Split Linked List in Parts

Given a (singly) linked list with head node `root`, write a function to 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 1. This may lead to some parts being null.

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

Return a List of ListNode's representing the linked list parts that are formed.

Examples 1->2->3->4, k = 5 // 5 equal parts [[1], [2], [3], [4], null]

Example 1:

```
Input:
root = [1, 2, 3], k = 5
Output: [[1],[2],[3],[],[]]
Explanation:
The input and each element of the output are ListNodes, not arrays.
For example, the input root has root.val = 1, root.next.val = 2, \root.next.next.val =
3, and root.next.next.next = null.
The first element output[0] has output[0].val = 1, output[0].next = null.
The last element output[4] is null, but it's string representation as a ListNode is []
.
```

Example 2:

```
Input:
root = [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 e
arlier parts are a larger size than the later parts.
```

Note:

- The length of `root` will be in the range `[0, 1000]`.
- Each value of a node in the input will be an integer in the range `[0, 999]`.
- `k` will be an integer in the range `[1, 50]`.

Solution 1

Java

```java
class Solution {
    public ListNode[] splitListToParts(ListNode root, int k) {
        ListNode[] parts = new ListNode[k];
        int len = 0;
        for (ListNode node = root; node != null; node = node.next)
            len++;
        int n = len / k, r = len % k; // n : minimum guaranteed part size; r : extr
a nodes spread to the first r parts;
        ListNode node = root, prev = null;
        for (int i = 0; i < k; i++, r--) {
            parts[i] = node;
            for (int j = 0; j < n + (r > 0 ? 1 : 0); j++) {
                prev = node;
                node = node.next;
            }
            if (prev != null) prev.next = null;
        }
        return parts;
    }
}
```

C++

```cpp
class Solution {
public:
    vector<ListNode*> splitListToParts(ListNode* root, int k) {
        vector<ListNode*> parts(k, nullptr);
        int len = 0;
        for (ListNode* node = root; node != nullptr; node = node->next)
            len++;
        int n = len / k, r = len % k; // n : minimum guaranteed part size; r : extr
a nodes spread to the first r parts;
        ListNode* node = root, *prev = nullptr;
        for (int i = 0; i < k; i++, r--) {
            parts[i] = node;
            for (int j = 0; j < n + (r > 0); j++) {
                prev = node;
                node = node->next;
            }
            if (prev) prev->next = nullptr;
        }
        return parts;
    }
};
```

written by alexander original link here

Solution 2

Find the size of the linked list and store that in `n`. Divide `n` by `k` to find out the size of each individual linked list `chunk` of the original linked list. `leftover` is the remainder of `n/k` and is `> 0` when `n` is NOT evenly divisible by `k`. These `leftover`s are evenly distributed one at a time onto the end of each of the linked list `chunk`s (starting at the beginning, not all chunks may be modified). If a `chunk` contains a `leftover` then that `chunk` size is a usual `chunk` size plus 1. Otherwise, the usual `chunk` size is used. Then iterate through the linked list, keeping track of the `head` of each `chunk`, using `prev` to follow `curr` in order to set the last `LinkNode`'s `next` pointer to `nullptr` and keep track of where the next iteration of the original linked list should re-occur. Push each `head` of each `chunk` into the vector `res`, which is the result to be returned. Lastly for the corner case (my one bug which cost me 5 minutes time in this contest) if k is larger than n, then append empty linked lists onto the end of `res`.

```
class Solution {
public:
    vector<ListNode*> splitListToParts(ListNode* root, int k) {
        vector<ListNode*> res;
        int n=0,chunk=0,leftover=0;
        ListNode* itr=root;
        while(itr){
            ++n;
            itr=itr->next;
        }
        chunk=n/k;
        leftover=n%k;
        itr=root;
        while(itr){
            int size=leftover-- > 0 ? chunk+1 : chunk;
            ListNode *head=itr, *prev=nullptr, *curr=itr;
            while(size){
                prev=curr;
                curr=curr->next;
                --size;
            }
            prev->next=nullptr;
            itr=curr;
            res.push_back(head);
        }
        while (k>n){
            res.push_back(nullptr);
            --k;
        }
        return res;
    }
};
```

More concise solution: initialize the vector with k nullptrs

```
class Solution {
public:
    vector<ListNode*> splitListToParts(ListNode* root, int k) {
        vector<ListNode*> res(k,nullptr);
        int n=0,chunk=0,leftover=0;
        ListNode* itr=root;
        while(itr){
            ++n;
            itr=itr->next;
        }
        chunk=n/k;
        leftover=n%k;
        itr=root;
        for (int i=0; itr; ++i){
            int size=leftover-- > 0 ? chunk+1 : chunk;
            ListNode *head=itr, *prev=nullptr, *curr=itr;
            for (int j=0; j<size; ++j){
                prev=curr;
                curr=curr->next;
            }
            prev->next=nullptr;
            itr=curr;
            res[i]=head;
        }
        return res;
    }
};
```

written by claytonjwong original link here

Solution 3

```
class Solution {
public:
    vector<ListNode*> splitListToParts(ListNode* root, int k) {
        int len = 0;
        for (ListNode *x = root; x; ++len, x = x->next);

        int n = len / k, r = len % k;
        vector<ListNode*> ret(k);
        for (int i = 0; i < k && root; ++i) {
            ret[i] = root;
            for (int j = 1; j < n + (i < r); ++j)
                root = root->next;
            ListNode *next = root->next;
            root->next = nullptr;
            root = next;
        }
        return ret;
    }
};
```

written by xpfxzxc original link here

From Leetcoder.