Name : Ramzy Izza Wardhana
Class : IUP CS 1
NIM : 21/472698/PA/20322

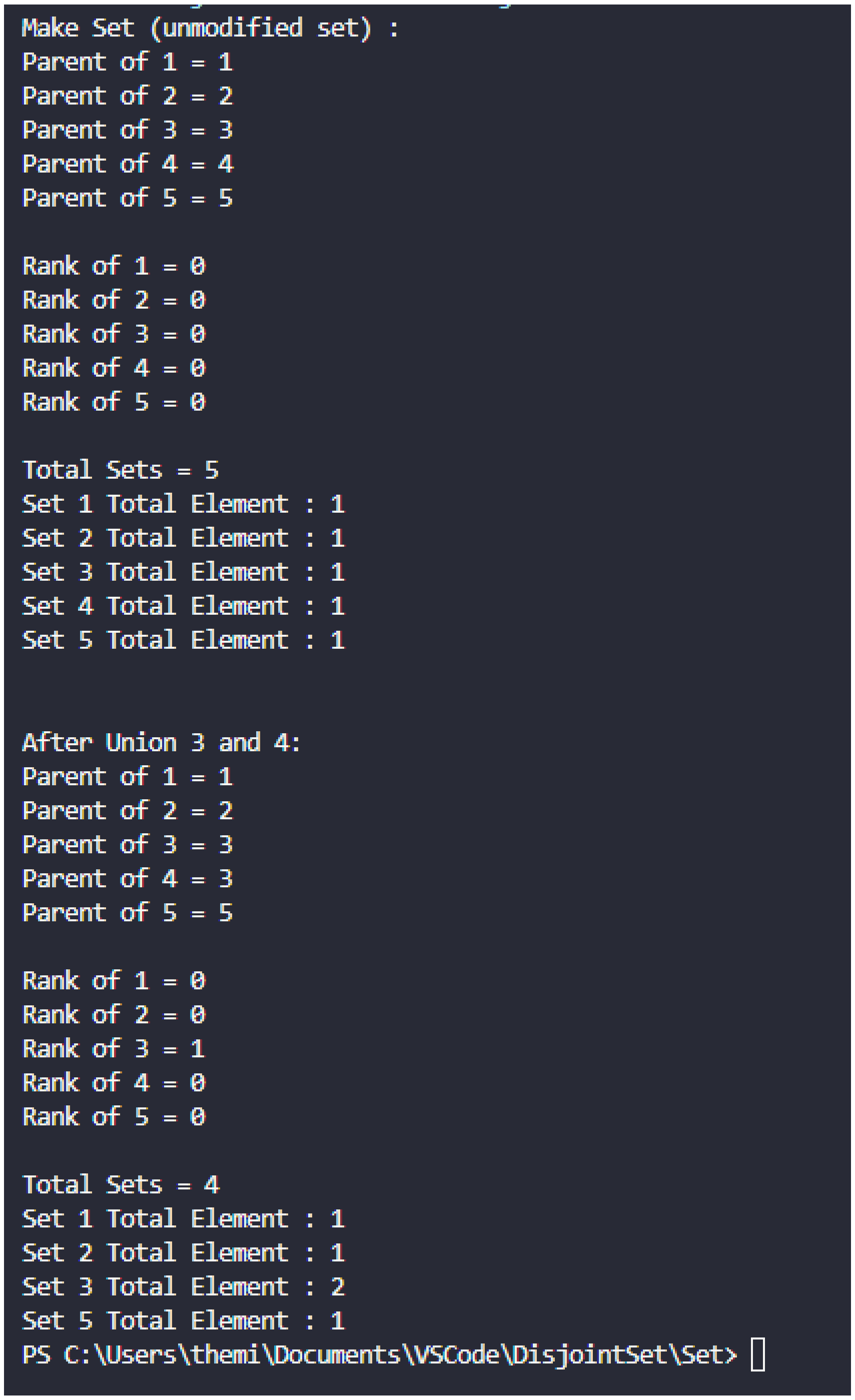
Assignment 9 – Disjoint Set

Lab Algorithm and Data Structures

Full Source Code in Zipped Format:

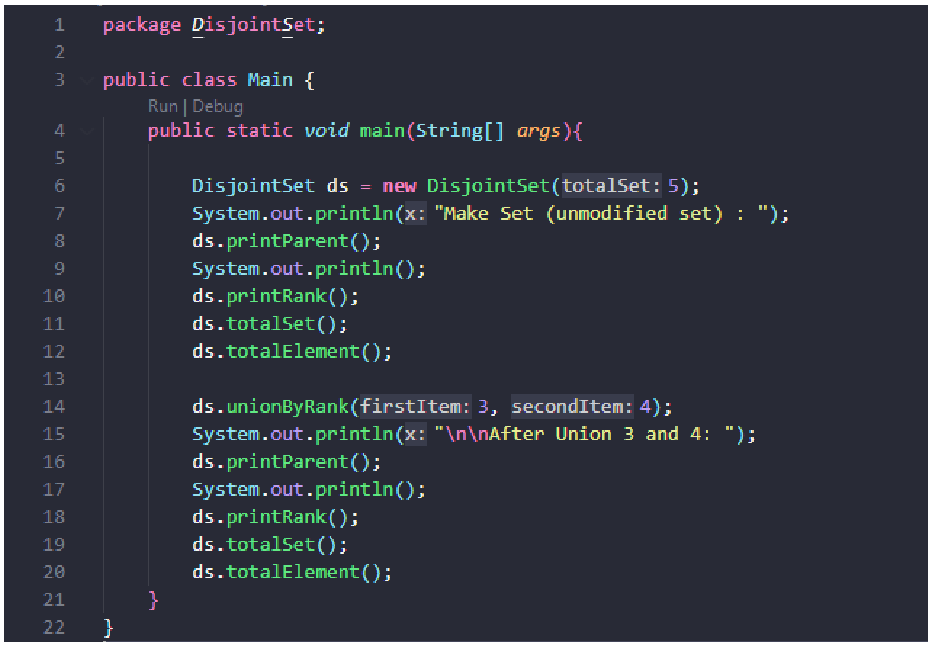
https://drive.google.com/file/d/17tTUDViK-fGPfqebv8P1KN_Ob-k5rnmd/view?usp=sharing

Output :



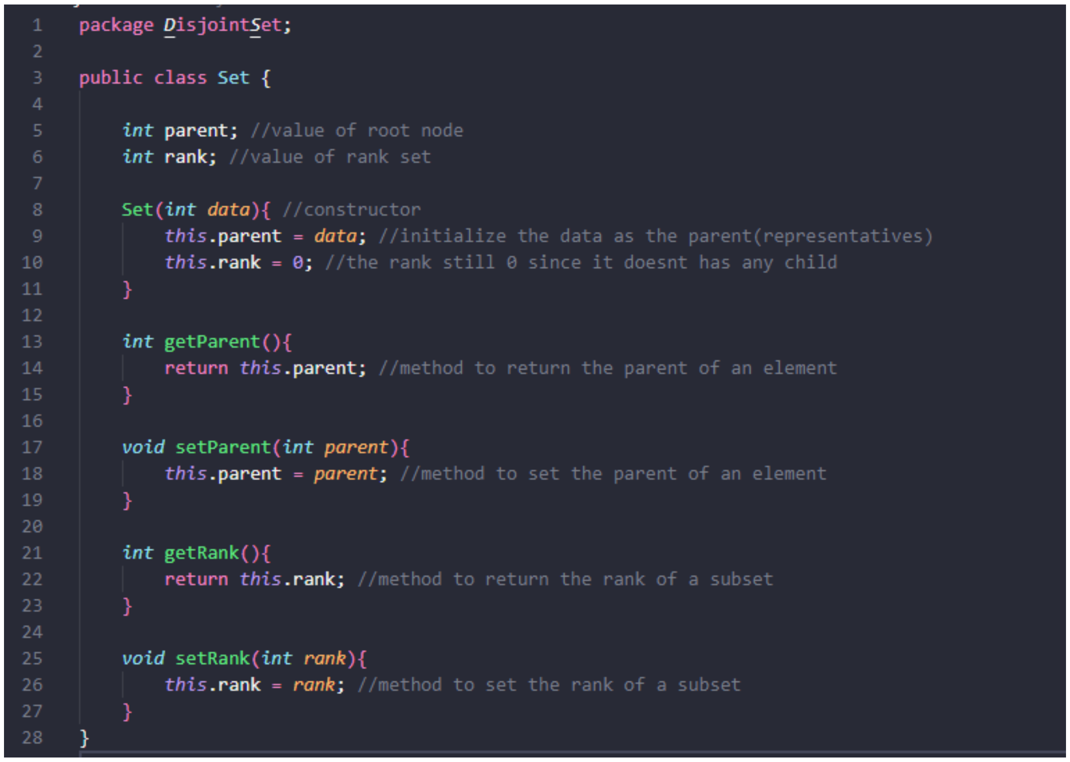
```
Make Set (unmodified set) :
Parent of 1 = 1
Parent of 2 = 2
Parent of 3 = 3
Parent of 4 = 4
Parent of 5 = 5

Rank of 1 = 0
Rank of 2 = 0
Rank of 3 = 0
Rank of 4 = 0
Rank of 5 = 0

Total Sets = 5
Set 1 Total Element : 1
Set 2 Total Element : 1
Set 3 Total Element : 1
Set 4 Total Element : 1
Set 5 Total Element : 1


After Union 3 and 4:
Parent of 1 = 1
Parent of 2 = 2
Parent of 3 = 3
Parent of 4 = 3
Parent of 5 = 5

Rank of 1 = 0
Rank of 2 = 0
Rank of 3 = 1
Rank of 4 = 0
Rank of 5 = 0

Total Sets = 4
Set 1 Total Element : 1
Set 2 Total Element : 1
Set 3 Total Element : 2
Set 5 Total Element : 1
PS C:\Users\themi\Documents\VSCode\DisjointSet\Set>
```

Main.java



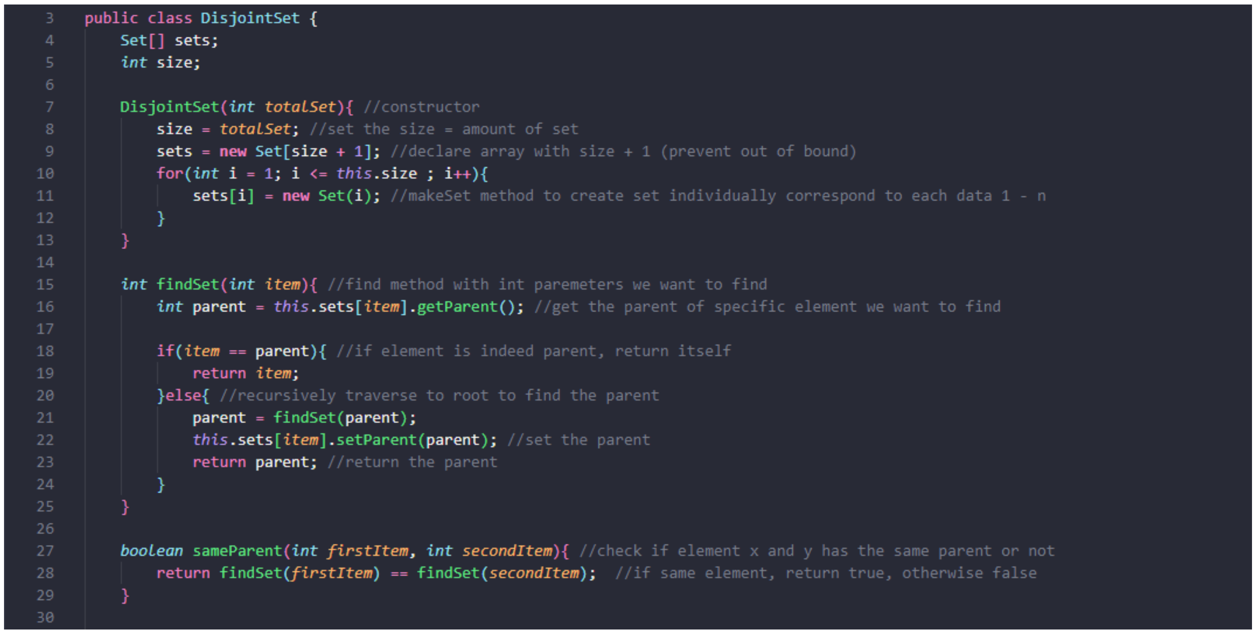
```java
package DisjointSet;

public class Main {
    Run | Debug
    public static void main(String[] args){

        DisjointSet ds = new DisjointSet(totalSet: 5);
        System.out.println(x: "Make Set (unmodified set) : ");
        ds.printParent();
        System.out.println();
        ds.printRank();
        ds.totalSet();
        ds.totalElement();

        ds.unionByRank(firstItem: 3, secondItem: 4);
        System.out.println(x: "\n\nAfter Union 3 and 4: ");
        ds.printParent();
        System.out.println();
        ds.printRank();
        ds.totalSet();
        ds.totalElement();
    }
}
```

Set.java

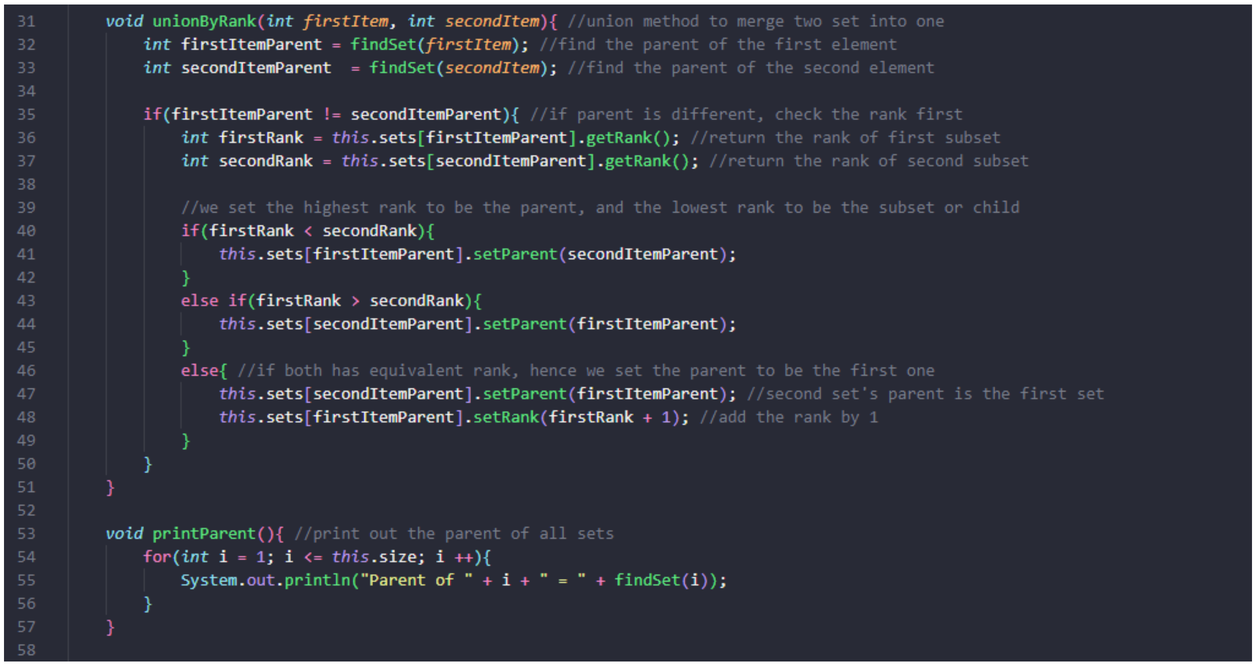


```java
package DisjointSet;

public class Set {

    int parent; //value of root node
    int rank; //value of rank set

    Set(int data){ //constructor
        this.parent = data; //initialize the data as the parent(representatives)
        this.rank = 0; //the rank still 0 since it doesnt has any child
    }

    int getParent(){
        return this.parent; //method to return the parent of an element
    }

    void setParent(int parent){
        this.parent = parent; //method to set the parent of an element
    }

    int getRank(){
        return this.rank; //method to return the rank of a subset
    }

    void setRank(int rank){
        this.rank = rank; //method to set the rank of a subset
    }
}
```

Disjoint.java

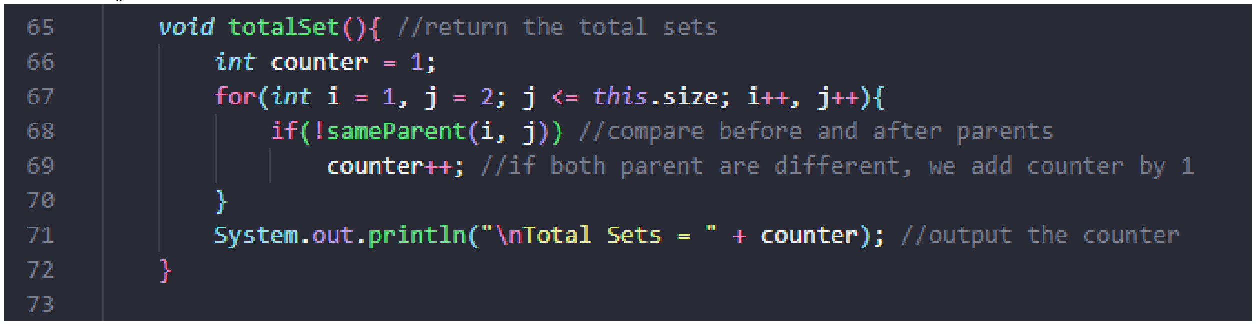


```java
public class DisjointSet {
    Set[] sets;
    int size;

    DisjointSet(int totalSet){ //constructor
        size = totalSet; //set the size = amount of set
        sets = new Set[size + 1]; //declare array with size + 1 (prevent out of bound)
        for(int i = 1; i <= this.size ; i++){
            sets[i] = new Set(i); //makeSet method to create set individually correspond to each data 1 - n
        }
    }

    int findSet(int item){ //find method with int paremeters we want to find
        int parent = this.sets[item].getParent(); //get the parent of specific element we want to find

        if(item == parent){ //if element is indeed parent, return itself
            return item;
        }else{ //recursively traverse to root to find the parent
            parent = findSet(parent);
            this.sets[item].setParent(parent); //set the parent
            return parent; //return the parent
        }
    }

    boolean sameParent(int firstItem, int secondItem){ //check if element x and y has the same parent or not
        return findSet(firstItem) == findSet(secondItem);  //if same element, return true, otherwise false
    }

```



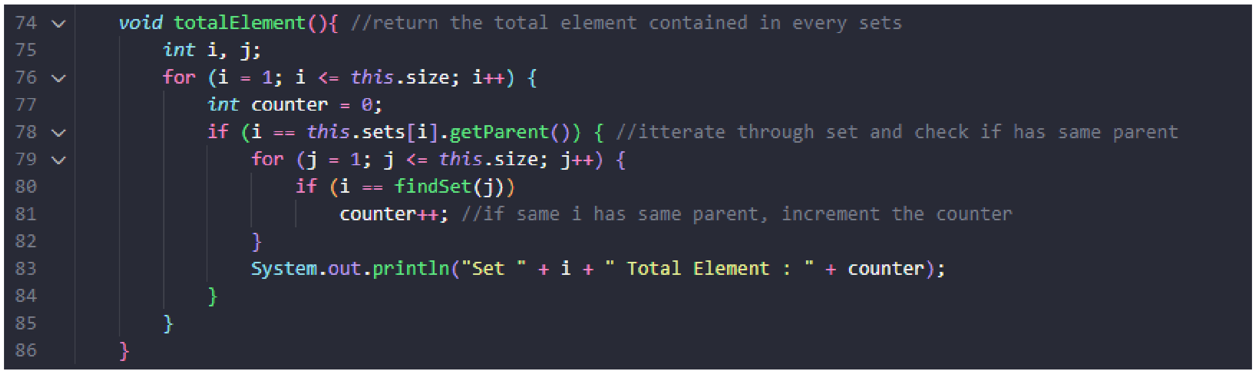
```java
    void unionByRank(int firstItem, int secondItem){ //union method to merge two set into one
        int firstItemParent = findSet(firstItem); //find the parent of the first element
        int secondItemParent  = findSet(secondItem); //find the parent of the second element

        if(firstItemParent != secondItemParent){ //if parent is different, check the rank first
            int firstRank = this.sets[firstItemParent].getRank(); //return the rank of first subset
            int secondRank = this.sets[secondItemParent].getRank(); //return the rank of second subset

            //we set the highest rank to be the parent, and the lowest rank to be the subset or child
            if(firstRank < secondRank){
                this.sets[firstItemParent].setParent(secondItemParent);
            }
            else if(firstRank > secondRank){
                this.sets[secondItemParent].setParent(firstItemParent);
            }
            else{ //if both has equivalent rank, hence we set the parent to be the first one
                this.sets[secondItemParent].setParent(firstItemParent); //second set's parent is the first set
                this.sets[firstItemParent].setRank(firstRank + 1); //add the rank by 1
            }
        }
    }

    void printParent(){ //print out the parent of all sets
        for(int i = 1; i <= this.size; i ++){
            System.out.println("Parent of " + i + " = " + findSet(i));
        }
    }

```

totalSet() Method



```
    void totalSet(){ //return the total sets
        int counter = 1;
        for(int i = 1, j = 2; j <= this.size; i++, j++){
            if(!sameParent(i, j)) //compare before and after parents
                counter++; //if both parent are different, we add counter by 1
        }
        System.out.println("\nTotal Sets = " + counter); //output the counter
    }

```

totalElement() Method



```
    void totalElement(){ //return the total element contained in every sets
        int i, j;
        for (i = 1; i <= this.size; i++) {
            int counter = 0;
            if (i == this.sets[i].getParent()) { //itterate through set and check if has same parent
                for (j = 1; j <= this.size; j++) {
                    if (i == findSet(j))
                        counter++; //if same i has same parent, increment the counter
                }
                System.out.println("Set " + i + " Total Element : " + counter);
            }
        }
    }
```