Ashwin Bhanderi 44164971

Algorithm Report

Part 1(Reading the input)

At the starting of designing my algorithm, I was reading each group and storing all the values in a HashMap as key being the group number and element being Arraylist of group members. While reading the input via the Scanner in, a while loop was used to ensure we received all the groups. Hence the exit condition of the while loop was to ensure group size was zero, meaning that all groups have been read. Inside this while loop, the input keys was been checked if it was a “0”(end of group) then reduce the group size by 1. Also, another HashMap was used to store unique members as the key and Arraylist to store the occurrence in which group it was found. Later I realized that the first HashMap was not needed hence I removed it.

The complexity of reading in groups members by scanner in was, O(n) where n was the numbers of members in all groups combined including 0s.

Part 2(applying the algorithm)

Once the HashMap containing unique occurrence of members in what groups was filled, another while loop was used with exit condition being the size of HashMap == 0.

Inside the while loop:

- A for loop was used to find out the largest occurrence of unique numbers in all groups and each group size was checked to ensure the size was not 0(meaning the occurrence was 0, more on that later) and adding that group to array list to be removed. The complexity of this for loop was 0(n) where n was the size of unique members of the collection.
- Then another for loop was used to remove empty groups from the collection. The complexity of this was dependent if there were empty groups found earlier. The worst case for this will be removing all the groups O(n), where n is the number of unique members. This for loop is likely to be empty then running got the first time, but in later runs it will find empty groups.
- After that the largest unique member selected earlier was added to the final list, and its group deleted, however the members were stored in temporary array. The operation was O (1).
- Another for loop was used to iterate through all the groups in the unique collections and remove the elements that were same in the largest group selected earlier. The operation for this was O (n log n) because initially it has to go through all the members and later the members can be deleted and size becomes smaller. The worst case for this will be close to O(n^2) but every iteration of the while loop we reduce by at least one group (largest selected).
- Finally, when the frequency size == 0, we can print the finalist array size and its content.

Why this algorithm provides the best solution?

This algorithm works because we select the highest occurrence of unique member, because it's more likely that it will be covered in many different groups. Then we add that number in our final list and delete that group, but temporary store the members to be removed from all the groups that contains them. This reduces the size of the problem every iteration. However, the performance is very dependent on the input groups. This algorithm was inspired from set cover problem. I tested the algorithm with large numbers and the result was decent. In future I would like to randomly select from the list of largest groups and randomly pick one unique group, but this can make the output different every time. Hence, I selected the last largest in this algorithm.

Reference List

Java Program to Solve Set Cover Problem - GeeksforGeeks. (2021). Retrieved 5 September 2021, from https://www.geeksforgeeks.org/java-program-to-solve-set-cover-problem/

How to iterate any Map in Java - GeeksforGeeks. (2021). Retrieved 5 September 2021, from https://www.geeksforgeeks.org/iterate-map-java/

HashMap (Java Platform SE 8). (2021). Retrieved 5 September 2021, from https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html