Juan Ruelas SID 861116223

CS141 Assignment 3
due Friday, June 3

Solution 1: Consulting Firm

A

If our moving cost $M = 10$ and the number of operational months $n = 4$, then we have the table below to analyze.

	Month1	Month2	Month3	Month4
NY	1	3	20	30
SF	50	20	2	4

We are given the optimal plan already, however

B

C

D

Solution 2: Pretty Print

The entire basis of this problem is to be able to take some text that is "not balanced" and turn it into text whose right margin is as even as possible. Look below to see what I mean.

```
Call me Ishmael.
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.
```

$\rightarrow$

```
Call me Ishmael. 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.
```

In order to accomplish this we will need to make use of dynamic programming. Here is the overview of how we will make use of this programming technique:

- Find a recurrence relation for the optimal solution
- Based on our recurrence relation, create an algorithm to solve our problem

A: Recurrence Relation

In order to come up with a recurrence relation we need to understand what it is we are exactly computing. We are trying to re-arrange the text such that the "slack" or amount of spaces from the last word of every line are evenly distributed among the entirety of the text. This becomes a trivial task until we define what "even" means. Let us assume that "even" means to minimize the sum of all the "slacks". If we were to take this approach we would be left with several different viable solutions. See below for more details. Assume we have a max row width of 10.

```
        SOLUTION 1                               SOLUTION 2

Line   0123456789     Slack             Line   0123456789    Slack
---------------------------             --------------------------
 0     Ruelas      -->  4                0     Ruelas     -->   4
 1     Juan is my  -->  0        VS      1     Juan is    -->   3
 2     name        -->  6                2     my name    -->   3

     4 + 0 + 6 = 10                          4 + 3 + 3 = 10
```

As you can see from the figure above, since we define "even" to be the minimization of all the slacks of every line, we will have multiple "optimal" solutions. This is a problem since we can reproduce identical slack summations with different text patterns and as we can visually see, the pattern on the right appears to be more "even" than the left one. Because of this, we will define "even" to mean the summation of all the $slacks^2$. This will enable us to be greedy with our spaces and will force us to minimize the amount of slack on *every* line. Look below to see what I mean.

SOLUTION 1: $4 + 0 + 6 = 10$ VS $4^2 + 0^2 + 6^2 = 52$
SOLUTION 2: $4 + 3 + 3 = 10$ VS $4^2 + 3^2 + 3^2 = 34$

From this we can see that the optimal solution which minimizes the amount of slack on every line is the second solution. We will use this property of squaring the slack values to aid us in creating a recurrence relation. Essentially we will compute the minimum $slack^2$ for all combinations of words that fit within our row width (accounting for spaces where needed) and then find the "least cost slack" solution of each sub problem to determine our optimal solution for the pretty print.

Recurrence Relation: $OPT[n] = \min_{1 \leq j \leq n} S_{i,n}^2 + OPT[j-1])$

B: The Algorithm

In order to begin the algorithm we need to understand what the recurrence relation itself is doing. It is returning to us the optimal solution on a subset of the word list W, and returns to us the least cost word arrangement which minimizes the amount of spaces used on a given line. We then use this to pre-compute all our values. Once we finish, we apply the same recurrence relation to our pre-computed values to find the optimal solution. We can break it down into 3 steps.

- Given our word list W, we generate another array A of equal length L, but instead of every value being a word, it will be the length of the word.
- Generate a $slack^2$ matrix (accounting for spaces where needed) from A of size $L \times L$
- Find the optimal solutions to our sub problems, keeping track of each solution and "marking" our array for line breaks (aka our traceback)

Look at the pseudo code below

```
S = slack squared 2d matrix
OPT[0] = 0
#traverse the 2d array to find our traceback
for k in range(0, len(S)):
        for j in range(0, len(S[k]):
                # find the optimal solution within a given row
                if (OPT[k] > ( S[j][k] + OPT[j - 1] )) :
                        OPT[k] = S[j][k] + OPT[j - 1]

return OPT[len(S)]
```

Solution 3: