Course Name:Competitive Programming

Name:Swetha Dixit

Batch:33

Hall-Ticket:2303A52415

Assignment:9

Week:5

Community Detection in a Social Networking Platform
You are developing a social networking platform where users can form friendships. If two users are directly or indirectly connected through friendships, they belong to the same community.
Initially, each user is independent. As friendships are formed, communities merge.
Your task is to determine whether two users belong to the same community after processing friendship relationships.
To efficiently manage and query user connections, use the Disjoint Set Union (Union–Find) data structure.
Task
1. Initialize each user as a separate set.
2. Process friendship connections using Union operations.
3. Answer queries to check if two users are in the same community.
Operations
Union(u, v): Create a friendship between users u and v
Find(u): Identify the community representative of user u
Connected(u, v): Check if users u and v belong to the same community
Example Test Case 1
Input
Number of users: 7
Number of friendships: 5
23th Feb,
2026,
5:00PM
Friendships:
(0,1)
(1,3)
(2,4)
(5,6)

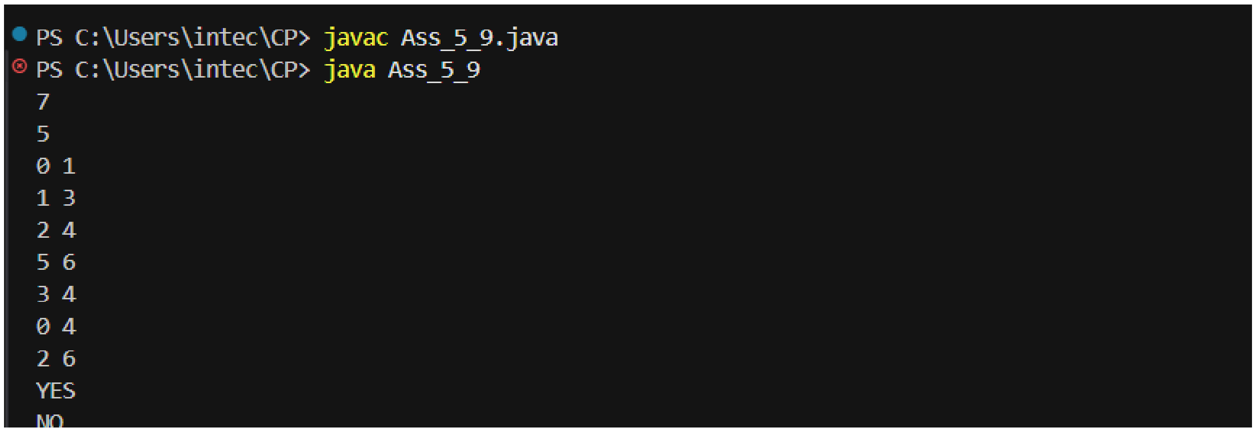
(3,4)
Queries:
(0,4)
(2,6)
(5,6)
Output
YES
NO
YES
Wednesday City Road Connectivity Management
You are managing the road network of a city. The city has multiple intersections, and roads are constructed between them over time. If two intersections are connected directly or indirectly, they belong to the same road network zone.
Initially, no roads exist. As new roads are added, you must determine whether two intersections are in the same zone.
To efficiently handle multiple connectivity checks, use the Disjoint Set Union (Union–Find) data structure.
Task
Write a program to:
1. Initialize each intersection as a separate set.
2. Merge intersections when a road is constructed.
3. Answer connectivity queries using DSU.
Example Test Case 1
Input
Number of intersections: 8
Number of roads constructed: 5
Roads:
(1,2)
(2,3)
(4,5)
(6,7)
(5,6)
Connectivity queries:
(1,3)
(1,7)
(4,7)
Output
YES
NO
YES
Python code:

```python
class DSU:
    def __init__(self, n):
        # Initialize parent of i as i itself
        self.parent = list(range(n))

    def find(self, i):
        # Path Compression: Point directly to the root
        if self.parent[i] != i:
            self.parent[i] = self.find(self.parent[i])
        return self.parent[i]

    def union(self, i, j):
        root_a = self.find(i)
        root_b = self.find(j)
        if root_a != root_b:
            self.parent[root_a] = root_b

    def is_connected(self, i, j):
        return self.find(i) == self.find(j)

# --- Main Execution ---
def solve():
    number_of_users = 7
    dsu = DSU(number_of_users)

    # Friendships (0-indexed)
    friendships = [
        (0, 1),
        (1, 3),
        (2, 4),
        (5, 6),
        (3, 4)
    ]

    for u, v in friendships:
        dsu.union(u, v)

    # Queries
    queries = [
        (0, 4),
        (2, 6),
        (5, 6)
    ]

    print("Query Results:")
    for u, v in queries:
        if dsu.is_connected(u, v):
            print("YES")
        else:
            print("NO")

if __name__ == "__main__":
    solve()
```

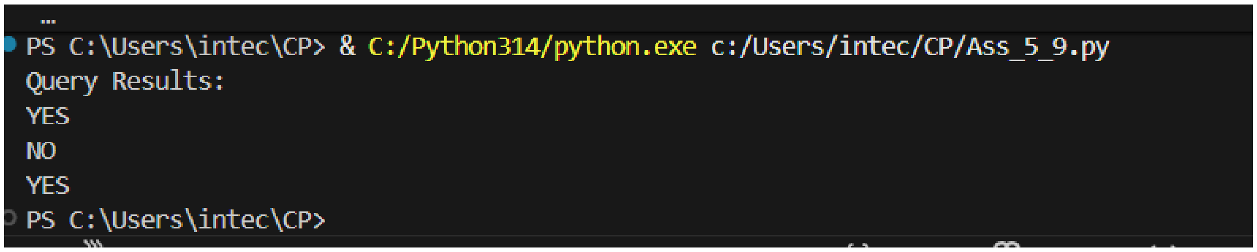
Java code:

```
import java.util.*;

public class Ass_5_9 {

    // Disjoint Set Union (DSU) / Union-Find Data Structure
    static class DSU {
        int[] parent;

        // Initialize: Each user is their own parent (independent set)
        public DSU(int n) {
            parent = new int[n];
            for (int i = 0; i < n; i++) {
                parent[i] = i;
            }
        }

        // FIND Operation: Returns the representative (root) of the set
        // Uses "Path Compression" optimization
        public int find(int i) {
            if (parent[i] == i) {
                return i;
            }
            // Recursively find the root and point i directly to it
(flatten the tree)
            parent[i] = find(parent[i]);
            return parent[i];
        }

        // UNION Operation: Merges two sets
        public void union(int i, int j) {
            int rootA = find(i);
            int rootB = find(j);
```

```java
            if (rootA != rootB) {
                // Connect one root to the other
                parent[rootA] = rootB;
            }
        }

        // CONNECTED Operation: Checks if two users are in the same
community
        public boolean isConnected(int i, int j) {
            return find(i) == find(j);
        }
    }

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

        // Check if input exists to avoid crashes
        if (sc.hasNextInt()) {

            // 1. Read Number of Users (N)
            int N = sc.nextInt();

            // Initialize DSU with N users (0 to N-1)
            DSU dsu = new DSU(N);

            // 2. Read Number of Friendships (M)
            int M = sc.nextInt();

            // Process M friendships
            for (int i = 0; i < M; i++) {
                int u = sc.nextInt();
                int v = sc.nextInt();
                dsu.union(u, v);
            }

            // 3. Process Queries
            // The problem statement didn't specify the number of queries
(Q),
            // so we read until there are no more integers (End of File).
            // If your input has a specific Q count, uncomment the next
line:
            // int Q = sc.nextInt(); while(Q-- > 0) {

            while (sc.hasNextInt()) {
                int u = sc.nextInt();
                int v = sc.nextInt();

                if (dsu.isConnected(u, v)) {
                    System.out.println("YES");
                } else {
                    System.out.println("NO");
                }
            }
        }
        sc.close();
    }
}
```
Output:



```
PS C:\Users\intec\CP> javac Ass_5_9.java
PS C:\Users\intec\CP> java Ass_5_9
7
5
0 1
1 3
2 4
5 6
3 4
0 4
2 6
YES
NO
```



```
...
PS C:\Users\intec\CP> & C:/Python314/python.exe c:/Users/intec/CP/Ass_5_9.py
Query Results:
YES
NO
YES
PS C:\Users\intec\CP>
```