National University of Computer and Emerging Sciences

School of Computing Fall 2017 Islamabad Campus

CS201: Data Structures (Fall 2017)

Semester Project

(Deadline: 28th December, 2017 01:00 AM)

Project groups: This project can be done within a group of three (3) students. There is no restriction on the selection of group members. Students are allowed to make groups according to their preferences. The group members may belong to the same or different sections.

Submission: All submissions MUST be uploaded on slate. Solutions sent to the emails will not be graded. To avoid last minute problems (unavailability of slate, load shedding, network down etc.), you are strongly advised to start working on the project from day one.

You are required to use Visual Studio 2017 for the project. Combine all your work (solution folder) in one .zip file after performing "Clean Solution". Submit zip file on slate within given deadline. If only .cpp file is submitted it will not be considered for evaluation.

Deadline: Deadline to submit project is **28th December, 2017 01:00 AM**. Late submission without any deduction will be accepted till **28th December, 2017 09:00 AM**. No submission will be considered for grading outside slate or after **28th December, 2017 09:00 AM**. Correct and timely submission of project is responsibility of every group; hence no relaxation will be given to anyone.

Plagiarism: -50% marks in the project if any significant part of project is found plagiarized. A code is considered plagiarized if **more than 20%** code is not your own work.

The Network Emulator

A network is a set of connected electronic devices such as routers, and computers etc., where each device has a unique identification address (e.g., IP address). A message is sent from one device to another using the destination (recipient) address. Clearly, the message will have to follow a route (i.e., path of routers) in the network in order to get delivered to the desired destination device. On the path, each router checks the destination address of the message and forward it accordingly to the next router or the destination computer. For this purpose, a router typically maintains a table (named as routing table) containing the information about where to forward a message with certain destination address.

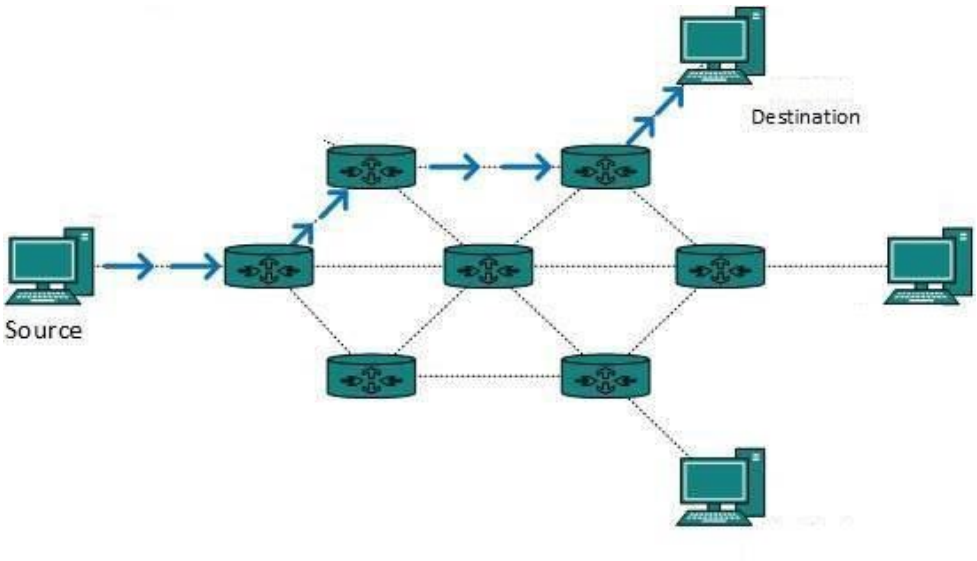

Figure 1: An example Network

Router

A router consists of three main components namely incoming queue, routing table and outgoing queues as shown in Figure 2.

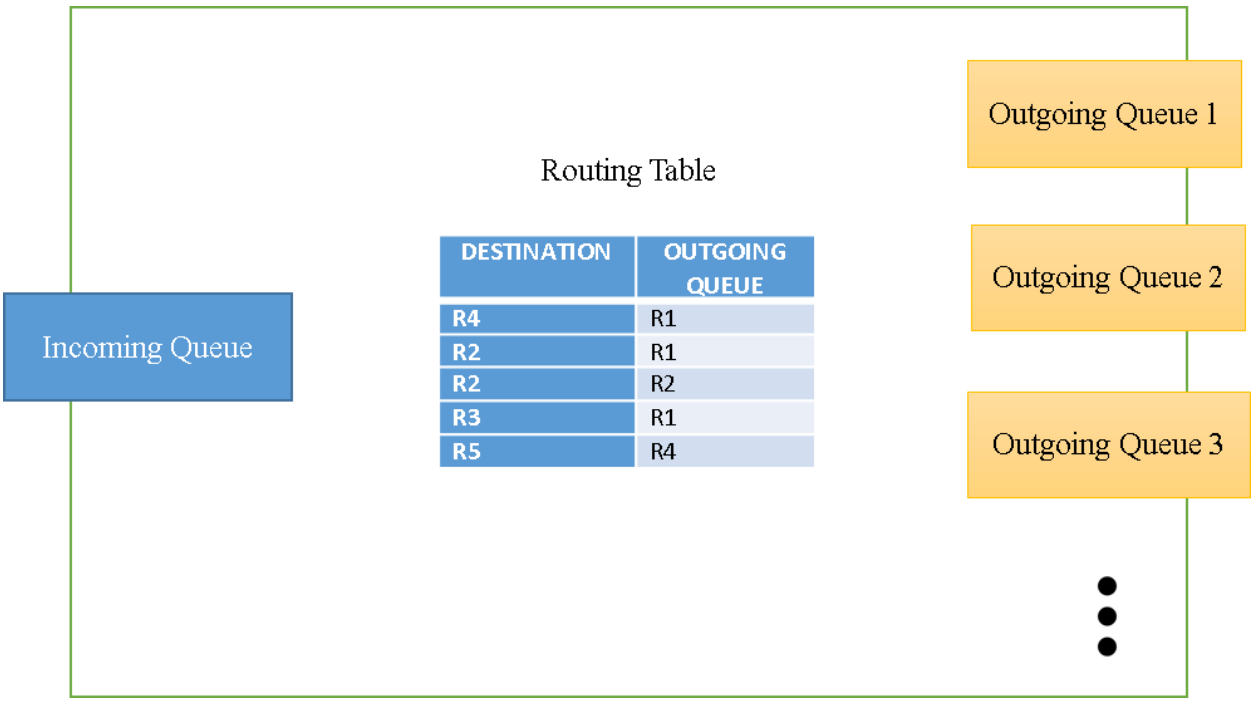

Figure 2: Router

Incoming Queue: A router maintains a single queue (for the sake of this project) to receives all incoming messages. Messages are received either from previous routers in the path or directly from the end machines/computers. Router retrieves messages from the incoming queue one by one, makes the routing decision (i.e., identifies the next router) using the routing table, and forwards the message to the corresponding outgoing queue for transmission to the next router on the path to the destination machine.

Incoming messages can be of different priority and therefore are kept in a priority queue. A router always processes the message with highest priority first out of all the messages in the incoming queue.

Outgoing Queues: A separate queue is maintained for each outgoing connection to a router or to a destination machine. Each outgoing queue is a normal queue (FIFO) because priority is already handled by the incoming queue. Messages in the outgoing queues are forwarded to the corresponding routers and machines.

Routing Table: A routing table stores a list of destination, outgoing queue (i.e., router) pairs. In this project, we assume that routing table contains a separate entry for each (possible) destination in the system. Given the destination address in the message the routing table is used to find the next router (or machine if the destination machine is directly connected to the router) in the path to the desired destination machine.

Phase 1: Implementation of a router

In this phase, you are required to implement the router functionality. Implementation details for individual router components are provided below.

Incoming Queue (binary heap): In this project priority can be represented by an integer value between 0 and 10 (10 means higher priority).

Outgoing Queue (FIFO): Usually the message takes some time to be forwarded on the physical link from one router to another, this is called latency or delay. To simulate the concept of latency we assume that the messages are buffered in the outgoing queue of previous router for 1 second before moving into the incoming queue of the next router (on the path). In other words, messages are moved from the outgoing queue of previous router/machine to the incoming queue of the next router - one by one (each experiencing 1sec delay at its turn).

Routing Table: Table 1 shows a sample routing table. Just to make things easy, we have assumed that the routing table consists of only two fields. The *destination* field indicates the target machine (represented with prefix M) to which the message is being forwarded and the *outgoing queue* field specifies the next best router (represented with prefix R) to reach the target/destination.

DESTINATION	OUTGOING QUEUE
M4	R1
M2	R1
M1	R2
M3	R1
M5	R4

Table 1: Sample Routing Table

You are required to implement the functionality of routing table using three different types of data structures, i.e.,

1) Linear lists
2) Splay trees
3) B trees

The user should be able to specify which data structure to be used for the routing table at the start of the program. For implementing Splay trees reading material will be uploaded on slate separately. Linear lists and B trees are already covered in the lectures.
Note that each of the above mentioned data structures has its own benefits, you should be able to justify the benefits and disadvantages of these data structures w.r.t. routing in your demo.

Message Format: Each message will have six fields as shown in Table 2 namely,

- **Message id (integer):** A unique identifier that is used to differentiate between different messages from the same machine.
- **Priority (integer):** An integer with values from 0 to 10 indicating the priority of the message.
- **Source address (string):** The address of the machine that sends the message. In this project, addresses of the machines/computers are represented by M prefix followed by a unique identifier, e.g., M1, M2, M6 etc. Likewise, the addresses of the routers are represented by R prefix followed by a unique identifier such as R1, R2 etc.
- **Destination address (string):** The address of the target machine that is the recipient of this message. Destination address is used by the routing table to make decision about the next router to forward the message.
- **Payload (string):** Payload contains the actual text/data of the message, e.g., "hello world", "how are you" etc.
- **Trace (string):** This field is used for testing the correct working of your network emulator. It indicates the path taken by a message from the source address throught the intermediate routers on the path until the message reaches the destination. Initially this field only contains the source address. However, as message is forwarded on the path towards the destination each intermediate router add its own address to this field.

MSG ID	3
PRIORITY	6
SRC	M6
DEST	M9
PAYLOAD	Hello World!
Trace	

Table 2: Sample message

Commands: You code should recognize and implement following commands.

- **Send message (send msg):** It should be possible to send the message using command line and file. There can be multiple message in a file (see Table 3) in colon separated format, e.g., message id: priority: src: dest: payload etc. Each message should be on a separate line.

MSG ID	PRIORITY	SRC	DEST	PAYLOAD	TRACE
1	2	M7	M1	Hey	
2	7	M3	M6	Call me	
3	1	M3	M5	Wats up	
4	4	M1	M8	No Idea	
5	9	M9	M4	123456	

Table 3: Messages

- **Change routing table (change RT):** The project should provide the functionality to change (i.e., add or remove) entries from routing table entries using both command line and file. The

file can contain multiple entries in colon separated format, e.g., dest address: router. Each entry should be on separate line.

- **Print path:** During routing *trace* field of each message is updated with the path information (as mentioned above). Once the message is delivered to the destination, your program should write the path followed by the message from the source to destination in a file (path.txt) in the colon separated format, e.g., message id: source address: router 1: … : router n: destination address. Each trace should be on a separate line.

Query Format: Your code should cater the following query formats

- command + filename
- command + contents of file

Sample Queries: The following sample queries must be handled by your program

- **send msg** hello.txt
- **change RT R1 add** rt_1.csv
- **change RT R2 remove** rt_2.csv
- **print path** M6 **to** M9
- **print path** M6 to *
- **print path** * to M9

Phase 2: Network Emulator

The phase 1 mostly focusses on implementing the functionality of a single router. In phase 2, you will implement the complete computer network and emulate the sending/routing of messages from the source machine to the destination via multiple intermediate routers. In this project, assume that in your network each machine/computer is always connected to one router as shown in Figure 3. Therefore, each machine maintains one outgoing queue and one incoming queue. Both of these queues are normal queues (FIFO) but the outgoing queue will have timer based mechanism as discussed in phase 1.
In phase 2, you are required to read a .csv adjacency list and create the emulated network from it. A sample .csv file of a graph containing machines and routers is attached along the project as "network.csv" (see Figure 3). In the first step you will read the graph from network.csv and construct the routing tables for each router. Do not make any routing table for the machines because they will not perform any routing and solely forward the messages to the routers they are connected with. You must use Dijkstra algorithm for constructing the routing table of each router. In particular, you are supposed to run Dijkstra algorithm for each router R1 and accordingly create its (i.e., R1's) routing table.

Example: Consider Figure 3 as a sample network. In this network each machine is only connected with one router and each router has its routing table. Note that each edge in the network has a weight and you'll use this weight to compute the shortest path for each destination/machine from your current router using Dijkstra algorithm and thus will populate the routing tables accordingly.

CVS file format: Attached .csv file is in the form of a symmetric matrix in which the symbol '?' depict that there is no edge between the correspondents (routers/machine) and any numeric value (e.g., 3, 4, 5 etc.) depicts that there is a weighted edge between the correspondents.

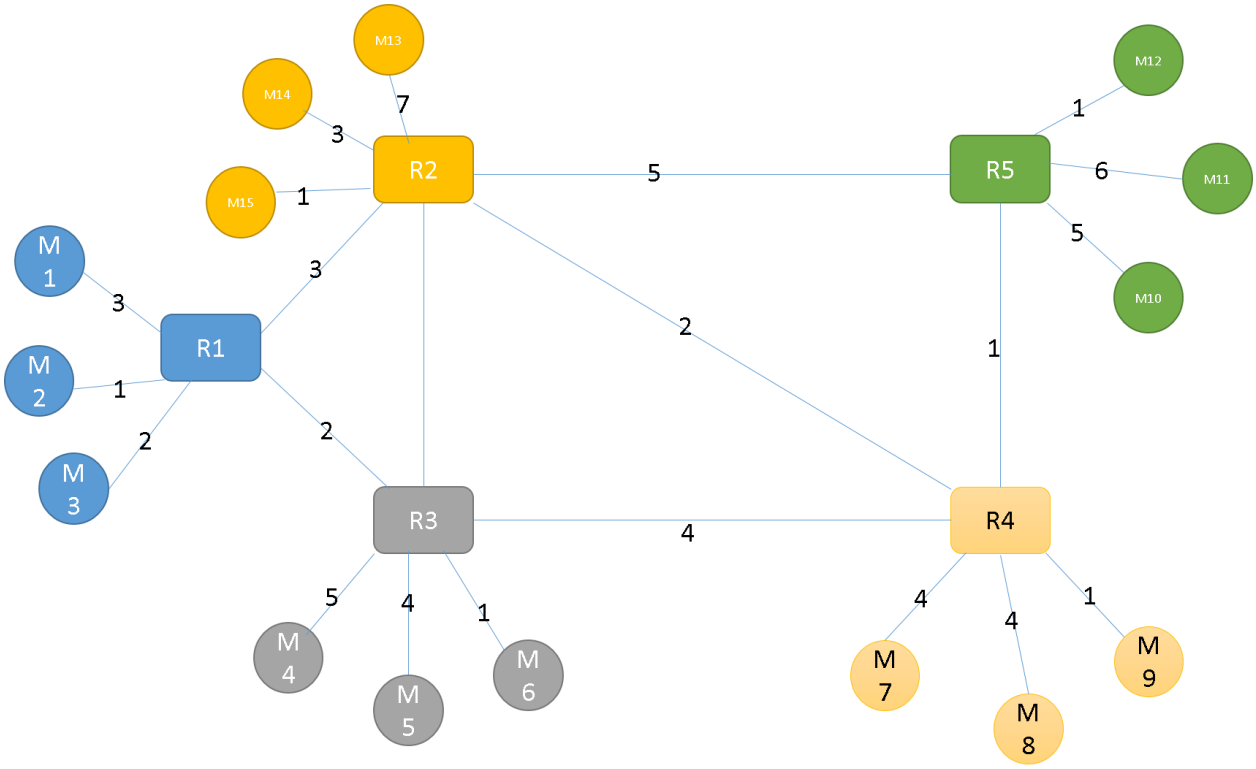

Figure 3: Network

Message forwarding: To send a message m, a machine adds m to its outgoing queue from where m is moved to the incoming queue of the connected router after emulating a network delay of one second (at its turn in case multiple messages are already in the outgoing queue of the machine) as mentioned above. This message after getting processed by the router (i.e., after routing decision) will be forwarded to its outgoing queue. Same procedure will be followed on each router unless the message reaches its destination, i.e., target machine.

Interrupt mechanism: To ensure that each message in any of the outgoing queues emulates a network delay of one second, you may have to implement timer-based interrupt mechanism. In particular, after every one second you will halt the normal execution of your code and call a function that will remove one message from each outgoing queue (in your network) and will insert it in the incoming queue of next router or machine and then resume the normal execution of the code. As a consequence, message in any of the outgoing queue will not leave the outgoing queue unless the (interrupt) function is called. For this purpose, your program may register for timer interrupt using the functional calls of the operating system. As a result, rough structure of your program may be as follows:

- A function to handle timer interrupts and move messages from outgoing to incoming queues.
- A for or while loop to perform routing and gets user input from command line or through files.

Commands: In addition to the commands mentioned in the phase 1, you have to implement the following additional commands.

- **Change Edge Weight (change edge)**: Your program should be able to change the weights of the edges in the network graph dynamically during execution and accordingly update the routing tables

(by re-applying Dijkstra algorithm) and thereby influencing routing of messages at runtime. The new edge weights can be entered from the command prompt or by providing an adjacency matrix in cvs file in the format described above. For example,

- **change edge** network.cvs,
- **change edge** R1, R2, 25
- **change edge** M1, R5, 25

Additional details and advice

Your program should be menu driven and enable users to perform different operations mentioned above. Moreover, memory allocation should be dynamic. Make sure to define a constructor and destructor for all classes. Your destructors must deallocate all dynamically allocated memory.
For understanding Splay trees you have to read the chapter 22 of the book "Data structures and problem solving using C++" by Mark Allen Weiss. The book is uploaded along with the project. The chapter 22 starts at page 795 of the book. In chapter 22, you can safely skip the discussions about the time complexity as well as proofs and bounds. In particular, you have to implement top-down Splay trees for this project. It is important to mention that Splay trees are part of your data structure course and may appear in your final exam.

What to submit

Submit your code for this project, programmed in C++ using Visual Studio 2017. A document highlighting the design in terms of relationships/associations between different classes of your program must be submitted. The code needs to be well documented so that grader can get a good idea of what each of your procedures do.
Program modularity, clarity, documentation etc., along with correct implementation will be considered for grading.

Good Luck!