Ankita Senapati (001003695)

Program Structures & Algorithms

Fall 2021

Assignment No. 3

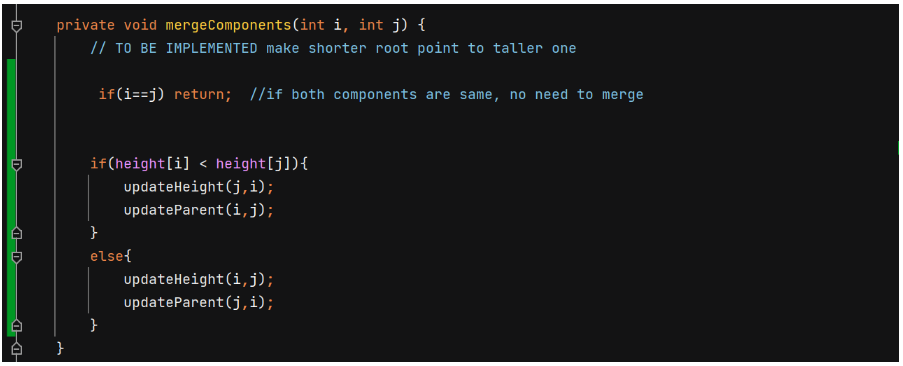
- ## Task 1

- (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.

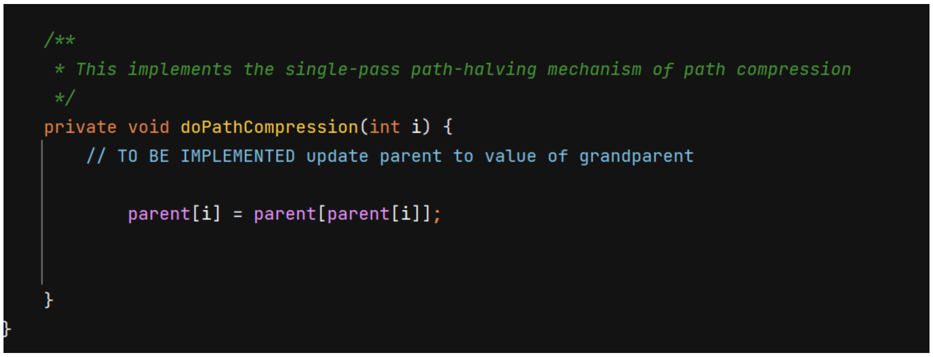
- (b) Check that the unit tests for this class all work. You must show "green" test results in your submission (screenshot is OK).

Output:

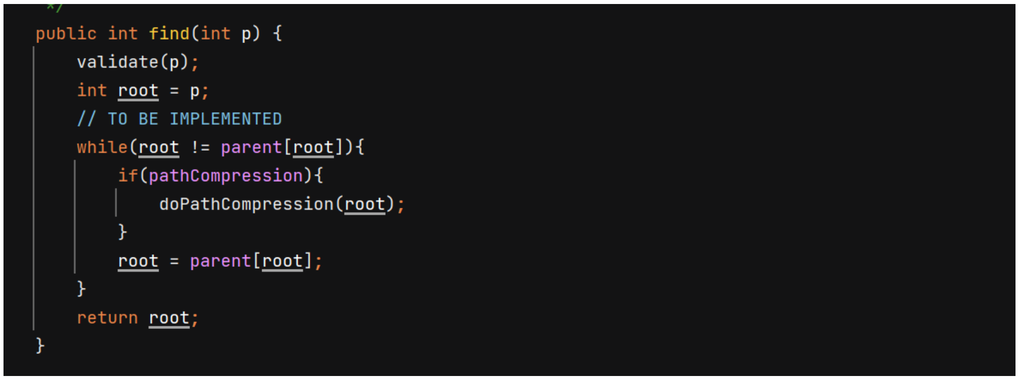
(a)



```
private void mergeComponents(int i, int j) {
    // TO BE IMPLEMENTED make shorter root point to taller one

     if(i==j) return;  //if both components are same, no need to merge


    if(height[i] < height[j]){
        updateHeight(j,i);
        updateParent(i,j);
    }
    else{
        updateHeight(i,j);
        updateParent(j,i);
    }
}
```

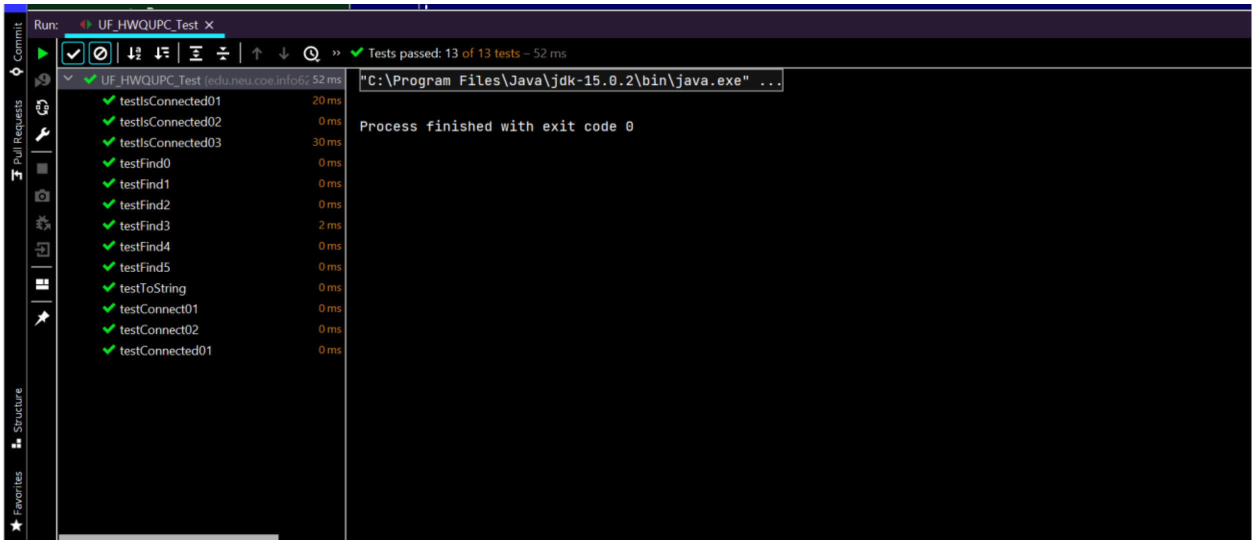


```
/**
 * This implements the single-pass path-halving mechanism of path compression
 */
private void doPathCompression(int i) {
    // TO BE IMPLEMENTED update parent to value of grandparent

        parent[i] = parent[parent[i]];


}
}
```



```
    */
public int find(int p) {
    validate(p);
    int root = p;
    // TO BE IMPLEMENTED
    while(root != parent[root]){
        if(pathCompression){
            doPathCompression(root);
        }
        root = parent[root];
    }
    return root;
}
```

(b)

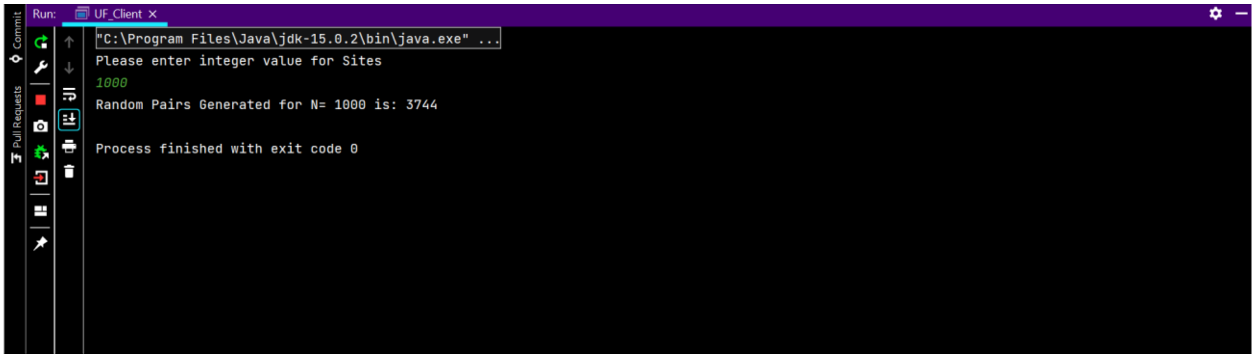


```
✓ Tests passed: 13 of 13 tests – 52 ms

✓ UF_HWQUPC_Test (edu.neu.coe.info62 52 ms
    ✓ testIsConnected01      20 ms
    ✓ testIsConnected02       0 ms
    ✓ testIsConnected03      30 ms
    ✓ testFind0               0 ms
    ✓ testFind1               0 ms
    ✓ testFind2               0 ms
    ✓ testFind3               2 ms
    ✓ testFind4               0 ms
    ✓ testFind5               0 ms
    ✓ testToString            0 ms
    ✓ testConnect01           0 ms
    ✓ testConnect02           0 ms
    ✓ testConnected01         0 ms
```

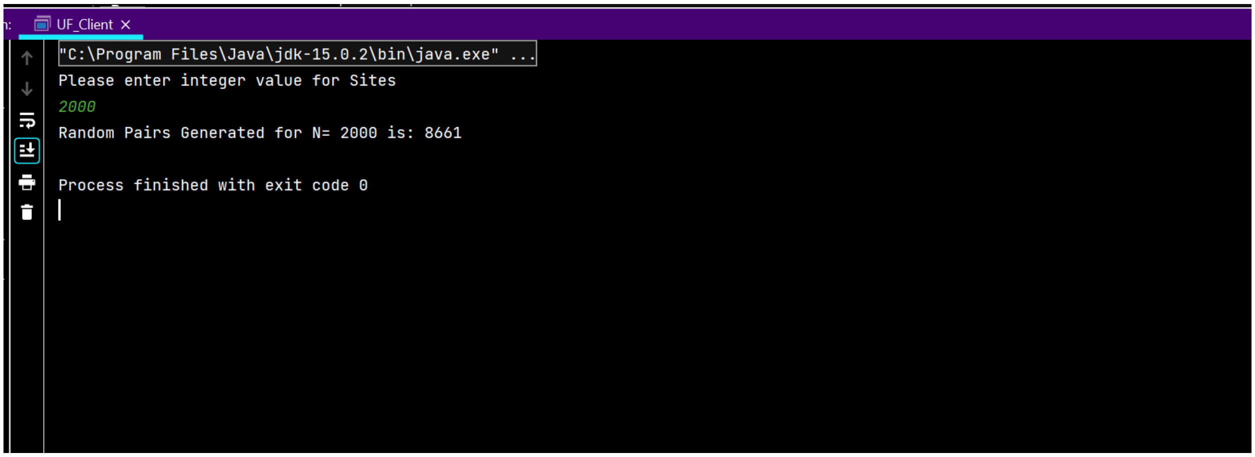
```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" ...

Process finished with exit code 0
```

⊙ Task 2

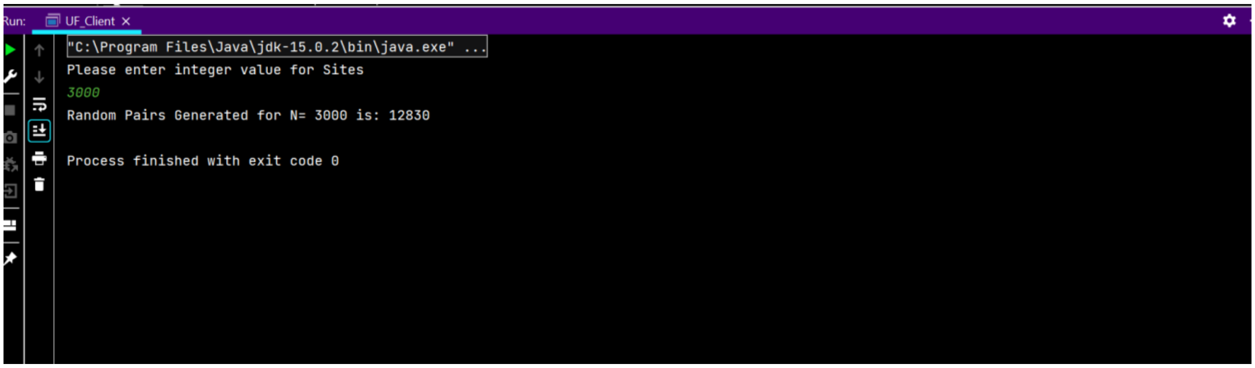
Using your implementation of UF_HWQUPC, develop a UF ("union-find") client that takes an integer value n from the command line to determine the number of "sites." Then generates random pairs of integers between 0 and n-1, calling connected() to determine if they are connected and union() if not. Loop 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).



```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" ...
Please enter integer value for Sites
1000
Random Pairs Generated for N= 1000 is: 3744

Process finished with exit code 0
```



```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" ...
Please enter integer value for Sites
2000
Random Pairs Generated for N= 2000 is: 8661

Process finished with exit code 0
```



```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" ...
Please enter integer value for Sites
3000
Random Pairs Generated for N= 3000 is: 12830

Process finished with exit code 0
```

⊙ Task 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.

⊙ Relationship Conclusion:

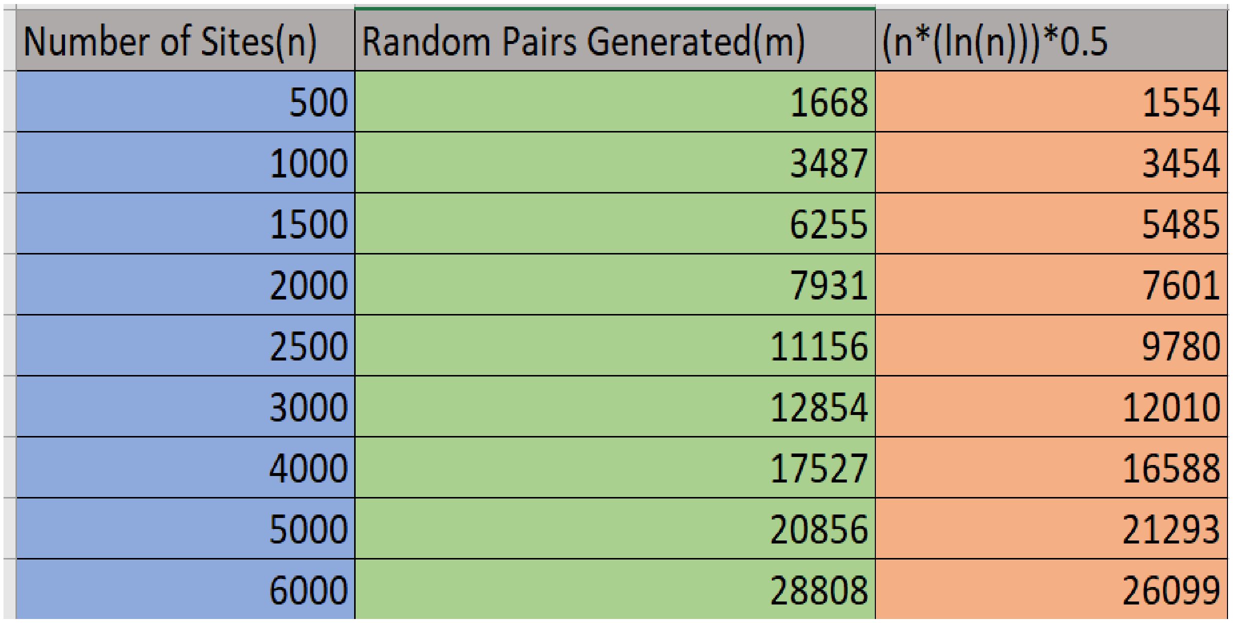
From the varied experiment runs conducted, I came to a conclusion that the number of generated pairs(m), given the number of sites (n) is approximately equal to **(n * ln(n) * 0.5)** barring the minor variance.

m ≈ (n * ln(n) * 0.5)

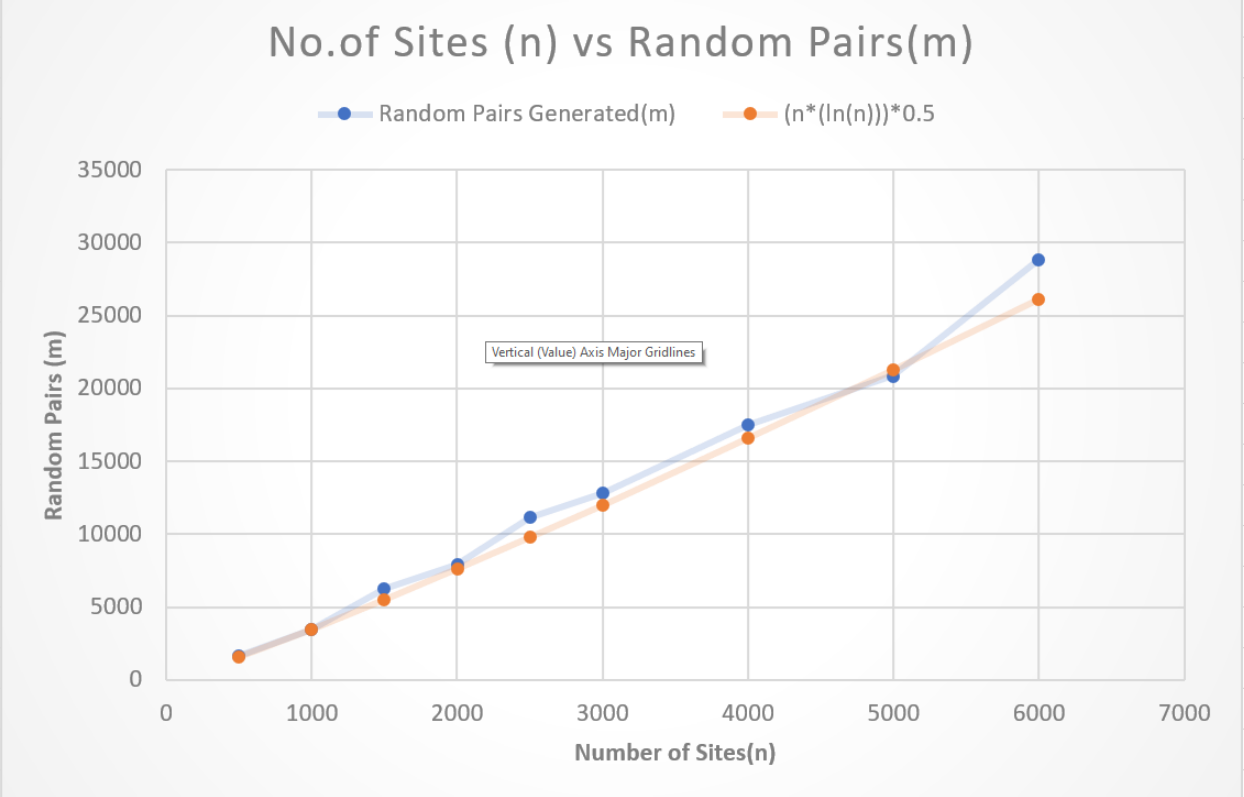
⊙ Evidence to support the conclusion:

I have documented the results of the experiment in excel sheet by noting the values of m, n and n*ln(n)*0.5. Through this data, I plotted a graph which provides evidence to the relationship conclusion that I came to.

The data shows that Weighted Quick Union with Path Compression shows a near linear growth.



Number of Sites(n)	Random Pairs Generated(m)	(n*(ln(n)))*0.5
500	1668	1554
1000	3487	3454
1500	6255	5485
2000	7931	7601
2500	11156	9780
3000	12854	12010
4000	17527	16588
5000	20856	21293
6000	28808	26099


No.of Sites (n) vs Random Pairs(m)
Random Pairs Generated(m)
(n*(ln(n)))*0.5
Vertical (Value) Axis Major Gridlines
Random Pairs (m)
35000
30000
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000
5000
6000
7000
Number of Sites(n)