Assignment 7
Due: Nov 21, 2023 at 11:59PM

1 Assignment 7 Specification

1.1 Goals

1. Practice implementing a dynamic programming solution

1.2 Implementation

The skeleton code at `https://classroom.github.com/a/_J9Xt7b-` is the start of a program to perform sequence alignment. It is used in applications such as BLAST for DNA comparison, and `diff` for comparing lines of two text files. Two text files are given as input arguments and the output should be the best alignment of the two strings. A simple depth-first search is provided, but it performs very poorly on larger inputs. Use dynamic programming to improve on the current solution's runtime. (DFS will only run on `text/a.txt` and `text/b.txt`, not any of the other file pairs.)

There is one algorithm already implemented:

findLCSdfs Recursively searches through each string for character matches.

For this assignment, you will implement the **findLCSdyn** method (and any helper methods you want) in **A7.java**. Your method should return an array with two aligned strings, for example:

```
BEN-T--
--NOTES
```

Use the '-' character to represent a gap. That character will not show up in any input text. In addition to returning an array with the two aligned string values, you should also update the **A7** class variable `longest` with the optimal number of overlapping characters in a solution.

The algorithm in the book shows how to reconstruct only the subsequence characters, but you should modify this algorithm to build the aligned strings with gaps in them. You can do this by adding a gap to one string and letter to the other string whenever their current characters don't match. (So: if you move up, prepend the row's character to one string and a dash to the other. If you move left, prepend a dash to the first string and the column's character to the second string.)

Note: Java's default stack size is not very large, so I recommend implementing iterative methods rather than recursive ones.

1.3 Testing

For this assignment, JUnit tests are provided in the **edu.wit.cs.comp2350.tests** package. The tests check if alignments of varying sizes work correctly. For grading this assignment, I will use some different strings in addition to the ones supplied. Make sure you test your code on different inputs for potential edge cases.

A **ChartMaker** class is included, which will create a chart of runtimes of dfs vs. dynamic algorithms. You can use this chart to verify that your dynamic programming solution runtime is what you expect.

Note: some Windows IDEs automatically change text line endings. If you are passing all of the tests except for the `znc` ones, you should change the znc text files to UNIX line endings and they should work correctly.

2 Grading

LCS length: 35%

LCS string: 65%