AnthroChassidus

Max Friedman

In no more than three paragraphs, explain your solution design: You may "explain by reference" to material covered in lecture or textbook. If you choose to use a "reference to covered material", you must offer a convincing explanation that this material is relevant to solving our problem.

To begin, to find a way to connect the pair of arrays of Chassidim to each other in an efficient way, I implemented the following data structure which gives us an ability to connect mutual elements into a network: **Weighted-Quick-Union with Path Compression**, a data structure that runs at about O(n) time. The Weighted-Quick-Union holds information in two arrays: a main array (ID[])which shows who connects to who, and a size array (size[]) that shows how many connections each Chassid has.

The Weighted-Quick-Union takes about O(n) runtime (n being the number of Chassidim interviewed which is also the length of the two helper arrays) to connect all inputted data, because it needs to create the two helper arrays (ID[] and size[]) and also union each index to each other. Obviously creating the helper arrays is O(n) because it must iterate over its actual length. Union is around O(n) time because, first of all, the a and b arrays holding the connections will range somewhere between 1 to n in length. Second of all, each Union calls Find(p) which is almost always basically constant, because Find(p) simply does a quick check where p is connected to in the ID array. As we add more elements however, instead of p simply being connected to one element, p can become connected to an element that is connected to another element (and so on), causing multiple checks. To solve this issue, I implemented Path Compression, which simply unions p to its grandparent instead of its parent. That halves the overall runtime it would have been, causing as few checks as possible.

The methods "getLowerBoundOnChassidusTypes()" and "nShareSameChassidus(id)" are O(1) because they are each simply returning an instance variable.

1. The method getLowerBoundOnChassidusTypes() returns an instance variable "count". "Count", originally equal to n, gets subtracted by 1 when a union occurs between two Chassidim. This is because when a pair occurs, there is one less possibility of types of Chassidim in the population.
2. The method nShareSameChassidus(id) finds where id occurs in the ID array (this will be the element id is connected to), and checks that value in the size array. Since the size array holds the amount of connections each element has, it will successfully return the number of people who follow the same Chassidus.

Example:

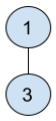
n = 7

a[] = [1,4,3,4]

b[] = [3,5,6,1]

This means that we have four unions, let's walk through each one.

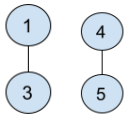
Union(1,3):



index	0	1	2	3	4	5	6
ID[]	0	1	2	1	4	5	6

index	0	1	2	3	4	5	6
SIZE[]	1	2	1	1	1	1	1

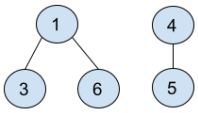
Union(4,5)



index	0	1	2	3	4	5	6
ID[]	0	1	2	1	4	4	6

index	0	1	2	3	4	5	6
SIZE[]	1	2	1	1	2	1	1

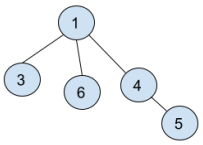
Union(3,6)



index	0	1	2	3	4	5	6
ID[]	0	1	2	1	4	4	1

index	0	1	2	3	4	5	6
SIZE[]	1	3	1	1	2	1	1

Union(4,1)



index	0	1	2	3	4	5	6
ID[]	0	1	2	1	1	1	1

index	0	1	2	3	4	5	6
SIZE[]	1	5	1	1	2	1	1

Because of Path Compression, instead of 5 being connected to 4, 5 is connected to 1 in the ID array (since it is its grandparent.)