import java.util.\*;

public class App {

public static void main(String[] args) {

Graph graph = new Graph(5, false);

graph.addEdge(0, 1);

graph.addEdge(0, 4);

graph.addEdge(1, 2);

graph.addEdge(1, 4);

graph.addEdge(1, 3);

graph.addEdge(3, 4);

graph.addEdge(2, 3);

graph.printGraph();

System.out.println("DFS starting from vertex 0:");

graph.dfs(0, new boolean[5]);

System.out.println();

System.out.println("BFS starting from vertex 0:");

graph.bfs(0);

System.out.println();

}

}

public class Graph {

private ArrayList<HashSet<Integer>> graph;

private int vertices;

private boolean isDirected;

public Graph(int vertices, boolean isDirected) {

this.vertices = vertices;

this.isDirected = isDirected;

graph = new ArrayList<>();

for (int i = 0; i < vertices; i++) {

graph.add(new HashSet<>());

}

}

private boolean isValidVertex(int v) {

return v >= 0 && v < vertices;

}

public void addEdge(int src, int dest) {

if (!isValidVertex(src) || !isValidVertex(dest)) {

System.out.println("Invalid source or destination");

return;

}

graph.get(src).add(dest);

if (!isDirected) {

graph.get(dest).add(src);

}

}

public void removeEdge(int src, int dest) {

if (!isValidVertex(src) || !isValidVertex(dest)) {

System.out.println("Invalid source or destination");

return;

}

graph.get(src).remove(dest);

if (!isDirected) {

graph.get(dest).remove(src);

}

}

public void printGraph() {

System.out.println("Graph adjacency list:");

for (int i = 0; i < vertices; i++) {

System.out.print(i + ": ");

for (int adj : graph.get(i)) {

System.out.print(adj + " ");

}

System.out.println();

}

}

// DFS recursive helper

public void dfs(int v, boolean[] visited) {

if (!isValidVertex(v)) {

System.out.println("Invalid start vertex for DFS");

return;

}

visited[v] = true;

System.out.print(v + " ");

for (int neighbor : graph.get(v)) {

if (!visited[neighbor]) {

dfs(neighbor, visited);

}

}

}

// BFS iterative

public void bfs(int start) {

if (!isValidVertex(start)) {

System.out.println("Invalid start vertex for BFS");

return;

}

boolean[] visited = new boolean[vertices];

Queue<Integer> queue = new LinkedList<>();

visited[start] = true;

queue.offer(start);

while (!queue.isEmpty()) {

int current = queue.poll();

System.out.print(current + " ");

for (int neighbor : graph.get(current)) {

if (!visited[neighbor]) {

visited[neighbor] = true;

queue.offer(neighbor);

}

}

}

}

}