# 题目

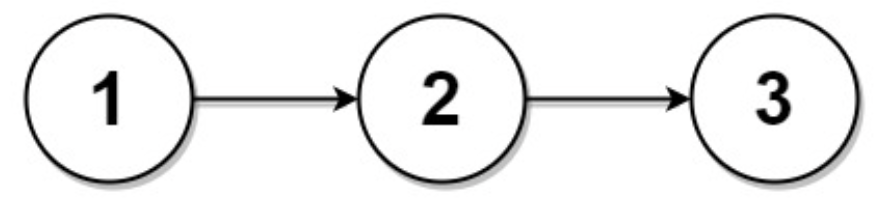
给你一个头结点为 head 的单链表和一个整数 k ，请你设计一个算法将链表分隔为 k 个连续的部分。

每部分的长度应该尽可能的相等：任意两部分的长度差距不能超过 1 。这可能会导致有些部分为 null 。

这 k 个部分应该按照在链表中出现的顺序排列，并且排在前面的部分的长度应该大于或等于排在后面的长度。

返回一个由上述 k 部分组成的数组。

示例 1：



输入：head = [1,2,3], k = 5

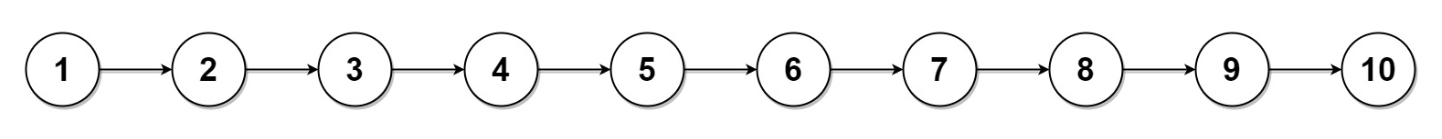
输出：[[1],[2],[3],[],[]]

解释：

第一个元素 output[0] 为 output[0].val = 1 ，output[0].next = null 。

最后一个元素 output[4] 为 null ，但它作为 ListNode 的字符串表示是 [] 。

示例 2：



输入：head = [1,2,3,4,5,6,7,8,9,10], k = 3

输出：[[1,2,3,4],[5,6,7],[8,9,10]]

解释：

输入被分成了几个连续的部分，并且每部分的长度相差不超过 1 。前面部分的长度大于等于后面部分的长度。

提示：

链表中节点的数目在范围 [0, 1000]

0 <= Node.val <= 1000

1 <= k <= 50

# 分析

/\*\*

\* 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) {}

\* };

\*/

class Solution {

public:

vector<ListNode\*> splitListToParts(ListNode\* head, int k) {

int len = 0;

ListNode\* curr = head;

// 计算链表长度

while (curr) {

len++;

curr = curr->next;

}

// 计算每个部分的基础长度和额外的部分数量

int baseLen = len / k;

int extra = len % k;

vector<ListNode\*> result;

curr = head;

for (int i = 0; i < k; i++) {

int partLen = baseLen + (i < extra ? 1 : 0);

if (partLen == 0) {

result.push\_back(nullptr);

} else {

result.push\_back(curr);

for (int j = 1; j < partLen; j++) {

curr = curr->next;

}

ListNode\* nextNode = curr->next;

curr->next = nullptr;

curr = nextNode;

}

}

return result;

}

};