**TREE :**

since arrays and linked list are linear data structure that time require to search a linear lint is directly proportional to the size of the data set we would need a DS that helps in this easily retrieve the data

a data structure that store the data in hierarchical format for easy pic of data .

**BINARY TREE :**

Binary tree is hierarchical data structure each node has at most two children .

**Type :**

1.full binary tree : a binary tree in which every node has either two child or no child .

2.complete binary tree : a binary tree have all level are completely fill except last level which has to fill from left to right .

3.perfect binary tree : a binary tree in which all internal node have two children . and leave are in the same level .

4. balanced binary tree : a binary tree in which the height of the left and right sub tree of any node differ by at most one

Type :

1. AVl tree   
 2.red black tree

5.skewed tree or Regenerate : a binary tree in which every internal node has only one child .either left or right .

package day19;

class Node {

int data ;

Node left , right;

public Node(int data)

{

this.data=data;

left=right=null;

}

}

class BinaryTree{

Node root = null;

}

public class task1 {

public static void main(String args[])

{

BinaryTree t = new BinaryTree();

t.root = new Node(1);

t.root.left=new Node(2);

t.root.right=new Node(3);

t.root.left.left= new Node(4);

t.root.left.right=new Node(5); }}

Linked list representing .

BINARY TRAVERSAL :

class Node {

int data ;

Node left , right;

public Node(int data)

{

this.data=data;

left=right=null;

}

}

class BinaryTree{

Node root = null;

public void preOrder(Node root)

{

if(root==null)

{System.***out***.println("empty");return ;}

System.***out***.println(root.data);

preOrder(root.left);

preOrder(root.right);

}

public void postOrder(Node node)

{

if(node==null)

{System.***out***.println("empty"); return;}

postOrder(node.left);

postOrder(node.right);

System.***out***.println(node.data);

}

public void inOrder(Node node)

{

if(node==null)

{System.***out***.println("empty");return;}

inOrder(node.left);

System.***out***.println(node.data);

inOrder(node.right);

}

}

public class task1 {

public static void main(String args[])

{

BinaryTree t = new BinaryTree();

t.root = new Node(1);

t.root.left=new Node(2);

t.root.right=new Node(3);

t.root.left.left= new Node(4);

t.root.left.right=new Node(5);

System.***out***.println("Pre order");

t.preOrder(t.root);

System.***out***.println("Post order");

t.postOrder(t.root);

System.***out***.println("In order");

t.inOrder(t.root);

}

}

Pre order

1

2

4

empty

empty

5

empty

empty

3

empty

empty

Post order

empty

empty

4

empty

empty

5

2

empty

empty

3

1

In order

empty

4

empty

2

empty

5

empty

1

empty

3

empty

**Level order traversal :**

public void levelOrder()

{

if(root==null)

{return;}

Queue<Node> q = new LinkedList();

q.add(root);

while(!q.isEmpty())

{

Node temp = q.poll();

System.***out***.println(temp.data);

if(temp.left!=null)

{

q.add(temp.left);

}

if(temp.right!=null)

{

q.add(temp.right);

}

}

}

1 2 3 4 5

Binary tree insertion :

public void insertion(int data)

{

root=ef(root,data);

}

public Node ef(Node root , int data)

{

if(root==null)

{

root=new Node(data);

return root;

}

if(data<root.data)

{

root.left=ef(root.left,data);

}

else if(data>root.data)

{

root.right=ef(root.right,data);

}

return root;

}

**Width :**

Is the len of longest path b/w any two node .

This part may are may not pass to the node .

The len of the path 🡪 by no of edges b/w nodes

Leftheight + righthight + 1;

**Right view** of the binary tree can be obtain using BFS the idea is to travel to the last node at the each level .

**Right view and left view :**

public void rightView(Node1 root)

{

if(root==null)

{return;}

Queue<Node1> q = new LinkedList();

q.add(root);

while(!q.isEmpty())

{

int n = q.size();

for(int i =1;i<=n;i++)

{

Node1 temp = q.poll();

if(i==n)

{System.***out***.println(temp.data);}

if(temp.left!=null)

{

q.add(temp.left);

}

if(temp.right!=null)

{

q.add(temp.right);

}

}

}

}

public void leftView(Node1 root)

{

if(root==null)

{return;}

Queue<Node1> q = new LinkedList();

q.add(root);

while(!q.isEmpty())

{

int n = q.size();

for(int i =1;i<=n;i++)

{

Node1 temp = q.poll();

if(i==n)

{System.***out***.println(temp.data);}

if(temp.left!=null)

{

q.add(temp.left);

}

if(temp.right!=null)

{

q.add(temp.right);

}

}

}

}

System.***out***.println("right view");

t.rightView(t.root);

System.***out***.println("left view");

t.leftView(t.root);

o/p :

right view

1

3

6

left view

1

3

6

Height and calculate depth :

public int height\_depth(Node1 node , int arr[])

{

if(node==null)

{return 0;}

int leftH = height\_depth(node.left,arr);

int rightH = height\_depth(node.right,arr);

arr[0]=Math.*max*(arr[0], leftH+rightH+1);

return 1+Math.*max*(leftH, rightH);

}

HEIGHT & DEPTH

3

System.***out***.println("HEIGHT & DEPTH");

System.***out***.println(t.height\_depth(t.root,arr));

Height :

public int height(Node1 node)

{

if(node==null)

{return 0;}

int leftH = height(node.left);

int rightH = height(node.right);

return 1+Math.*max*(leftH, rightH);

}

System.***out***.println("height");

System.***out***.println(t.height(t.root));

height

3

**SERIALIZED METHOD :**

public String serialize(Node1 root)

{

if(root==null)

{return "empty";}

Queue<Node1> q = new LinkedList();

q.add(root);

StringBuilder sb = new StringBuilder();

while(!q.isEmpty())

{

Node1 temp = q.poll();

if(temp==null)

{

sb.append("null , ");

continue;

}

sb.append(temp.data).append(" , ");

q.add(temp.left);

q.add(temp.right);

}

return sb.toString();

}

1 , 2 , 3 , 4 , 5 , null , 6 , null , null , null , null , null , null ,