CS4102 - ALGORITHMS - SPRING 2022

DUE FRIDAY, APRIL 15, 2022 AT 11:30 PM VIA GRADESCOPE

B PROGRAMMING

Collaboration Policy: You are encouraged to collaborate with up to 3 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in a comment at the top of your files. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite by naming the book etc. or listing a website's URL. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: List computing ids in comments at the top of your file

Sources: List URLs etc. in comments at the top of your file

Supply Chain

In recent years we've all become aware of problems with the supply chain that have led to shortages of products and increased prices. One of the major store chains, TarMart, has asked you to help reduce the costs related to how its transportation network is used to get imported goods from ports to stores. After goods arrive by ship at a port, there are many possible ways they can be transported by truck or rail to a distribution center, and from there to individual stores. Your job will be to find the minimum cost for connecting up each "node" in their transportation network subject to certain constraints.

Goods can be transported between nodes in the network (in either direction), but each connection has a cost represented by a positive number. (This number could be the cost per unit weight for the distance between nodes, or something similar. You just need to know that we want to use this to find minimum costs.) There are four kinds of nodes: *Ports*, *Rail Hubs*, *Distribution Centers*, and *Stores*.

Here are further details, including some of the constraints for how goods can be moved through the network.

Port: Items that arrive at a port can be transported by rail to a rail hub, or by truck to a distribution center, but not directly to a store.

Rail Hub: Items that arrive at a rail hub can be transported by rail to another rail hub, or by truck to a distribution center, but not directly to a store.

Distribution Center: Each distribution center exists to get goods to a particular set of stores. Goods arrive by rail or truck from a port or rail hub, and then a truck leaves the distribution center and goes to one or more stores that it serves. Goods will never be transported from one distribution center to another.

Store: Goods will arrive at a store from a distribution center or from another store. Each store can only be served by one distribution center.

Some additional problem constraints will be given later in this document, but we know enough to formally describe the problem.

Problem Input:

A graph G that defines the nodes in the transportation network, the possible transportation links between pairs of nodes, and the cost value for each of those links between the nodes.

Problem Output:

Find a tree that has the smallest total edge-weight sum of all possible spanning trees that connect all nodes and meet the constraints of this problem, and return the edge-weight sum for that tree.

Note: the constraints mean that the normal minimum spanning tree of G will almost certainly not produce the correct answer, since the constraints will mean that some edges that a MST algorithm would choose are not valid choices for this problem.

For the autograder, you will have a function that returns this total sum. For debugging on your own machine, you may want to print the edges you found that lead to this result so you can verify your results on your own test cases.

Input format:

1. As in the previous assignment, you'll process input that has been read from a file and is passed to your code as a list of strings.
2. The input will begin with one line containing integers N and L, the number of nodes in the transportation network and the number of possible links respectively.
3. The next N lines will specify the name of the node and its type. The type will be one of the following four strings: *port*, *rail-hub*, *dist-center*, or *store*. The node name is a string with no spaces; make no other assumptions about the name's value.
4. Recall that stores can only belong to one distribution center. When a distribution center is defined in the input, the stores associated with that will be listed immediately after that to reflect this dependency.
5. After the N nodes are defined, the next L lines specify the possible links by providing the name of the two nodes and the cost to move goods between them. The cost will be an integer.
6. Unfortunately TarMart has provided us "dirty" data for the set of links in the input: they may contain connections that do not meet the constraints. (E.g. a link connecting a port to a store, or one connecting a distribution center to a store that's not part of its set of stores.) These links should not be considered when solving the problem.

Constraints and Advice

- Your solution **must** use Kruskal's algorithm as part of the solution. The purpose of this assignment is to practice with this algorithm.
- As noted earlier, the problem cannot be solved by just finding the MST on the entire graph. You will need to work on parts of the graph in different stages, or in different ways. There are several ways to solve this problem. Some of these are easier if you write a function to do Kruskal's so that it can work on part of the entire graph (e.g. a subset of the vertices). Or, your solution might need a Kruskal's function that that gives you more control about which graph vertices are initially in sets together when the algorithm begins. These are just hints or suggestions, and there are multiple ways to solve this. But as you think about your approach, consider if any of these ideas might help you.

- When implementing Kruskal's, we do not care if you sort edges or use a heap (as taught in class). If you use a heap, you may use library code or solutions from the web for the heap components. (Be sure to give your source.)
- When implementing Kruskal's, you **must** implement the disjoint-sets data structure and its union/find operations yourself. You cannot use a library for this. A human will check to see that you've done this, so we require that you name the array or list object that stores the set information `disjointsets` so that we can find this part of your code easily. Use this exact name for the variable!
- You are not required to implement path-compression or union-by-rank. You can just implement the basic form of the find and union operations.
- It may be clear from the description above, but we'll repeat this: As goods are moved from a port to a store, once they reach a distribution center they can only be moved on to a store. Your solution cannot be based on a tree that uses edges that have a port or rail hub "between" a distribution center (or store) and a store.
- While this may seem counter-intuitive, do not worry if a path from a port to a rail-hub ends without connecting to a distribution center. TarMart tells us they want a solution that connects all nodes in the network, even these. We also asked TarMart if they really wanted a solution based on a shortest-path algorithm, and they replied "no" and that a minimum-connection tree models their business need better.
- As you read the input, you'll need to use data structures that allow you to have a graph to store edges between nodes. You will probably also need to be able to associate a vertex id-number with a node, find a given node's type, and find information about a node given either its name or id-number.

Other Constraints

Your program should have the following properties:

- Your program must be written in Python 3 (3.6 or lower), Java (11), or C++.
- You must download the appropriate wrapper code from Collab based on the language you choose: `Main.py` and `Supply.py` for Python, `Main.java` and `Supply.java` for Java, `Main.cpp` and `Supply.cpp` for C++.
- The `compute()` method in `Supply` receives as input the body of the input file as a list of strings. You must have `compute()` return the results described above. An example input file is shown in Figure 1, which has answer 7.
- You should write **another** function in the `Supply` file to implement the algorithm. Your `compute()` method will then invoke this function and return the correct result. Do **not** modify the method definition for `compute`.
- You *may* modify the `Main` files to test your algorithm, but your modified `Main` file **will not** be used during grading.
- You must submit your `Supply` file to Gradescope. Do **not** submit `Main.java`, `Main.py`, `Main.cpp`, or any test files.
- You may submit your code a **maximum of 15 times** to Gradescope. After the 15th submission, Gradescope will not run your code.

- A few other notes:
  - Your code will be run as:
    `python3 Main.py` for Python,
    `javac *.java && java Main` for Java,
    or `g++ Supply.cpp Main.cpp && ./a.out`.
  - You may upload multiple files if you need additional classes (such as Java or C++), but if using Java **do not put your class inside a Java package**.
  - You may use any editor or IDE you prefer as long as the code can be executed as stated above.
  - We strongly encourage you to practice good development as you write code: use meaningful variable names, break your code into functions, "code a little then test a little," use the debugger, etc.

Sample Input:

```
6 10
p1 port
rh1 rail-hub
rh2 rail-hub
dc1 dist-center
s1 store
s2 store
p1 rh1 5
p1 rh2 1
p1 dc1 4
rh1 dc1 1
rh1 rh2 2
rh2 dc1 2
rh2 s1 1
dc1 s2 1
dc1 s1 6
s2 s1 2
```

Figure 1: Example input, for which your algorithm should return 7.