Date: 28/04/2021

Practical 8:

2CSDE75 - Advanced Data Structures

Name: Shrey Viradiya

Roll No: 18BCE259

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:

Prac8_UnionFind.cpp

```cpp
#include "UnionFind.h"
#include <iostream>

int main(){
    using namespace std;
    UnionFind UF = UnionFind(10);

    // UF.printStructureArray();
    UF.printSet(1);

    cout << "\nJoining Element 1 and 2" << endl;
    UF.join(1,2);
    UF.printSet(1);
    // UF.printStructureArray();

    cout << "\nJoining Element 3 and 4" << endl;
    UF.join(3,4);
    UF.printSet(3);
    // UF.printStructureArray();

    cout << "\nJoining Element 4 and 2" << endl;
    UF.join(4,2);
    UF.printSet(1);
    // UF.printStructureArray();

    return 0;
}
```

UnionFind.h

```cpp
#pragma once
#include<iostream>
#include<utility>

class UnionFind
{
private:
    int elements;
    int *UnionFindArray;
public:
    UnionFind() = delete;
    UnionFind(int elements);
    ~UnionFind();
    int root(int element);
```

```
    void join( int x, int y );
    void printStructureArray();
    void printSet(int x);
};

UnionFind::UnionFind(int elements)
{
    this->elements = elements;
    UnionFindArray = new int[elements];
    for (int i = 0; i < elements; i++)
        UnionFindArray[i] = -1;
}

UnionFind::~UnionFind()
{
    delete[]UnionFindArray;
}

int UnionFind::root(int element){
    int root = element;

    /* Find root */
    while(UnionFindArray[root] >= 0)
        root = UnionFindArray[root];

    /* Compress path to root */
    while(UnionFindArray[element] >= 0) {
        int tmp = UnionFindArray[element];
        UnionFindArray[element] = root;
        element = tmp;
    }

    return root;
}

void UnionFind::join( int x, int y ) {
    x = root(x);
    y = root(y);

    if(x != y) {
        if(UnionFindArray[x] < UnionFindArray[y]) {
            UnionFindArray[x] += UnionFindArray[y];
            UnionFindArray[y] = x;
        }
        else {
            UnionFindArray[y] += UnionFindArray[x];
            UnionFindArray[x] = y;
        }
```

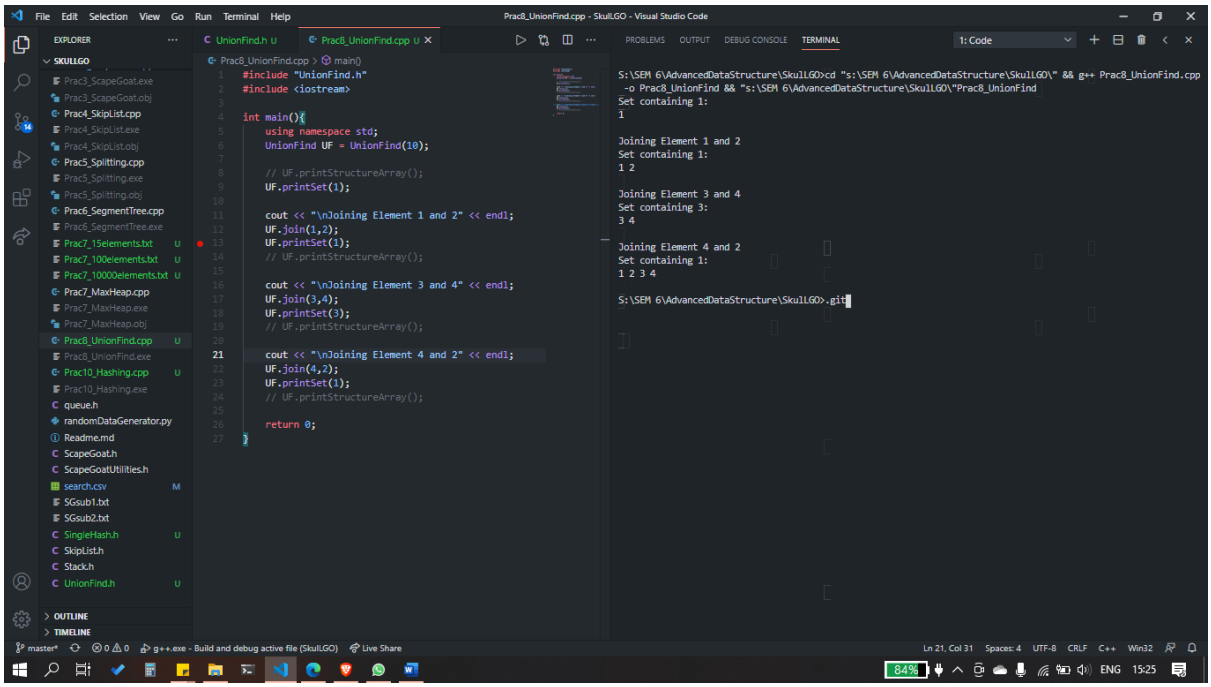
```
    }
}

void UnionFind::printStructureArray(){
    using namespace std;
    for (int i = 0; i < elements; i++)
        cout << UnionFindArray[i] << " ";
    cout << endl;
}

void UnionFind::printSet(int element){
    using namespace std;
    int r = root(element);
    cout << "Set containing " << element << ": " << endl;

    for (int i = 0; i < elements; i++)
        if(root(i) == r) cout << i << " ";
    cout << endl;
}
```

Snapshot of the output:



Conclusion:

Union Find structure is very important in problem solving. It is used in Kruskal’s Algorithm to find minimum spanning tree.