Program Structures and Algorithms

Spring 2023(SEC – 1)

NAME: Venkat Pavan Munaganti

NUID: 002722397

Task: Assignment 4: **Weighted Union Find with Path Compression**

Step 1:
(a) Implement height-weighted Quick Union with Path Compression. For this, you will flesh out the class UF_HWQUPC. All you have to do is to fill in the sections marked with // TO BE IMPLEMENTED ... // ...END IMPLEMENTATION.

(b) Check that the unit tests for this class all work. You must show "green" test results in your submission (screenshot is OK).

Step 2:
Using your implementation of UF_HWQUPC, 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. Package your program as a static method count() that takes n as the argument and returns the number of connections; and a main() that takes n from the command line, calls count() and prints the returned value. If you prefer, you can create a main program that doesn't require any input and runs the experiment for a fixed set of n values. Show evidence of your run(s).

Step 3:
Determine the relationship between the number of objects (*n*) and the number of pairs (*m*) generated to accomplish this (i.e. to reduce the number of components from *n* to 1). Justify your conclusion in terms of your observations and what you think might be going on.

NOTE: although I'm not going to tell you in advance what the relationship is, I can assure you that it is a *simple* relationship.

Don't forget to follow the submission guidelines. And to use sufficient (and sufficiently large) different values of n.

Conclusion about the relationship: The relationship between the number of objects (n) and number of pairs (m) to reduce the components from n to 1 is,

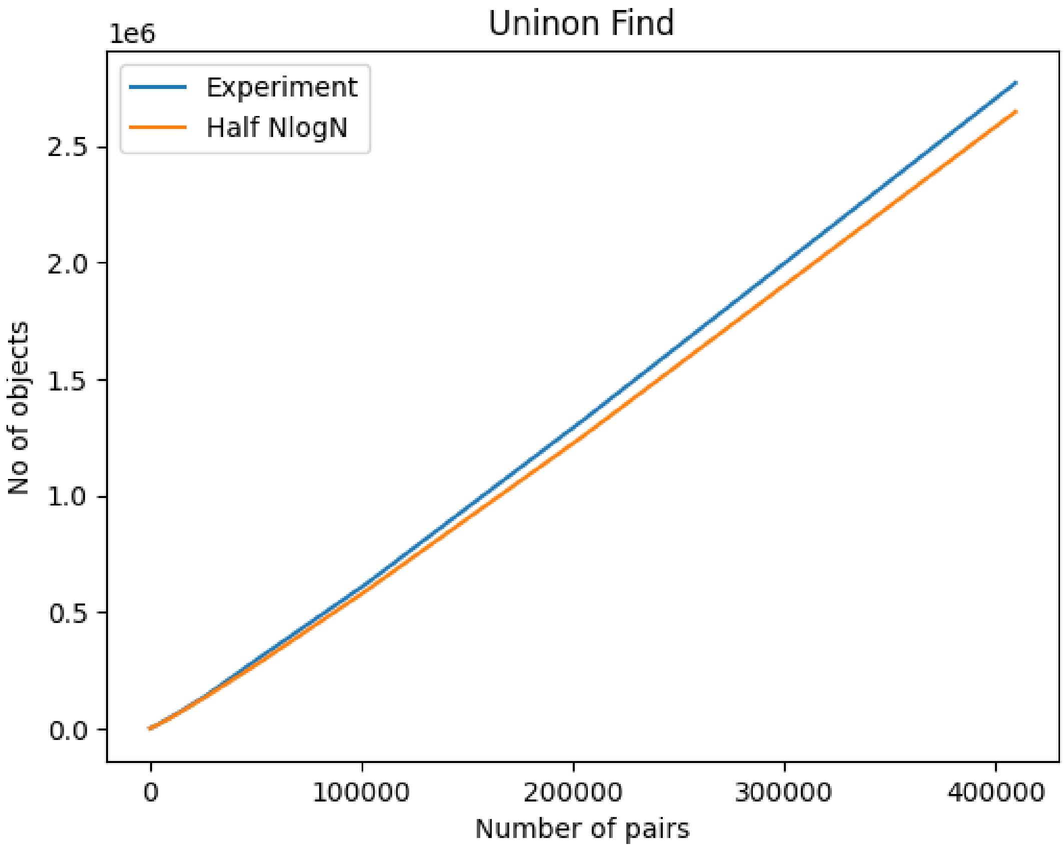
$$\boldsymbol{m \approx \frac{1}{2} n * log(n)}$$

Where m is the number of pairs, and n is the number of objects

Experimental proof: Let's consider a data set generated by the UF_Client.java for n in range 100 to 409600 generated using doubling method, and mean of number of pairs (m) for each n over 100 runs. Below is the experimental data in tabular form.

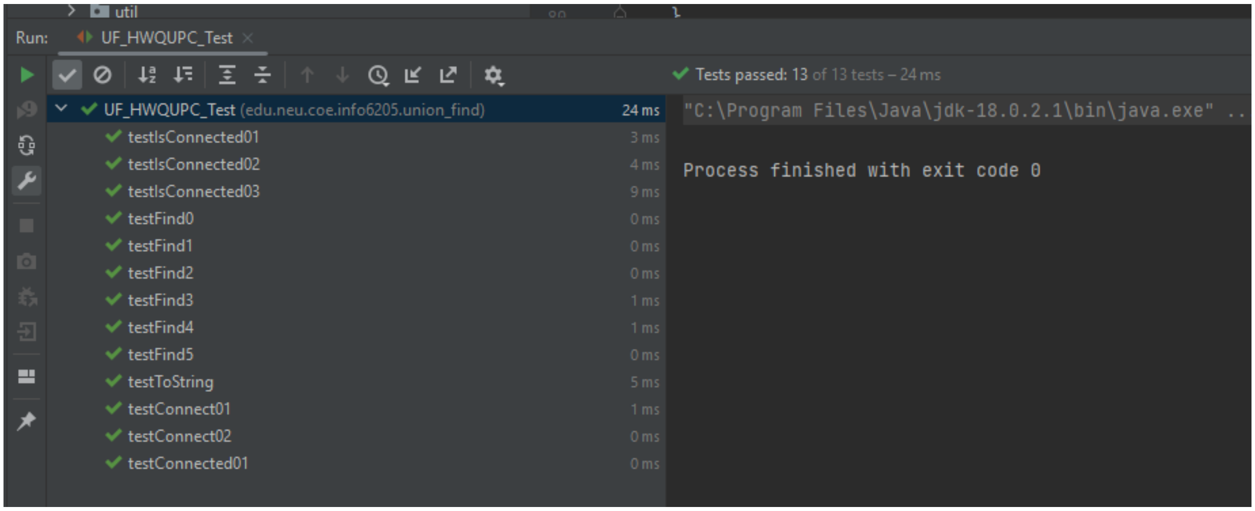
N	M	$\boldsymbol{\frac{1}{2} n * log(n)}$
100	254.95	230.258509
200	603.52	529.831737
400	1359.52	1198.29291
800	2908.04	2673.84469
1600	6304.13	5902.20713
3200	13525.99	12913.4497
6400	30423.56	28044.9705
12800	63250.19	60526.0829
25600	134399.66	129924.45
51200	298967.24	277593.467
102400	620226.9	590676.07
204800	1321211.3	1252330.41
409600	2770013.64	2646617.37

Above table contains experimental data generated by the UF_Client.java class using doubling method. N is the number of objects and M is the number of pairs. The graph plotted by the above data is almost like the that of $\boldsymbol{\frac{1}{2} n * log(n)}$



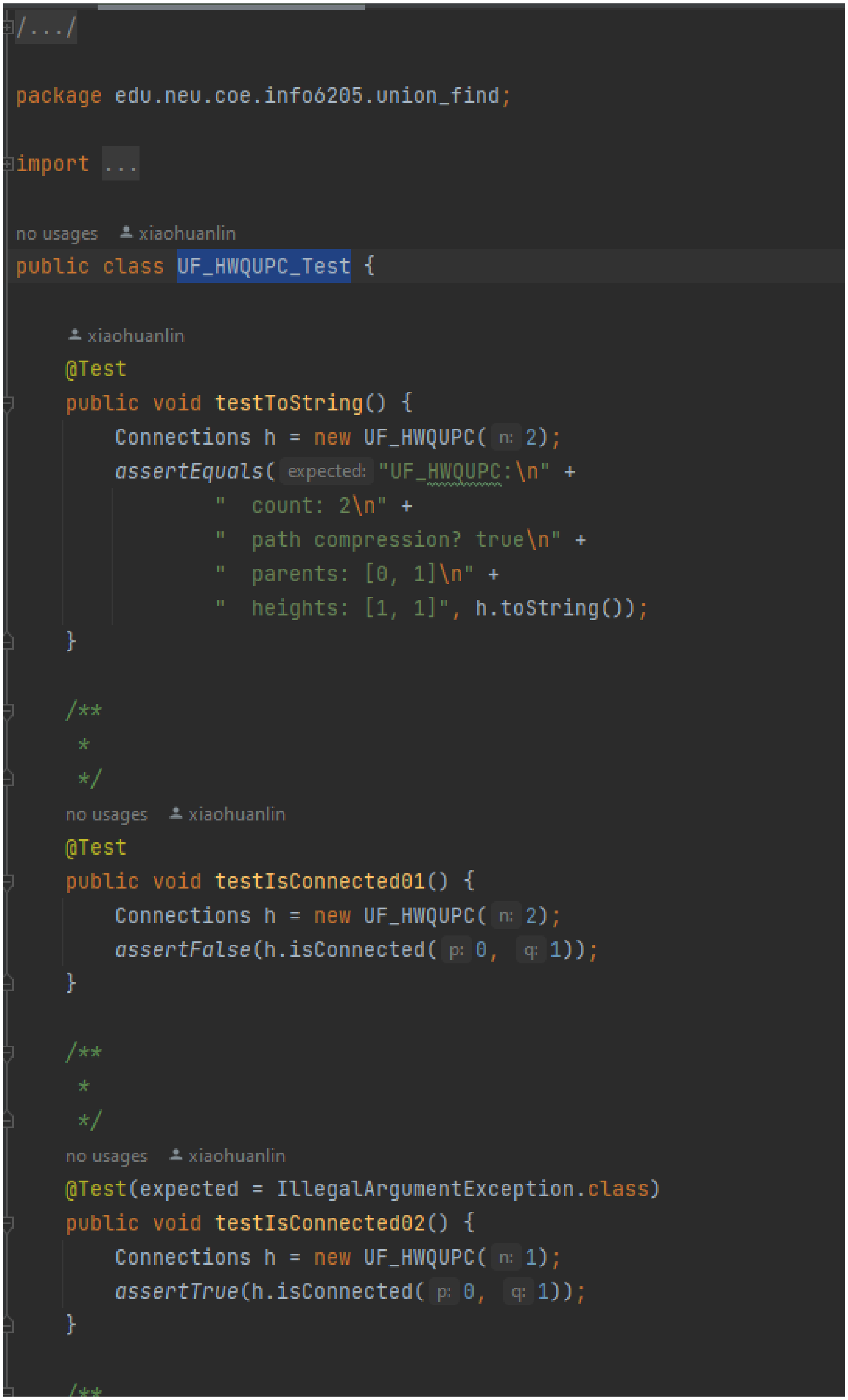
From the above graph it is evident that the graph plotted between the function $\frac{1}{2}\boldsymbol{n} * \boldsymbol{log}(\boldsymbol{n})$ and the graph plotted with the experiment data is almost identical and hence justifies the relationship between the number of objects (n) and number of steps (m)

Unit Test Cases:

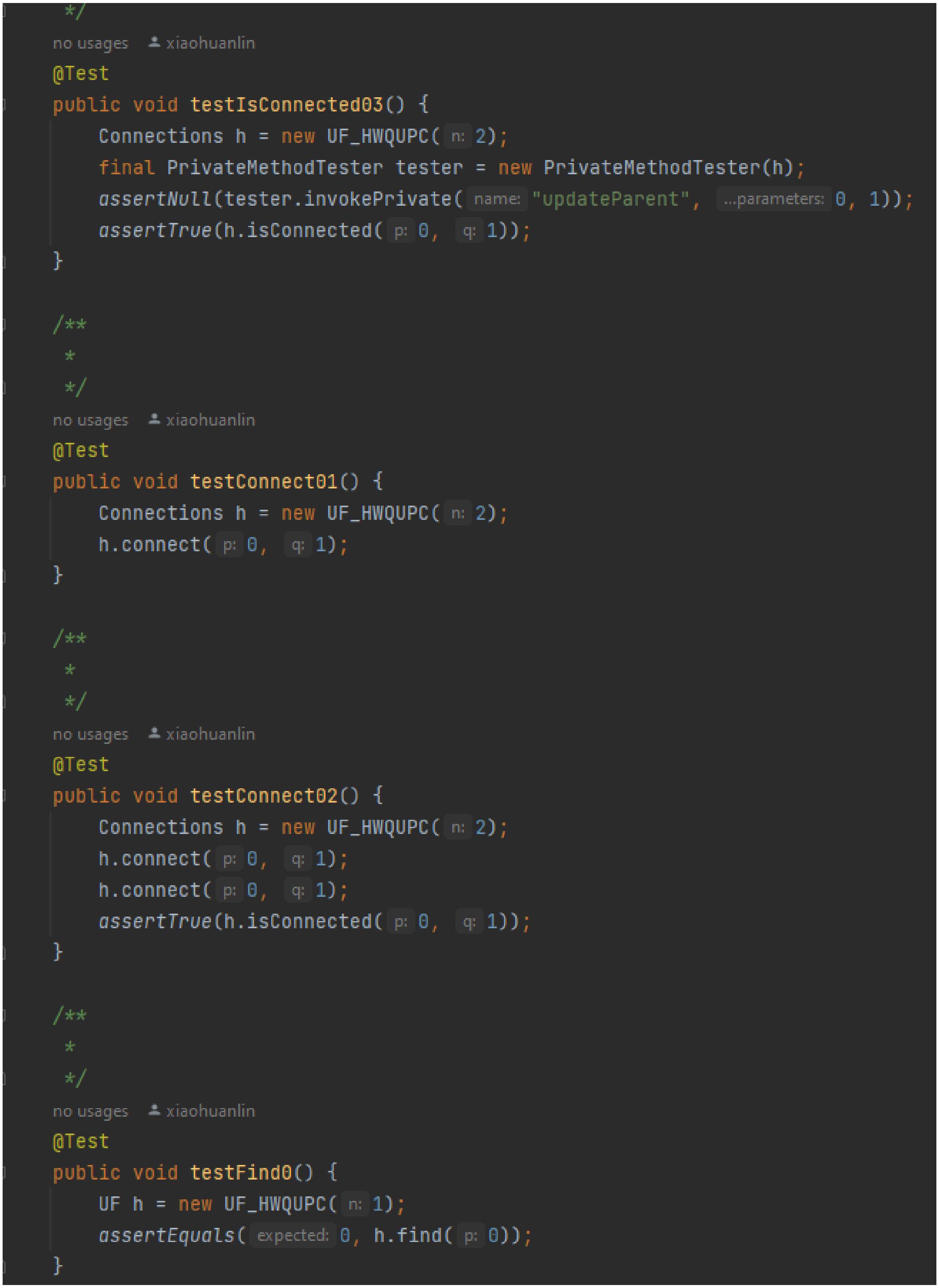


```
✓ Tests passed: 13 of 13 tests – 24 ms

✓ UF_HWQUPC_Test (edu.neu.coe.info6205.union_find)  24 ms    "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ..
    ✓ testIsConnected01                              3 ms
    ✓ testIsConnected02                              4 ms    Process finished with exit code 0
    ✓ testIsConnected03                              9 ms
    ✓ testFind0                                      0 ms
    ✓ testFind1                                      0 ms
    ✓ testFind2                                      0 ms
    ✓ testFind3                                      1 ms
    ✓ testFind4                                      1 ms
    ✓ testFind5                                      0 ms
    ✓ testToString                                   5 ms
    ✓ testConnect01                                  1 ms
    ✓ testConnect02                                  0 ms
    ✓ testConnected01                                0 ms
```

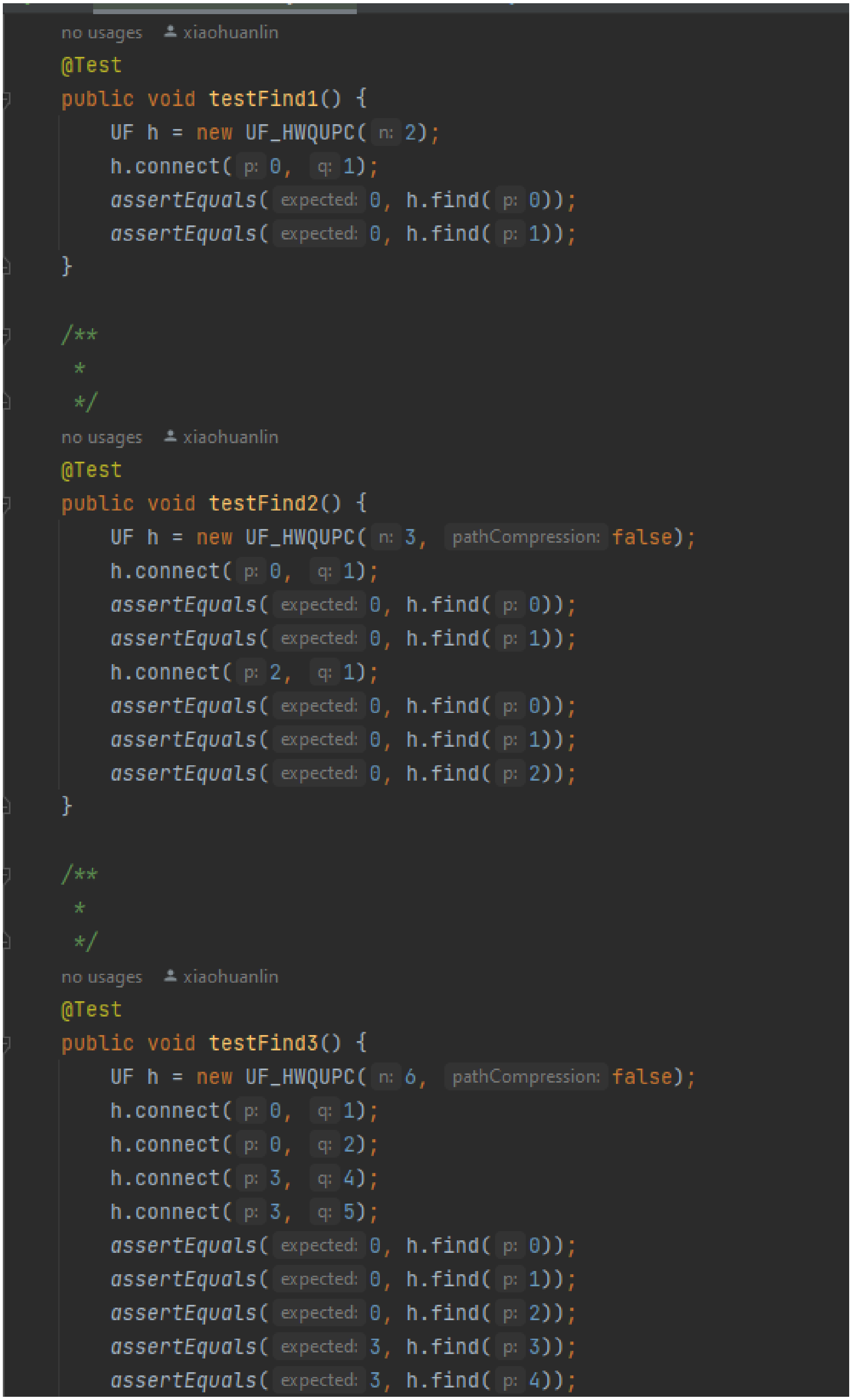
UF_HWQUPC_Test:



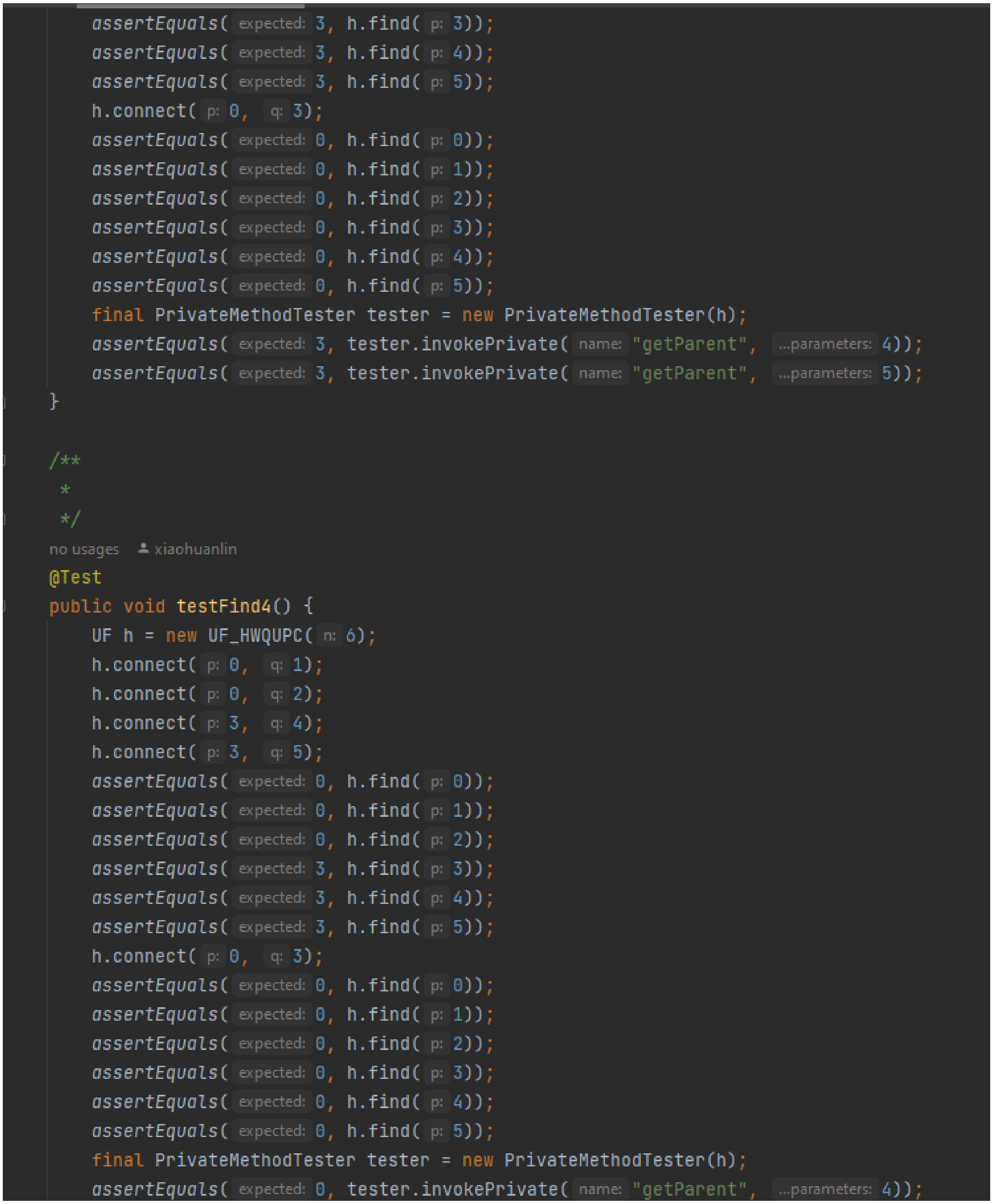
```
/.../

package edu.neu.coe.info6205.union_find;

import ...

no usages   xiaohuanlin
public class UF_HWQUPC_Test {

    xiaohuanlin
    @Test
    public void testToString() {
        Connections h = new UF_HWQUPC( n: 2);
        assertEquals( expected: "UF_HWQUPC:\n" +
                "  count: 2\n" +
                "  path compression? true\n" +
                "  parents: [0, 1]\n" +
                "  heights: [1, 1]", h.toString());
    }

    /**
     *
     */
    no usages   xiaohuanlin
    @Test
    public void testIsConnected01() {
        Connections h = new UF_HWQUPC( n: 2);
        assertFalse(h.isConnected( p: 0,  q: 1));
    }

    /**
     *
     */
    no usages   xiaohuanlin
    @Test(expected = IllegalArgumentException.class)
    public void testIsConnected02() {
        Connections h = new UF_HWQUPC( n: 1);
        assertTrue(h.isConnected( p: 0,  q: 1));
    }

    /**
```



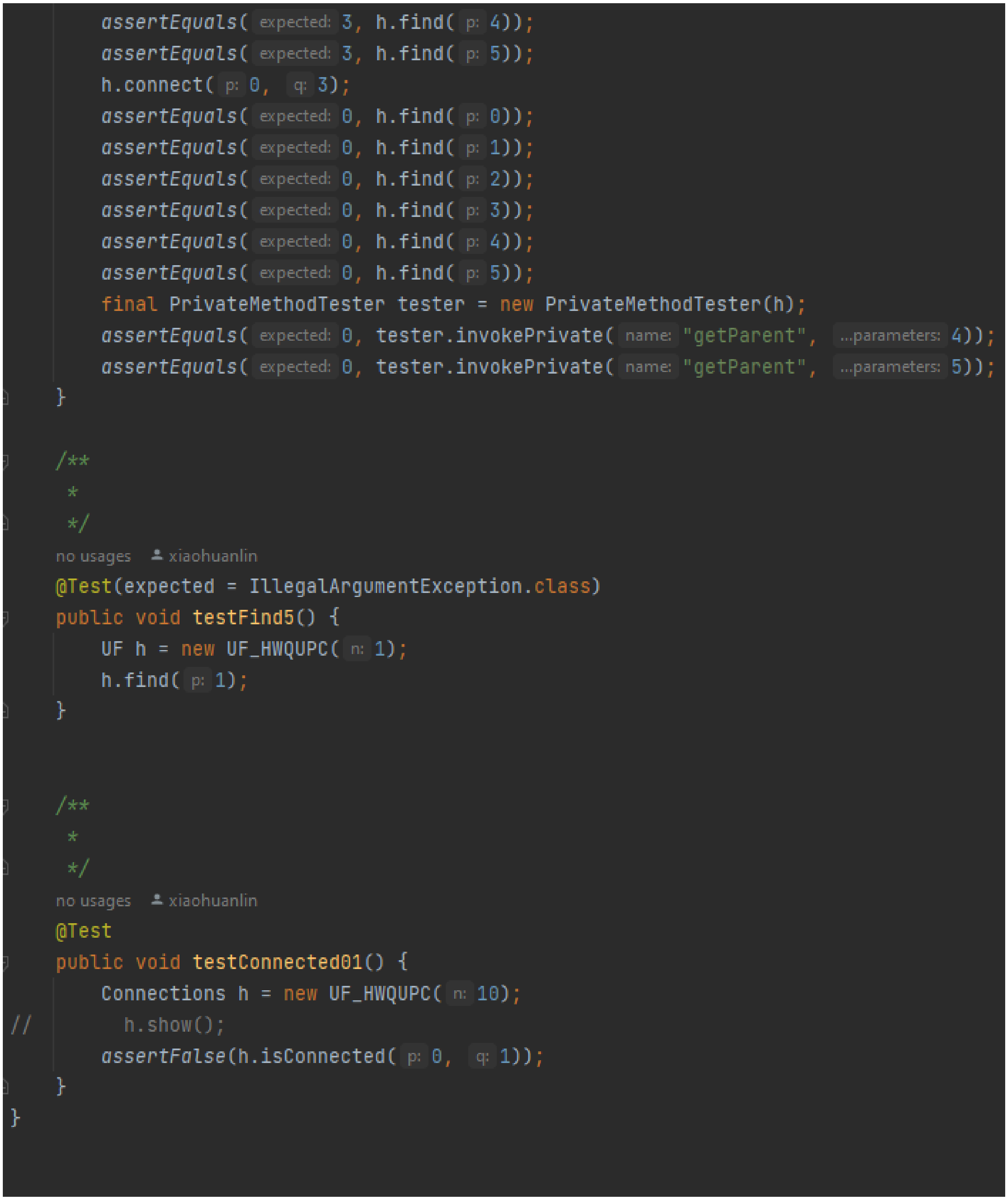
```
     */
    no usages  xiaohuanlin
    @Test
    public void testIsConnected03() {
        Connections h = new UF_HWQUPC( n: 2);
        final PrivateMethodTester tester = new PrivateMethodTester(h);
        assertNull(tester.invokePrivate( name: "updateParent",  ...parameters: 0, 1));
        assertTrue(h.isConnected( p: 0,  q: 1));
    }

    /**
     *
     */
    no usages  xiaohuanlin
    @Test
    public void testConnect01() {
        Connections h = new UF_HWQUPC( n: 2);
        h.connect( p: 0,  q: 1);
    }

    /**
     *
     */
    no usages  xiaohuanlin
    @Test
    public void testConnect02() {
        Connections h = new UF_HWQUPC( n: 2);
        h.connect( p: 0,  q: 1);
        h.connect( p: 0,  q: 1);
        assertTrue(h.isConnected( p: 0,  q: 1));
    }

    /**
     *
     */
    no usages  xiaohuanlin
    @Test
    public void testFind0() {
        UF h = new UF_HWQUPC( n: 1);
        assertEquals( expected: 0, h.find( p: 0));
    }
```



```
no usages   xiaohuanlin
@Test
public void testFind1() {
    UF h = new UF_HWQUPC( n: 2);
    h.connect( p: 0,  q: 1);
    assertEquals( expected: 0, h.find( p: 0));
    assertEquals( expected: 0, h.find( p: 1));
}

/**
 *
 */
no usages   xiaohuanlin
@Test
public void testFind2() {
    UF h = new UF_HWQUPC( n: 3,  pathCompression: false);
    h.connect( p: 0,  q: 1);
    assertEquals( expected: 0, h.find( p: 0));
    assertEquals( expected: 0, h.find( p: 1));
    h.connect( p: 2,  q: 1);
    assertEquals( expected: 0, h.find( p: 0));
    assertEquals( expected: 0, h.find( p: 1));
    assertEquals( expected: 0, h.find( p: 2));
}

/**
 *
 */
no usages   xiaohuanlin
@Test
public void testFind3() {
    UF h = new UF_HWQUPC( n: 6,  pathCompression: false);
    h.connect( p: 0,  q: 1);
    h.connect( p: 0,  q: 2);
    h.connect( p: 3,  q: 4);
    h.connect( p: 3,  q: 5);
    assertEquals( expected: 0, h.find( p: 0));
    assertEquals( expected: 0, h.find( p: 1));
    assertEquals( expected: 0, h.find( p: 2));
    assertEquals( expected: 3, h.find( p: 3));
    assertEquals( expected: 3, h.find( p: 4));
```

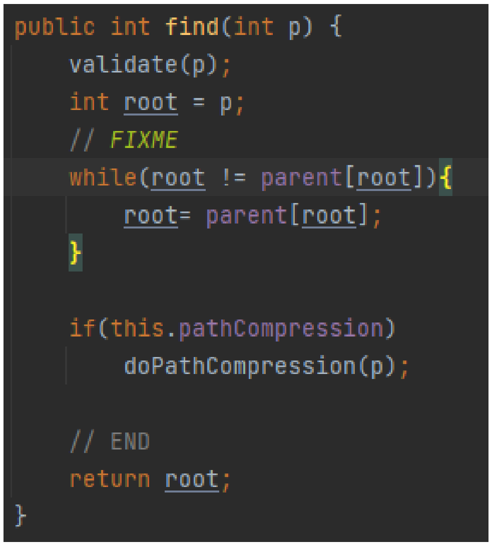


```
    assertEquals( expected: 3, h.find( p: 3));
    assertEquals( expected: 3, h.find( p: 4));
    assertEquals( expected: 3, h.find( p: 5));
    h.connect( p: 0,  q: 3);
    assertEquals( expected: 0, h.find( p: 0));
    assertEquals( expected: 0, h.find( p: 1));
    assertEquals( expected: 0, h.find( p: 2));
    assertEquals( expected: 0, h.find( p: 3));
    assertEquals( expected: 0, h.find( p: 4));
    assertEquals( expected: 0, h.find( p: 5));
    final PrivateMethodTester tester = new PrivateMethodTester(h);
    assertEquals( expected: 3, tester.invokePrivate( name: "getParent",  ...parameters: 4));
    assertEquals( expected: 3, tester.invokePrivate( name: "getParent",  ...parameters: 5));
}

/**
 *
 */
no usages  xiaohuanlin
@Test
public void testFind4() {
    UF h = new UF_HWQUPC( n: 6);
    h.connect( p: 0,  q: 1);
    h.connect( p: 0,  q: 2);
    h.connect( p: 3,  q: 4);
    h.connect( p: 3,  q: 5);
    assertEquals( expected: 0, h.find( p: 0));
    assertEquals( expected: 0, h.find( p: 1));
    assertEquals( expected: 0, h.find( p: 2));
    assertEquals( expected: 3, h.find( p: 3));
    assertEquals( expected: 3, h.find( p: 4));
    assertEquals( expected: 3, h.find( p: 5));
    h.connect( p: 0,  q: 3);
    assertEquals( expected: 0, h.find( p: 0));
    assertEquals( expected: 0, h.find( p: 1));
    assertEquals( expected: 0, h.find( p: 2));
    assertEquals( expected: 0, h.find( p: 3));
    assertEquals( expected: 0, h.find( p: 4));
    assertEquals( expected: 0, h.find( p: 5));
    final PrivateMethodTester tester = new PrivateMethodTester(h);
    assertEquals( expected: 0, tester.invokePrivate( name: "getParent",  ...parameters: 4));
```

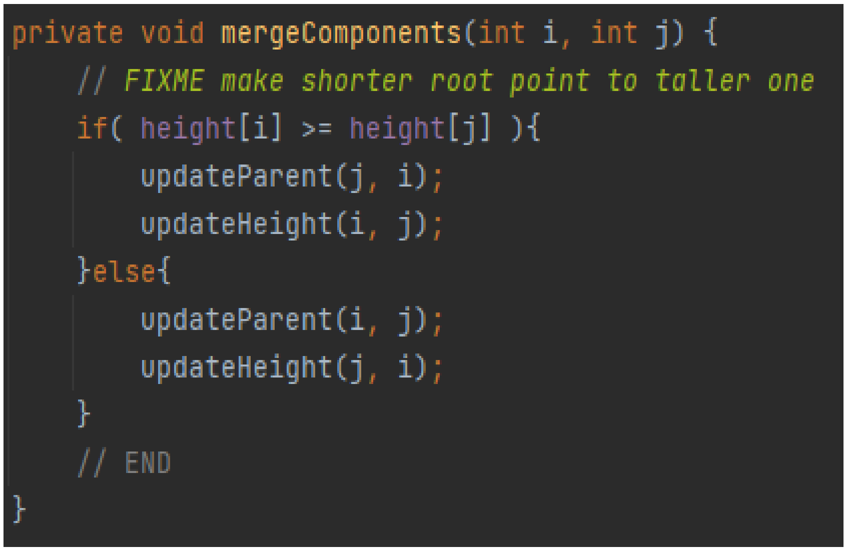


```
        assertEquals( expected: 3, h.find( p: 4));
        assertEquals( expected: 3, h.find( p: 5));
        h.connect( p: 0,  q: 3);
        assertEquals( expected: 0, h.find( p: 0));
        assertEquals( expected: 0, h.find( p: 1));
        assertEquals( expected: 0, h.find( p: 2));
        assertEquals( expected: 0, h.find( p: 3));
        assertEquals( expected: 0, h.find( p: 4));
        assertEquals( expected: 0, h.find( p: 5));
        final PrivateMethodTester tester = new PrivateMethodTester(h);
        assertEquals( expected: 0, tester.invokePrivate( name: "getParent",  ...parameters: 4));
        assertEquals( expected: 0, tester.invokePrivate( name: "getParent",  ...parameters: 5));
    }

    /**
     *
     */
    no usages  xiaohuanlin
    @Test(expected = IllegalArgumentException.class)
    public void testFind5() {
        UF h = new UF_HWQUPC( n: 1);
        h.find( p: 1);
    }


    /**
     *
     */
    no usages  xiaohuanlin
    @Test
    public void testConnected01() {
        Connections h = new UF_HWQUPC( n: 10);
//        h.show();
        assertFalse(h.isConnected( p: 0,  q: 1));
    }
}
```

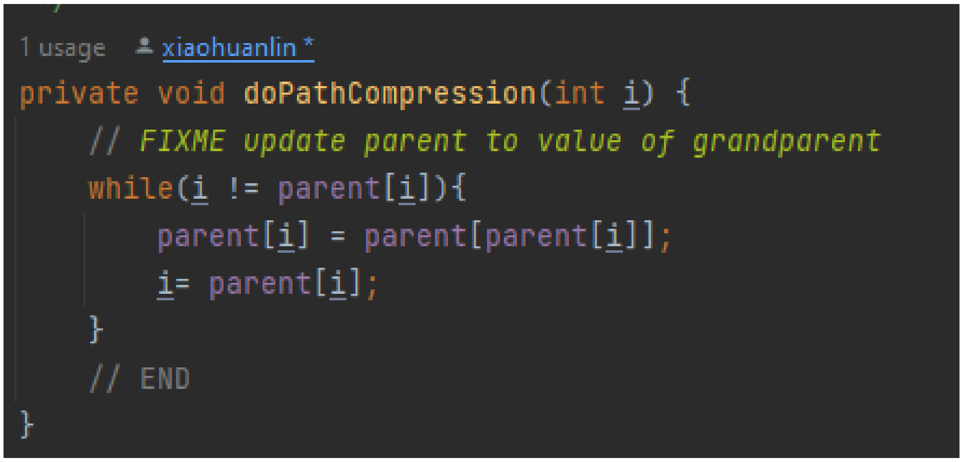
UF_HWQUPC.java



```java
public int find(int p) {
    validate(p);
    int root = p;
    // FIXME
    while(root != parent[root]){
        root= parent[root];
    }

    if(this.pathCompression)
        doPathCompression(p);

    // END
    return root;
}
```

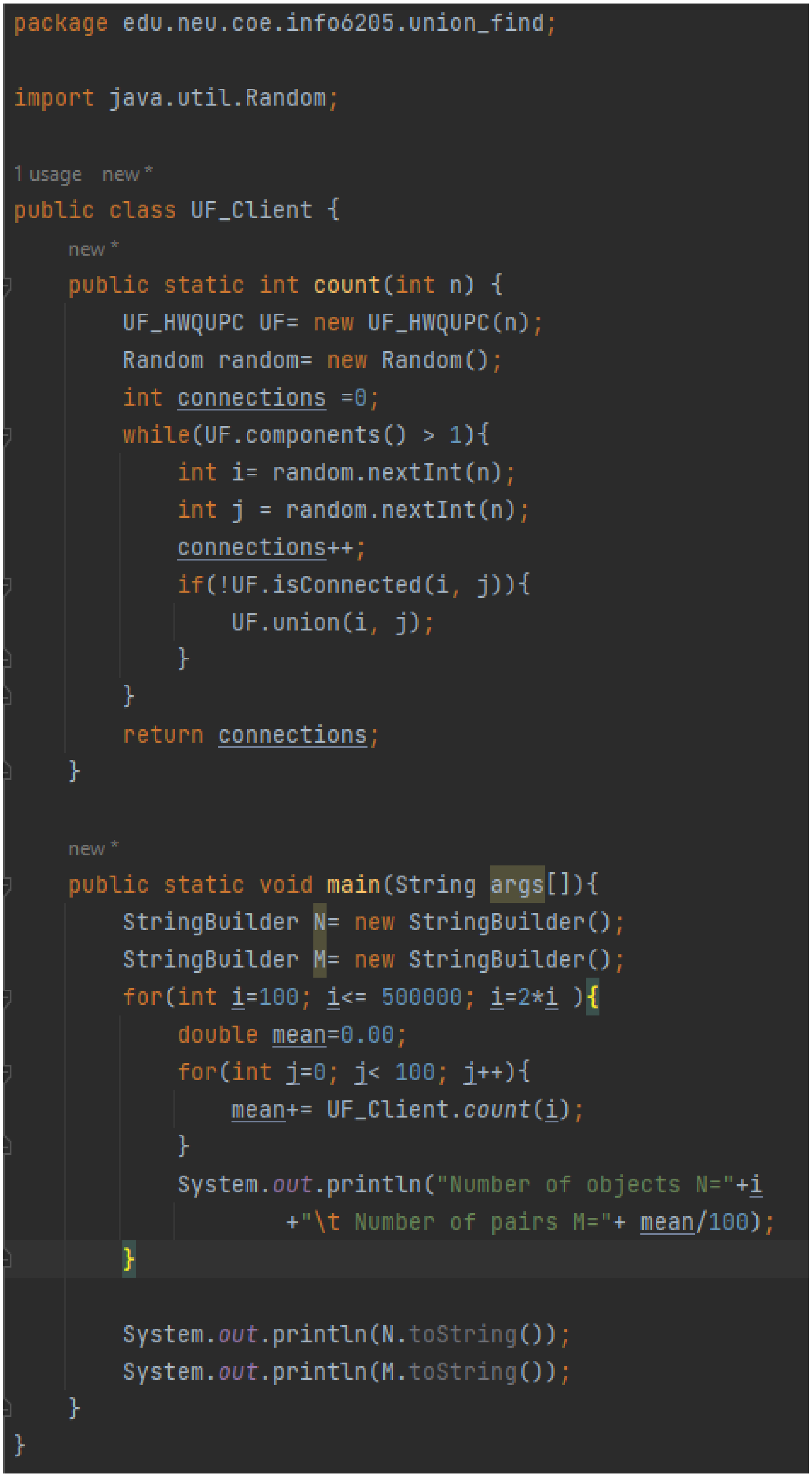


```java
private void mergeComponents(int i, int j) {
    // FIXME make shorter root point to taller one
    if( height[i] >= height[j] ){
        updateParent(j, i);
        updateHeight(i, j);
    }else{
        updateParent(i, j);
        updateHeight(j, i);
    }
    // END
}
```

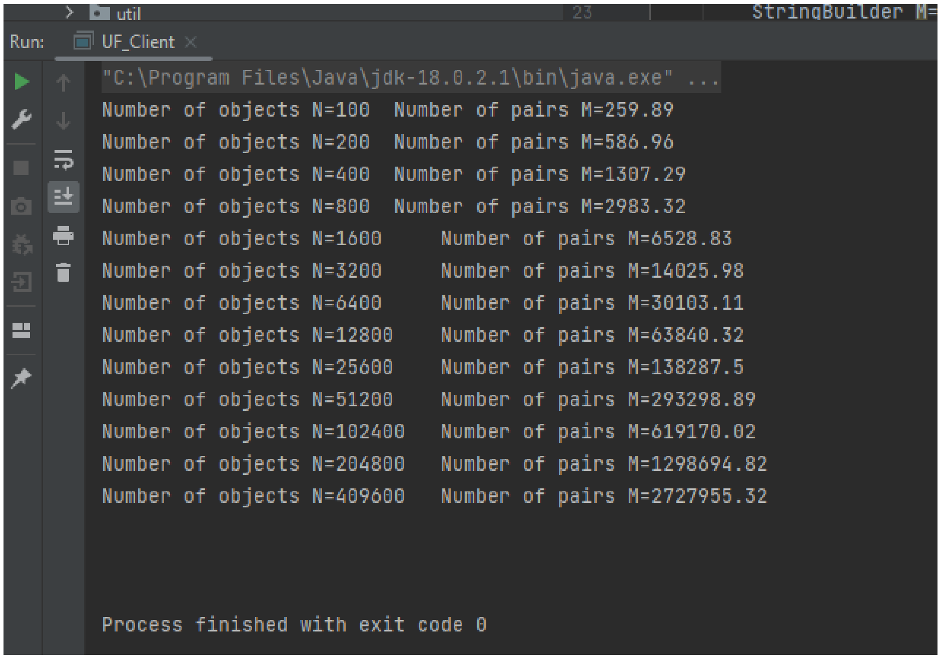


```java
private void doPathCompression(int i) {
    // FIXME update parent to value of grandparent
    while(i != parent[i]){
        parent[i] = parent[parent[i]];
        i= parent[i];
    }
    // END
}
```

UF_Client.java



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

import java.util.Random;

1 usage   new *
public class UF_Client {
    new *
    public static int count(int n) {
        UF_HWQUPC UF= new UF_HWQUPC(n);
        Random random= new Random();
        int connections =0;
        while(UF.components() > 1){
            int i= random.nextInt(n);
            int j = random.nextInt(n);
            connections++;
            if(!UF.isConnected(i, j)){
                UF.union(i, j);
            }
        }
        return connections;
    }

    new *
    public static void main(String args[]){
        StringBuilder N= new StringBuilder();
        StringBuilder M= new StringBuilder();
        for(int i=100; i<= 500000; i=2*i ){
            double mean=0.00;
            for(int j=0; j< 100; j++){
                mean+= UF_Client.count(i);
            }
            System.out.println("Number of objects N="+i
                    +"\t Number of pairs M="+ mean/100);
        }

        System.out.println(N.toString());
        System.out.println(M.toString());
    }
}
```



```
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
Number of objects N=100  Number of pairs M=259.89
Number of objects N=200  Number of pairs M=586.96
Number of objects N=400  Number of pairs M=1307.29
Number of objects N=800  Number of pairs M=2983.32
Number of objects N=1600     Number of pairs M=6528.83
Number of objects N=3200     Number of pairs M=14025.98
Number of objects N=6400     Number of pairs M=30103.11
Number of objects N=12800    Number of pairs M=63840.32
Number of objects N=25600    Number of pairs M=138287.5
Number of objects N=51200    Number of pairs M=293298.89
Number of objects N=102400   Number of pairs M=619170.02
Number of objects N=204800   Number of pairs M=1298694.82
Number of objects N=409600   Number of pairs M=2727955.32



Process finished with exit code 0
```