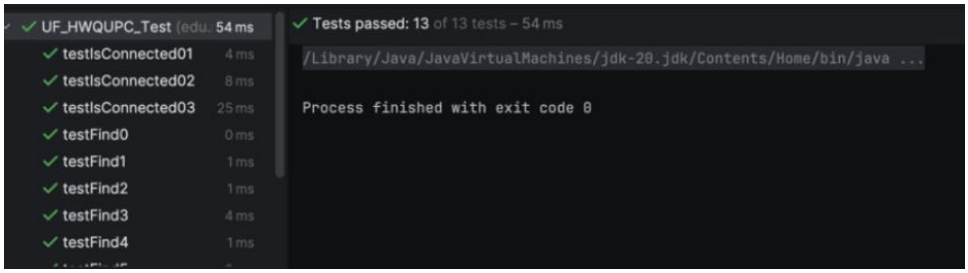
Program Structures and Algorithms
Spring 2024

NAME: Rohit Varma Mudundi
NUID: 002688431
GITHUB LINK: https://github.com/rohit26300/Rohit-Varma.git

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).

Part 1:

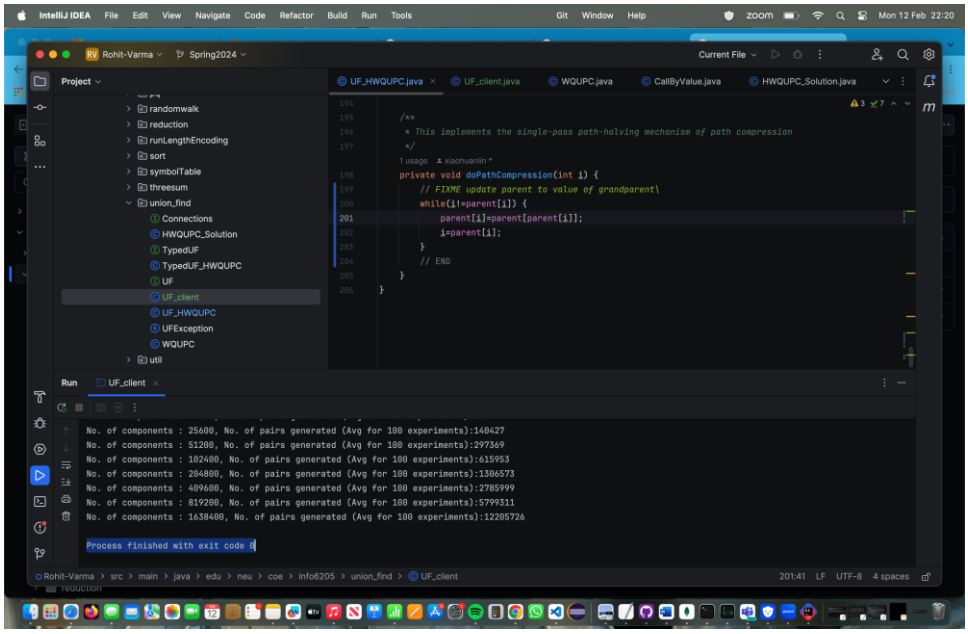


All test cases passed

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).

Part 2:

Created a new java class UF_client.java for step 2. Results obtained were:



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.

Relation between n and m:
No. of components(n) : 100, No. of pairs generated (m)(Avg for 100 experiments):270
No. of components(n) : 200, No. of pairs generated (m)(Avg for 100 experiments):602
No. of components(n) : 400, No. of pairs generated (m)(Avg for 100 experiments):1314
No. of components(n) : 800, No. of pairs generated (m)(Avg for 100 experiments):2953
No. of components (n) : 1600, No. of pairs generated (m)(Avg 100 experiments):6440

Conclusions based upon the Observations:
As the main method iterates through values of 'n' , increasing 'n' exponentially by a factor of 10 in each iteration, we observe a significant rise in the number of pairs generated('m'). As 'n' grows, the number of pairs generated increases rapidly due to the greater number of possible pairs of elements needing connection to form a single component. This indicates a non-linear time complexity for the algorithm, likely worsening as 'n' increases, with a particularly steep rise in pairs generated as 'n' grows larger.