```
/*________________________________________________________________________________
Assignment no :- 1
Title:- Implement Depth first search algorithm use an  undirected graph  and develop a recursive
algorithm for  searching all the vertices of a graph or tree
data structure

Name:-Pravin Jain          Roll No:-74        Batch:-T4          Subject:-AI
_______________________________________________________________________________*/
import java.util.*;

// A class to store a graph edge
class Edge{
   int source, dest;

   public Edge(int source, int dest)
   {
      this.source = source;
      this.dest = dest;
   }
   int getSource(){
      return this.source;
   }
   int getDest(){
      return this.dest;
   }
}

// A class to represent a graph object
class Graph{
   // A list of lists to represent an adjacency list
   List<List<Integer>> adjList = null;

   // Constructor
   Graph(List<Edge> edges, int n)
   {
      adjList = new ArrayList<>();

      for (int i = 0; i < n; i++) {
         adjList.add(new ArrayList<>());
      }

      // add edges to the undirected graph
      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_dfs){
      // mark the current node as discovered
      discovered_dfs[v] = true;

      // print the current node
      System.out.print((v+1) + " ");

      // do for every edge (v, u)
      for (int u: graph.adjList.get(v))
      {
         // if `u` is not yet discovered
         if (!discovered_dfs[u]) {
            DFS(graph, u, discovered_dfs);
         }
      }
   }

   public static void main(String[] args)
   {
      int sc;
      Scanner s = new Scanner(System.in);
      System.out.print("DFS Traversal Techniques :-");

               //Recursive DFS Algorithm

               List<Edge> edges_dfs = 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)
               );
               System.out.println("\nAdjacency List for DFS: ");
               for(int i = 0; i < edges_dfs.size(); i++) {
                  System.out.println(edges_dfs.get(i).getSource()+" -> "+edges_dfs.get(i).getDest());
               }
               System.out.println("");

               // total number of nodes in the graph (labelled from 1 to 13)
               int n_dfs = 13;

               // build a graph from the given edges
               Graph graph = new Graph(edges_dfs, n_dfs);

               // to keep track of whether a vertex is discovered or not
            boolean[] discovered_dfs = new boolean[n_dfs];

            // Perform DFS traversal from all undiscovered nodes to cover all connected components
of a graph
            for (int i = 0; i < n_dfs; i++)
            {
              if(i==0){
                System.out.println("DFS Starting from vertex "+(i+1)+" :");
              }
              if (!discovered_dfs[(i)]) {
                DFS(graph, i, discovered_dfs);
              }
            }

        }
    }
```

```
/*______________________________________________________________________________
Output:-
DFS Traversal Techniques :-
Adjacency List for DFS:
1 -> 2
1 -> 7
1 -> 8
2 -> 3
2 -> 6
3 -> 4
3 -> 5
8 -> 9
8 -> 12
9 -> 10
9 -> 11

DFS Starting from vertex 1 :
1 2 3 4 5 6 7 8 9 10 11 12 13
______________________________________________________________________________*/
```