

《算法设计与分析》课程作业一

尹达恒

2020/10/14

1 快速排序

1.1 问题描述

Description

给定一维 `int` 型数组 `a[0,1,...,n-1]`，使用快速排序方法，对其进行从小到大排序，请输出递归过程中自顶自下第二层的划分结果，其中最顶层为第一层，即最终的排序结果层。

划分时请用第 1 个元素作为划分基准，并使用课件上的方法进行一次扫描实现划分。

Input

输入第 1 行有一个 `int` 型正整数 `m(m<100)`，表示有 `m` 行输入。每行输入的第一个数为 `int` 型正整数 `n(8<n<1000)`，后面接着输入 `n` 个 `int` 型整数。

Output

对每组数据, 输出自顶自下第二层的划分结果。

Sample Input

```
2
11 6 3 7 8 5 1 4 2 4 9 10
12 6 3 7 8 4 5 1 11 2 4 9 10
```

Sample Output

```
2 3 1 4 4 5 6 7 8 9 10
2 3 1 4 4 5 6 10 8 7 9 11
```

1.2 算法思路

1. 划分：以数组中的第一个元素为基准值，从数组中的第二个元素开始扫描，比基准值小的放右边，比基准值大的放左边。
2. 处理：
 - 初始时基准坐标 `p=` 数组开头位置；

- i 从数组第二个元素开始遍历, 若 i 位置的值大于基准值, 则与 p 位置后一位的值交换, 并令 p 自增;
 - 最后令数组开头的值与 p 位置的值交换。
3. 递归: 使用同样的方法递归地处理左边和右边的子数组。
 4. 输出第二层结果: 设置一个层标记 $l=1$, 每次递归都将此值加 1 后作为参数传入, 递归函数内若检测到 $l>2$ 则退出并输出结果。

1.3 算法伪代码

见算法 1。

Algorithm 1: 快速排序算法伪代码

```

1 Function QuickSort( $S, n$ ) begin
    Input: 未排序的数组  $S$ 、数组长度  $n$ 、递归层数  $l$ 
    Output: 排好序的数组  $S$ 
2   if  $(l \leq 2) \wedge (n > 1)$  then
3        $p = 0$ ;
4       for  $i \in \{1, 2, 3, \dots, n\}$  do
5           if  $S_0 > S_i$  then
6               交换  $S_{p+1}$  和  $S_i$ ;
7                $p$  自增 1;
8           end
9       end
10      交换  $S_0$  和  $S_p$ ;
11      QuickSort( $S, p, l + 1$ );
12      QuickSort( $S' = \{S_i | i \in [p + 1, n - 1]\}, n - (p + 1), l + 1$ );
13  end
14  return
15 end

```

2 找第 2 大数

2.1 问题描述

Description

给定一维 `int` 型数组，请找到第 2 大的数。

Input

输入第 1 行有一个 `int` 型正整数 $m(m < 100)$ ，表示有 m 行输入。

每行输入的第一个数为 `int` 型正整数 $n(0 < n < 1000)$ ，后面接着输入 n 个 `int` 型整数。

Output

输出 m 行，每行为找第 2 大数。

Sample Input

```
2
8 3 8 4 1 6 7 3 2
9 2 4 5 9 8 7 6 4 3
```

Sample Output

```
2
3
```

2.2 算法思路

由于题干并没有要求找任意第 k 大的数，因此本题可以直接使用线性查找，时间复杂度为 $O(n)$ 。

2.3 算法伪代码

见算法 2。

Algorithm 2: 找第二大数算法伪代码

```

1 Function FindNumber(S, n) begin
    Input: 数组 S、数组长度 n
    Output: 数组 S 中第二大的数
2    $R = \begin{cases} \{S_0, S_1\} (S_0 < S_1) \\ \{S_1, S_0\} (S_0 > S_1) \end{cases};$ 
3   for  $i \in \{2, 3, \dots, n\}$  do
4       if  $S_i < R_0$  then
5            $R_1 = R_0;$ 
6            $R_0 = S_i;$ 
7       end
8       else
9            $R_1 = S_i;$ 
10      end
11  end
12  return  $R_1;$ 
13 end

```

3 寻找两个正序数组的中位数

3.1 问题描述

Description

给定两个大小为 *m* 和 *n* 的正序（从小到大）数组 *nums1* 和 *nums2*。

请你找出并返回这两个正序数组的中位数。

进阶：你能设计一个时间复杂度为 $O(\log(m + n))$ 的算法解决此问题吗？

Input

第一行输入 *nums* 表示有 *nums* 组测试

每组测试输入 *n* 和 *m*，分别表示数组 *nums1* 和 *nums2* 的长度

然后输入正序数组 *nums1*

接着输入正序数组 *nums2*

Output

对每组测试数据输出两个正序数组的中位数

Sample Input

```
2
2 1
1 3
2
2 2
1 2
3 4
```

Sample Output

```
2.00000
2.50000
```

3.2 算法思路

设要查找的两个正序数组分别为 \mathbb{M} 和 \mathbb{N} ，其组合成的正序数组为 \mathbb{U} 。
满足：

$$\begin{aligned} &(\forall M_i \in \mathbb{M})(M_{i+1} \geq M_i) \wedge (\forall N_i \in \mathbb{N})(N_{i+1} \geq N_i) \wedge \\ &\mathbb{U} = \{U_i | U_i \in \mathbb{M} \cup \mathbb{N}\} \wedge (\forall U_i \in \mathbb{U})(U_{i+1} \geq U_i) \wedge \\ &k < |\mathbb{M}| \wedge k < |\mathbb{N}| \end{aligned}$$

则有如下定理：

$$M_k < N_k \Rightarrow M_k < U_{2k}$$

证明：反证法。

$$\begin{aligned} &M_k < N_k \wedge M_k \geq U_{2k} \\ \Leftrightarrow &U_{2k} \leq M_k < N_k \\ \Rightarrow &U_{2k} \in \{M_i \in \mathbb{M} | 1 \leq i \leq k\} \cup \{N_i \in \mathbb{N} | 1 \leq i \leq k-1\} \\ \Rightarrow &|\{U_i \in \mathbb{U} | 1 \leq i \leq 2k\}| < |\{M_i \in \mathbb{M} | 1 \leq i \leq k\} \cup \{N_i \in \mathbb{N} | 1 \leq i \leq k-1\}| \\ \Rightarrow &2k \leq 2k-1 \Rightarrow false \end{aligned}$$

$\therefore M_k < N_k \rightarrow M_k < U_{2k}$ 为真命题。

查找两个正序数组 M 和 N 的中位数的算法可以等价为一个查找其组合成的正序数组 U 中第 $2k$ 大数 U_{2k} 的算法：

1. 若两数组长度和为偶数，则 $2k = (|M| + |N|)/2$ ，查找 U_{2k} 和 U_{2k+1} 取平均值；
2. 若两数组长度和为奇数，则 $2k = (|M| + |N| + 1)/2$ ，查找 U_{2k} 。

而根据前述定理 $M_k < N_k \Rightarrow M_k < U_{2k}$ ，若 $M_k < N_k$ ，则 U_{2k} 必然不在 $\{M_i \in M | 1 \leq i \leq k\}$ 中，因此可以直接舍弃该部分，在 $\{M_i \in M | k+1 \leq i \leq |M|\}$ 和 N 中查找第 k 大的元素即可。显然，对于 $M_k > N_k$ 时类似的情况也成立。此过程可以递归进行，直到 $M_k = N_k$ 或某一轮的 M 或 N 为空。算法的时间复杂度为 $O(\log|M| + \log|N|)$ 。

3.3 算法伪代码

见算法 3 和算法 4。

4 搜索二维矩阵

4.1 问题描述

Description

编写一个高效的算法来搜索 $m \times n$ 矩阵 `matrix` 中的一个目标值 `target`。该矩阵具有以下特性：

1. 每行的元素从左到右升序排列。
2. 每列的元素从上到下升序排列。

Input

第一行输入 `nums` 表示有 `nums` 组测试。

每组测试输入 `m`、`n`、`target`，分别表示矩阵的行列数以及目标值。

接下来输入 `m * n` 的二维矩阵。

Output

对每组测试数据输出能否在矩阵中找到 `target`。

若能找到，输出 `true`。

若找不到，输出 `false`。

Algorithm 3: 在两个正序数组中找第 k 大数

```

1 Function  $FindK(\mathbb{M}, \mathbb{N}, k)$  begin
   Input: 数组  $\mathbb{M}$  和  $\mathbb{N}$ 、整数  $k$ 
   Output: 数组  $\mathbb{M}$  和  $\mathbb{N}$  中第  $k$  大的数
2   if  $\mathbb{M} = \emptyset$  then return  $N_k$ ;
3   if  $\mathbb{N} = \emptyset$  then return  $M_k$ ;
4   if  $k = 1$  then return  $\min(M_1, N_1)$ ;
5    $d = \lfloor \frac{k}{2} \rfloor$ ;
6   if  $d > |\mathbb{M}|$  then
7     if  $M_{|\mathbb{M}|} \leq N_d$  then
8       return  $FindK(\emptyset, \mathbb{N}, k - |\mathbb{M}|)$ ;
9     else
10      return  $FindK(\mathbb{M}, \{N_i \in \mathbb{N} | i > d\}, k - d)$ ;
11    end
12  end
13  if  $d > |\mathbb{N}|$  then
14    if  $N_{|\mathbb{N}|} \leq M_d$  then
15      return  $FindK(\mathbb{M}, \emptyset, k - |\mathbb{N}|)$ ;
16    else
17      return  $FindK(\{M_i \in \mathbb{M} | i > d\}, \mathbb{N}, k - d)$ ;
18    end
19  end
20  if  $M_d = N_d$  then
21    if  $k$  为偶数 then return  $M_d$  (或  $N_d$ );
22    if  $k$  为奇数 then return  $\min(M_{d+1}, N_{d+1})$ ;
23  end
24  if  $M_d < N_d$  then
25    return  $FindK(\{M_i \in \mathbb{M} | i > d\}, \mathbb{N}, k - d)$ ;
26  else
27    return  $FindK(\mathbb{M}, \{N_i \in \mathbb{N} | i > d\}, k - d)$ ;
28  end
29 end

```

Algorithm 4: 找两个正序数组的中位数

```

1 Function FindMedium( $\mathbb{M}$ ,  $\mathbb{N}$ ) begin
    Input: 正序数组  $\mathbb{M}$ 、 $\mathbb{N}$ 
    Output: 数组  $\mathbb{M}$  和  $\mathbb{N}$  的中位数
2 if  $|\mathbb{M}| + |\mathbb{N}|$  为奇数 then
3     if  $\mathbb{M} = \emptyset$  then return  $N_{\frac{|\mathbb{M}|+|\mathbb{N}|+1}{2}}$ ;
4     if  $\mathbb{N} = \emptyset$  then return  $M_{\frac{|\mathbb{M}|+|\mathbb{N}|+1}{2}}$ ;
5     return FindK( $\mathbb{M}$ ,  $\mathbb{N}$ ,  $\frac{|\mathbb{M}|+|\mathbb{N}|+1}{2}$ );
6 else
7     if  $\mathbb{M} = \emptyset$  then return  $(N_{\frac{|\mathbb{M}|+|\mathbb{N}|}{2}} + N_{\frac{|\mathbb{M}|+|\mathbb{N}|}{2}+1})/2$ ;
8     if  $\mathbb{N} = \emptyset$  then return  $(M_{\frac{|\mathbb{M}|+|\mathbb{N}|}{2}} + M_{\frac{|\mathbb{M}|+|\mathbb{N}|}{2}+1})/2$ ;
9     return
         $(\text{FindK}(\mathbb{M}, \mathbb{N}, \frac{|\mathbb{M}|+|\mathbb{N}|}{2}) + \text{FindK}(\mathbb{M}, \mathbb{N}, \frac{|\mathbb{M}|+|\mathbb{N}|}{2} + 1))/2$ ;
10 end
11 end

```

Sample Input

```

1
5 5 5
1 4 7 11 15
2 5 8 12 19
3 6 9 16 22
10 13 14 17 24
18 21 23 26 30

```

Sample Output

```

true

```

提示

```

m <= 1000
n <= 1000

```

4.2 算法思路

设目标值为 x ，要查找的矩阵为 $A_{m \times n} = (a_{i,j})$ ，满足：

$$\begin{cases} a_{i,j} \leq a_{i,j+1} & (i \in [1, m], j \in [1, n-1]) \\ a_{i,j} \leq a_{i+1,j} & (i \in [1, m-1], j \in [1, n]) \end{cases}$$

显然有如下定理:

$$x > a_{0,n} \Rightarrow (\forall a_{1,j}, j \in [1, n])(x > a_{0,j})$$

$$x < a_{0,n} \Rightarrow (\forall a_{i,n}, i \in [1, m])(x > a_{i,n})$$

即当目标值大于矩阵右上角值时, 目标值必大于矩阵第一行的值; 目标值小于矩阵右上角值时, 目标值必小于矩阵第一列的值。因此, 我们可以从矩阵右上角开始搜索, 若目标值较大, 则向下搜索; 若目标值较小, 则向左搜索, 直到找到目标值或超出矩阵范围。此算法时间复杂度为 $O(m+n)$ 。

更进一步, 对于向下或向左的搜索过程, 我们可以使用二分查找, 找出当前行不大于目标值的最大元素 (目标值较小时) 或当前列不小于目标值的最小元素 (目标值较大时)。改进后算法的时间复杂度可以达到 $O(\log(m) + \log(n))$ 。

4.3 算法伪代码

见算法 5。

Algorithm 5: 搜索二维矩阵算法伪代码

```

1 Function FindNumber( $A_{m,n}, x$ ) begin
    Input: 待查矩阵  $A_{m,n} = (a_{i,j})$ 、目标值  $x$ 
    Output:  $A_{m,n}$  中是否存在  $a_{i,j} = x$ 
2   if  $m = 0 \vee n = 0$  then return false;
3   if  $x = a_{1,n}$  then return true;
4   if  $x > a_{1,n}$  then
5       二分查找满足条件的  $k$ :  $k > 1 \wedge a_{k,n} \geq x \wedge a_{k-1,n} < x$ ;
6       return FindNumber( $A'_{k,n} = (a_{i,j})(i \in [k, n], j \in [1, n]), x$ );
7   else
8       二分查找满足条件的  $k$ :  $k > 1 \wedge a_{1,k} \leq x \wedge a_{1,k+1} > x$ ;
9       return FindNumber( $A'_{m,k} = (a_{i,j})(i \in [1, n], j \in [1, k]), x$ );
10  end
11 end

```

5 平台截图

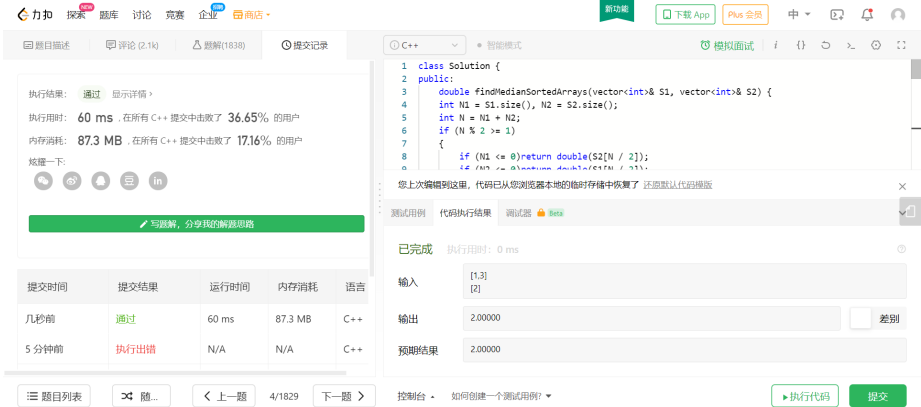


图 1: LeetCode 4 Accepted 截图

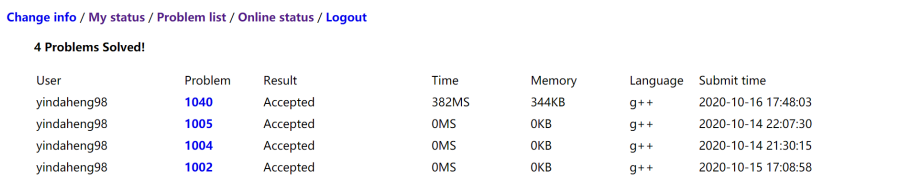


图 2: 47.99.179.148 Accepted 截图