Introduction to Algorithms	**Problem Set 4 (3 problems)**
CS 4820 Fall 2018	**Due 11:59pm Thursday, September 20**

Your homework submissions need to be typeset (hand-drawn figures are OK). See the course web page for suggestions on typing formulas.

The solution to each question needs to be uploaded to CMS as a separate pdf file. To help provide anonymity in your grading, do not write your name on the homework (CMS will know it's your submission). Remember that when a problem asks you to design an algorithm, you must also prove the algorithm's correctness and analyze its running time. The running time must be bounded by a polynomial function of the input size.

Some Comments on Dynamic Programming Problems. Proving dynamic programming algorithms correct requires a number of components. If your solution consists of a dynamic programming algorithm, you must clearly specify the set of sub-problems you are using, and the recurrence you are using - describing what they mean in English as well as any notation you define. You must also explain *why* your recurrence leads to the correct solution of the sub-problems; this is the heart of a correctness proof for a dynamic programming algorithm. Finally, you should describe the complete algorithm that makes use of the recurrence and sub-problems. A description of a DP algorithm consisting of a piece of pseudo-code without these explanations will not get full credit.

(1) Pretty Printing. (Extension of Exercise 6.6) In a word processor, the goal of "pretty printing" is to take text with a ragged right margin, like this:

```
Call me Ishamel.
Some years ago -
never mind how long precisely -
having little or no money in my purse,
and nothing particular to interest me on shore,
I thought I would sail about a little
and see the watery part of the world.
```

and turn it into text with the most "even" right margin possible, like this:

```
Call me Ishamel. Some years ago - never mind how
long precisely - having little or no money in my
purse, and nothing particular to interest me on
shore, I thought I would sail about a little and
see the watery part of the world.
```

To make this precise enough for us to start to think about how to write a pretty-printer, we need to figure out what it means for the right margin to be "even." Suppose the text consists of n words (or *tokens*) $W = \{w_1, w_2, \ldots, w_n\}$ where w_i has c_i characters (with $c_i > 0$). We assume each character has the same width, and that punctuation is treated the same way as alphanumeric characters (a period at the sentence would be a character in the word preceding it, for instance).

A *formatting* of W consists of a partition of words W into *lines*. In the words assigned to a single line, there should be a space after each word except the last on the line, so for the line ℓ, with words $w_j, \cdots, w_k$, the number of characters would be

$$C_\ell = c_k + \sum_{i=j}^{k-1}(c_i + 1).$$

An assignment of words to a line is *valid* if the number of characters is less than the position of the right margin L, i.e. $C_\ell \leq L$. The number of spaces between the end of the characters in the line and the right margin, $L - C_\ell$, will be called the *slack* of the line.

Given an ordered list of words W and a right margin position L, find an efficient algorithm to partition the words into valid lines such that the sum of the *squares* of the slacks of the lines—not including the last line—is minimized. (This choice of loss function reflects that we prefer to have a number of lines with small slack to a few lines with a lot of slack.) You may assume that a valid printing exists, i.e. that no word is longer than L.

(2) Counting Shortest Paths. Consider a directed graph $G = (V, E)$ where the cost of edge e, c_e, can be negative. Given two nodes s and t, we have seen in class how to compute the length of the min-cost path in G assuming G has no negative cost cycles. In this problem, assume that all cycles have strictly positive costs. This problem concerns two extensions of this problem:

Suppose you have found a new apartment in Ithaca, and you want to assess how long your commute would be from there before you sign a lease. You could initially use the algorithm above to measure the shortest path from your apartment to class, but because of the frequency of construction in Ithaca, you also want to make sure that there is more than one shortest path available to work. In this case, in addition to the length (cost) of the shortest path from s to t, you would also like to know how many path there are from s to t of this shortest length.

(a) *(3 points)* Previously, we computed the length of the shortest path from s to v using at most i edges and stored the value in $OPT(i, v)$. Here, consider the min-cost path using *exactly* i edges (rather than at most i edges). Give a polynomial time algorithm to find the minimum cost path from s to t using *exactly* i edges.

(b) *(5 points)* In addition to the length of the shortest path with i edges, also compute the number of different shortest paths with the same length that use exactly i edges. (There can be exponentially many shortest paths, so outputting them all may not be possible in polynomial time.)

(c) *(2 points)* Show how to compute the total number of shortest paths, this time using any number of edges.

(3) Graph Coloring (probability review). Next week we'll start on randomized algorithms: taking advantage of random selection in algorithm design. Recall that probability (conditional probability, expectation, variance) was discussed in CS 2800, which is a pre-requisite of this course. For a quick review of probability you may want to read Sections 13.1 and 13.3. We will also generate a quick summary/reminder by Monday of things you need to remember about probabilities, and two of the TAs will run an optional session to help you review the probability basics we need.

Consider an undirected graph G on n nodes. G is called k-colorable if there is a way to color the n nodes with k colors so that the two ends of each edge get different colors. For example, a cycle is always 3-colorable, but only even cycles are 2-colorable. As we will see latter in the course, deciding if a graph is 3-colorable is computationally hard. Instead, we can aim to color the nodes with 3 colors, so that as few edges get identical colors as possible. In this problem, we test out a naive algorithm to do this: choose a color for each node uniformly at random among the three options, with the color sampled independently for each node.

(a) *(3 points)* For a given edge $e = (v, w)$, what is the probability that the two ends of edge e get the same color?

(b) *(3 points)* Assuming G has m edges (that is $m = |E|$), what is the expected number of edges that are colored properly (with the two ends of each edge different colors), and what is the expected number of edges that are badly colored?

(c) *(4 points)* Show that the probability that more than half of the edges are badly colored is less than 67%.