**TREE TYPE :**

1.BST : binary tree that contains the left sub trees are lesser than the root and right sub tree is greater than the root .

2 . balanced BST : a BST where the height is min to ensure the efficient operation .

1.AVL tree : self balance it self to rotate the node .

2. red black tree : balanced it self balances the color

3.splate : most frequently access node

4.Binary heap : a complete binary that follow the heap properties .

Max : parent is great then child

Min : parent <= child

5.prefix tree (or) tries tree : storing the string efficient use auto complementation and dictionary implementation .

6.B Tree : a self balanced tree used in database and file system each node have multiple keys and children .

7.B+ tree : a variant of b tree but only store leaf node used in database .

8.fenwic tree (or) binary index tree: dsa for handling range query and update easily .

9 . suffix tree : a tree rep the all suffix of the given string using pattern match .

package day20;

class Node {

int data;

Node left, right;

public Node(int data) {

this.data = data;

left = right = null;

}

}

class Tree {

Node root = null;

public void insert(int d) {

root = insertRecursive(root, d);

}

private Node insertRecursive(Node root, int d) {

if (root == null) {

return new Node(d);

}

if (d < root.data) {

root.left = insertRecursive(root.left, d);

} else {

root.right = insertRecursive(root.right, d);

}

return root;

}

public void preOrder(Node node) {

if (node == null) {

return;

}

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

preOrder(node.left);

preOrder(node.right);

}

public boolean search(int data) {

return searchRecursive(root, data);

}

public boolean searchRecursive(Node root, int data) {

if (root == null) {

return false;

}

if (data == root.data) {

return true;

}

return data < root.data ? searchRecursive(root.left, data) : searchRecursive(root.right, data);

}

public int minVal(Node root)

{

while(root.left!=null)

{

root = root.left;

}

return root.data;

}

public int maxVal(Node root)

{

while(root.right!=null)

{

root = root.right;

}

return root.data;

}

public boolean isValid(Node root,Integer min , Integer max)

{

if(root==null)

{return true;}

if((min!=null && root.data<=min) || (max!=null && root.data>=max))

{return false;

}

return isValid(root.left,min,root.data) && isValid(root.right,root.data,max);

}

public int KthSmall(Node root,int k)

{

int[] count= {0};

int[] result= {-1};

kSmallEle(root,k,count,result);

return result[0];

}

public void kSmallEle(Node root,int k,int[] count , int[] result)

{

if(root==null || result[0]!=-1)

{return;}

kSmallEle(root.left,k,count,result);

count[0]++;

if(count[0]==k)

{

result[0]=root.data;

}

kSmallEle(root.right,k,count,result);

}

public Node delete(Node root, int key) {

if (root == null) {

return null;

}

if (key < root.data) {

root.left = delete(root.left, key);

} else if (key > root.data) {

root.right = delete(root.right, key);

} else {

if (root.left == null) {

return root.right;

} else if (root.right == null) {

return root.left;

}

root.data = minVal(root.right);

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

}

return root;

}

}

public class task1 {

public static void main(String args[]) {

Tree t = new Tree();

t.insert(15);

t.insert(16);

t.insert(14);

t.insert(1);

t.insert(4);

t.insert(17);

t.insert(20);

System.***out***.println("PreOrder Traversal:");

t.preOrder(t.root);

System.***out***.println("\nSearch Results:");

System.***out***.println("Search 17: " + t.search(17));

System.***out***.println("Search 10: " + t.search(10));

System.***out***.println("min val "+t.minVal(t.root));

System.***out***.println("max val "+t.maxVal(t.root));

System.***out***.println(t.isValid(t.root, null, null));

System.***out***.println(t.KthSmall(t.root, 3));

t.delete(t.root, 15);

System.***out***.println("after delete 15");

t.preOrder(t.root);

}

}

PreOrder Traversal:

15

14

1

4

16

17

20

Search Results:

Search 17: true

Search 10: false

min val 1

max val 20

true

14

after delete 15

16

14

1

4

17

20

--------------------------------------------XXXXX-----------------------------------------------------------------------

package day20;

class Node1 {

int data,height;

Node1 left, right;

public Node1(int data) {

this.data = data;

left = right = null;

height=1;

}

}

class AVL

{

Node1 root=null;

public Node1 insert(Node1 node , int data)

{

if(node==null)

{

node = new Node1(data);

}

if(data<node.data)

{

node.left=insert(node.left,data);

}

else if(data>node.data)

{

node.right=insert(node.right,data);

}

else {

return node;

}

node.height=Math.*max*(height(node.left),height(node.right))+1;

int bal=getBalance(node);

if(bal>1 && data<node.left.data)

{

return rightRotate(node);

}

if(bal<-1 && data>node.right.data)

{

return leftRotate(node);

}

if(bal>1 && data>node.left.data)

{

node.left=leftRotate(node.left);

return rightRotate(node);

}

if(bal<-1 && data<node.right.data)

{

node.right=rightRotate(node.right);

return leftRotate(node);

}

return node;

}

public int getBalance(Node1 node)

{

return node==null ?0: height(node.left)-height(node.right);

}

public int height(Node1 node)

{

return node==null?0:node.height;

}

public Node1 rightRotate(Node1 y )

{

Node1 x, t2 ;

x = y.left;

t2=x.right;

x.right=y;

y.left=t2;

y.height=(Math.*max*(height(y.left), height(y.right)))+1;

x.height=(Math.*max*(height(x.left), height(x.right)))+1;

return x;

}

public Node1 leftrotate(Node1 x)

{

Node1 y, t2 ;

y= x.left;

t2=y.right;

x.right=t2;

x.height=(Math.*max*(height(x.left), height(x.right)))+1;

y.height=(Math.*max*(height(y.left), height(y.right)))+1;

return y;

}

public Node1 min(Node1 node)

{

while(node.left!=null)

{

node = node.left;

return node;

}

}

}

public class task2 {

public static void main(String args[]) {

AVL t = new AVL();

t.insert(t.root,30);

//t.delete(t.root, 12);

}

}

Greedy approach :

Digestra algorithm

Well man

Floyds algorithm

Warshell algorithm

MST – min spanning tree .

**Topological algorithm**

Hanhs algorithm

Kosarajus algorithm

**GRAPH :**

In DS are used to relationship b/w object each graph is connected of a set of node is called vertices .

Connects by a line is called degree .

Graph is collection of edges and nodes .

**Type :**

Undirected graph . -> edges with no direction .

Directed graph (or) digraph -> edges have direction

Weighted graph -> each edge has a cost

Unweighted graph -> all edges are equal .

Degree : the total no of edges obtain in vertices in a graph is degree.

Outdegree : In digraph the total no of outcoming edges .

Indegree : the total no of incoming edges .

Cycle : start and end with same vertices

Simple Path : path with unique direction .

Strongly connected Graph : each pair of vertex x and y the graph is strongly connected it contains directed path from x to y and directed path from y to x .

Spanning Tree : spanning sub graph the is also a tree is known as spanning tree .

Forest : is graph without a cycle .

🡪Tree is a connected forest .

ADJENCENCY MATRIX OF THE GRAPH :

package day20;

import java.util.\*;

public class task3 {

public static void adjMatrix(int[][] arr, int row) {

System.***out***.println("\nAdjacency Matrix:");

for (int i = 1; i < row; i++) {

for (int j = 1; j < row; j++) {

System.***out***.print(arr[i][j] + " ");

}

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

}

}

public static void main(String args[]) {

Scanner s = new Scanner(System.***in***);

System.***out***.print("Number of nodes: ");

int row = s.nextInt();

System.***out***.print("Number of edges: ");

int edges = s.nextInt();

System.***out***.print("Is the graph directed (yes/no): ");

String dir = s.next();

int arr[][] = new int[row + 1][row + 1];

for (int i = 1; i <= edges; i++) {

System.***out***.println("Enter source, destination, and weight:");

int sn = s.nextInt();

int dn = s.nextInt();

int weight = s.nextInt();

if (dir.equals("yes")) {

arr[sn][dn] = weight;

} else {

arr[sn][dn] = weight;

arr[dn][sn] = weight;

}

}

*adjMatrix*(arr, row + 1);

}

}

Number of nodes:

4

Number of edges: 4

Is the graph directed (yes/no): YES

Enter source, destination, and weight:

1

2

11

Enter source, destination, and weight:

2

3

33

Enter source, destination, and weight:

3

4

12

Enter source, destination, and weight:

4

1

13

Adjacency Matrix:

0 11 0 13

11 0 33 0

0 33 0 12

13 0 12 0