725. Split Linked List in Parts

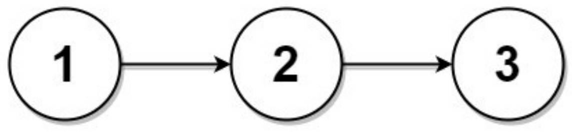
Given the `head` of a singly linked list and an integer `k`, split the linked list into `k` consecutive linked list parts.

The length of each part should be as equal as possible: no two parts should have a size differing by more than one. This may lead to some parts being null.

The parts should be in the order of occurrence in the input list, and parts occurring earlier should always have a size greater than or equal to parts occurring later.

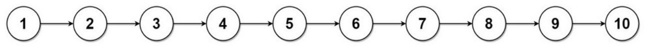
Return *an array of the* `k` *parts*.

Example 1:



```
Input: head = [1,2,3], k = 5
Output: [[1],[2],[3],[],[]]
Explanation:
The first element output[0] has
output[0].val = 1, output[0].next = null.
The last element output[4] is null, but
its string representation as a ListNode
is [].
```

Example 2:



```
Input: head = [1,2,3,4,5,6,7,8,9,10], k =
3
Output: [[1,2,3,4],[5,6,7],[8,9,10]]
Explanation:
The input has been split into consecutive
parts with size difference at most 1, and
earlier parts are a larger size than the
later parts.
```

Constraints:

- The number of nodes in the list is in the range `[0, 1000]`.
- `0 <= Node.val <= 1000`
- `1 <= k <= 50`

725. Split Linked List in Parts

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */

/*
    Pointers
    Time complexity: O(n+k)
    Space complexity: O(k)
*/
class Solution {
    public:
        int count_nodes(ListNode* head){
            ListNode* trav=head;
            int cnt=0;
            while(trav){
                cnt++;
                trav=trav->next;
            }
            return cnt;
        }
std::vector<ListNode*> splitListToParts(ListNode* head,int k){
            int size=count_nodes(head);

            int part_size=size/k;
            if(part_size==0) part_size=1;

            std::vector<int> sizes(k,part_size);

            if(size>k){
                int rem=size%k;
                std::fill(sizes.begin(),sizes.begin()+rem,part_size+1);
            }

            std::vector<ListNode*> ans(k,nullptr);

            ListNode* fast=head;
            ListNode* slow=head;
            ListNode* push=head;

            int i=0,j=0;
            while(fast){
                sizes[i]--;
                if(sizes[i]==0) {
                    ListNode* tmp=fast;
                    slow=fast->next;
                    tmp->next=nullptr;
                    ans[j++]=push;
                    push=fast=slow;
                    i++;
                }
                else fast=fast->next;
            }

            return ans;
        }
};
```