Hacettepe University - Computer Engineering Department

Make-Up Programming Assignment

BBM204 - Software Practicum II - Spring 2022

Topics: Graphs - Shortest Path, Regular Expressions

Course Instructors: Assoc. Prof. Dr. Erkut Erdem, Prof. Dr. Suat Özdemir, Asst. Prof. Dr. Adnan Özsoy
TAs: **Ali Burak Erdoğan (assignment author),** Alperen Çakın, Selma Dilek
Programming Language: Java 1.8.0
Due Date: Sunday, 12.06.2022 (23:59:59)

In the Internet, network packages between nodes are transferred via intermediary nodes called *routers*. (see Figure 1) The objective of a router is to receive a network package and pass it to another in such a way that the package is delivered to the target node via the most optimal *route*. Since a router is connected to numerous others, it should make a decision between them before forwarding to ensure the package is directed towards the optimal path.

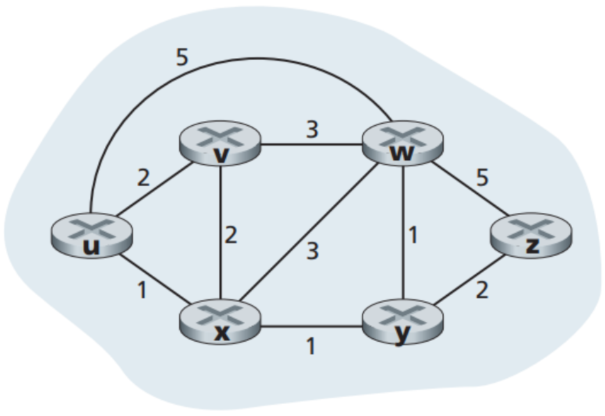


Figure 1. Abstract graph model of a computer network with routers. [1]

A router makes use of internal structures named *routing tables* to store this information. Figure 2 shows an example of routing tables with three columns for router *u*. The first column shows the destination node, the second shows the next router to pass the network package for the optimal path, and the last is the cost metric to the final destination. For instance, to send a network package from router *u* to destination router *w* in the shortest way, the package should be passed to router *x* and the total cost of sending via router *x* is 3.

Destination	Next Router	Total Route Cost
v	v	2
x	x	1
w	x	3
y	x	2
z	x	4

Figure 2. Routing table of router *u.*

Assignment Tasks

As the network administrator of the Computer Engineering department, you were assigned to implement software that automatically fills *routing tables* of all connected routers. The default link costs between routers are calculated as inversely proportional to their bandwidths. Also, you are expected to implement some methods for the capability of dynamically adding, removing, or temporarily disabling routers (for maintenance) from the network. Furthermore, internal networking policies require that the software should be able to manually configure link costs of some connections between routers to discourage (or encourage) their usage, for the purpose of preventing traffic imbalances. As soon as any such change occurs in the network, your software should update the *routing tables* of all routers.

Task I

Task I-A: Network Initialization

In your department, the routers are divided into different sub-networks (an example network topology can be seen in Figure 3.). The third division of a router's IP address indicates the subnet which that router belongs to (e.g., 192.168.2.11 belongs to subnet 2). The numbers given above the links indicate the bandwidth of the connection in Mbps. Notice that the links between different subnets have lower bandwidths, and the internal connections in a subnet have higher bandwidth capacities.

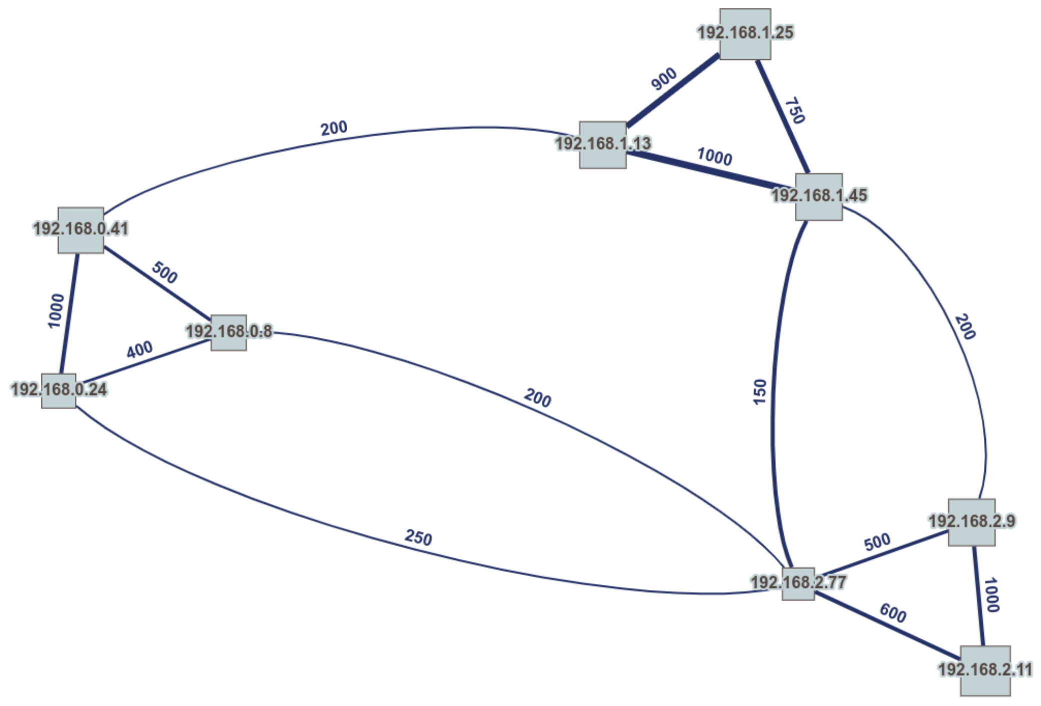


Figure 3. Example network topology of your department.

The link costs are to be calculated in accordance with these bandwidths when the network is first loaded in your program. Since link costs should be inversely proportional to bandwidths (because high bandwidth connections can handle more traffic), and the maximum bandwidth in this network is 1Gbps, the formula for cost calculation is defined by your department as given below:

$$C = \frac{1000}{B}$$ (C: Cost, B: Bandwidth in Mbps)

The current network's topology is given to you as an input file (`<networkfile>.in`) as a **first command-line argument**. An example network input file is given below:

```
RouterIP:192.168.0.41
RouterIP:192.168.0.8
RouterIP:192.168.0.24
RouterIP:192.168.1.13
Link:192.168.0.41-192.168.1.13 Bandwidth:200 Mbps
Link:192.168.0.41-192.168.0.8 Bandwidth:500 Mbps
Link:192.168.0.24-192.168.0.8 Bandwidth:400 Mbps
```

It's your job to parse this file correctly (*hint: You should make use of regular expressions for easier and more robust parsing*) and implement the public constructor of `Network` class.

The constructor should read that file and generate necessary `Router` and `Link` objects and initialize link and router arrays. Also, you should implement `Link` class's `calculateAndSetCost()` method in order for the costs to be calculated based on the formula given above. After you complete these steps, you can run `TestReadNetwork.java` class's `main` function to see if you've parsed and loaded the network successfully. If successful, the test will print a score of 1.0.

WARNING: For our tests to be performed correctly, you should instantiate `Router` and `Link` objects in the same order as the input file lists. Otherwise, you cannot get a score of 1.0 and the following tasks will result in lower scores as well.

Task I-B: Recovering Corrupted Network Files via Regular Expressions

Sometimes network input files get slightly corrupted such that some extra characters may get added in random places. You are expected to implement `subnetsRegularExpression()` and `routersInSubnetRegularExpression(int subnet)` functions from `Network` class. These functions help recovering the IP addresses. Sample corrupted network input files are shown below.

File: test_io/input/subnets_regex_input.txt

```
Rou%*(#terI.P:$*&@(*192.168.0.41sd.fdfgRouccterIxP:192.168.1.8$_x7A3
$#*@RouterI222P:192.168.8.24v+).3.2&*RouXterI##Psd:192.168.1.13WQ=?2
_t$#*@RouterI222P:192.168.6.24v+).3.2dfgRouccterIxP:192.168.4.8$_x7A
3Rou%*(#terI.P:$*&@(*192.168.14.41sd.f&*RouXterI##Psd:192.168.22.13W
Q=?2_t&*RouXterI##Psd:192.168.4.13WQ=?2_t$#*@RouterI222P:192.168.0.2
4v+).3.2Rou%*(#terI.P:$*&@(*192.168.22.41sd.f
```

- `subnetsRegularExpression():`
  The regular expression that you will return from this method should extract the subnet number which the router belongs to (e.g. 192.168.3.55 belongs to Subnet 3) from each IP address.

- `routersInSubnetRegularExpression(int subnet):`
  The regular expression that you will return from this method should return all IP addresses within the given subnet from input file.

After you implement these two methods, you can run `TestRegex.java` class's `main` function to see if you've recovered all the router IP addresses. The ratio of successful recoveries will determine your score in this task.

WARNING: IP addresses can have **up to 3 digits** in each segment. Full IP addresses will not be corrupted in any case in this assignment (for simplification). In this task it's not possible that a random character gets between an IP address like 192xj.16x8.0&.8_$7. So after a dot (.) character, try to match consecutive digits (up to 3) and if any non-digit character appears, it means it was the end of the IP address. Any digits appearing after a non-digit character does not belong to the IP address.

Task II - Initialization of Routing Tables

In computer networks, each router is responsible for filling its own routing table according to the current situation of the network topology. After you have generated the `Network` object properly, `updateAllRoutingTables()` method will be executed and all the `Router` objects will start calculating their routing tables by calling their own `RoutingTable` object's `updateTable()` function.

A `RoutingTable` object contains a reference to the `Router` object to which that table belongs, also to the `Network` object that gives access to the entire topological information of the `Network`. There are some methods predefined in the starter code of `Network` class which you can make use of during calculating routing tables. Some of them are given below:

- **`public List<Router> getRouters()`**
- **`public List<Link> getLinks()`**
- **`public Router getRouterWithIp(String ip)`**
- **`public Link getLinkBetweenRouters(String ipAddr1, String ipAddr2)`**
- **`public List<Link> getLinksOfRouter(Router router)`**
- **`public void printNetwork()`**
- **`public void printRoutingTable(Router router)`**

Your task is to implement the `RoutingTable` object's `updateTable()` and `pathTo(Router destination)` functions.

- `pathTo(Router destination)` should return a `Stack<Link>` object which contains a stack of `Link` objects, which represents a valid path from the owner `Router` to the destination `Router`.
- `updateTable()` should calculate routing information and then instantiate `RoutingTableEntry` objects, and finally add them to `RoutingTable` object's `entryList`.

After you implement those two methods, the given starter code should be functioning correctly for all `Router` object's routing table calculation. You can run `TestInitializeRoutingTables.java` class's `main` function to see if your code calculates each router's routing table correctly. Full score is earned if each entry in each routing table is correct. The percentage of correct entries produces the final score of this task.

Task III - Dynamic Network Configuration

The nature of computer networks is dynamic. It means that sometimes connections get lost, some routers become unavailable or there may be changes in connections between routers. Your program should behave stable in all of these scenarios and routing tables should always be representing the network's current situation.

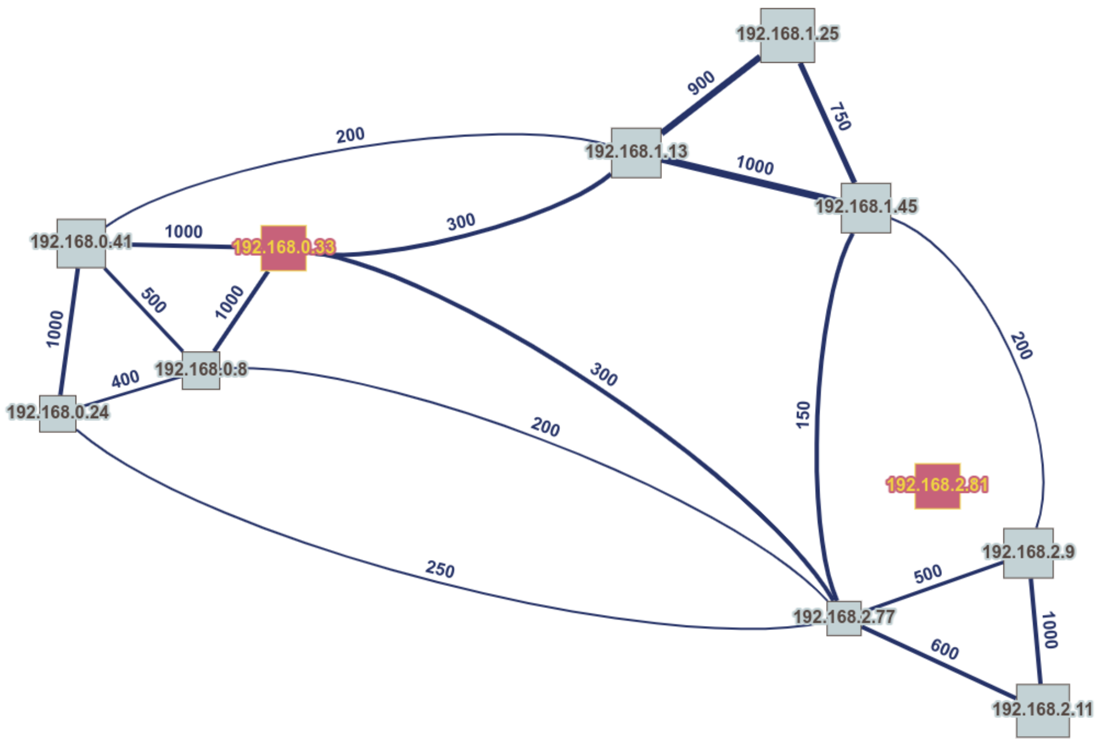
`Network` class has some interfaces for making such topological configurations, your job is to implement the functions listed below. These methods should make necessary changes in the `Network` object, and update all router's routing tables via the method `updateAllRoutingTables().`

- **`public void addRouter(Router router)`**
  Add a new Router to the Network, and update all routing tables.
- **`public void addLink(Link link)`**
  Add a new Link to the Network, and update all routing tables.
- **`public void removeLink(Link link)`**
  Remove a Link from the Network, and update all routing tables.
- **`public void removeRouter(Router router)`**
  Remove a Router from the Network, and update all routing tables. Beware that removing a router also causes the removal of any links connected to it from the Network. Also beware that a router which was removed should not appear in any routing table entry.
- **`public void changeLinkCost(Link link, double newCost)`**
  Change the cost of the link with a new value, and update all routing tables.
- **`public void changeStateOfRouter(Router router, boolean isDown)`**
  Change the state of the router (down or live), and update all routing tables. Beware that a router which is down should not be reachable and should not appear in any routing table entry's path. However, this router might appear in other router's routing-tables as a separate entry with a `totalRouteCost=Infinity` value because it was not completely removed from the network.

After you implement these methods, your program should be ready to be tested with the following test classes.

WARNING: *All these test files are prepared for you to test your code during the development. After your submission, your code will be tested with the same tests but with larger and different network inputs.*

- **`TestAddRouter.java`**



Test scenario: In the department you observed a latency in connections from Subnet 0 to other subnets and you purchased a brand new router with higher bandwidth cables. You connected this router to all other subnets' gateway routers with 300Mbps cables, as well as to other nodes in its own subnet with 1000Mbps cables. Also, you added a new router to Subnet 2 but could not make any connections to other routers yet. After these changes, all routing tables should update themselves according to this change.

You can run `TestAddRouter.java` class's `main` function to see if your code calculated each router's routing table correctly. Full score is earned if each entry in each routing table is correct. The percentage of correct entries produces the final score of this task.

- **`TestChangeLinkCosts.java`**

Test scenario: You received numerous complaints from the department that the computers connected to the router (192.168.0.24) have a slow internet connection. You realized that this router is heavily loaded by the traffic between Subnet 0 and Subnet 2. You've decided to perform load-balancing and transfer the traffic between Subnets 0 and 2 to the gateway router (192.168.0.8). You cannot achieve this by disabling the router (192.168.0.24) because it would also disable the internal LAN traffic of computers connected to it. Your final solution is to manually increase the cost (independently from bandwidth) of the link between 192.168.0.24 and 192.168.2.77, so that this link is not preferred by routers. Furthermore, the cable connecting 192.168.1.45 to 192.168.2.9 was damaged and you had to quickly replace it with a lower bandwidth 100Mbps back-up cable.

You can run `TestChangeLinkCosts.java` class's `main` function to see if your code calculates each router's routing table correctly.

- **`TestChangeRouterState.java`**

  Test scenario: Subnet 0 and Subnet 1 were connected via gateway 192.168.1.13 (see Figure 3 for topology) but unfortunately that router went down due to anomalous traffic. This means all traffic from Subnet 0 should use Subnet 2 to reach final destinations in Subnet 1. This will cause much higher costs for network traffic.

  You can run `TestChangeRouterState.java` class's `main` function to see if your code calculates each router's routing table correctly.

- **`TestRemoveLink.java`**

  Test scenario: Some old cables stopped functioning and those links are no longer visible by the routers. Your program should automatically adapt to this event.

  You can run `TestRemoveLink.java` class's `main` function to see if your code calculates each router's routing table correctly.

- **`TestRemoveRouter.java`**

  Test scenario: You had to remove router 192.168.1.45 and remove all cables plugged to it for maintenance.

  You can run `TestRemoveRouter.java` class's `main` function to see if your code calculates each router's routing table correctly.

Must-Use Starter Codes

You MUST use this starter code. All classes should be placed directly inside your *zip* archive. Feel free to create other additional classes if necessary, but they should also be directly inside the *zip*.

Grading Policy

Submission: 1%
Task I-A: Network Initialization: 9%
Task I-B: Recovering corrupted Network files via Regular Expressions: 10%
Task II: Initialization of Routing Tables: 45%
Task III: Dynamic Network Configuration: 35% (for each test 7%)

Important Notes

- Do not miss the deadline: **Sunday, 12.06.2022 (23:59:59)**.
- Save all your work until the assignment is graded.
- The assignment solution you submit must be your original, individual work. Duplicate or similar assignments are both going to be considered as cheating.
- You can ask your questions via Piazza, and you are supposed to be aware of everything discussed on Piazza.
- You can only test your code via http://34.159.86.133/ (**does not count as submission!**).
- You must submit your work via https://submit.cs.hacettepe.edu.tr/ with the file hierarchy given below:
    - **`b<studentID>.zip`**
        - `Network.java`
        - `Router.java`
        - `Link.java`
        - `RoutingTable.java`
        - `RoutingTableEntry.java`
- All classes should be placed directly in your *zip* archive. Feel free to create other additional classes if necessary, but they should also be inside the *zip*.
- This file hierarchy must be zipped before submitted (not .rar, only .zip files are supported).
- **Usage of any external libraries is forbidden**.

Run Configuration

Your code will be compiled and run as follows:

```
javac -cp . *.java
java -cp . <TestFile> <networkInputFile>
```

Academic Integrity Policy

All work on assignments must be done individually. You are encouraged to discuss the given assignments with your classmates, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in pseudocode) **will not be tolerated**. In short, turning in someone else's work (including work available on the internet), in whole or in part, as your own will be considered as **a violation of academic integrity**. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.

The submissions will be subjected to a similarity check, especially against the submissions for Programming Assignment 4. Any submissions that fail the similarity check will not be graded and will be reported to the ethics committee as a case of academic integrity violation, which may result in suspension of the involved students.

References

[1] Kurose, James F, and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Boston: Addison-Wesley, 2001. Print.