Narendra Raj R K (001553969)

Program Structures & Algorithms

Spring 21

Assignment No. 4

Tasks:

i) For weighted quick union, store the depth rather than the size.

⇨ Successfully added the logic to store the **depth** of the Quick Union find rather than storing the size of the tree in WQUPC_Alternate1.java.

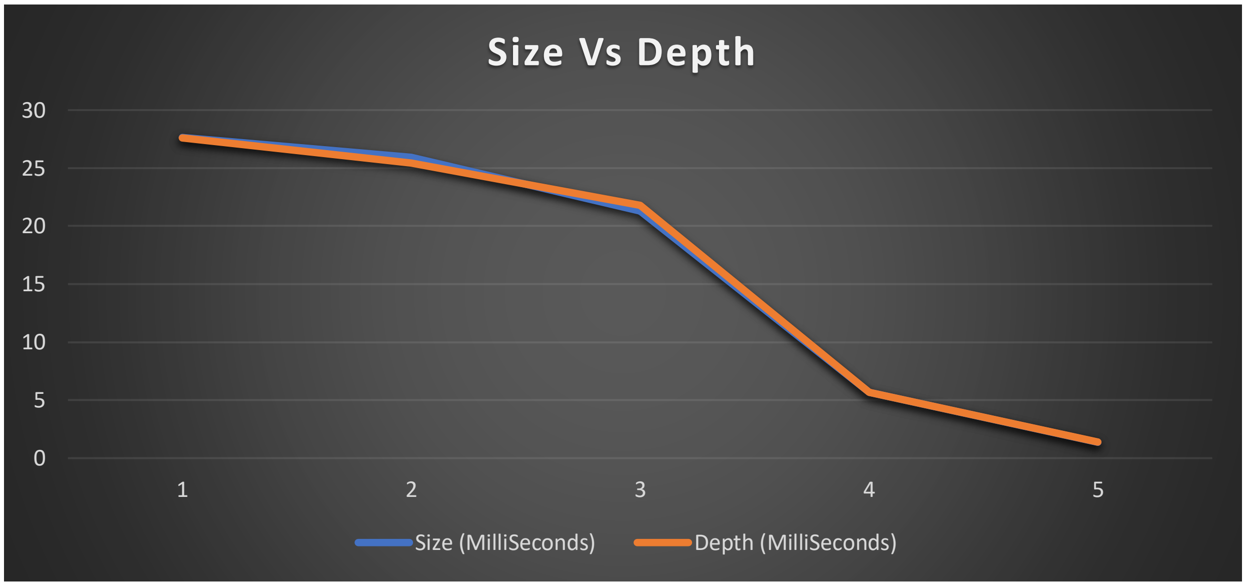
ii) For weighted quick union with path compression, do two loops, so that all intermediate nodes point to the root, not just the alternates.

⇨ Added the logic under **WQUPC.java** that makes the intermediate nodes point to the root rather than the alternates node(grandparent).

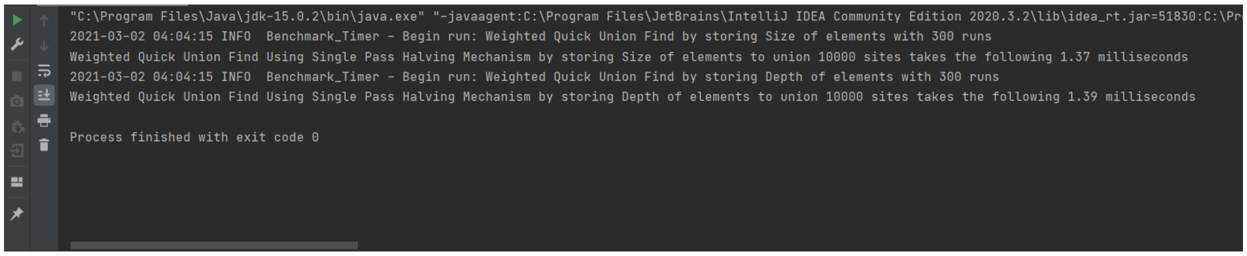
Evidence to Support Conclusion:

Task 1:

In the first half of this assignment I was instructed to implement the logic to store the Depth of the nodes and benchmark it against when storing the size of the nodes. Following is the graphical representation of running the benchmark on the above criteria.



Console Output:



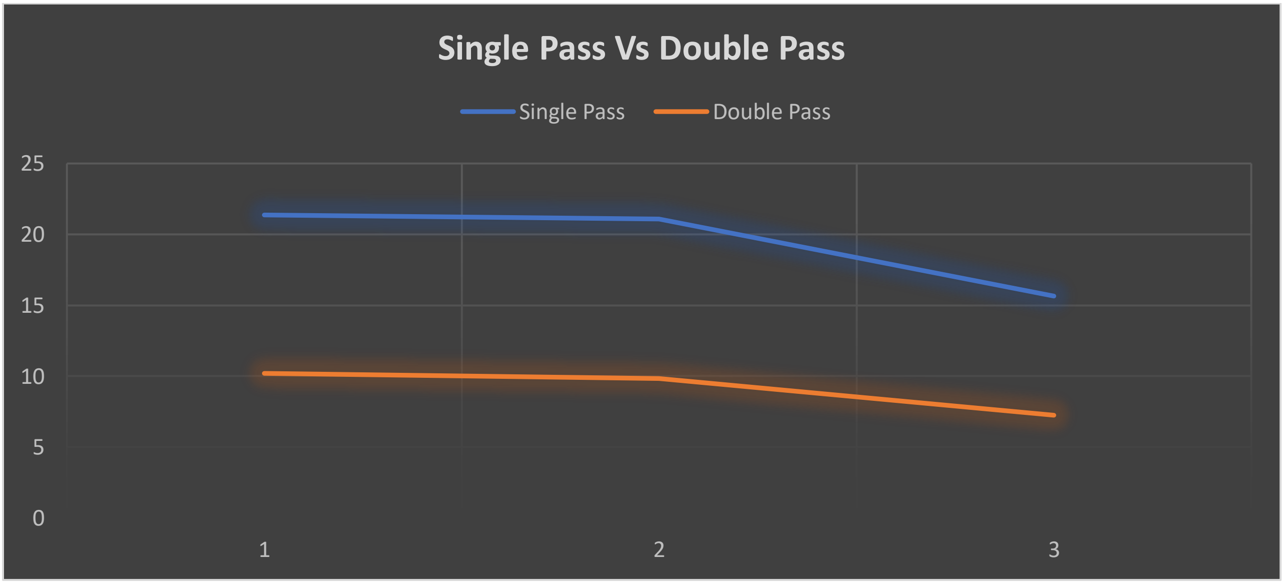
```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.2\lib\idea_rt.jar=51830:C:\Pr
2021-03-02 04:04:15 INFO  Benchmark_Timer - Begin run: Weighted Quick Union Find by storing Size of elements with 300 runs
Weighted Quick Union Find Using Single Pass Halving Mechanism by storing Size of elements to union 10000 sites takes the following 1.37 milliseconds
2021-03-02 04:04:15 INFO  Benchmark_Timer - Begin run: Weighted Quick Union Find by storing Depth of elements with 300 runs
Weighted Quick Union Find Using Single Pass Halving Mechanism by storing Depth of elements to union 10000 sites takes the following 1.39 milliseconds

Process finished with exit code 0
```

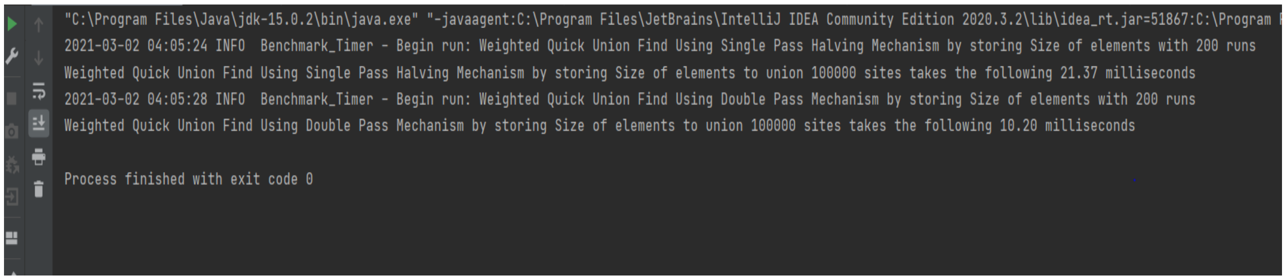
As the difference on benchmarking Weighted Quick Union using Size against Weighted Quick Union using Depth we can observe that the difference is **almost non-existent** and hence can considered to be **negligible**.

Task 2:

The next task was to implement Double Pass on the Weighted Quick Union with Path Compression (two loops) to make the intermediate nodes point to the root rather than the alternates. After implementing the aforementioned requirement, I benchmarked it against the class that has Single Pass halving mechanism. Following is the graphical representation of the benchmark results.



Console Output:



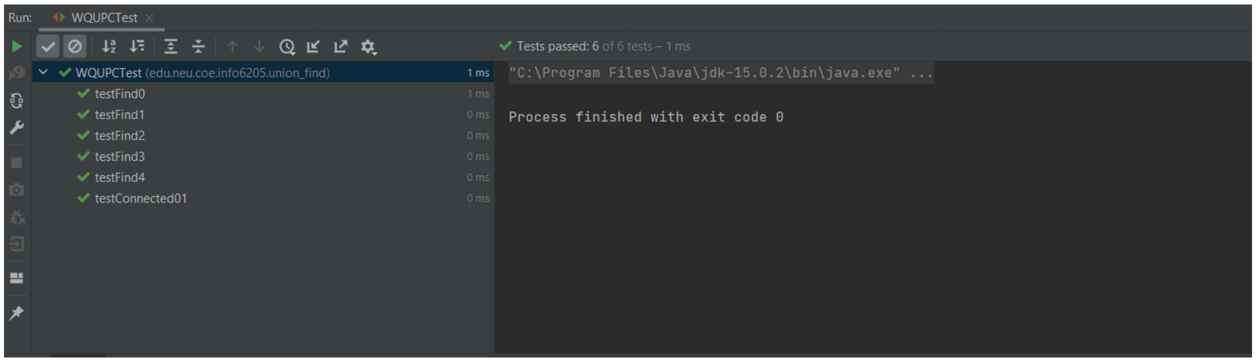
```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.2\lib\idea_rt.jar=51867:C:\Program
2021-03-02 04:05:24 INFO  Benchmark_Timer - Begin run: Weighted Quick Union Find Using Single Pass Halving Mechanism by storing Size of elements with 200 runs
Weighted Quick Union Find Using Single Pass Halving Mechanism by storing Size of elements to union 100000 sites takes the following 21.37 milliseconds
2021-03-02 04:05:28 INFO  Benchmark_Timer - Begin run: Weighted Quick Union Find Using Double Pass Mechanism by storing Size of elements with 200 runs
Weighted Quick Union Find Using Double Pass Mechanism by storing Size of elements to union 100000 sites takes the following 10.20 milliseconds

Process finished with exit code 0
```

As we can observe on implementing the double pass technique in the Weighted Quick Union Find we can evidently conclude that it is **significantly faster** (approx. twice faster) when compared to using Single pass Weighted Quick Union find.

Unit Test Results:

6 Test Cases under the WQUPCTest.java file passed successfully.



```
Run:  WQUPCTest
Tests passed: 6 of 6 tests – 1 ms
WQUPCTest (edu.neu.coe.info6205.union_find)   1 ms
    testFind0                                 1 ms
    testFind1                                 0 ms
    testFind2                                 0 ms
    testFind3                                 0 ms
    testFind4                                 0 ms
    testConnected01                           0 ms

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

Process finished with exit code 0
```