Assignment 5

DUE DATE: SUNDAY AUGUST 14, 2022 AT 11:59 PM

Notes:

- Submit ONE .java file for question 1. The .java file must contain ALL of the source code for the question. See the Programming Standards document for how to store several classes in the same file.
- Submit ONE .pdf file for question 2.
- Each filename must have the format `<your last name><your first name>A5Q1.java` (e.g. `SmithJohnA5Q1.java`). Please use your name as shown in UM Learn.
- Do not submit any output. Your code will be run during marking.
- Your program must compile, run, and end normally (not crash or get stuck in an infinite loop) to receive any marks. See the Assignment Information file for some tips on how to make sure your code runs for the markers.
- Assignments must follow the Programming Standards posted in UM Learn.
- Assignment submissions are only accepted via UM Learn. Submissions by email will not be accepted.
- You may submit your assignment multiple times, but only the most recent version will be marked.
- Assignments become late immediately after the posted due date and time. Late assignments will be accepted up to 49 hours after that time, at a penalty of 2% per hour or portion thereof. After 49 hours, an assignment is worth 0 marks and will no longer be accepted.
- The time of the last submission controls the late penalty for the entire assignment.
- These assignments are your chance to learn the material for the tests. *Code your assignments independently*. We use software to compare all submitted assignments to each other, and pursue academic dishonestly vigorously.

Question 1: Simulating a Hospital Emergency Room [17 marks]

This question can be done after Week 11.

You will implement a priority queue and use it to simulate a hospital emergency room. Input will be a list of people arriving at the ER, read from a txt file (described below). Output will be a list of events in the ER, where each event is either (i) a patient arriving at the hospital, or (ii) the doctor completing a treatment and becoming available to treat another patient, or (iii) a patient being called in to see the doctor.

The Patient Class

Create a Patient class as follows:

- Each patient will have a unique patient number (integer). For the emergency room simulation, the patients will be given a patient number in the order they arrive at the emergency room, beginning with 1.
- Each patient will have an urgency (an integer between 1 and 10), where 1 is low priority (the least urgent cases) and 10 is high priority (the most urgent cases).
- Each patient will have a treatment time (an integer representing a number of minutes). This represents the amount of time that the doctor will be occupied treating that patient.
- Include public methods to get the patient number, urgency, and treatment time.
- Write a toString method that returns a String containing the Patient information (and use it while debugging). Depending on the desired format of the information, methods outside the Patient class that need to print patient information could use the toString method, or use the get methods to retrieve data to be included in output.

The Priority Queue Class

You will use a priority queue of patients to determine the order that patients are seen by the doctor. Before writing the code to simulate the emergency room, you should implement the priority queue as below. Remember to test the priority queue as you develop it.

- Your priority queue must be implemented using a **heap** of Patients.
- The public methods for your priority queue will be **insert** (add a new item to the queue), **deleteMax** (remove the highest priority item from the priority queue), **peek** (peek at the highest priority item in the priority queue), and **isEmpty**.
- The highest priority patients should be located at the top of the heap (front of the priority queue).
- The **isEmpty** method will return true if the queue contains no items and false otherwise.
- The **insert** method will insert a patient into the priority queue, so that the highest priority patient is at the front of the queue (i.e. you will pass insert a Patient object).
- The **deleteMax** method will return the patient at the front of the priority queue (i.e. deleteMax will return a Patient object), and will remove that patient from the queue.

- The **peek** method will return the patient at the front of the priority queue, but will leave that patient in the queue.

The Emergency Room Simulation

Simulate an emergency room with one doctor. As patients arrive, they are placed in a priority queue according to the urgency of their case. When the doctor is available (e.g. when one treatment ends), the next patient in the queue will be called in.

- Assume that the input file is named `patients.txt` and stored in the same directory as your .java file. The input file contains a list of patients, one per line. Each line contains the arrival time, the urgency, and the required time with the doctor. The integers are separated by a single space. See the sample input below.
- Use a "clock" that starts at 0 when the simulation begins, and tracks the number of minutes that have elapsed in the simulation. For efficiency, the clock time should "jump" from one event to the next, and not run through all minutes one by one.
- The arrivals (input) file is in order by time of arrival. You should read in only one arrival at a time, and only read the next arrival when the previous has been entered in the queue.
- A patient should only be entered in the queue when the clock time is equal to the patient's arrival time. That is, the queue should always represent the situation in the waiting room at the current time.
- When the doctor is free, and there is a patient waiting, the patient at the front of the priority queue will be called in for treatment.
- Print a statement for each event (when a patient arrives at the hospital, when a patient is called in to see the doctor, when the doctor finishes treating a patient and becomes available). Use a format similar to the sample output below.
- The output must be ordered by the event time, but might not be exactly as below. For example, depending on your implementation, you might have an arrival printed before or after a treatment starting at the same time.

Sample Program Input

```
2 4 15

5 3 5

7 9 12

14 5 9

131 2 5

138 3 10
```

Sample Program Output

```
Doctor is available at time = 0

Patient 1 arrived at time = 2 with urgency = 4 and treatment time =
15.

Doctor is available at time = 2

Patient 1 in for treatment at time = 2 with urgency = 4 and treatment
time = 15.

Patient 2 arrived at time = 5 with urgency = 3 and treatment time = 5.

Patient 3 arrived at time = 7 with urgency = 9 and treatment time =
12.

Patient 4 arrived at time = 14 with urgency = 5 and treatment time =
9.

Doctor is available at time = 17

Patient 3 in for treatment at time = 17 with urgency = 9 and treatment
time = 12.

Doctor is available at time = 29

Patient 4 in for treatment at time = 29 with urgency = 5 and treatment
time = 9.

Doctor is available at time = 38

Patient 2 in for treatment at time = 38 with urgency = 3 and treatment
time = 5.

Doctor is available at time = 43

Patient 5 arrived at time = 131 with urgency = 2 and treatment time =
5.

Doctor is available at time = 131

Patient 5 in for treatment at time = 131 with urgency = 2 and
treatment time = 5.

Doctor is available at time = 136

Patient 6 arrived at time = 138 with urgency = 3 and treatment time =
10.

Doctor is available at time = 138

Patient 6 in for treatment at time = 138 with urgency = 3 and
treatment time = 10.

Doctor is available at time = 148
```

Additional Notes

- You may assume that the input file will be free of errors. Each line in the file will represent one patient and consist of three integers, separated by a space. The urgency will always be a valid integer between 1 and 10. The treatment duration will be a valid positive integer.
- You may assume that there will be a maximum of one arrival per minute. That is, no two patients listed in the input file will have the same arrival time.
- It is *STRONGLY* recommended that you create your own test data, and thoroughly test your code.
- Submit ONE .java file containing all the source code for question 1 (i.e. place all classes in the same file). Do not submit any output. Your code will be run during marking. Make sure that your code compiles and runs without errors (see the Assignment Information file for some common issues).

[Programming Standards are worth 4 marks]

Question 2: Graphs [24 marks]

This question requires material from Week 12.

For **EACH** of the two graphs given below, complete the following four items. No code is required for this question.

- Draw the graph corresponding to the given adjacency matrix (graph 1) / adjacency list (graph 2). These graphs have more nodes and edges than many of the examples in the course materials. There will be many edges and it will not be possible to draw a pretty graph. Do your best to make it legible.
- Complete a depth-first traversal of the graph, beginning at **vertex B**. Format your answer in **a table** containing (i) the event (visit/pop), (ii) the stack contents, and (iii) the current depth-first list of nodes, as in the depth-first traversal example in Week 12. Once the traversal is complete, write out the depth-first **list of nodes**, in the order they were visited.
- Complete a breadth-first traversal of the graph, beginning at **vertex L**. Format your answer in **a table** containing (i) the event (visit/remove), (ii) the queue contents, (iii) the current vertex, and (iv) the current breadth-first list of nodes, as in the breadth-first traversal example in Week 12. Once the traversal is complete, write out the breadth-first **list of nodes**, in the order they were visited.
- Use Dijkstra's shortest path algorithm to find the lowest-cost paths from **vertex H** to all other vertices. Submit **a table** showing the details of each step of the algorithm. At each step, show the current least cost distance to each vertex, the previous vertex on that least cost path, and mark completed paths with an asterisk. That is, format your answer in a table, as in the Dijkstra's algorithm example in Week 12. Once the algorithm is complete, write out the lowest-cost **path from H to each other vertex** (i.e. list the vertices along each path and state the total cost of each path).

Caution: Be aware of how the graphs would be stored in a computer. Use the adjacency matrix or adjacency list to determine the order that nodes are visited. Remember that a computer will process the vertices in a very systematic way.

You may use Photoshop, Powerpoint, or other software to draw your graphs and tables, or you may draw your graphs and tables by hand on paper and scan/photograph your solution. Whatever method you choose, convert your solution into a pdf file. Each graph should occupy a full page. Your answer (including all vertices, lines, and labels) must be legible after conversion to pdf, and be well-organized so that it can be easily marked.

Your pdf file must be named `<your last name><your first name>A5Q2.pdf` (e.g. `SmithJohnA5Q2.pdf`). Submit your pdf file to the Assignment 5 dropbox in UM Learn, along with your solution to question 1.

Graph 1: This graph is stored in an adjacency matrix.

to from	A	B	C	D	E	F	G	H	I	J	K	L
A	0	40	20	20	30	80	0	0	0	0	70	0
B	0	0	0	20	40	0	0	0	0	0	5	0
C	0	0	0	0	0	30	0	50	0	0	30	30
D	0	0	10	0	0	0	0	0	0	30	50	0
E	0	0	0	0	0	0	40	0	0	20	0	20
F	10	0	0	0	0	0	0	30	0	0	0	60
G	0	0	30	90	0	60	0	0	0	0	80	0
H	60	0	0	50	10	0	70	0	0	0	0	0
I	0	0	0	0	0	0	0	40	0	70	0	10
J	10	60	0	0	0	80	10	0	0	0	90	0
K	20	60	0	0	20	0	20	0	0	0	0	0
L	0	70	60	0	0	90	0	0	0	0	0	0

Graph 2: This graph is stored in an adjacency list, where each list lists the vertices adjacent to a given vertex, and each entry in the list gives the adjacent vertex and the edge weight. (That is, the arrows below represent links in the linked lists, not edges in the graph.)

A: (E, 70) → (G, 40) → (H, 60) → (K, 30)

B: (D, 70) → (E, 50) → (I, 20) → (L, 40)

C: (F, 40) → (G, 20) → (H, 70) → (K, 10) → (L, 50)

D: (B, 70) → (G, 50) → (H, 90) → (I, 30) → (J, 60) → (L, 10)

E: (A, 70) → (B, 50) → (I, 10) → (L, 30)

F: (C, 40) → (G, 60) → (I, 80) → (J, 50) → (L, 30)

G: (A, 40) → (C, 20) → (D, 50) → (F, 60) → (I, 10) → (K, 30)

H: (A, 60) → (C, 70) → (D, 90) → (J, 20)

I: (B, 20) → (D, 30) → (E, 10) → (F, 80) → (G, 10) → (K, 60)

J: (D, 60) → (F, 50) → (H, 20) → (K, 80)

K: (A, 30) → (C, 10) → (G, 30) → (I, 60) → (J, 80)

L: (B, 40) → (C, 50) → (D, 10) → (E, 30) → (F, 30)