CS 499 (Fall 2006)- Assignment 5
Due: Thursday, 10/05/2006

(1) Write a program that serves as a combined spell-checker and text formatter. Here is what it should do: it gets a piece of text (say, from a file). It first spell-checks it, using a dictionary you provide as a file. All words not in the dictionary should be automatically replaced with the closest match from your dictionary, using the edit distance discussed in class (you can break ties for closest match any way you want). You do not need to have a full table for different penalties for different character overwrites (as discussed in class), but you should allow reasonably easy adjustments to the relative penalties of overwrites, deletes, and inserts.

After "correcting" the spelling, the program should format the resulting text to fit in c columns as neatly as possible. c can be a command-line parameter, or read in some other natural way. "As neatly as possible" means that the row lengths are roughly balanced. Specifically, if the number of trailing whitespaces in line i is t_i for each i, then you incur a penalty of $\sum_i t_i^2$. For instance, if you want to format "Algorithms are way cool" to 14 columns, you could do either of the following:

```
Algorithms are
way cool

Algorithms
are way cool
```

The first incurs a penalty of $0^2 + 6^2 = 36$, while the second has penalty $4^2 + 2^2 = 20$. So the second one is a better way of displaying this text.

You are supposed to find the best output with respect to this measure, and write it to a new file. Of course, if a single word doesn't fit into c characters, there is nothing you can do, so you will always have to put it on a line by itself. You cannot break a word across lines.

Your program should be able to deal at least with some minimal amount of punctuation, and it should respect paragraphs. If two paragraphs are separated by an empty line, then they should still be separated by an empty line after your re-formatting.

Here are some hints:

1. The first part requires getting your hands on a dictionary, i.e., a list of words. It doesn't really matter where you get it, or what it contains. You probably don't want a full dictionary of all English words, or your program may be a bit slow. Feel free to reduce the size, e.g., by random sampling. Also, you will probably have a lot more fun with your "spell-checked" output if your dictionary is missing many words.
2. The second part is probably as substantial as the first. In particular, it does require dynamic programming to do optimally. Notice for instance in the simple example given above that squeezing as much as possible into each line need not be optimal.
3. There are many ways to improve this program beyond what is specified. As discussed in class, having different penalties for different replacements (or perhaps even different deletions or inserts), or adding character swaps, makes the spell-checking better. If you have enough free time on your hands, you are welcome to implement them for your own fun, but it will not affect your grade.

(2) Let us look now at some ways we could improve the two parts we discussed above:

1. We already discussed in class different replacement costs, pairwise swaps, etc. What other ways do you see to improve a spell checker? For instance, is there a way to get a spell checker to perform better if it is always used by the same user? How? What ways would help in general?

2. The proposed approach does a decent job of ensuring that rows are all "roughly equally full". Perhaps, some function other than $\sum_i t_i^2$ would do an even better job, but that's not so much our concern here. Of course, having all rows roughly equally full may not really be the best way to make the text look nice or be readable. Think of some ways to improve upon the algorithm. What rules would you recommend for making text layout more reader-friendly? Discuss how easily your rules would combine with your solution to the problem, i.e., how easily you think they could actually be implemented.