Manik Kumar (001063023) – Section 02

INFO 6205 - Fall 2020

Program Structures & Algorithms

Assignment 3

1. Task:

- Implement height-weighted quick union with path compression and run unit-tests.
- Create a UFClient class with a count() method which would create a height-weighed quick union with path compression problem with n elements. Then we'll generate m random pairs until the number of components in our quick find problem is 1.
- Determine the relationship between the number of objects (n) and the number of pairs(m) generated.

2. Output

To produce the desired output, I coded the quick union problem to be run for 50 times (i.e. ran my count() function), for the number of values ranging from 0 to 10,000 increasing by 200 elements each time. For each number of elements, the count() function was ran for 200 times and all the resulted values were averaged.

The following table represents the values generated by the program.

Number of Objects(n)	Average Number of Pairs generated(m)
1	1
1000	3722
2000	8529
3000	12582
4000	17715
5000	22785
6000	27652
7000	33659
8000	38603
9000	43372
10000	48352

Table 1: Number of Pairs (m) for values of (n)

Example terminal output:

```
For n = 1 average number of pairs generated over 200 runs were = 0.0
For n = 200 average number of pairs generated over 200 runs were = 5856.0
For n = 400 average number of pairs generated over 200 runs were = 13146.0
For n = 600 average number of pairs generated over 200 runs were = 20238.0
For n = 800 average number of pairs generated over 200 runs were = 29806.0
...
...
...
For n = 9200 average number of pairs generated over 200 runs were = 448538.0
For n = 9400 average number of pairs generated over 200 runs were = 465655.0
For n = 9600 average number of pairs generated over 200 runs were = 475065.0
For n = 9800 average number of pairs generated over 200 runs were = 476832.0
For n = 10000 average number of pairs generated over 200 runs were = 489690.0
```

3. Observation

From the outputs above, we could plot the following graph to depict the relationship between objects(n) and pairs(m).

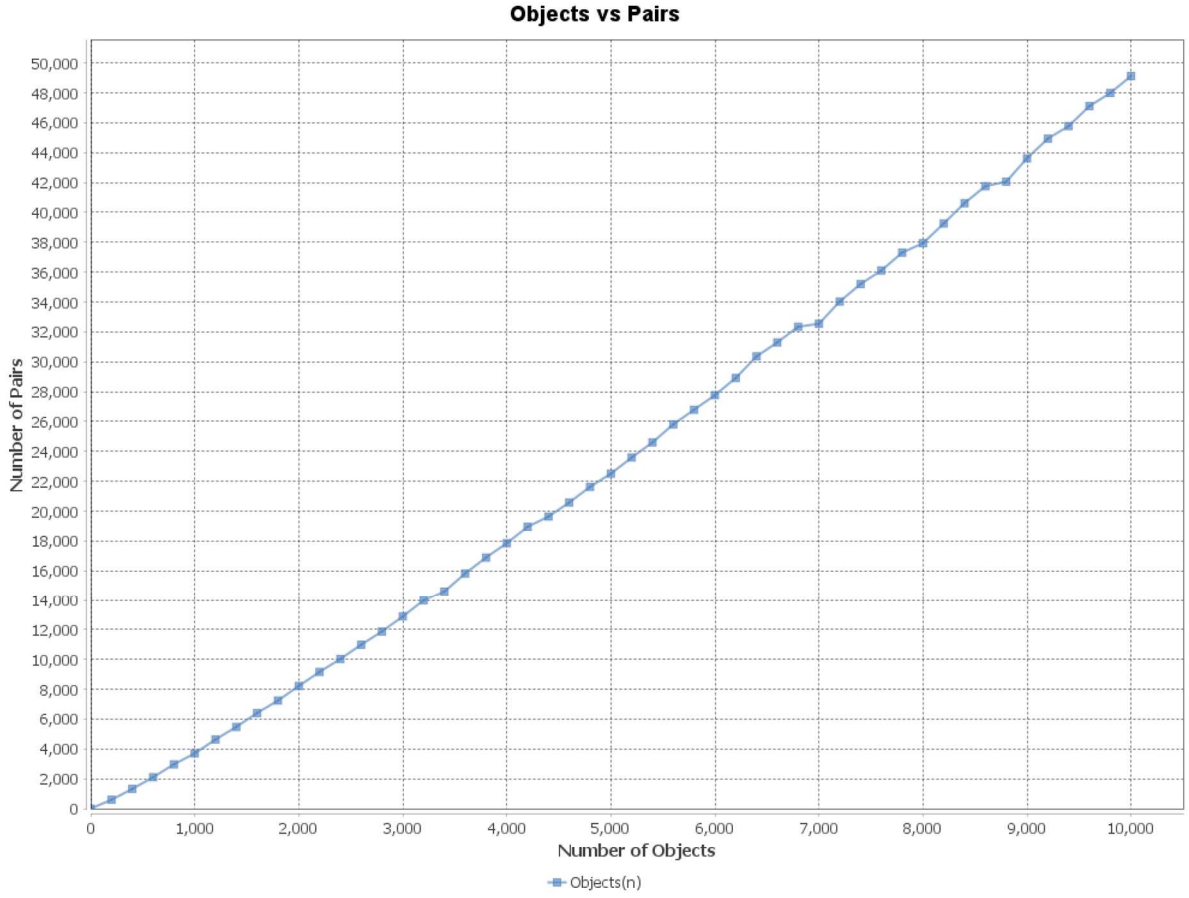

Figure 2: Number of Objects vs Pairs

From the graph, we can observe an almost linear relationship between m and n. Although that isn't true, and it can be proven by calculating slope from the starting point.

n	m	slope
1000	3722	3.722
3000	12582	4.194
6000	27652	4.594
9000	43372	4.819

Table 2: Slope for values of m and n

The table above depicts an almost linear relationship with the slope increasing as the size of n grows. In weighted quick-union, we keep track of the height of the tree and merge the smaller tree with the parent of the bigger tree. Since the height of a tree of 2^k items is k, the weighted quick union guarantees logarithmic performance, because the max height of the tree would be log(k).

Now in our case, assuming we would have to perform n unions, and the maximum height of our tree is log(n), m should be proportional to n.

$$m \propto n * \log(n)$$
$$m = k * n * \log(n)$$
$$\textit{where k is the coefficient.}$$

Now to prove this hypothesis, I calculated the coefficient k and plotted the graph of *k*n*log(n) vs n* and compared to the*m vs n* graph.

The coefficient was calculated by dividing m and n * log(n) for averaging for the values of n ranging from {1,10000}. The value of k was calculated to be approximately 0.53. Following is a table depicting some examples of the values,

n	m	m / n * log(n)
1000	3722	0.535
3000	12582	0.527
6000	27652	0.527
9000	43372	0.528

*Table 3: m / n * log(n) for values of m and n.*

The equation can be calculated to be,

$$m = 0.53 * n * \log(n).$$

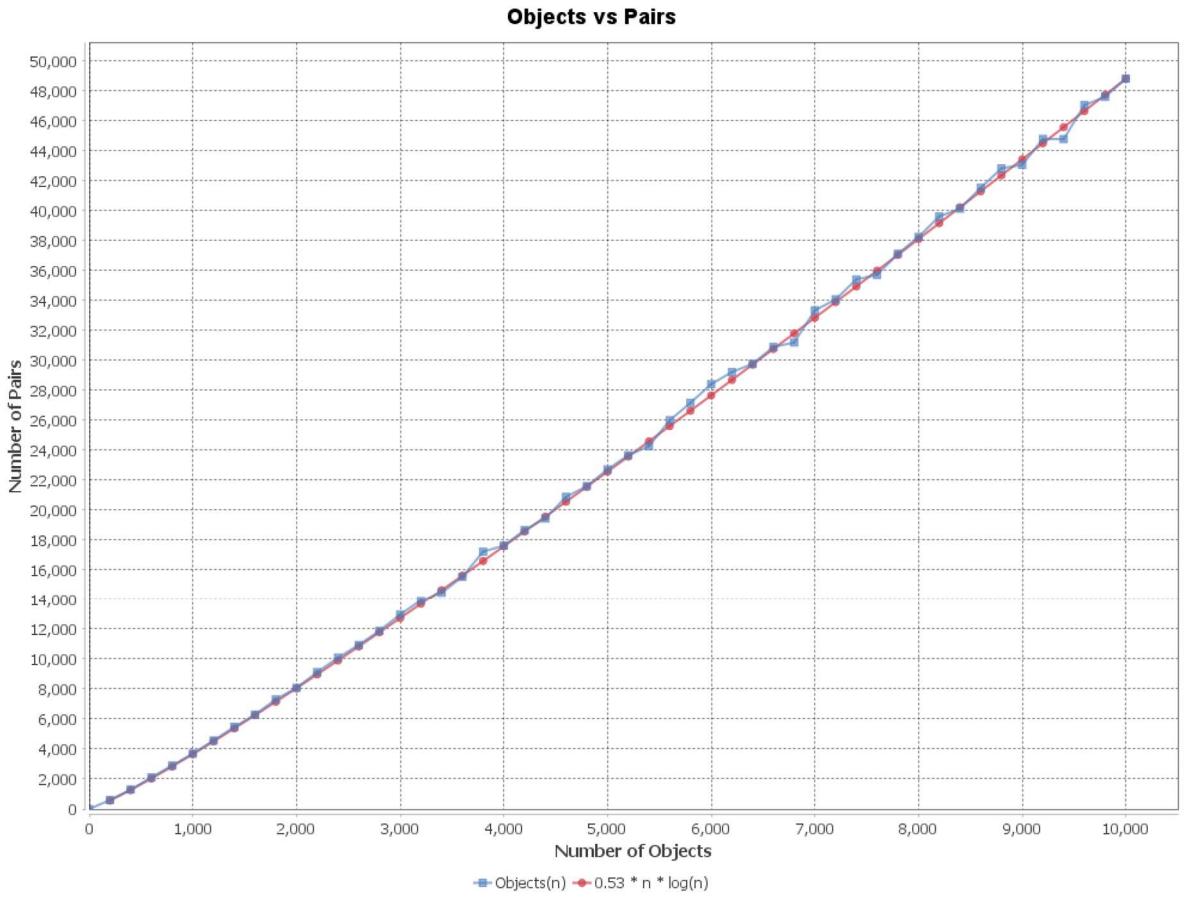

Figure 3: Graph with Objects(n) and our Hypothesis

From the above graph, we can see the two graphs overlapping almost perfectly. Hence proving the hypothesis that m is directly proportional to n * *log(n).*

4. Conclusion

Even though the relationship between m and n appears to be a linear relationship and is extremely similar to a straight line, it's rather,

$$m = k * n * \log(n)$$
where k can be calculated to be $0.53(approx.)$.

5. Screenshot of Unit-test passing

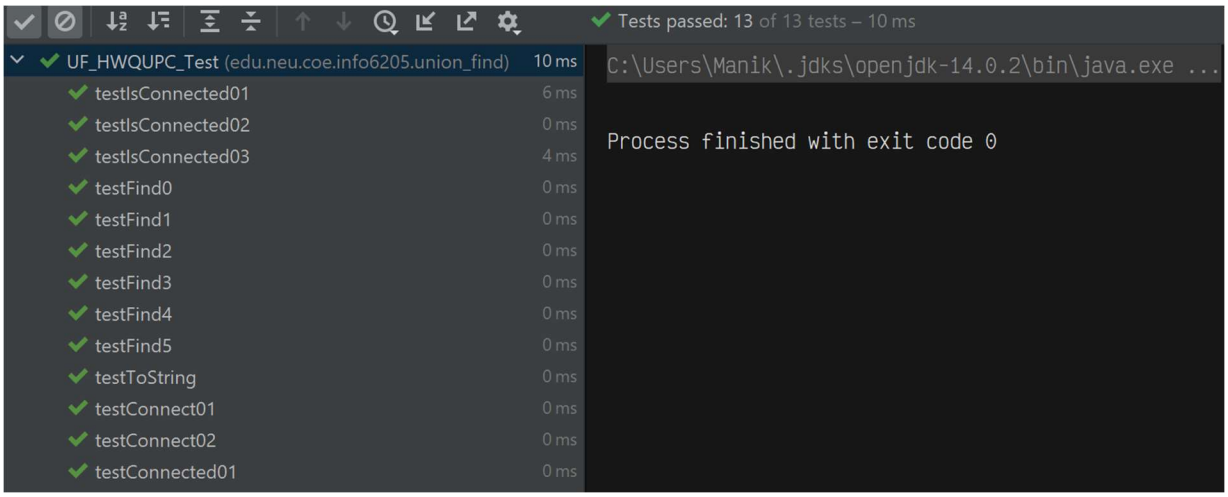

Figure 4: Unit-tests passing.