快速排序

对于包含 n 个数的输入数组来说，快速排序是一种最坏情况时间复杂度为 $\Theta(n^2)$ 的排序算法。虽然最坏情况时间复杂度很差，但是快速排序通常是实际排序应用中最好的选择，因为它的平均性能非常好：它的期望时间复杂度是 $\Theta(n\lg n)$，而且 $\Theta(n\lg n)$ 中隐含的常数因子非常小。另外，它还能够进行原址排序(见 2.1 节)，甚至在虚存环境中也能很好地工作。

7.1 节将描述快速排序算法及它的一个重要的划分子程序。因为快速排序的运行情况比较复杂，在 7.2 节中，我们先对其性能进行一个直观的讨论，在本章的最后会给出一个准确的分析。在 7.3 节中，我们会介绍一个基于随机抽样的快速排序算法。这一算法的期望时间复杂度较好，而且没有什么特殊的输入会导致最坏情况的发生。7.4 节对这一随机算法的分析表明，其最坏情况时间复杂度是 $\Theta(n^2)$；在元素互异的情况下，期望时间复杂度 $O(n\lg n)$。

7.1 快速排序的描述

与归并排序一样，快速排序也使用了 2.3.1 节介绍的分治思想。下面是对一个典型的子数组 $A[p..r]$ 进行快速排序的三步分治过程：

分解：数组 $A[p..r]$ 被划分为两个(可能为空)子数组 $A[p..q-1]$ 和 $A[q+1..r]$，使得 $A[p..q-1]$ 中的每一个元素都小于等于 $A[q]$，而 $A[q]$ 也小于等于 $A[q+1..r]$ 中的每个元素。其中，计算下标 q 也是划分过程的一部分。

解决：通过递归调用快速排序，对子数组 $A[p..q-1]$ 和 $A[q+1..r]$ 进行排序。 170

合并：因为子数组都是原址排序的，所以不需要合并操作：数组 $A[p..r]$ 已经有序。

下面的程序实现快速排序：

```
QUICKSORT(A, p, r)
1  if  p < r
2       q = PARTITION(A, p, r)
3       QUICKSORT(A, p, q−1)
4       QUICKSORT(A, q+ 1, r)
```

为了排序一个数组 A 的全部元素，初始调用是 QUICKSORT(A，1，$A.length$)。

数组的划分

算法的关键部分是 PARTITION 过程，它实现了对子数组 $A[p..r]$ 的原址重排。

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p−1
3  for j = p to r−1
4       if A[j]≤x
5            i = i + 1
6            exchange A[i] with A[j]
7  exchange A[i+ 1] with A[r]
8  return i + 1
```

图 7-1 显示了 PARTITION 如何在一个包含 8 个元素的数组上进行操作的过程。PARTITION 总是选择一个 $x=A[r]$ 作为**主元**(pivot element)，并围绕它来划分子数组 $A[p..r]$。

随着程序的执行，数组被划分成4个(可能有空的)区域。在第3～6行的**for**循环的每一轮迭代的开始，每一个区域都满足一定的性质，如图7-2所示。我们将这些性质作为循环不变量：

在第3～6行循环体的每一轮迭代开始时，对于任意数组下标k，有：

1. 若$p \leqslant k \leqslant i$，则$A[k] \leqslant x$。
2. 若$i+1 \leqslant k \leqslant j-1$，则$A[k] > x$。
3. 若$k=r$，则$A[k]=x$。

171

但是上述三种情况没有覆盖下标j到$r-1$，对应位置的值与主元之间也不存在特定的大小关系。

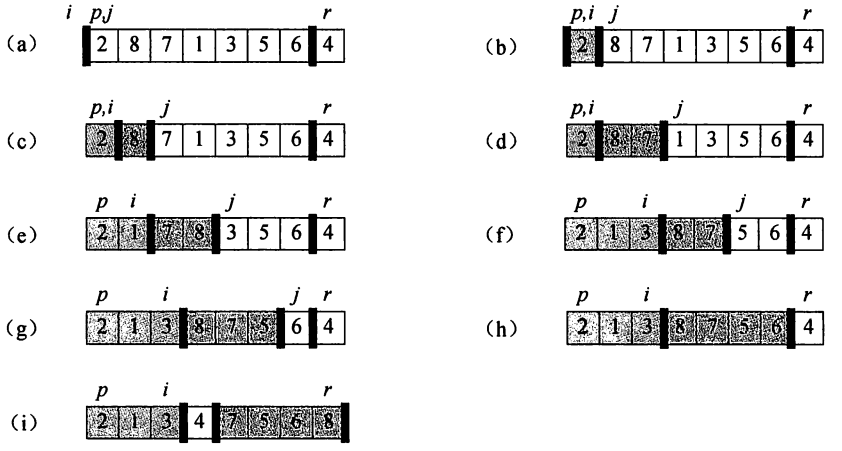

图7-1 在一个样例数组上的PARTITION操作过程。数组项$A[r]$是主元x。浅阴影部分的数组元素都在划分的第一部分，其值都不大于x。深阴影部分的元素都在划分的第二部分，其值都大于x。无阴影的元素则是还未分入这两个部分中的任意一个。最后的白色元素就是主元x。(a)初始的数组和变量设置。数组元素均未被放入前两个部分中的任何一个。(b)2与它自身进行交换，并被放入了元素值较小的那个部分。(c)～(d)8和7被添加到元素值较大的那个部分中。(e)1和8进行交换，数值较小的部分规模增加。(f)数值3和7进行交换，数值较小的部分规模增加。(g)～(h)5和6被包含进较大部分，循环结束。(i)在第7～8行中，主元被交换，这样主元就位于两个部分之间

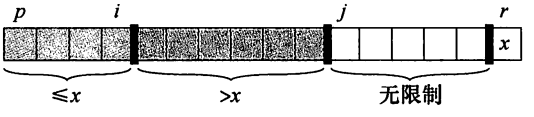

图7-2 在子数组$A[p..r]$上，PARTITION维护了4个区域。$A[p..i]$区间内的所有值都小于等于x，$A[i+1..j-1]$区间内的所有值都大于x，$A[r]=x$。子数组$A[j..r-1]$中的值可能属于任何一种情况

我们需要证明这个循环不变量在第一轮迭代之前是成立的，并且在每一轮迭代后仍然都成立。在循环结束时，该循环不变量还可以为证明正确性提供有用的性质。

172

初始化：在循环的第一轮迭代开始之前，$i=p-1$和$j=p$。因为在p和i之间、$i+1$和$j-1$之间都不存在值，所以循环不变量的前两个条件显然都满足。第1行中的赋值操作满足了第三个条件。

保持：如图7-3所示，根据第4行中条件判断的不同结果，我们需要考虑两种情况。图7-3(a)显示当$A[j] > x$时的情况：循环体的唯一操作是j的值加1。在j值增加后，对$A[j-1]$，条件2成立，且所有其他项都保持不变。图7-3(b)显示当$A[j] \leqslant x$时的情况：将i值加1，交换$A[i]$和$A[j]$，再将j值加1。因为进行了交换，现在有$A[i] \leqslant x$，所以条件1得到满足。类似地，我们也能得到$A[j-1] > x$。因为根据循环不变量，被交换进$A[j-1]$的值总是大于x的。

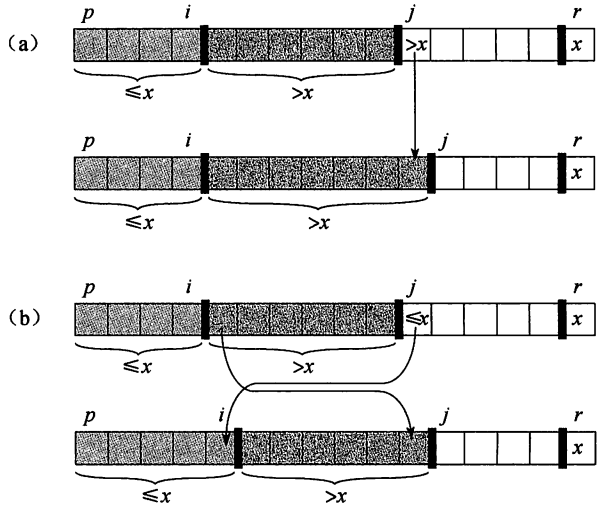

图 7-3 PARTITION 的一次迭代中会有两种可能的情况：(a)如果 $A[j]>x$，需要做的只是将 j 的值加 1，从而使循环不变量继续保持。(b)如果 $A[j]\leqslant x$，则将下标 i 的值加 1，并交换 $A[i]$和 $A[j]$，再将 j 的值加 1。此时，循环不变量同样得到保持

终止：当终止时，$j=r$。于是，数组中的每个元素都必然属于循环不变量所描述的三个集合的一个，也就是说，我们已经将数组中的所有元素划分成了三个集合：包含了所有小于等于 x 的元素的集合、包含了所有大于 x 的元素的集合和只有一个元素 x 的集合。

在 PARTITION 的最后两行中，通过将主元与最左的大于 x 的元素进行交换，就可以将主元移到它在数组中的正确位置上，并返回主元的新下标。此时，PARTITION 的输出满足划分步骤规定的条件。实际上，一个更严格的条件也可以得到满足：在执行完 QUICKSORT 的第 2 行之后，$A[q]$严格小于 $A[q+1..r]$内的每一个元素。

PARTITION 在子数组 $A[p..r]$上的时间复杂度是 $\Theta(n)$，其中 $n=r-p+1$(见练习 7.1-3)。

练习

7.1-1 参照图 7-1 的方法，说明 PARTITION 在数组 $A=\langle 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11\rangle$上的操作过程。

7.1-2 当数组 $A[p..r]$中的元素都相同时，PARTITION 返回的 q 值是什么？修改 PARTITION，使得当数组 $A[p..r]$中所有元素的值都相同时，$q=\lfloor (p+r)/2\rfloor$。

7.1-3 请简要地证明：在规模为 n 的子数组上，PARTITION 的时间复杂度为 $\Theta(n)$。

7.1-4 如何修改 QUICKSORT，使得它能够以非递增序进行排序？

7.2 快速排序的性能

173 ~ 174

快速排序的运行时间依赖于划分是否平衡，而平衡与否又依赖于用于划分的元素。如果划分是平衡的，那么快速排序算法性能与归并排序一样。如果划分是不平衡的，那么快速排序的性能就接近于插入排序了。在本节中，我们将给出划分为平衡或不平衡时快速排序性能的非形式化的分析。

最坏情况划分

当划分产生的两个子问题分别包含了 $n-1$ 个元素和 0 个元素时，快速排序的最坏情况发生了(证明见 7.4.1 节)。不妨假设算法的每一次递归调用中都出现了这种不平衡划分。划分操作的时间复杂度是 $\Theta(n)$。由于对一个大小为 0 的数组进行递归调用会直接返回，因此 $T(0)=\Theta(1)$，

于是算法运行时间的递归式可以表示为：

$$T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$$

从直观上来看，每一层递归的代价可以被累加起来，从而得到一个算术级数(公式(A.2))，其结果为 $\Theta(n^2)$。实际上，利用代入法可以直接得到递归式 $T(n)=T(n-1)+\Theta(n)$ 的解为 $T(n)=\Theta(n^2)$(见练习 7.2-1)。

因此，如果在算法的每一层递归上，划分都是最大程度不平衡的，那么算法的时间复杂度就是 $\Theta(n^2)$。也就是说，在最坏情况下，快速排序算法的运行时间并不比插入排序更好。此外，当输入数组已经完全有序时，快速排序的时间复杂度仍然为 $\Theta(n^2)$。而在同样情况下，插入排序的时间复杂度为 $O(n)$。

最好情况划分

在可能的最平衡的划分中，PARTITION 得到的两个子问题的规模都不大于 $n/2$。这是因为其中一个子问题的规模为 $\lfloor n/2 \rfloor$，而另一个子问题的规模为 $\lceil n/2 \rceil - 1$。在这种情况下，快速排序的性能非常好。此时，算法运行时间的递归式为：

$$T(n) = 2T(n/2) + \Theta(n)$$

在上式中，我们忽略了一些余项以及减 1 操作的影响。根据主定理(定理 4.1)的情况 2，上述递归式的解为 $T(n)=\Theta(n\lg n)$。通过在每一层递归中都平衡划分子数组，我们得到了一个渐近时间上更快的算法。

平衡的划分

快速排序的平均运行时间更接近于其最好情况，而非最坏情况。详细的分析可以参看本书 7.4 节。理解这一点的关键就是理解划分的平衡性是如何反映到描述运行时间的递归式上的。

175

例如，假设划分算法总是产生 9∶1 的划分，乍一看，这种划分是很不平衡的。此时，我们得到的快速排序时间复杂度的递归式为：

$$T(n) = T(9n/10) + T(n/10) + cn$$

这里，我们显式地写出了 $\Theta(n)$ 项中所隐含的常数 c。图 7-4 显示了这一递归调用所对应的递归树。注意，树中每一层的代价都是 cn，直到在深度 $\log_{10} n = \Theta(\lg n)$ 处达到递归的边界条件时为

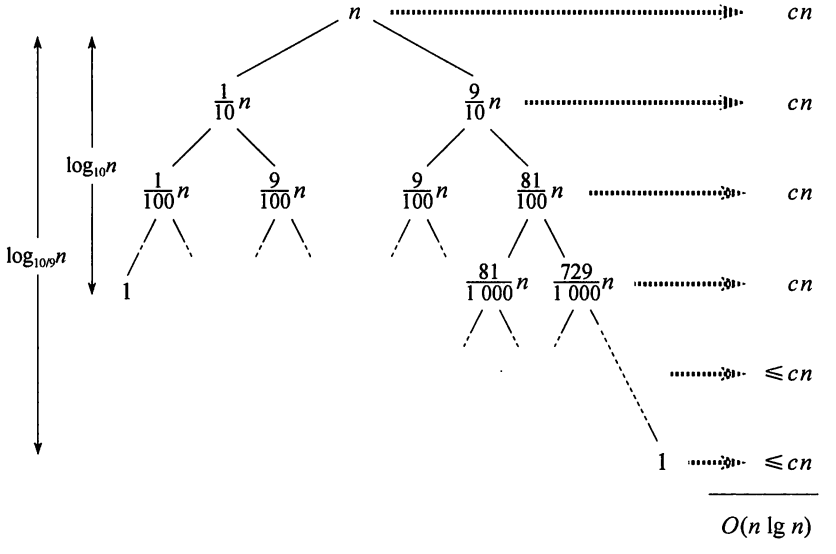

图 7-4 QUICKSORT 的一棵递归树，其中 PARTITION 总是产生 9∶1 的划分。该树的时间复杂度为 $O(n\lg n)$。每个结点的值表示子问题的规模，每一层的代价显示在最右边。每一层的代价包含了 $\Theta(n)$ 项中隐含的常数 c

止，之后每层代价至多为 cn。递归在深度为 $\log_{10/9} n = \Theta(\lg n)$ 处终止。因此，快速排序的总代价为 $O(n \lg n)$。因此，即使在递归的每一层上都是 9∶1 的划分，直观上看起来非常不平衡，但快速排序的运行时间是 $O(n \lg n)$，与恰好在中间划分的渐近运行时间是一样的。实际上，即使是 99∶1的划分，其时间复杂度仍然是 $O(n \lg n)$。事实上，任何一种常数比例的划分都会产生深度为 $\Theta(\lg n)$的递归树，其中每一层的时间代价都是 $O(n)$。因此，只要划分是常数比例的，算法的运行时间总是 $O(n \lg n)$。 176

对于平均情况的直观观察

为了对快速排序的各种随机情况有一个清楚的认识，我们需要对遇到各种输入的出现频率做出假设。快速排序的行为依赖于输入数组中元素的值的相对顺序，而不是某些特定值本身。与 5.2 节中对雇用问题所做的概率分析类似，这里我们也假设输入数据的所有排列都是等概率的。

当对一个随机输入的数组运行快速排序时，想要像前面非形式化分析中所假设的那样，在每一层上都有同样的划分是不太可能的。我们预期某些划分会比较平衡，而另一些则会很不平衡。例如，在练习 7.2-6 中，会要求读者说明 PARTITION 所产生的划分中 80%以上都比 9∶1 更平衡，而另 20%的划分则比 9∶1 更不平衡。

在平均情况下，PARTITION 所产生的划分同时混合有“好”和“差”的划分。此时，在与 PARTITION 平均情况执行过程所对应的递归树中，好和差的划分是随机分布的。基于直觉，假设好和差的划分交替出现在树的各层上，并且好的划分是最好情况划分，而差的划分是最坏情况划分，图 7-5(a)显示出了递归树的连续两层上的划分情况。在根结点处，划分的代价为 n，划分产生的两个子数组的大小为 $n-1$ 和 0，即最坏情况。在下一层上，大小为 $n-1$ 的子数组按最好情况划分成大小分别为 $(n-1)/2-1$ 和 $(n-1)/2$ 的子数组。在这里，我们假设大小为 0 的子数组的边界条件代价为 1。

在一个差的划分后面接着一个好的划分，这种组合产生出三个子数组，大小分别为 0、$(n-1)/2-1$ 和 $(n-1)/2$。这一组合的划分代价为 $\Theta(n)+\Theta(n-1)=\Theta(n)$。该代价并不比图 7-5(b)中的更差。在图 7-5(b)中，一层划分就产生出大小为 $(n-1)/2$ 的两个子数组，划分代价为 $\Theta(n)$。但是，后者的划分是平衡的！从直观上看，差划分的代价 $\Theta(n-1)$可以被吸收到好划分的代价$\Theta(n)$中去，而得到的划分结果也是好的。因此，当好和差的划分交替出现时，快速排序的时间复杂度与全是好的划分时一样，仍然是 $O(n \lg n)$。区别只是 O 符号中隐含的常数因子要略大一些。在 7.4.2 节中，我们将给出一个关于随机输入情况下快速排序的期望时间复杂度的更严格的分析。

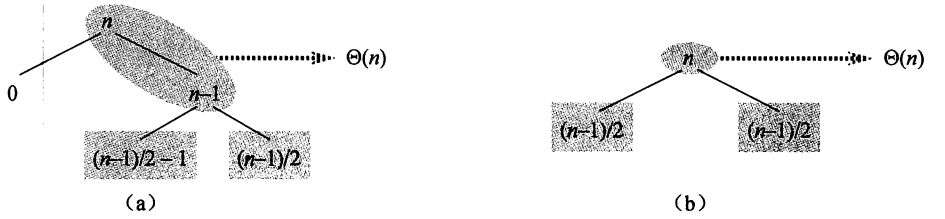

图 7-5 (a)一棵快速排序递归树的两层。在根结点这一层的划分代价是 n，产生了一个“坏”的划分：两个子数组的大小分别为 0 和 $n-1$。对大小为 $n-1$ 的子数组的划分代价为 $n-1$，并产生了一个“好”的划分：大小分别为 $(n-1)/2-1$ 和 $(n-1)/2$ 的子数组。(b)一棵非常平衡的递归树中的一层。在两棵树中，椭圆阴影所示的子问题的划分代价都是 $\Theta(n)$。可以看出，(a)中以矩形阴影显示的待解决子问题的规模并不大于(b)中对应的待解决子问题

练习

7.2-1 利用代入法证明：正如 7.2 节开头提到的那样，递归式 $T(n)=T(n-1)+\Theta(n)$的解为 $T(n)=\Theta(n^2)$。