# 细节——不可忽视的要素

广东北江中学　李锐喆

# 一、引论

在程序设计中，算法是最核心的部分，往往一个优秀的算法能够取得比其他算法好的多的效率，但是在追求更好的算法的时候，我们往往会忽视算法中的一些细节处理。

在很多时候，我们自认为用到了最优的算法，但效果往往不尽如人意，为什么呢？**细节！**往往是细节上的一些瑕疵，导致算法的关键地方时间效率低下，甚至导致算法的时间复杂度远远高于正常的情况，最为严重的后果是导致整个算法的错误。

所以，我们说，细节在程序设计中是相当重要的。借用哲学原理来说，细节是算法这个整体的关键部分，而这个关键部分往往会对算法这个整体的性能状态起决定作用

细节——不可忽视的要素

# 一、引论

按照对算法的影响的性质和程度，我把细节分为这几种情况：

- **基本不影响算法的时间复杂度的细节处理。**这类细节处理对时间复杂度没有根本性的影响，仅仅对时间复杂度的系数产生影响。
- **影响到算法的时间复杂度的细节处理。**这类细节相当隐蔽，往往不为人所注意。但是这种细节对算法的影响相当大，处理得好与处理得不好往往会使程序的时间效率有质的区别。
- **影响到算法正确或错误的细节处理。**这类细节影响最大，在竞赛中很多选手在某些题目中已经找到解决方法却不能通过全部测试数据，往往就是这类的细节处理得不当导致。

# 一、引论

- 第三种情况大家都肯定感受颇深，要讲的话也必然是长篇大论，这里就不再赘述了。下面，我们主要对前两种情况分别进行分析讨论。

3. 基本不影响算法的时间复杂度的细节处理。

4. 影响到算法的时间复杂度的细节处理。

# 二、基本不影响算法时间复杂度的细节处理

这类细节对算法影响不算太大，但往往在关键的时候，时间复杂度的系数的大小对算法的效率也是有比较明显的影响的。例如对于某个细节，用方法 A 来处理的时间复杂度为$O(2n)$，而方法 B 来处理的时间复杂度为$O(n)$（这里为了方便描述，在复杂度式中加入不需要加的系数），如果用方法 A 来处理整个算法的时间消耗为 1s，那么用方法 B 来处理整个算法的时间消耗就为 2s！因此，尽可能地优化细节处理，从而使算法的时间复杂度的系数降低到一个比较低的程度。

下面我们通过一个例子来看看对这类细节的处理。

# 二、基本不影响算法时间复杂度的细节处理

***〖例 1〗线性表维护***

给出一个以字符串为数据类型的线性表，以及相应的若干个维护操作的规则，编写一个程序，模拟线性表的维护操作。

定义一个浮动指针，指向要处理的线性表元素。

定义四种对线性表的操作：

- 操作 1：插入。在线性表末尾插入数据。
- 操作 2：移动指针。把浮动指针向后移若干个单位。
- 操作 3：删除。删除从浮动指针所指元素开始的若干个元素。
- 操作 4：输出。输出浮动指针所指位置的元素。

请编写程序完成给出的任务。

# 二、基本不影响算法时间复杂度的细节处理

## 简要的分析

这道题目的模型很简单，由于需要动态增加、删除元素，而且空间没有限定范围，所以我们应该用动态指针来实现。

插入、移动、输出这些操作都不必细说，然而对于删除操作，这里是值得我们深究的。

# 二、基本不影响算法时间复杂度的细节处理

一般情况下，我们会用**方法 1** 来实现：每次删除元素时就把删除掉的每一个元素所占的空间释放出来，等以后需要插入元素时才重新分配使用。

删除的问题解决了，但是还是感到有些许不妥，那就是这样操作就是要频繁地分配和释放内存空间。释放内存空间操作的时间消耗并不大，但是分配内存空间时候由于内存空间的占用并不一定是连续的，会出现碎片的情况，因此寻找空间的时间消耗是相当可观的。有没有什么好的办法呢？

细节——不可忽视的要素

# 二、基本不影响算法时间复杂度的细节处理

- 删除元素要释放内存。
- 添加元素则要分配内存。

那么是否可以直接利用要删除元素的空间来进行添加元素操作，而不需要分配呢?

实际上这是完全可行[illegible]有了**方法 2**：

我们不妨把线性表称[illegible]表”，然后我们另外建一个链表，称之为“回[illegible]们把删除的元素放入回收链中，需要时再取出来[illegible]

# 二、基本不影响算法时间复杂度的细节处理

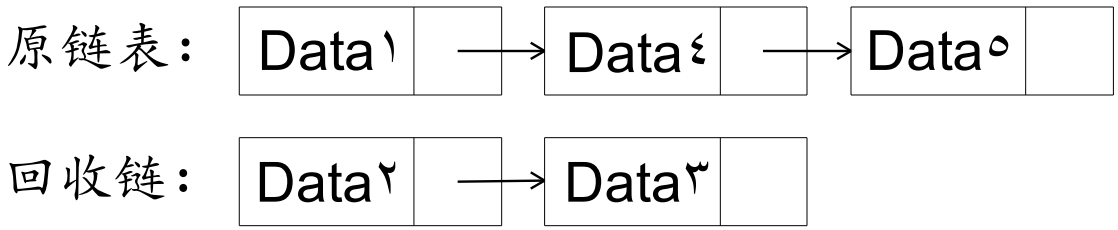
下面我们具体来看一下这个方法：

设现在我们有如下状态的原链表：

# 二、基本不影响算法时间复杂度的细节处理

删除Data2、Data3后

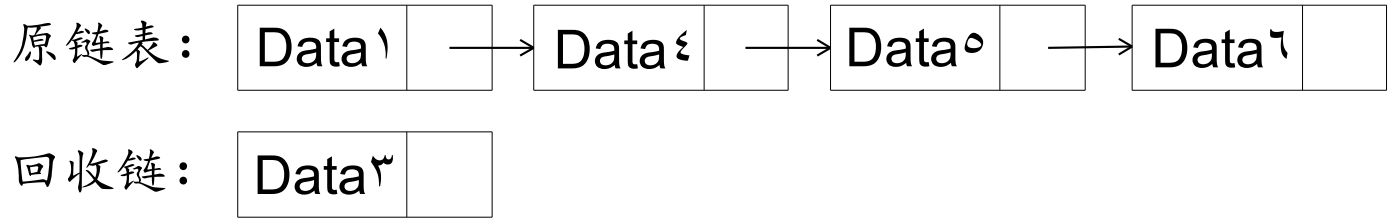
删除一系列元素，把它们放入回收链中。




添加Data6元素

由于回收链非空，所以把回收链的首元素Data2所用空间来存储Data6

加入一个元素，直接从回收链获取空间。




细节——不可忽视的要素

# 二、基本不影响算法时间复杂度的细节处理

经过这样优化之后，实际上需要向系统提出分配内存要求的次数就是整个维护操作中链表节点数目的最大值，而通常情况下，添加节点时是从回收链中获取空间进行分配，这样进行 n 次添加操作的时间复杂度仅仅是 $O(n)$ 。

**实际测试结果**

测试环境： AMD Duron 1.1GHz, 256MB SDRAM, Debian Linux + Kernel 2.6.0

测试系统： Celiz 0.0.3

| 测试点 | 原算法 | 优化后的算法 |
|---|---|---|
| 1 | 2.149s | 2.012s |
| 2 | 6.588s | 5.672s |
| 3 | 13.202s | 12.924s |
| 4 | 5.086s | 4.911s |
| 5 | 10.297s | 9.918s |

# 二、基本不影响算法时间复杂度的细节处理

## 小结

由此看来，虽然这类细节处理对算法的时间复杂度影响不算太大，但是处理得好，还是有不少益处的。

# 三、影响到算法时间复杂度的细节处理

相对于上面的那类细节，这种细节对算法的时间效率有相当的影响。而且我们在处理这类细节的时候往往会误入不恰当的处理方式之中，使算法的时间复杂度升高。因此，能否处理好这类细节，是算法是否能有效解决问题的关键。

下面我们来看一个例子，看看我们一个相当熟悉的算法是怎么处理好细节的。

# 三、影响到算法时间复杂度的细节处理

***〖例 2 〗银河英雄传说（ NOI2002 第一试第一题）***

（问题描述略）

分析这道题目，可以把每列划分成一个集合，那么，舰队的合并、查询就是对集合的合并和查询了，这样就是一个很典型的并查集算法的模型。

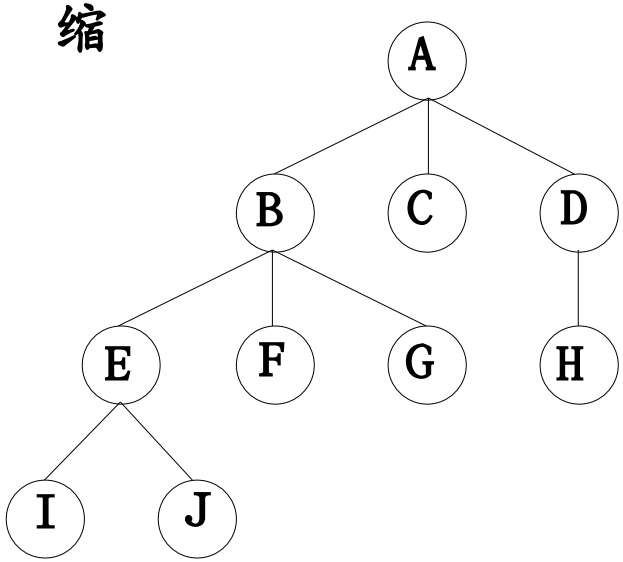
# 三、影响到算法时间复杂度的细节处理

但是单纯的并查集算法还是有缺点的，合并可能使 n 个节点的树退化成一条链，这样将大大影响查询的效率，因此必须进行优化。并查集常用的两种优化方式是**把小树合并到大树上**以及**路径压缩**。

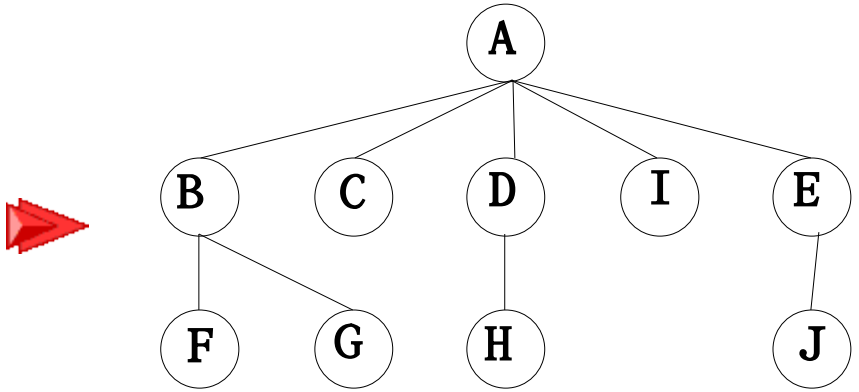
各类教材都告诉我们，路径压缩是在查找的过程中进行的对此我们也许会有这样的疑问，为什么要在查找中进行，而合并过程中并不进行呢?

# 三、影响到算法时间复杂度的细节处理

**查找的过程中进行路径压缩**




原并查集树



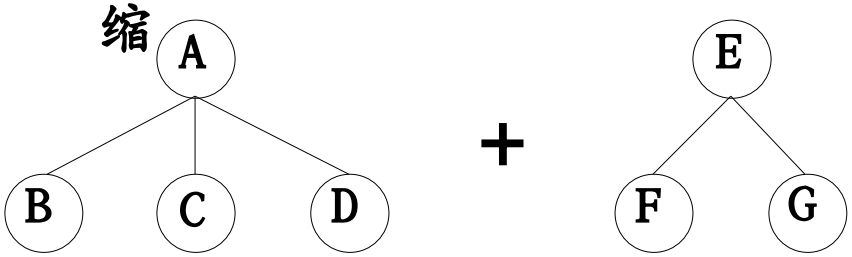

在查找I的过程中对从I到A的路径进行压缩后的并查集树

查找 :$O(\log n)$　　合并 :$O(\log n)$

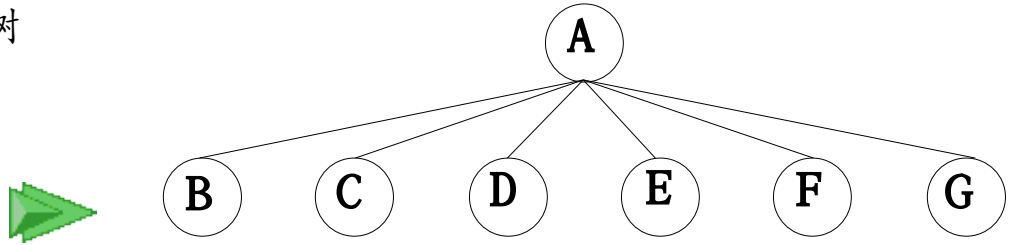
细节——不可忽视的要素

# 三、影响到算法时间复杂度的细节处理

**合并的过程中进行路径压缩**




合并前的两棵并查集树




合并时进行路径压缩后的并查集树

查找：$O(1)$　　合并：$O(n)$

细节——不可忽视的要素

# 三、影响到算法时间复杂度的细节处理

| 时间复杂度 | 查找 | 合并 |
| --- | --- | --- |
| 查找时路径压缩 | $O(\log n)$ | $O(\log n)$ |
| 合并时路径压缩 | $O(1)$ | $O(n)$ |

效率似乎各有千秋。

但在查找上，前者查找 n 次的实际复杂度为 $O(n\alpha(n))$ ，即平均每次操作接近 $O(1)$ 。

而在合并上，$O(\log n)$ 对比 $O(n)$ 还是有比较大的优势的。

因此，查找时进行路径压缩的时间效率比合并时进行路径压缩的时间效率是要高的，在查找时进行路径压缩是有道理的。

细节——不可忽视的要素

# 三、影响到算法时间复杂度的细节处理

***〖例 3〗铁路***

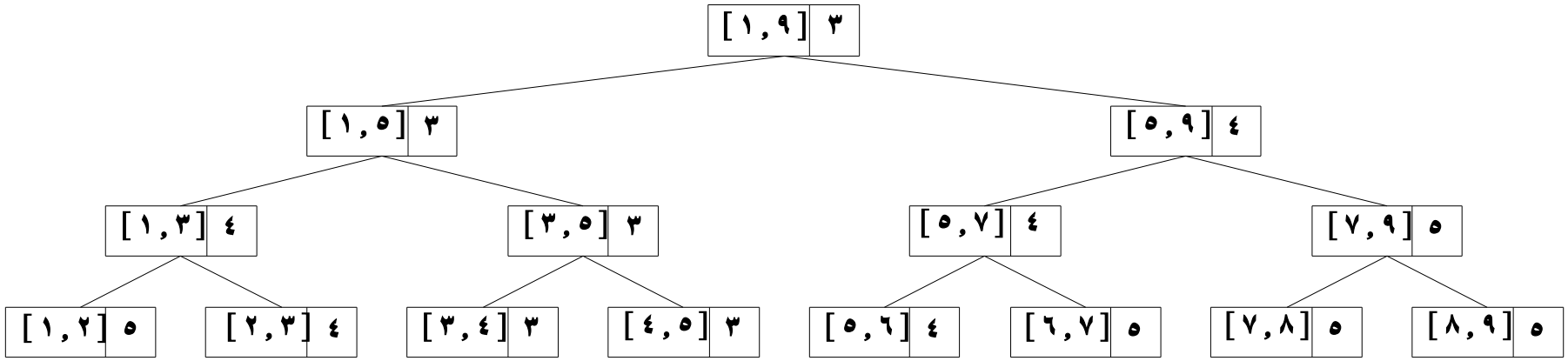
Byteotian 州铁道部决定赶上时代，为此他们引进了城市联网。假设城市联网顺次连接着 $n$ 个城市，从 1 到 $n$ 编号（起始城市编号为 1，终止城市编号为 $n$）。每辆火车有 $m$ 个座位且在任何两个车站之间运送更多的乘客是不允许的。电脑系统将收到连续的预订请求并决定是否满足他们的请求。当火车在被请求的路段上有足够的空位时，就通过这个请求，否则不通过。通过请求的一部分是不允许的。通过一个请求之后，火车里的空位数目将得到更新。请求应按照收到的顺序依次处理。（$n$<=60000）

任务：计算哪些请求可以通过，哪些请求不能通过。

# 三、影响到算法时间复杂度的细节处理

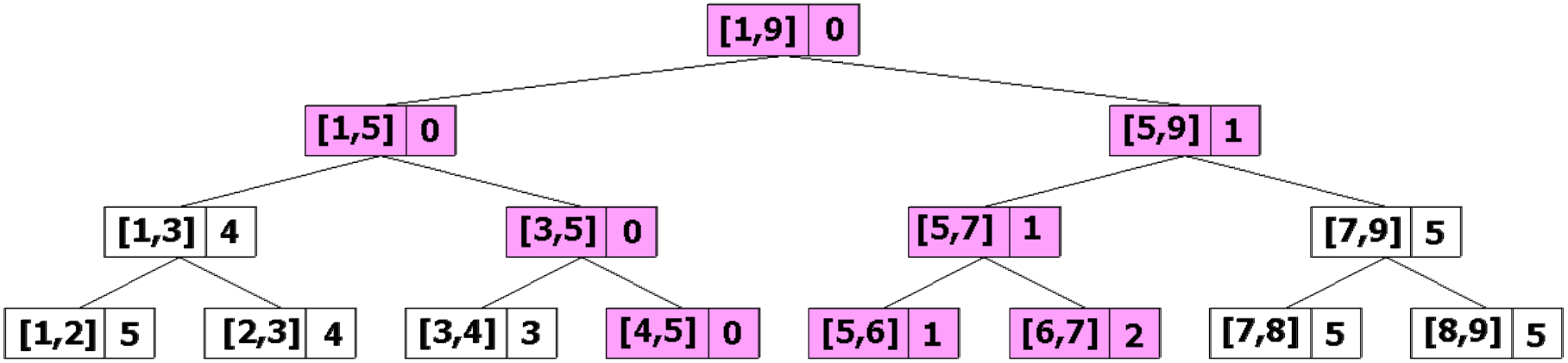
## 简要的分析

因为 n ⩾ 60000 ，所以用线性表来存储明显是行不通的。需要采用线段树的结构来实现，把整个铁路划分好之后，按照二叉树的思想来存储。



# 三、影响到算法时间复杂度的细节处理

我们往往会这样做：




✓查找操作　　时间复杂度：$O(\log n)$

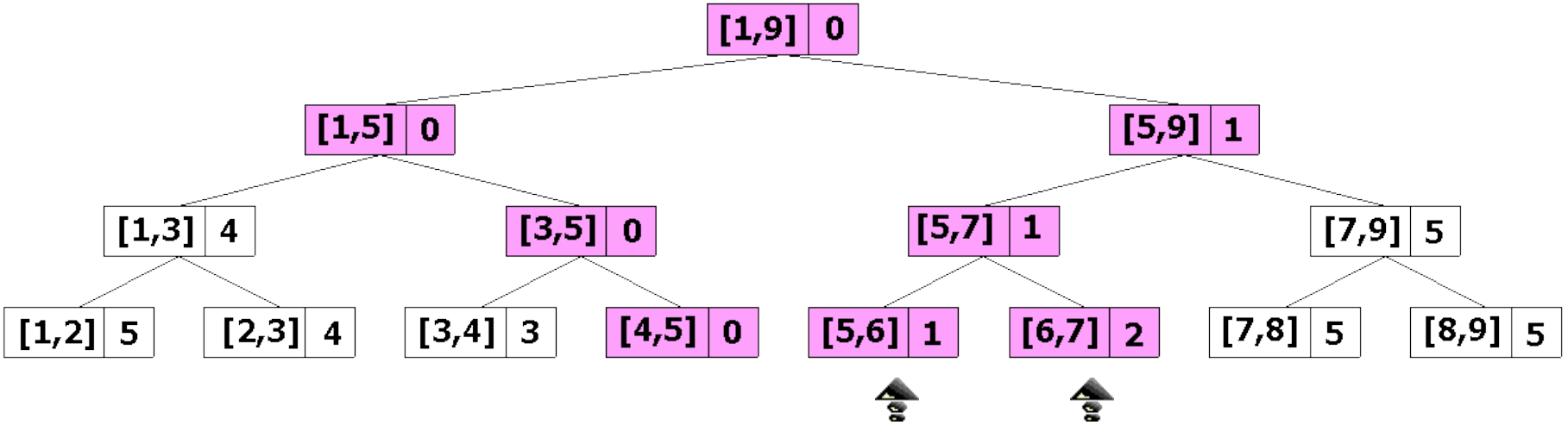
✓修改操作　　时间复杂度：$\geq O(n)$

**复杂度太高了！**

细节——不可忽视的要素

# 三、影响到算法时间复杂度的细节处理

我们换一个思维角度来考虑。



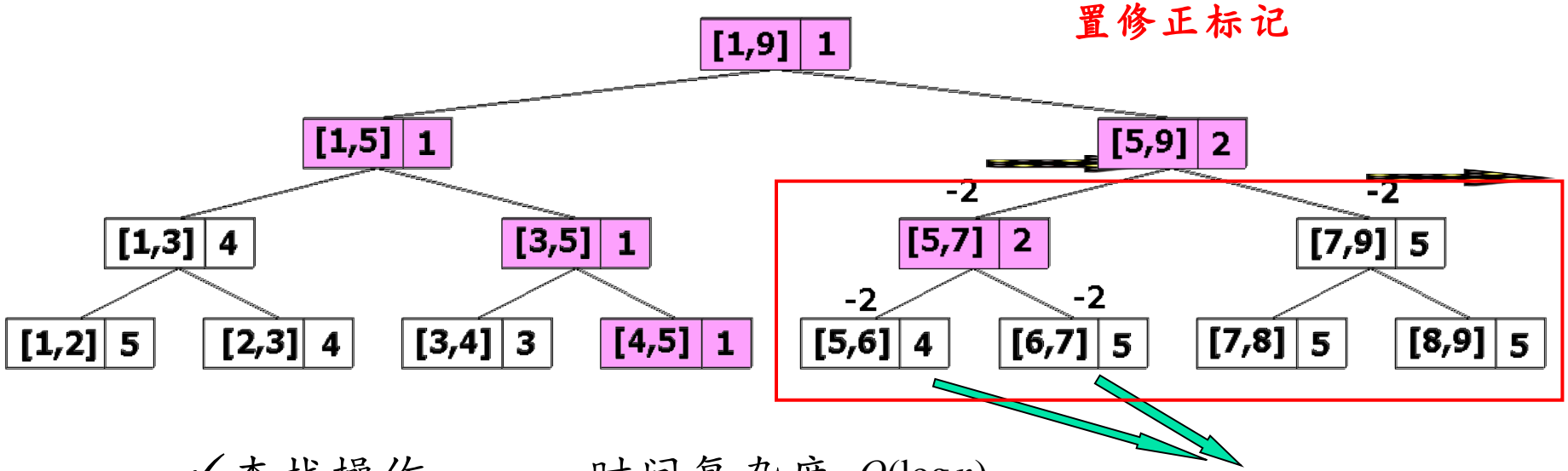

这两个也许是不必要修改的！

**方向：** 减少不必要的修改！

细节——不可忽视的要素

# 三、影响到算法时间复杂度的细节处理

根据这个方向，我们有了这个方法：




✓查找操作　　时间复杂度 :$O(\log n)$

✓修改操作　　时间复杂度 :$O(\log n)$

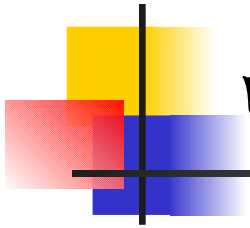
以后查找或修改时，用修改标记修正，并把标记向下传递。

细节——不可忽视的要素

# 三、影响到算法时间复杂度的细节处理

## 小结

由上看来，看似小小的一些细节处理，却让算法的时间复杂度产生了截然不同的两种情况。因此我们在注重算法的同时，也不应该忽略细节的处理，应该仔细揣度每个细节应该怎么处理，而不要在细节上犯错误，进而影响到整个算法的实现。



# 四、总结

关键性的细节对整个算法起着举足轻重的作用，正如我们上面所阐述的那样，关键细节处理的优劣，直接影响到算法的正确性，就算不影响正确性，也会或多或少地影响到算法的时间效率。

因此，我们在算法实现时要认真进行分析，仔细研究自己的细节处理是否恰当，不应该疏忽大意，否则将得不偿失。

# 谢谢

大