目录

湖南工商大学 算法导论 课程教案

授课题目（教学章、节或主题） 课时安排: 2学时

第六讲：快速排序 授课时间 :第六周周一第1、2节

教学内容（包括基本内容、重点、难点）：

基本内容： （1）基本概念：快速排序算法；

（2）快速排序程序；

（3）时间复杂度分析；

（4）快速排序的随机化版本.

教学重点、难点：重点为快速排序算法原理、时间复杂度分析

教学媒体的选择：本章使用大数据分析软件Jupyter教学，Jupyter集课件、Python程序运行、HTML网页制作、Pdf文档生成、Latex文档编译于一身，是算法导论课程教学的最佳选择。

板书设计：黑板分为上下两块，第一块基本定义，推导证明以及例子放在第二块。第一块 整个课堂不擦洗，以便学生随时看到算法流程图以及基本算法理论等内容。

课程过程设计： （1）讲解基本算法理论； （2）举例说明； （3）程序设计与编译； （4）对本课堂进行总 结、讨论； （5）布置作业与实验报告

第六讲 快速排序

一. 快速排序算法的描述

快速排序是最常用的一种排序算法，包括C的qsort，C++和Java的sort，都采用了快排（C++和Java的sort经过了优化，还混合了其他排序算法）。

快排最坏情况 $O(n^2)$，但平均效率 $O(n \lg n)$，而且这个 $O(n \lg n)$ 记号中隐含的常数因子很小，快排可以说是最快的排序算法，它还是就地排序。

快速排序是基于分治策略的。对一个子数组A[p…r]快速排序的分治过程的三个步骤为：

1、分解

数组A[p…r]被划分成两个（可能空）子数组A[p…q-1]和A[q+1…r]，使得A[p…q-1]中的每个元素都小于等于A[q]，且小于等于A[q+1…r]中的元素。下标q也在这个划分过程中进行计算。

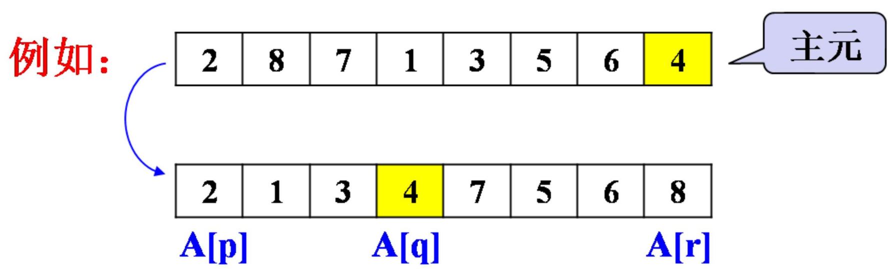
2、解决

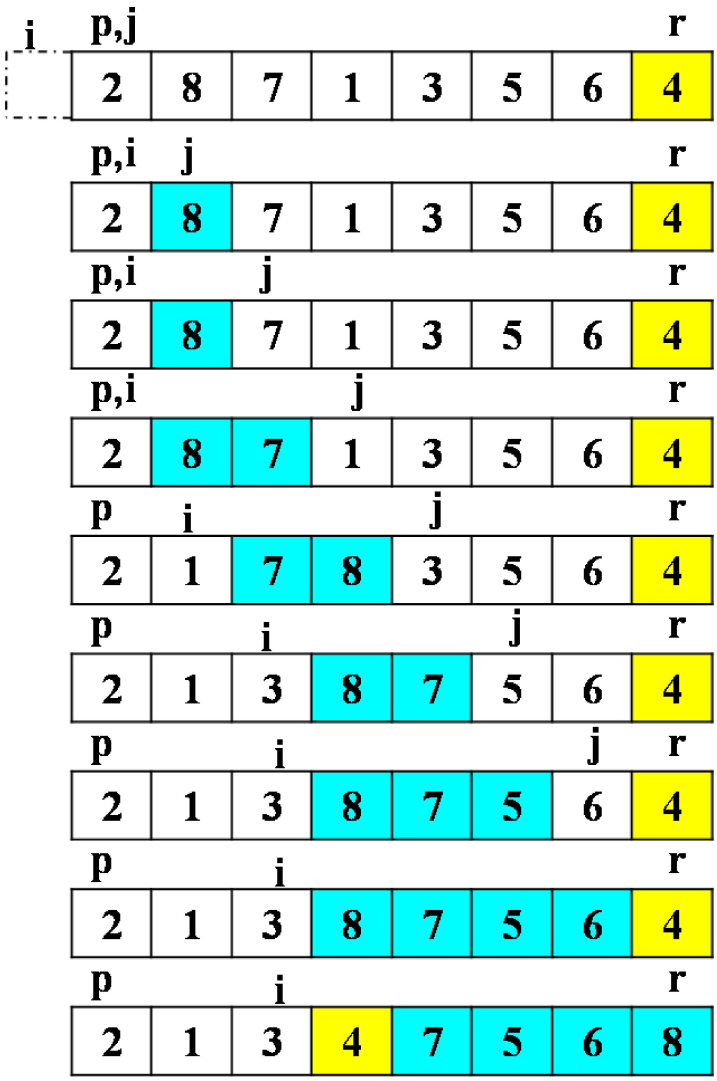
通过递归调用快速排序，对子数组A[p…q-1]和A[q+1…r]排序。

3、合并

因为两个子数组就是原地排序的，将它们的合并不需要操作：整个数组A[p…r]已排序。

数组的划分：主元归位过程





二. 快速排序代码实现

快速排序伪代码

```
 QuickSort(A, p, r)
if  p < r
      q=Partition(A, p, r)
      QuickSort(A, p, q-1)
      QuickSort(A, q+1, r)

Partition(A, p, r)
x = A[r]
i  = p-1
for  j = p  to r-1
     if  A[j]<=x
         i=i+1
         exchange A[i] with A[j]
exchange A[i+1] with A[r]
Return  i+1
```

我们给出PARTITION代码中第3行至第8行迭代的循环不变式:

每一轮迭代的开始，对于任何数组下标k，有:

- 如果 $p \leq k \leq i$，则 $A[k] \leq x$。
- 如果 $i+1 \leq k \leq j-1$，则 $A[k] > x$。
- 如果 $k = r$，则 $A[k] = x$。

下面便是证明这个循环不变式:

- 初始化：循环开始前，有 $i = p-1$ 和 $j = p$。不存在k使得 $p \leq k \leq i$ 或 $i+1 \leq k \leq j-1$，所以1)和2)成立。程序第一行代码里的 $x = A[r]$ 使得条件3)成立。
- 保持：根据第4行代码的比较结果，有两种情况:
  (1) $A[j] > x$ 时，仅做一个j增加1的操作，所以条件1)和3)不受影响。j增加后 $A[j-1] > x$，又因为 $A[1...j-2]$ 在迭代前同样都大于x，所以条件2)成立;
  (2) $A[j] \leq x$ 时，i增加1，因为 $A[p...i-1]$ 都小于等于x，而 $A[i]$ 也小于等于x，所以 $A[p...i]$ 都小于等于x，条件1)成立。j也增加1，与情况1一样，条件2)和条件3)也都成立。
  终止：循环结束时j=r，根据条件1) $A[p...i]$ 都会小于等于x，而 $A[i+1...r-1]$ 都大于x。

快速排序Python代码实现:

In [2]:

```
data = [45,3,2,6,3,78,5,44,22,65,46]
def quickSort(data, start, end):
    i = start
    j = end
    #i与j重合时，一次排序结束
    if i >= j:
        return
    #设置最左边的数为基准值
    flag = data[start]
    while i < j:
        while i<j and data[j] >= flag:
            j -= 1
        #找到右边第一个小于基准的数，赋值给左边i。此时左边i被记录在flag中
        data[i] = data[j]
        while i<j and data[i] <= flag:
            i += 1
        #找到左边第一个大于基准的数，赋值给右边的j。右边的j的值和上面左边的i的
        data[j] = data[i]
    #由于循环以i结尾，循环完毕后把flag值放到i所在位置。
    data[i] = flag
    #除去i之外两段递归
    quickSort(data, start, i-1)
    quickSort(data, i+1, end)

quickSort(data,0, len(data)-1)
print(data)
```

```
[2, 3, 3, 5, 6, 22, 44, 45, 46, 65, 78]
```

快速排序C语言实现：

```
//对区间 [left, right] 进行划分
int Partition(int A[ ],
                        int left,int right)
   int  temp = A[left];
   while (left < right) {
       while(left<right &&
                 A[right]>temp)  right--;
       A[left] = A[right];
       while(left<right &&
                 A[left]<=temp)  left++;
       A[right] = A[left];
    }
    A[left] = temp;
    return  left;
}
//对区间 [left, right] 进行划分
void  quickSort (int A[ ],
                int left, int right)  {
 if (left < right)  {
//将[left, right]按A[left]一分为二
int pos=Partition(A, left, right);
//对左子区间递归进行快速排序
quickSort(A, left, pos-1);
//对右子区间递归进行快速排序
quickSort(A, pos+1, right);  }}
```

三. 快速排序的性能

快速排序算法的性能与数组如何被切分有关。在最坏情况下，n个元素的数组被切分为n-1个元素和0个元素的两部分，PARTITION因为要经历n-1次迭代，所以运行代价为 $\Theta(n)$。即：

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

其中，元素数为0时，QUICKSORT直接返回，所以运行代价为 $\Theta(1)$。利用代换法，可以得到最坏情况下快速排序算法的运行时间为 $\Theta(n^2)$。

在最好情况下，每次PARTITION都得到两个元素数分别为floor(n/2)和ceiling(n/2)-1的子数组，这种情况下：

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

所以最佳情况下快速排序算法的运行时间为 $\Theta(n\lg(n))$。

例 1 考虑平均情况，假设每次都以9:1的例划分数组，则得到：

$$T(n) \le T(9n/10) + T(n/10) + \Theta(n)$$

四. 快速排序的随机化版本

因为在平均情况下，快速排序的运行时间为 $O(n\lg(n))$，还是比较快的，所以使所有的输入都能获得较好的平均情况性能，可以使快速排序随机化，即随机选择作为分割点的元素，而不总是数组尾部的元素：

随机快速排序伪代码:

```
RANDOMIZED_PARTITION(A, p, r)
1 i = RANDOM(p, r)
2 swap(A[i], A[r])
3 return PARTITION(A, p, r)

RANDOMIZED_QUICKSORT(A, p, r)
1 if p <</SPAN> r
2 q = RANDOMIZED_PARTITION(A, p, r);
3 RANDOMIZED_QUICKSORT(A, q-1, p);
4 RANDOMIZED_QUICKSORT(A, q+1, r);
```

随即快速排序C语言代码:

```
int ranPartition(int A[ ],int left, int right)  {
Int p=(round(1.0*rand( )
/RAND_MAX*(right-left)+left);
swap(A[p], A[left]);
int temp = A[left];
while(left < right) {
    while(left <right && A[right]>temp) right---;
A[left]= A[right];
    while(left<right && A[left]<= temp) left++;
A[right] = A[left];
}
A[left] = temp;
return  left;  }
```

随机快速排序的Python代码：

In [ ]:

```python
#随机快速排序
import random
def random_quicksort(a,left,right):
    if(left<right):
        mid = random_partition(a,left,right)
        random_quicksort(a,left,mid-1)
        random_quicksort(a,mid+1,right)


def random_partition(a,left,right):
    t = random.randint(left,right)     #生成[left,right]之间的一个随机数
    a[t],a[right] = a[right],a[t]
    x = a[right]
    i = left-1                         #初始i指向一个空，保证0到i都小于等于 x
    for j in range(left,right):        #j用来寻找比x小的，找到就和i+1交换，保
        if(a[j]<=x):
            i = i+1
            a[i],a[j] = a[j],a[i]
    a[i+1],a[right] = a[right],a[i+1]  #0到i 都小于等于x ,所以x的最终位置就是
    return i+1

while(True):
    try:
        s = input("输入待排序数组：\n")             #待排数组
        l =s.split()
        a = [int(t) for t in l]
        random_quicksort(a,0,len(a)-1)
        print ("排序后：")
        for item in a:
            print(item,end=' ')
        print("\n")
    except:
        break
```

期望运行时间：

快速排序主要在递归地调用PARTITION过程。我们先看下PARTITION调用的总次数，因为每次划分时，都会选出一个主元元素（作为基准、将数组分隔成两部分的那个元素），它将不会参与后续的QUICKSORT和PATITION调用里，所以PATITION最多只能执行n次。在 PARTITON过程里，有一段循环代码（第3至第8行，将各元素与主元元素比较，并根据需要将元素调换）。我们把这

段循环代码单独提出来考虑，这样在每次PATITIOIN调用里，除循环代码外的其它代码的运行时间为$O(1)$，所以在整个排序过程中，除循环代码外的其它代码的总运行时间为$O(n*1)=O(n)$。

接下来分析整个排序过程中，上述循环代码的总运行时间（注意：不是某次PATITION调用里的循环代码的运行时间）。可以看到在循环代码里，数组中的各个元素之间进行比较。设总的比较次数为X，因为一次比较操作本身消耗常量时间，所以比较的总时间为 $O(X)$。如此整个排序过程的运行时间为 $O(n+X)$。

为了得到算法总运行时间，我们需要确定总的比较次数X的值。为了便于分析，我们将数组A中的元素重新命名为 z_1 ，z_2 ，z_3 ，. . . ，z_n 。其中 z_i 是数组A中的第i小的元素。此外，我们还定义 $Z_{ij}=\{z_i,z_(i+1),\ldots,z_j\}$ 为 z_i 和 z_j 之间（包含这两个元素）的元素集合。

我们用指示器随机变量

$$X_{ij}=I\{z_i\text{与}z_j\text{进行比较}\}$$

这样总的比较次数：

$$X=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}X_{ij}$$

求期望得：

$$E[X]=E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}X_{ij}\right]=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}EX_{ij}=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}Pr\{z_i\text{与}z_j\text{进行比较}\}$$

注意两个元素一旦被划分到两个不同的区域后，则不可能相互进行比较。它们能进行比较的条件只能为：z_i 和 z_j 在同一个区域，且 z_i 或 z_j 被选为主元元素，这样：

$$\begin{aligned}Pr\{z_i\text{与}z_j\text{进行比较}\}&=Pr\{z_i\text{或}z_j\text{是从}Z_{ij}\text{中选出的主元元素}\}\\&=Pr\{z_i\text{是从}Z_{ij}\text{中选出的主元元素}\}\\&\quad+Pr\{z_j\text{是从}Z_{ij}\text{中选出的主元元素}\}\\&=\frac{1}{j-i+1}+\frac{1}{j-i+1}=\frac{2}{j-i+1}\end{aligned}$$

得到 x_{ij} 的概率后，就可以得到总的比较次数：

$$E[X]=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}Pr\{z_i\text{与}z_j\text{进行比较}\}=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1}$$

设变量 $k=j-1$，则上式变为：

$$E[X] = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} 2/(k+1) < \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} 2/k = \sum_{i=1}^{n-1} O(\lg(n)) = O(n \lg(n))$$

我们得出结论：使用RANDOMIZED-PARTITION，在输入元素互异的情况下，快速排序算法的期望运行时间为 $O(n\lg(n))$。

引用及参考：

[1]《Python数据结构与算法分析》布拉德利.米勒等著，人民邮电出版社，2019年9月.
[2] https://blog.csdn.net/weixin_42018258/article/details/80670067 (https://blog.csdn.net/weixin_42018258/article/details/80670067)
[3] https://www.cnblogs.com/raincute/p/8759117.html (https://www.cnblogs.com/raincute/p/8759117.html)
[4] http://blog.sina.com.cn/s/blog_73428e9a01017f9x.html (http://blog.sina.com.cn/s/blog_73428e9a01017f9x.html)

课后练习

1. 写出快速排序的完整Python代码或 C语言代码。

2. 写出利用快速排序算法对数组 A={13,19,9,5,12,8,7,4,21,2,6,11}的操作流程，并用程序实现。

3. 写出随机化的快速排序算法的完整Python代码或 C语言代码，并对时间复杂度进行分析。

讨论、思考题、作业：

参考资料（含参考书、文献等）：算法导论. Thomas H. Cormen等，机械工业出版社，2017.

授课类型（请打√）：理论课□ 讨论课□ 实验课□ 练习课□ 其他□

教学过程设计（请打√）：复习□ 授新课□ 安排讨论□ 布置作业□

教学方式（请打√）：讲授□ 讨论□ 示教□ 指导□ 其他□

教学资源（请打√）：多媒体□ 模型□ 实物□ 挂图□ 音像□ 其他□

填表说明：1、每项页面大小可自行添减；2、教学内容与讨论、思考题、作业部分可合二为一。