1. Trees

2. Binary Search Tree

3. Implement BST

4. AVL Trees

5. Graphs

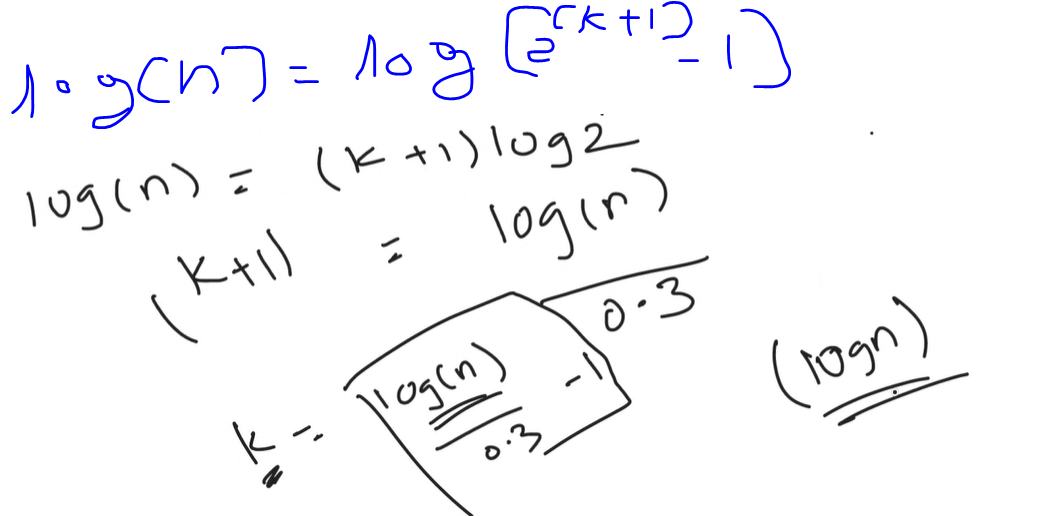
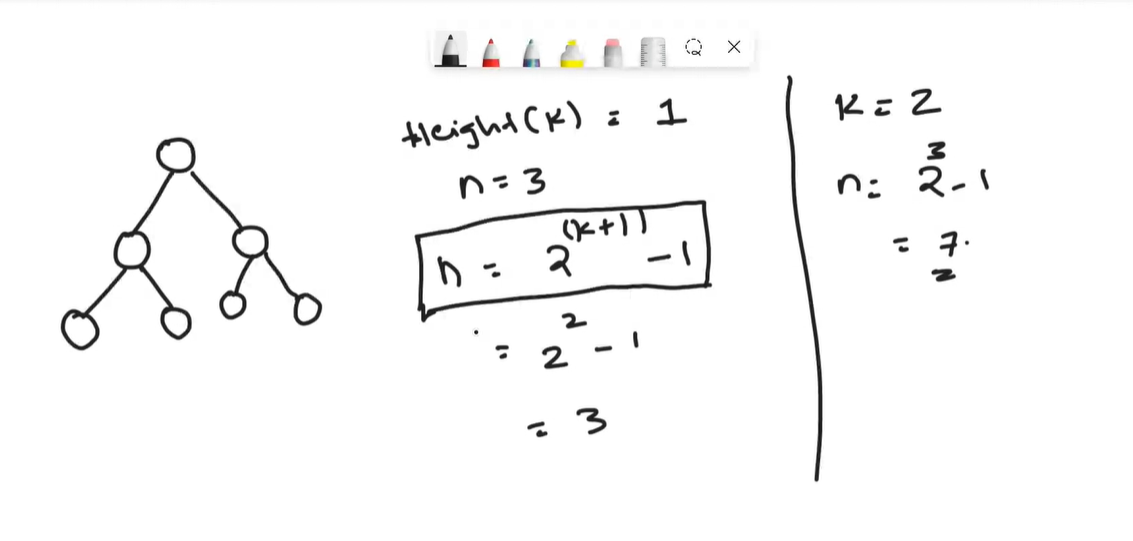
6. Graphs Traversal - BFS,DFS

7. Dynamic Programming

More on AVL Trees

<https://www.geeksforgeeks.org/insertion-in-an-avl-tree/>

https://www.geeksforgeeks.org/insertion-in-red-black-tree/



**BINARY SEARCH TREE IMPLEMENTATINO**

class BST {

    TreeNode root;

    class TreeNode {

        int data;

        TreeNode left, right;

    TreeNode (int data) {

            this.data = data;

        }

    }

    BST() {

        this.root = null;

    }

    void insert(int data) {

        root = insertRecursively(root, data);

    }

    TreeNode insertRecursively(TreeNode root, int data) {

        if (root == null) {

            return new TreeNode(data);

        }

        if (root.data > data) {

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

        } else {

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

        }

        return root;

    }

    void inOrder() {

        inOrderRecursive(root);

    }

    void inOrderRecursive(TreeNode root) {

        if (root != null) {

            inOrderRecursive(root.left);

            System.out.print(root.data + " ");

            inOrderRecursive(root.right);

        }

    }

    void preOrder() {

        preOrderRecursive(root);

    }

    void preOrderRecursive(TreeNode root) {

        if (root != null) {

            System.out.print(root.data + " ");

            preOrderRecursive(root.left);

            preOrderRecursive(root.right);

        }

    }

    void postOrder() {

        postOrderRecursive(root);

    }

    void postOrderRecursive(TreeNode root) {

        if (root != null) {

            postOrderRecursive(root.left);

            postOrderRecursive(root.right);

            System.out.print(root.data + " ");

        }

    }

    void search (int val) {

        searchRecursively(root, val);

    }

    void searchRecursively(TreeNode root, int val) {

        if (root == null) {

            System.out.println("Not found");

            return;

        }

        if (root.data == val) {

            System.out.println("Found");

            return;

        }

        if (root.data > val) {

            System.out.print("L" + " ");

            searchRecursively(root.left, val);

        } else {

            System.out.print("R" + " ");

            searchRecursively(root.right, val);

        }

    }

    void delete(int val) {

        deleteRecursively(root, val);

    }

    TreeNode deleteRecursively(TreeNode root, int val) {

        if (root == null) {

            return root;

        }

        if (val < root.data) {

            root.left = deleteRecursively(root.left, val);

        } else if (val > root.data) {

            root. right = deleteRecursively(root.right, val);

        } else {

            if (root.left == null) {

                return root.right;

            }

            if (root.right == null) {

                return root.left;

            }

            root.data = root.left.data;

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

        }

        return root;

    }

}

public class Main{

    public static void main(String[] args){

        BST bst = new BST();

        bst.insert(5);

        bst.insert(6);

        bst.insert(4);

        bst.insert(7);

        bst.insert(3);

        bst.insert(10);

        bst.inOrder();

        System.out.println();

        bst.preOrder();

        System.out.println();

        bst.postOrder();

        System.out.println();

        System.out.println("---------------");

        bst.delete(6);

        bst.inOrder();

        System.out.println();

        bst.preOrder();

        System.out.println();

        bst.postOrder();

        System.out.println();

        System.out.println("---------------");

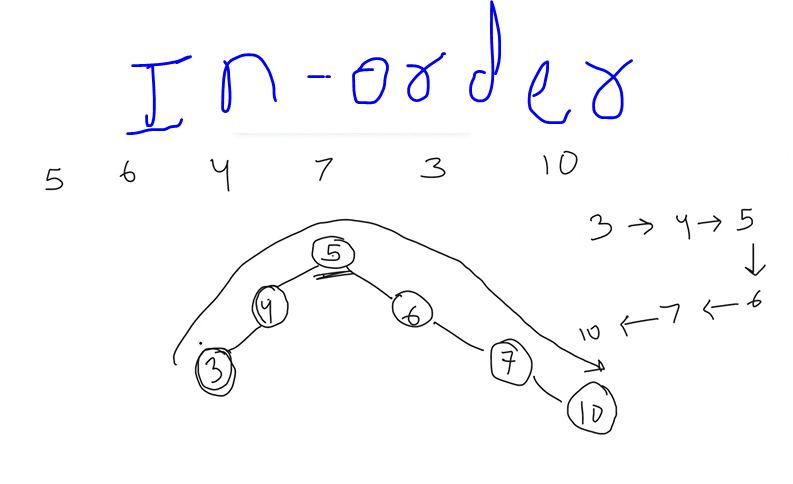
        bst.search(3);

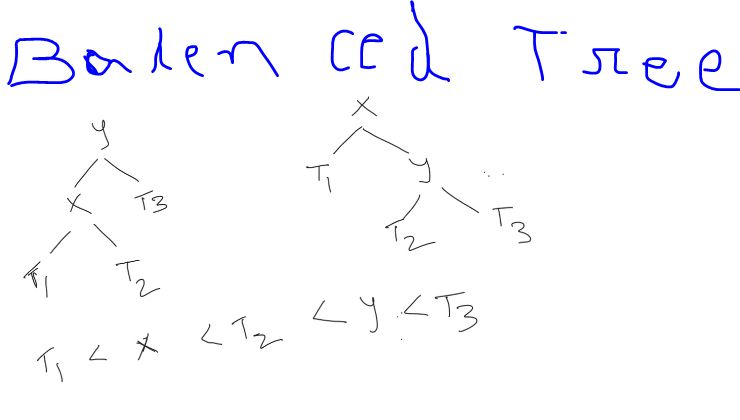
        bst.search(55);

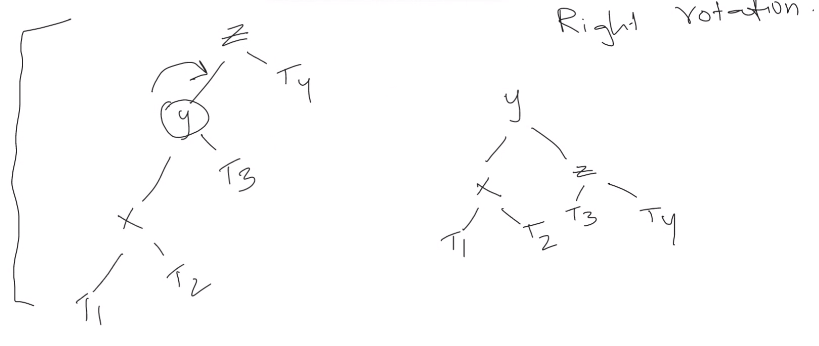
        bst.search(8);

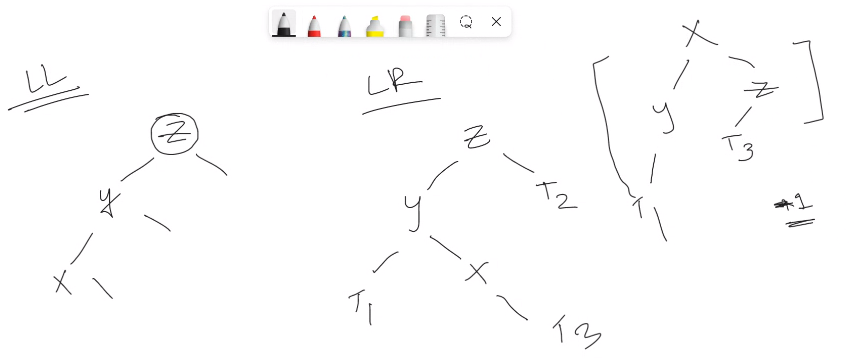
    }

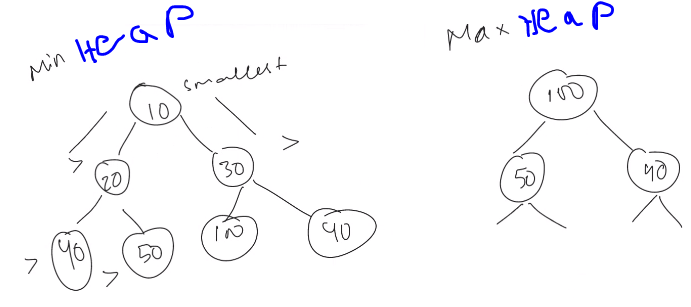
}

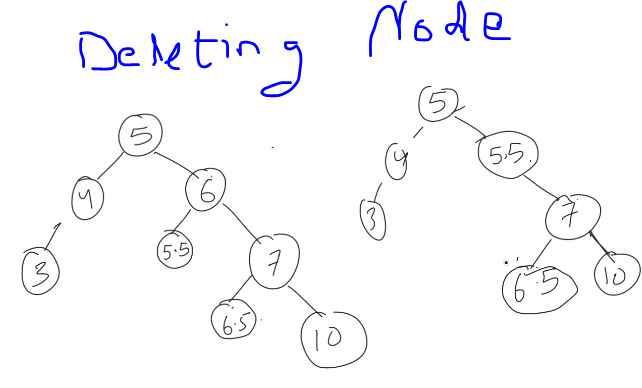


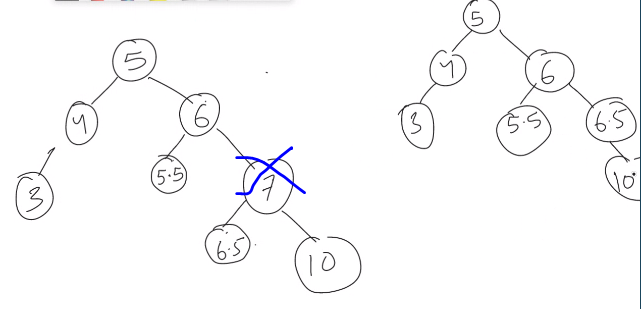


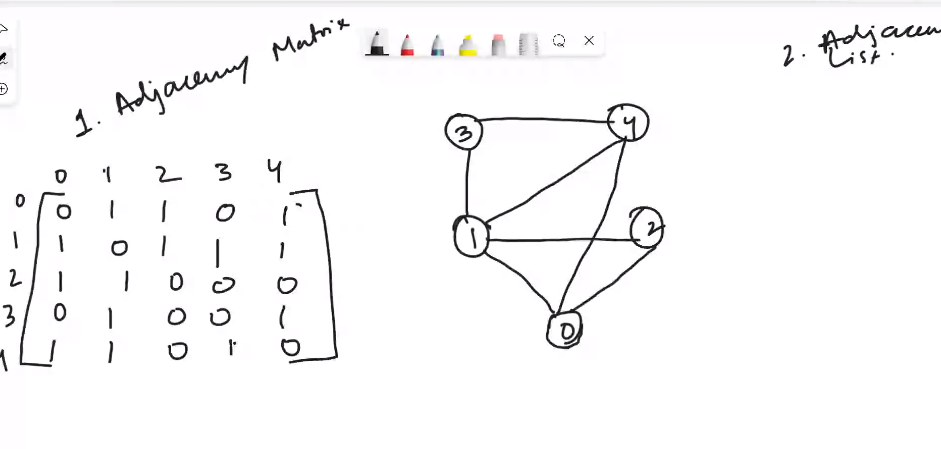
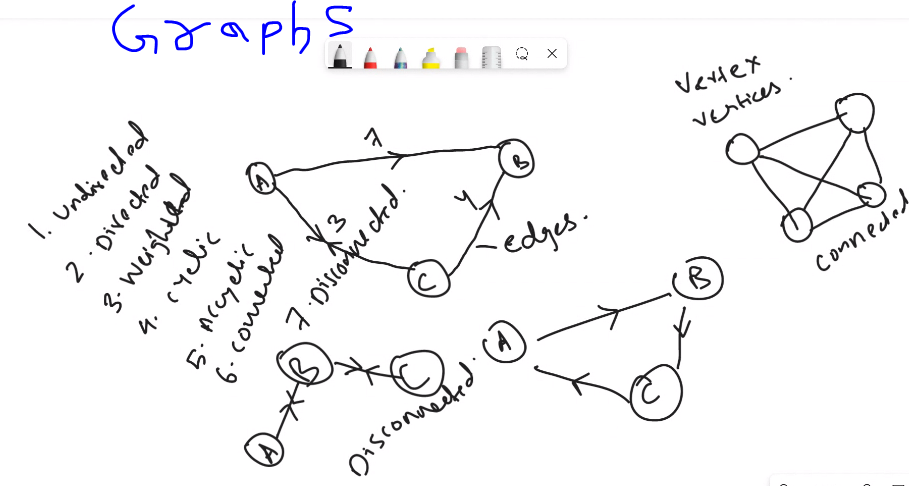
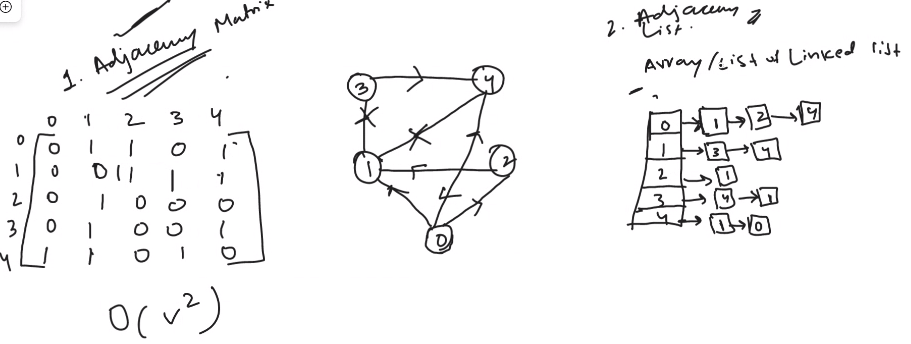










**GRAPH IMPLEMENTAION**

import java.util.LinkedList;

class AdjecentMatrix {

        int[][] edge;

        int vertices;

        AdjecentMatrix(int v) {

            vertices = v;

            edge = new int [v][v];

        }

        void addEdge(int v, int e) {

            edge[v][e] = 1;

            edge[e][v] = 1;

        }

         void removeEdge(int v, int e) {

            edge[v][e] = 0;

            edge[e][v] = 0;

        }

        void allEdges() {

            for (int i = 0; i<vertices; i++) {

                for (int j = 0; j<vertices; j++) {

                    if (edge[i][j] == 1) {

                        System.out.println(i + " - " + j);

                    }

                }

            }

        }

    }

class AdjecentList {

    LinkedList<Integer>[] edges;

    int vertices;

    AdjecentList(int v) {

        vertices = v;

        edges = new LinkedList[v];

        for (int i = 0; i < vertices; i++) {

            edges[i] = new LinkedList<>();

        }

    }

    void addEdge(int s, int d) {

        edges[s].add(d);

        edges[d].add(s);

    }

    void removeEdge(int s, int d) {

        edges[s].remove(d);

        edges[d].remove(s);

    }

    void allEdges() {

        for (int i = 0; i < vertices; i++) {

            for (int j = 0; j< edges[i].size(); j++) {

                System.out.println(i + " " + edges[i].get(j));

            }

        }

    }

}

public class Graph {

    public static void main(String[] args) {

        AdjecentMatrix m = new AdjecentMatrix(5);

        m.addEdge(0,1);

        m.addEdge(0,2);

        m.addEdge(0,3);

        m.addEdge(1,4);

        m.addEdge(1,3);

        m.addEdge(3,4);

        m.addEdge(3,2);

        m.allEdges();

        System.out.println("----------------------");

        AdjecentList l = new AdjecentList(5);

        l.addEdge(0,1);

        l.addEdge(0,2);

        l.addEdge(0,3);

        l.addEdge(1,4);

        l.addEdge(1,3);

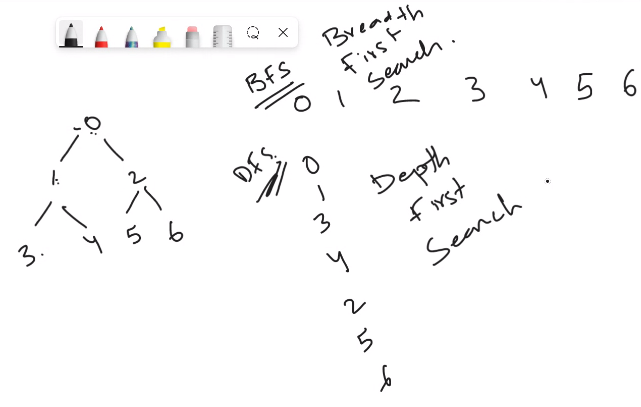
        l.addEdge(3,4);

        l.addEdge(3,2);

        l.allEdges();

    }

}



**BFS & DFS IMPLEMENTATION**

class Queue {

int[] arr;

int start = 0, rear = 0;

Queue(int n) {

arr = new int[n];

}

void enqueue(int n) {

arr[rear++] = n;

}

int getLength() {

return rear - start;

}

void viewQueue() {

for (int i = start; i < rear; i++) {

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

}

System.out.println();

}

int dequeue() {

int n = arr[start++];

return n;

}

boolean isEmpty() {

return (start == rear);

}

}

class Stack {

int[] arr;

int top=-1;

Stack(int n){

arr = new int[n];

}

void push(int n) {

arr[++top] = n;

}

int pop(){

return arr[top--];

}

boolean isEmpty(){

return (top==-1);

}

}

class Graph {

int[][] edges;

int vertices;

Queue q;

Stack s;

Graph(int n) {

vertices = n;

edges = new int[n][n];

q = new Queue(vertices);

s = new Stack(vertices);

}

void DFS(int start){

int curVertex;

boolean[] visited = new boolean[vertices];

visited[start]=true;

s.push(start);

while(!s.isEmpty()){

curVertex = s.pop();

System.out.println(curVertex);

for (int i = 0; i < vertices; i++) {

if (edges[curVertex][i] == 1 && (!visited[i])) {

visited[i] = true;

s.push(i);

}

}

}

}

void BFS(int start) {

boolean[] visited = new boolean[vertices];

q.enqueue(start);

visited[start] = true;

int curVertex;

while (!q.isEmpty()) {

curVertex = q.dequeue();

System.out.println(curVertex);

for (int i = 0; i < vertices; i++) {

if (edges[curVertex][i] == 1 && (!visited[i])) {

visited[i] = true;

q.enqueue(i);

}

}

}

}

void addEdge(int source, int destination) {

edges[source][destination] = 1;

// if undirected only

edges[destination][source] = 1;

}

void removeEdge(int source, int destination) {

edges[source][destination] = 0;

// if undirected only

edges[destination][source] = 0;

}

}

public class Main {

public static void main(String args[]) {

Graph g = new Graph(5);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(0, 3);

g.addEdge(1, 4);

g.addEdge(1, 3);

g.addEdge(3, 4);

g.addEdge(3, 2);

System.out.println("DFS");

g.DFS(0);

System.out.println("BFS");

g.BFS(0);

}

}

public class Main {

    static void binarySearch(int[] arr, int start, int end, int search) {

        int mid = start + (end-start)/2;

        if(arr[mid]==search){

            System.out.println(mid);

            return ;

        }

        else if(arr[mid]<search){

            binarySearch(arr, mid+1, end, search);

        }

        else {

            binarySearch(arr,start,mid-1,search);

        }

    }

    public static void main(String args[]){

        int[] arr = {10,20,30,40,50,60,70,80,90,100};

        binarySearch(arr,0,arr.length,90);

    }

}