Edit Distance

The problem of finding an edit distance between two strings is as follows:

Given an initial string s, and a target string t, what is the minimum number of chances that have to be applied to s to turn it into t. The list of valid changes are:

1) Inserting a character
2) Deleting a character
3) Changing a character to another character.

In initially looking for a recursive solution, you may think that there are simply too many recursive cases. We could insert a character in quite a few locations! (If the string is length n, then we can insert a character in n+1 locations.) However, the key observation that leads to a recursive solution to the problem is that ultimately, the last characters will have to match. So, when matching one word to another, on consider the last characters of strings s and t. If we are lucky enough that they ALREADY match, then we can simply "cancel" and recursively find the edit distance between the two strings left when we delete this character from both strings. Otherwise, we MUST make one of three changes:

1) delete the last character of string s

2) delete the last character of string t

3) change the last character of string s to the last character of string t.

Also, in our recursive solution, we must note that the edit distance between the empty string and another string is the length of the second string. (This corresponds to having to insert each letter for the transformation.)

So, an outline of our recursive solution is as follows:

1) If either string is empty, return the length of the other string.

2) If the last characters of both strings match, recursively find the edit distance between each of the strings without that last character.

3) If they don't match then return 1 + minimum value of the following three choices:

- **a) Recursive call with the string s w/o its last character and the string t**
- **b) Recursive call with the string s and the string t w/o its last character**
- **c) Recursive call with the string s w/o its last character and the string t w/o its last character.**

Now, how do we use this to create a DP solution? We simply need to store the answers to all the possible recursive calls. In particular, all the possible recursive calls we are interested in are determining the edit distance between prefixes of s and t.

Consider the following example with s="hello" and t="keep". To deal with empty strings, an extra row and column have been added to the chart below:

		h	e	l	l	o
	0	1	2	3	4	5
k	1	1	2	3	4	5
e	2	2	1	2	3	4
e	3	3	2	2	3	4
p	4	4	3	3	3	4

An entry in this table simply holds the edit distance between two prefixes of the two strings. For example, the highlighted square indicates that the edit distance between the strings "he" and "keep" is 3. In order to fill in all the values in this table we will do the following:

1) Initialize values corresponding to the base case in the recursive solution. These are all the values dealing with edit distances with the empty string. (They are the first row and first column inside the table.)

2) Loop through the table from the top left to the bottom right. In doing so, simply follow the recursive solution.

If the characters you are looking at match, store the number in the square diagonally up and left from the square you are filling in. This square holds the edit distance between the two strings w/o their last character.

If the characters don't match, look that the square to your left, above you, and the square diagonally up and left of the one you are filling in. Take the minimum of these and add 1. This corresponds exactly to the recursive call, except that instead of making it, you just look it up in the table.

Finally we can reconstruct the path as follows:

		h	e	l	l	o
	0	1	2	3	4	5
k	1	1	2	3	4	5
e	2	2	1	2	3	4
e	3	3	2	2	3	4
p	4	4	3	3	3	4

Start at the bottom right corner of the auxiliary array. Compare the corresponding characters. If they match, automatically go up the diagonal and do not edit anything. If they don't match, as in the case of 'o' and 'p', the look at the three squares we mentioned before: up, left, and up&left. If any of these is one less than the current square value, go to that square. (I chose one of the two possible choices.) Then make the edit that corresponds to this choice. For the choice above, that means deleting the o:

hello -> hell

From here on, the edit path dictated is

hello->hell->help->heep->keep.