1. Write a Java program to construct a graph using adjacency matrix and implement DFS and BFS traversing of it.

Code:

```
package graph;

import java.util.Iterator;
import java.util.Scanner;

public class Graph {
	private int[][] adjacencyMatrix;
	private int numVertices;
	public Graph(int numVertices) {
		super();
		this.numVertices = numVertices;
		adjacencyMatrix = new int[numVertices][numVertices];
	}
	public void addEdge(int start,int end)
	{
		adjacencyMatrix[start][end]=1;
		adjacencyMatrix[end][start]=1;
	}
	public void bfs(int startVertex)
	{
		boolean[] visited=new boolean[numVertices];
		int[]queue=new int[numVertices];
		int front=0,rear=0;

		visited[startVertex]=true;
		queue[rear++]=startVertex;
		System.out.println("BFS Traversal Starting from vertex "+ startVertex+":");

		while (front<rear)
		{
			int currentVertex=queue[front++];
			System.out.print(currentVertex+" ");

			for(int i=0;i<numVertices;i++)
			{
				if (adjacencyMatrix[currentVertex][i]==1 && !visited[i])
				{
					visited[i]=true;
					queue[rear++]=i;
				}
			}
		}
		System.out.println();
	}
	public void dfs(int startVertex)
```

```
{
        boolean[] visited=new boolean[numVertices];
        int[] stack=new int[numVertices];
        int top=-1;

        stack[++top]=startVertex;
        visited[startVertex]=true;
        System.out.println("DFS traversal staring from vertex"+ startVertex+" : ");
        while (top>=0)
        {
                int currentVertex=stack[top--];
                System.out.print(currentVertex+" ");

                for(int i= numVertices-1;i>=0;i--)
                {
                        if (adjacencyMatrix[currentVertex][i]==1 && !visited[i])
                        {
                                visited[i]=true;
                                stack[++top]=i;
                        }
                }
        }
        System.out.println();
}
public void displayAdjacencyMatrix()
{
        System.out.println("Adjancy Matrix: ");
        for(int i=0;i<numVertices;i++)
        {
                for(int j=0;j<numVertices;j++)
                {
                        System.out.print(adjacencyMatrix[i][j]+" ");
                }
                System.out.println();
        }
}
public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);

   System.out.print("Enter the number of vertices: ");
   int numVertices = scanner.nextInt();

   Graph graph = new Graph(numVertices);

   System.out.print("Enter the number of edges: ");
   int numEdge = scanner.nextInt();

   System.out.println("Enter the edge (Source and Destination): ");
   for (int i = 0; i < numEdge; i++) {
      int src = scanner.nextInt();
```

```
            int dest = scanner.nextInt();
            graph.addEdge(src, dest);
        }

        graph.displayAdjacencyMatrix();

        System.out.println("Enter the start vertex for BFS: ");
        int bfsStartVertex = scanner.nextInt();
        graph.bfs(bfsStartVertex);

        System.out.println("Enter the start vertex for DFS: ");
        int dfsStartVertex = scanner.nextInt();
        graph.dfs(dfsStartVertex);  // Corrected the variable name here

        scanner.close();
    }
}
```

Output:

```
Enter the number of vertices: 4
Enter the number of edges: 4
Enter the edge (Source and Destination):
0 1
1 2
2 3
0 2
Adjancy Matrix:
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0
Enter the start vertex for BFS:
```

2. Write a Java program to find the minimum spanning tree of a graph.

Code:

```
package graph;

import java.util.Scanner;

class Edge{
    int src,dest,weight;

    public Edge(int src,int dest, int weight)
    {
        this.src=src;
        this.dest=dest;
        this.weight=weight;
    }
```

```
}

public class KruskalAlgorithm {
	private int V;
	private int E;
	private Edge[] edges;
	private int edgeCount=0;


	public KruskalAlgorithm(int verices,int edgeCount)
	{
		this.V=verices;
		this.E=edgeCount;
		edges=new Edge[edgeCount];
	}

	public void addEdge(int src,int dest,int weight)
	{
		edges[edgeCount++]=new Edge(src, dest, weight);
	}
	private int findParent(int[] parent,int vertex)
	{
		if (parent[vertex]!=vertex)
		{
			parent[vertex]=findParent(parent, parent[vertex]);
		}
		return parent[vertex];

	}
	private void union(int[] parent,int[] rank,int x,int y)
	{
		int rootX=findParent(parent, x);
		int rootY=findParent(parent, y);

		if (rootX!=rootY)
		{
			if (rank[rootX]<rank[rootY])
			{
				parent[rootX]=rootY;
			}else if (rank[rootX]>rank[rootY])
			{
				parent[rootY]=rootX;
			}else {
				parent[rootY]=rootX;
				rank[rootX]++;
			}
		}
	}
	private void sortEdges()
	{
```

```
		for(int i=0;i<E-1;i++)
		{
			for(int j=0;j<E-i-1;j++)
			{
				if (edges[j].weight>edges[j+1].weight)
				{
					Edge tempEdge=edges[j];
					edges[j]=edges[j+1];
					edges[j+1]=tempEdge;
				}
			}
		}
}
public void kruskalMST()
{
		sortEdges();

		int[] parent=new int[V];
		int[] rank=new int[V];

		for(int i=0;i<V;i++)
		{
			parent[i]=i;
			rank[i]=0;
		}
		Edge[] mstEdges=new Edge[V-1];
		int mstIndex=0;
		int mstWeight=0;

		for(int i=0;i<E;i++)
		{
			if (mstIndex==V-1)break;

			Edge edge=edges[i];
			int srcParent=findParent(parent, edge.src);
			int destParent=findParent(parent, edge.dest);

			if (srcParent!=destParent)
			{
				mstEdges[mstIndex++]=edge;
				mstWeight+=edge.weight;
				union(parent, rank, srcParent, destParent);
			}
		}
		System.out.println("Edge in the MST: ");
		System.out.println("src des weight");

		for(int i=0;i<mstIndex;i++)
		{
```

```
                        System.out.println(mstEdges[i].src+"--
"+mstEdges[i].dest+"=="+mstEdges[i].weight);
                }
                System.out.println("Total Wieght of MST: "+mstWeight);
        }
        public static void main(String[] args)
        {
                Scanner scanner=new Scanner(System.in);

                System.out.println("Ente rthe number of vertices");
                int V=scanner.nextInt();

                System.out.println("Enter the number of edge: ");
                int E=scanner.nextInt();

                KruskalAlgorithm graph=new KruskalAlgorithm(V, E);

                System.out.println("Enter the Edge in the format:src dest weight");
                for(int i=0;i<E;i++)
                {
                        int src=scanner.nextInt();
                        int dest=scanner.nextInt();
                        int weight=scanner.nextInt();
                        graph.addEdge(src, dest, weight);
                }
                graph.kruskalMST();
                scanner.close();
        }

}
```

Output:

```
Ente rthe number of vertices
5
Enter the number of edge:
7
Enter the Edge in the format:src dest weight
0 1 2
0 3 6
1 2 3
1 3 8
1 4 5
2 4 7
3 4 7
Edge in the MST:
src des weight
0--1==2
1--2==3
1--4==5
0--3==6
Total Wieght of MST: 16
```

Conclusion:

Graphs are a fundamental data structure used to represent connections or relationships between entities, often modeled as vertices (nodes) connected by edges. The first program demonstrates constructing a graph using an adjacency matrix, which is a simple way to represent edges between vertices. It implements **Breadth-First Search (BFS)** and **Depth-First Search (DFS)** to explore the graph. BFS explores level by level, while DFS dives deep into one branch before backtracking. Both are crucial for tasks like network analysis, pathfinding, and traversing connected components.

The second program uses **Kruskal's Algorithm** to find the **Minimum Spanning Tree (MST)** of a graph, which connects all vertices with the least total edge weight. MSTs are vital for optimizing networks like electrical grids or road systems. By sorting edges by weight and using union-find to prevent cycles, the algorithm efficiently selects the edges of the MST. Together, these programs illustrate essential graph concepts and their applications in problem-solving and optimization.