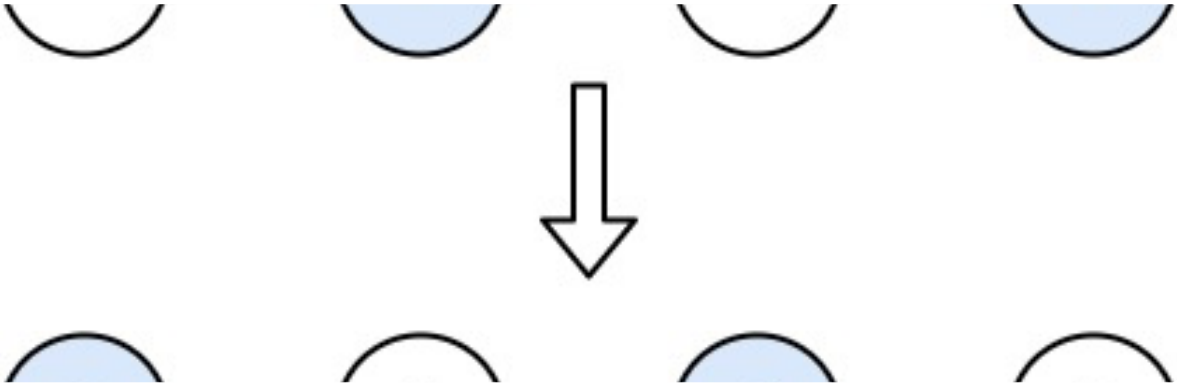
Assignment Solutions | Linkedlist - 4 | Week 15

1. Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed.)

   Example 1:



   Input: head = [1,2,3,4]

   Output: [2,1,4,3]

   Example 2:

   Input: head = []

   Output: []

   Example 3:

   Input: head = [1]

   Output: [1]

   Constraints:

   - The number of nodes in the list is in the range `[0, 100]`.
   - `0 <= Node.val <= 100`

Solution:

```
class Solution {
public:
```

```
int length(ListNode *head){
    int cnt = 0;
    ListNode *temp = head;
    while(temp){
        cnt++;
        temp=temp->next;
    }
    return cnt;
}
ListNode* reverseKGroup(ListNode* head, int k) {
    int len = length(head);
    cout<<len<<endl;
    if(len < k or !head)return head;
    ListNode *dummy = new ListNode(0);
    dummy->next = head;
    ListNode *curr = dummy;
    ListNode *prev = dummy;
    ListNode *nex = dummy;
    while(len >= k){
        curr = prev->next;
        nex = curr->next;
        for(int i=1;i<k;i++){
            curr->next = nex->next;
            nex->next = prev->next;
            prev->next = nex;
            nex = curr->next;
        }
        prev = curr;
        len -= k;
    }
    return dummy->next;
}
ListNode* swapPairs(ListNode* head) {
    return reverseKGroup(head , 2);
}
};
```

2. You are given the `head` of a linked list, which contains a series of integers **separated** by `0`'s. The **beginning** and **end** of the linked list will have `Node.val == 0`.

For **every** two consecutive `0`'s, **merge** all the nodes lying in between them into a single node whose value is the **sum** of all the merged nodes. The modified list should not contain any `0`'s.

Return *the* `head` *of the modified linked list*.

Example 1:

Input: head = [0,3,1,0,4,5,2,0]

Output: [4,11]

Explanation:

The above figure represents the given linked list. The modified list contains

- The sum of the nodes marked in green: 3 + 1 = 4.
- The sum of the nodes marked in red: 4 + 5 + 2 = 11.

Example 2:

Input: head = [0,1,0,3,0,2,2,0]

Output: [1,3,4]

Explanation:

The above figure represents the given linked list. The modified list contains

- The sum of the nodes marked in green: 1 = 1.
- The sum of the nodes marked in red: 3 = 3.
- The sum of the nodes marked in yellow: 2 + 2 = 4.

Constraints:

- The number of nodes in the list is in the range `[3, 2 * 105]`.
- `0 <= Node.val <= 1000`
- There are **no** two consecutive nodes with `Node.val == 0`.
- The **beginning** and **end** of the linked list have `Node.val == 0`.

Solution:

```
class Solution {
public:
    ListNode* mergeNodes(ListNode* head) {
        ListNode *dummy = new ListNode(0);
        dummy->next = head;
        ListNode *temp = dummy;
        int sum = 0;
        while(head){
            if(head->val == 0){
                temp->next = new ListNode(sum);
                temp = temp->next;
                ;
                sum = 0;
            }
            else{
                sum += head->val;
            }
            head = head->next;
        }
        return dummy->next->next;
    }
};
```
