Graph Algorithms in Java

Graphs model relationships, and their algorithms solve problems like shortest paths and connectivity. Let's cover DFS, BFS, Dijkstra's, and Kruskal's.

1. DFS: Depth-First Search

DFS explores as far as possible along each branch before backtracking.

Java Implementation

```java
import java.util.*;

public class DFS {
    static class Graph {
        int V;
        LinkedList<Integer>[] adj;

        Graph(int v) {
            V = v;
            adj = new LinkedList[v];
            for (int i = 0; i < v; i++) adj[i] = new LinkedList<>();
        }

        void addEdge(int v, int w) { adj[v].add(w); }

        void DFS(int v, boolean[] visited) {
            visited[v] = true;
            System.out.print(v + " ");
            for (int n : adj[v]) if (!visited[n]) DFS(n, visited);
        }
    }

    public static void main(String[] args) {
        Graph g = new Graph(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
```

```java
        System.out.print("DFS: ");
        g.DFS(2, new boolean[4]);
    }
}
```

Output: `DFS: 2 0 1 3`

2. BFS: Breadth-First Search

BFS explores level by level using a queue.

Java Implementation

```java
import java.util.*;

public class BFS {
    static class Graph {
        int V;
        LinkedList<Integer>[] adj;

        Graph(int v) {
            V = v;
            adj = new LinkedList[v];
            for (int i = 0; i < v; i++) adj[i] = new LinkedList<>();
        }

        void addEdge(int v, int w) { adj[v].add(w); }

        void BFS(int s) {
            boolean[] visited = new boolean[V];
            Queue<Integer> queue = new LinkedList<>();
            visited[s] = true;
            queue.add(s);
            while (!queue.isEmpty()) {
                s = queue.poll();
                System.out.print(s + " ");
                for (int n : adj[s]) {
                    if (!visited[n]) {
                        visited[n] = true;
                        queue.add(n);
```

```java
                    }
                }
            }
        }
    }

    public static void main(String[] args) {
        Graph g = new Graph(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        System.out.print("BFS: ");
        g.BFS(2);
    }
}
```

Output: `BFS: 2 0 3 1`

3. Dijkstra's Algorithm: Shortest Paths

Dijkstra's finds the shortest path in a weighted graph.

Java Implementation

```java
import java.util.*;

public class Dijkstra {
    static class Graph {
        int V;
        List<List<int[]>> adj;

        Graph(int v) {
            V = v;
            adj = new ArrayList<>(v);
            for (int i = 0; i < v; i++) adj.add(new ArrayList<>());
        }

        void addEdge(int u, int v, int weight) { adj.get(u).add(new int[]{v, weight}); }
```

```java
        void dijkstra(int src) {
            PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[1]));
            int[] dist = new int[V];
            Arrays.fill(dist, Integer.MAX_VALUE);
            dist[src] = 0;
            pq.add(new int[]{src, 0});
            while (!pq.isEmpty()) {
                int u = pq.poll()[0];
                for (int[] neighbor : adj.get(u)) {
                    int v = neighbor[0], weight = neighbor[1];
                    if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
                        dist[v] = dist[u] + weight;
                        pq.add(new int[]{v, dist[v]});
                    }
                }
            }
            System.out.println("Distances from " + src + ": " + Arrays.toString(dist));
        }
    }

    public static void main(String[] args) {
        Graph g = new Graph(4);
        g.addEdge(0, 1, 4);
        g.addEdge(0, 2, 1);
        g.addEdge(2, 1, 2);
        g.addEdge(1, 3, 5);
        g.dijkstra(0);
    }
}
```

Output: `Distances from 0: [0, 3, 1, 8]`

4. Kruskal’s Algorithm: Minimum Spanning Tree

Kruskal’s builds an MST by sorting edges and adding them greedily.

Java Implementation

```
import java.util.*;

public class Kruskal {
    static class Edge implements Comparable<Edge> {
        int src, dest, weight;
        Edge(int src, int dest, int weight) { this.src = src; this.dest = dest; this.weight = weight; }
        public int compareTo(Edge other) { return this.weight - other.weight; }
    }

    static class Graph {
        int V, E;
        Edge[] edges;

        Graph(int v, int e) {
            V = v;
            E = e;
            edges = new Edge[e];
        }

        int find(int[] parent, int i) {
            if (parent[i] != i) parent[i] = find(parent, parent[i]);
            return parent[i];
        }

        void union(int[] parent, int[] rank, int x, int y) {
            int rootX = find(parent, x), rootY = find(parent, y);
            if (rank[rootX] < rank[rootY]) parent[rootX] = rootY;
            else if (rank[rootX] > rank[rootY]) parent[rootY] = rootX;
            else { parent[rootY] = rootX; rank[rootX]++; }
        }

        void kruskalMST() {
            Edge[] result = new Edge[V - 1];
            int e = 0, i = 0;
            Arrays.sort(edges);
            int[] parent = new int[V];
            int[] rank = new int[V];
            for (int v = 0; v < V; v++) parent[v] = v;
```

```
            while (e < V - 1 && i < E) {
                Edge next = edges[i++];
                int x = find(parent, next.src), y = find(parent, next.dest);
                if (x != y) result[e++] = next;
            }
            System.out.println("MST Edges:");
            for (i = 0; i < e; i++) System.out.println(result[i].src + " -- " + result[i].dest + " : " + resul
        }
    }

    public static void main(String[] args) {
        Graph g = new Graph(4, 5);
        g.edges[0] = new Edge(0, 1, 10);
        g.edges[1] = new Edge(0, 2, 6);
        g.edges[2] = new Edge(0, 3, 5);
        g.edges[3] = new Edge(1, 3, 15);
        g.edges[4] = new Edge(2, 3, 4);
        g.kruskalMST();
    }
}
```

Output:

```
MST Edges:
2 -- 3 : 4
0 -- 3 : 5
0 -- 1 : 10
```