Narendra Raj R K (001553969)

Program Structures & Algorithms

Spring 21

Assignment No. 3

Tasks:

Part 1: In this task I am to implement Height Weighted Quick Union with Path Compression by adding logic to the following methods: **find(), mergeComponents() & doPathCompression()**.

- ⇨ Successfully added the logic to the methods required in order to achieve Height Weighted Quick Union with Path Compression. All the unit tests for this class passed.

Part 2: Develop a **UF ("union-find") client** that takes an integer value n from the command line to determine the number of "sites". Finally arrive on the number of connections generated.

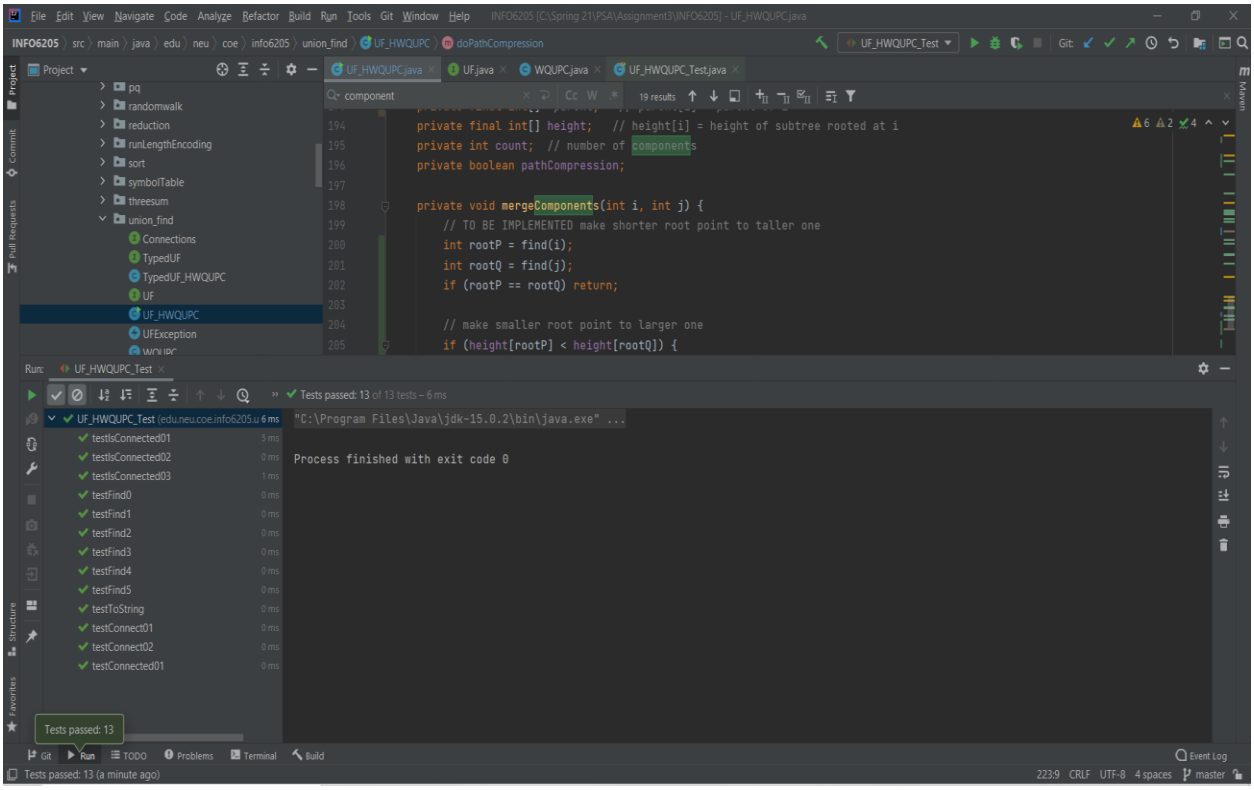
- ⇨ Generated a random pair of integers and called the connect() which iterates until all pairs are connected and finally returned the number of connections "m" value.

Part 3: Determine the relationship between the number of objects (***n***) and the number of pairs (***m***).

⇨ In order to establish a relation between the number of objects (n) and the number of pairs (m) I reduced the size of the component from size of n to 1.

Unit Test Results:

13 test cases under the UF_HWQUPC_Test.java file passed successfully.



Snippets of Methods Implemented:

I. find()

```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;
    // TO BE IMPLEMENTED
    while (root != parent[root])
        root = parent[root];

    if(pathCompression) {
        doPathCompression(p);
        return root;
    }
    else
         return root;
}
```

II. doPathCompression()

```java
/**
 * This implements the single-pass path-halving mechanism of path compression
 */
private void doPathCompression(int n) {
    // TO BE IMPLEMENTED update parent to value of grandparent
        parent[n] = parent[parent[n]];
}
```

III. mergeComponents()

```java
private void mergeComponents(int i, int j) {
    // TO BE IMPLEMENTED make shorter root point to taller one
    int rootP = find(i);
    int rootQ = find(j);
    if (rootP == rootQ) return;

    // make smaller root point to larger one
    if (height[rootP] < height[rootQ]) {
        parent[rootP] = rootQ;
    }
    //If Height is equal then point root of one component to another
    else if(height[rootP] == height[rootQ]){
        parent[rootQ] = rootP;
        height[rootP]++;
    }
    else {
        parent[rootQ] = rootP;
    }
}
```

IV. generatePairs()

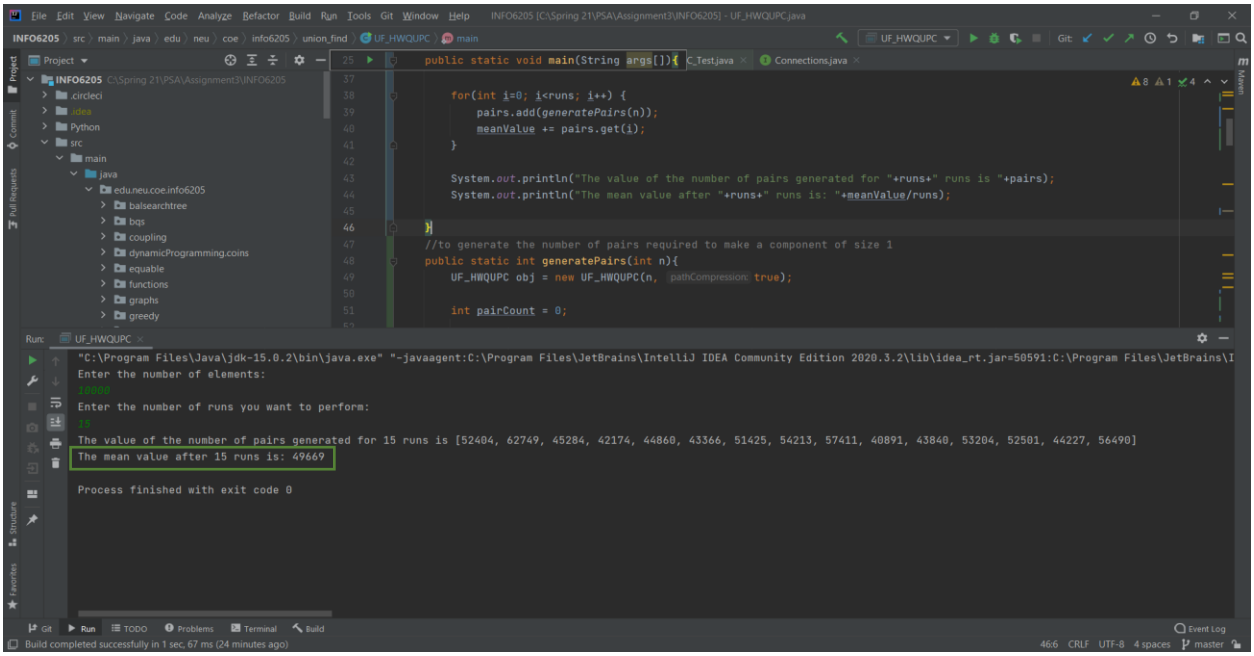
```java
//to generate the number of pairs required to make a component of size 1
public static int generatePairs(int n){
    UF_HWQUPC obj = new UF_HWQUPC(n, pathCompression: true);

    int pairCount = 0;

        while (obj.components() != 1) {
            int pair_val1 = ThreadLocalRandom.current().nextInt( origin: 0, n);
            int pair_val2 = ThreadLocalRandom.current().nextInt( origin: 0, n);
            obj.connect(pair_val1, pair_val2);
            pairCount++;
        }
        return pairCount;
}
```

Note: The above method was added by me to generate the number of pairs.

Evidence to Support Conclusion:

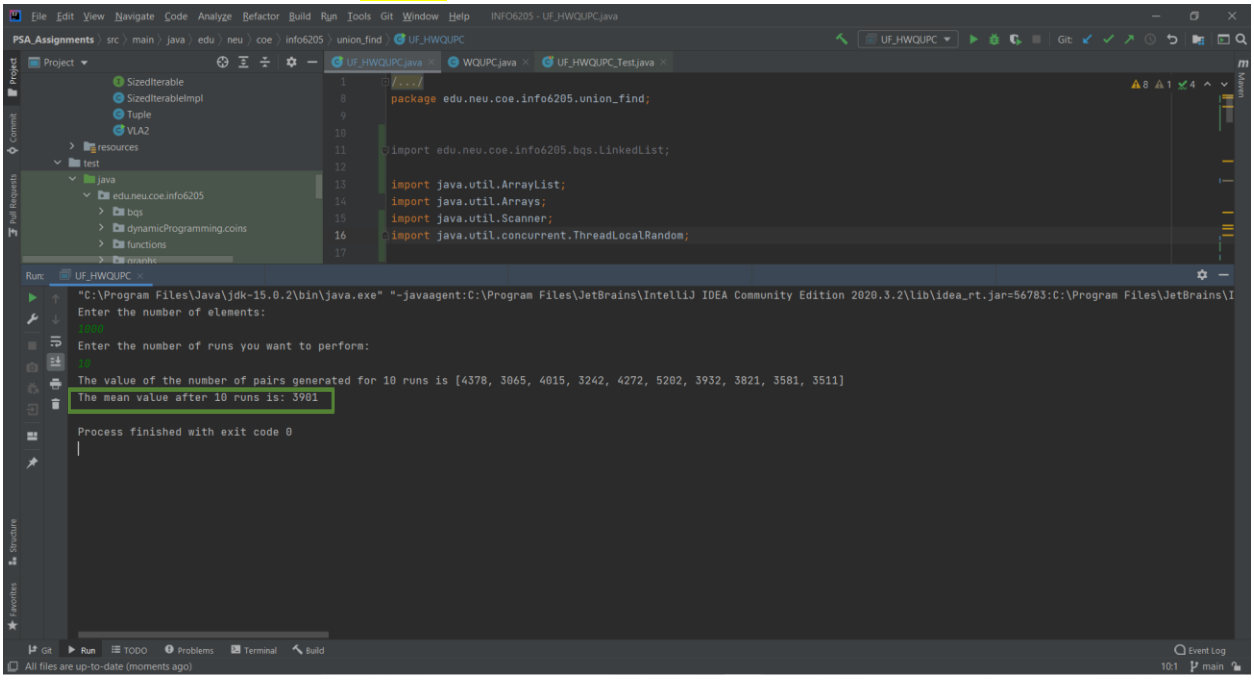
Attempt 1: Number of Elements is 10000 & the Number of Runs is 15

➔ Number of Pairs: **49669**



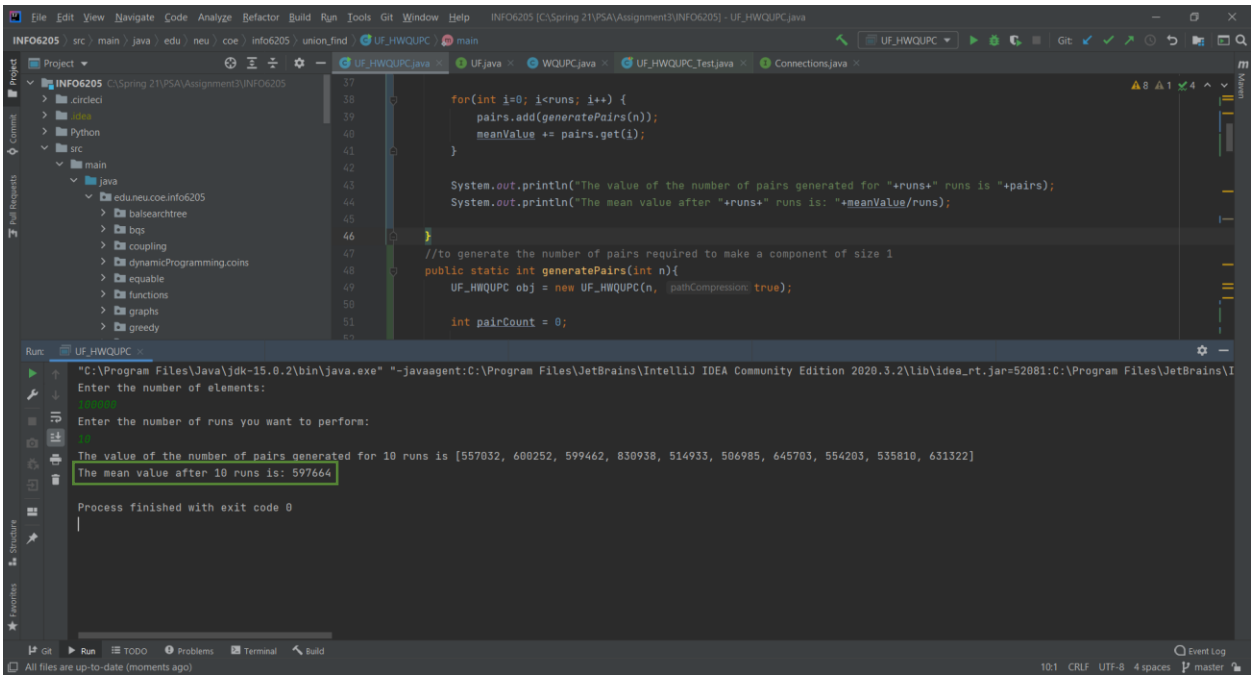
Attempt 2: Number of Elements is 1000 & the Number of Runs is 10

➔ Number of Pairs: **3901**



Attempt 3: Number of Elements is 100000 & the Number of Runs is 10

➔ Number of Pairs: **597664**



Final Conclusion:

After running the Path Compression logic on Height Weighted Quick Union multiple times and getting the mean value since we are dealing with random pairs of integers, I have arrived at the following relation between the number of elements (n) and the number of pairs (m).

$$M = (N * \ln N) / 2$$

ln – It is the Natural Logarithm with base e whose value is **2.718281828459**