INFO 6205 - Program Structures and Algorithms
Spring 2024

NAME: Neha Devarapalli
NUID: 002883137
GITHUB LINK: https://github.com/nehadevarapalli/INFO6205

Task: Assignment 4 (WQUPC)

Part 1:

Implement height-weighted Quick Union with Path Compression by completing marked sections in **UF_HWQUPC** and ensure all unit tests for **UF_HWQUPC_Test** pass.

Part 2:

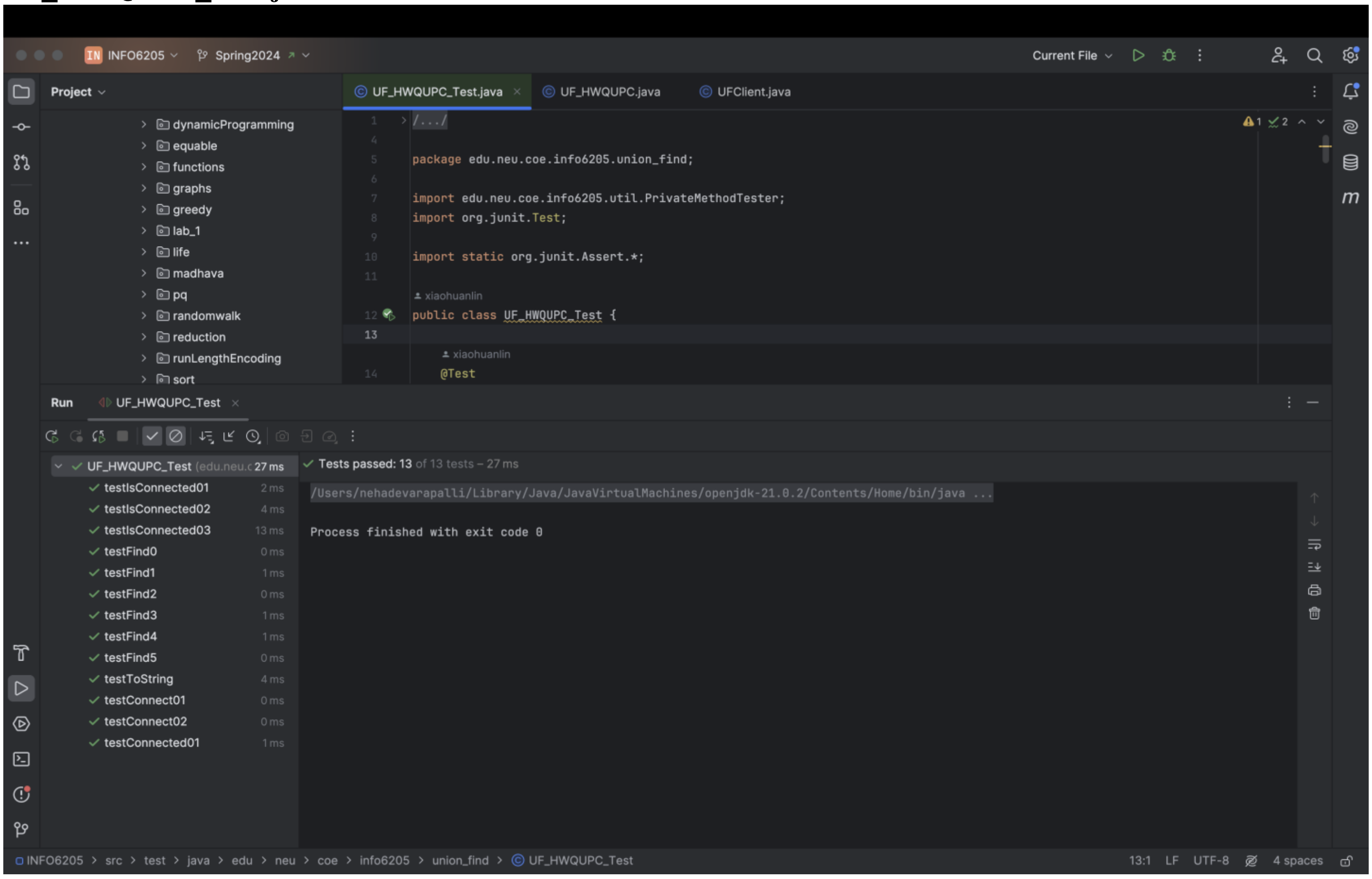
- Develop a UF client to connect all sites:
  - Take **n** number of sites.
  - Generate random pairs and connect them until all sites are connected.
  - Print the number of connections made.
- Package the program:
  - Implement a **count()** method to return the number of connections for a given **n**.
  - Implement a **main()** method to call **count()** and print the result.
  - Optionally, create the main program for a fixed set of **n** values.

Part 3:

Determine the relationship between **n** (number of objects) and **m** (number of pairs) needed for full connectivity and justify it.

Unit Tests Screenshot (Part 1):

UF_HWQUPC_Test.java



```
/.../

package edu.neu.coe.info6205.union_find;

import edu.neu.coe.info6205.util.PrivateMethodTester;
import org.junit.Test;

import static org.junit.Assert.*;

public class UF_HWQUPC_Test {

    @Test
```

```
UF_HWQUPC_Test (edu.neu.c 27 ms
  testIsConnected01   2 ms
  testIsConnected02   4 ms
  testIsConnected03   13 ms
  testFind0           0 ms
  testFind1           1 ms
  testFind2           0 ms
  testFind3           1 ms
  testFind4           1 ms
  testFind5           0 ms
  testToString        4 ms
  testConnect01       0 ms
  testConnect02       0 ms
  testConnected01     1 ms
```

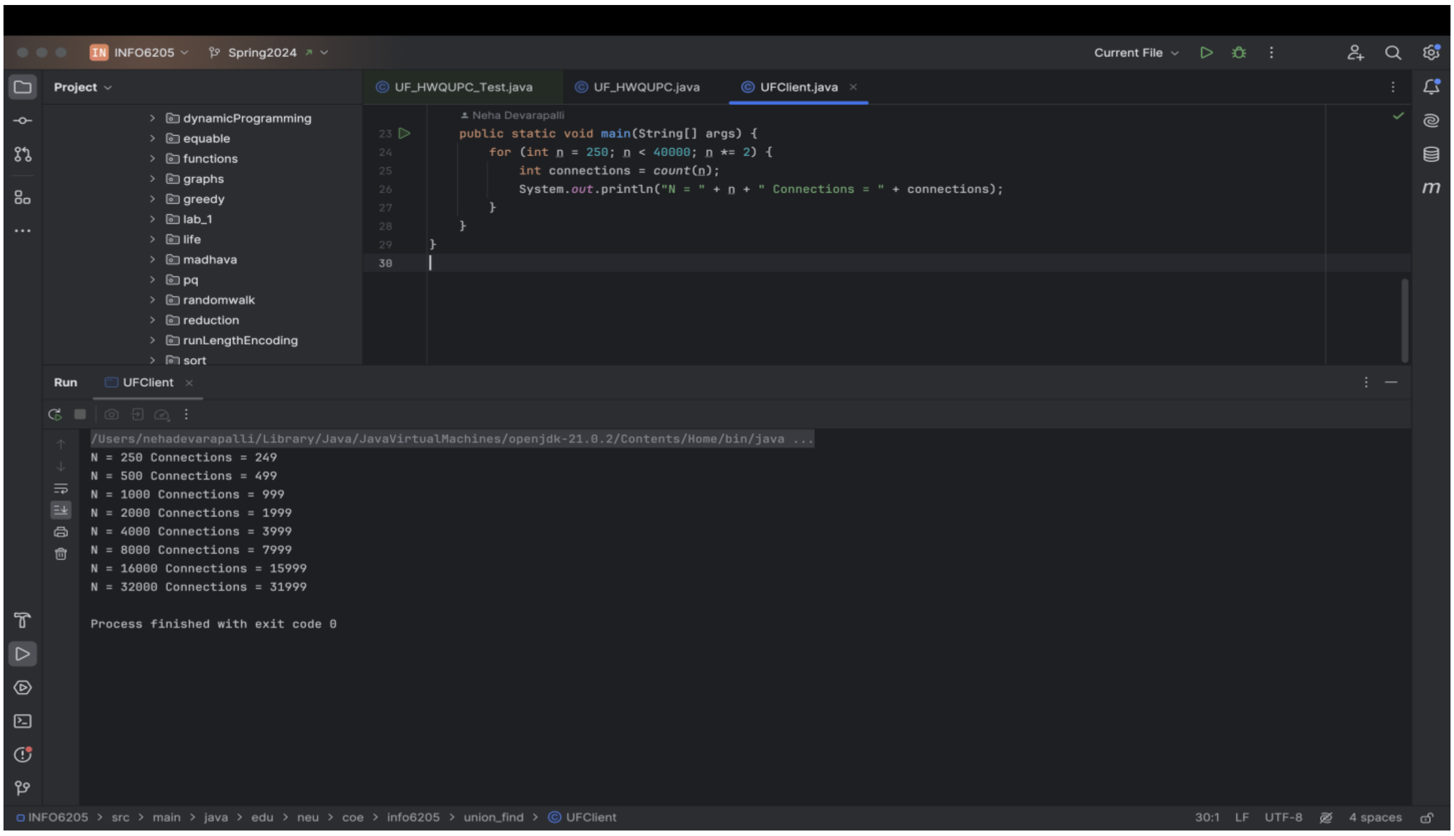
```
Tests passed: 13 of 13 tests – 27 ms

/Users/nehadevarapalli/Library/Java/JavaVirtualMachines/openjdk-21.0.2/Contents/Home/bin/java ...

Process finished with exit code 0
```

Results for (Part 2):

Implemented a new file **UFClient.java** for Part 2 of the assignment. The following are the observed results:



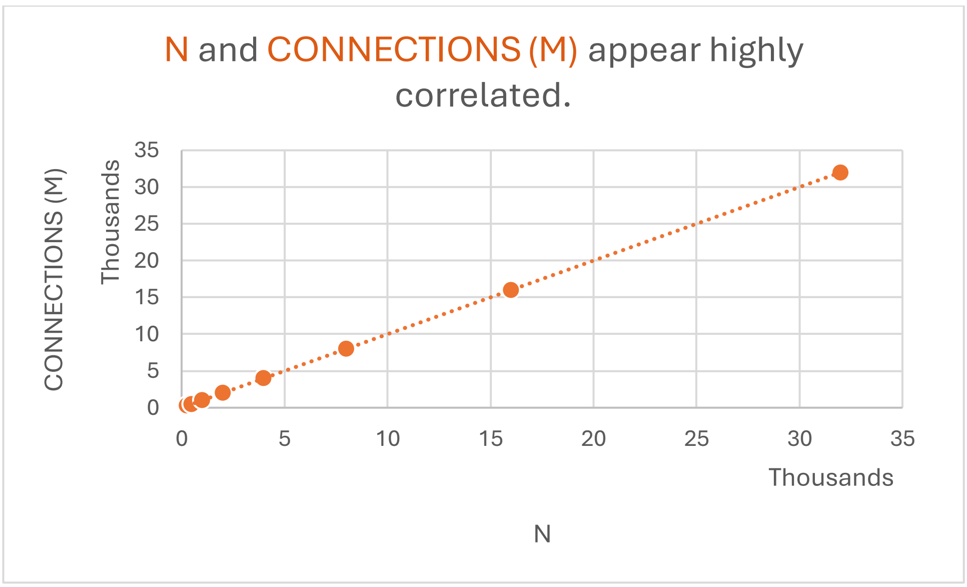
```
    public static void main(String[] args) {
        for (int n = 250; n < 40000; n *= 2) {
            int connections = count(n);
            System.out.println("N = " + n + " Connections = " + connections);
        }
    }
}
```

```
/Users/nehadevarapalli/Library/Java/JavaVirtualMachines/openjdk-21.0.2/Contents/Home/bin/java ...
N = 250 Connections = 249
N = 500 Connections = 499
N = 1000 Connections = 999
N = 2000 Connections = 1999
N = 4000 Connections = 3999
N = 8000 Connections = 7999
N = 16000 Connections = 15999
N = 32000 Connections = 31999

Process finished with exit code 0
```

Observations and Relationship Conclusion (Part 3):

As we can see, the number of connections (m) to be made to construct a fully connected graph is n-1 in the above results.



Initially, each node is in its own component and when connecting two components, if one tree is taller than the other, we require one connection to connect the shorter tree to the taller tree.

Since all nodes start in separate sets (disconnected graph), n–1 connections are needed for complete connectivity and path compression significantly reduces the number of steps involved in making these connections. Hence, the relationship should be:

$$\mathbf{m = n - 1}$$

where m = number of connections

n = number of nodes

Justification:

1. Each connection merges two sets into one.
2. Since every node starts in a separate set, we need to merge pairs of sets until only one set remains (complete connectivity).
3. Merging n separate sets requires n-1 connections.