```
/*______________________________________________________________________________
Assignment no :- 1
Title:- Implement Breath 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 BFS(Graph graphBFS, Queue<Integer> q, boolean[] discovered_bfs){
      if (q.isEmpty()) {
         return;
      }

      int v = q.poll();
      System.out.print((v+1) + " ");

      // do for every edge (v, u)
      for (int u: graphBFS.adjList.get(v))
      {
         if (!discovered_bfs[u])
         {
            // mark it as discovered and enqueue it
            discovered_bfs[u] = true;
            q.add(u);
         }
      }
      BFS(graphBFS, q, discovered_bfs);
   }



   public static void main(String[] args)
   {
      int sc;
      Scanner s = new Scanner(System.in);

      // List of graph edges as per the above diagram
      List<Edge> edges_BFS = 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)
            // vertex 0, 13, and 14 are single nodes
      );
      System.out.println("\nAdjacency List for BFS: ");
      for(int i = 0; i < edges_BFS.size(); i++) {
         System.out.println(edges_BFS.get(i).getSource()+" -> "+edges_BFS.get(i).getDest());
      }
      System.out.println("");

      // total number of nodes in the graph (labelled from 1 to 15)
      int n = 15;
```

```
        // build a graph from the given edges
        Graph graphBFS = new Graph(edges_BFS, n);

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

        // create a queue for doing BFS
        Queue<Integer> q = new ArrayDeque<>();

        // Perform BFS traversal from all undiscovered nodes to cover all connected components
of a graph
        for (int i = 0; i < n; i++)
        {
            if(i==0){
                System.out.println("BFS Starting from vertex "+(i+1)+" :");
            }
            if (!discovered[i])
            {
                // mark the source vertex as discovered
                discovered[i] = true;

                // enqueue source vertex
                q.add(i);

                // start BFS traversal from vertex `i`
                BFS(graphBFS, q, discovered);
            }
        }
    }
}

/*_____________________________________________________________________________
Output:-
Adjacency List for BFS:
1 -> 2
1 -> 3
1 -> 4
2 -> 5
2 -> 6
5 -> 9
5 -> 10
4 -> 7
4 -> 8
7 -> 11
7 -> 12

BFS Starting from vertex 1 :
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
_____________________________________________________________________________*/
```