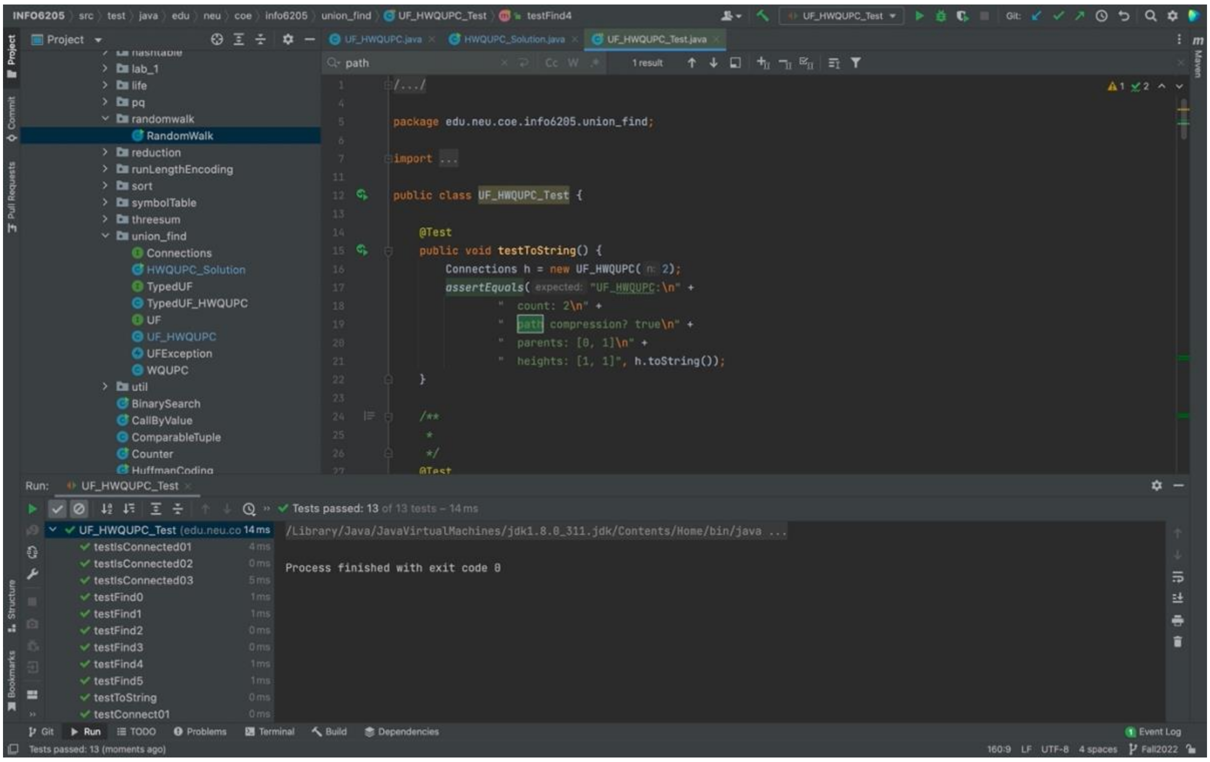
Assignment 3

Weighted Quick Union with Path Compression

Unit Tests



```java
/.../

package edu.neu.coe.info6205.union_find;

import ...

public class UF_HWQUPC_Test {

    @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());
    }

    /**
     *
     */
    @Test
```

Run: UF_HWQUPC_Test

Tests passed: 13 of 13 tests – 14 ms

```
UF_HWQUPC_Test (edu.neu.co 14 ms
    testIsConnected01    4 ms
    testIsConnected02    0 ms
    testIsConnected03    5 ms
    testFind0            1 ms
    testFind1            1 ms
    testFind2            0 ms
    testFind3            0 ms
    testFind4            1 ms
    testFind5            1 ms
    testToString         0 ms
    testConnect01        0 ms
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_311.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Tests passed: 13 (moments ago)

Conclusion

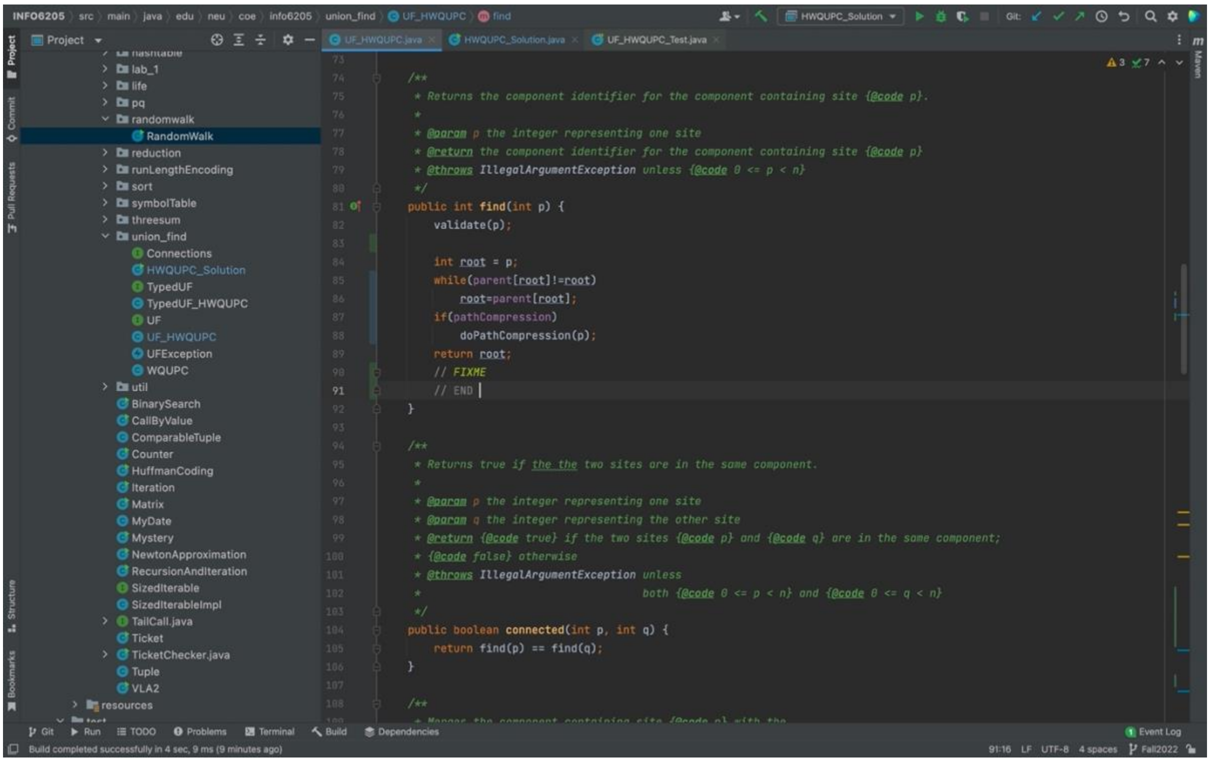
I initially noticed that the number of connections to make the number of components 1 is n-1.

I also tested it out for values of n up to 100,000.

This makes sense when we think about the fact that for any component we have, we can rearrange the nodes by connecting every child node to the root. This is similar to what happens during path compression and doesn’t change the total number of connections in any way.

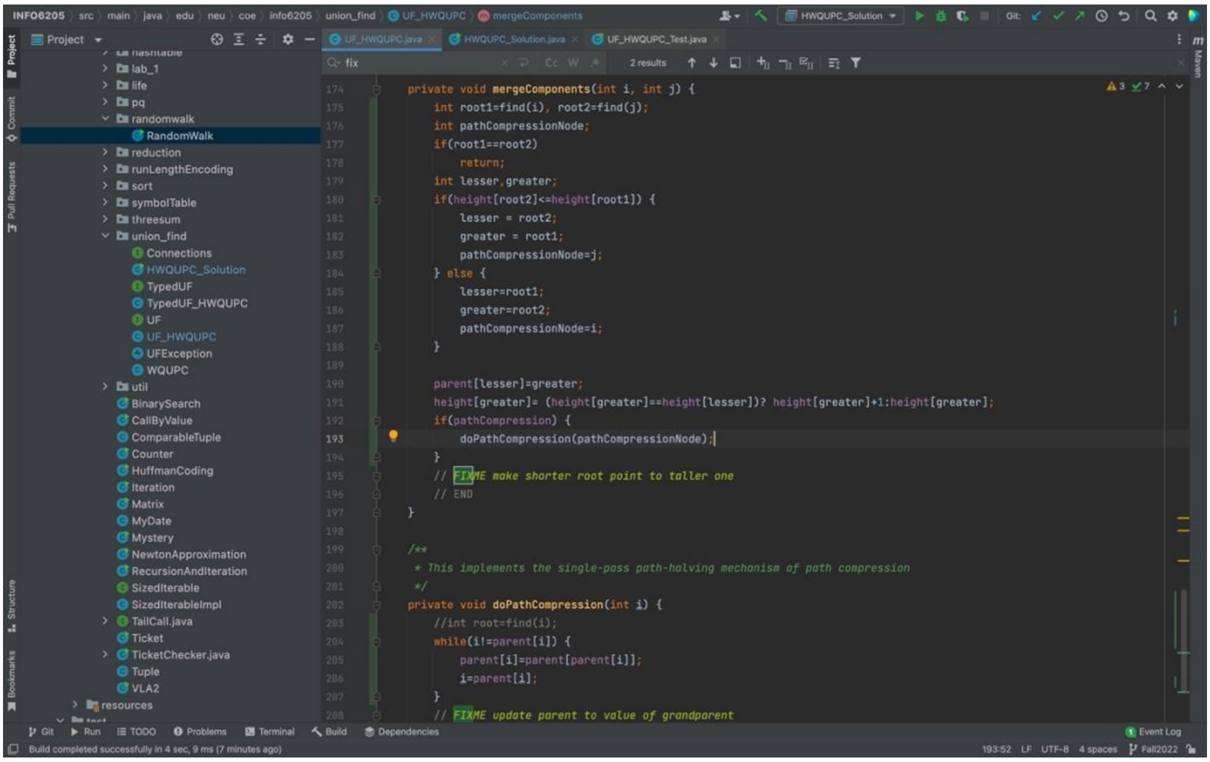
So, we can reimagine any final component we have as a root connected to the rest of the nodes (n-1).

Source Code



```java

    /**
     * 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;
        while(parent[root]!=root)
            root=parent[root];
        if(pathCompression)
            doPathCompression(p);
        return root;
        // FIXME
        // END
    }

    /**
     * Returns true if the the two sites are in the same component.
     *
     * @param p the integer representing one site
     * @param q the integer representing the other site
     * @return {@code true} if the two sites {@code p} and {@code q} are in the same component;
     * {@code false} otherwise
     * @throws IllegalArgumentException unless
     *                                  both {@code 0 <= p < n} and {@code 0 <= q < n}
     */
    public boolean connected(int p, int q) {
        return find(p) == find(q);
    }

    /**
     * Merges the component containing site {@code p} with the
```

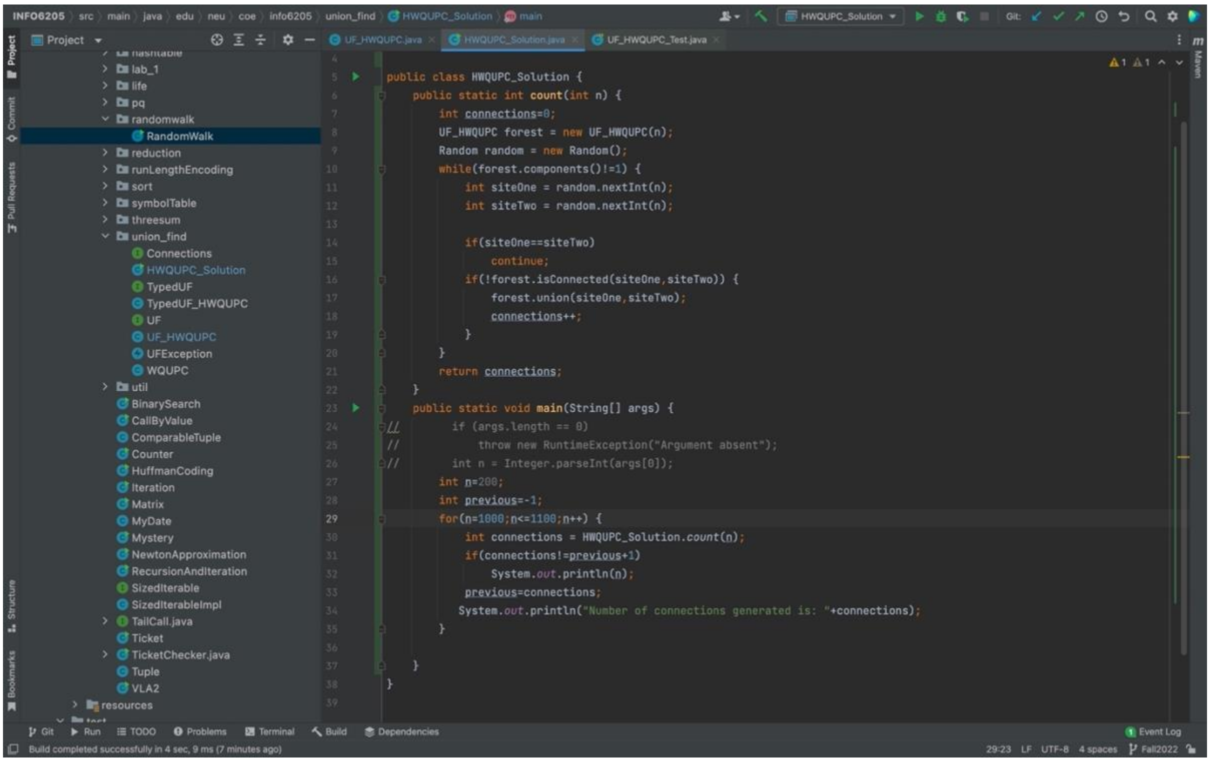
Build completed successfully in 4 sec, 9 ms (9 minutes ago)



```java
    private void mergeComponents(int i, int j) {
        int root1=find(i), root2=find(j);
        int pathCompressionNode;
        if(root1==root2)
            return;
        int lesser,greater;
        if(height[root2]<=height[root1]) {
            lesser = root2;
            greater = root1;
            pathCompressionNode=j;
        } else {
            lesser=root1;
            greater=root2;
            pathCompressionNode=i;
        }

        parent[lesser]=greater;
        height[greater]= (height[greater]==height[lesser])? height[greater]+1:height[greater];
        if(pathCompression) {
            doPathCompression(pathCompressionNode);
        }
        // FIXME make shorter root point to taller one
        // END
    }

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

Build completed successfully in 4 sec, 9 ms (7 minutes ago)

UF Client



```java
public class HWQUPC_Solution {
    public static int count(int n) {
        int connections=0;
        UF_HWQUPC forest = new UF_HWQUPC(n);
        Random random = new Random();
        while(forest.components()!=1) {
            int siteOne = random.nextInt(n);
            int siteTwo = random.nextInt(n);

            if(siteOne==siteTwo)
                continue;
            if(!forest.isConnected(siteOne,siteTwo)) {
                forest.union(siteOne,siteTwo);
                connections++;
            }
        }
        return connections;
    }
    public static void main(String[] args) {
//        if (args.length == 0)
//            throw new RuntimeException("Argument absent");
//        int n = Integer.parseInt(args[0]);
        int n=200;
        int previous=-1;
        for(n=1000;n<=1100;n++) {
            int connections = HWQUPC_Solution.count(n);
            if(connections!=previous+1)
                System.out.println(n);
            previous=connections;
           System.out.println("Number of connections generated is: "+connections);
        }

    }
}
```

Build completed successfully in 4 sec, 9 ms (7 minutes ago)