CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 8, Oct 2018

Heaps / Priority Queues

OBJECTIVES

1. **Implement a priority queue using an array.**
2. **Add and remove elements to/from the priority queue.**

Background

Imagine a hospital emergency room. The emergency room does not see patients on a first-come first-served basis, but rather the most urgent patients are seen before patients whose injuries are less severe or non-life-threatening. You will implement a priority queue such that patients with the more severe injuries are treated before those who can wait. You will also track the total time that the ER doctors have spent treating patients.

Assignment

- You will be creating a class called PriorityQueue, which is defined in the PriorityQueue.hpp header file on Moodle. **Do not** modify this header file.
- You will also need to write a main function. We will assume your main function is written in a separate file while autograding. If you would like to write all of your code in one file, you will have to split it up when submitting it.
- Your program will provide an option to read patient data from a file. An example file can be found on Moodle called *patientFile.txt*, and it is in the format:

```
<Name 1> <Severity 1> <Treatement Time 1>
<Name 2> <Severity 2> <Treatement Time 2>
…
<Name n> <Severity n> <Treatement Time n>
```

- Your program will take exactly one command line argument - a number that represents the maximum queue size.

CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 8, Oct 2018

- Your main function should create an instance of the PriorityQueue class with the size that is passed in as a command line argument. It should then repeatedly display a menu to the user, just like in previous labs. The code to print this menu can be written like:

```
cout << "======Main Menu======" << endl;
cout << "1. Get Patient Information from File" << endl;
cout << "2. Add Patient to Priority Queue" << endl;
cout << "3. Show Next Patient" << endl;
cout << "4. Treat Next Patient" << endl;
cout << "5. Treat Entire Queue" << endl;
cout << "6. Quit" << endl;
```

Each option should do the following:

1. ***Get Patient Information from File***

   This option should ask the user for a filename using the following print statement:

   ```
   cout << "Enter filename:" << endl;
   ```

   It should then read the patient data from that file into the queue based on each patient's injury severity. If there are too many patients for the queue to store, it should read as many as it can before printing the following:

   ```
   cout << "Priority Queue full. Send remaining patients
   to another hospital." << endl;
   ```

2. ***Add Patient to Priority Queue:***

   This option should ask for a new patient's name, injury severity, and treatment time, then insert them into the priority queue based on their injury severity. You should use the following print statements to ask the user for information:

   ```
   cout << "Enter Patient Name:" << endl;
   cout << "Enter Injury Severity:" << endl;
   cout << "Enter Treatment Time:" << endl;
   ```

   If the priority queue is already full, it should print the following message instead:

   ```
   cout << "Priority Queue full. Send Patient to another
        hospital." << endl;
   ```

3. ***Show Next Patient***

   This option should print out information on the next patient using the print statements below:

   ```
   cout << "Patient Name: " << patientName << endl;
   cout << "Injury Severity: " << patientInjurySeverity
        << endl;
   cout << "Treatment Time: " << patientTreatmentTime
        << endl;
   ```

   If the queue is empty, it should print the following instead:

   ```
   cout << "Queue empty." << endl;
   ```

4. ***Treat Next Patient***

   This option should remove a patient from the queue, and print the name of the treated patient as well as the total time spent treating patients. It should use the following format to print:

   ```
   cout<<"Patient Name: "<< patientName
        << " - Total Time Treating Patients: "
        << totalTreatmentTime << endl;
   ```

   If the queue is already empty, it should print the following instead:

   ```
   cout << "Queue empty." << endl;
   ```

5. ***Treat Entire Queue***

   This option should treat all the patients until the queue is empty. For each one, it should print the same information as option 4 (Treat Next Patient). If the queue is already empty, it should print the following instead:

   ```
   cout << "Queue empty." << endl;
   ```

6. ***Quit***

   This option should print out the following friendly goodbye, then quit:

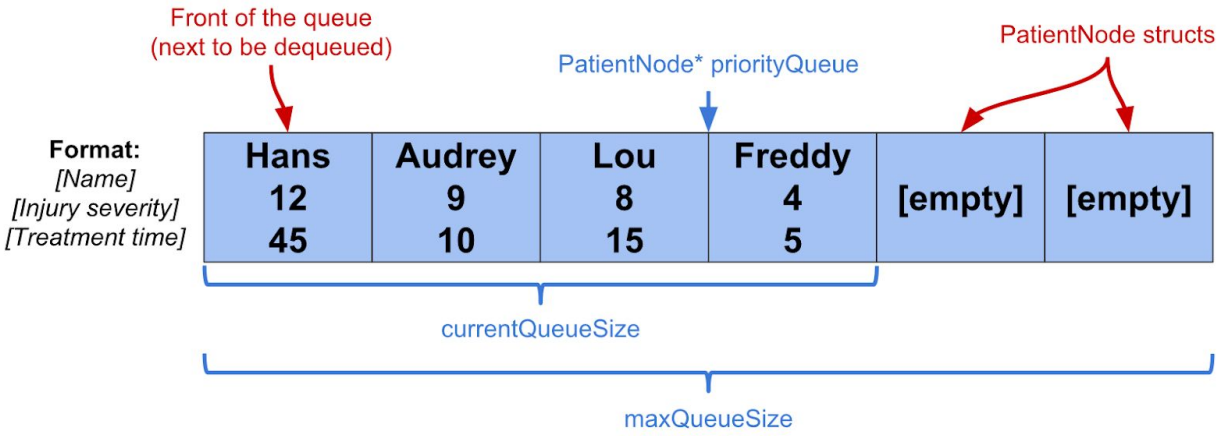
   ```
   cout << "Goodbye!" << endl;
   ```

CSCI 2270 – Data Structures - Section 100

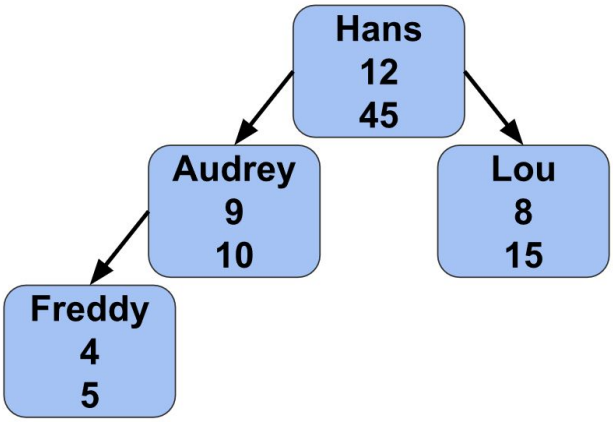
Instructor: Shayon Gupta

Assignment 8, Oct 2018

- To build your queue, you will use the PriorityQueue class defined in the PriorityQueue.hpp header file on Moodle. Visually, the priority queue that this class implements would look like this:



This implementation uses an array to store all the values linearly, although the underlying structure can also be thought of as a tree as discussed in class:



The class implements the following functions:

- **`PriorityQueue::PriorityQueue(int queueSize)`**
  The constructor function should dynamically allocate an array to store the patients of size `queueSize`. This size is passed as a command line argument.

- **`PriorityQueue::~PriorityQueue()`**
  The deconstructor should free all the memory that the priority queue allocated.

CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 8, Oct 2018

- **`void enqueue(string _name, int _injurySeverity, int _treatmentTime)`**
  This function should add a new patient to the appropriate position in the queue, based on their injury severity. It should use the `repairUpwards` function to accomplish this.

- **`void dequeue()`**
  This function should remove the patient at the front of the queue. It should use the repairDownwards function to accomplish this.

- **`string peekName()`**
  This function should return the name of the patient at the front of the queue.

- **`int peekInjurySeverity()`**
  This function should return the injury severity of the patient at the front of the queue.

- **`int peekTreatmentTime()`**
  This function should return the treatment time of the patient at the front of the queue.

- **`bool isFull()`**
  This function should return whether or not the priority queue is full.

- **`bool isEmpty()`**
  This function should return whether or not the priority queue is empty.

- **`void repairUpwards(int nodeIndex)`**
  This function should re-order a tree after a new node has been added at index nodeIndex, and do so by swapping that node with its parent until the tree is valid again.

- **`void repairDownwards(int nodeIndex)`**
  This function should re-order a tree after a node has been removed at index nodeIndex, and do so by swapping that node with one of its children until the tree is valid again.

Submission

Submit your code on Moodle by following the directions on Assignment 8 Submit. You must submit an answer to be eligible for interview grading!