算法设计与分析 Week 3

第三周作业-Solution

Lecturer: 杨启哲 Last modified: 2024 年 10 月 15 日

1. 考虑算法 $SlowMinmax$，它是将算法 $Minmax$ 的检验条件 **if** $high - low = 1$ 修改为 **if** $high = low$，并对此算法做一些相应改变而得出的。这样，在算法 $SlowMinmax$ 中，当输入数组的大小为 1 时，递归停止。计算由此算法找出数组 $A[1,\dots,n]$ 中的最大值和最小值所需要的比较次数，这里 n 是 2 的幂。并解释为什么此算法的比较次数大于算法 $Minmax$ 的比较次数。

   Hint: 在这种情形下，初始条件是 $C(1) = 0$

   解答. 注意到在题设条件下，算法 $SlowMinmax$ 的递归式为

   $$C(n) = \begin{cases} 0, & n = 1 \\ 2C(\frac{n}{2}) + 2, & n > 1 \end{cases}$$

   则考察 $n = 2^k$ 我们有:

   $$\begin{aligned} C(n) &= 2C(\frac{n}{2}) + 2 \\ &= 2(2C(\frac{n}{4}) + 2) + 2 \\ &\vdots \\ &= 2^k C(\frac{n}{2^k}) + 2(1 + 2 + \cdots + 2^{k-1}) \\ &= 2^k C(1) + 2(1 + 2 + \cdots + 2^k) \\ &= 2(2^k - 1) \\ &= 2n - 2 \end{aligned}$$

   其比较次数大于书上 MINMAX 的算法,这是因为在这个算法中,递归深度更深了一层,MINMAX 递归到 $n = 2$ 就停止了，而 $SlowMinmax$ 则递归到 $n = 1$ 才停止。 □

2. 对于某个整数 $g \geqslant 3$，用 g 来表示算法 $Select$ 中每组的规模，导出用 g 表示的算法的运行时间。当 $g = 3, 7, 9, 11$ 时，哪个选择可以保证算法在最坏情况下执行的次数依旧是 $\Theta(n)$?

   解答. 我们用 $2k + 1$ 来表示 g 这个奇数，则算法的递推式为:

   $$T(n) = T(\frac{1}{2k+1}n) + T(\frac{3k+1}{4k+2}n) + cn$$

   注意到 $g = 3$ 时 $k = 1$，从而 $\frac{1}{2k+1} + \frac{3k+1}{4k+2} = 1$，而 $g > 5$ 时有 $k > 1$，则 $\frac{1}{2k+1} + \frac{3k+1}{4k+2} < 1$，并且该值随 g 增加而递减，从而递归展开可知:

- 当 $g = 3$ 时 $T(n) = \Theta(n \log n)$。
- 当 $g = 7, 9, 11$ 时 $T(n) = \Theta(n)$。

□

> **Remark 0.1**
>
> 由上述分析可知，$g = 5$ 时算法有最快的运行速度。
>
> 一开始的答案版本混淆了 k 与 g，会有一些理解的困难，在新版已经完成了改正。

3. 有个木匠有一堆混合的 N 个螺母和 N 个螺栓。他希望能找到相应的螺母和螺栓配对。每个螺母只能配对一个螺栓，每个螺栓也只能配对一个螺母。通过将螺母和螺栓组合在一起，木匠可以看出哪个更大，但木匠不能直接比较两个螺母或两个螺栓。

   设计一种算法来解决这个问题，该算法使用 NlogN 次比较（以概率方式）。

   Hint: 好好利用 Quicksort 中的划分过程。

   解答. 该问题的想法是利用快速排序中的划分过程，即利用螺母将螺栓进行划分，然后再利用螺栓将螺母进行划分，如此反复，直到螺母和螺栓都被划分完毕。记螺母集合为 L，螺栓集合为 I，算法流程如下：

   (1) 随机选择一个螺母 x。
   (2) 利用螺母将螺栓分成三部分：与其匹配的 N_x，比其大的螺栓集合 L_1，比其小的螺栓集合 L_2。
   (3) 利用 N_x 将剩余的螺母分成两部分，比其大的螺母集合 I_1 和比其小的螺母集合 I_1。
   (4) 分别在 (I_1, L_1) 和 (I_2, L_2) 中递归的调用上述过程，直到螺母和螺栓都被划分完毕。

   粗略来看，每次划分的时间复杂性为 $O(n)$，而每次划分后，螺母和螺栓的数量都会减半，因此递归的深度为 $\log n$，从而算法的时间复杂性为 $O(n \log n)$.

   我们下面给出一个准确的分析。事实上，我们只需要关注螺母的排序时间，因为每轮螺栓的操作个数与螺母的操作个数相同。设 $T(n)$ 为对 n 个螺母的排序时间，特别的我们令 L 中的螺母以升序排列 $x_1, \ldots, x_n$。我们考察 x_i 与 x_j 被比较次数。

   - x_i 与 x_j 会被比较当且仅当 x_i 或者 x_j 被选为了划分的主元，从而其会被比较的概率 $p_{ij} = \frac{2}{j-i+1}$（这要求在 x_i 到 x_j 中的螺母中先选取这两个元素作为划分主元的概率）。

   现在定义随机变量 X_{ij} 为：

$$X_{ij} = \begin{cases} 1, & \text{如果 } x_i \text{ 与 } x_j \text{ 被比较} \\ 0, & \text{其他情况} \end{cases}$$

   则我们有算法对螺母的总比较次数为：

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

计算其期望可得:

$$\begin{aligned} E[T(n)] &= E[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}] \\ &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}] \\ &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1}\sum_{k=2}^{n-i+1} \frac{2}{k} \\ &\approx 2n\ln n \end{aligned}$$

从而算法的期望比较次数为 $O(n\log n)$. □

> **Remark 0.2**
>
> 随机算法的分析需要利用到很多概率论的知识。事实上，上述也只是一个不太严谨的推理。仅供大家参考。但希望大家能认识到的是，随即挑选主元进行划分的快速排序算法的期望时间复杂性已经为 $O(n\log n)$.

4. 求出唯一的 4 次多项式 p，满足 $p(1)=2, p(2)=1, p(3)=0, p(4)=4, p(5)=0$。

解答. 解如下方程组:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 5 & 25 & 125 & 625 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 4 \\ 0 \end{bmatrix}$$

可得该多项式为:

$$p = -\frac{3}{4}x^4 + \frac{25}{3}x^3 - \frac{125}{4}x^2 + \frac{137}{3}x - 20$$

□

> **Remark 0.3**
>
> 也可以利用拉格朗日插值法直接算出:
>
> $$\begin{aligned} p =& 2\cdot\frac{(x-2)(x-3)(x-4)(x-5)}{(1-2)(1-3)(1-4)(1-5)} + 1\cdot\frac{(x-1)(x-3)(x-4)(x-5)}{(2-1)(2-3)(2-4)(2-5)} \\ &+ 4\cdot\frac{(x-1)(x-2)(x-3)(x-5)}{(4-1)(4-2)(4-3)(4-5)} \end{aligned}$$
>
> 实际计算略显复杂，可借助其他工具。

5. 给出一个有序数组 $A[1,\ldots,n]$，其元素各不相同，请给出一个 $O(\log n)$ 的算法，找出一个下标 i，使得 $A[i]=i$。如果不存在这样的 i，则返回 -1。

> Remark 0.4
>
> 需要补充说明下，这个数组中的元素都应当是**整数**。

解答. 只需要注意到，当数组是有序不同的整数的时候，$B[i]=A[i]-i$ 也是有序的。(整数性质：若 $x<y$，则 $x\leqslant y-1$)。从而问题可以转化为在 B 中找到一个 0，这可以通过二分查找来实现。伪代码如下：

算法: 判断是否存在与值相同的下标

输入: 有序数组 $A[1,\ldots,n]$
输出: i 使得 $A[i]=i$，不存在则返回 -1

```
low ← 1, high ← n
while low ⩽ high do
    mid ← ⌊(low + high)/2⌋
    if A[mid] − mid = 0 then
        return mid
    else if A[mid] − mid > 0 then
        high ← mid − 1
    else
        low ← mid + 1
    end if
end while
return −1
```

□

6. 求解下列递推式，并针对每个递推式给出一个 Θ 界限：

(1) $T(n)=5T(\frac{n}{3})+n$
(2) $T(n)=8T(\frac{n}{2})+n^3$
(3) $T(n)=2T(\frac{n}{3})+1$

解答. 由主定理可得

(1) $T(n)=\Theta(n^{\log_3 5})$.
(2) $T(n)=\Theta(n^3\log n)$.
(3) $T(n)=\Theta(n^{\log_3 2})$.

□

7. 给定一个数组 $A[1,\ldots,n]$，请设计一个算法，找出 A 中和最大的非空连续子数组。例如，如果 $A=[-2,1,-3,4,-1,2,1,-5,4]$，则和最大的非空连续子数组为 $[4,-1,2,1]$，其和为 6，并分析你设计的算法的时间复杂性。

解答. 显然暴力搜索需要 $O(n^2)$ 的时间。我们利用分治思想给出一个 $O(n\log n)$ 的算法。

假设要计算数组 $A[low,high]$ 的最大连续子数组，我们将其分成两部分 $A[low,mid]$ 和 $A[mid+1,high]$，则其最大子数组一定是下列情况之一：

(1) $A[low,mid]$ 中的最大连续子数组。

(2) $A[mid,high]$ 中的最大连续子数组。

(3) 一个跨越 mid 的最大连续子数组。

关于最后一个，由于其必须经过 mid 位置，我们可以从 mid 开始向两边遍历，找到最大的连续子数组，这只需要 $O(n)$ 的时间。递归算法流程如下：

算法: 找出数组中的最大连续非空子数组

输入: 数组 $A[1,\ldots,n]$ 和整数 x
输出: 数组中的最大连续非空子数组之和

```
return Count(A, 1, n)
left ← 1, right ← n
过程: Count(A, low, high)
if low = high then
    return A[low]
end if
mid ← ⌊(low + high)/2⌋
leftMax ← Count(A, low, mid)
rightMax ← Count(A, mid + 1, high)
crossMax ← FindCrossMax(A, low, mid, high)
return max(leftMax, rightMax, crossMax)
过程: FindCrossMax(A, low, mid, high)
leftSum ← −∞
sum ← 0
for i = mid to low do
    sum ← sum + A[i]
    leftSum ← max(leftSum, sum)
end for
rightSum ← −∞
sum ← 0
for i = mid + 1 to high do
    sum ← sum + A[i]
```

```
    rightSum ← max(rightSum, sum)
end for
return leftSum + rightSum
```

□

Remark 0.5

事实上，利用**动态规划**的思想我们可以进一步将时间复杂性降低到 $O(n)$，这里的核心关键在于如果一段连续子数组的和 < 0，则在计算包含之后数组元素的子数组时我们可以直接丢弃。具体来说，令 $S[i]$ 表示以 $A[i]$ 结尾的最大连续子数组和，则有：

$$S[i] = \max\{S[i-1] + A[i], A[i]\}$$

从而最终答案为 $\max_{i \in n} S[i]$。

8. 3−SAT 问题的定义如下：其实例是一个合取范式，其中每个子句都包含三个变量的析取。例如，$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee x_5) \wedge (x_3 \vee \neg x_5 \vee x_6)$，其中 $x_1, x_2, x_3, x_4, x_5, x_6$ 是布尔变量。3−SAT 问题是判断是否存在一种赋值方式，使得合取范式为真。

   在课程的后半段，我们会知道该问题是一个 NP 完全问题。现在，考虑一个其受限的版本，假设将其变量从 $1, \ldots, n$ 编号，公式中的每个子句包含变量的编号差异在 ± 8 以内。请给出一个线性时间的算法，判断这样的公式是否是可满足的。

   解答. 注意到变量具有局部性的性质，我们只需要保留 8 个变量的所有可能取值，从而得到一个线性时间的算法。我们先介绍一些记号，令输入的实例为 $C_1 \wedge C_2 \wedge \cdots \wedge C_m$，变量记为 $x_1, \ldots, x_n$，定义如下集合：

   - $V_1 = \{C_i \mid \exists\, j \in \{1, 2, \ldots, 8\}\ x_j \in C_i \text{ or } \neg x_j \in C_i\}$
   - $V_k = \{C_i \mid \exists\, j \in \{8k-7, 8k-6, \ldots, 8k\}\ x_j \in C_i \text{ or } \neg x_j \in C_i\} \setminus V_{k-1}$

   注意到对每个集合 V_k 其满足出现的变量至多为 $x_{8k-15}, \ldots, x_{8k+8}$，从而可以构造算法如下：

   (1) 枚举变量 $x_1, \ldots, x_{16}$ 的所有可能赋值，并记录下所有使得 V_1 中成真的对 $x_9, \ldots, x_{16}$ 的赋值，令该集合为 A。
   (2) 若 A 为空，则返回 False。
   (3) 对于 $j = 2, 3, \ldots k$，重复如下步骤：
      (i) 枚举集合 A 当前的每种赋值情况，对每个赋值情况，枚举 $x_{8j-7}, \ldots, x_{8j+8}$ 的所有可能赋值。
      (ii) 记录下所有使得 V_j 中成真的对 $x_{8j-7}, \ldots, x_{8j}$ 的赋值，枚举结束后将 A 更新成该集合。
   (4) 若 A 不为空，则返回 True。

   注意到 16 个变量的所有取值情况为 $2^{16} = 65536$ 是常数，集合 A 的大小至多为 $2^8 = 256$ 也是常数，从而上述算法的时间复杂性为 $O(m)$，即是线性时间的。 □