Zeynep Tufekci (NUID: 001565591)

INFO 6205

Program Structures & Algorithms

Fall 2020

Assignment 3

- ## Task and Observations

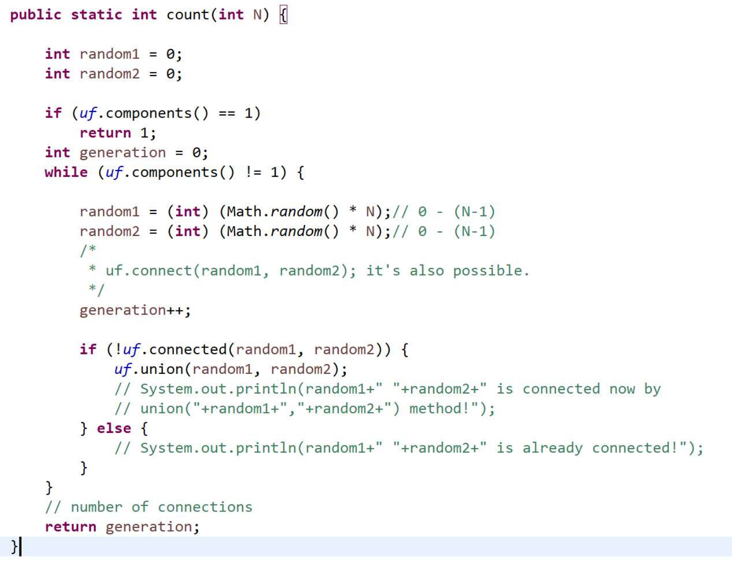
Step 1:

(a) UF_HWQUPC (Height-weighted Quick Union with Path Compression) class is implemented.

(b) Unit tests are run. Unit test screenshots are at the end of report. Tests are all pass (green).

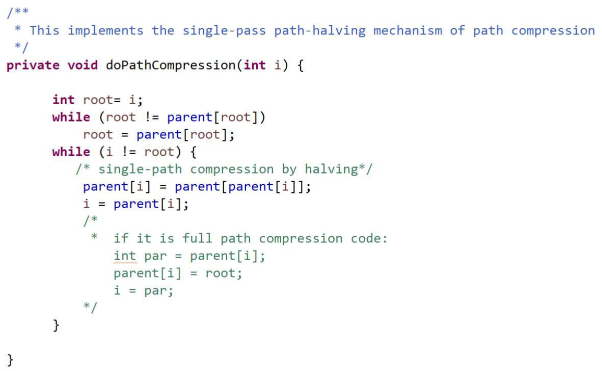
In the union method, there is a question "CONSIDER can we avoid doing find again?" Yes, we can avoid doing find by just calling parent[p] and parent[q]. isConnected() method calls find() method. find() method calls doPathCompression() method. So, after find method p and q are connected to their root. So, just reaching their parents [O(1)] is less expensive than calling find [O(lgn)] again.

Step 2: count() function and main method is written in UnionFind.java class.



```
public static int count(int N) {

    int random1 = 0;
    int random2 = 0;

    if (uf.components() == 1)
        return 1;
    int generation = 0;
    while (uf.components() != 1) {

        random1 = (int) (Math.random() * N);// 0 - (N-1)
        random2 = (int) (Math.random() * N);// 0 - (N-1)
        /*
         * uf.connect(random1, random2); it's also possible.
         */
        generation++;

        if (!uf.connected(random1, random2)) {
            uf.union(random1, random2);
            // System.out.println(random1+" "+random2+" is connected now by
            // union("+random1+","+random2+") method!");
        } else {
            // System.out.println(random1+" "+random2+" is already connected!");
        }
    }
    // number of connections
    return generation;
}
```

Zeynep Tufekci (NUID: 001565591)



```
/**
 * This implements the single-pass path-halving mechanism of path compression
 */
private void doPathCompression(int i) {

      int root= i;
      while (root != parent[root])
          root = parent[root];
      while (i != root) {
         /* single-path compression by halving*/
          parent[i] = parent[parent[i]];
          i = parent[i];
          /*
           *  if it is full path compression code:
              int par = parent[i];
              parent[i] = root;
              i = par;
          */
      }

  }
```

In path compression, there are two type of path compression. Single-pass path-halving path compression and fully path compression. While first one updates its root as a grandparent's root, second one updates sequentially all parents' root as a new root. In this assignment, I wrote first one but second one is also available in comment line.

Step 3:
The relationship between the number of objects (n) and the number of pairs (m) generated from n components to 1 component are observed. The results are shared in this report.

- **Output**

In these experiments, I repeated the weighted quick union with path compression program more than 4 times. I observe that all m values have relation with n values.

Experiment 1

44 (m) random pairs are generated for 20 (n) sites.
273 (m) random pairs are generated for 100 (n) sites.
2985 (m) random pairs are generated for 1000 (n) sites.
7880 (m) random pairs are generated for 2500 (n) sites.
13153 (m) random pairs are generated for 5000 (n) sites.
27825 (m) random pairs are generated for 10000 (n) sites.
70535 (m) random pairs are generated for 25000 (n) sites.
138706 (m) random pairs are generated for 50000 (n) sites.
283118 (m) random pairs are generated for 100000 (n) sites.

Experiment 2

18 (m) random pairs are generated for 10 (n) sites.
79 (m) random pairs are generated for 20 (n) sites.
103 (m) random pairs are generated for 50 (n) sites.
191 (m) random pairs are generated for 100 (n) sites.
2647 (m) random pairs are generated for 1000 (n) sites.
8660 (m) random pairs are generated for 2500 (n) sites.
14382 (m) random pairs are generated for 5000 (n) sites.
29282 (m) random pairs are generated for 10000 (n) sites.
72354 (m) random pairs are generated for 25000 (n) sites.
137558 (m) random pairs are generated for 50000 (n) sites.
269193 (m) random pairs are generated for 100000 (n) sites.

Experiment 3

18 (m) random pairs are generated for 10 (n) sites.
30 (m) random pairs are generated for 20 (n) sites.
101 (m) random pairs are generated for 50 (n) sites.
250 (m) random pairs are generated for 100 (n) sites.
2543 (m) random pairs are generated for 1000 (n) sites.
9539 (m) random pairs are generated for 2500 (n) sites.
14970 (m) random pairs are generated for 5000 (n) sites.
27463 (m) random pairs are generated for 10000 (n) sites.
70374 (m) random pairs are generated for 25000 (n) sites.
136171 (m) random pairs are generated for 50000 (n) sites.
273348 (m) random pairs are generated for 100000 (n) sites.

Experiment 4

27 (m) random pairs are generated for 10 (n) sites.
31 (m) random pairs are generated for 20 (n) sites.
98 (m) random pairs are generated for 50 (n) sites.
196 (m) random pairs are generated for 100 (n) sites.
2731 (m) random pairs are generated for 1000 (n) sites.
7663 (m) random pairs are generated for 2500 (n) sites.
16225 (m) random pairs are generated for 5000 (n) sites.
28866 (m) random pairs are generated for 10000 (n) sites.
71359 (m) random pairs are generated for 25000 (n) sites.
142045 (m) random pairs are generated for 50000 (n) sites.
283354 (m) random pairs are generated for 100000 (n) sites.
548783 (m) random pairs are generated for 200000 (n) sites.

- **Relationship conclusion**

The number of objects (n) and the number of pairs (m) generated are observed.
When I draw a graph using 4 different experimental results. I observed that the tangent of line is equal to Euler number $e = 2.7182818$ ... So, the formula is

$$M = e * N$$

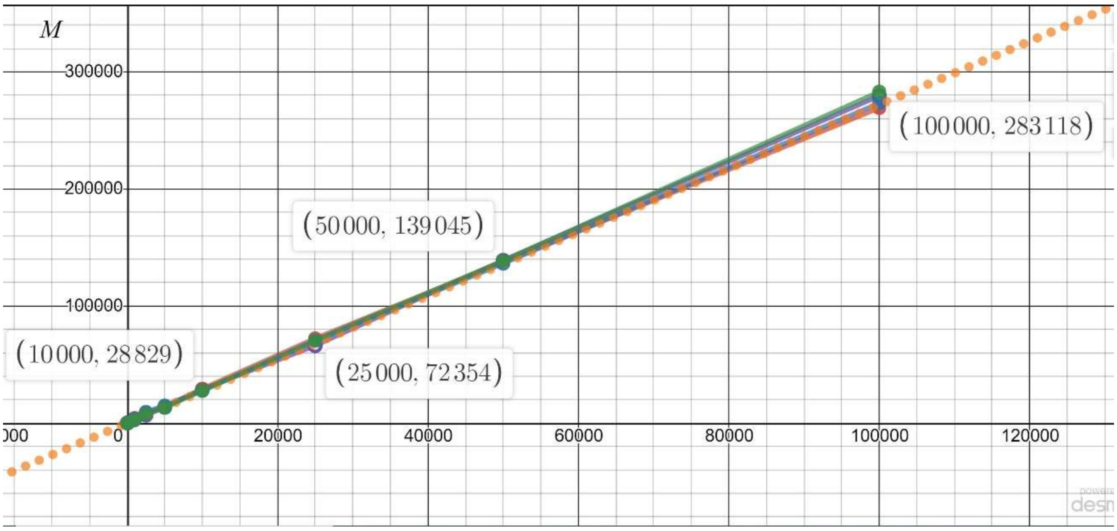
- **Evidence to support relationship**

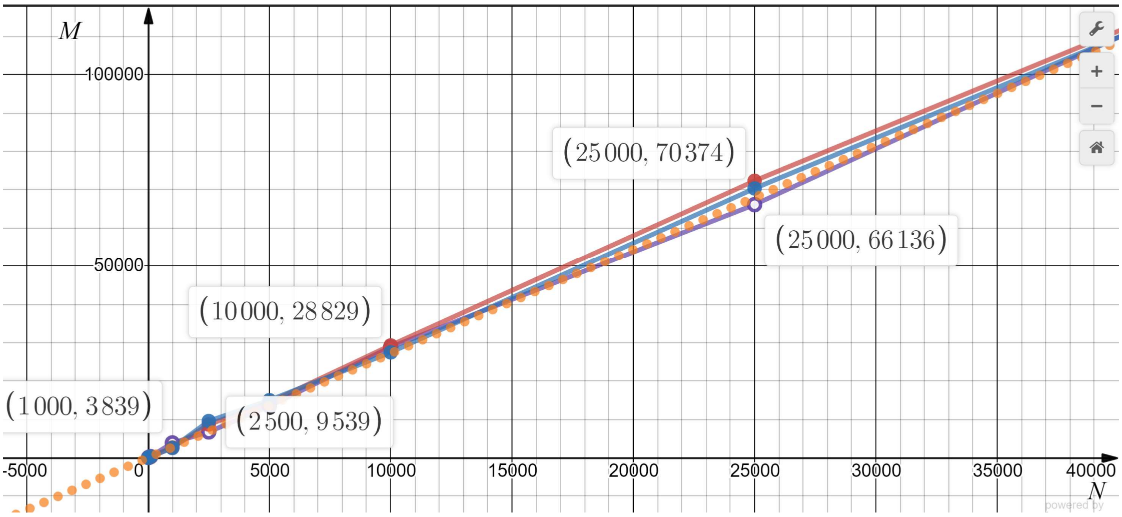
X-axis shows N number of objects and Y-axis shows M random number of pairs.

Orange dot-line shows $M = e * N$ line. 3 different experiments are shown as green, blue and red lines. Some values in the experiments are shown below.

12 different N numbers between 10 to 100,000 are taken into account. The tangent of line is stable and equals to Euler number e.

Zeynep Tufekci (NUID: 001565591)

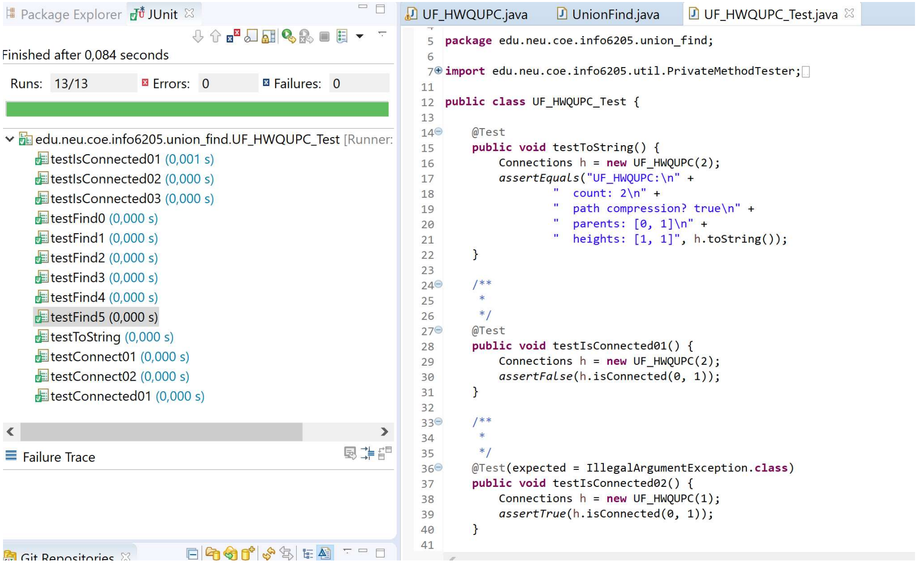




Find and Union operations repeat M times on a set of N objects. Find and Union operations takes O(lgn) time. Because of that, asymptotic notation of this code is O (N + M lg* N).

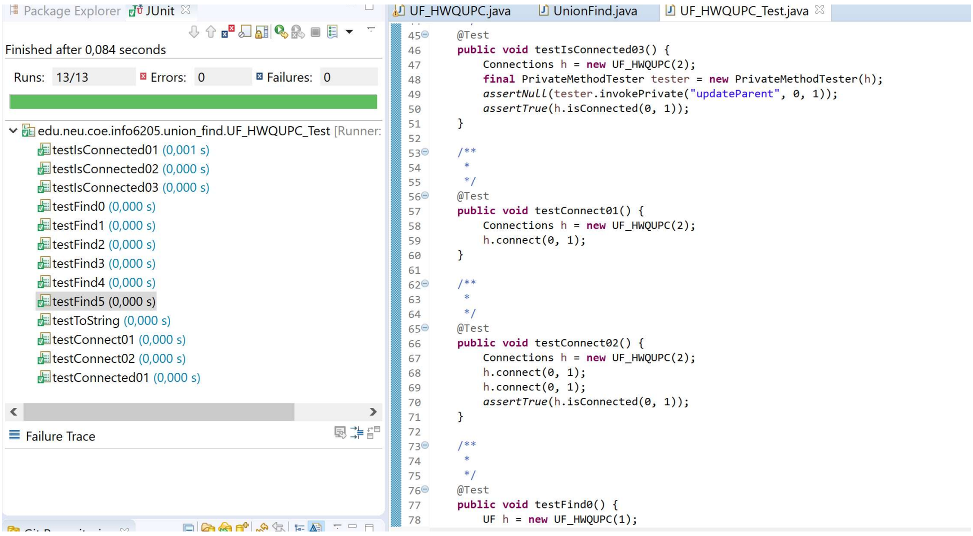
Zeynep Tufekci (NUID: 001565591)

- **Unit Test**



```
Finished after 0,084 seconds

Runs: 13/13     Errors: 0     Failures: 0

edu.neu.coe.info6205.union_find.UF_HWQUPC_Test [Runner:
    testIsConnected01 (0,001 s)
    testIsConnected02 (0,000 s)
    testIsConnected03 (0,000 s)
    testFind0 (0,000 s)
    testFind1 (0,000 s)
    testFind2 (0,000 s)
    testFind3 (0,000 s)
    testFind4 (0,000 s)
    testFind5 (0,000 s)
    testToString (0,000 s)
    testConnect01 (0,000 s)
    testConnect02 (0,000 s)
    testConnected01 (0,000 s)
```

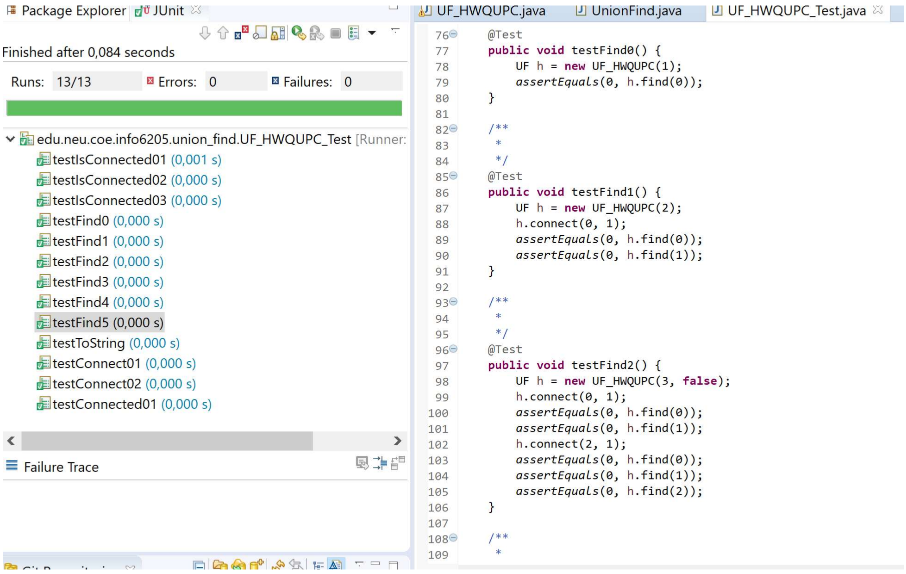
```java
package edu.neu.coe.info6205.union_find;

import edu.neu.coe.info6205.util.PrivateMethodTester;

public class UF_HWQUPC_Test {

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

    /**
     *
     */
    @Test
    public void testIsConnected01() {
        Connections h = new UF_HWQUPC(2);
        assertFalse(h.isConnected(0, 1));
    }

    /**
     *
     */
    @Test(expected = IllegalArgumentException.class)
    public void testIsConnected02() {
        Connections h = new UF_HWQUPC(1);
        assertTrue(h.isConnected(0, 1));
    }

```



```
Finished after 0,084 seconds

Runs: 13/13     Errors: 0     Failures: 0

edu.neu.coe.info6205.union_find.UF_HWQUPC_Test [Runner:
    testIsConnected01 (0,001 s)
    testIsConnected02 (0,000 s)
    testIsConnected03 (0,000 s)
    testFind0 (0,000 s)
    testFind1 (0,000 s)
    testFind2 (0,000 s)
    testFind3 (0,000 s)
    testFind4 (0,000 s)
    testFind5 (0,000 s)
    testToString (0,000 s)
    testConnect01 (0,000 s)
    testConnect02 (0,000 s)
    testConnected01 (0,000 s)
```

```java
    @Test
    public void testIsConnected03() {
        Connections h = new UF_HWQUPC(2);
        final PrivateMethodTester tester = new PrivateMethodTester(h);
        assertNull(tester.invokePrivate("updateParent", 0, 1));
        assertTrue(h.isConnected(0, 1));
    }

    /**
     *
     */
    @Test
    public void testConnect01() {
        Connections h = new UF_HWQUPC(2);
        h.connect(0, 1);
    }

    /**
     *
     */
    @Test
    public void testConnect02() {
        Connections h = new UF_HWQUPC(2);
        h.connect(0, 1);
        h.connect(0, 1);
        assertTrue(h.isConnected(0, 1));
    }

    /**
     *
     */
    @Test
    public void testFind0() {
        UF h = new UF_HWQUPC(1);
```

Zeynep Tufekci (NUID: 001565591)

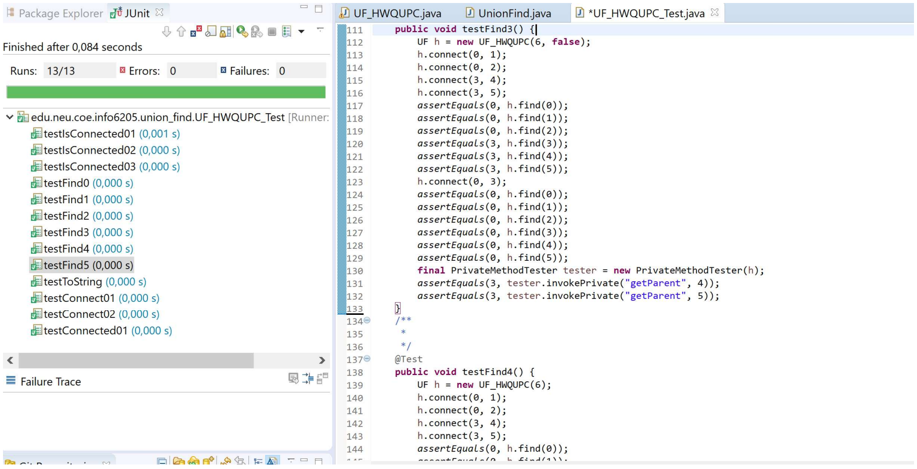


Finished after 0,084 seconds

Runs: 13/13 Errors: 0 Failures: 0

```
edu.neu.coe.info6205.union_find.UF_HWQUPC_Test [Runner:
    testIsConnected01 (0,001 s)
    testIsConnected02 (0,000 s)
    testIsConnected03 (0,000 s)
    testFind0 (0,000 s)
    testFind1 (0,000 s)
    testFind2 (0,000 s)
    testFind3 (0,000 s)
    testFind4 (0,000 s)
    testFind5 (0,000 s)
    testToString (0,000 s)
    testConnect01 (0,000 s)
    testConnect02 (0,000 s)
    testConnected01 (0,000 s)
```

```java
    @Test
    public void testFind0() {
        UF h = new UF_HWQUPC(1);
        assertEquals(0, h.find(0));
    }

    /**
     *
     */
    @Test
    public void testFind1() {
        UF h = new UF_HWQUPC(2);
        h.connect(0, 1);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
    }

    /**
     *
     */
    @Test
    public void testFind2() {
        UF h = new UF_HWQUPC(3, false);
        h.connect(0, 1);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
        h.connect(2, 1);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
        assertEquals(0, h.find(2));
    }

    /**
     *
```

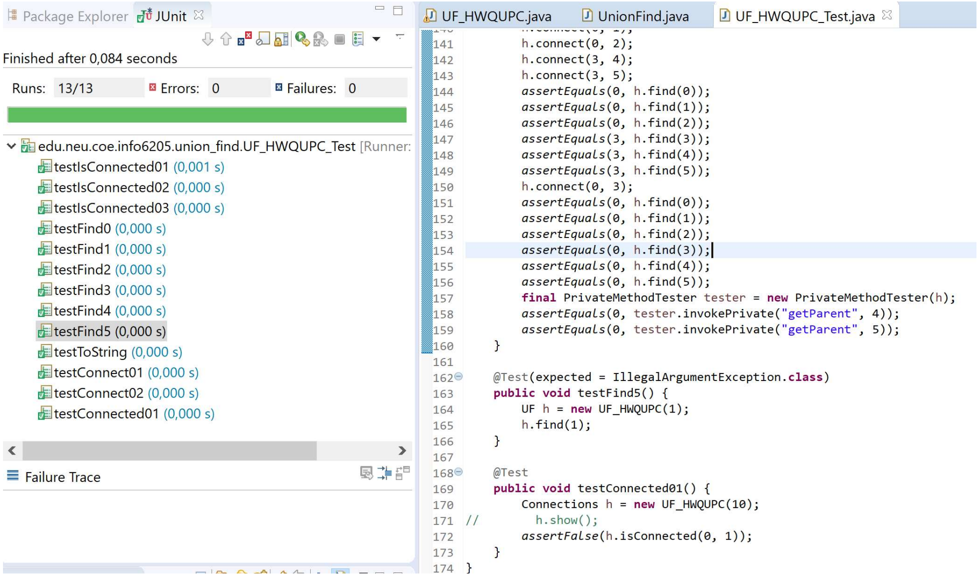


Finished after 0,084 seconds

Runs: 13/13 Errors: 0 Failures: 0

```
edu.neu.coe.info6205.union_find.UF_HWQUPC_Test [Runner:
    testIsConnected01 (0,001 s)
    testIsConnected02 (0,000 s)
    testIsConnected03 (0,000 s)
    testFind0 (0,000 s)
    testFind1 (0,000 s)
    testFind2 (0,000 s)
    testFind3 (0,000 s)
    testFind4 (0,000 s)
    testFind5 (0,000 s)
    testToString (0,000 s)
    testConnect01 (0,000 s)
    testConnect02 (0,000 s)
    testConnected01 (0,000 s)
```

```java
    public void testFind3() {
        UF h = new UF_HWQUPC(6, false);
        h.connect(0, 1);
        h.connect(0, 2);
        h.connect(3, 4);
        h.connect(3, 5);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
        assertEquals(0, h.find(2));
        assertEquals(3, h.find(3));
        assertEquals(3, h.find(4));
        assertEquals(3, h.find(5));
        h.connect(0, 3);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
        assertEquals(0, h.find(2));
        assertEquals(0, h.find(3));
        assertEquals(0, h.find(4));
        assertEquals(0, h.find(5));
        final PrivateMethodTester tester = new PrivateMethodTester(h);
        assertEquals(3, tester.invokePrivate("getParent", 4));
        assertEquals(3, tester.invokePrivate("getParent", 5));
    }
    /**
     *
     */
    @Test
    public void testFind4() {
        UF h = new UF_HWQUPC(6);
        h.connect(0, 1);
        h.connect(0, 2);
        h.connect(3, 4);
        h.connect(3, 5);
        assertEquals(0, h.find(0));
```



Finished after 0,084 seconds

Runs: 13/13 Errors: 0 Failures: 0

```
edu.neu.coe.info6205.union_find.UF_HWQUPC_Test [Runner:
    testIsConnected01 (0,001 s)
    testIsConnected02 (0,000 s)
    testIsConnected03 (0,000 s)
    testFind0 (0,000 s)
    testFind1 (0,000 s)
    testFind2 (0,000 s)
    testFind3 (0,000 s)
    testFind4 (0,000 s)
    testFind5 (0,000 s)
    testToString (0,000 s)
    testConnect01 (0,000 s)
    testConnect02 (0,000 s)
    testConnected01 (0,000 s)
```

```java
        h.connect(0, 2);
        h.connect(3, 4);
        h.connect(3, 5);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
        assertEquals(0, h.find(2));
        assertEquals(3, h.find(3));
        assertEquals(3, h.find(4));
        assertEquals(3, h.find(5));
        h.connect(0, 3);
        assertEquals(0, h.find(0));
        assertEquals(0, h.find(1));
        assertEquals(0, h.find(2));
        assertEquals(0, h.find(3));
        assertEquals(0, h.find(4));
        assertEquals(0, h.find(5));
        final PrivateMethodTester tester = new PrivateMethodTester(h);
        assertEquals(0, tester.invokePrivate("getParent", 4));
        assertEquals(0, tester.invokePrivate("getParent", 5));
    }

    @Test(expected = IllegalArgumentException.class)
    public void testFind5() {
        UF h = new UF_HWQUPC(1);
        h.find(1);
    }

    @Test
    public void testConnected01() {
        Connections h = new UF_HWQUPC(10);
//          h.show();
        assertFalse(h.isConnected(0, 1));
    }
}
```