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	5	10	5	10
SF	10	5	10	5

We know the solution should be NY, NY, NY, NY with a total cost of $30, but instead the solution given by the algorithm would be NY, SF, NY, SF and would cost $50.

B

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	100	1	100	1
SF	1	100	1	100

In every month (excluding the first month) it is more profitable to change locations.

C

Lets say that we are operating for a total of n months. There are only two outcomes by recursively calculating the cost. We will either end up in NY or in SF. Given that there are only two possible outcomes, let us denote the recurrence relations of both locations as PN and PS

We can then denote the optimal cost at any given month, i, from 1 to n as

$$\text{PN[i]} = N_i + min\ \{\ \text{PN[i - 1]},\ \text{M} + \text{PS[i - 1]}\}$$
$$\text{PS[i]} = S_i + min\ \{\ \text{PS[i - 1]},\ \text{M} + \text{PN[i - 1]}\}$$

D

```
OptimalSolution:
        PN[1] = N1
        PS[1] = S1
        for i in range (2, n):
                PN[i] = Ni + min {PN[i - 1], M + PS[i - 1]}
                PS[i] = Si + min {PS[i - 1], M + PN[i - 1]}
        # After computing, return the traceback
        return TraceBack(PN, PS) # this returns the optimal plan
```

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.	→	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

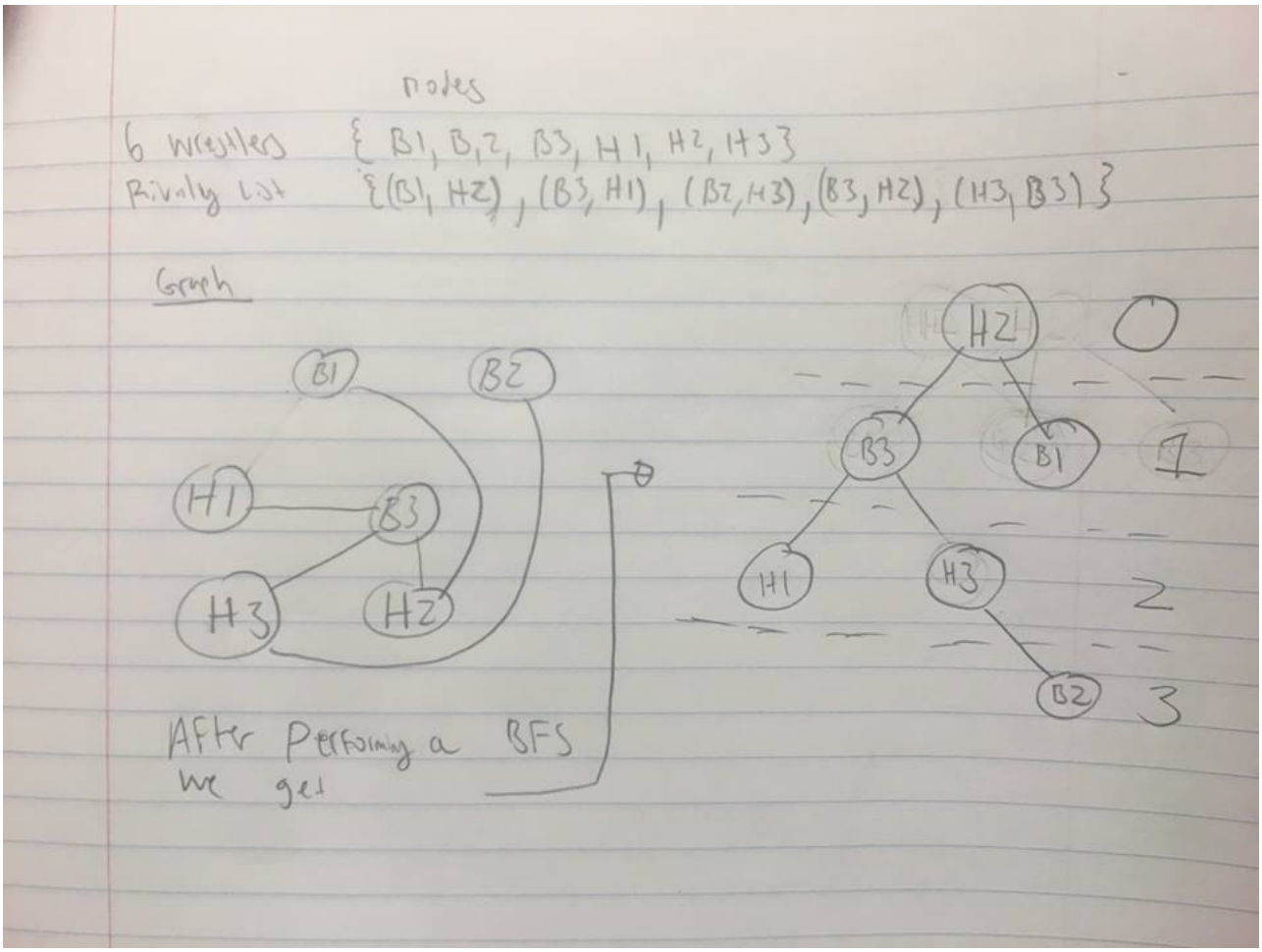
```
S = slack squared 2d matrix
OPT[0] = 0
#traverse the 2d array to find our traceback
for k in range(0, len(S)):
       for j in range(0, len(S[k]):
              # find the optimal solution within a given row
              if (OPT[k] > ( S[j][k] + OPT[j - 1] )) :
                     OPT[k] = S[j][k]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

return OPT[len(S)]
```

Solution 3: Baby Faces v.s. Heels

This is a simple graph traversal problem that can be solved by running a BFS. Since we are given the number n of professional wrestlers (nodes) a rivalry list r (edges), we can create a graph G by using the rivalry list to connect accordingly. Then we perform a BFS until we visit all vertices. Once we sort out the levels of connection, we can call all wrestlers who are on an even level "Babyfaces" and all wrestlers who are on an odd level "Heels". Once all wrestlers are assigned to their respective camps, we check each edge to verify that it only connects rivals. Because we know that a BFS has a $O(V + E)$ running time, we know that it will run in $O(r + n)$ time.

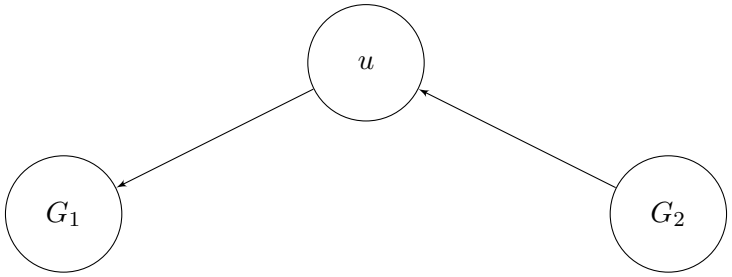
Below is an example:



Solution 4: Directed Graphs

The only way this can happen is if u has all outgoing edges directed to a subgraph of G which has already been visited. If all the nodes in the subgraph G_1 were visited before u, and u is chosen next in the DFS, then it will form its own DFT since there are no new vertices it can visit.

Below is an example

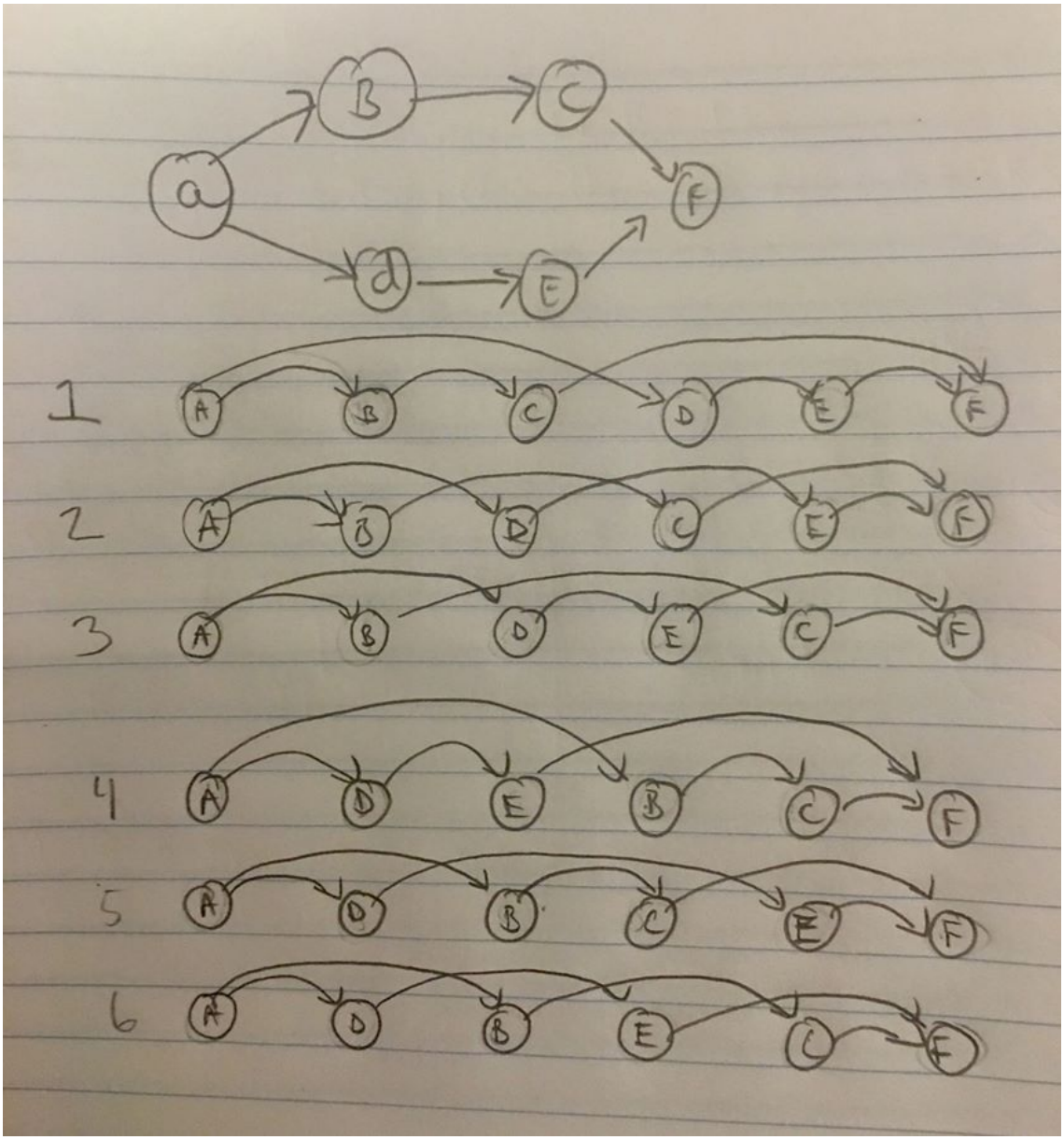


Solution 5: Topological Ordering

First let us define what topological sorting is.

A topological order of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, , v_n$ so that for every edge (vi, vj) we have $i < j$. However, we will be applying this in alphabetical terms instead of numerical terms.

With the given graph we will be able to produce 6 unique topological orderings. Below is the answer.



Solution 6:Clu Net

Because we are trying to maximize our bottleneck rate, we will want to find the maximum of the paths between u and v. Since there are algorithms out there that do essentially the opposite, we can apply those same algorithms by modifying the weights. In order to do this, we will first traverse the graph in O(E) time and either negate (as in multiply by -1) or the take the reciprocal of the bottleneck rate. After we do this slight modification we can now apply any algorithm to find a minimal spanning tree. Either Kruskals or Primms will give us the answer we want.