

Urvi V. Aryamane (001040582)
Program Structures and Algorithms
Spring 2021(SEC 05)

Task:

Step 1:

- (a) Implement height-weighted Quick Union with Path Compression.
- (b) Check that the unit tests for this class all work.

Step 2:

Using the 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.

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).

Relationship Conclusion:

$$M \approx (n/2) * \ln(n)$$

Where,

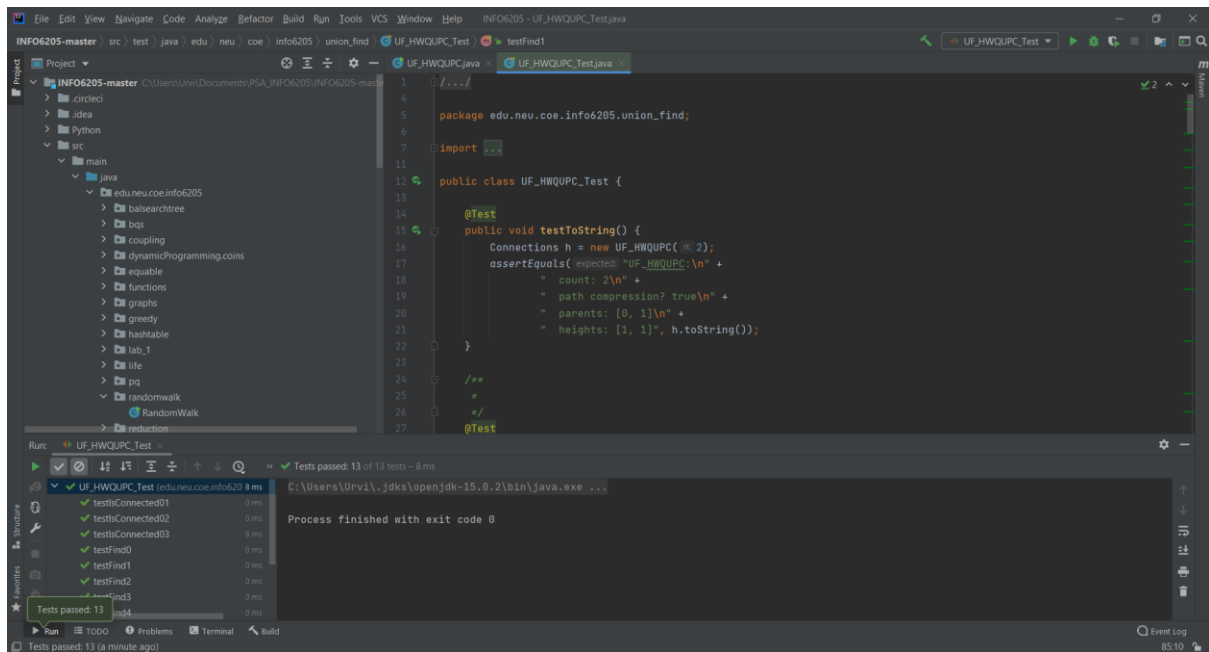
M = number of pairs generated

N = number of nodes

Evidence to support that conclusion:

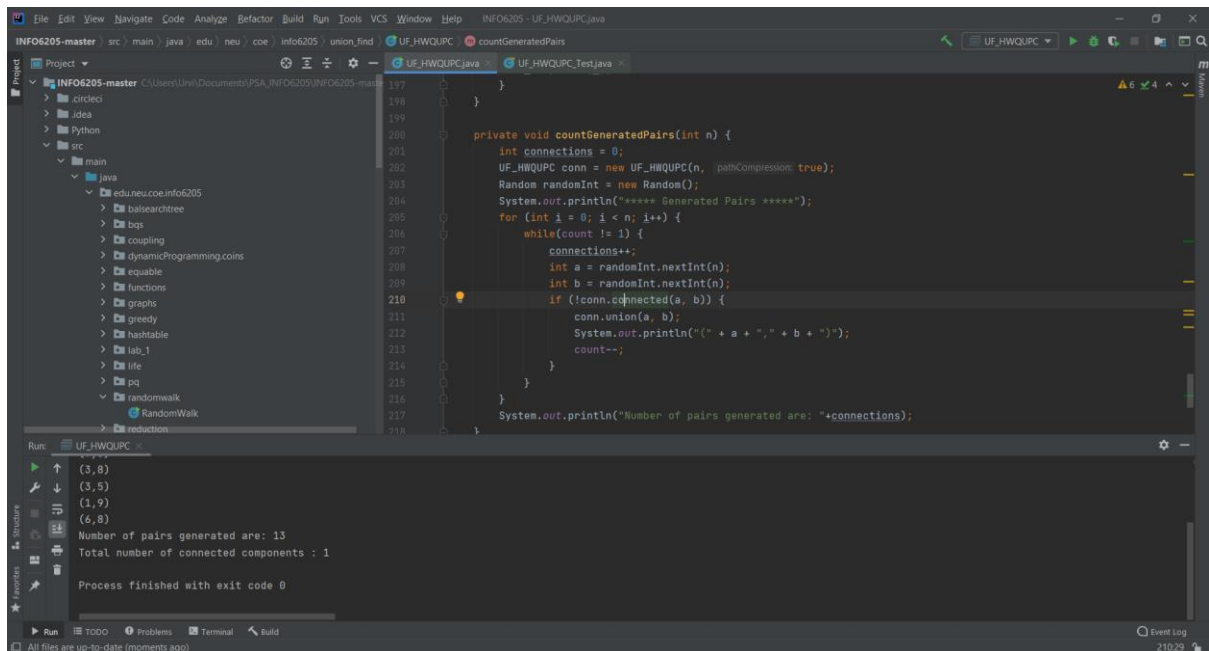
	A	B	C	D
1	Number of nodes(n)	Total pairs generated	$\ln(n)$	$n/2 * \ln(n)$
2	10	17	2.3	11.5
3	100	243	4.6	230
4	500	2089	6.21	1552
5	1000	4260	6.9	3450
6	10000	61761	9.2	46000
7	100000	688406	11.5	575000
8	1000000	6801536	13.81	6905000
9				

Unit Test Screenshots



Output Screenshots

For n = 10



For $n = 100$

```
197 }
198
199
200 private void countGeneratedPairs(int n) {
201     int connections = 0;
202     UF_HWQUPC conn = new UF_HWQUPC(n, pathCompression: true);
203     Random randomInt = new Random();
204     System.out.println("***** Generated Pairs *****");
205     for (int i = 0; i < n; i++) {
206         while(count != 1) {
207             connections++;
208             int a = randomInt.nextInt(n);
209             int b = randomInt.nextInt(n);
210             if (!conn.isConnected(a, b)) {
211                 conn.union(a, b);
212                 System.out.println("(" + a + ", " + b + ")");
213                 count--;
214             }
215         }
216     }
217     System.out.println("Number of pairs generated are: " + connections);
218 }
```

Run: UF_HWQUPC

(49,3)
(17,65)
(5,17)
(32,79)
Number of pairs generated are: 285
Total number of connected components : 1
Process finished with exit code 0

For $n = 1000$

```
197 }
198
199
200 private void countGeneratedPairs(int n) {
201     int connections = 0;
202     UF_HWQUPC conn = new UF_HWQUPC(n, pathCompression: true);
203     Random randomInt = new Random();
204     System.out.println("***** Generated Pairs *****");
205     for (int i = 0; i < n; i++) {
206         while(count != 1) {
207             connections++;
208             int a = randomInt.nextInt(n);
209             int b = randomInt.nextInt(n);
210             if (!conn.isConnected(a, b)) {
211                 conn.union(a, b);
212                 System.out.println("(" + a + ", " + b + ")");
213                 count--;
214             }
215         }
216     }
217     System.out.println("Number of pairs generated are: " + connections);
218 }
```

Run: UF_HWQUPC

(414,719)
(126,852)
(51,489)
(943,72)
Number of pairs generated are: 3342
Total number of connected components : 1
Process finished with exit code 0