DAY 15:

ASSIGNMENT 1:

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.

ANSWER:

```
package day15;

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");
        }
    }
}
```

EXPLANATION:

The Union-Find data structure, also known as Disjoint Set Union (DSU), is an efficient way to manage a partition of a set into disjoint (non-overlapping) subsets. It supports two main operations:

1. *Find*: Determine which subset a particular element is in.

2. *Union*: Join two subsets into a single subset.

Key Concepts

1. Graph Representation:

- The graph is represented by a list of edges. Each edge connects two vertices.

- For undirected graphs, each edge is bidirectional.

2. Union-Find Structure:

- Parent Array: Keeps track of the parent or representative of each subset.

- Rank Array: Helps in keeping the tree flat by storing the depth of each tree.

3. Find Operation with Path Compression:

- The find operation determines the root or representative of the subset containing a particular element.

- Path compression is used to make the tree flatter, ensuring that every node points directly to the root. This optimizes future operations.

4. Union Operation with Union by Rank:

- The union operation merges two subsets into one.

- Union by rank ensures that the smaller tree (in terms of depth) is always attached under the root of the larger tree, keeping the overall tree flat.

Steps for Cycle Detection

1. Initialize Union-Find Structure:

- Create an instance of the Union-Find structure with the same number of elements as there are vertices in the graph.

2. Process Each Edge:

- For each edge in the graph, perform the following steps:

- Use the find operation to determine the roots of the subsets containing the two vertices of the edge.

- If both vertices belong to the same subset (i.e., their roots are the same), a cycle is detected, as this indicates that there is already a path connecting these two vertices.

- If the vertices belong to different subsets, perform the union operation to merge the two subsets into one.

3. Cycle Detection Result:

- If a cycle is detected during the edge processing, the algorithm returns true, indicating that the graph contains a cycle.

- If no cycles are detected after processing all edges, the algorithm returns false, indicating that the graph does not contain any cycles.

Example Scenario

Imagine a graph with 6 vertices and edges connecting them as follows:

- Edges: (0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (1, 3)

Steps:

- Initialize the Union-Find structure for 6 vertices.

- Add edges one by one and check if they form a cycle:

- Adding (0, 1): No cycle.

- Adding (1, 2): No cycle.

- Adding (2, 3): No cycle.

- Adding (3, 4): No cycle.

- Adding (4, 5): No cycle.

- Adding (1, 3): Cycle detected, as vertices 1 and 3 are already connected indirectly through vertices 2 and 3.

Thus, the graph contains a cycle.

By using the Union-Find data structure with path compression and union by rank, we efficiently manage and merge subsets, making the cycle detection process both quick and effective. This method is particularly useful for large graphs where performance and scalability are critical.