CS 1501 Project 4

Released: Friday, March 23

Due: Thursday, April 5, 11:59 PM

Goal

To gain a better understanding of graphs and graph algorithms through practical implementation.

High-level description

Your program will analyze a given graph representing a computer network according to several specified metrics. The vertices of these graphs will represent switches in the network, while the edges represent either fiber optic or copper cables run between the switches. Your program should operate entirely via a console interface menu (no GUI).

Specifications

1. Your program should accept a single command-line argument that specifies the name of a file containing the description of a graph. Two such files are provided, `network_data1.txt` and `network_data2.txt`. The format of these files is as follows:
    - The first line contains a single int stating the number of vertices in the graph. These vertices will be numbered 0 to v-1.
    - Each subsequent line describes a single edge in the graph, with each of the following data items listed separated by spaces.
        - First, two integers specify the endpoints of the edge.
        - Next, a string describes the type of cable the edge represents (either "optical" or "copper").
        - Next, an integer states the bandwidth of the cable in megabits per second.
        - Finally, an integer states the length of the edge in meters.
        - As an example, the line `0 5 optical 10000 25` describes an edge between vertex 0 and vertex 5 that represents a 25 meter optical cable with bandwidth of 10 gigabits per second.
    - Assume that all cables are full duplex and hence represent connections in both directions (e.g., in the

example above, data can flow from vertex 0 to vertex 5 at 10 gigabits per second and from vertex 5 to vertex 0 at 10 gigabits per second simultaneously).

2. You must internally represent the graph as an adjacency list.
3. After loading the graph from the specified file, your program should present the user with a menu with the following options:

    1. Find the **lowest latency path** between any two points, and give the bandwidth available along that path.

        1. First, your program should prompt the user for the two vertices between which they wish to find the lowest latency path.
        2. Then, your program should output the sequence of vertices that comprise the lowest-latency path, in order from the first user-specified vertex to the second.

            You must find the path between these vertices that will require the least amount of time for a single data packet to travel. For this project, assume that the time required to travel along a path through the graph is the sum of the times required to travel each link, and that the time to travel each link is equal to the length of the cable divided by the speed at which data can be sent (determined by the link type).

            - Assume data can be sent along a copper cable at a speed of 230,000,000 meters per second.
            - Assume data can be sent along a fiber optic cable at a speed of 200,000,000 meters per second.

        3. Finally, your program should output the bandwidth that is available along the resulting path (i.e., the minimum bandwidth of all the edges in the path).

    2. Determine whether or not the graph is **copper-only connected**, or connected when considering only copper links (i.e., ignore fiber optic cables).

    3. Find the **maximum amount of data** that can be transmitted from one vertex to another.

        1. First, your program should prompt the user for the two vertices between which they wish to find the bandwidth.
        2. Then, your program should output the maximum amount of data that can be transmitted from the first to second user-specified vertices.

    4. Find the **minimum average latency spanning tree** for the graph, the spanning tree with the lowest average latency per edge.

5. Determine whether the graph would remain connected even if **any two vertices fail**.

   Note: You are not prompting the user for two vertices that could fail, you need to determine whether there is *any pair* of vertices that, should they both fail, would cause the graph to become disconnected.

6. Quit the program.

Submission Guidelines:

- **DO NOT** upload any IDE package files.
- You must name the primary driver for your program `NetworkAnalysis.java`.
- You must be able to compile your program by running `javac NetworkAnalysis.java`.
- You must be able to run your program with `java NetworkAnalysis [data_filename]` (e.g., `java NetworkAnalysis network_data1.txt`).
- You must fill out `info_sheet.txt`.
- The project is due at 11:59 PM on Thursday, April 5. Upload your progress to Box frequently, even far in advance of this deadline. **No late assignments will be accepted.** At the deadline, your Box folder will automatically be changed to read-only, and no more changes will be accepted. Whatever is present in your Box folder at that time will be considered your submission for this assignment—no other submissions will be considered.

Additional Notes and Hints:

- Code for some of the algorithms needed in this assignment has been provided by the authors of your textbook. Use of this code will require extensive adaptations to account for the two weights of each edge in the graph.
- The assumed calculation of network latency used here is a drastic simplification for this project. Interested students are encouraged to investigate a more detailed study of computer networks independently (recommended reading: *Computer Networks: A Systems Approach* by Peterson and Davie).

Grading Rubric

Feature	Points
Menu interface is user-friendly	5
Graph is properly read and represented	5

Query	Points
Lowest latency path	15
Copper-only connectivity	10
Maximum data flow	20
Minimum average latency spanning tree	15
Surviving 2 vertex failures	25

Other	Points
Assignment info sheet/submission	5