Striver’s SDE Sheet

Day 7: Linked List and Arrays

Rotate List

Given the head of a linked list, rotate the list to the right by k places.

Ans:

Observation and approach :

Just do the thing that it stay

Check if the head of the list is null. If it is, return null since there is no list to rotate.

Initialize variables: newHead and tail as pointers to the head node, and size as 1 since the list has at least one node.

Iterate through the list to find the tail node and calculate the size of the list.

After finding the tail, connect it to the head, forming a circular list.

Adjust the value of k if it is larger than the size of the list by taking the modulus of k and the size.

Traverse the list to the node where the rotation should start, which is size - k nodes away from the tail.

Update newHead to the node next to the tail, as it becomes the new head of the rotated list.

Break the circular list by setting the tail's next pointer to null.

Return the new head of the rotated list as the result.

Code:

```
class Solution {

public:

ListNode* rotateRight(ListNode* head, int k) {

    if (head == nullptr) {

        return nullptr;

    }


    ListNode* newHead;

    ListNode* tail;
    newHead = tail = head;
    int size = 1;

    // Find the tail of the list and calculate its size
    while (tail->next) {
        tail = tail->next;
        size++;
    }

    // Connect the tail to the head to form a circular list
    tail->next = head;

    // Adjust the value of k if it is larger than the size of the list
    k = k % size;

    // Traverse to the node where the rotation should start
    for (int i = 0; i < size - k; i++) {
        tail = tail->next;
    }

    // The node next to the tail becomes the new head of the rotated list
    newHead = tail->next;

    // Break the circular list by setting the tail's next pointer to null
    tail->next = nullptr;

    return newHead;
    }
};
```

3Sum

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

3 <= nums.length <= 3000

Ans:

Observation and approach :

Remove Duplicates from Sorted Array

Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return *the number of unique elements in* nums.

Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

- Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.
- Return k

Ans:

Straight forward question

Code:

```
int removeDuplicates(vector<int>& nums) {
        int i = 0;
        for (int j = 1; j < nums.size(); j++) {
            if (nums[i] != nums[j]) {
                i++;
                nums[i] = nums[j];
            }
        }
        return i + 1;
    }
```

Max Consecutive Ones

Given a binary array nums, return the maximum number of consecutive 1's in the array.

Ans:

We can use kadane's approach to solve this problem

Code:

```
int i=0, j=0, maxcount=0;
    int n = nums.size();
    while(j<n){


        if(nums[j] == 0){
            i = j+1;
        }


        maxcount = max(maxcount, j-i+1);
        j++;
    }
   return maxcount;
```