Isaac Gutt

Intro to Algorithms

October 28, 2021

AnthroChassidus.java

Anthro Chassidus's methods fit perfectly with the union find algorithms. Union find algorithms are built to do two things, connect integers to each other, and find which integers are connected to one another. There is an ID array that starts with 0 to n-1, and every time union is called, one of the integers becomes the root of the other, with the root taking the place of its descendant in the array. So when Reuven says he's in the same group as Shimon, Shimon's spot in the array says Reuven, so now they are connected. In addition, we can add another array to the mix that instead of having the root of the integer at its index, holds the number of integers descending from the root at each root's index. In this way, the user can call find on any integer, finds its root, and then just return what's in the size array at the root's index. When the user calls union(root, new integer) the size array increments the root's index in the size array by one, and in a case when union(root, other root) is called, one root's size is added to the other. So now if Reuven is connected to Levi, Reuven will be at Levi's slot in the ID array, and in the size array Reuven's slot will have 3 (Shimon has been connected already).

Again, this is exactly what we need for AnthroChassidus. To begin, I have an integer named chassidus that's set equal to population. Each time two new integers are connected, the number of possible groups of chassidus goes down by one. In code, when connected() returns false, the numbers are connected with union and chassidus is decremented by one. Next, I decided to use WeightedQuickUnion, as it was the fastest of the union find algorithms based on the testing I did. I have a loop going through the two arrays, connecting the two integers whenever necessary, and this automatically sets the size array and the ID array. When that's done, theGetLowerBoundOnChassidusTypes integer is set, so I just need to finish the constructor so nShareSameChassidus is O(1) time too. For that, I just loop through the union array one more time, adding each integer and its size (as explained above) to a hashmap. Now, the constructor is done and all subsequent calls will be constant time.

To be sure that this was a linear solution, I used the strategy I used for EstimateSecretAlgorithms, a stopwatch class. I have a loop that doubles the size of an integer i (for population) each time, and each time two population sized arrays are filled with random numbers from 0 to population -1. The stopwatch timer is then initiated, AnthroChassidus is called with the two new arrays and population, and when the constructor is done the timer stops, and the program prints the population size, the time the constructor took, and the ratio comparing how the

ratio that it grew from previous time. The results were quite good, the trend line was increasing at a rate of close to 1.8 and very close to 2 the last few times, proving that it is a linear solution.

n	log n	f(n)	log(f(n))	ratio
8192	3.9	5	0.6	
16384	4.2	5	0.6	1
32768	4.5	8	0.9	1.6
65536	4.8	13	1.1	1.6
131072	5.1	39	1.5	3
262144	5.4	48	1.6	1.2
524288	5.7	115	2	2.3
1048576	6	340	2.5	2.9
2097152	6.3	619	2.7	1.8
4194304	6.6	1285	3.1	2
8388608	6.9	2821	3.4	2.1