GRAPH

```java
/*Storing graph using adjacency  matrix representation
 */

import java.util.Scanner;

class adjacencyMatrix {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of vertices:");
        int n=sc.nextInt();
        System.out.println("Enter edges");
        int a[][]=new int[n][n];
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                a[i][j]=sc.nextInt();
            }
        }
        System.out.println("\nAdjacency Matrix Representation is : ");
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(a[i][j]==1){
                    System.out.print(i+"-->"+j+"    ");
                }
            }
            System.out.println();
        }
    }
}
```

```java
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Scanner;

public class adjacencyList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of vertices:");
        int n = sc.nextInt();

        LinkedHashMap<Integer, ArrayList<Integer>> map = new LinkedHashMap<>();

        for (int i = 0; i < n; i++) {
            map.put(i, new ArrayList<>());
        }
        int ch=1;
        System.out.println("Enter edges");
        while (true) {
            int v = sc.nextInt();
```

```
            int w = sc.nextInt();
            if (v < 0 || v > n || w < 0 || w > n) {
                System.out.println("Invalid vertex. Vertex numbers should be between 1 and
" + n + ".");
                continue;
            }
            map.get(v).add(w);
            System.out.println("Do you want to continue");
            ch=sc.nextInt();
            if(ch!=1)break;
        }
        System.out.println("Adjacency List:");
        for (int key : map.keySet()) {
            System.out.print(key + " --> ");
            for (int val : map.get(key)) {
                System.out.print(val + " ");
            }
            System.out.println();
        }
    }
}
```

DFSWithoutRecursion using AdjacencyMatrix:

```
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Scanner;

public class DFSWithoutRecursionAm {
    static Scanner sc = new Scanner(System.in);
    static int n;
    static int a[][];
    public static void main(String[] args) {
        System.out.println("Enter edges");
        n=sc.nextInt();
        a=new int[n][n];
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                System.out.println("Enter does there is edge between "+i+"    "+j);
                a[i][j]=sc.nextInt();
            }
        }
        System.out.println("Enter start vertex");
        int v=sc.nextInt();
        int visited[]=new int[n];
        Arrays.fill(visited,0);
        dfs(v,visited);
    }
    private static void dfs(int v,int [] visited){
```

```
        LinkedList<Integer> ll=new LinkedList<>();
        ll.push(v);
        while(!ll.isEmpty()){
            int s=ll.poll();
            if(visited[s]==1) continue;
            else{
                visited[s]=1;
                System.out.print(s+"   ");
            }
            for(int i=0;i<visited.length;i++){
                if(visited[i]==0 && a[s][i]==1){
                    ll.push(i);
                }
            }
        }
    }
}
```

DFSWithRecursion using AdjacencyMatrix:

```
import java.util.Arrays;
import java.util.Scanner;

public class DFSusingAMRecursion {
    static Scanner sc = new Scanner(System.in);
    static int n;
    static int a[][];
    public static void main(String[] args) {
        System.out.println("Enter edges");
        n=sc.nextInt();
        a=new int[n][n];
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                System.out.println("Enter does there is edge between "+i+"    "+j);
                a[i][j]=sc.nextInt();
            }
        }
        System.out.println("Enter start vertex");
        int v=sc.nextInt();
        int visited[]=new int[n];
        Arrays.fill(visited,0);
        dfs(v,visited);
    }
    private static void dfs(int v,int [] visited){
        int i;
        if(visited[v]!=1)
        {
```

```
            visited[v]=1;
            System.out.print(v+"   ");
        }
        for(i=0;i<n;i++){
            if(visited[i]==0 && a[v][i]==1){
                dfs(i,visited);
            }
        }
    }
}
```

DFSWithoutRecursion using AdjacencyList:

```
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.Scanner;

public class DFSWithoutRecusionAL {
    static class Graph{
        int n;
        LinkedHashMap<Integer, ArrayList<Integer>> adjList = new LinkedHashMap<>();
        Graph(int vertices) {
            n = vertices;
            for (int i = 0; i < vertices; i++) {
                adjList.put(i, new ArrayList<>());
            }
        }
        void addEdge(int v, int w) {
            if (!adjList.containsKey(v)) {
                adjList.put(v, new ArrayList<>());
            }
            adjList.get(v).add(w);
        }
        void DFS(int s) {
            boolean[] visited = new boolean[n];
            LinkedList<Integer> ll=new LinkedList<>();
            ll.push(s);
            while (!ll.isEmpty()) {
                s = ll.pop();
                if (!visited[s]) {
                    System.out.print(s + " ");
                    visited[s] = true;
                }
                for (int e: adjList.get(s)) {
                    if (!visited[e])
                        ll.push(e);
```

```
                }
            }
        }
    }
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter no of vertices");
        int n=sc.nextInt();
        Graph graph = new Graph(n);
        int ch=1;
        System.out.println("Enter edges");
        while (true) {
            int v = sc.nextInt();
            int w = sc.nextInt();
            if (v < 0 || v > n || w < 0 || w > n) {
                System.out.println("Invalid vertex. Vertex numbers should be between 1 and
" + n + ".");
                continue;
            }
            graph.addEdge(v, w);
            System.out.println("Do you want to continue");
            ch=sc.nextInt();
            if(ch!=1)break;
        }
        System.out.println("Enter starting vertex");
        int s=sc.nextInt();
        graph.DFS(s);
    }
}
```

WithRecusion:

```
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Scanner;

public class DFSWithRecusionAL {
    static class Graph{
        int n;
        LinkedHashMap<Integer, ArrayList<Integer>> adjList = new LinkedHashMap<>();
        Graph(int vertices) {
            n = vertices; // assign vertices to n
            for (int i = 0; i < vertices; i++) {
                adjList.put(i, new ArrayList<>());
            }
        }
        void addEdge(int v, int w) {
            if (!adjList.containsKey(v)) {
                adjList.put(v, new ArrayList<>());
```

```
            }
            adjList.get(v).add(w);
        }
        void DFS(int s) {
            boolean[] visited = new boolean[n];
            recursiveDFS(s, visited);
        }

        void recursiveDFS(int current, boolean[] visited) {
            visited[current] = true;
            System.out.print(current + " ");
            for (int neighbor : adjList.get(current)) {
                if (!visited[neighbor]) {
                    recursiveDFS(neighbor, visited);
                }
            }
        }

    }
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter no of vertices");
        int n=sc.nextInt();
        Graph graph = new Graph(n);
        int ch=1;
        System.out.println("Enter edges");
        while (true) {
            int v = sc.nextInt();
            int w = sc.nextInt();
            if (v < 0 || v > n || w < 0 || w > n) {
                System.out.println("Invalid vertex. Vertex numbers should be between 1 and
" + n + ".");
                continue;
            }
            graph.addEdge(v, w);
            System.out.println("Do you want to continue");
            ch=sc.nextInt();
            if(ch!=1)break;
        }
        System.out.println("Enter starting vertex");
        int s=sc.nextInt();
        graph.DFS(s);
    }
}
```