Longest Common Subsequence (LCS) Visualizer

A Qt-based GUI application that visualizes the Longest Common Subsequence algorithm using dynamic programming. This project implements the algorithm described in Chapter 14 of "Introduction to Algorithms" by Cormen et al. (4th edition).

Algorithm Overview

The Longest Common Subsequence (LCS) problem is a classic computer science problem that finds the longest subsequence present in two given sequences. A subsequence is a sequence that appears in the same relative order but not necessarily contiguous.

Dynamic Programming Approach

The solution uses dynamic programming with the following key components:

1. **DP Table Structure**:
    - A 2D table `dp[i][j]` where:
        - `i` represents the position in sequence X
        - `j` represents the position in sequence Y
        - `dp[i][j]` stores the length of LCS of X[1..i] and Y[1..j]
2. **Recurrence Relation**:

```
if X[i-1] == Y[j-1]:
    dp[i][j] = dp[i-1][j-1] + 1
else:
    dp[i][j] = max(dp[i-1][j], dp[i][j-1])
```

3. **Direction Table**:
    - A 2D table `direction[i][j]` that stores the path information:
        - 'D' (Diagonal): Characters match, move diagonally
        - 'U' (Up): Take value from top cell
        - 'L' (Left): Take value from left cell

4. **Time Complexity**: O(mn) where m and n are lengths of input sequences

5. **Space Complexity**: O(mn) for storing the DP and direction tables

How the Dynamic Programming Algorithm Works

1. Problem Decomposition

The LCS problem is solved by breaking it down into smaller subproblems:

- For each position (i,j) in the sequences, we find the LCS of the prefixes X[1..i] and Y[1..j]
- The solution to the original problem is found in dp[m][n], where m and n are the lengths of X and Y

2. Base Cases

- dp[0][j] = 0 for all j (empty first sequence)
- dp[i][0] = 0 for all i (empty second sequence)

3. Filling the DP Table

For each position (i,j), we have three cases:

a. **Characters Match** (X[i-1] == Y[j-1]):

   - We can extend the LCS by including this matching character
   - dp[i][j] = dp[i-1][j-1] + 1
   - Direction = 'D' (diagonal)

b. **Characters Don't Match** (X[i-1] != Y[j-1]):

   - We take the maximum of two possibilities:
     1. Skip character from X: dp[i-1][j]
     2. Skip character from Y: dp[i][j-1]
   - Direction = 'U' if dp[i-1][j] ≥ dp[i][j-1], else 'L'

4. Example Walkthrough

For sequences X = "ABCBDAB" and Y = "BDCABA":

1. Initialize table with zeros
2. For each position:
   - If characters match (e.g., 'B' at X[1] and Y[0]):
     - dp[1][0] = dp[0][-1] + 1 = 1

        - direction[1][0] = 'D'
    - If characters don't match:
        - Take max of left and top values
        - Set direction accordingly

5. Solution Reconstruction

To find the actual LCS:

1. Start at dp[m][n]
2. Follow the direction table:
    - 'D': Include character, move diagonally
    - 'U': Move up
    - 'L': Move left
3. Stop when reaching dp[0][0]

6. Visualization in the Application

The application shows this process through:

- Step-by-step filling of the DP table
- Direction arrows indicating the path
- Highlighting of current computation
- Final LCS reconstruction

Implementation Details

Core Components

1. **LCS Class** (`lcs.h`, `lcs.cpp`):

```
class LCS {
    // Input sequences
    string X, Y;

    // DP tables
    vector<vector<int>> dp;         // Value table
    vector<vector<char>> direction; // Direction table

    // Key methods
    void compute();                 // Main computation
    string getLCS();               // Solution reconstruction
};
```

2. **GUI Implementation** (`mainwindow.h`, `mainwindow.cpp`):
    - Input handling for sequences
    - Dynamic visualization of DP table
    - Animation of computation steps
    - File I/O operations

Key Features

1. **Interactive Input**:
    - Direct text input for sequences
    - File loading/saving support
    - Real-time sequence display
2. **Visualization**:
    - Step-by-step animation of DP table filling
    - Direction arrows showing the path
    - Adjustable animation speed
    - Clear highlighting of current computation
3. **Solution Reconstruction**:
    - Backtracking using direction table
    - Visual path showing the LCS construction
    - Final LCS display

Building and Running

Requirements

- Qt 5.x or later
- C++ compiler with C++11 support
- CMake 3.10 or later

Build Instructions

```
mkdir build
cd build
cmake ..
make
```

Usage

1. Launch the application
2. Enter sequences or load from file
3. Click "Compute LCS"
4. Watch the visualization
5. Adjust speed if needed
6. View the final LCS result

File Format

Input/output files use a simple text format:

```
sequence_x
sequence_y
```

License

This project is open source and available under the MIT License.