Program Structures and Algorithms Spring 2023(SEC –1)
Assignment-4

Name: Naga Venkata Nishanth Sayana

NU ID: 002930970

Task:

The here is to implement height weighted quick union with path compression (HWQUPC). In this task, instead checking the weight of each tree, we would be checking the height of each tree and making the union based on it.

In the step 1 we have to implement the find(), mergeComponents(), and doPathCompression() methods.

Code Screenshots:

1)find method

```
    public int find(int p) {
        validate(p);
        int root = p;
        // FIXME
        if(pathCompression == true)
            doPathCompression(p);
        while (root != parent[root]) {
            root = parent[root];
        }
        // END
        return root;
    }

    /**
```

UFClient

```
C:\Users\snvni\.jdks\openjdk-19\bin\java.exe ...
```

2)Merge Components

```
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(j,i);
        height[i] += 1;
    }
    else if(height[i] > height[j]){
        updateParent(j,i);
    }
    else{
        updateParent(i,j);
    }
    // END
}
```

```
Client
:\Users\snvni\.jdks\openjdk-19\bin\java.exe ...
```

3)doPathCompression()

```
1 usage   xiaohuanlin *
private void doPathCompression(int i) {
    // FIXME update parent to value of grandparent
    while(i != parent[i]){
        parent[i] = parent[parent[i]];
        i = parent[i];
    }
    // END
}
}
```

```
UFClient
C:\Users\snvni\.jdks\openjdk-19\bin\java.exe ...
```

In Step 2, we have to write a UFClient class(Union Find Client), which generates random pairs of integers between 0 to n-1, and checks if they are connected, if not it makes a union of them. I have a static

method count(int n) which takes n as an argument(No of Sites) and returns the total number of connections made(m).

UF_HWQUPC.java | UFClient.java | UF_HWQUPC_Test.java

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

import java.util.Random;

public class UFClient {

    public static int count(int n){
        int totalCount=0;
        for(int i=0;i<1000;i++){
            UF_HWQUPC uf_hwqupc=new UF_HWQUPC(n, pathCompression: true);
            int pairCount=0;

            while(uf_hwqupc.components()!=1){
                int[] pair=randomPair(n);
                pairCount++;
                if(!uf_hwqupc.connected(pair[0],pair[1])){
                    uf_hwqupc.union(pair[0],pair[1]);
                }
            }
            totalCount+=pairCount;
        }
        return totalCount/1000;
    }

    1 usage
    private static int[] randomPair(int n){
        Random r = new Random();
        int p = r.nextInt(n);
        int q = r.nextInt(n);
        while(p == q){
            q = r.nextInt(n);
        }
        return new int[]{p,q};
    }
```

```
    1 usage
    private static int[] randomPair(int n){
        Random r = new Random();
        int p = r.nextInt(n);
        int q = r.nextInt(n);
        while(p == q){
            q = r.nextInt(n);
        }
        return new int[]{p,q};
    }

    public static void main(String[] args){
        int[] siteValues={100,200,400,800,1600,3200,6400,12800};

        for(int n:siteValues){
            int pairCount=count(n);

            System.out.println("No of Sites(n): "+n+" | No of connections generated(m): "+pairCount);
        }
    }
}
```

UFClient

```
C:\Users\snvni\.jdks\openjdk-19\bin\java.exe ...
```

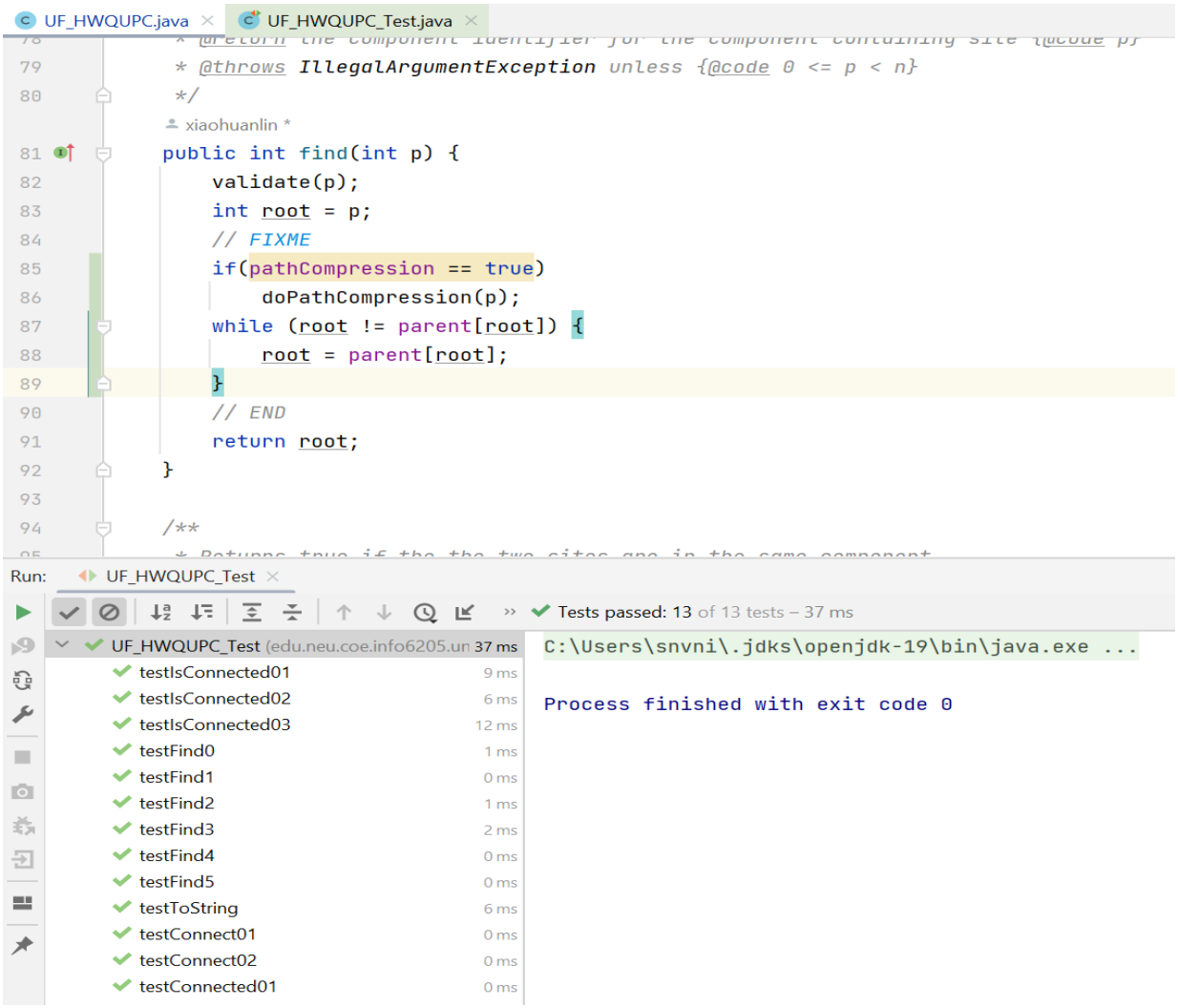
Output:

```
34
35      public static void main(String[] args){
36          int[] siteValues={100,200,400,800,1600,3200,6400,12800};
37
38          for(int n:siteValues){
39              int pairCount=count(n);
40
41              System.out.println("No of Sites(n): "+n+" | No of connections generated(m): "+pairCount);
42          }
43      }
44  }
```

Run: UFClient

```
C:\Users\snvni\.jdks\openjdk-19\bin\java.exe ...
No of Sites(n): 100 | No of connections generated(m): 258
No of Sites(n): 200 | No of connections generated(m): 586
No of Sites(n): 400 | No of connections generated(m): 1292
No of Sites(n): 800 | No of connections generated(m): 2909
No of Sites(n): 1600 | No of connections generated(m): 6364
No of Sites(n): 3200 | No of connections generated(m): 13839
No of Sites(n): 6400 | No of connections generated(m): 29911
No of Sites(n): 12800 | No of connections generated(m): 64260
```

Test Cases Screenshot:



Observations:

Here I have taken n values starting from 100 to 12800 using doubling method and I have run the experiment for 1000 times each and calculated the average of number of connections made.

I have illustrated the results I have obtained below. Based on the values, I have been able to derive the relationship between n and m, that would be $m = (n/2) * \ln(n)$.

As we can see from the graphs below, the plot of m and $m = (n/2) * \ln(n)$ are very close.

No of Sites (n)	No of connections generated (m)	1/2 n ln(n)
100	258	230
200	586	529
400	1292	1198
800	2909	2673
1600	6364	5902
3200	13839	12913
6400	29911	28044
12800	64260	60526