Day 15

Assignment: 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.

A)

Introduction*:*

*The code showcases a practical application of the Union-Find data structure in graph theory. By efficiently detecting cycles in an undirected graph, it addresses a common problem encountered in various fields such as computer science, network analysis, and data structures. With its modular structure and clear implementation, the code provides an accessible solution for cycle detection, demonstrating the effectiveness of Union-Find in algorithmic problems involving graphs***.**

Java code :

```
import java.util.*;

public class UnionFindCycleDetection {

  class UnionFind {
    private int[] parent, rank;


    public UnionFind(int size) {
      parent = new int[size];
      rank = new int[size];
      for (int i = 0; i < size; i++) {
        parent[i] = i;
        rank[i] = 0;
      }
    }


    public int find(int p) {
      if (parent[p] != p) {
        parent[p] = find(parent[p]);
      }
      return parent[p];
    }


    public boolean union(int p, int q) {
      int rootP = find(p);
      int rootQ = find(q);
```

```
        if (rootP == rootQ) {
            return false;
        }

        if (rank[rootP] > rank[rootQ]) {
            parent[rootQ] = rootP;
        } else if (rank[rootP] < rank[rootQ]) {
            parent[rootP] = rootQ;
        } else {
            parent[rootQ] = rootP;
            rank[rootP]++;
        }

        return true;
        }
}

class Graph {
    private int numVertices;
    private List<int[]> edges;


    public Graph(int numVertices) {
        this.numVertices = numVertices;
        edges = new ArrayList<>();
    }


    public void addEdge(int v, int w) {
        edges.add(new int[]{v, w});
    }


    public boolean hasCycle() {
        UnionFind uf = new UnionFind(numVertices);

        for (int[] edge : edges) {
            int u = edge[0];
            int v = edge[1];

            if (!uf.union(u, v)) {
                return true;
            }
        }

        return false;
    }
    }

    public static void main(String[] args) {
        UnionFindCycleDetection ufc = new UnionFindCycleDetection();
        Graph graph = ufc.new Graph(6);

        graph.addEdge(0, 1);
        graph.addEdge(1, 2);
        graph.addEdge(2, 3);
        graph.addEdge(3, 4);
        graph.addEdge(4, 5);
        graph.addEdge(1, 3);

        if (graph.hasCycle()) {
            System.out.println("Graph contains cycle");
        } else {
            System.out.println("Graph doesn't contain cycle");
        }
    }
}
```

Summary:

The code demonstrates how to use the Union-Find data structure to detect cycles in an undirected graph efficiently. It provides implementations for both Union-Find and the graph data structure, allowing you to easily create a graph, add edges, and check for cycles. Overall, it's a well-organized and concise solution for cycle detection in graphs using Union-Find.