The 2021 Triad Programming Contest Problems

April 10, 2021

Contents

1... Locating an Antenna

2.. Password Validator

3.. Connecting Wires

4.. Efficient Snow Removal

5.. Election

6.. Comment Removal

7... OPS Optimal Packing Service

8.. Connectivity

Sponsored by the

Institute of Electrical & Electronic Engineers
Central North Carolina (CNC) Section

1 Locating an Antenna

Imagine you need to erect an antenna to communicate with a number of fixed wireless stations. You want to position the antenna to get the best possible reception given the location of the wireless stations. The strength of the radio signal decreases proportionally to the cube of the distance of the transmission. Therefore, you want to position the antenna so that the sum of the cubes of the straight line distance from the antenna to each wireless station is a minimum. The location should be determined to within 0.1 meters.

Write a program that reads in a list of X Y station locations and then displays the optimal X,Y location of the antenna such that the sum of the cubes of the straight line distance to each station is a minimum. Input must be read from the file **`antenna.txt`**

The first line of input will contain an integer (referred to as N) that indicates the number of stations.

The following N lines will each contain two positive floating point numbers giving the X Y coordinates of a wireless station in meters. There will be no more than 25 stations.

The output must be "Optimal antenna location is xxx.x yyy.y" where xxx.x and yyy.y is the optimal location. The numbers can be displayed with any number of digits to the right of the decimal point.

Example input

```
5
100 100
500 150
300 190.5
300 250
150 450
```

Output for the Example Input

```
Optimal antenna location is 264.1 235.3
```

2 Password Validator

Write a program that will validate a password.

Passwords must have,

1) 8 to 32 characters
2) at least one numeric digit
3) at least one uppercase letter
4) at least one lowercase letter
5) at least one symbol from the set `~!@#$%^&*`
6) must not have other characters than the above
7) starting character must not be a number

Sample passwords will be read in from a file named `password.txt` with the first line containing the number of passwords that will follow. Each password is on a separate line of the file.

You should display YES if the password if valid, or NO if the password is not valid followed by the password

Sample Input:

```
7
Pass5!
password1password2password3&password4
p%4223076
Password#1
PASSWORD#47
Pass Word#6
7th*Password
```

Sample Output:

```
NO  Pass5!
NO  password1password2password3&password4
NO  p%4223076
YES Password#1
NO  PASSWORD#47
NO  Pass Word#6
NO  7th*Password
```

3 Connecting Wires

You've accepted a job at Wires-R-Us. Your pitch to your boss was that you were the best person for the job because you could apply your computer science skills to minimize cost at the company.

In particular, you are in charge of cutting wire to connect terminals. There are n positive terminals and n negative terminals. They are all at distinct locations along the positive x-axis. You must place n wires, each connecting a positive terminal to a negative terminal, so that at the end each terminal is connected to exactly one wire. Your goal is to find out where to place the wires to minimize the total length of wire used. You must report the total amount of wire used. The length of a wire is the difference in the start and end locations.

Input is from the file `wires.txt` The first line of input is an integer (called ***n***) indicating the number of wires required. The value ***n*** will never be greater than 8. This is followed by 2 times ***n*** locations of the positive and negative terminals. There is one line for each terminal containing an integer location of the terminal followed by a space and either the character "+" for a positive terminal or "-" for a negative terminal.

Example input:

```
4
27 +
0 -
19 -
8 -
13 +
15 +
10 +
23 -
```

Example output:

```
23
```

4 Efficient Snow Removal

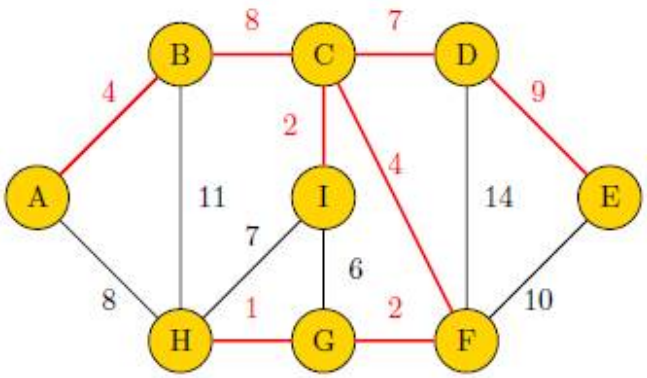
You've just started your first job out of college as the CIO for a city in the snowy north (think Michigan). Your city has a road network that can be depicted as a graph, where road segments are edges, and every intersection is a vertex. You need to make sure that, in the case of a blizzard, your snowplows clear a path from every place in the city to every other place in the city. However, some road segments are more expensive to plow than others, e.g., because of the street grade, the pavement type, or zoning restrictions in place. More specifically, each road segment has a cost associated with plowing it.

Your goal is to determine which streets to plow to minimize the city's overall cost of snow plowing while ensuring that all pairs of intersections in the city are reachable from one another via plowed road segments. As output, you will report the minimum cost. A solution will always exist.

Input is from the file `snow.txt` and contains lines that have a letter representing an intersection, a space, another letter representing the intersection at the other end of the road segment followed by a space and an integer representing the cost of plowing that segment. There will be no gaps in the letters used, although the input may not be in order. There are no more than 16 intersections in the city.

Example input

```
A B 4
A H 8
B C 8
B H 11
C D 7
C I 2
C F 4
D E 9
D F 14
E F 10
F G 2
G I 6
G H 1
H I 7
```



Example output:

```
37
```

Given graphically, the answer is:

5 Election

One way to vote that reduces the chance of a run-off election works as follows:

Each voter marks the ballot with his first, second and third choice of candidates. (We assume the ballot has at least 3 candidates.) The "total vote" is the number of voters.

The first choice votes for each candidate are counted. If one candidate has more than 50% of the total vote, that candidate is elected.

Otherwise, the second choice votes are added to each candidate's total. If any candidate has more than 50% of the total vote, the candidate with the largest percent over 50% wins. For example, if one candidate has 55% and one candidate has 57%, the 57% candidate wins.

If there is a tie after the second choice votes are considered or if no candidate has over 50% of the total vote, the third choice votes are added to each candidate's total, and the candidate with the largest percent over 50% wins. If no candidate has over 50% of the total votes or there is a tie, a run-off election must be held.

Example: Candidates A, B, and C are running, and there are 20 voters. The ballots result in the following count of votes for each candidate.

	A	B	C	
1st choice	7	7	6	*no one over 50%*
2nd choice	8	8	4	tie between A and B
3rd choice	5	4	11	C wins with 21 votes to A's 20 and B's 19

Write a program that determines the winner using this voting procedure. The input data is in a file named **ballot.txt**. You may assume that all voters always indicate a first, second, and third choice. The data has the ballots for each voter, with first, second, and third choices in that order. Candidates are named by single capital letters starting with A. The output is the letter of the candidate who won, or in the case of a tie, the program should display "tie". There will be no more than 100 voters and no more than 10 different candidates (identified as A to J).

Sample Input:

```
A B C
A B D
B C A
D C B
A D B
B D C
```

Output for the Sample Input

```
B wins
```

6 Comment Removal

A sneaky student has gotten hold of the instructor's solution to a programming assignment written in Conbol. The student wants to submit the program as his own. To make it look like a student program, he wants to remove all the comments. You are to write a program that reads a text file called **comment.txt** containing the Conbol program source code and display the program text without the comments.

There are two types of comments in Conbol. Comments can begin with two exclamation characters, **!!** and end with two exclamation characters, **!!** This form of comment can extend across multiple lines or only a part of a line. All characters in the between !! and !! should be removed, including the !! and !! and any end of line characters inside the comment. The other form of a comment starts with exclamation dash, **!-** and continues to the end of the line. When you remove a comment of this form, eliminate the **!-** and all following characters. You should leave the end of line character.

Conbol programs have string constants that are in "double quotes". If the start characters of a comment (either !- or !!) are inside a string constant, then they are part of the string constant and do not represent the start of a comment. String constants never span more than one line. It is possible that double quotes may appear inside a comment. If they do, they should be removed just like any other character. You may assume that the input program is syntactically correct.

Example input from comment.txt

```
public class { !!  My commented program !!
     public static void main(String[] unused) { !- main method
          String thing = "fake !- comment ";
          thing = !! short comment !! "real stuff";
          !!  long
              comment
          !!
          thing = "!! " + "!! "; !- weird
          int num = 5 / 6;
     }
}
```

Example output

```
public class {
     public static void main(String[] unused) {
          String thing = "fake !- comment ";
          thing =  "real stuff";

          thing = "!! " + "!! ";
          int num = 5 / 6;
     }
}
```

7 OPS Optimal Packing Service

Optimal packing has become a necessary requirement for many package delivery services. The idea of getting as much as you can into a small compact space for delivery is key to a successful business. You are to write a program that will find the optimal combination of items which will utilize the given space. Each item has a value and a weight. You want to select the items to put into a box so that you maximize the sum of the value of all items in the box while not exceeding the weight limit for the box. To make things simple, all weights and values are integer numbers.

Input is from the file `ops.txt` where the first input will be the maximum weight a box can hold. The sum of the weight of all items put in the box cannot exceed this amount.

The next input, ***n***, is the number of items available for shipping. There will be no more than 10 possible items. This is followed by ***n*** lines where each line contains a one character identifier (i.e. 'A' or 'X'), the value of the item and the weight of the item, each separated by a space.

The output should be the total value, total weight (formatted as shown below) followed on the next line by the one character identifiers of the items to be put in the box (in any order).

total value: *V* total weight: *W*

Sample input

```
8
4
A 1 2
B 2 2
C 2 2
Z 3 4
```

Sample output

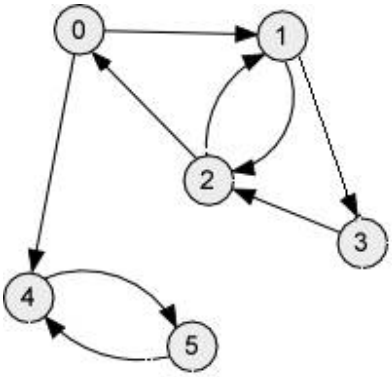
```
total value: 7 total weight: 8
B C Z
```

8 Connectivity

Some social networks allow you to specify other users as friends. Friends have additional access rights to your information, as does any other user who is a friend of the friend. This relationship extends to any level, such as a friend of a friend of a friend of a friend. The friend relationship is one way. If user A friends user B, this does not mean that user B friends user A. The owners of the social network want to know if everyone in the social network is connected to every other user. The friendship relationship can be modeled as a directed graph where the users are nodes and friendship is represented as a directed edge between the nodes. A user is never friends with themselves, so there are no edges that start and end at the same node. The problem is to determine if the friendship graph is fully connected. Is there a path from each node to every other node? The graph below is disconnected. While nodes 0 – 3 are connected, there is not a path from nodes 4 and 5 to the other nodes.

Write a program that will read a set of friendship relationships and display "connected" or "disconnected" if the directed graph is connected or not. Input to the program will come from a text file names `connectivity.txt`. The file will contain:

- An integer specifying the number of graphs to check.



For each graph:

- An integer specifying the number of nodes or users in the graph. There can be from 1 to 64 nodes in the graph. Each node is numbers from 0 to N.
- An N x N adjacency matrix where element i, k (column k of row i) of the matrix is 1 if node i has an edge to node k and zero if it does not.

Example input in connectivity.txt:

```
2
6
0 1 0 0 1 0
0 0 1 1 0 0
1 1 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 1
0 0 0 0 1 0
4
0 1 0 0
0 0 1 1
1 1 0 0
0 0 1 0
```

Adjacency matrix for the above graph

Example output:

```
disconnected
connected
```