CSCI 2270 – Data Structures - Section 100

Instructors: Shayon Gupta, Ashutosh Trivedi, Maciej Zagrodzki

Assignment 8 - Heaps / Priority Queues

OBJECTIVES

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

Overview

Imagine you run a fast food restaurant and you want to streamline the way you do business to serve as many people as possible each day. You decide that the best strategy for maximizing customer turnover is to prioritize smaller groups over larger groups. For any two groups of the same size, you prioritize customers whose meals are faster to cook. Now, your task is to implement some software that automates this kind of priority queue with a binary heap. Each party of patrons will be represented by the following struct (already included in the header file)

```
struct GroupNode
{
    std::string groupName; // name of the person who ordered
    int groupSize; // number of people in the group
    int cookingTime; // time in minutes it takes to cook the order
};
```

PriorityQueue Class

PriorityQueue(int queueSize)

➔ Parameterized constructor. Initialize **maxQueueSize** to **queueSize** and **currentQueueSize** to 0. Dynamically allocate an array from class variable **priorityQueue** of size **queueSize**.

~PriorityQueue()

➔ Deconstructor. Deallocate all memory that was dynamically allocated.

void enqueue (std::string _groupName, int _groupSize, int _cookingTime)

➔ Add a GroupNode to the end of the queue with fields **_groupName**, **_groupSize**, and **_cookingTime**. This entails adding it to the end of your **priorityQueue** array (the end of the array is specified by the index **currentQueueSize**). Then repair the heap with your **repairUpward** function.

➔ If the queue is full, then show the below message

```
cout << "Heap full, cannot enqueue" << endl;
```

void dequeue()

- ➔ Replace the first GroupNode at index 0 with the last node in the priority queue and repair the heap with your **repairDownward** function
- ➔ If the queue is empty, then show the below message

```
cout << "Heap empty, cannot dequeue" << endl;
```

GroupNode peek()

- ➔ Return the front node of the priority queue. If the queue is empty, then show the below error message

```
cout << "Heap empty, nothing to peek" << endl;
```

bool isFull()

- ➔ Returns true if the the array is at full capacity

bool isEmpty()

- ➔ Returns true if the array is empty

void repairUpward(int nodeIndex)

- ➔ If the node at **nodeIndex** has higher priority than its parent, swap it with its parent. Continue doing this until the heap property is satisfied. *You may assume that no two nodes have the same priority.*

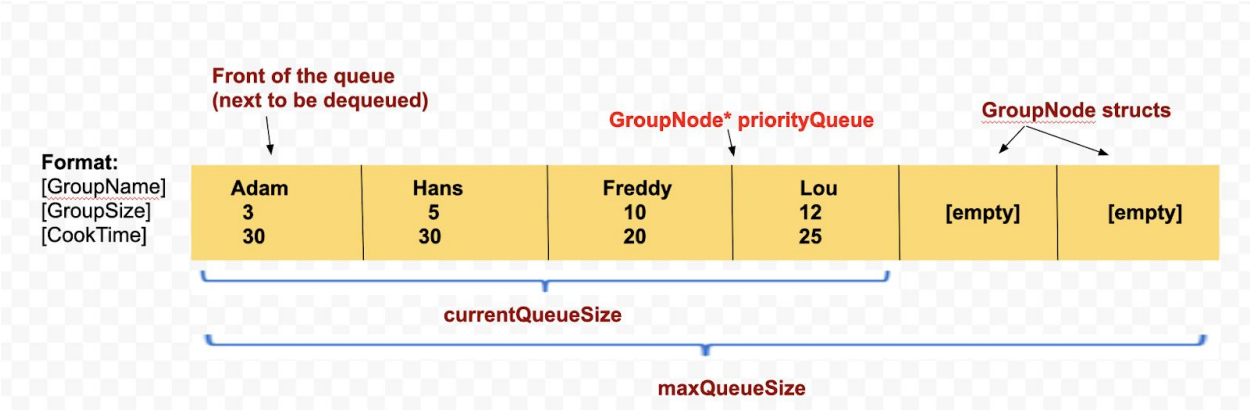
void repairDownward(int nodeIndex)

- ➔ If the node at **nodeIndex** has lower priority than one or both of its children then swap it with its highest priority child. Continue doing this until the heap property is satisfied. *You may assume that no two nodes have the same priority.*

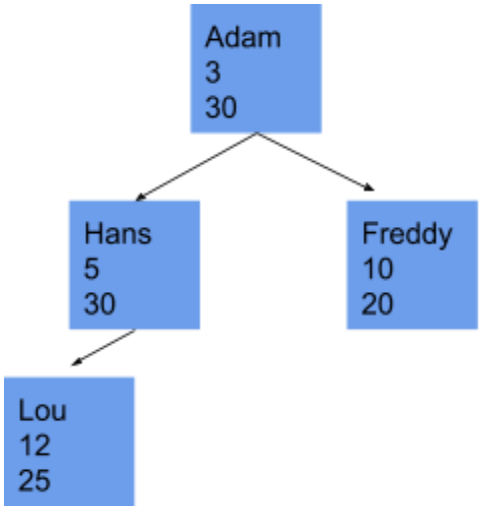
Visually, the priority queue that this class implements would look like this:

CSCI 2270 – Data Structures - Section 100

Instructors: Shayon Gupta, Ashutosh Trivedi, Maciej Zagrodzki



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:



Driver

- 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 groups data from a file. An example file can be found on Moodle callede RestaurantData.*txt*, and it is in the format:

```
<GroupName 1> <GroupSize 1> <CookingTime 1>
<GroupName 2> <GroupSize 2> <CookingTime 2>
```

…
<GroupName n> <GroupSize n> <CookingTime n>

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

- 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 group information from file" << endl;
cout << "2. Add a group to Priority Queue" << endl;
cout << "3. Show next group in the queue" << endl;
cout << "4. Serve Next group" << endl;
cout << "5. Serve Entire Queue" << endl;
cout << "6. Quit" << endl;
```

Each option should do the following:

1. ***Get Group 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 group data from that file into the queue based on each group's size. If there are too many groups for the queue to store, it should read as many as it can before printing the following:

   ```
   cout << "Heap full, cannot enqueue" << endl;
   ```

2. ***Add Group to Priority Queue:***
   This option should ask for a new Group name, Group size, and Estimated Cooking time, then insert them into the priority queue based on their Group size (***smaller groups get served first)***. If there is a tie in the Group size, it should prioritize the Group with the ***lower cooking time***. You should use the following print statements to ask the user for information:

```
cout << "Enter Group Name:" << endl;
cout << "Enter Group Size:" << endl;
cout << "Enter Estimated Cooking Time:" << endl;
```

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

```
cout << "Heap full, cannot enqueue" << endl;
```

3. ***Show Next Group in the queue***
   This option should print out information of the next group waiting in the queue using the print statements below:

```
/* For a priority queue object myQueue */
cout << "Group Name:" << myQueue.peek().groupName << endl;
cout << "Group Size:" << myQueue.peek().groupSize << endl;
cout << "Estimated Cooking Time:"<< myQueue.peek().cookingTime
<< endl;
```

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

```
cout << "Heap empty, nothing to peek" << endl;
```

4. ***Serve Next Group***
   This option should remove a group from the queue, and print the name of the served group as well as the total cook time for the group (**including the cooking time of the highest priority groups**). It should use the following format to print:

```
/* For a priority queue object myQueue
  totalCookTime = totalCookTime of prioritized groups that
were served previously + cook time of the current group
*/
cout << "Group Name:" << myQueue.peek().groupName
     << " - Total Cook Time for the Group: "
     << totalCookTime << endl;
```

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

```
cout << "Heap empty, cannot dequeue" << endl;
```

5. ***Serve Entire Queue***
   This option should serve all the groups until the queue is empty. For each one, it should print the same information as option 4 (Serve Next Group). If the queue is already empty, it should print the following instead:

```
cout << "Heap empty, cannot dequeue" << endl;
```

6. ***Quit***
   This option should print out the following friendly goodbye, then quit:

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

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!