Assignment 3

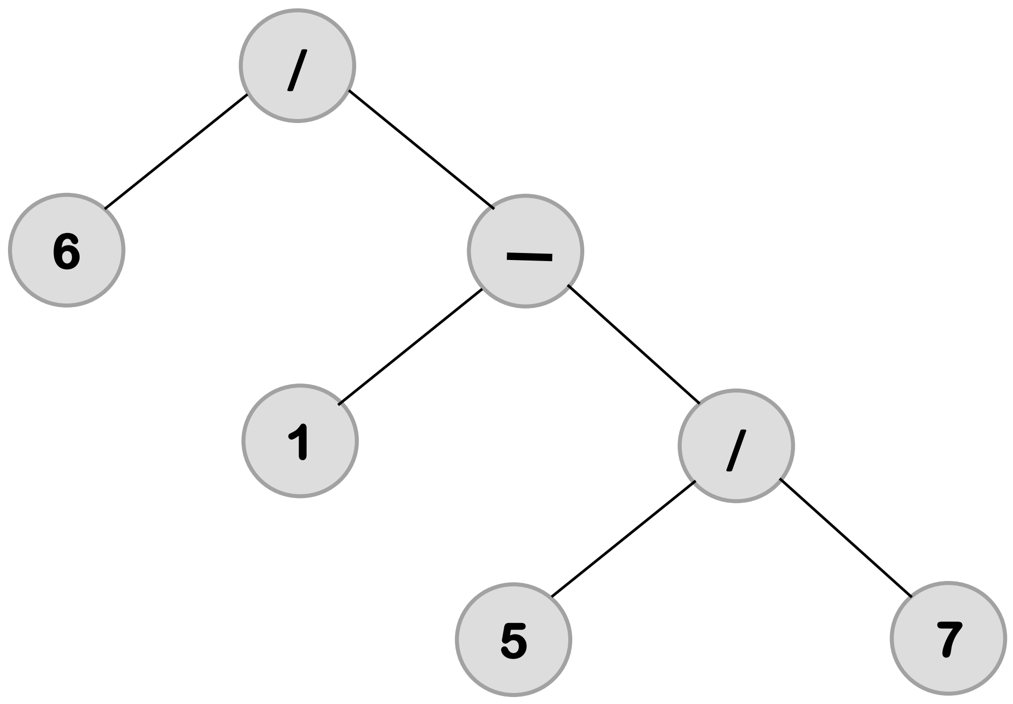
EECS 2011 – Fundamentals of data structures.

Rajkumar Balakrishnan Lakshmi - 213141197

11/17/17

Problem 1: Arithmetic Expression Tree

For this question, the value 21 can be achieved by the following binary tree arrangement. The four external nodes consist of the values {6, 1, 5, 7} respectively and the three internal nodes are {/, -, /}.



Infix - **6 / (1 – 5 / 7)** = 21.

Problem 2: Diameter of a binary tree.

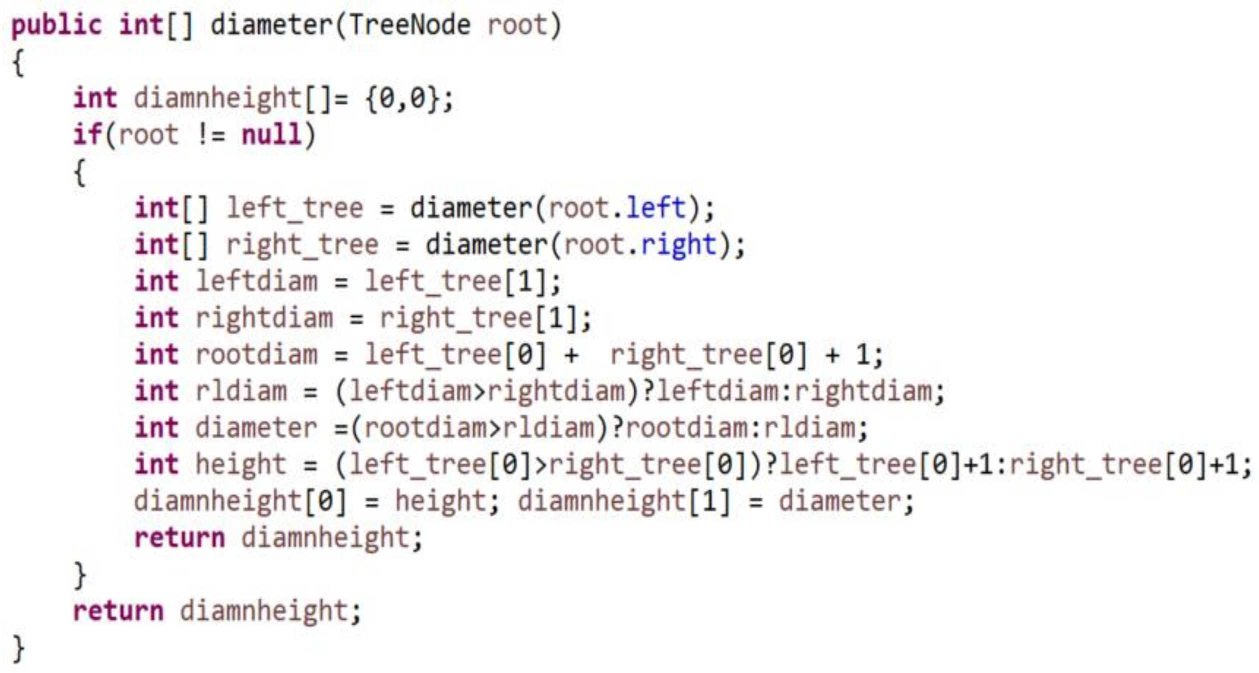
For this question, we are supposed to calculate the diameter of the binary tree i.e., the maximum distance between two external or leaf nodes. There is a probability of three routes to find the diameter of tree:

- The diameter of the tree's left subtree.
- The diameter of the tree's right subtree.
- The sum of height of left and right subtrees with addition of for the root itself.

So, we calculate the diameter of the binary tree by calculating the diameter of the subtrees and their height simultaneously and return the maximum value among them.

Input: Root of the binary tree.

Output: Array of diameter and height of the tree.



```
public int[] diameter(TreeNode root)
{
    int diamnheight[]= {0,0};
    if(root != null)
    {
        int[] left_tree = diameter(root.left);
        int[] right_tree = diameter(root.right);
        int leftdiam = left_tree[1];
        int rightdiam = right_tree[1];
        int rootdiam = left_tree[0] +  right_tree[0] + 1;
        int rldiam = (leftdiam>rightdiam)?leftdiam:rightdiam;
        int diameter =(rootdiam>rldiam)?rootdiam:rldiam;
        int height = (left_tree[0]>right_tree[0])?left_tree[0]+1:right_tree[0]+1;
        diamnheight[0] = height; diamnheight[1] = diameter;
        return diamnheight;
    }
    return diamnheight;
}
```

Time Complexity: o(n)

I used array adt to store the diameter and height of the tree respectively in diamnheight. And similarly, the left_tree and right_tree arrays store their respective diameter and height. After getting the values the diameter of the tree(leftheight + rightheight + 1) is compared to the diameter of left and right subtree and the max among them is stored as the diamter of the tree. All the caluclations are done by recursively visiting each node of the tree only once, thus the time complexity of the algorithm is 0(n). An entire implementation of the algorithm is attached with this assignment as: TreeDiameter.java

Problem 3: Top log n flyers of Tamarindo Airlines.

For this question, we are supposed to find the top log n frequent flyers of Tamarindo airlines. This can be achieved by storing the flyers of the list in a Maximum heap or a heap priority queue with miles as the key respectively. In case of max heap, the details are stored in descending order having miles as the key, where as in a normal heap priority queue we use a technique to store them in descending order having 10/miles as the key. This way most frequent flyers are stores near to the top and the least frequent flyers are stored at the bottom. Then we remove the top log n frequent flyers from the heap created and store them in an array. By this means the top frequent flyers can be isolated and returned. The pseudo code of the above algorithm is:

Input: List of n number of flyers.

Output: Array of top log n frequent flyers.

```
int[] findtopflyers(List nflyer)
{
  Create a heap with all the nflyer stored in it having (10/miles) as the key;

  logn = (Math.log((nflyer.size()<=0)?0:nflyer.size())/Math.log(2));
                                        or
  //logn = (nflyer.size()<=0)?0:31-Integer.numberOfLeadingZeros(nflyer.size());

  for(i=0; i < logn; i++)
     top[i] = heap.removeMin();

  return top;
}
```

Time Complexity: o(n)

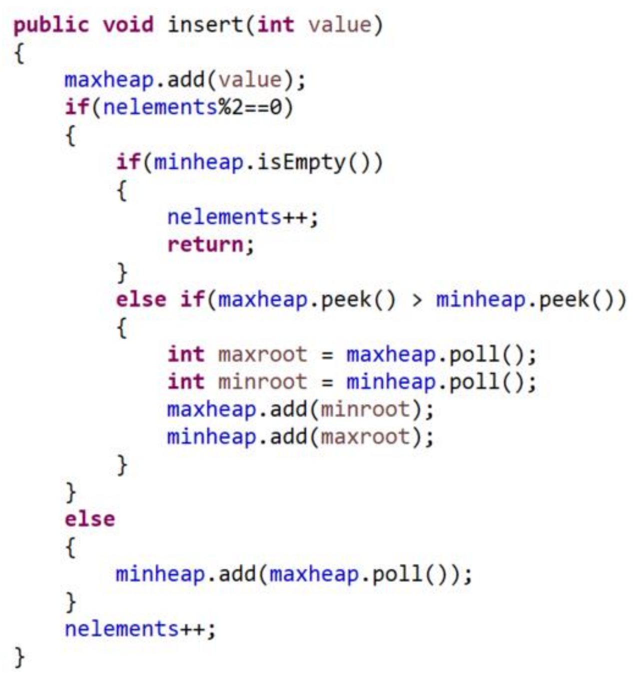
The time complexity of the total algorithm is o(n), where the time taken to construct a heap is 0(n) and the action of remove min takes o(logn). Therefore on an overall view it is o(n+$(\log n)^2$) = o(n) time.

Problem 4: Dynamic Median Finder (DMF)

For this question, we must design an DMF ADT that maintains a collection of comparable elements using two heaps as the only instance variables. Where insertion and remove median takes o(log n) time and get median takes o(1) time.

As mentioned in the question this can be achieved with the help of two heap design, which I take it to be minheap and maxheap respectively. As the name suggests each heap is used to store only half set of values i.e. the min heap is used to store all the values which is less than the value of the median and the max heap is used to store all the values which is greater than or equal to median. In case of even number of elements(n) then both heap contains exactly half of the number of elements(n/2), otherwise the max heap contains one element more than that of min heap. Hence the median will be at the root of the max heap so, at any given situation the median of the set of values can be obtained in o(1) time and in order to construct such min and max heaps it only takes o(log n), where the elements are spread out equally as mentioned above. The entire java implementation of it is attached with this assignment as: DynamicMedianFinder.java

Code snippet:



```
public void insert(int value)
{
    maxheap.add(value);
    if(nelements%2==0)
    {
        if(minheap.isEmpty())
        {
            nelements++;
            return;
        }
        else if(maxheap.peek() > minheap.peek())
        {
            int maxroot = maxheap.poll();
            int minroot = minheap.poll();
            maxheap.add(minroot);
            minheap.add(maxroot);
        }
    }
    else
    {
        minheap.add(maxheap.poll());
    }
    nelements++;
}
```

Example:

Input:

```
public static void main(String[] args)
 {
   DynamicMedianFinder dmf1 = new DynamicMedianFinder();
   System.out.println("DMF1 : ");
   dmf1.insert(17);
   dmf1.insert(-4);
   dmf1.insert(13);
   dmf1.insert(-7);
   dmf1.insert(12);
   dmf1.insert(15);
   dmf1.insert(5);
   dmf1.insert(2);
   System.out.println("Min Heap : "+dmf1.minheap);
   System.out.println("Max Heap : "+dmf1.maxheap);
   System.out.println("Median is: "+dmf1.getMedian());

}
```

Output:

```
DMF1 :
Min Heap : [12, 13, 15, 17]
Max Heap : [5, 2, -4, -7]
Median is: 5
```