Homework 3, CSCE 240, Fall 2014

Overview

The algorithms to compute "edit distance" are some of the most important in all of computing. The basic algorithm has been invented many times and goes by many names (Needleman-Wunsch, Wagner-Fischer, etc.). The edit distance computation is what is done in the DNA sequence comparison and the longstanding Unix command `diff`.

(You are encouraged to Google such things as edit distance, longest common subsequence, diff, Needleman-Wunsch, and Wagner-Fischer.)

The idea in edit distance is to compute the cost of transforming one sequence of tokens into another. In DNA sequence comparison, the tokens are individual letters A, C, G, T, of nucleic acids. In this homework assignment, the tokens are individual words (and we can assume no capital letters, no punctuation, etc.).

The cost is based on

- the cost of inserting one token into a string, or of deleting one token.
- the cost of substituting one token for another.

For this exercise, and in many computations we assume

- cost of insertion or deletion = 1
- the cost of substituting one token for another = 2 (that is, a substitution can be viewed as a deletion followed by an insertion)

For example, the cost of transforming the sentence
this is the first sentence
into the sentence
this is the second sentence
is 2, since we must substitute "second" for "first".

The dynamic programming algorithm for edit distance for this problem would compute the following matrix.

	(blank)	this	is	the	first	sentence
(blank)	0	1	2	3	4	5
this	1	0	1	2	3	4
is	2	1	0	1	2	3
the	3	2	1	0	1	2
second	4	3	2	1	2	3
sentence	5	4	3	2	3	2

Going across row zero, or down column zero, the costs are obvious. The cost of changing a blank sequence into "this" is clearly 1, the cost of inserting one word. The cost of changing a blank sequence into "this is" is 2. And so on.

Beyond row 0 and column 0, in the (row, col) location, we look at

$$valueUp = cost[row - 1][col] + 1,$$

$$valueLeft = cost[row][col - 1] + 1,$$

and

$$valueDiagonally = cost[row - 1][col - 1]$$

or

$$valueDiagonally = cost[row - 1][col - 1] + 2,$$

depending on whether the token at the top of column col is the same as the token at the left of row row or not.

Going across a row, the cost of moving from one column to the next one by inserting the token at the top of the column is 1.

Going down a column, the cost of moving from one row to the next by inserting the token at the left of the row is 1.

Going diagonally down one row and one column, the cost is unchanged if the tokens match, and 2 if they are different.

The cost we store in the (row, col) location is the minimum of $valueUp$, $valueLeft$, and $valueDiagonally$.

We start the algorithm by initializing row zero and column zero. We then proceed down the backward diagonals and fill in the values. Thus

- (1,1) based on (1,0), (0,1), and (0,0)
- (1,2) based on (1,1), (0,2), and (0,1)
- (2,1) based on (2,0), (1,1), and (1,0)

- (1,3) based on (1,2), (0,2), and (0,2)
- (2,2) based on (2,1), (1,2), and (1,1)
- (3,1) based on (3,0), (2,1), and (2,0)
- and so forth.

At the end, the value at the bottom right corner is the minimum cost to transform the entire first sequence into the second sequence.

A second example is this, from your sample input and output.

	XX	this	is	the	first	line	XX	XX
XX	**0**	1	2	3	4	5	6	7
this	1	**0**	1	2	3	4	5	6
is	2	1	**0**	1	2	3	4	5
the	3	2	1	**0**	1	2	3	4
second	4	3	2	**1**	2	3	3	4
and	5	4	3	**2**	3	4	4	4
final	6	5	4	3	**4**	5	5	5
line	7	6	5	4	5	**4**	5	6

In this example, I have used the dummy string XX to indicate the blank, and I have padded the shorter string with blanks to make the computation a little simpler.

This Assignment

You are to write an edit distance program.

This program is largely an issue of proper subscripting. The computation is trivial. The subscripting is not so trivial.

There are several simplifications I suggest.

First, I suggest you pad the shorter string with a dummy string (rather than blanks) and then test for the dummy. This will make it unnecessary for you to keep track of which sequence is longer, because they will both be the same length.

I suggest using a defined constant for the dummy string.

You could do this without using a blank row at the top and a blank column at the left. But putting those in also simplifies the row subscripting since your "subscript minus one" value is less likely to be out of bounds.

There is a second phase to this algorithm that you do not have to implement. That is to start at the end (the 4 at the bottom right corner in the second example) and work backwards up to the upper left. This second phase is how you determine the sequence of edits necessary to do the conversion from one string into the other.