Fundamental Algorithms, Binary Search Tree, HW-6

Tulsi Jain

1. Non-Recursive Inorder Walk

1.1 With Stack

Algorithm 1 Inorder Walk

```
function InorderWalk(root)                          ▷ where root RootNode
    current = root;
    Stack s;
    while current != null && s.size() > 0 do
        s.push(root.key);
        current = current.left;
        if current == null then
            node n = s.pop();
            print n.val
            current = n.right;
        end if
    end while
end function
```

1.2 Single Threaded

Assumption: Tree is single threaded with right child of leaf node pointing to in-order successor

Algorithm 2 Inorder Walk

```
Class Node (
    key
    leftChild, rightChild
    isRightThreaded
)

function InorderWalk(root)                          ▷ where root RootNode
    current = Minimum(root);
    while current != null do
        print current.key;
        if current.isRightThreaded == TRUE then
            temp = current.rightChild;
            current.isRightThreaded = FALSE;
            current.rightChild = NULL;
            current = temp;
        else
            current = Minimum(current.rightChild);
        end if
    end while
end function
function Minimum(root)                              ▷ where root Subtree root node
    min = root;
    while min.left != null do
        min = min.left;
    end while
    return min;
end function
```

2. Tree Insert

Algorithm 3 TREE-INSERT

```
function TREE-INSERT(root, z)                    ▷ where root RootNode, z node
    if root == null then
        root = z;
        z.p = null;
        return
    end if
    if root.key <= z.key then
        if root.right != null then
            return TREE-INSERT( root.right, z);
        else
            root.right = z;
            z.p = root;
        end if
    else
        if root.left != null then
            return TREE-INSERT( root.left, z);
        else
            root.left = z;
            z.p = root;
        end if
    end if
end function
```

3. Randomly Built Binary Search

3.1 a

Since, On left hand side, we are adding depth for each and dividing by total numbers of nodes. Both side represents the average depth as per definition.

3.2

Path length of T_l is P(L) and path length of T_r is P(R). Path length of its parent would be P(L) + P(R) + n - 1. Because now every non-root node has to travel one step more and there are n - 1 non root node.

3.3

Given tree is build randomly so there are n - 1 more possible location where a root can be put. Nodes in T_L and T_R are equally likely to be present in remaining n - 1 location. $P(n)$ as per definition represents the average depth of a tree. Hence average depth of a tree would be as follows,

$$P(n) = \frac{1}{n}\sum_{i=0}^{n-1}(P(i) + P(n-i-1) + n - 1)$$

3.4

From part three

$$P(n) = \frac{1}{n}\sum_{i=0}^{n-1}(P(i) + P(n-i-1) + n - 1)$$

$$P(n) = \frac{1}{n}\sum_{i=0}^{n-1}(P(i) + \frac{1}{n}\sum_{i=0}^{n-1}P(n-i-1) + \frac{1}{n}\sum_{i=0}^{n-1}n - 1$$

$$P(n) = \frac{1}{n}\sum_{i=0}^{n-1}(P(i) + \frac{1}{n}\sum_{i=0}^{n-1}P(n-i-1) + n(n-1)$$

Put $n - i - 1$ as j

$$P(n) = \frac{1}{n}\sum_{i=0}^{n-1}(P(i) + \frac{1}{n}\sum_{j=n-1}^{0}P(j) + \Theta(n)$$

Both summation are same but in written is opposite order. Hence can be written as follows.

$$P(n) = \frac{2}{n}\sum_{i=0}^{n-1}(P(i) + \Theta(n)$$

P(0) would be zero

$$P(n) = \frac{2}{n}\sum_{i=1}^{n-1}(P(i) + \Theta(n)$$

3.5

Prove this by induction
Inductive hypothesis $P(q) = O(q\log(q))$,

for boundary case $n = 2$; $a2log2 + b+ > P(2)$ it holds as we can choose a and b.
Let's consider it hold for all q less than n and prove for n.

$$P(n) <= \frac{2}{n}\sum_{i=1}^{n-1} q\log(q) + \Theta(n)$$

P(1) would be zero

$$P(n) <= \frac{2}{n}\sum_{i=2}^{n-1} q\log(q) + \Theta(n)$$

By CLRS 7.7, $n^2\log(n) - \frac{1}{8}n^2 >= \sum_{q=2}^{n-1} q\log(q)$

$$P(n) <= \frac{2}{n}(\frac{1}{2}n^2\log(n) - \frac{1}{8}n^2) + \Theta(n)$$

$$P(n) <= n\log(n) - \frac{1}{4}n + \Theta(n)$$

$$P(n) <= n\log(n) + \Theta(n) - \frac{1}{4}n$$

$$P(n) <= n\log(n) + c_1 n - \frac{1}{4}n$$

For this to be hold true, $c_1 >= .25$

3.6

Like, In insertion for BST we first compare every element with root, in quick sort every element is getting compared with pivot. We pick the root of a tree as a pivot. As this is Binary Search Tree all nodes greater and less than root would be on right and left sub-tree respectively. Then for left sub-tree pivot is root of left sub-tree and for right sub-tree pivot is root of right sub-tree. Two subtrees are like two partition arry. We do this recursively for each of the sub-tree until en-counter a leaf and BST property is valid for all its. Child sub-trees would always be compared

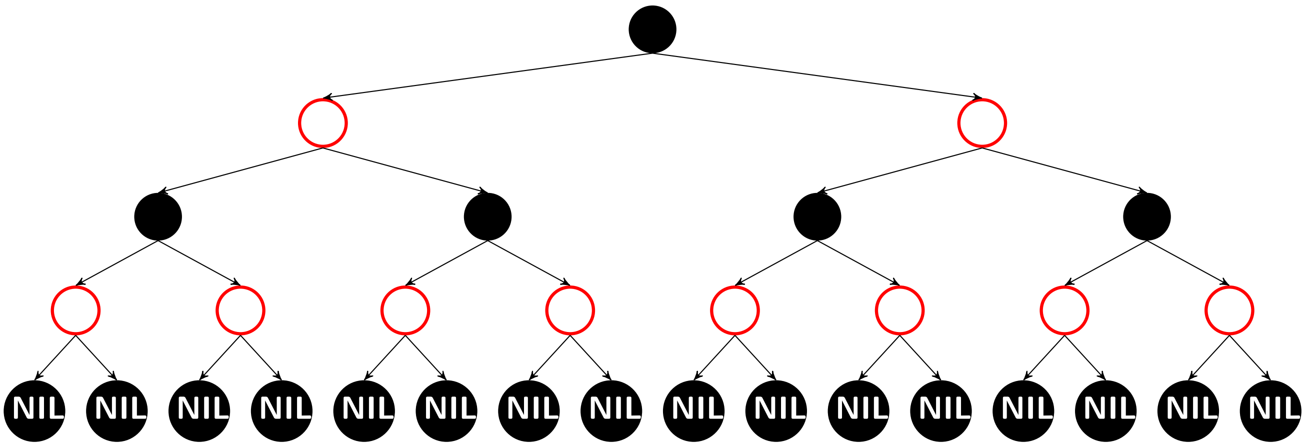
4. Fourth Problem

As per properties of Red-Black tree, children of a red node can not be a red node and any path from a node to its descendents must encountered same number of black node.
Let's say node, x has a black height, b. It is longest possible path would be of length 2b as no two red node can be in immediate parent child relation, when in walk red and black nodes encountered alternatively. And shortest possible path would be of length b, when in walk all encountered nodes are black.
Longest path <= 2*(Shortest path)

5. Largest/Smallest Possible Number

5.1 Smallest

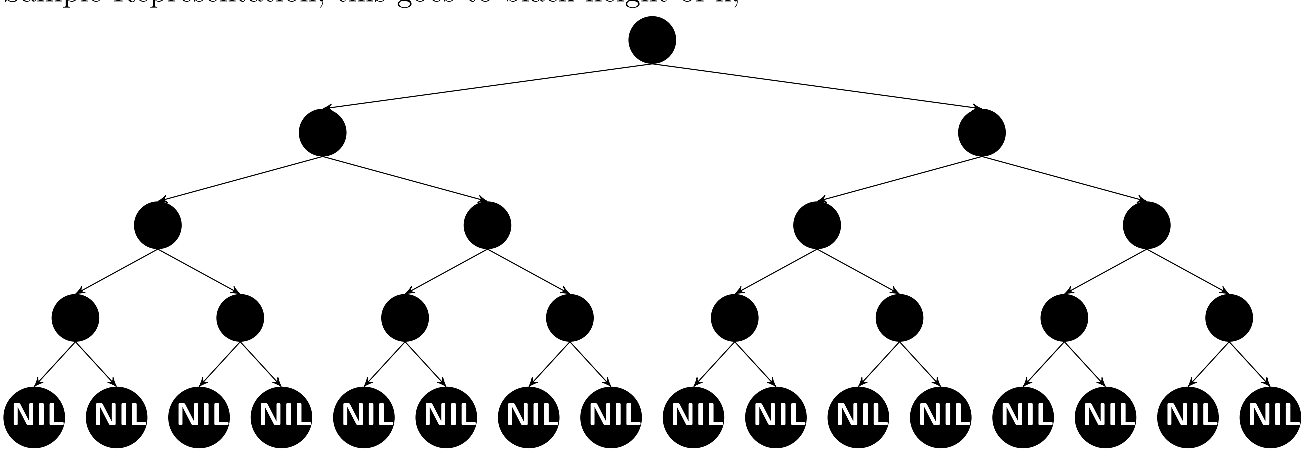
Minimum number of possible internal nodes is $2^k - 1$, This would occur when all of its descendent are black. This is summation of following:
Sample Representation, this goes to black height of k,



$2^0(Black) + 2^1(Black) + 2^2(Black)....2^{k-1}(Black) = 2^k - 1$

5.2 Largest

Maximum number of possible internal nodes is $2^{2k} - 1$. This would occur when root node is black. And its immediate children are red, followed by black and red children until b black nodes encountered. This is summation of following:
Sample Representation, this goes to black height of k,

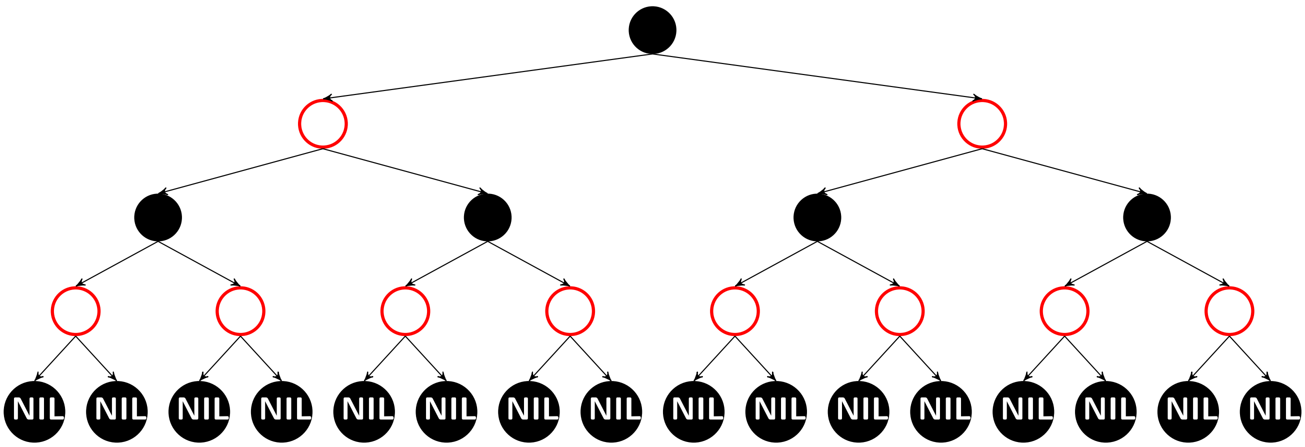


$2^0(Black) + 2^1(Red) + 2^2(Black).........2^{2k-1}(Red) = 2^{2k} - 1$

6. Largest/Smallest Ratio

6.1 Largest

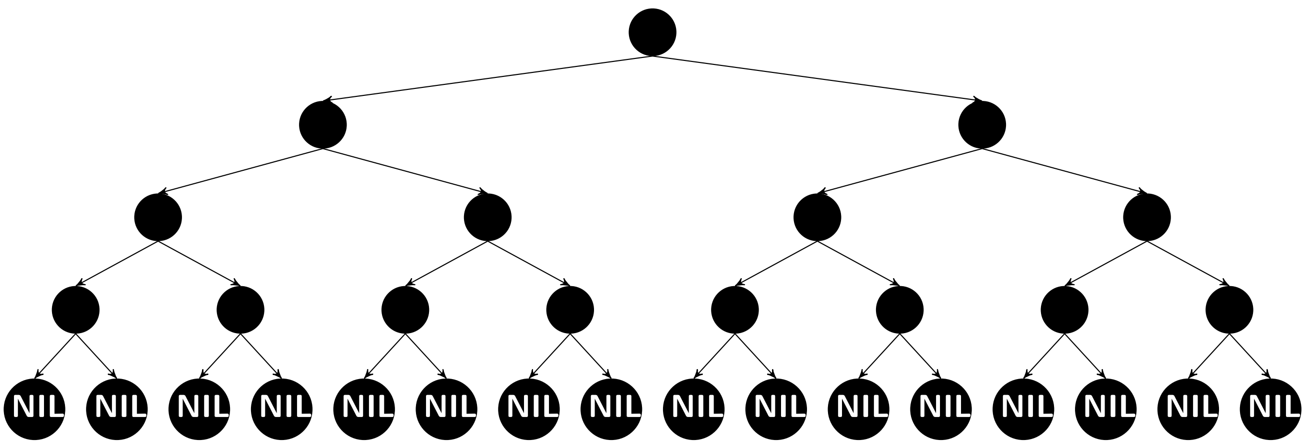
Example: Representation for largest ratio



Maximum ratio is 2, when root node is black and all of its immediate children are red. And children of red are black an so on. Nodes at same depth has are same color and parent as well children are of opposite color.

6.2 Smallest

Example: Representation for Smallest ratio



Minimum ratio is 0, when number of red node is zero