入选理由

- 合并 k 个有序数组相信大家都会，那你觉得这是分治么？链表你会么？

标签

- 分治

难度

- 中等

题目地址（23. 合并 K 个排序链表）

https://leetcode-cn.com/problems/merge-k-sorted-lists/

题目描述

```
合并 k 个排序链表，返回合并后的排序链表。请分析和描述算法的复杂度。

示例：

输入：
[
  1->4->5,
  1->3->4,
  2->6
]
输出：1->1->2->3->4->4->5->6
```

前置知识

- 链表
- 归并排序

公司

- 阿里
- 百度
- 腾讯
- 字节

思路

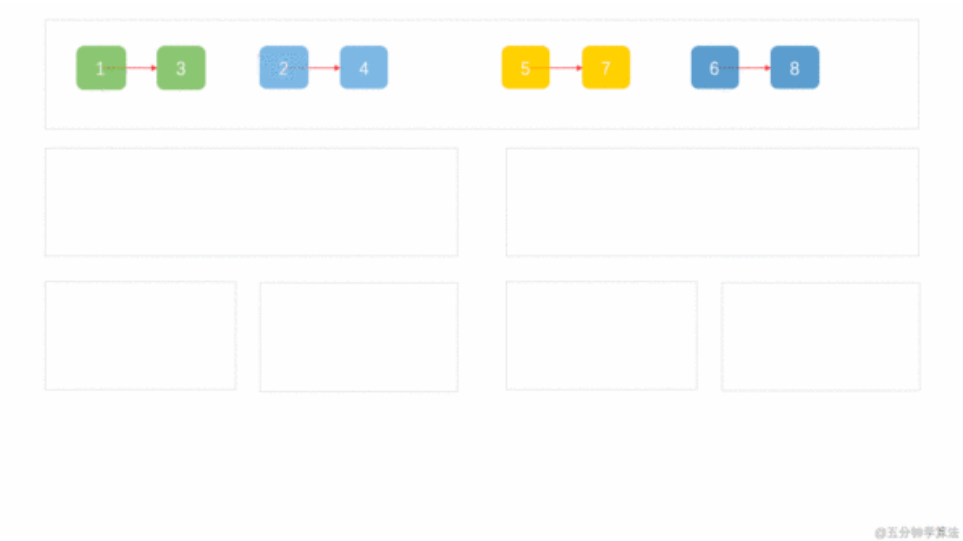
这道题目是合并 k 个已排序的链表，号称 leetcode 目前 最难 的链表题。 和之前我们解决的**88.merge-sorted-array**很像。 他们有两点区别：

1. 这道题的数据结构是链表，那道是数组。这个其实不复杂，毕竟都是线性的数据结构。

2. 这道题需要合并 k 个元素，那道则只需要合并两个。这个是两题的关键差别，也是这道题难度为 hard 的原因。

因此我们可以看出，这道题目是 88.merge-sorted-array 的进阶版本。其实思路也有点像，我们来具体分析下第二条。 如果你熟悉合并排序的话，你会发现它就是 合并排序的一部分 。

具体我们可以来看一个动画



23.merge-k-sorted-lists

（动画来自 https://zhuanlan.zhihu.com/p/61796021 ）

关键点解析

- 分治
- 归并排序(merge sort)

代码

代码支持 JavaScript， Python3, CPP

JavaScript Code：

```
/*
 * @lc app=leetcode id=23 lang=javascript
 *
 * [23] Merge k Sorted Lists
 *
 * https://leetcode.com/problems/merge-k-sorted-lists/description/
 *
 */
function mergeTwoLists(l1, l2) {
  const dummyHead = {};
  let current = dummyHead;
  // l1: 1 -> 3 -> 5
  // l2: 2 -> 4 -> 6
  while (l1 !== null && l2 !== null) {
    if (l1.val < l2.val) {
      current.next = l1; // 把小的添加到结果链表
      current = current.next; // 移动结果链表的指针
      l1 = l1.next; // 移动小的那个链表的指针
    } else {
      current.next = l2;
      current = current.next;
      l2 = l2.next;
    }
  }

  if (l1 === null) {
    current.next = l2;
  } else {
    current.next = l1;
  }
  return dummyHead.next;
}
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @param {ListNode[]} lists
 * @return {ListNode}
```

```
 */
var mergeKLists = function (lists) {
  // 图参考：  https://zhuanlan.zhihu.com/p/61796021
  if (lists.length === 0) return null;
  if (lists.length === 1) return lists[0];
  if (lists.length === 2) {
    return mergeTwoLists(lists[0], lists[1]);
  }

  const mid = lists.length >> 1;
  const l1 = [];
  for (let i = 0; i < mid; i++) {
    l1[i] = lists[i];
  }

  const l2 = [];
  for (let i = mid, j = 0; i < lists.length; i++, j++) {
    l2[j] = lists[i];
  }

  return mergeTwoLists(mergeKLists(l1), mergeKLists(l2));
};
```

Python3 Code：

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def mergeKLists(self, lists: List[ListNode]) -> ListNode:
        n = len(lists)

        # basic cases
        if lenth == 0: return None
        if lenth == 1: return lists[0]
        if lenth == 2: return self.mergeTwoLists(lists[0], lists[1])

        # divide and conqure if not basic cases
        mid = n // 2
        return self.mergeTwoLists(self.mergeKLists(lists[:mid]), self.mergeKLists(lists[mid:n]))


    def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
        res = ListNode(0)
        c1, c2, c3 = l1, l2, res
        while c1 or c2:
```

```
            if c1 and c2:
                if c1.val < c2.val:
                    c3.next = ListNode(c1.val)
                    c1 = c1.next
                else:
                    c3.next = ListNode(c2.val)
                    c2 = c2.next
                c3 = c3.next
            elif c1:
                c3.next = c1
                break
            else:
                c3.next = c2
                break

        return res.next
```

CPP Code:

```
class Solution {
private:
    ListNode* mergeTwoLists(ListNode* a, ListNode* b) {
        ListNode head(0), *tail = &head;
        while (a && b) {
            if (a->val < b->val) { tail->next = a; a = a->next; }
            else { tail->next = b; b = b->next; }
            tail = tail->next;
        }
        tail->next = a ? a : b;
        return head.next;
    }
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.empty()) return NULL;
        for (int N = lists.size(); N > 1; N = (N + 1) / 2) {
            for (int i = 0; i < N / 2; ++i) {
                lists[i] = mergeTwoLists(lists[i], lists[N - 1 - i]);
            }
        }
        return lists[0];
    }
};
```

复杂度分析

- 时间复杂度：$O(kn * logk)$
- 空间复杂度：$O(logk)$

相关题目

- 88.merge-sorted-array

扩展

这道题其实可以用堆来做，感兴趣的同学尝试一下吧。

大家对此有何看法，欢迎给我留言，我有时间都会一一查看回答。更多算法套路可以访问我的 LeetCode 题解仓库：https://github.com/azl397985856/leetcode 。 目前已经 37K star 啦。 大家也可以关注我的公众号《力扣加加》带你啃下算法这块硬骨头。

上一页 下一页

© 2020 lucifer. 保留所有权利