BLG 336E - Analysis of Algorithms II , Spring 2015

Project 3

Total Worth : 10% of your grade

Handed Out : 16.04.2015 Thursday

Due : 30.04.2015 Thursday - 23:00

Overview

The Damerau-Levenshtein edit distance metric is used to measure the orthographic difference between two strings. The most common application is in spell checkers, which compare a misspelled word with any number of well-typed words (comprising the list of correction candidates), and accept the well-typed word with the smallest edit distance to the misspelled word as the most likely correction candidate.

You will implement a dynamic programming algorithm to compute the minimum Damerau-Levenshtein edit distance between two strings. In the memoization matrix, the row with index *i* and the column with index *j* will respectively correspond to the sub-strings containing the first *i* characters of the misspelled word and the first *j* characters of the correction candidate. The cell at (*i*, *j*) will contain the minimum edit distance between the corresponding sub-strings. The minimum edit distance between a misspelled word of length *m* and a correction candidate of length *c* will be contained in the cell at (*m*, *c*).

Edit distance assigns the following costs to the given set of basic edit operations:

The **deletion** (e.g. *wrd* vs. *w**o**rd*) operation has a cost of 2.

The **insertion** (e.g. *w**po**rd* vs. *w**o**rd*) operation has a cost of 2.

The **substitution** (e.g. *w**p**rd* vs. *w**o**rd*) operation has a cost of 3.

The **transposition** (e.g. *w**ro**d* vs. *w**or**d*) operation has a cost of 2.

For instance, the minimum edit distance between the misspelled string ***wros*** and the correction candidate ***word*** in the minimum-cost setup corrects the substitution of the character ***s*** for the character ***d*** (cost=3) and the transposition of the letters ***o*** and ***r*** (cost=2), yielding a minimum edit distance of 3+2=5.

Your implementation must be in accordance with the bottom-up dynamic programming methodology, initializing trivial distances and then gradually computing distances between pairs of sub-strings in terms of the distances between smaller sub-strings that were previously computed and stored.

Before you get started on your implementation of the algorithm, you must first model the trivial distances and the **deletion**, **insertion**, **substitution** and **transposition** operations into an optimization *(OPT)* function. This function will be used in your implementation to determine new cell values in the memoization matrix in terms of previously computed cell values.

You will need to provide a mathematical representation for this optimization function in your report, and implement the algorithm in accordance with the same function.

Code (50 points)

Implement the edit distance algorithm in C++ exactly as defined in the project overview section. Write your program in a class called **EditDistance** encapsulating the necessary methods and attributes related to the algorithm.

Your class must have a **compare(…)** method that takes two string values as arguments (representing the misspelled word and the correction candidate) and returns an integer value (denoting the minimum edit distance between the strings).

You must also include a static **main(…)** method in your code expecting two extra strings as command line arguments. Invoking the program from the command line should process these strings and print the minimum edit distance between them.

Your program should compile and run using the following commands:

```
g++ yourStudentID.cpp -o project3
```

```
./project3 wros word
```

Your program will be tested with different input words of various lengths. You should make sure the program runs properly with any pair of word, otherwise you might not get points for the code.

Report (50 points)

Q1) [10 points] Provide a piecewise function representation for your optimization *(OPT)* function in your report. Take care to fully specify all preconditions.

Q2) [10 points] What parameters does the complexity of your implementation depend on? How would you represent the worst case complexity in big O notation?

Q3) Suppose you have a lexicon (word list) of well-typed words of size ***N***, and you would like to implement a spell checker using this resource. In order to generate a correction for a given misspelled word, you decide to compare it to each word in the lexicon, and then propose the word with the smallest minimum edit distance to the misspelled word as your correction.

a) [5 points] How would you represent the complexity of generating a correction for a single misspelled word in big O notation?

b) [10 points] Give one example of a modification in your implementation in order to reduce run time.

Q4) [15 points] Do you have to keep the whole memoization matrix in memory for the entire course of the edit distance algorithm? What could you adjust in your implementation to preserve memory?

Submission

You should be aware that the Ninova system clock may not be synchronized with your computer, watch, or cell phone. Do not e-mail the teaching assistant or the instructors your submission after the Ninova site submission has closed. If you have submitted to Ninova once and want to make any changes to your report, you should do it before the Ninova submission system closes. Your changes will not be accepted by e-mail. Connectivity problems to the Internet or to Ninova in the last few minutes are not valid excuses for being unable to submit. You should not risk leaving your submission to the last few minutes. After uploading to Ninova, check to make sure that your project appears there.

Policy: You may discuss the problem addressed by the project at an abstract level with your classmates, but you should not share or copy code from your classmates or from the Internet. **You should submit your own, individual project.** Plagiarism and any other forms of cheating will have serious consequences, including failing the course.

Submission Instructions: Please submit your homework through Ninova. Please zip and upload all your files using filename HW3_studentID.rar. In the archived file, you must include your completed report_StudentId file and all your program and header files.

All your code must be written in C++, and we must be able to compile and run on it on ITU's Linux Server (you can access it through SSH) using g++. You should supply one yourStudentID.cpp file that calls necessary routines for all questions (Multiple files are acceptable, as long as you state the compilation instructions in your report).

When you write your code, try to follow an object-oriented methodology with well-chosen variable, method, and class names and comments where necessary**. Your code must compile without any errors; otherwise, you may get a grade of zero on the coding part of the assignment.**

If a question is not clear, please let the teaching assistant know by email (sulubacak@itu.edu.tr).