Task 3: Union-Find for Cycle Detection

Write a Union-Find data structure with path compression. Use this data structure to detect a cycle in an undirected graph.

```java
package Day9_10;

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;
 }
}

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

     // add edges to the undirected graph (add each edge once only to avoid
     // detecting cycles among the same edges, say x -> y and y -> x)
     for (Edge edge: edges) {
         adjList.get(edge.source).add(edge.dest);
     }
 }
}

//A class to represent a disjoint set
class DisjointSet
{
 private Map<Integer, Integer> parent = new HashMap<>();

 // perform MakeSet operation
 public void makeSet(int n)
 {
     // create `n` disjoint sets (one for each vertex)
     for (int i = 0; i < n; i++) {
         parent.put(i, i);
     }
 }

 // Find the root of the set in which element `k` belongs
 public int find(int k)
```

```
 {
     // if `k` is root
     if (parent.get(k) == k) {
         return k;
     }

     // recur for the parent until we find the root
     return find(parent.get(k));
 }

 // Perform Union of two subsets
 public void union(int a, int b)
 {
     // find the root of the sets in which elements `x` and `y` belongs
     int x = find(a);
     int y = find(b);

     parent.put(x, y);
 }
}

class UnionFind
{
 // Returns true if the graph has a cycle
 public static boolean findCycle(Graph graph, int n)
 {
     // initialize `DisjointSet` class
     DisjointSet ds = new DisjointSet();

     // create a singleton set for each element of the universe
     ds.makeSet(n);

     // consider every edge (u, v)
     for (int u = 0; u < n; u++)
     {
         // Recur for all adjacent vertices
         for (int v: graph.adjList.get(u))
         {
             // find the root of the sets to which elements `u` and `v` belongs
             int x = ds.find(u);
             int y = ds.find(v);

             // if both `u` and `v` have the same parent, the cycle is found
             if (x == y) {
                 return true;
             }
             else {
                 ds.union(x, y);
             }
         }
     }

     return false;
 }

 // Union-find algorithm for cycle detection in a graph
 public static void main(String[] args)
 {
     // List of graph edges
```

```
    List<Edge> edges = Arrays.asList(
                        new Edge(0, 1), new Edge(0, 6), new Edge(0, 7),
                        new Edge(1, 2), new Edge(1, 5), new Edge(2, 3),
                        new Edge(2, 4), new Edge(7, 8), new Edge(7, 11),
                        new Edge(8, 9), new Edge(8, 10), new Edge(10, 11)
                        // edge (10, 11) introduces a cycle in the graph
                    );

    // total number of nodes in the graph (labelled from 0 to 11)
    int n = 12;

    // construct graph
    Graph graph = new Graph(edges, n);

    if (findCycle(graph, n)) {
        System.out.println("Cycle Found");
    }
    else {
        System.out.println("No Cycle is Found");
    }
 }
}
```

```
107     return false;
108 }
109
110 // Union-find algorithm for cycle detection in a graph
111 public static void main(String[] args)
112 {
113     // List of graph edges
114     List<Edge> edges = Arrays.asList(
115                         new Edge(0, 1), new Edge(0, 6), new Edge(0, 7),
116                         new Edge(1, 2), new Edge(1, 5), new Edge(2, 3),
117                         new Edge(2, 4), new Edge(7, 8), new Edge(7, 11),
118                         new Edge(8, 9), new Edge(8, 10), new Edge(10, 11)
119                         // edge (10, 11) introduces a cycle in the graph
120                     );
121
122     // total number of nodes in the graph (labelled from 0 to 11)
123     int n = 12;
124
125     // construct graph
126     Graph graph = new Graph(edges, n);
127
128     if (findCycle(graph, n)) {
129         System.out.println("Cycle Found");
130     }
131     else {
132         System.out.println("No Cycle is Found");
133     }
134 }
135 }
```

```
Day9_10
Edge
    source : int
    dest : int
    Edge(int, int)
Graph
    adjList : List<List<Integer>>
    Graph(List<Edge>, int)
DisjointSet
    parent : Map<Integer, Integer>
    makeSet(int) : void
    find(int) : int
    union(int, int) : void
UnionFind
    findCycle(Graph, int) : boolean
    main(String[]) : void
```

```
Markers  Properties  Terminal  Console  Coverage
<terminated> UnionFind [Java Application] C:\Users\Nikita\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20210721-1149\jre\bin\javaw.exe  (Jun 4, 2024, 4:21:18 PM – 4:21:
Cycle Found
```