```java
// Q1. Represent a graph using adjacency matrix/list and perform basic
operations
import java.util.*;
class Q1GraphBasics {
    static int outDegree(int v, int[][] mat) {
        int n = mat.length, deg = 0;
        for (int j = 0; j < n; j++) if (mat[v][j] == 1) deg++;
        return deg;
    }
    static int inDegree(int v, int[][] mat) {
        int n = mat.length, deg = 0;
        for (int i = 0; i < n; i++) if (mat[i][v] == 1) deg++;
        return deg;
    }
    static int countEdgesMatrix(int[][] mat) {
        int n = mat.length, edges = 0;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (mat[i][j] == 1) edges++;
        return edges;
    }
    static void printAdjacent(int v, ArrayList<Integer>[] adjList) {
        for (int x : adjList[v]) System.out.print(x + " ");
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(), m = sc.nextInt();
        int[][] adjMat = new int[n][n];
        @SuppressWarnings("unchecked")
        ArrayList<Integer>[] adjList = new ArrayList[n];
        for (int i = 0; i < n; i++) adjList[i] = new ArrayList<>();
        for (int i = 0; i < m; i++) {
            int u = sc.nextInt(), v = sc.nextInt();
            adjMat[u][v] = 1;
            adjList[u].add(v);
        }
        int v = sc.nextInt();
        System.out.println(outDegree(v, adjMat));
        System.out.println(inDegree(v, adjMat));
        System.out.println(countEdgesMatrix(adjMat));
        printAdjacent(v, adjList);
        sc.close();
    }
}
```

```java
// Q2. Breadth First Search (BFS)
import java.util.*;
class Q2BFS {
    static ArrayList<Integer> bfs(int n, ArrayList<Integer>[] adj) {
        boolean[] vis = new boolean[n];
        Queue<Integer> q = new LinkedList<>();
        ArrayList<Integer> res = new ArrayList<>();
        vis[0] = true;
        q.add(0);
        while (!q.isEmpty()) {
            int node = q.poll();
            res.add(node);
            for (int x : adj[node]) {
                if (!vis[x]) {
                    vis[x] = true;
                    q.add(x);
                }
            }
        }
        return res;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(), m = sc.nextInt();
        @SuppressWarnings("unchecked")
        ArrayList<Integer>[] adj = new ArrayList[n];
        for (int i = 0; i < n; i++) adj[i] = new ArrayList<>();
        for (int i = 0; i < m; i++) {
            int u = sc.nextInt(), v = sc.nextInt();
            adj[u].add(v);
            adj[v].add(u);
        }
        ArrayList<Integer> ans = bfs(n, adj);
        for (int x : ans) System.out.print(x + " ");
        sc.close();
    }
}
```

```java
// Q3. Depth First Search (DFS)
import java.util.*;
class Q3DFS {
    static void dfs(int node, boolean[] vis, ArrayList<Integer>[] adj,
ArrayList<Integer> res) {
        vis[node] = true;
        res.add(node);
        for (int x : adj[node]) {
            if (!vis[x]) dfs(x, vis, adj, res);
        }
    }
    static ArrayList<Integer> dfsOfGraph(int n, ArrayList<Integer>[] adj) {
        boolean[] vis = new boolean[n];
        ArrayList<Integer> res = new ArrayList<>();
        dfs(0, vis, adj, res);
        return res;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(), m = sc.nextInt();
        @SuppressWarnings("unchecked")
        ArrayList<Integer>[] adj = new ArrayList[n];
        for (int i = 0; i < n; i++) adj[i] = new ArrayList<>();
        for (int i = 0; i < m; i++) {
            int u = sc.nextInt(), v = sc.nextInt();
            adj[u].add(v);
            adj[v].add(u);
        }
        ArrayList<Integer> ans = dfsOfGraph(n, adj);
        for (int x : ans) System.out.print(x + " ");
        sc.close();
    }
}
```

```java
// Q4. Kruskal's Minimum Spanning Tree
import java.util.*;
class Q4Kruskal {
    static class DisjointSet {
        int[] parent, size;
        DisjointSet(int n) {
            parent = new int[n + 1];
            size = new int[n + 1];
            for (int i = 0; i <= n; i++) {
                parent[i] = i;
                size[i] = 1;
            }
        }
        int find(int x) {
            if (parent[x] == x) return x;
            parent[x] = find(parent[x]);
            return parent[x];
        }
        void unionSet(int u, int v) {
            u = find(u);
            v = find(v);
            if (u == v) return;
            if (size[u] < size[v]) {
                parent[u] = v;
                size[v] += size[u];
            } else {
                parent[v] = u;
                size[u] += size[v];
            }
        }
    }
    static class Edge {
        int u, v, w;
        Edge(int u, int v, int w) {
            this.u = u;
            this.v = v;
            this.w = w;
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(), m = sc.nextInt();
        ArrayList<Edge> edges = new ArrayList<>();
        for (int i = 0; i < m; i++) {
            int u = sc.nextInt(), v = sc.nextInt(), w = sc.nextInt();
            edges.add(new Edge(u, v, w));
        }
        Collections.sort(edges, new Comparator<Edge>() {
            public int compare(Edge a, Edge b) {
                return Integer.compare(a.w, b.w);
            }
        });
        DisjointSet ds = new DisjointSet(n);
        int mst = 0;
        for (Edge e : edges) {
            if (ds.find(e.u) != ds.find(e.v)) {
                mst += e.w;
                ds.unionSet(e.u, e.v);
            }
        }
        System.out.println(mst);
        sc.close();
    }
}

// Q5. Prim's Minimum Spanning Tree
import java.util.*;
class Q5Prim {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(), m = sc.nextInt();
        @SuppressWarnings("unchecked")
        ArrayList<int[]>[] adj = new ArrayList[n];
        for (int i = 0; i < n; i++) adj[i] = new ArrayList<>();
        for (int i = 0; i < m; i++) {
            int u = sc.nextInt(), v = sc.nextInt(), w = sc.nextInt();
            adj[u].add(new int[]{v, w});
            adj[v].add(new int[]{u, w});
        }
        boolean[] vis = new boolean[n];
        PriorityQueue<int[]> pq = new PriorityQueue<>(new Comparator<int[]>()
{
            public int compare(int[] a, int[] b) {
                return Integer.compare(a[0], b[0]);
            }
        });
        pq.add(new int[]{0, 0});
        int sum = 0;
        while (!pq.isEmpty()) {
            int[] top = pq.poll();
            int wt = top[0], node = top[1];
            if (vis[node]) continue;
            vis[node] = true;
            sum += wt;
            for (int[] edge : adj[node]) {
                int adjNode = edge[0], weight = edge[1];
                if (!vis[adjNode]) pq.add(new int[]{weight, adjNode});
            }
        }
        System.out.println(sum);
        sc.close();
    }
}

// Q6. Dijkstra's Shortest Path
import java.util.*;
class Q6Dijkstra {
    static int[] dijkstra(int n, ArrayList<int[]>[] adj, int s) {
        int[] dist = new int[n];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[s] = 0;
        PriorityQueue<int[]> pq = new PriorityQueue<>(new Comparator<int[]>()
{
            public int compare(int[] a, int[] b) {
                return Integer.compare(a[0], b[0]);
            }
        });
        pq.add(new int[]{0, s});
        while (!pq.isEmpty()) {
            int[] top = pq.poll();
            int d = top[0], node = top[1];
            if (d > dist[node]) continue;
            for (int[] edge : adj[node]) {
                int adjNode = edge[0], wt = edge[1];
                if (d + wt < dist[adjNode]) {
                    dist[adjNode] = d + wt;
                    pq.add(new int[]{dist[adjNode], adjNode});
                }
            }
        }
        return dist;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(), m = sc.nextInt();
        @SuppressWarnings("unchecked")
        ArrayList<int[]>[] adj = new ArrayList[n];
        for (int i = 0; i < n; i++) adj[i] = new ArrayList<>();
        for (int i = 0; i < m; i++) {
            int u = sc.nextInt(), v = sc.nextInt(), w = sc.nextInt();
            adj[u].add(new int[]{v, w});
        }
        int s = sc.nextInt();
        int[] dist = dijkstra(n, adj, s);
        for (int i = 0; i < n; i++) System.out.print(dist[i] + " ");
        sc.close();
    }
}
```