CS 180 Programming Project 2

Due Date: June 2, 2009

Description: For this assignment, you have been given a file with the spaces removed. Your goal is to recover the words, add the proper spaces back, and output it in a reasonable format. The file has no punctuation and does not necessarily have any meaning.

Input: Your program takes 3 command-line parameters. The first is the dictionary file. The second is the name of the spaces-removed text file. Both files will reside in the same directory as the executable. The third parameter is the maximum number of characters per output line.

Your input dictionary will be a text file containing one word per line. Only the words in this file are deemed valid words for purposes of this assignment.

The second file was once a readable text document, but has since had the spaces removed. Each line was once a paragraph, but is now consecutive letters. This file is only English letters (capital and lowercase) and line breaks.

Assignment Part I: In this part, you will determine where the spaces belong, for use in the next part. Your goal here is to break the input paragraphs into word breaks. You do *not* need to make any determination about the meanings of sentences, nor should you. The file might be sentence fragments or simply a random assortment of words.
If the input were to be "theyouthevent", then both "the you the vent" and "the youth event" would be considered equally valid spacings, assuming that the dictionary contained the words {the, you, the, vent, youth, event}.
Note: ignore capitalization when determining words. EvEnT, evENT, and event are all the same words. Your output should have the same case for each letter as was input.

Assignment Part II: Output the text document, complete with spaces. Your goal here is to minimize the cost of outputting the paragraph, subject to the following:

- All lines must fit within the margin specified in the command line parameters.
- One space should be inserted between two consecutive words on the same line.
- A word may not be split between two lines.
- Do not put spaces after the last word on each line.
- The last line of each paragraph has zero cost
- The cost to output any other line is the *cube* of the number of extra spaces at the end of the line. That is, the margin minus the total characters output on the line.

Once you have determined the output pragraphs - words, spacing, and returns - output them to standard out. This is `cout` in C++ and `System.out` in Java. After each paragraph, output the cost of the paragraph as a single integer on a line by itself. Do not output anything other than the paragraphs and their costs for this program.

Regardless of other considerations, you *must* output an integer on a line by itself between paragraph. The grading script will interpret this as both the stated cost of the above paragraph and as the end of a paragraph of output.

Submission Requirements and Grading Criteria

- Any code you submit must be an original piece of code. All ideas must come from the textbook, course staff, or project members. As students taking an upper-division computer science course, we expect you to understand and adhere to UCLA's standards of academic honesty. Course staff may compare all submitted code this quarter to code submitted by other students this quarter and to code submitted in previous quarters by using an automated plagiarism detection tool. Any instances of academic dishonesty will be reported according to UCLA policy. High level discussion of concepts is acceptable, but sharing of code is not.
- Your code should be reasonably neat and well organized.
- Your code must compile on the SEAS Windows machines in such a way that typing "`Printing X Y Z`", or your programming language's equivalent, at the command (DOS) prompt will run the program. Any programs that do not meet this requirement will receive no credit.
- Please include a `README.txt` with your submission that includes:
  - The names and ID numbers of everyone in your group (up to 2 students per group).
  - A description of the algorithms you used to break up the text document into words and to determine where the line breaks in the output should be. Your algorithms for this part must be based on dynamic programming.
  - A brief description of the status of your implementation. If your code compiles but does not run correctly, please let us know what it *does* do correctly.
- Part of your grade is correctness:
  - Does your program correctly insert spaces into the document?
  - Does your program correctly do the paragraph printing?
  - Does your program output the paragraphs and their costs - and nothing else?
- Part of your grade is timing and efficiency. Your runtime must be reasonable. My Part I is $\Theta(n^3)$ and Part II is $\Theta(n^2)$, both with small constants. My runtime is well under a minute per test case. As long as the actual running time is reasonable, asymptotic runtime is negotiable.

Submissions are due via CourseWeb by 11:59PM on June 2. Submissions may be made up to 5 days late at a cost of 10 points per day. If you have a compelling reason to submit your assignment late, please let Michael know via email so that alternate arrangements can be made.

Please submit a single ZIP file containing:

- All source code.
- If you are submitting in C++, include project files that will compile your code to `Printing.exe`.
- If you are using Java or Python, please ensure that main is in `Printing.java` or `Printing.py`
- Your `README.txt`.
- Do not include any pre-compiled code (executables, `.class` files, etc)