**Aim: Implement depth first search algorithm and Breadth First Search algorithm, Use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure**

**DEPTH FIRST SEARCH ALGORITHM**

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

class Edge

{

int source, dest;

public Edge(int source, int dest)

{

this.source = source;

this.dest = dest;

}

}

class Graph

{

List<List<Integer>> adjList = null;

Graph(List<Edge> edges, int n)

{

adjList = new ArrayList<>();

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

adjList.add(new ArrayList<>());

}

for (Edge edge: edges)

{

int src = edge.source;

int dest = edge.dest;

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

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

}

}

}

class Main

{

public static void DFS(Graph graph, int v, boolean[] discovered)

{

discovered[v] = true;

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

for (int u: graph.adjList.get(v))

{

if (!discovered[u]) {

DFS(graph, u, discovered);

}

}

}

public static void main(String[] args)

{

List<Edge> edges = Arrays.asList(

new Edge(1, 2), new Edge(1, 7), new Edge(1, 8), new Edge(2, 3),

new Edge(2, 6), new Edge(3, 4), new Edge(3, 5), new Edge(8, 9),

new Edge(8, 12), new Edge(9, 10), new Edge(9, 11)

);

int n = 13;

Graph graph = new Graph(edges, n);

boolean[] discovered = new boolean[n];

for (int i = 0; i < n; i++)

{

if (!discovered[i]) {

DFS(graph, i, discovered);

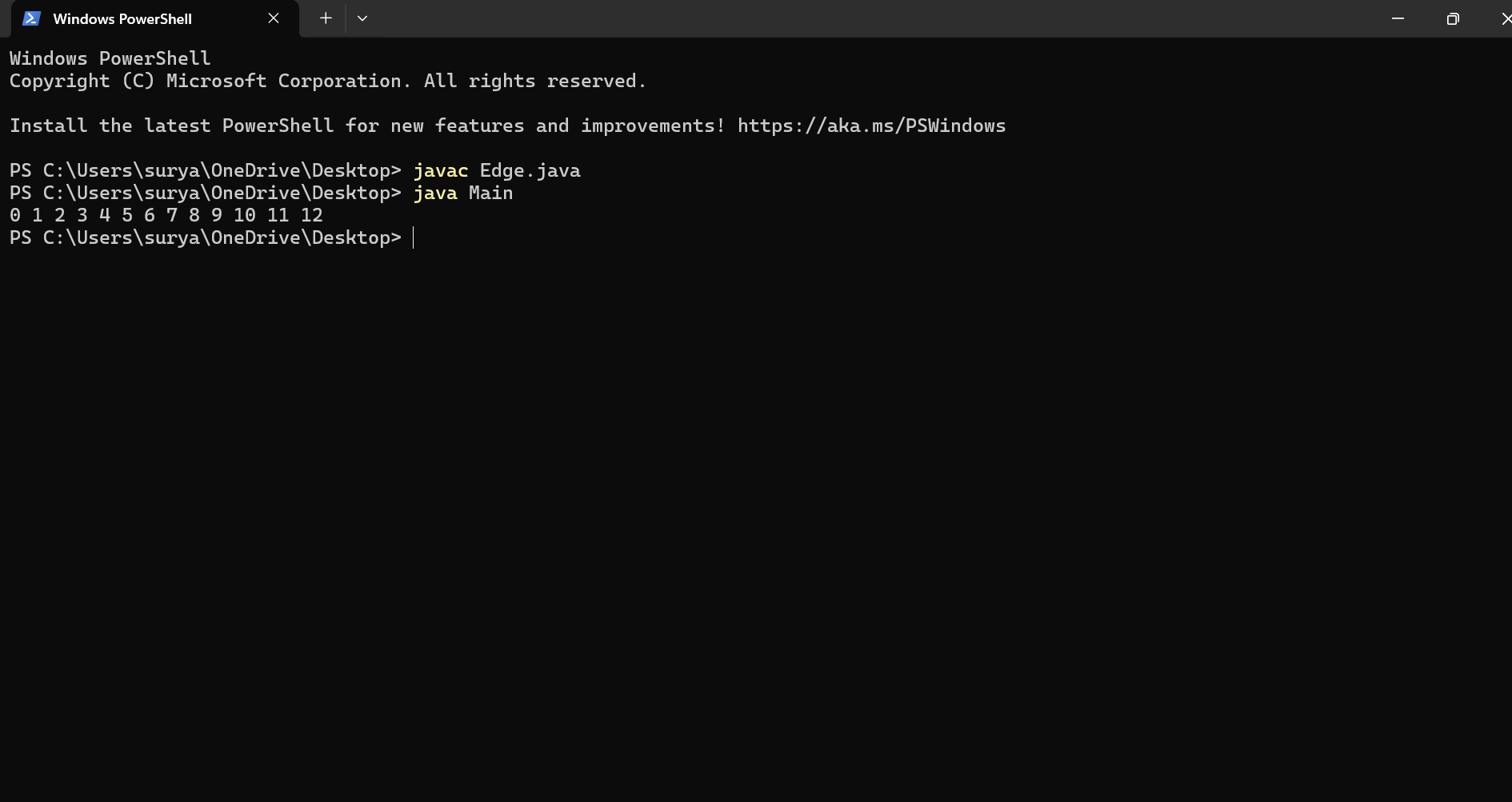
}

}

}

}

**OUTPUT:** 0 1 2 3 4 5 6 7 8 9 10 11 12



**BREADTH FIRST SEARCH ALGORITHM**

import java.util.\*;

class EdgeBFS

{

int source, dest;

public EdgeBFS(int source, int dest)

{

this.source = source;

this.dest = dest;

}

}

class Graph

{

List<List<Integer>> adjList = null;

Graph(List<Edge> edges, int n)

{

adjList = new ArrayList<>();

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

adjList.add(new ArrayList<>());

}

for (Edge edge: edges)

{

int src = edge.source;

int dest = edge.dest;

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

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

}

}

}

class Main

{

public static void recursiveBFS(Graph graph, Queue<Integer> q,

boolean[] discovered)

{

if (q.isEmpty()) {

return;

}

int v = q.poll();

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

for (int u: graph.adjList.get(v))

{

if (!discovered[u])

{

discovered[u] = true;

q.add(u);

}

}

recursiveBFS(graph, q, discovered);

}

public static void main(String[] args)

{

List<Edge> edges = Arrays.asList(

new Edge(1, 2), new Edge(1, 3), new Edge(1, 4), new Edge(2, 5),

new Edge(2, 6), new Edge(5, 9), new Edge(5, 10), new Edge(4, 7),

new Edge(4, 8), new Edge(7, 11), new Edge(7, 12)

);

int n = 15;

Graph graph = new Graph(edges, n);

boolean[] discovered = new boolean[n];

Queue<Integer> q = new ArrayDeque<>();

for (int i = 0; i < n; i++)

{

if (discovered[i] == false)

{

discovered[i] = true;

q.add(i);

recursiveBFS(graph, q, discovered);

}

}

}

}

**OUTPUT:** 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

