CSCE 2211 Exercises
Exercises (3): Trees , Binary Search Trees

Trees

Prove that for a full binary tree of height h:

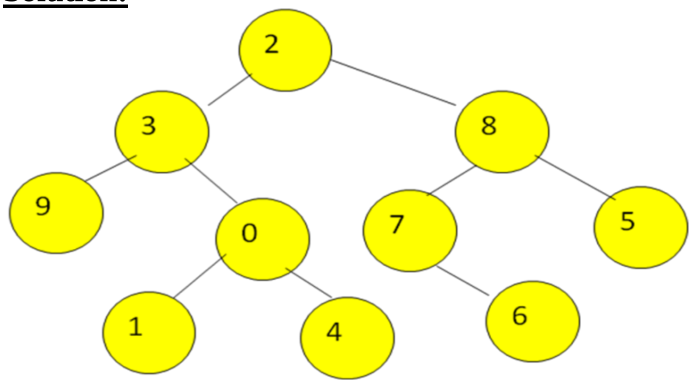
- The total number of nodes is $n = 2^h - 1$
- The total number of branches is $2^h - 2$
- The total number of leaves is 2^{h-1}
- The total number of internal nodes is $2^{h-1} - 1$
- The average search cost for a node is $O(\log n)$

Solution:

Proofs are given in the course slides.

Draw a binary tree with 10 nodes labeled 0, 1, . . . , 9 in such a way that the inorder and postorder traversals of the tree yield the following lists: 9, 3,1, 0, 4, 2, 7, 6, 8, 5 (inorder) and 9, 1, 4, 0, 3, 6, 7, 5, 8, 2 (postorder).

Solution:



(a) A full binary tree has a total of 32767 nodes.

- What is the height of the tree?
- What is the number of internal nodes in the tree?
- What is the number of leaves in the tree?

(b) In a football cup match, the defeated team goes out. If the initial number of teams is 128, how many matches will be played to get the final winner of the cup?

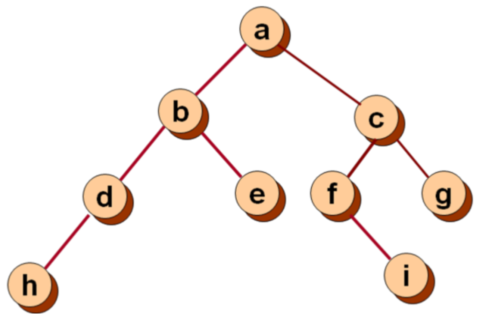
Solution:

(a) $h = \log(n+1) = \log(32768) = \log 2^{15} = 15$, ***No. of internal nodes*** $= 2^{h-1} - 1 = 2^{14} - 1$, ***No. of leaves*** $= 2^{14}$

(b) ***No. of teams = no. of leaves*** $= 128 = 2^7$, ***No. of matches = No. of internal nodes*** $= 127$

Traverse the shown tree in:

- Inorder Traversal
- Preorder Traversal
- Postorder Traversal



Left as an exercise

Assume binary trees in which the leaf nodes hold integer numbers and the non-leaf nodes hold the binary operations '+', '-', '*', and '/'. Provide an algorithm that, when given the root of a tree, evaluates the expression represented by the tree.

Left as an exercise

Binary Search Trees

In the following, assume that you are implementing member functions for the BST class

---

Write a function INTERNAL to return the number of internal nodes in a BST.

Solution:

```
template <class keyType, class dataType>
int binaryTree<keyType, dataType>::INTERNAL()
{       return INTERNAL2(root); }
template <class keyType, class dataType>
int binaryTree<keyType, dataType>::INTERNAL2(NodePointer aRoot)
{
        if(aRoot != NULL)
        {
            if(aRoot->left != NULL || aRoot->right != NULL)
          return 1 + INTERNAL2(aRoot->left) + INTERNAL2(aRoot->right);
            else return  0;
        }
     else
           return 0;
}
```

Write a function LEAVES to return the number of leaves in a BST.

Write a recursive function that returns the total number of nodes in a BST.

Solution:

```
template <class keyType, class dataType>
int binaryTree<keyType, dataType>::numberNodesRec()
{
    return numberNodesRec2(root);
} // end of public
//____________ Private__________________
template <class keyType, class dataType>
int binaryTree<keyType, dataType>:: numberNodesRec2(NodePointer aRoot)
{
    if (aRoot != NULL)
        return 1 + numberNodesRec2(aRoot->left) + numberNodesRec2(aRoot->right);
    return 0;
}
```

Write a recursive function to return the height H of a BST.

Hint:

For an empty tree, H(t) = 0

For a non-empty tree, H(t) = 1 (for the root) + max { H(left subtree) , H(right subtree) }

Write a function MININTREE to return the minimum key value in a BST.

Hint:

See algorithm in the course slides.

Write a function that receives a BST of root (T) and two keys (K1) and (K2) with $K1 \leq K2$ and prints all keys in the tree satisfying the condition $K1 \leq key \leq K2$.

Hint:

Modify the In-Order traversal function

Write a function MAXINTREE to return the largest key value in a BST.

Write a function MED to receive a BST of integer keys and the number of nodes in the tree (N) and to return the median key value.

(Consider N to be an odd number. The median value is then the key value below which and above which there is an equal number of keys).

Hint:

Modify the In-Order traversal function

Code a recursive function BACKTRAVERSE to display the keys in the nodes of a BST in descending order.

Hint:

Modify the In-Order traversal function

Code a recursive function COPYTREE to receive a pointer (t) to a binary tree and to return a pointer to an exact copy of the tree.

Solution:

Code the following algorithm to make it a member function of the BST class.

```
treeNode *CopyTree ( treeNode *t  )
{       treeNode *p;
        if  (t)
                {           p = new treeNode ;
                            p->left = CopyTree(t->left) ;
                            p->right = CopyTree(t->right);
                            p->info = t->info ;    return p ;  };
        else  return NULL ;
}
```