**Q.1) Implementation of binary search tree : Create BST , Recursive traversals of BST ,**

**Search largest and smallest node, Count number of nodes.**

import java.util.Scanner;

class Node {

    public int data;

    public Node left;

    public Node right;

    Node(int data) {

        this.data = data;

        this.left = null;

        this.right = null;

    }

}

class BinarySearchTree {

    Scanner sc = new Scanner(System.in)

    public Node insert(Node root, int value) {

        if (root == null) {

            return new Node(value);

        }

        if (value < root.data) {

            root.left = insert(root.left, value);

        } else if (value > root.data) {

            root.right = insert(root.right, value);

        }

        return root;

    }

    public Node buildTree() {

        Node root = null;

        System.out.println("Enter values to insert into the BST (-1 to stop):");

        while (true) {

            int value = sc.nextInt();

            if (value == -1) {

                break;

            }

            root = insert(root, value);

        }

        return root;

    }

    public void inorder(Node root) {

        if (root == null) {

            return;

        }

        inorder(root.left);

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

        inorder(root.right);

    }

    public void preorder(Node root) {

        if (root == null) {

            return;

        }

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

        preorder(root.left);

        preorder(root.right);

    }

    public void postorder(Node root) {

        if (root == null) {

            return;

        }

        postorder(root.left);

        postorder(root.right);

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

    }

    public int findSmallest(Node root) {

        if (root == null) {

            System.out.println("Tree is empty");

            return -1;

        }

        Node current = root;

        while (current.left != null) {

            current = current.left;

        }

        return current.data;

    }

    public int findLargest(Node root) {

        if (root == null) {

            System.out.println("Tree is empty");

            return -1;

        }

        Node current = root;

        while (current.right != null) {

            current = current.right;

        }

        return current.data;

    }

    public int countNodes(Node root) {

        if (root == null) {

            return 0;

        }

        return 1 + countNodes(root.left) + countNodes(root.right);

    }

}

public class BST {

    public static void main(String[] args) {

        BinarySearchTree bst = new BinarySearchTree();

        Node root = bst.buildTree();

        System.out.print("Inorder : ");

        bst.inorder(root);

        System.out.print("\npostorder : ");

        bst.postorder(root);

        System.out.print("\npreorder : ");

        bst.preorder(root);

        System.out.println("\nLargest Node in BST : " + bst.findLargest(root));

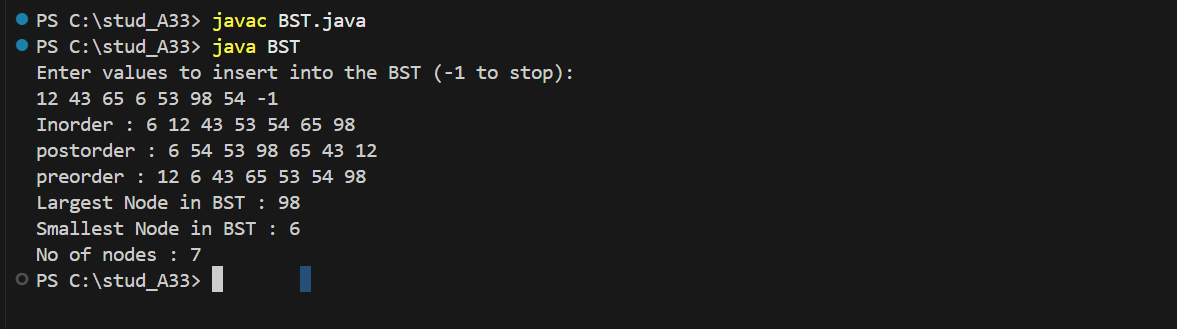
        System.out.println("Smallest Node in BST : " + bst.findSmallest(root));

        System.out.println("No of nodes : " + bst.countNodes(root));

    }

}

Output :



**Q.2) Implementation of Heap : MinHeap, MaxHeap (reHeapup, reHeapdown, delete).**

1. **MaxHeap**

class MaxHeap {

private int[] heap;

private int size;

private int capacity;

public MaxHeap(int capacity) {

this.capacity = capacity;

this.size = 0;

heap = new int[capacity];

}

private int parent(int index) {

return (index - 1) / 2;

}

private int leftChild(int index) {

return 2 \* index + 1;

}

private int rightChild(int index) {

return 2 \* index + 2;

}

public void insert(int value) {

if (size == capacity) {

throw new IllegalStateException("Heap is full");

}

heap[size] = value;

size++;

reheapUp(size - 1);

}

private void reheapUp(int index) {

while (index != 0 && heap[parent(index)] < heap[index]) {

swap(index, parent(index));

index = parent(index);

}

}

public int delete() {

if (size == 0) {

throw new IllegalStateException("Heap is empty");

}

int root = heap[0];

heap[0] = heap[size - 1];

size--;

reheapDown(0);

return root;

}

private void reheapDown(int index) {

int largest = index;

int left = leftChild(index);

int right = rightChild(index);

if (left < size && heap[left] > heap[largest]) {

largest = left;

}

if (right < size && heap[right] > heap[largest]) {

largest = right;

}

if (largest != index) {

swap(index, largest);

reheapDown(largest);

}

}

private void swap(int i, int j) {

int temp = heap[i];

heap[i] = heap[j];

heap[j] = temp;

}

public void printHeap() {

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

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

}

System.out.println();

}

}

public class maxHeap {

public static void main(String[] args) {

MaxHeap maxHeap = new MaxHeap(10);

maxHeap.insert(20);

maxHeap.insert(15);

maxHeap.insert(30);

maxHeap.insert(10);

maxHeap.insert(40);

System.out.println("Heap after insertions:");

maxHeap.printHeap();

System.out.println("Deleted element: " + maxHeap.delete());

System.out.println("Heap after deletion:");

maxHeap.printHeap();

}

}

Output :



1. **Min Heap**

class MinHeap {

private int[] heap;

private int size;

private int capacity;

public MinHeap(int capacity) {

this.capacity = capacity;

this.size = 0;

heap = new int[capacity];

}

private int parent(int index) {

return (index - 1) / 2;

}

private int leftChild(int index) {

return 2 \* index + 1;

}

private int rightChild(int index) {

return 2 \* index + 2;

}

public void insert(int value) {

if (size == capacity) {

throw new IllegalStateException("Heap is full");

}

heap[size] = value;

size++;

reheapUp(size - 1);

}

private void reheapUp(int index) {

while (index != 0 && heap[parent(index)] > heap[index]) {

swap(index, parent(index));

index = parent(index);

}

}

public int delete() {

if (size == 0) {

throw new IllegalStateException("Heap is empty");

}

int root = heap[0];

heap[0] = heap[size - 1];

size--;

reheapDown(0);

return root;

}

private void reheapDown(int index) {

int smallest = index;

int left = leftChild(index);

int right = rightChild(index);

if (left < size && heap[left] < heap[smallest]) {

smallest = left;

}

if (right < size && heap[right] < heap[smallest]) {

smallest = right;

}

if (smallest != index) {

swap(index, smallest);

reheapDown(smallest);

}

}

private void swap(int i, int j) {

int temp = heap[i];

heap[i] = heap[j];

heap[j] = temp;

}

public void printHeap() {

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

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

}

System.out.println();

}

}

public class minHeap1 {

public static void main(String[] args) {

MinHeap minHeap = new MinHeap(10);

minHeap.insert(20);

minHeap.insert(15);

minHeap.insert(30);

minHeap.insert(10);

minHeap.insert(40);

System.out.println("Heap after insertions:");

minHeap.printHeap();

System.out.println("Deleted element: " + minHeap.delete());

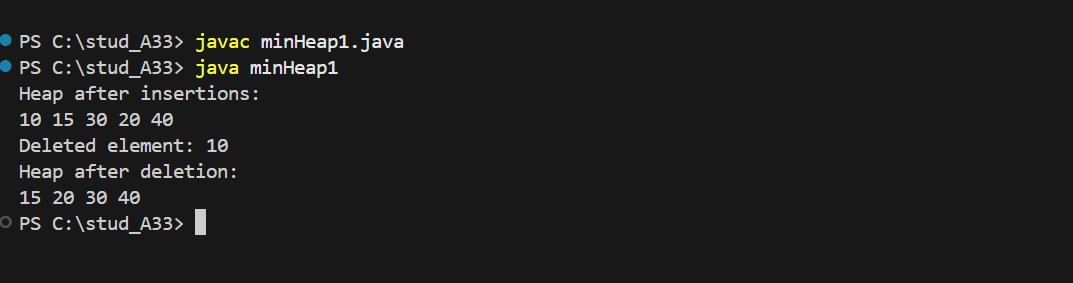
System.out.println("Heap after deletion:");

minHeap.printHeap();

}

}

Output :



**Q.3) Represent a graph using the Adjacency matrix.**

import java.util.Scanner;

class AdjacencyMatrix {

    private int[][] adjMatrix;

    private int n;

    public AdjacencyMatrix() {

        adjMatrix = new int[20][20];

        n = 0;

    }

    public void createGraph() {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter number of vertices: ");

        n = scanner.nextInt();

        int maxEdges = n \* (n - 1) / 2;

        System.out.println("Enter the edges (origin destination), or enter 0 0 to stop:");

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

            System.out.print("Edge " + i + ": ");

            int origin = scanner.nextInt();

            int destination = scanner.nextInt();

            if (origin == 0 && destination == 0) {

                break;

            }

            if (isInvalidVertex(origin) || isInvalidVertex(destination)) {

                System.out.println("Invalid vertices. Please try again.");

                i--;

            } else {

                adjMatrix[origin][destination] = 1;

                adjMatrix[destination][origin] = 1;

            }

        }

    }

    public void insertVertex() {

        if (n >= 20) {

            System.out.println("Maximum vertices reached. Cannot insert more.");

            return;

        }

        n++;

        System.out.println("Vertex added. Total vertices: " + n);

    }

    public void insertEdge(int origin, int destination) {

        if (isInvalidVertex(origin) || isInvalidVertex(destination)) {

            System.out.println("Invalid vertices. Edge not added.");

            return;

        }

        adjMatrix[origin][destination] = 1;

        adjMatrix[destination][origin] = 1;

        System.out.println("Edge added between " + origin + " and " + destination);

    }

    public void deleteVertex() {

        if (n <= 0) {

            System.out.println("No vertices to delete.");

            return;

        }

        for (int i = 0; i < n; i++) {

            adjMatrix[i][n - 1] = 0;

            adjMatrix[n - 1][i] = 0;

        }

        n--;

        System.out.println("Last vertex deleted. Total vertices: " + n);

    }

    public void deleteEdge(int origin, int destination) {

        if (isInvalidVertex(origin) || isInvalidVertex(destination)) {

            System.out.println("Invalid vertices. Edge not deleted.");

            return;

        }

        if (adjMatrix[origin][destination] == 0) {

            System.out.println("No edge exists between " + origin + " and " + destination);

        } else {

            adjMatrix[origin][destination] = 0;

            adjMatrix[destination][origin] = 0;

            System.out.println("Edge deleted between " + origin + " and " + destination);

        }

    }

    public void displayMatrix() {

        System.out.println("Adjacency Matrix:");

        for (int i = 0; i < n; i++) {

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

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

            }

            System.out.println();

        }

    }

    private boolean isInvalidVertex(int vertex) {

        return vertex < 1 || vertex > n;

    }

}

public class GraphEx {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        AdjacencyMatrix graph = new AdjacencyMatrix();

        char ch = 'y';

        while (ch == 'y' || ch == 'Y') {

            System.out.println("1. Create Graph");

            System.out.println("2. Insert Vertex");

            System.out.println("3. Insert Edge");

            System.out.println("4. Delete Vertex");

            System.out.println("5. Delete Edge");

            System.out.println("6. Display Matrix");

            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();

            switch (choice) {

                case 1:

                    graph.createGraph();

                    break;

                case 2:

                    graph.insertVertex();

                    break;

                case 3:

                    System.out.print("Enter origin and destination: ");

                    int origin = scanner.nextInt();

                    int destination = scanner.nextInt();

                    graph.insertEdge(origin, destination);

                    break;

                case 4:

                    graph.deleteVertex();

                    break;

                case 5:

                    System.out.print("Enter origin and destination: ");

                    origin = scanner.nextInt();

                    destination = scanner.nextInt();

                    graph.deleteEdge(origin, destination);

                    break;

                case 6:

                    graph.displayMatrix();

                    break;

                default:

                    System.out.println("Invalid choice. Please try again.");

            }

            System.out.print("Do you want to continue? (y/n): ");

            ch = scanner.next().charAt(0);

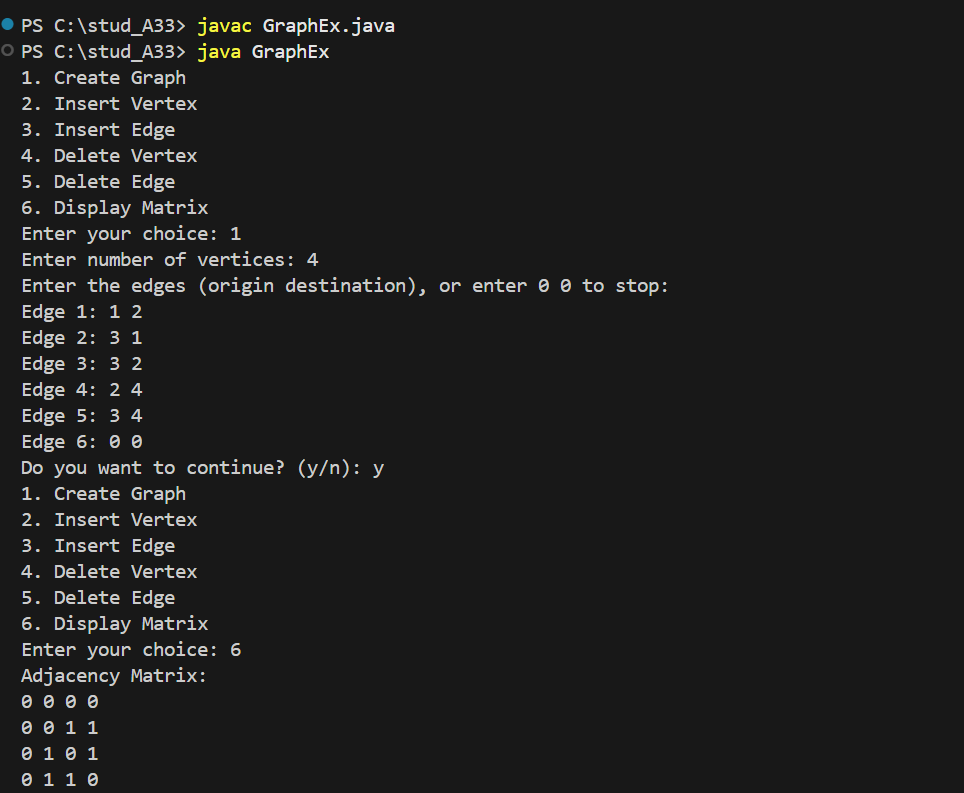
        }

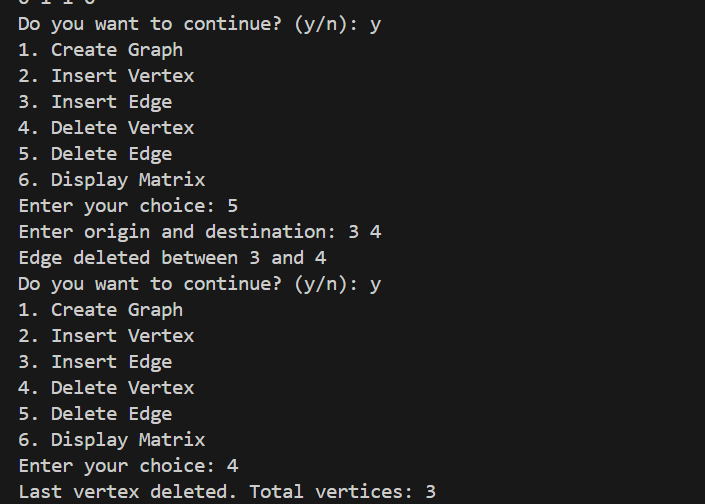
        scanner.close();

    }

}

Output :





**Q.4) DFS and BFS of graph.**

**DFS**

import java.util.Scanner

class DFS {

private int[][] adjMatrix = new int[20][20];

private int[] visitedarr = new int[20];

private int n;

public DFS() {

for (int i = 0; i < 20; i++) {

for (int j = 0; j < 20; j++) {

adjMatrix[i][j] = 0;

}

visitedarr[i] = 0;

}

}

public void createGraph() {

Scanner scanner = new Scanner(System.in);

int i, maxEdge, origin, destination;

System.out.print("Enter Number of vertices: ");

n = scanner.nextInt();

maxEdge = (n \* (n - 1)) / 2;

System.out.println("\nEnter the value of edges in adjacency matrix:");

for (i = 1; i <= maxEdge; i++) {

System.out.print("Enter 0 0 to exit or enter origin and destination for:" + i + "\n");

origin = scanner.nextInt();

destination = scanner.nextInt();

if ((origin == 0) || (destination == 0)) {

break;

}

if ((origin > n) || (origin < 0) || (destination > n) || (destination < 0)) {

System.out.println("Invalid inputs");

i--;

return;

} else {

adjMatrix[origin][destination] = 1;

adjMatrix[destination][origin] = 1;

}

}

}

public void displayMatrix() {

System.out.println("\nFinal Adjacency Matrix:");

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

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

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

}

System.out.println();

}

}

void dfs(int x) {

int j;

visitedarr[x] = 1;

System.out.println(x + " is visited");

for (j = 1; j <= n; j++) {

if (adjMatrix[x][j] == 1 && visitedarr[j] == 0) {

dfs(j);

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

DFS a = new DFS();

a.createGraph();

a.displayMatrix();

System.out.println("Enter the first vertice : ");

int x = scanner.nextInt();

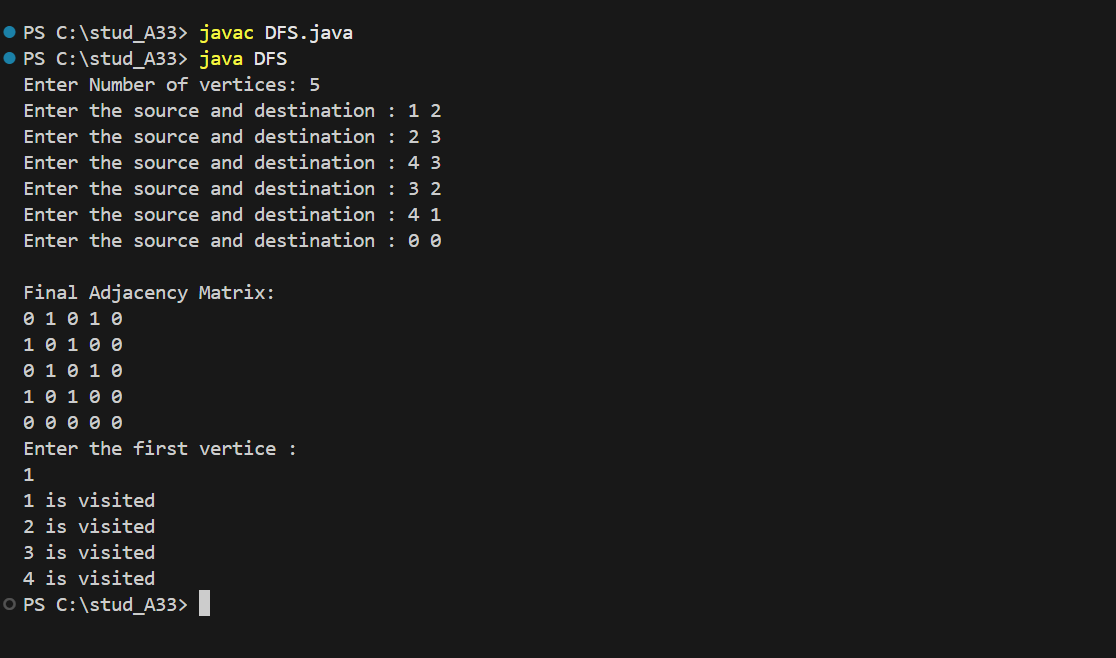
a.dfs(x);

scanner.close();

}

}

Output :



**BFS**

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

class BFS {

    private int[][] adjMatrix = new int[20][20];

    private int[] visitedarr = new int[20];

    private int n;

    public BFS() {

        for (int i = 0; i < 20; i++) {

            for (int j = 0; j < 20; j++) {

                adjMatrix[i][j] = 0;

            }

            visitedarr[i] = 0;

        }

    }

    public void createGraph() {

        Scanner scanner = new Scanner(System.in);

        int i, maxEdge, origin, destination;

        System.out.print("Enter Number of vertices: ");

        n = scanner.nextInt();

        maxEdge = (n \* (n - 1)) / 2;

        for (i = 1; i <= maxEdge; i++) {

            System.out.print("Enter the source and destination : ");

            origin = scanner.nextInt();

            destination = scanner.nextInt();

            if ((origin == 0) || (destination == 0)) {

                break;

            }

            if ((origin > n) || (origin < 0) || (destination > n) || (destination < 0)) {

                System.out.println("Invalid inputs");

                i--;

                return;

            } else {

                adjMatrix[origin][destination] = 1;

                adjMatrix[destination][origin] = 1;

            }

        }

    }

    public void displayMatrix() {

        System.out.println("\nFinal Adjacency Matrix:");

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

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

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

            }

            System.out.println();

        }

    }

    void bfs(int x) {

        Queue<Integer> queue = new LinkedList<>();

        queue.add(x);

        visitedarr[x] = 1;

        while (!queue.isEmpty()) {

            int node = queue.poll();

            System.out.println(node + " is visited");

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

                if (adjMatrix[node][j] == 1 && visitedarr[j] == 0) {

                    queue.add(j);

                    visitedarr[j] = 1;

                }

            }

        }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        BFS a = new BFS();

        a.createGraph();

        a.displayMatrix();

        System.out.println("Enter the first vertice : ");

        int x = scanner.nextInt();

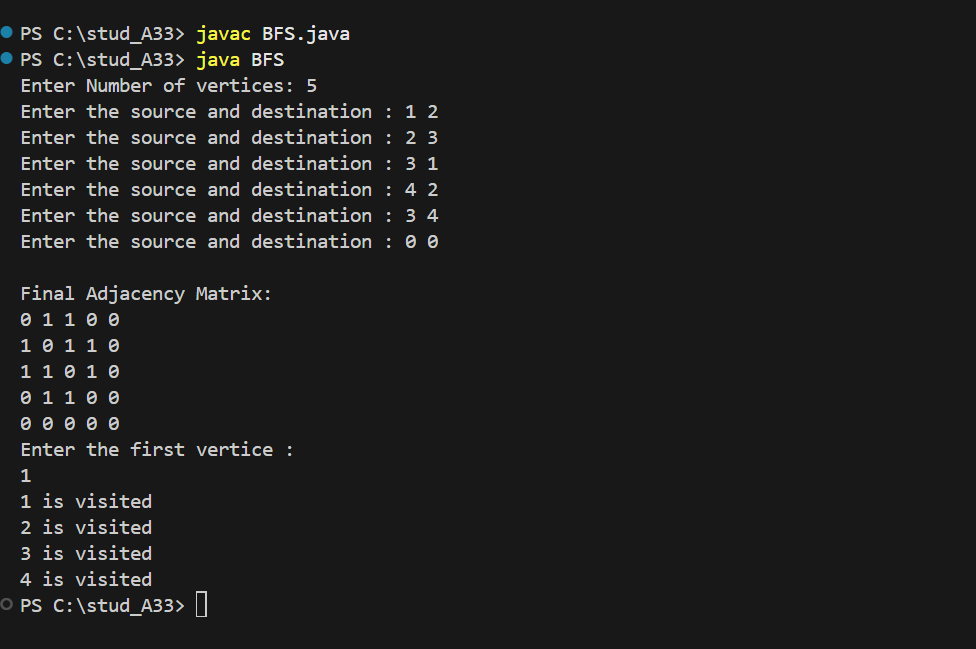
        a.bfs(x);

        scanner.close();

    }

}

Output :



**Q.4) Find the minimum spanning tree using prims algorithm.**

import java.util.Scanner;

class Prims {

    private int[][] graph;

    private int vertices;

    public Prims(int vertices) {

        this.vertices = vertices;

        graph = new int[vertices][vertices];

    }

    public void addEdge(int source, int destination, int weight) {

        graph[source - 1][destination - 1] = weight;

        graph[destination - 1][source - 1] = weight;

    }

    public void findMST() {

        int[] key = new int[vertices];

        boolean[] includedInMST = new boolean[vertices];

        int[] parent = new int[vertices];

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

            key[i] = Integer.MAX\_VALUE;

            includedInMST[i] = false;

        }

        key[0] = 0;

        parent[0] = -1;

        for (int i = 0; i < vertices - 1; i++) {

            int u = findMinKey(key, includedInMST);

            includedInMST[u] = true;

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

                if (graph[u][v] != 0 && !includedInMST[v] && graph[u][v] < key[v]) {

                    parent[v] = u;

                    key[v] = graph[u][v];

                }

            }

        }

        printMST(parent);

    }

    private int findMinKey(int[] key, boolean[] includedInMST) {

        int min = Integer.MAX\_VALUE, minIndex = -1;

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

            if (!includedInMST[i] && key[i] < min) {

                min = key[i];

                minIndex = i;

            }

        }

        return minIndex;

    }

    private void printMST(int[] parent) {

        int totalWeight = 0;

        System.out.println("Edge \t Weight");

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

            System.out.println((parent[i] + 1) + " - " + (i + 1) + "\t " + graph[i][parent[i]]);

            totalWeight += graph[i][parent[i]];

        }

        System.out.println("Total Weight of MST: " + totalWeight);

    }

}

public class MST {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in)

        System.out.print("Enter the number of vertices: ");

        int vertices = scanner.nextInt();

        Prims prims = new Prims(vertices);

        System.out.print("Enter the number of edges: ");

        int edges = scanner.nextInt();

        System.out.println("Enter the edges with weights (source destination weight):");

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

            int source = scanner.nextInt();

            int destination = scanner.nextInt();

            int weight = scanner.nextInt();

            prims.addEdge(source, destination, weight);

        }

        prims.findMST();

        scanner.close();

    }

}

Output :

