generated

**Time Complexity**

The time complexity of the sequence alignment algorithm is an important consideration, as it determines the feasibility of using the algorithm for large sequences. The algorithm's complexity is determined by the process of filling the DP table and performing the traceback.

**DP Table Filling:** The DP table has dimensions ‘(m + 1) x (n + 1)’, where ‘m’ and ‘n’ are the lengths of the two input sequences. Filling each cell in the table involves a constant amount of work (comparing characters and computing minimum values), leading to an overall time complexity of **‘O (m \* n)’**.

**Traceback:** The traceback process involves traversing the DP table from the bottom-right corner to the top-left corner, which takes **‘O (m + n)’** time.

Therefore, the overall time complexity of the sequence alignment algorithm is dominated by the DP table filling step, resulting in a time complexity of **‘O (m \* n)’**. This quadratic time complexity makes the algorithm suitable for moderate-sized sequences, but optimizations or different approaches may be needed for very large sequences.

**Testing Results**

The testing phase involved running the sequence alignment algorithm with sample input sequences to validate its correctness and efficiency.

A black screen with white text

Description automatically generated

**Future Scope**

The current implementation of the sequence alignment project is limited to basic cases with predefined gap and mismatch penalties. However, there are several potential enhancements and extensions that can be pursued:

**Affine Gap Penalties:** Implementing affine gap penalties, which apply different penalties for opening and extending gaps, to more accurately model biological sequence alignments.

**Optimizing for Large Sequences:** Optimizing the algorithm to handle larger sequences efficiently, potentially through parallel processing or memory optimization techniques.

**Graphical User Interface (GUI):** Developing a graphical interface to facilitate user interaction and visualization of the alignment results, making the tool more accessible and user-friendly.

**Integration with Databases:** Integrating the tool with biological sequence databases to allow for automated fetching and alignment of sequences from various sources.

**Advanced Scoring Matrices:** Incorporating advanced scoring matrices like PAM or BLOSUM for protein sequence alignment, providing more biologically relevant alignments.

These future developments can significantly enhance the functionality, performance, and usability of the sequence alignment tool, making it a valuable resource for researchers and practitioners in bioinformatics.