AnthroChassidus

Jonathan Wenger

Solution Design

Immediately when I saw this problem, I realized that the way to solve this would be using the Union-Find data structure. This structure allows us to group two elements that we know are "similar" (i.e. two people who we know share the same Chassidus). Realizing we learnt and saw the code already, the most efficient implementation of the Union-Find data structure is the weighted quick union along with path compression, which allows for an (extremely close to, but not exactly) O(n) initialization, an O(1) union call, and an O(1) find call. My code was basically copied from the textbook (page 228), with some minor changes (including path compression, which changed the speed of union and find from O(log n) to extremely close to O(1).

My constructor (which took nearly O(n) time) first initialized an underlying array (called underlyingArray, equivalent to array id[] in Sedgewick's code) which was the size of the number of people in the group, and set each index in the array to that index (i.e. index 0=0, index 1=1, etc). Another array, called sizeArray (equivalent to array sz[] in Sedgewick's code), was also the size of the number of people in the group, and initialized each index to 1 (as that is the size of the component for the roots). I also initiated an int called numComponents (equivalent to the int count in Sedgewick's

code), which starts off as the number of people in the group, but can diminish when union is called if two trees merge into one larger tree. I then iterated through the Arrays A and B, calling union(a[i], b[i]) on each index, which was a nearly O(1) operation on each call (as path compression allows for). Therefore, because all my constructor does is iterate through arrays a constant number of times (through underlyingArray, sizeArray, and arrays A and B), that is O(N).

getLowerBoundOnChassidusTypes just returned numComponents (as the union method dealt with the number of distinct groups (i.e. types of Chassidus)), which is an O(1) operation. nShareSameChassidus returned the find(n'th) index of sizeArray, which is the size of the root of that given group of Chassidus (the union method dealt with updating the sizeArray). Finally, the only change I made from the find() method from Sedgewick p. 228 was I added the line underlyingArray[p]=underlyingArray[underlyingArray[p]];, which allowed for path compression, changing the runtime of find() from O(log n) to extremely close to O(1), as both Sedgewick and the slides we learnt in class say.