

分治算法在链表中的应用

原创 尚子皓 LOA算法学习笔记 2021-02-20 20:08

分治算法将原问题分解为规模较小的独立子问题，并将子问题的解组合为原问题的解。分治在数组结构的应用非常广泛，例如排序问题中有基于下标划分的归并排序，基于元素值划分的快速排序等。链表和数组的访问方式有所不同，本文通过两个例子展示在链表结构中如何应用分治算法。

01 排序链表

1.1 问题描述

给出一个单链表的头节点head，将链表中元素从小到大排序，并返回排序后的链表。

例如：输入链表[4,2,1,3]，正确输出为[1,2,3,4]。

1.2 算法描述

首先回顾一下数组中的常用排序算法。冒泡排序，插入排序，简单选择排序的平均时间复杂度均为 $O(n^2)$ ，暂不考虑。归并排序，堆排序和快速排序的时间复杂度为 $O(n\log n)$ 。数组中可以通过下标直接访问元素，链表则需要指针间接访问，在快速排序和堆排序中都有下标访问元素的需求，因此归并排序最适合于链表结构。

对链表进行归并排序的步骤如下：

(1) 找到链表的中点，将链表以中点为界分为左右两个子链表。寻找中点可以使用快慢指针。令快指针fast和慢指针slow均指向头结点head，fast每次移动两步，slow每次移动一步，当slow移动到链表末尾（指向NULL）时，fast指向位置即为链表的中点。

(2) 使用递归的方法对两个子链表进行排序。

(3) 将排序后的两个子链表进行合并（merge）。合并方法为用两个指针分别指向子链表的头结点，将当前两个指针中值较小的节点加入新建的合并链表，同时原指向该节点的指针后移，直到某一指针移动到链表末尾（指向NULL）停止。最后若某一链表有剩余节点，直接接在合并链表后面即可。

1.3 代码实现

```
1 struct ListNode {
2
3     int val;
4     ListNode *next;
5     ListNode() : val(0), next(nullptr) {}
6     ListNode(int x) : val(x), next(nullptr) {}
7     ListNode(int x, ListNode *next) : val(x), next(next) {}
8 };
9
10
11 ListNode* mergeSort(ListNode *head)
12 {
13     if(head == nullptr || head->next == nullptr) return head;
14     if(head->next->next == nullptr)
15     {
16         if(head->val > head->next->val)
17         {
```

```

18         int temp = head->val;
19         head->val = head->next->val;
20         head->next->val = temp;
21     }
22     return head;
23 }
24 ListNode *fast = head, *slow = head;
25 while(fast != nullptr && fast->next != nullptr)
26 {
27     fast = fast->next->next;
28     slow = slow->next;
29 }
30 ListNode *mid = slow->next;
31 slow->next = nullptr;
32 ListNode *leftHalf = mergeSort(head);
33 ListNode *rightHalf = mergeSort(mid);
34 return merge(leftHalf, rightHalf);
35 }
36 ListNode* merge(ListNode *l, ListNode *r)
37 {
38     ListNode *dummyNode = new ListNode(0);
39     ListNode *cur = dummyNode;
40     while(l != nullptr && r != nullptr)
41     {
42         if(l->val < r->val)
43         {
44             cur->next = l;
45             cur = cur->next;
46             l = l->next;
47         }
48         else
49         {
50             cur->next = r;
51             cur = cur->next;
52             r = r->next;
53         }
54     }
55     cur->next = (l == nullptr ? r : l);
56     return dummyNode->next;
57 }

```

02 合并k个排序链表

2.1 问题描述

排序链表指的是链表中的元素是从小到大排序的。给出k个排序链表，将所有链表合并为一个排序链表并返回。
 例如：输入3个链表为[[1,4,5],[1,3,4],[2,6]]，正确输出为[1,1,2,3,4,4,5,6]。

2.2 算法描述

(1) 顺序合并

合并两个排序链表的方法即排序链表问题中的merge过程，已经给出。对于k个排序链表的合并，最朴素的想法是先将其两个链表进行合并，合并的结果res再和第三个链表合并，以此类推，直到res和第k个链表合并。

时间复杂度：假设每个链表的长度为n，第i次合并后，res长度为i*n，第i次合并花费时间为O(i*n)，总时间复杂度为 $O(\sum_{i=1}^k (i * n)) = O(k^2 * n)$ 。

(2) 分治合并

我们尝试用分治法优化时间复杂度，采用类似归并排序的思想。先将k个链表两两配对，进行合并，得到k/2个链表，平均长度为2n/k，然后将k/2个链表两两合并为k/4个链表，以此类推，直到得到一个有序链表。

具体实现可以用递归的方法。递归终止条件为：输入链表数量为0时，返回NULL；数量为1，返回其本身；数量为2时，将两个排序链表合并后返回。输入链表数量大于2时，按顺序从中间分为左右两组链表，分别递归地进行合并，得两条排序链表。最后将这两条排序链表进行合并并返回结果。

时间复杂度：第一轮合并k/2组链表，每一组耗时O(2n)；第二轮合并k/4组链表，每一组耗时O(4n)。以此类推，总时间复杂度为 $O(\sum_{i=1}^{\infty} k/2^i * 2^i n) = O(\log k * kn)$ 。

2.3 代码实现

```
1 struct ListNode
2
3 {
4     int val;
5     ListNode *next;
6     ListNode(): val(0), next(nullptr){}
7     ListNode(int x): val(x), next(nullptr){}
8     ListNode(int x, ListNode *next): val(x), next(next){}
9 };
10
11
12 //顺序合并
13 ListNode* mergeKLists(vector& lists)
14 {
15     int N = lists.size();
16     if(N == 0) return nullptr;
17     ListNode* res = lists[0];
18     for(int i = 1; i < N; i++)
19     {
20         res = mergeTwoLists(res, lists[i]);
21     }
22     return res;
23 }
24 //分治合并
25 ListNode* mergeKLists(vector& lists)
26 {
27     int N = lists.size();
28     if(N == 0) return nullptr;
29     if(N == 1) return lists[0];
30     if(N == 2) return mergeTwoLists(lists[0], lists[1]);
```

```

31     int mid = N / 2;
32     vector leftLists(lists.begin(), lists.begin() + mid);
33     vector rightLists(lists.begin() + mid, lists.end());
34     return mergeTwoLists(mergeKLists(leftLists), mergeKLists(rightLists));
35 }
36 //合并两个排序链表
37 ListNode* mergeTwoLists(ListNode* l1, ListNode* l2)
38 {
39     if(l1 == nullptr && l2 == nullptr) return nullptr;
40     ListNode *l3 = new ListNode(-1);
41     ListNode *l3_head = l3;
42     while(l1 != nullptr && l2 != nullptr)
43     {
44         if(l1->val < l2->val)
45         {
46             l3->next = l1;
47             l3 = l3->next;
48             l1 = l1->next;
49         }
50         else
51         {
52             l3->next = l2;
53             l3 = l3->next;
54             l2 = l2->next;
55         }
56     }
57     l3->next = (l1 == nullptr ? l2 : l1);
58     return l3_head->next;
59 }

```

03 总结

本文通过排序链表和合并k个排序链表两个问题展示了分治算法在链表结构中的应用，往往可以有效地将时间复杂度从 $O(n^2)$ 级别降低到 $O(n \log n)$ 级别。链表中的元素不像数组可以用下标直接访问，因此对其处理时需要一些技巧，如用快慢指针求链表的中点。而对于多个链表的处理，我们可以先将每个链表看作一个整体，进行宏观上的处理，然后再用技巧处理链表中的每个节点，从而达到目的。

喜欢此内容的人还喜欢

LOA公众号关闭通知

LOA算法学习笔记



陈望道的学习观

团结报文史e家



锐评 | 日本年度防卫预算“脱缰”，总额首超6万亿日元
央广军事

