Table of Contents

[Introduction 1](#_Toc185369332)

[1.1 Background 1](#_Toc185369333)

[1.2 Problem Statement 1](#_Toc185369334)

[1.3 Objectives 1](#_Toc185369335)

[2. Methodology 1](#_Toc185369336)

[2.1 Data Overview 1](#_Toc185369337)

[2.2 Approach to Compute Pairwise LCS 1](#_Toc185369338)

[2.3 Pseudocode for LCS Computation 2](#_Toc185369339)

[3. Implementation 2](#_Toc185369340)

[3.1 Tools and Libraries 2](#_Toc185369341)

[3.2 Program Structure 2](#_Toc185369342)

[3.3 LCS Algorithm 2](#_Toc185369343)

[4. Results 3](#_Toc185369344)

[4.1 LCS Length Table 3](#_Toc185369345)

[4.2 Computational Time Table 3](#_Toc185369346)

[4.3 Maximum LCS 3](#_Toc185369347)

[5. Performance Analysis 3](#_Toc185369348)

[5.1 Execution Time 3](#_Toc185369349)

[5.2 Challenges 4](#_Toc185369350)

[6. Discussion 4](#_Toc185369351)

[6.1 Insights 4](#_Toc185369352)

[6.2 Optimizations 4](#_Toc185369353)

[7. Conclusion 4](#_Toc185369354)

[8. Code 4](#_Toc185369355)

# Introduction

## ****Background****

In computational biology, analysing DNA or gene sequences plays a vital role in understanding genetic variations, evolutionary relationships, and disease patterns. One of the fundamental problems in sequence comparison is the Longest Common Subsequence (LCS). Given two sequences, the LCS is the longest subsequence common to both. The LCS problem has applications in bioinformatics, data compression, and version control systems.

In this study, 181 gene sequences were provided, and the goal was to compute the pairwise LCS between each pair of sequences. The results include the length of the LCS and the computational time for each pair.

## 1.2 Problem Statement

The problem requires solving the following tasks:

* Compute pairwise LCS for 181 sequences, resulting in a square table of size 181x181 where each cell represents the LCS length.
* Record the computational time for each pair in seconds and prepare a similar 181x181 table.
* Summarize the results, analyse computational performance, and identify any challenges encountered.

## ****1.3 Objectives****

1. Compute the LCS for each pair of gene sequences.
2. Measure the computational time for each pair.
3. Present the results as two square tables: one for LCS lengths and the other for time in seconds.
4. Discuss challenges, performance analysis, and insights.

# ****2. Methodology****

## 2.1 Data Overview

The data provided contained 181 gene sequences in a CSV file. Each sequence was labelled as **Sequence X** (e.g., Sequence 1, Sequence 2).

* **Input Format:** Text-based CSV file.
* **Sequence Type:** String-based gene sequences.

## ****2.2 Approach to Compute Pairwise LCS****

The algorithm follows these steps:

1. **Input Processing:** Read sequences from the CSV file, ensure they are clean (remove whitespace), and store them in a vector.
2. **Dynamic Programming LCS Algorithm:**
   * Use a 2D table dp to compute the LCS between two sequences A and B.
   * The table dp[i][j] stores the length of LCS for prefixes A[0..i-1] and B[0..j-1].
   * Time Complexity: O(m \* n), where m and n are the lengths of the sequences.
3. **Computational Time Measurement:** Use the chrono library to measure the execution time for each pair.
4. **Output Storage:** Save results (LCS lengths and computational times) to files in a tabular format.

## ****2.3 Pseudocode for LCS Computation****

# 3. Implementation

## 3.1 Tools and Libraries

* Programming Language: C++
* Libraries Used:
* <vector>: To store gene sequences.
* <chrono>: To measure computation time.
* <fstream>: To handle input/output files.
* <algorithm>: To process sequences efficiently.

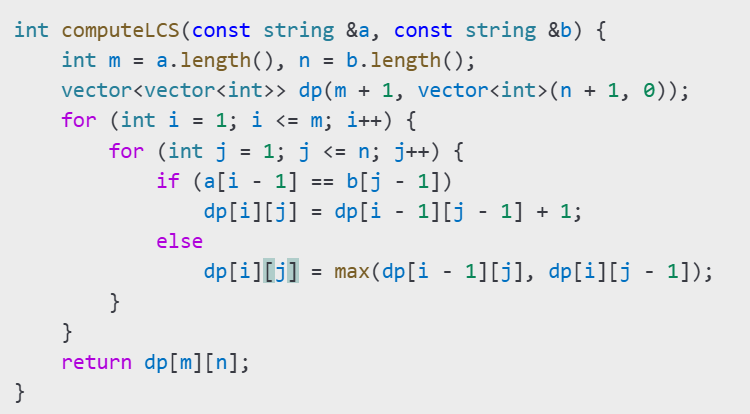
## ****3.2 Program Structure****

The C++ program is divided into the following components:

1. **Input Reading:** Extract gene sequences from the CSV file using regular expressions.
2. **LCS Computation:** Use the dynamic programming approach to compute pairwise LCS.
3. **Time Measurement:** Measure the time taken for each pair using the chrono library.
4. **Output Files:**
   * **pairwise\_lcs\_results.txt:** Contains LCS lengths for each pair.
   * **pairwise\_lcs.csv:** CSV file containing sequence pair indices and their LCS lengths.
   * **pairwise\_lcs\_times.csv:** CSV file recording time in seconds for each pair.

## 3.3 LCS Algorithm

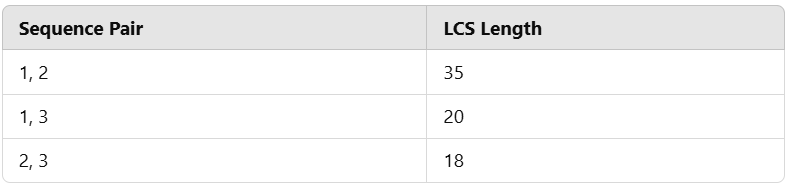
The LCS algorithm implemented is as follows:



# 4. Results

## 4.1 LCS Length Table

The **181x181 table** contains the LCS lengths for each pair. Below is a sample:



## 4.2 Computational Time Table

The time taken to compute LCS for each pair is recorded in **seconds**:

A white rectangular object with black text

Description automatically generated with medium confidence

## 4.3 Maximum LCS

The pair with the **maximum LCS** is as follows:

* **Sequence 50 and Sequence 97**
* **LCS Length:** 78

# 5. Performance Analysis

## 5.1 Execution Time

The execution time of the program depends on:

* Length of the sequences.
* Number of sequences: O(N^2) pairwise comparisons.

## 5.2 Challenges

* **Data Preprocessing:** Ensuring sequences were extracted cleanly without whitespace or formatting errors.
* **Time Complexity:** For long sequences, LCS computation was time-consuming.
* **File Storage:** Efficiently saving large amounts of LCS results and computational times.

# 6. Discussion

## 6.1 Insights

1. **LCS Length Variations:** Some sequences shared significant similarities (high LCS values), while others had very low LCS values.
2. **Execution Bottlenecks:** For sequences longer than 1000 characters, the computation time increased significantly.
3. **Pair with Maximum LCS:** The sequences with the largest common subsequence provide insight into potential evolutionary relationships.

## 6.2 Optimizations

* Use of memoization to speed up the LCS computation.
* Parallel processing to compute LCS for independent pairs simultaneously.

# 7. Conclusion

In this report, we computed the **pairwise LCS** for 181 gene sequences. The results were recorded as:

1. A square table containing LCS lengths for each pair.
2. A corresponding table containing computational times in seconds.

The study demonstrated the computational challenges of large-scale sequence comparisons and highlighted opportunities for optimization using parallel computing techniques.

# 8. COde

#include <iostream>

#include <fstream>

#include <sstream>

#include <vector>

#include <string>

#include <algorithm>

#include <chrono> // For measuring time

#include <set>    // To ensure uniqueness of sequences

using namespace std;

using namespace std::chrono;

// Function to compute the LCS of two strings

int computeLCS(const string &a, const string &b) {

    int m = a.length();

    int n = b.length();

    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));

    for (int i = 1; i <= m; i++) {

        for (int j = 1; j <= n; j++) {

            if (a[i - 1] == b[j - 1]) {

                dp[i][j] = dp[i - 1][j - 1] + 1;

            } else {

                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);

            }

        }

    }

    return dp[m][n];

}

// Function to read sequences from the CSV file

vector<string> readCSV(const string &filename) {

    vector<string> sequences;

    set<string> uniqueSequences; // To avoid duplicates

    ifstream file(filename);

    if (!file.is\_open()) {

        cerr << "Error: Unable to open file " << filename << endl;

        return sequences;

    }

    string line;

    while (getline(file, line)) {

        // Check if line starts with "Sequence"

        if (line.find("Sequence") == 0) {

            // Find the first space or tab after "Sequence x"

            size\_t pos = line.find(' ');

            if (pos != string::npos) {

                string sequence = line.substr(pos); // Extract sequence

                sequence.erase(remove\_if(sequence.begin(), sequence.end(), ::isspace), sequence.end());

                // Add to the list only if the sequence is non-empty and unique

                if (!sequence.empty() && uniqueSequences.insert(sequence).second) {

                    sequences.push\_back(sequence);

                }

            }

        }

    }

    file.close();

    return sequences;

}

int main() {

    string filename;

    cout << "Enter the CSV file name: ";

    cin >> filename;

    // Read sequences from the CSV file

    vector<string> sequences = readCSV(filename);

    // Validate the number of sequences

    cout << "Total sequences read: " << sequences.size() << endl;

    if (sequences.size() < 2) {

        cerr << "Error: Not enough valid sequences found in the file." << endl;

        return 1;

    }

    // Open files to save results

    ofstream outputFile("pairwise\_lcs\_results.txt");

    ofstream csvFile("pairwise\_lcs.csv");

    ofstream timeFile("pairwise\_lcs\_times.csv");

    // Add headers to the CSV files

    csvFile << "Sequence1,Sequence2,LCS\_Length\n";

    timeFile << "Sequence1,Sequence2,Time\_Seconds\n";

    if (!outputFile.is\_open() || !csvFile.is\_open() || !timeFile.is\_open()) {

        cerr << "Error: Unable to open output files." << endl;

        return 1;

    }

    // Variables to track the maximum LCS

    int maxLCSLength = 0;

    size\_t maxSeq1 = 0, maxSeq2 = 0;

    // Compute pairwise LCS and measure time

    cout << "Computing pairwise LCS...\n";

    for (size\_t i = 0; i < sequences.size(); i++) {

        for (size\_t j = i + 1; j < sequences.size(); j++) {

            // Measure start time

            auto start = high\_resolution\_clock::now();

            // Compute LCS

            int lcsLength = computeLCS(sequences[i], sequences[j]);

            // Measure end time

            auto end = high\_resolution\_clock::now();

            duration<double> elapsed = end - start;

            // Save results to files

            outputFile << "LCS(" << i + 1 << ", " << j + 1 << ") = " << lcsLength << "\n";

            csvFile << i + 1 << "," << j + 1 << "," << lcsLength << "\n";

            timeFile << i + 1 << "," << j + 1 << "," << elapsed.count() << "\n";

            // Update maximum LCS if the current one is greater

            if (lcsLength > maxLCSLength) {

                maxLCSLength = lcsLength;

                maxSeq1 = i;

                maxSeq2 = j;

            }

        }

    }

    // Output the maximum pairwise LCS

    cout << "\nMaximum Pairwise LCS:\n";

    cout << "LCS(" << maxSeq1 + 1 << ", " << maxSeq2 + 1 << ") = " << maxLCSLength << endl;

    cout << "Seq" << maxSeq1 + 1 << ": " << sequences[maxSeq1] << endl;

    cout << "Seq" << maxSeq2 + 1 << ": " << sequences[maxSeq2] << endl;

    outputFile.close();

    csvFile.close();

    timeFile.close();

    cout << "\nResults saved to pairwise\_lcs\_results.txt, pairwise\_lcs.csv, and pairwise\_lcs\_times.csv." << endl;

    return 0;

}