LinkedPQ

(from https://stackoverflow.com/questions/31257243/how-do-i-implement-a-priority-queue-with-explicit-links-using-a-triply-linked-d; 20160929.s

I posted this in case someone gets stuck doing this exercise from Sedgewick, because he doesn't provide a solution for it.

I have written an implementation for maximum oriented priority queue, which can be modified according for any priority.

What I do is assign a size to each subtree of the binary tree, which can be defined recursively as size(x.left) + size(x.right) + 1. I do this do be able to find the last node inserted, to be able to insert and delete maximum in the right order.

How sink() works: Same as in the implementation with an array. We just compare x.left with x.right and see which one is bigger and swap the data in x and max(x.left, x.right), moving down until we bump into a node, whose data is <= x.data or a node that doesn't have any children.

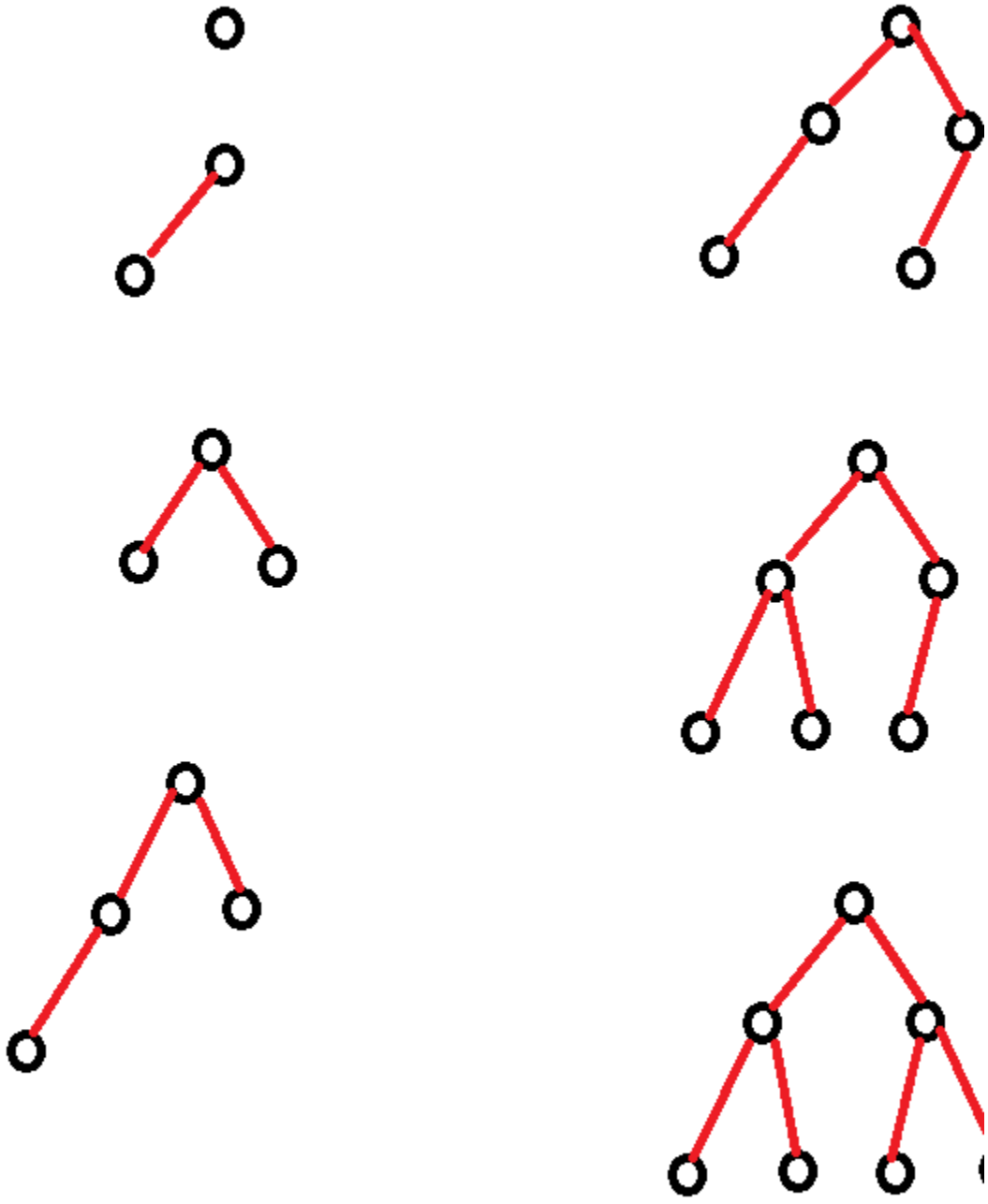
How swim() works: Here I just go up by doing x = x.parent, and swapping the data in x and x.parent, until x.parent == null, or x.data <= x.parent.

How max() works: It just returns root.data.

How delMax() works: I keep the last inserted node in a separate field, called lastInserted. So, I first swap root.data with lastInserted.data. Then I remove lastInserted by unhooking a reference to it, from its parent. Then I reset the lastInserted field to a node that was inserted before. Also we must not forget to decrease the size of every node on the path from root to the deleted node by 1. Then I sink the root data down.

How insert() works: I make a new root, if the priority queue is empty. If it's not empty, I check the sizes of x.left and x.right, if x.left is bigger in size than x.right, I recursively call insert for x.right, else I recursively call insert for x.left. When a null node is reached I return new Node(data, 1). After all the recursive calls are done, I increase the size of all the nodes on the path from root to the newly inserted node.

Here are the pictures for insert():



And here's my java code:

```
public class LinkedPQ<Key extends Comparable<Key>>{
    private class Node{
```

```
    int N;
    Key data;
    Node parent, left, right;
    public Node(Key data, int N){
        this.data = data; this.N = N;
    }
}
// fields
private Node root;
private Node lastInserted;
//helper methods
private int size(Node x){
    if(x == null) return 0;
    return x.N;
}
private void swim(Node x){
    if(x == null) return;
    if(x.parent == null) return; // we're at root
    int cmp = x.data.compareTo(x.parent.data);
    if(cmp > 0){
        swapNodeData(x, x.parent);
        swim(x.parent);
    }
}
private void sink(Node x){
    if(x == null) return;
    Node swapNode;
    if(x.left == null && x.right == null){
        return;
    }
    else if(x.left == null){
        swapNode = x.right;
        int cmp = x.data.compareTo(swapNode.data);
        if(cmp < 0)
            swapNodeData(swapNode, x);
    } else if(x.right == null){
        swapNode = x.left;
        int cmp = x.data.compareTo(swapNode.data);
        if(cmp < 0)
            swapNodeData(swapNode, x);
    } else{
        int cmp = x.left.data.compareTo(x.right.data);
        if(cmp >= 0){
            swapNode = x.left;
        } else{
            swapNode = x.right;
        }
        int cmpParChild = x.data.compareTo(swapNode.data);
        if(cmpParChild < 0) {
            swapNodeData(swapNode, x);
            sink(swapNode);
        }
    }
}
private void swapNodeData(Node x, Node y){
    Key temp = x.data;
```

```
        x.data = y.data;
        y.data = temp;
    }
    private Node insert(Node x, Key data){
        if(x == null){
            lastInserted = new Node(data, 1);
            return lastInserted;
        }
        // compare left and right sizes see where to go
        int leftSize = size(x.left);
        int rightSize = size(x.right);

        if(leftSize <= rightSize){
            // go to left
            Node inserted = insert(x.left, data);
            x.left = inserted;
            inserted.parent = x;
        } else{
            // go to right
            Node inserted = insert(x.right, data);
            x.right = inserted;
            inserted.parent = x;
        }
        x.N = size(x.left) + size(x.right) + 1;
        return x;
    }
    private Node resetLastInserted(Node x){
        if(x == null) return null;
        if(x.left == null && x.right == null) return x;
        if(size(x.right) < size(x.left))return
resetLastInserted(x.left);
        else                            return
resetLastInserted(x.right);
    }
    // public methods
    public void insert(Key data){
        root = insert(root, data);
        swim(lastInserted);
    }
    public Key max(){
        if(root == null) return null;
        return root.data;
    }
    public Key delMax(){
        if(size() == 1){
            Key ret = root.data;
            root = null;
            return ret;
        }
        swapNodeData(root, lastInserted);
        Node lastInsParent = lastInserted.parent;
        Key lastInsData = lastInserted.data;
        if(lastInserted == lastInsParent.left){
            lastInsParent.left = null;
        } else{
            lastInsParent.right = null;
```

```
        }

        Node traverser = lastInserted;

        while(traverser != null){
            traverser.N--;
            traverser = traverser.parent;
        }

        lastInserted = resetLastInserted(root);

        sink(root);

        return lastInsData;
    }
    public int size(){
        return size(root);
    }
    public boolean isEmpty(){
        return size() == 0;
    }
}
```

answered Jul 6 '15 at 23:14

shareimprove this answer edited Jul 6 '15 at 23:21

Pavel
1,169319

add a comment