算法作业

丁元杰 17231164

2019 年 9 月 30 日

1

1.1

解 可知：

$$T(n) = \left\lceil \frac{n}{2} \right\rceil$$

由归纳法，$T(1) = 1$，$T(2) = 1$，满足边界条件。
如果

$$T(n-2) = \left\lceil \frac{n-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil - 1$$

，那么

$$T(n) = \left\lceil \frac{n}{2} \right\rceil - 1 + 1 = \left\lceil \frac{n}{2} \right\rceil$$

综上所述，

$$T(n) = \left\lceil \frac{n}{2} \right\rceil = O(n)$$

1.2

解 可知：

$$T(n) = \log_2 n + 1$$

由归纳法，

$$T(1) = 1$$

边界满足条件，如果

$$T(n/2) = \log_2 (n/2) + 1 = \log_2 n - \log_2 2 + 1 = \log_2 n$$

那么

$$T(n) = T(n/2) + 1 = \log_2 n + 1$$

由此得到：

$$T(n) = \log_2 n + 1 = O(\log n)$$

1.3

解 由主定理：

$$T(n) = O(n^{\log_2 2} \lg n) = O(n \log n)$$

1.4

解 由主定理：

$$T(n) = O(n^{\log_2 2}) = O(n)$$

1.5

解 由主定理：

$$T(n) = O(n^{\log_2 4}) = O(n^2)$$

1.6

解 由主定理：

$$T(n) = O(n^2)$$

1.7

解 由观察可以发现，设$n = 2^m$，则

$$\begin{aligned}
T(n) &= T(2^m) \\
&= \sum_{i=0}^{m} \log 2^i \\
&= \sum_{i=0}^{m} i \log 2 \\
&= \log 2 \sum_{i=0}^{m} i \\
&= \log 2 \cdot O(m^2) = O(m^2) \\
&= O(\log n^2)
\end{aligned}$$

2

算法思路

先考虑另一个问题KTH-ELEMENT(A, k)，表示求解数组A中第k小的元素，并用这个元素对原序列进行划分。设A排序后的数组为A'，那么$A[1..n]$中的第k小元素就被定义为$A'[k]$。

对于 KTH-ELEMENT 问题，需要使用到快速排序中的函数 PARTITION。在 KTH-ELEMENT$(A[1..n], k)$问题中，随机从$A[1..n]$中选取一个元素$A[p]$作为 pivot，设经过PARTITION之后的数组为A'，pivot 在A'中的下标为p'（即$A[p]$是A中的第p'小），那么

1. 如果$p' \geq k$，原问题化为子问题KTH-ELEMENT$(A'[1..p'], k)$
2. 如果$p' < k$，原问题化为子问题KTH-ELEMENT$(A'[p'+1..n], k - p')$

设原问题为ROUGHLY-SORT(A, k)，表示将A数组化为k重有序数组。那么可以通过不断地对原数组进行 KTH-ELEMENT$(A, n/2)$，直到长度小于n/k。或者形式地讲，解决 ROUGHLY-SORT$(A[1..n], k)$ 问题，

1. 如果$k = 1$，那么$A[1..n]$即是所求
2. 否则先对序列进行 KTH-ELEMENT$(A[1..n], n/2)$，然后分别进行 ROUGHLY-SORT$(A[1..n/2], k/2)$，以及 ROUGHLY-SORT$(A[n/2+1..n], k/2)$

伪代码

算法 1 将序列排成k重有序

输入: A数组，k有序数组的重数

输出: 排序后的数组A

```
function KthElement(A, k)
    randomly choose a number p ∈ [1, n]
    call Partition(A, p), and set the new location of A[p] to p'
    if p' ≥ k then
        KthElement(A[1..p'], k)
    else
        KthElement(A[p' + 1..n], k − p')
    end if
end function
function RoughlySort(A[1..n], k)
    if k = 1 then
        return
    else
        call KthElement(A[1..n], n/2)
        call RoughlySort(A[1..n/2], k/2)
        call RoughlySort(A[n/2 + 1..1], k/2)
    end if
end function
```

复杂度分析

首先分析KTH-ELEMENT$(A[1..n], k)$的复杂度$f(n)$。声明$f(n)$的期望复杂度为：

$$f(n) = O(n)$$

现用归纳法证明当n足够大时，$f(n) < 6n$。显然地有

$$f(1) = 1 < 6$$

假设对$\forall i < n$，都有

$$f(i) < 6i$$

设选取到用于 PARTITION 的 pivot 元素是$A[p]$，那么$f(n)$由这两部分组成：

1. n的时间用于 PARTITION 操作，此时间与p的选取无关

2. 以$\frac{1}{n}$的概率选到某个元素p作 pivot 之后，需要花费 $f(p)$或$f(n-p)$的时间进行后面的 KTH-ELEMENT 操作。更加明确地讲，是$\max\{p, n-p+1\}$

所以可以看出

$$
\begin{aligned}
f(n) &= n + \frac{1}{n}\sum_{i=1}^{n} f\left(\max\{n-i+1, i\}\right) \\
&= n + \frac{2}{n}\sum_{i=1}^{n/2} f\left(\frac{n}{2}+i-1\right) \\
&< n + \frac{2}{n} \times 6 \times \sum_{i=1}^{n/2} \frac{n}{2}+i-1 \\
&= n + \frac{12}{n} \times \left(\frac{n^2}{4} - \frac{n}{2} + \frac{n^2+2}{8}\right) \\
&= n + 6 \times \frac{3n-4+\frac{2}{n}}{4} \\
&< \frac{11}{2}n - 6 + \frac{3}{n} \\
&= 5.5n + o(n) \\
&< 6n = O(n)
\end{aligned}
$$

所以可以得出结论

$$f(n) = O(n)$$

其次分析ROUGHLY-SORT的时间复杂度$T(n,k)$，可以发现递归式：

$$
T(n,k) = \begin{cases} 1 & ,k=1 \\ 2T(n/2, k/2) + f(n) & ,\text{else} \end{cases}
$$

其中$f(n) = O(n)$。所以，可以得出，$T(n,k) = O(n \log k)$。

3

算法思路

考虑$A[1..n]$序列的差分序列$B[1..n-1]$，满足条件$B[i] = A[i+1] - A[i]$。
如果$A[i^*]$是A序列中的极大值，则必有$B[i^*-1] > 0$且$B[i^*] < 0$。
此时可以考虑在B序列上进行二分查找，设初始$l_0 = 1, r_0 = n-1$，可知$B[l_0] < 0$，而$B[r_0] > 0$。
接下来进行迭代，在第k次迭代中，维护循环不变性$l_k < r_k$，以及$B[l_k]B[r_k] < 0$：

1. 如果$l_k + 1 = r_k$，那么算法结束，$A[r_k]$就是所求的局部最大值。
2. 记$m_k = \left\lfloor \frac{l_k + r_k}{2} \right\rfloor$，显然有$l_k < m_k < r_k$，且以下两个条件之一成立：
   (a) 如果$B[l_k]B[m_k] < 0$，则$l_{k+1} = l_k$，$r_{k+1} = m_k$
   (b) 如果$B[r_k]B[m_k] < 0$，则$l_{k+1} = m_k$，$r_{k+1} = r_k$

算法 2 求局部最大值

输入: A数组，n数组长度

输出: 局部最大值在A中的下标

```
function LocalMaximum(A, n)
    l ← 1, r ← n − 1
    while l < r do
        if l + 1 = r then
            return r
        else
            m ← ⌊(l+r)/2⌋
            b_l ← A[l + 1] − A[l]
            b_m ← A[m + 1] − A[m]
            if b_l ∗ b_m < 0 then
                r ← m
            else
                l ← m
            end if
        end if
    end while
end function
```

伪代码

复杂度分析

设算法时间复杂度为$T(n)$，其中n为A数组长度。那么很明显的有：

$$T(n) = \begin{cases} 1 & , n = 1 \\ T(n/2) + 1 & , \text{else} \end{cases}$$

显然，$T(n) = O(\log n)$

4

算法思路

递归比较字符串A和B的问题 EQUIV(A, B)，将A分为等长的A_1，A_2，将B分为B_1，B_2。如果 EQUIV(A_1, B_1) & EQUIV(A_2, B_2) 或者 EQUIV(A_1, B_2) & EQUIV(A_2, B_1) 成立，那么原问题成立，否则原问题不成立。

算法 3 判断字符串等价

输入: A数组，B数组

输出: A和B是否等价

```
1: function Equiv(A[1..n], B[1..n])
2:     if n is not multiple of 2 then
3:         return A = B
4:     end if
5:     if Equiv(A[1..n/2], B[1..n/2]) and Equiv(A[n/2+1..n], B[n/2+1..n]) or Equiv(A[1..n/2], B[n/2+1..1])
   and Equiv(A[n/2+1..n], B[1..n/2]) then
6:         return True
7:     else
8:         return False
9:     end if
10: end function
```

伪代码

复杂度分析

设$n = 2^k q$，其中$q \perp 2$。那么原问题 EQUIV$(A[1..n], B[1..n])$ 的时间复杂度$T(n)$存在递归式：

$$T(n) = \begin{cases} n & , n \perp 2 \\ 3T(n/2) + 1 & , \text{else} \end{cases}$$

值得注意的是，由于短路求值的特性，递归方法中的第4行最多执行三个，因此在时间复杂度中的系数是3。

由主方法，可以知道

$$T(n) = O(3^k q)$$

5

算法思路

由于给出的序列结构具有高度的递归性，且构成一个二叉树。从S串的选取的任意子串$S[l..r]$，等价于从该二叉树中选取中序遍历排序在$[l..r]$范围内的全部结点。很明显，这些结点中深度最浅的结点是唯一的，对应到子串中，则表示字典序最大的字符是存在且唯一的，不妨将这个字符称作l到r的最大字符，记为$p(l, r)$，例如，$S[2..5] = baca$，那么$p(2, 5) = c$。

其次，我们还可以用下标i推出$S[i]$这个字符的内容。如果定义lowbit(i)为i的最大的2的整数次幂的因子，即i的二进制表示中的最低为的1及其后面的0组成的二进制数。那么

$$S[i] = \Sigma[\text{lowbit}(i) + 1]$$

（注意，这里用$\Sigma[1]$指代a，用$\Sigma[2]$指代b，以此类推）。同样的，定义highbit(i)为小于等于i的最大的2的整次幂，即i的二进制表示中的最高位的1。那么$p(l, r)$即是$l \le x \le r$的所有x中，lowbit值最大的那一个，即：

$$p(l, r) = \underset{l \le x \le r}{\arg\max} \{\text{lowbit}(x)\}$$

可以观察出，

$$p(l,r)=\Sigma[\text{highbit}((l-1)\oplus r)]$$

其中$\oplus$表示按位异或。

然后我们讨论S串的自相似性。对于任意的x，只要$\text{lowbit}(x)>p(l,r)$，就有

$$S[l..r]=S[l+x..r+x]$$

再次，我们尝试解决的最大公共子串问题，即是两个字符串的最大重合问题。在此例中，还可以转化为两个来源于同一棵二叉树的两个子部分的最大重叠问题。

设我们的问题为$\text{LCS}(l_1,r_1,l_2,r_2)$，表示求解$S[l_1..r_1]$和$S[l_2..r_2]$的最长公共子串。可以分情况讨论：

- 如果$p(l_1,r_1)=p(l_2,r_2)$，那么可以将两个子区间平移，使得其最大字符重叠。平移之后两个子区间的最大重叠长度就是所求的最长公共子串的长度。
- 如果$p(l_1,r_1)<p(l_2,r_2)$，那么仍然可以平移$[l_1..r_2]$，至$[l_1'..r_1']$中的最大字符与$[l_2,r_2]$中的最大字符两侧的第一个$p(l_1,r_1)$字符分别重合时，计算两次区间覆盖。较大者就是答案。

伪代码

算法 4 求最长公共子串

输入: l_1,r_1,l_2,r_2，表示所求的两个区间范围

输出: 最长公共子串的长度

function $\text{LCS}(l_1,r_1,l_2,r_2)$
  $p_1\leftarrow\text{highbit}((l_1-1)\oplus r_1)$
  $p_2\leftarrow\text{highbit}((l_2-1)\oplus r_2)$
  $l_1\leftarrow l_1 \mod 2p_1,\ r_1\leftarrow r_1 \mod 2p_1$
  $l_2\leftarrow l_2 \mod 2p_2,\ r_2\leftarrow r_2 \mod 2p_2$
  if $\Sigma[p_1]>\Sigma[p_2]$ **then**
    swap $[l_1,r_1]$ and $[l_2,r_2]$
  end if
  if $p_1=p_2$ **then**
    return $\text{Overlap}(l_1,r_1,l_2,r_2)$
  else
    $\delta_1\leftarrow p_2-2p_1$
    $\delta_2\leftarrow p_2+2p_1$
    return $\text{Max}(\text{Overlap}(l_1+\delta_1,r_1+\delta_1,l_2,r_2),\ \text{Overlap}(l_1+\delta_2,r_1+\delta_2,l_2+\delta_2,r_2+\delta_2))$
  end if
end function

复杂度

由于$\text{highbit}(x)$的操作复杂度为$O(\log x)$，所以原问题$\text{LCS}(l_1,r_1,l_2,r_2)$的复杂度$T(l_1,r_1,l_2,r_2)$为：

$$T(l_1,r_1,l_2,r_2)=O\left(\log\left(l_1\oplus r_1\right)+\log\left(l_2+\oplus r_2\right)\right)$$