Union and Intersection - Explanation

Union

This function uses a dictionary, which is an implementation of a hashtable. It iterates through both lists and add the value of a node as a key and a count as value. If a key already exists, we increase the count. This is helpful to keep track of distinct elements in the result. In the end, the functions iterates through all keys in the dict and appends them one by one to a Linked Lists.

Intersection

First, it determines the size to find out, which list is larger. Then it iterates through the longer lists and adds them the same way like in `union` to a hashtable. In the next part, it iterates through the second list and compares them one by one with the hash table if such key exists or not. If this key already exists, it adds them to the result linked list and changes its value to -1. By marking them with -1, it will indicate that this value was already appended to the result. The goal is to avoid having the distinct elements in the resulting linked list.

Time and Space Complexity

The time complexity both for `union()` and `intersection` is $O(M + N)$, where M and N are the lengths of each lists. The space complexity for `union` is $O(M + N)$. The space complexity for `intersection` is most likely $O(M)$, where M and N are the total length of each lists and `m < n`