Program Structures and Algorithms
Spring 2022

Name: Sagar Jalindar Khandagale
NUID: 002197761
Assignment No: 3 (Union Find)

Tasks:

1. Implement height-weighted Quick Union with Path Compression. Check all unit test cases.
2. Develop a UF ("union-find") client that takes an integer value n from the command line to determine the number of "sites." Then generates random pairs of integers between 0 and n-1, calling connected() to determine if they are connected and union() if not. Loop until all sites are connected then print the number of connections generated.
3. Determine the relationship between the number of objects (*n*) and the number of pairs (*m*) generated

Step 1:

Implemented required changes in find() method:

```
/**
 * Returns the component identifier for the component containing site {@code p}.
 *
 * @param p the integer representing one site
 * @return the component identifier for the component containing site {@code p}
 * @throws IllegalArgumentException unless {@code 0 <= p < n}
 */
public int find(int p) {
    validate(p);
    int root = p;
    // FIXME
    while (root != parent[root]) {
        if (pathCompression) {
            doPathCompression(root);
        }
        root = parent[root];
    }
    // END
    return root;
}
```

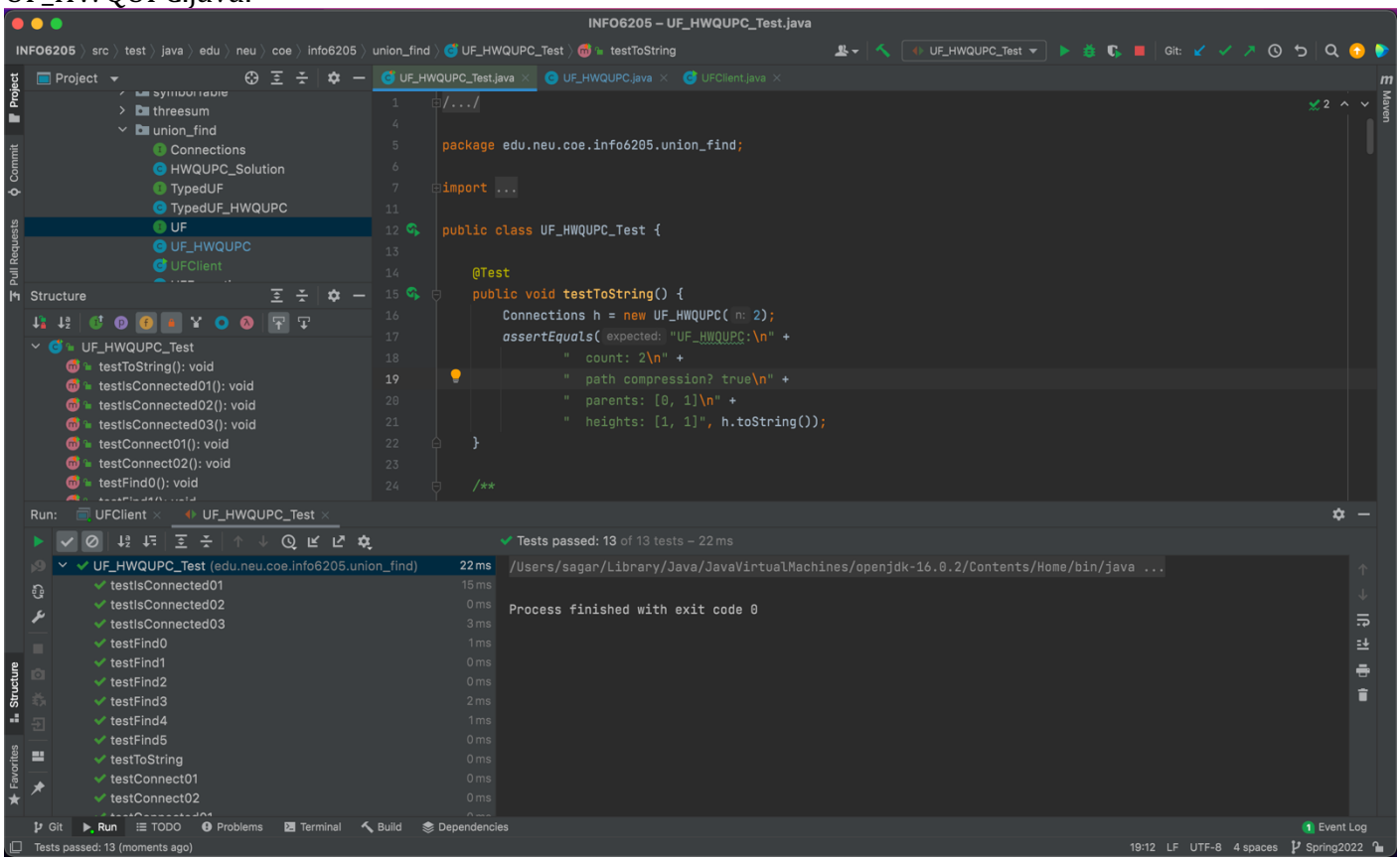
Implemented required changes in mergeComponents and doPathCompression method:

```
private void mergeComponents(int i, int j) {
    // FIXME make shorter root point to taller one
    if (i == j) return;
    if (height[i] < height[j]) {
        updateParent(i, j);
        updateHeight(j, i);
    } else {
        updateParent(j, i);
        updateHeight(i, j);
    }
    // END
}
```

```java
/**
 * This implements the single-pass path-halving mechanism of path compression
 */
private void doPathCompression(int i) {
    // FIXME update parent to value of grandparent
    parent[i]= parent[parent[i]];
    // END
}
```

All Unit Test Cases Passed:

UF_HWQUPC.java:



Step 2:

Implemented UFClient.java class:

```java
package edu.neu.coe.info6205.union_find;

import java.util.Random;
import java.util.Scanner;

public class UFClient {

    public static int count(int n) {
        UF_HWQUPC uf = new UF_HWQUPC(n);
        Random random = new Random();
        int count = 0;
        while (uf.components() > 1) {
            int x = random.nextInt(n);
            int y = random.nextInt(n);
            // make sure that if x and y is connected, if is not connected then union x
and y
```

```
            uf.connect(x, y);
            count++;
        }
        return count;
    }

    public static void main(String[] args) {

        System.out.println("please enter a number from the command line  (eg. 100) ");
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        System.out.println("the number of objects is " + n + ", and the number of
connections is " + count(n));

        System.out.println("part 3, test the relationship between m and n");

        for (int i = 1000; i < 160000; i *= 2) {
            int sum = 0;

            for (int j = 0; j < 10; j++) {
                sum += count(i);
            }
            int avgNum = sum / 10;
            System.out.println("the number of objects is " + i + ", and the number of
pairs is " + avgNum);
        }
    }
}
```

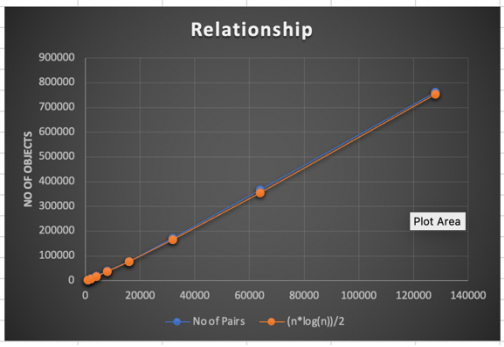
Evidence and Analysis:

Run main program in UFClient.java. We can input the number from command line to test the count method. We can run more n values and make their values bigger using doubling method, each with 10 times to test the relationship between m and n.

We mapped the 2 outputs using above method. First passed the input from command line as 100 and then passed the input from command line as 200. Output of both was mapped in a graphical format. Output and graphical representation is as below. I will be also uploading excel file of this analysis. Please check that.

Input: 100

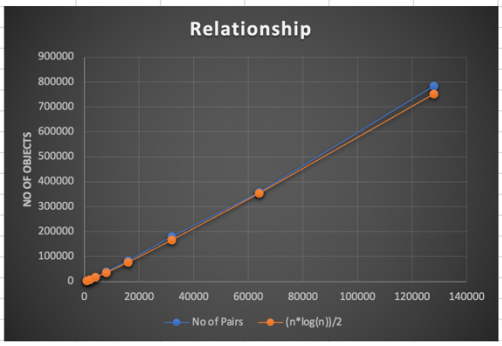
No of Objects	No of Pairs	(n*log(n))/2
1000	3403	3454
2000	7948	7601
4000	17902	16588
8000	39285	35949
16000	78039	77443
32000	172170	165976
64000	367595	354132
128000	762258	752626




```
/Users/sagar/Library/Java/JavaVirtualMachines/openjdk-16.0.2/Contents
please enter a number from the command line  (eg. 100)
100
the number of objects is 100, and the number of connections is 419
part 3, test the relationship between m and n
the number of objects is 1000, and the number of pairs is 3403
the number of objects is 2000, and the number of pairs is 7948
the number of objects is 4000, and the number of pairs is 17902
the number of objects is 8000, and the number of pairs is 39285
the number of objects is 16000, and the number of pairs is 78039
the number of objects is 32000, and the number of pairs is 172170
the number of objects is 64000, and the number of pairs is 367595
the number of objects is 128000, and the number of pairs is 762258
```

Input: 200

No of Objects	No of Pairs	(n*log(n))/2
1000	3739	3454
2000	8076	7601
4000	18002	16588
8000	39392	35949
16000	83276	77443
32000	179854	165976
64000	357956	354132
128000	783778	752626




```
/Users/sagar/Library/Java/JavaVirtualMachines/openjdk-16.0.2/Contents/
please enter a number from the command line  (eg. 100)
200
the number of objects is 200, and the number of connections is 519
part 3, test the relationship between m and n
the number of objects is 1000, and the number of pairs is 3739
the number of objects is 2000, and the number of pairs is 8076
the number of objects is 4000, and the number of pairs is 18002
the number of objects is 8000, and the number of pairs is 39392
the number of objects is 16000, and the number of pairs is 83276
the number of objects is 32000, and the number of pairs is 179854
the number of objects is 64000, and the number of pairs is 357956
the number of objects is 128000, and the number of pairs is 783778
```

In the above graph

- x axis shows the no of objects (n)
- Blue line represents No of pairs for respective no of objects(n)
- Orange line shows the relationship output respective to no of objects(n)

Relationsh Conclusion:

From the above observations we came to following conclusion:
Both the lines are approximately same. We can derive the relationship between m and n as below:

$$m = \frac{1}{2} (n \log(n))$$