DS Project Report

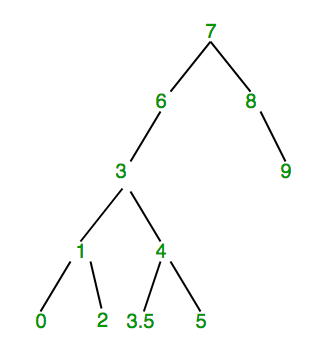
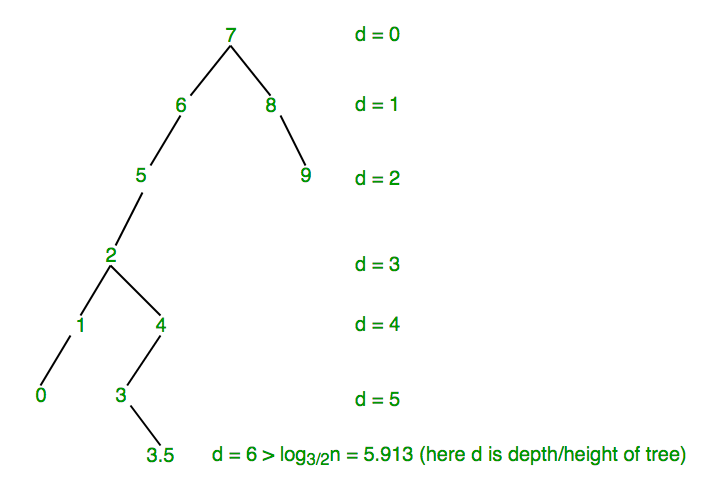
SEC A

# Group Members :

1. Shameer Ahmed (16K-4030)
2. Hasnain Haider (16K-4029)
3. Irfan Ali (16K-4017)

# Scapegoat Tree

A scapegoat tree is a self-balancing binary search tree, invented by Arne Andersson and again by Igal Galperin and Ronald L. Rivest. It provides worst-case O(log n) lookup time, and O(log n) insertion and deletion time.

When a new node is inserted using BST Insertion algorithm, if that node has depth greater than where n is number of nodes in the tree so we will find the scapegoat. Walk up from that node until we find a node that satisfy the equation 

# Why Scapegoat Tree?

Unlike other self-balancing BSTs, ScapeGoat tree doesn’t require extra space per node. For example, Red Black Tree nodes are required to have color. In below implementation of ScapeGoat Tree, we only have left, right and parent pointers in Node class. Use of parent is done for simplicity of implementation and can be avoided.

# Insertion:

void Insert(int Key)

{

SCGNode n= new SCGNode(Key);

int depth=InsertWithDepth(n);

if(depth>Log32(q)) //n can be used as well

{

//Depth exceeded, find scapegoat

SCGNode temp= n.Parent;

while(3\*GetSize(temp)<=2\*GetSize(temp.Parent))

{

temp=temp.Parent;

Rebuild(temp.Parent);

}

}

}

int InsertWithDepth(SCGNode N)

{

SCGNode Rt= Root;

if(Rt==null)

{

Root=N;

n++;

q++;

return 0;

}

boolean inserted=false;

int depth=0;

do

{

if(N.Value<Rt.Value)

{

if(Rt.Left==null)

{

Rt.Left=N;

N.Parent=Rt;

inserted=true;

}

else

{

Rt=Rt.Left;

}

}

else if(N.Value>Rt.Value)

{

if(Rt.Right==null)

{

Rt.Right=N;

N.Parent=Rt;

inserted=true;

}

else

{

Rt=Rt.Right;

}

}

else

{

return -1;

}

depth++;

}

while(!inserted);

n++;

q++;

return depth;

}