```
/*******************************************************************************

Problem Statement - 1
_____________________________
Given an undirected graph with n nodes labeled from 0 to n-1 and a list of edges, determine
if there is a path between two given nodes start and end.
Write a function that returns true if a path exists between start and end, otherwise return
false.

Example 1:
n = 6
edges = {(0,1), (0,2), (3,5), (5,4), (4,3)}
start = 0
end = 5
*******************************************************************************/
import java.util.*;

public class Main
{
   private static List<Integer>[] graph;

   private static boolean dfs(int st, int end, boolean[] vis){

      if(st == end){
         return true;
      }

      vis[st] = true;

      for(int d : graph[st]){

         if(!vis[d]){
            if(dfs(d,end,vis))
               return true;
         }
      }

      vis[st] = false;
      return false;
   }

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

                  int n = sc.nextInt();
                  int m = sc.nextInt();

                  graph = new ArrayList[n];
```

```
            for(int i = 0; i < n; ++i){
                graph[i] = new ArrayList<>();
            }

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

                int u = sc.nextInt();
                int v = sc.nextInt();

                graph[u].add(v);
                graph[v].add(u);
            }

            int start = sc.nextInt();
            int end = sc.nextInt();

            boolean[] vis = new boolean[n];

            boolean ans = dfs(start,end,vis);

            System.out.println(ans);
        }
}
```

```
/*****************************************************************************
```

Problem Statement - 2

Given an undirected graph with n nodes labeled from 0 to n-1, a list of edges, a starting node start, and an integer k, return all nodes that are exactly k edges away from the starting node. The result can be returned in any order.
Write a function that returns the list of such nodes.

Example 1:

n = 6
edges = {(0,1), (0,2), (1,3), (2,4), (2,5)}
start = 0
k = 2

Output: [3,4,5]

```
*****************************************************************************/
import java.util.*;

public class Main
```

```
{
    private static List<Integer>[] graph;
    private static List<Integer> ans;

    private static void dfs(int st, int k, boolean[] vis){

        if(k == 0){
            ans.add(st);
            return;
        }

        vis[st] = true;

        for(int d : graph[st]){

            if(!vis[d]){
                dfs(d,k-1,vis);
            }
        }

        vis[st] = false;
    }

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

                    int n = sc.nextInt();
                    int m = sc.nextInt();

                    graph = new ArrayList[n];

                    for(int i = 0; i < n; ++i){
                        graph[i] = new ArrayList<>();
                    }

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

                        int u = sc.nextInt();
                        int v = sc.nextInt();

                        graph[u].add(v);
                        graph[v].add(u);
                    }

                    int start = sc.nextInt();
                    int k = sc.nextInt();

                    boolean[] vis = new boolean[n];
```

```
                ans = new ArrayList<>();

                dfs(start,k,vis);

                System.out.println(ans);
        }
}
```