1.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.

Python code with dynamic input

*from* collections *import* deque  
  
  
# A class to represent a graph object  
*class* Graph:  
 # Constructor  
 *def \_\_init\_\_*(self, *edges*, *n*):  
 self.adjList = [[] *for* \_ *in* range(*n*)]  
  
 # add edges to the undirected graph  
 *for* (src, dest) *in edges*:  
 self.adjList[src].append(dest)  
 self.adjList[dest].append(src)  
  
 # Function to perform BFS recursively on the graph  
  
  
*def* recursiveBFS(*graph*, *q*, *discovered*):  
 *if not q*:  
 *return* # dequeue front node and print it  
 v = *q*.popleft()  
 print(v, end=' ')  
  
 # do for every edge (v, u)  
 *for* u *in graph*.adjList[v]:  
 *if not discovered*[u]:  
 # mark it as discovered and enqueue it  
 *discovered*[u] = *True  
 q*.append(u)  
  
 recursiveBFS(*graph*, *q*, *discovered*)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 # List of graph edges as per the above diagram  
 # edges = [  
 # Notice that node 0 is unconnected  
 # (1, 8), (1, 5), (1, 2), (8, 6), (8, 4), (8, 3),  
 # (6, 10), (6, 7), (2, 9)  
  
 # ]  
 edges = list(tuple(map(int, input().split())) *for* r *in* range(int(input("Enter edges:"))))  
 print(edges)  
  
 # total number of nodes in the graph  
 # n = 11  
 n = int(input("Enter value of n:"))  
  
 # build a graph from the given edges  
 graph = Graph(edges, n)  
  
 # to keep track of whether a vertex is discovered or not  
discovered = [*False*] \* n  
  
# create a queue for doing BFS  
q = deque()  
  
# Perform BFS traversal from all undiscovered nodes  
print("\nFollowing is Breadth First Traversal: ")  
*for* i *in* range(n):  
 *if not* discovered[i]:  
 # mark the source vertex as discovered  
 discovered[i] = *True* # enqueue source vertex  
 q.append(i)  
  
 # start BFS traversal from vertex i  
 recursiveBFS(graph, q, discovered)

Python Code

*from* collections *import* defaultdict, deque  
  
*class* Graph:  
 directed = *True  
  
 def \_\_init\_\_*(self):  
 self.graph = defaultdict(list)  
  
 *def* addEdge(self, *u*, *v*):  
 self.graph[*u*].append(*v*)  
  
 *if not* self.directed:  
 self.graph[*v*].append(*u*)  
  
 *def* DFS(self, *v*, *d*, *visitSet* = *None*) -> bool:  
 visited = *visitSet or* set()  
 visited.add(*v*)  
 print(*v*,end=" ")  
  
 *if v* == *d*:  
 *return True  
  
 for* neighbour *in* self.graph[*v*]:  
 *if* neighbour *not in* visited:  
 *if* self.DFS(neighbour, *d*, visited):  
 *return True  
  
 return False  
  
 def* BFS(self, *s*, *d*):  
 visited = defaultdict(bool)  
 queue = deque([*s*])  
 visited[*s*] = *True  
  
 while* queue:  
 s = queue.popleft()  
 print (*s*, end = " ")  
 *if s* == *d*:  
 *return  
 for* i *in* self.graph[*s*]:  
 *if* visited[i] == *False*:  
 queue.append(i)  
 visited[i] = *True*# A1.png  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 g = Graph()  
  
 g.addEdge('H', 'A')  
 g.addEdge('A', 'D')  
 g.addEdge('A', 'B')  
 g.addEdge('B', 'F')  
 g.addEdge('B', 'C')  
 g.addEdge('C', 'E')  
 g.addEdge('C', 'G')  
 g.addEdge('C', 'H')  
 g.addEdge('G', 'H')  
 g.addEdge('G', 'E')  
 g.addEdge('E', 'F')  
 g.addEdge('E', 'B')  
 g.addEdge('F', 'A')  
 g.addEdge('D', 'F')  
  
 print("Following is Depth First Traversal H -> E:")  
 g.DFS('H', 'E')  
  
 print ("\n\nFollowing is Breadth First Traversal H -> E:")  
 g.BFS('H', 'E')

Java Code :

import java.io.\*;

import java.util.\*;

class Graph {

private HashMap<String, LinkedList<String>> adj;

private boolean isDirected = true;

Graph() {

adj = new HashMap<String, LinkedList<String>>();

}

void addEdge(String v, String w) {

if (!adj.containsKey(v))

adj.put(v, new LinkedList<String>());

adj.get(v).add(w);

if (!isDirected) {

if (!adj.containsKey(w))

adj.put(w, new LinkedList<String>());

adj.get(w).add(v);

}

}

boolean DFS(String v, String d, HashSet<String> visitSet) {

HashSet<String> visited = visitSet == null ? new HashSet<String>() : visitSet;

visited.add(v);

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

if (v.equals(d)) {

return true;

}

Iterator<String> i = adj.get(v).listIterator();

while (i.hasNext()) {

String n = i.next();

if (!visited.contains(n))

if (DFS(n, d, visited))

return true;

}

return false;

}

void BFS(String s, String d) {

HashSet<String> visited = new HashSet<String>();

LinkedList<String> queue = new LinkedList<String>();

visited.add(s);

queue.add(s);

while (queue.size() != 0) {

s = queue.poll();

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

if (s.equals(d))

return;

Iterator<String> i = adj.get(s).listIterator();

while (i.hasNext()) {

String n = i.next();

if (!visited.contains(n)) {

visited.add(n);

queue.add(n);

}

}

}

}

// A1.png

public static void main(String args[]) {

Graph g = new Graph();

g.addEdge("H", "A");

g.addEdge("A", "D");

g.addEdge("A", "B");

g.addEdge("B", "F");

g.addEdge("B", "C");

g.addEdge("C", "E");

g.addEdge("C", "G");

g.addEdge("C", "H");

g.addEdge("G", "H");

g.addEdge("G", "E");

g.addEdge("E", "F");

g.addEdge("E", "B");

g.addEdge("F", "A");

g.addEdge("D", "F");

System.out.println("Following is Depth First Traversal H -> E:");

g.DFS("H", "E", null);

System.out.println("\n\nFollowing is Breadth First Traversal H -> E:");

g.BFS("H", "E");

}

}