INFO6205

PROGRAM STRUCTURES AND ALGORITHMS

UNION FIND

Problem Statement

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).

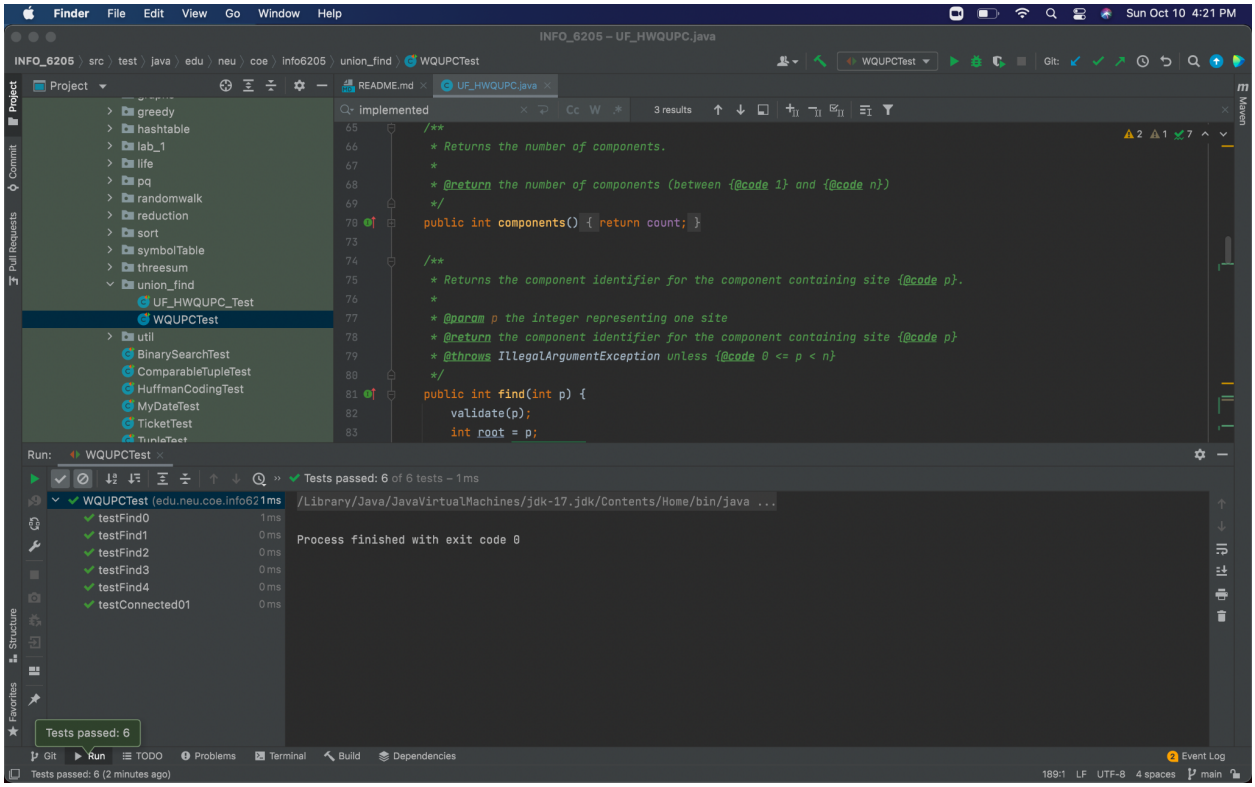
Step 3:

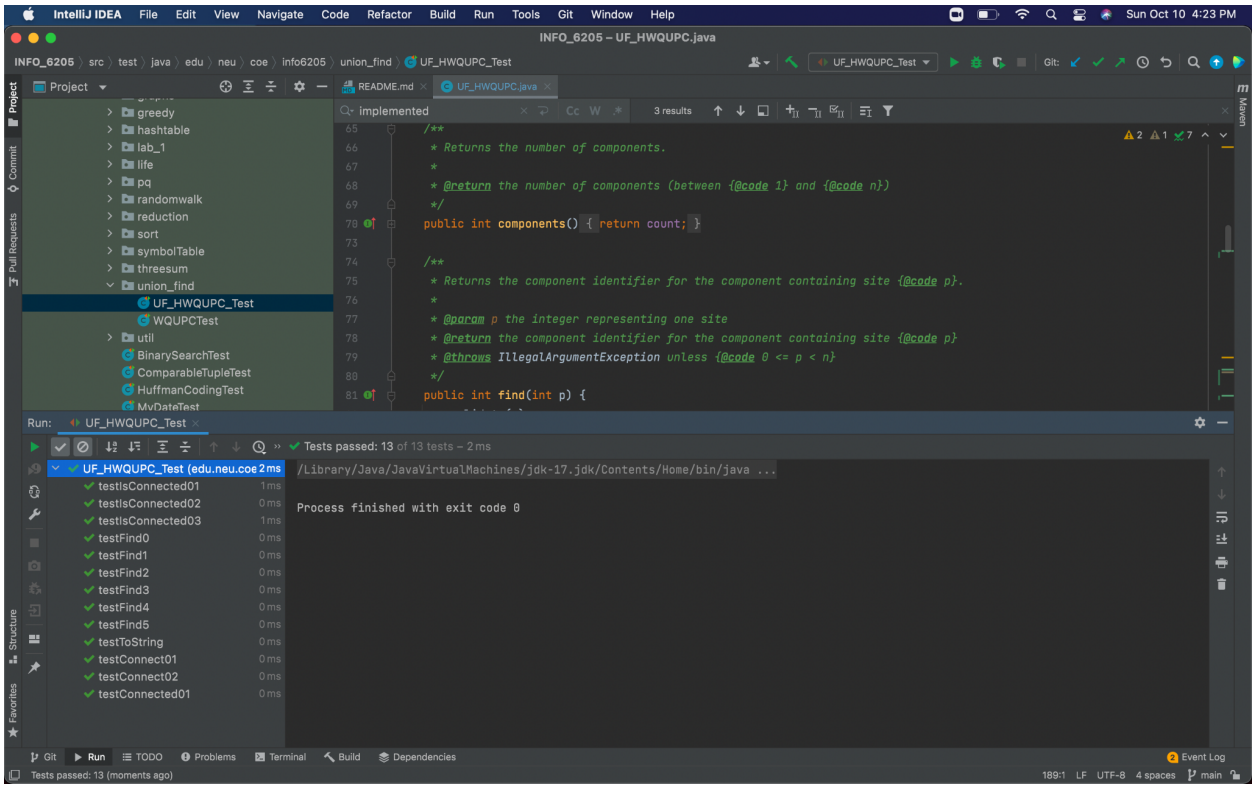
Determine the relationship between the number of objects (*n*) and the number of pairs (*m*) generated to accomplish this (i.e. to reduce the number of components from *n* to 1). Justify your conclusion in terms of your observations and what you think might be going on.

Solution

The methods mentioned in step 1 have been implemented and the test cases were run.

Screenshots can be seen below:





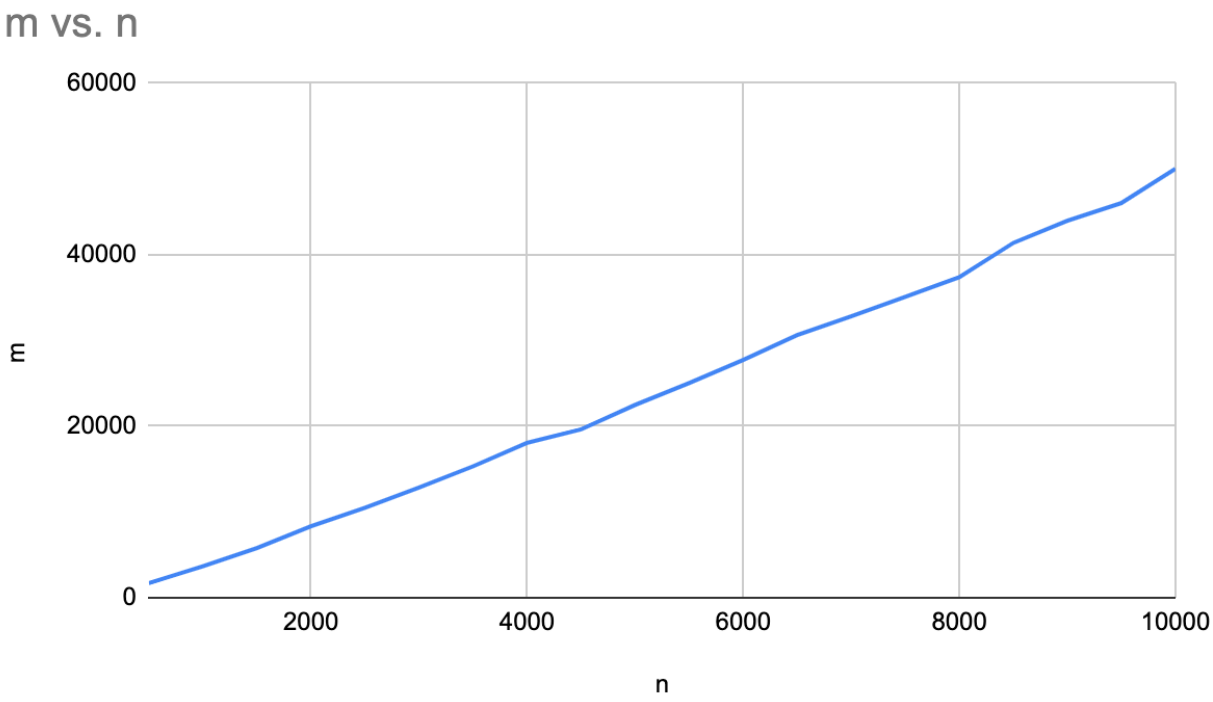
Main and Count methods:

The main method and the count method together try and generate the data that is necessary to infer a relationship between n and m.

For every value of n, random pairs are generated and checked for connection. If not connected they are sent to union(). The process is repeated until all are connected. The number of iterations it took to do this is called m.

The function is written to run a range between 500-10000 incrementing 100 each time for n and the corresponding m is calculated.

The graph between n and m is as follows:



We can see from the graph that the relationship is quite linear if not exactly linear. This variation can be accounted for by having a coefficient.

If the maximum height of the node is lof(n) and the number of unions performed is n, then we can say that m is equal to n*log(n).

The formula will then be:

M = Coefficient * n * log(n).

As seen from the data generated the coefficient is approximately: 0.5333959581719718

n	m
500	1701
1000	3660
1500	5797
2000	8331
2500	10475
3000	12837
3500	15317
4000	18060
4500	19641
5000	22490
5500	25034
6000	27716
6500	30624
7000	32810
7500	35087
8000	37385
8500	41387
9000	43959
9500	46035
10000	50023

Generated Data:

n: 500 m: 1701

Coefficient: 0.5474198768645965

n: 1000 m: 3660

Coefficient: 0.5298392679219672

n: 1500 m: 5797

Coefficient: 0.5284493645902958

n: 2000 m: 8331

Coefficient: 0.5480270299707216

n: 2500 m: 10475

Coefficient: 0.5355285480410193

n: 3000 m: 12837

Coefficient: 0.5344496069964753

n: 3500 m: 15317

Coefficient: 0.5362754645684993

n: 4000 m: 18060

Coefficient: 0.544366165614661

n: 4500 m: 19641

Coefficient: 0.5188722642147834

n: 5000 m: 22490

Coefficient: 0.5281082510295945

n: 5500 m: 25034

Coefficient: 0.5284916786036981

n: 6000 m: 27716

Coefficient: 0.5309874707992993

n: 6500 m: 30624

Coefficient: 0.5366312184935043

n: 7000 m: 32810

Coefficient: 0.5294013982389476

n: 7500 m: 35087

Coefficient: 0.5243131037435368

n: 8000 m: 37385

Coefficient: 0.5199758159581277

n: 8500 m: 41387

Coefficient: 0.5381470947989513

n: 9000 m: 43959

Coefficient: 0.5364463634638219

n: 9500 m: 46035

Coefficient: 0.5290713578207742

n: 10000 m: 50023

Coefficient: 0.5431178217061592

Average coefficient:0.5333959581719718

Process finished with exit code 0