CIS 2168: Assignment #9

Due on Wednesday, November 12, 2014

11:59pm

Anwar Mamat

Problem 1

Priority Queue (100%) In class, we implemented the priority queue ADT using binary heap stored in an array. In this assignment, you will implement the priority queue using a heap-ordered binary tree. We DO NOT store the binary tree in the array. Instead we use special binary tree, a triply linked structure. Each node has three links. "left" and "right" child to traverse down the tree and "parent" to traverse up the tree. For example: if you want to make node n2 as the left child of node n1, you will do:

```
n1.left = n2;
n2.parent = n1;
```

You will have to implement "insert" and "remove" using the algorithms "swim" and "sink" by operating the tree nodes, instead of manipulating the array. **Your implementation should guarantee logarithmic running time for "insert" and "remove", and constant time for "top" even if the size of the priority queue is not known ahead of the time.** A simple linked list implementation runs in linear time. If you do that, you will not get credit.
You will implement this priority queue interface:

```
public interface PriorityQueue<T extends Comparable<T> >
{
     void insert(T t);
     void remove() throws EmptyQueueException;
     T top() throws EmptyQueueException;
     boolean empty();
}
```

The Node class is:

```
class Node{
       T key;
       Node left,right;  //left, right children
       Node parent; //reference to the parent
}
```

For example: for 1,4,7,6,3,13,14,10,8, you will create the binary tree:

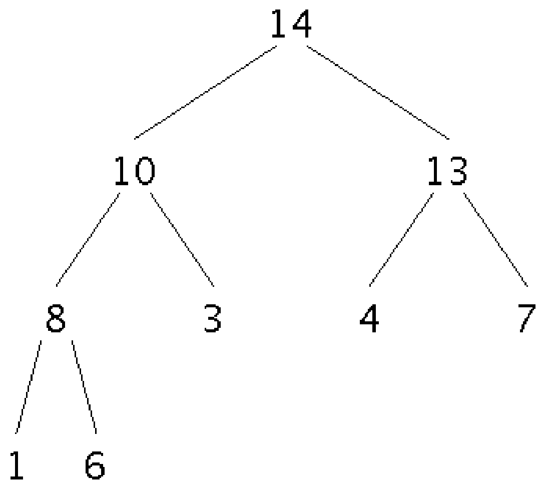

Figure 1: Binary Heap

Problem 1 continued on next page. . .

If you remove an item, 14 should be removed. 6 replaces 14, and we call “sink” to fix the violation. Test your code with different input data. Make sure “insert”, “top” and “remove” work as we learned in class. Download the attached “assignment9.zip” file, extract it and open the “assignment9” project using NetBeans. Implement the “insert”, “remove”, “top”, and ”empty” method in the “MaxPQTree” class. When you execute the project, entering “i 10” will insert 10 into the priority queue, while the command “r” removes the largest number from the queue. Entering “done” terminates the program.

Testing your program

Test 1

We insert 50 random numbers to the priority queue, and remove all numbers. “BinaryTreeView” class displays the binary heap, so that you can see if your priority queue is correct.

```java
MaxPQTree<Integer> pq = new MaxPQTree();
        BinaryTreeView<Integer> btv = new BinaryTreeView<Integer>(pq, 600, 500);
        for(int i = 1; i < 50; i++){
            pq.insert((int)(Math.random()*1000));
            btv.refresh();
            Thread.sleep(100L);
        }

        while(!pq.empty()){
            pq.remove();
            btv.refresh();
            Thread.sleep(100L);
        }
        btv.close();
```

This code will generate a tree as shown in Figure 2.

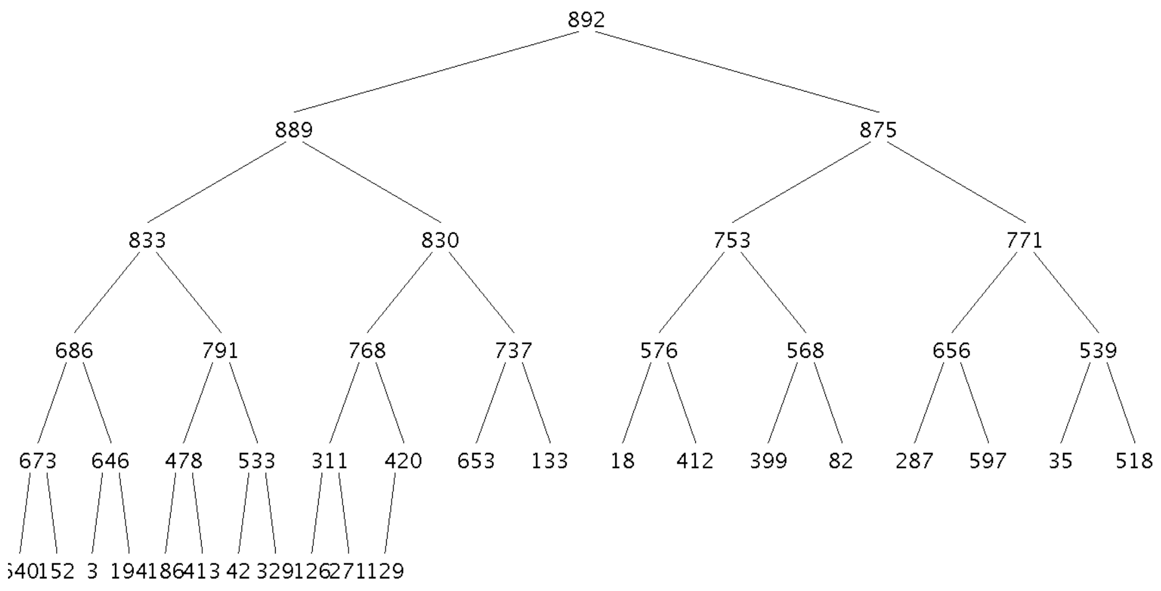

Figure 2: Binary Heap

Test 2

In this test, we manually enter the data to build priority queue. Run the program, at the command line, you will see “Enter next value:”. You can insert a number into the priority queue with the command “i number”, remove a number by entering “r”, and display the top value by entering “t”. The binary tree will be displayed, so that you can check if your priory queue is correct.

```java
MaxPQTree<Integer> pq = new MaxPQTree();
        BinaryTreeView<Integer> btv = new BinaryTreeView<Integer>(pq, 600, 500);
        String inp;
        inp = InputHelper.getStringInput("Enter next value:");
        while (!inp.equals("done")){
            String inputs[] = inp.split(" ");
            try{
                if(inputs[0].toLowerCase().equals("i")){
                    pq.insert(Integer.parseInt(inputs[1]));
                }else if(inputs[0].toLowerCase().equals("t")){
                    System.out.println(pq.top());
                }else if(inputs[0].toLowerCase().equals("r")){
                    pq.remove();
                    System.out.println("element is removed");
                }
                btv.refresh();
            }catch(EmptyQueueException ex)
            {
                System.err.println(ex);
            }catch(ArrayIndexOutOfBoundsException ex)
            {
                System.err.println(ex);
            }
            inp = InputHelper.getStringInput("Enter next value:");
        }
        System.out.println("done.");
```

Test 3

In this test, we test the efficiency of your priority queue. We pick M smallest numbers from N random numbers. N is huge and M is large. You can change N between 1,000,000-10,000,000 and change M between 1,000-100,000. For N=10,000,000 and M=100,000, the estimated execution time should be around one second. If your program runs slow, your “insert” and “remove” may not be taking worst case $O(logn)$ time. If your program takes $O(n)$ time, it is not correct.

```java
// pick M smallest numbers from N random numbers;
        Random r = new Random();
        MaxPQTree<Integer> pq = new MaxPQTree();
        final int N = 10000000;
        final int M = 100000;
        int i;
        long tStart = System.currentTimeMillis();
        for(i = 1; i < M; i++){
            int t = r.nextInt(N);
                pq.insert(t);
        }
        for(i = M; i < N; i++){
            int t = r.nextInt(N);
            if(pq.top() > t){
                pq.insert(t);
                pq.remove();
            }
        }
        long tEnd = System.currentTimeMillis();

        // display the selected numbers
        /*
        System.out.println(M +" smallest numbers");
        while(!pq.empty()){
            System.out.print(pq.top()+",");
            pq.remove();
        }
        System.out.println("");
        */
        long tDelta = tEnd - tStart;
        double elapsedSeconds = tDelta / 1000.0;
     System.out.println("Elapsed time: " + elapsedSeconds + " seconds");
```

Grading

Homework is 100 points. 80 will reflect functionality and correctness. 20 points on your program will reflect your programming style, documentation. **If you code does not compile, you will not receive any credit.**

Commenting and Documenting Code

Code must be properly commented. The main idea is that the grader should be able to understand your code easily, not have to tear his or her hair out wondering what some statement is doing. The first time you have to deal with poorly commented code (if you haven't already), you will understand how annoying it is. In particular, the top of each code file should contain your name, the course and assignment numbers, and a brief summary of what's in the file. Line-by-line comments should be included as necessary to make the code easy to read. A clear coding style, together with informative variable and function names, will reduce the number of comments required. Obscure code or cryptic function names will cause loss of points (for bad style) and also require more extensive comments.

What to submit

A single zip file called Assignment9_firstname_lastname.zip, where firstname is your first name, and lastname is your last name. In this zip file, put:

1. Java source
2. A README file with:
    - Instructions to compile and run of your code (include a description of command line options).
    - If your solution is not perfect, explain what parts you did and what part you did not do.

- List of files submitted
- All your data and results (in plain text files).
- Anything else you want TA know

3. Submit this zip file to Blackboard