

**Program structures and algorithms
Spring 2023 (Section-01)**

BY: PRANAV KAPOOR – NUID: 002998253

Assignment 4 (WQUPC)

Step 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).

Step 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).

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). Justify your conclusion in terms of your observations and what you think might be going on.

Solution:

```
... public int find(int p) {  
...     validate(p);  
...  
...     // FIXME  
...  
...     int n=p;  
...     while(p!=parent[n]) {  
...         n=parent[n];  
...     }  
...     if(pathCompression) {  
...         while(p!=n) {  
...             int next=parent[p];  
...             parent[p]=n;  
...             p=next;  
...         }  
...     }  
...     // END  
...     return n;  
... }  
...
```

```
... private void mergeComponents(int i, int j) {  
...     // FIXME make shorter root point to taller one  
...  
...     int rootI=find(i);  
...     int rootJ=find(j);  
...  
...     if (rootI==rootJ) {  
...         return;  
...     }  
...  
...     if (height[rootI]<height[rootJ]) {  
...         parent[rootI]=rootJ;  
...     } else if (height[rootI]>height[rootJ]) {  
...         parent[rootJ]=rootI;  
...     } else {  
...         parent[rootJ]=rootI;  
...         height[rootI]++;  
...     }  
...     // END  
... }  
...  
... private void doPathCompression(int i) {  
...     // FIXME update parent to value of grandparent  
...  
...     int node=find(i);  
...     while(i!=node) {  
...         int next=parent[i];  
...         parent[i]=node;  
...         i=next;  
...     }  
...     // END  
... }
```

```

public class HWQUPC_Solution {
    private static int solve(int n) {
        if (n <= 0) {
            throw new IllegalArgumentException("n must be a positive integer");
        }

        int no_connections = 0;
        int no_pairs = 0;

        UF_HWQUPC uf = new UF_HWQUPC(n);
        while (uf.components() != 1) {
            int x = (int) (Math.random() * (n));
            int y = (int) (Math.random() * (n));
            if (x == y) {
                continue;
            }
            no_pairs += 1;
            if (!uf.connected(x, y)) {
                uf.union(x, y);
                no_connections++;
            }
        }

        System.out.println("Number of connections: " + no_connections);
        System.out.println("Number of pairs: " + no_pairs);

        return no_pairs;
    }

    public static void main(String[] args) {
        int n = 408;
        int numTrials = 7;

        for (int i = 0; i < numTrials; i++) {
            System.out.println("Number of nodes: " + n);
            solve(n);


            n *= 2;
        }
    }
}

```

Unit Tests

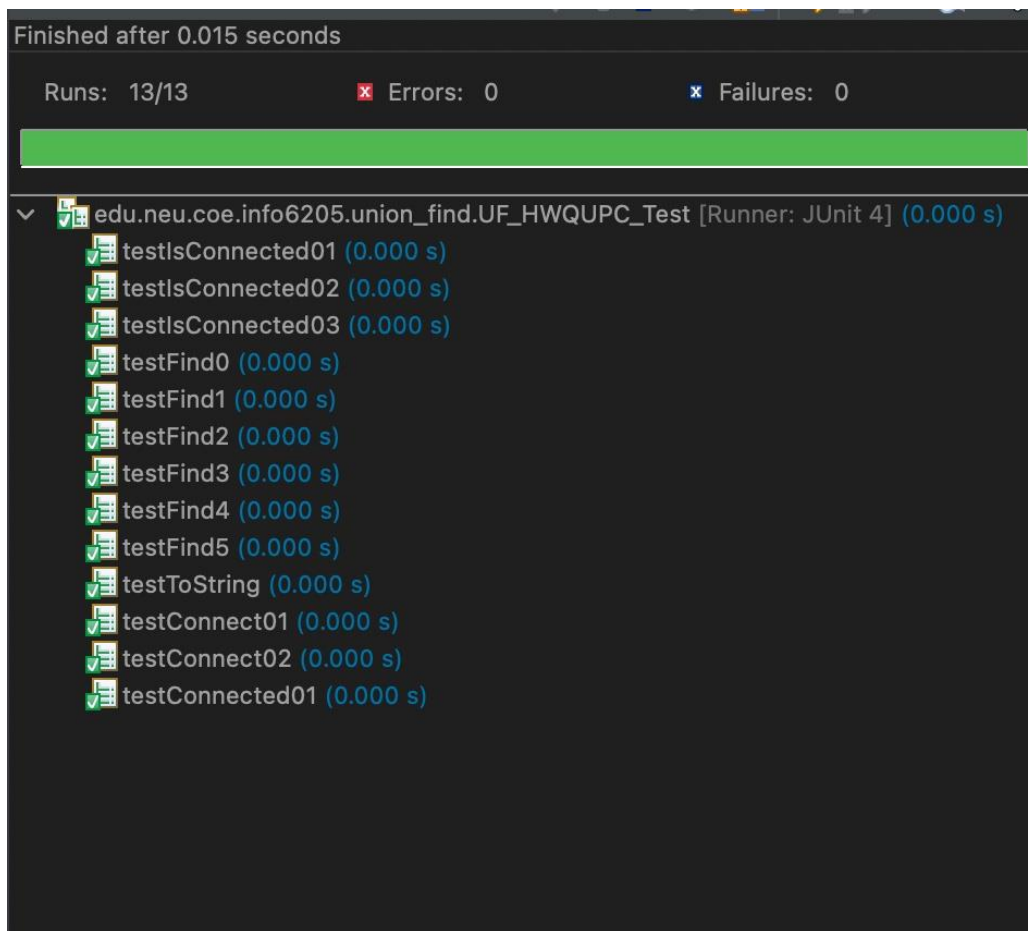
Finished after 0.013 seconds

Runs: 6/6 ✖ Errors: 0 ✖ Failures: 0

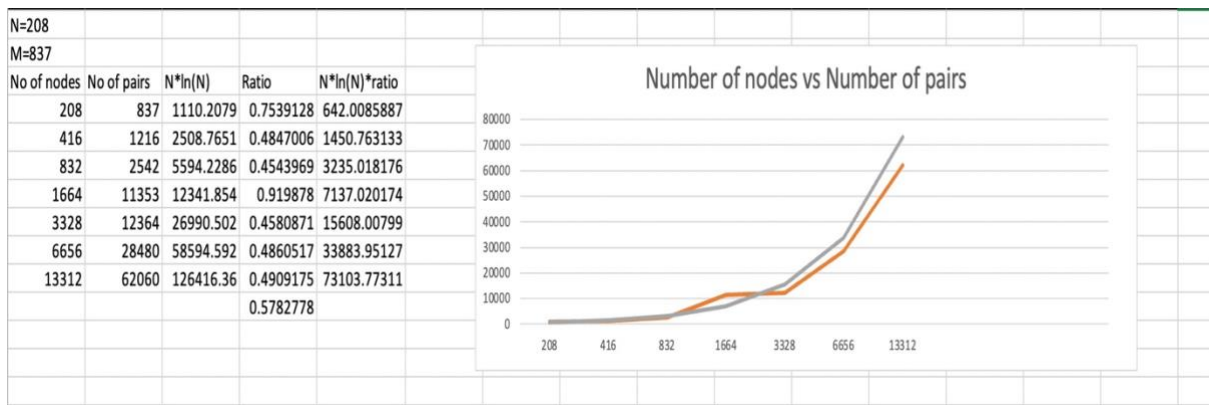


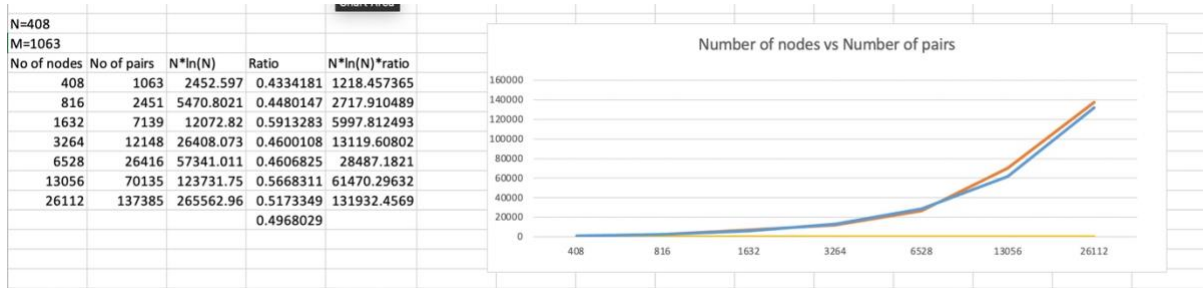
edu.neu.coe.info6205.union_find.WQUPCTest [Runner: JUnit 4] (0.000 s)

- testFind0 (0.000 s)
- testFind1 (0.000 s)
- testFind2 (0.000 s)
- testFind3 (0.000 s)
- testFind4 (0.000 s)
- testConnected01 (0.000 s)



Conclusion





The number of nodes gets multiplied by a factor of 2.
A graph for the number of nodes vs number of pairs is plotted.

Hence, we can see that as the graphs are colliding, hence the equation for the graph can be said as $m \cdot (N \log(N))$.

Hence, the above program uses union find structure with path compression, as the graph shows logarithmic relation between number of pairs and objects.

```

Number of nodes: 408
Number of connections: 407
Number of pairs: 1090
Number of nodes: 816
Number of connections: 815
Number of pairs: 3243
Number of nodes: 1632
Number of connections: 1631
Number of pairs: 5938
Number of nodes: 3264
Number of connections: 3263
Number of pairs: 15605
Number of nodes: 6528
Number of connections: 6527
Number of pairs: 30348
Number of nodes: 13056
Number of connections: 13055
Number of pairs: 76085
Number of nodes: 26112
Number of connections: 26111
Number of pairs: 119180

```