Department of Electrical Engineering and Computer Science
Lassonde School of Engineering
York University, Winter 2017

EECS 2011: Assignment 2

Due Date - Monday, Mar 27, in class!!!

Question 1 **Property of Binary Trees** [15 points]

Assume a proper and complete binary tree (T) with n>1 nodes. Let **E(T)** represent the sum of the depths of all external nodes in T (see Figure1), and **I(T)** represent the sum of the depths of all internal nodes in T. Prove that:

(a) **E(T) = O(n·log(n))**
(b) **E(T) – I(T) = 2i**, where i represents the actual number of internal nodes in T.

(Hint: for part (b) use mathematical induction.)

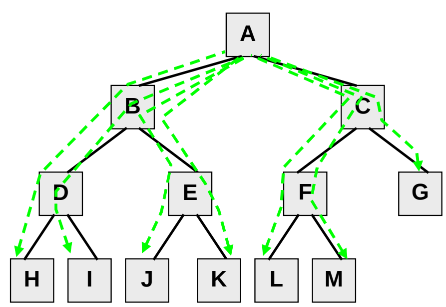


Figure 1 External Path Length

Solution:

Proof:

(a) Based on the properties of proper binary trees, discussed in class:

E <= e*h <= ((n+1)/2)*log(n) = O(n*log(n))

(b) Assume E-I=2i holds for some arbitrary i.

Now consider case i_{new}=i+1. We should prove: $E_{new} – I_{new} = 2*i_{new}=2*(i+1)$.

New tree with i_{new}1 internal nodes is created by removing an external node from the tree, and replacing it with an internal node with two attached external nodes.

Hence $e_{new} = (e-1) + 2 = e+1 = i + 2$.

Suppose the removed external node is at depth d. Accordingly:
$I_{new} = I + d$
$E_{new} = E – d + 2*(d+1)= E + d + 2$.

Subsequently:
$E_{new} – I_{new} = (E + d + 2) – (I + d) = (E – I) + 2 = (2i) + 2 = 2(i+1) = 2*i_{new}$

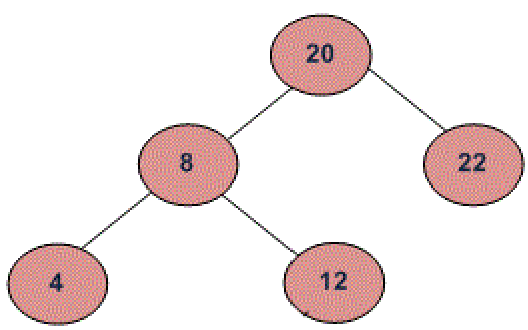
Question 2 Binary Search Tree [15 points]

Given a BST and two numbers – a and b (where a≤b) – propose an algorithm that prints all the elements k of the BST that satisfy: k≥a and k≤b. Your algorithm should return the found keys in ascending order. You can assume that there are no duplicate keys in the tree.
(You can also assume that a, b, and all the keys stored in the BST are integers.)

In addition to writing a brief description of the algorithm, you also need to implement the algorithm in Java (i.e., provide respective Java code), as well as justify the algorithm's running time.

Solution:

Suppose in this question, for the tree given below, if a = 10 and b = 22, your algorithm should print 12, 20 and 22.



One possible solution for this problem would be to do **inorder traversal** on the given tree (starting from the root). Continue on the left-subtree only if the current key > a, print the current node's key if the key is between a and b, continue on the right-subtree only in the current key < b..

The running time of this algorithm, in the worst case, would be O(n) in the worst case – which is the running time of the inorder traversal on the tree

```
public void  binaryInorderPrintKeys(Tree<E> T, Position<E> v, int a, int b) {
      E e = v.element();
      if  (T.hasLeft(v) && e > a)  binaryInorderPrintKeys(T, T.leftChild(v), a, b);
      if (e>= a && e <=b) { System.out.println(e+" "); }
      if  (T.hasRight(v) && e < b) binaryInorderPrintKeys(T, T.rightChild(v), a, b);
}
```