```
import java.util.List;
import java.util.LinkedList;
import java.util.Arrays;
import java.util.Queue;
import java.util.ArrayDeque;
import java.util.Iterator;
import java.util.Stack;

public class wtg {

    public static void main(String[] args) {
        List<Edge> edges = Arrays.asList(
            new Edge(0,1,10), new Edge(1,2,9), new Edge(1,3,8),
            new Edge(3,4,7), new Edge(4,5,6), new Edge(4,6,5)
        );
        Graph g = new Graph(edges);
        bfs(g, 0);
        System.out.println();
        dfs(g, 0);
        System.out.println();
        print(g);
        System.out.println();

    }


    private static void dfs(Graph g, int src) {
        Stack<Integer> stk = new Stack<>();
        boolean[] vis = new boolean[g.adj.size() + 1];
        vis[src] = true;
        stk.add(src);

        while (!stk.isEmpty()) {
            src = stk.pop();
            System.out.print(src + " ");
            for (Integer nodes : g.adj.get(src)) {
                if (!vis[nodes]) {
                    vis[nodes] = true;
                    stk.add(nodes);
                }
            }
        }
    }


    private static void print(Graph g) {
        List<Integer> travel;
        while (!g.adj.isEmpty()) {
            for (int i = 0; i < g.adj.size(); i++) {
```

```
            travel = ((LinkedList<List<Integer>>) g.adj).getFirst();
            Iterator<Integer> it = travel.iterator();
            while (it.hasNext()) {
                System.out.println(it.next());
            }
            travel = g.adj.remove(0);
        }
    }
}


private static void bfs(Graph g, int src) {
    Queue<Integer> q = new ArrayDeque<>();
    boolean[] vis = new boolean[g.adj.size() + 1];
    vis[src] = true;
    q.add(src);

    while (!q.isEmpty()) {
        src = q.poll();
        System.out.print(src + " ");

        for (Integer nodes : g.adj.get(src)) {
            if (!vis[nodes]) {
                vis[nodes] = true;
                q.add(nodes);
            }
        }
    }
}


static class Graph {
    List<List<Integer>> adj;

    public Graph(List<Edge> edges) {
        int max = Integer.MIN_VALUE;
        adj = new LinkedList<>();

        for (Edge edge : edges) {
            max = Integer.max(max, Integer.max(edge.dest, edge.src));
        }
        for (int i = 0; i < max + 1; i++) {
            adj.add(new LinkedList<>());
        }
        for (Edge edge : edges) {
            adj.get(edge.src).add(edge.dest);
        }
    }
}
```

```
    static class Edge {
        int src, dest, weight;

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


}
```