How Many Wedding Tables? Union-Find

Chandler Klein

03/12/2020

Report

The included Java program utilizes a disjoint-set (union-find) data structure to determine how many tables are necessary for a certain number of friends who may or may not know each other at a wedding. A disjoint-set data structure ensures that no item, or friend, can be in more than one set, or table, and allows us to perform union and find operations to combine groups of friends into tables and find the "parent" friend of the set, or table. Initially, the make set function creates a set for all friends such that the sets contain only one friend. This method has a worst-case complexity of $\mathcal{O}(n^2)$ because it inserts values into a Hash Map using the put method – which has a worst-case complexity of $\mathcal{O}(n)$ – for all n input values. This is unlikely unless very large values of n are being used. This method should almost always have a time complexity of $\mathcal{O}(n)$ – n values are inserted into the Hash Map with a constant $\mathcal{O}(1)$ complexity.

If the root of a friend is not the same as the friend itself, the find method recursively calls itself to find out what the root of the friend is and uses the put Hash Map method to update the root of the original friend. The method returns the root of the friend. This method has a worst-case complexity of $\mathcal{O}(n)$, because it uses Hash Map methods get and put in addition to a potential recursive call. In the best case, this method has a complexity of $\mathcal{O}(n)$ when getting a value from the root Hash Map without recursively calling itself, which is proportional to the size of the tree.

Two sets, or tables, can be combined when two friends know each other. When this happens, the union method is called on them. This procedure begins with calling the find method to figure out the root friend that each friend is pointing to. If they are the same, nothing is done, since the two friends are already in the same set, or sitting at the same table. If the root friend of the two friends is different, then the result will be a combination of the two disjoint sets, or tables. The rank, utilized as a Hash Map, is taken into consideration when the sets are going to be combined. In order to improve the efficiency and speed of the algorithm, if the rank of one root is greater than the other, the smaller tree will be placed under the larger one. This is done to prevent issues with unbalanced structures. The parent Hash Map gets updated with the parent that the other is now pointing to. If this union of 1 sets is successful, the union method will return true. When the union method returns true, a counter is incremented so that the final number of sets, or tables, can be calculated. The

final number of sets will be the number of friends at the wedding minus the number of unions successfully completed. This is because we start with the same number of tables as friends – each time two friends are joined, the table occupied by the single friend becomes empty and unnecessary because they joined the other table. The complexity of the union operation is proportional to the complexity of the find operation. This is because the union method uses the find method to figure out what the root of each friend is. Not considering the use of the find method, the union method has a worst-case time complexity of $\mathcal{O}(n)$, because it uses a Hash Map to access and insert values. This worst-case complexity can occur in cases where the Map is large and many collisions occur. The best-case complexity for access and insertion of Hash Map values is $\mathcal{O}(1)$.

In terms of space complexity of the program, since a Hash Map is used for both the rank and the roots of the sets, containing all values of the input value, the space complexity is $\mathcal{O}(n)$, where n is the number of friends specified in the input file. In the worst-case, this program could theoretically have a time complexity of $\mathcal{O}(n^2)$ when the union and/or find operations have a worst-case complexity, resulting from poor Hash Map performance being using on a very large input size n. The best-case and likely average-case time complexity of this algorithm is approximately $\mathcal{O}(n)$ almost every time.

References

https://www.techiedelight.com/disjoint-set-data-structure-union-find-algorithm/
https://en.wikipedia.org/wiki/Disjoint-set data structure