Institute Of Technology, Nirma university



BRANCH :- **Computer Science Engineering**

**PRACTICAL SUBMISSION**

**|\*|***STUDENT INFO***|\*|**

Name :- **Pratik Kansara**

Roll No. :- **20BCE510**

Division :- **E4**

**|\*|***SUBJECT INFO***|\*|**

Subject :- **Advanced Data Structures** Practical No. :- **8**

**Practical – 8**

**AIM :- Write a program to implement union-find structure. The program should demonstrate the structure representation of set and list the items of selected set.**

**Code:**

**Unionfind.java**

public class Unionfind {

    // The number of elements in this union find

    private int size;

    // Used to track the size of each of the component

    private int[] sz;

    // id[i] points to the parent of i, if id[i] = i then i is a root node

    private int[] id;

    // Tracks the number of components in the union find

    private int numComponents;

    public Unionfind(int size) {

        if (size <= 0)

            throw new IllegalArgumentException("Size <= 0 is not allowed");

        this.size = numComponents = size;

        sz = new int[size];

        id = new int[size];

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

            id[i] = i; // Link to itself (self root)

            sz[i] = 1; // Each component is originally of size one

        }

    }

    // Find which component/set 'p' belongs to, takes amortized constant time.

    public int find(int p) {

        // Find the root of the component/set

        int root = p;

        while (root != id[root])

            root = id[root];

        // Compress the path leading back to the root.

        // Doing this operation is called "path compression"

        // and is what gives us amortized time complexity.

        while (p != root) {

            int next = id[p];

            id[p] = root;

            p = next;

        }

        return root;

    }

    // Return whether or not the elements 'p' and

    // 'q' are in the same components/set.

    public boolean connected(int p, int q) {

        return find(p) == find(q);

    }

    // Return the size of the components/set 'p' belongs to

    public int componentSize(int p) {

        return sz[find(p)];

    }

    // Return the number of elements in this UnionFind/Disjoint set

    public int size() {

        return size;

    }

    // Returns the number of remaining components/sets

    public int components() {

        return numComponents;

    }

    // Unify the components/sets containing elements 'p' and 'q'

    public void unify(int p, int q) {

        // These elements are already in the same group!

        if (connected(p, q))

            return;

        int root1 = find(p);

        int root2 = find(q);

        // Merge smaller component/set into the larger one.

        if (sz[root1] < sz[root2]) {

            sz[root2] += sz[root1];

            id[root1] = root2;

            sz[root1] = 0;

        } else {

            sz[root1] += sz[root2];

            id[root2] = root1;

            sz[root2] = 0;

        }

        // Since the roots found are different we know that the

        // number of components/sets has decreased by one

        numComponents--;

    }

    public static void main(String[] args) {

    }

}

**OUTPUT**