CIS 313, Intermediate Data Structures
Fall 2018

CIS 313 Lab 3

Due: November 16, 2018

This lab involves implementing a Priority Queue using a Max-Heap

Overview

Use a binary Max-Heap to implement a Priority Queue.
Your priority queue class will be a wrapper for the functions in your heap class.
Your heap should be implemented using an array.
(ie) if a node is in array[k], then its left child is in array[k*2], and its right child is in array[k*2+1]
note: the root is in array[1]

Fill out all of the methods in the provided skeleton code.
You may add additional methods, but should NOT add additional public methods or public fields.
You also should not change the name of any of the classes or files.

In particular, you will implement the following functionality for your priority queue

Insert
Add priority p to the priority queue

Maximum
Return the highest priority from the priority queue

extractMax
Remove and return the highest priority from the priority queue

isEmpty
Print "Not Empty" if the priority queue is not empty, otherwise print "Empty"

print
Print out "Current Queue: " followed by each element separated by a comma. Do not add spaces between your elements. After you have printed all the elements in the Priority Queue make sure you print a new line character. This should represent the current structure of your heap.

Note: Your Priority Queue class should just be a wrapper for your heap class. Most of the work will be done in the heap class.

In particular, you will implement the following functionality for your heap

Insert
When adding a node to your heap, remember that for every node n, the value in n is greater than or equal to the values of its children, but your heap must also maintain the correct shape.

(ie there is at most one node with one child, and that child is the left child.)
(Additionally, that child is farthest left possible.)

Maximum
Return the maximum value in the heap

extractMax
Remove and return the maximum value in the heap

Heapify
Used to maintain the structure of the heap after an element is added or removed from the heap

Input Description

The input will be a text file, for example *inSample.txt* below will be provided. The first line will contain an integer N, which is the number of lines to follow. Each of the N lines contain a different set of words specifying a task along with a number (if applicable). You should create an empty heap (with root null), and then perform a sequence of actions on that tree.

Note: You should implement your priority queue and heap with generics, but create a priority queue that takes in integers.

```
11
insert 7
insert 10
insert 9
print
isEmpty
insert 8
maximum
insert 21
extractMax
print
extractMax
```

Note: When using an editor, you may also manually type in input to the command window. However, you will be tested with a file similar to *inSample.txt*.

Output Description

Using the sample input above, your program should output:

```
Current Queue: 10,7,9
Not Empty
10
21
Current Queue: 10,8,9,7
10
```

Testing Protocol

We will test your program in the same fashion as previous labs. We strongly suggest you test your program in the following ways:

- While creating it
- With inSample.txt
- With the provided test.sh file found in `/home/users/smergend/public/cs313/lab3` on ix-dev.

Grading

This assignment will be graded as follows:

Correctness 40%
Your program compiles without errors (including the submitted files NOT containing package names at the top. Delete the package name from the top of the files before you submit them): 10%
Your program runs without errors: 10%
Your program produces the correct output: 20%

Implementation 40%
Insert, extractMax, Maximum, isEmpty, print all work correctly for the priority queue: 25%
Insert, extractMax, and Maximum all work correctly for the heap: 15%

Documentation 20%
To earn points for implementation, your code must be clear enough for us to understand

Further, you may not use any data structures from the Java standard library or the C foreign function interface

Extra Credit 20%

To receive points for the extra credit you will create produce a heap from an unordered array in linear time:

- Create a new public method in your heap class called buildHeap
- This method will take in an array A (of arbitrary size n)
- After buildHeap finishes, the heap should be represented by an array that contains all the same elements as A, but also maintains all the heap properties
  - This process needs to be completed in $O(n)$ time
  - Note that you will only be given an array that contains a fully binary tree for simplicity
    - (ie) with $2^n - 1$ nodes for some n
- You will need to create an extra switch statement in lab3.java that runs on the command build
  - The command will look like the following:
    - build [1,2,3,4,5,6,7]