Instructions

- Download the following files from Sakai **Resources → Project4:** wordlist.txt, test1.txt, test2.txt, test3.txt, test4.txt, test5.txt. [If you are working on *epiphone* you can copy these files from /tmp to your own folder.] You'll need *wordlist.txt* to create a "dictionary" in your program. You'll use test[12345].txt as test files for your program.

Overview

- Read and think about Exercise 6.4, p.178 of the text. This project is a variation of that exercise.
- The idea for this project is that your program will read a file containing only lower case characters and no spaces or punctuation. It will need to implement a dynamic programming solution that will either (a) reconstitute a possible sequence of words based on words in a "dictionary", or (b) report that the string cannot be reconstituted into a sequence of words from the "dictionary".
- A dynamic programming solution will avoid repeated evaluation of the same subtask. As stated in the textbook, your algorithm should run in time proportional to O($|\text{input}|^2$)

Details

- STEP 0 – Before you write any code answer the following questions. Your answers to these questions should be submitted with your program:
  - What are the subproblems? What does a "solution" look like?
  - Identify a recurrence that shows how to solve a subproblem given the solutions to "smaller" subproblems.
- STEP I – The textbook Exercise describes a Boolean function dict(*w*) to efficiently determine whether or not *w* is in the dictionary of words. I'll leave the implementation up to you – but do not use sequential search to determine if *w* is in the dictionary. You can use a hash table, the C++ map class, a trie, binary search, or some other efficient search technique.
- STEP II – develop an algorithm and a program to determine if the input can be broken down into a sequence of words from the dictionary. The goal for this part is simply to be able to answer Yes or No (or True or False), not to report the words themselves.
- STEP III – For a particular input, if the answer from Part I is Yes (True), then generate a list of the words in the input. Note that in many cases the result may not be exactly the same as the original text. For example "`itrainedtoday`" could have originally been "i trained today" or "it rained to day".

Grading

- 10% - Answers to questions in STEP 0.
- 50% - Implementation of a dynamic programming solution to correctly determine if the input string could have been created from a list of words in the dictionary. Program doesn't fail if the answer is No.
- 20% - If the input is composed of dictionary words, program efficiently lists the words that could have been in the original string.
- 10% - Program is readable and well organized.
- 10% - Program is well-documented.