算法与复杂性 第二次课程作业

UNIkeEN

2023 年 3 月 12 日

问题解答

1. 证明

$$\sum_{k=1}^{n}\frac{1}{k}=\Theta(\log n)$$

解 易得 $\frac{1}{k}$ 单调递减，故可作积分近似

$$\int_1^{n+1}\frac{1}{x}\mathrm{d}x\le\sum_{k=1}^{n}\frac{1}{k}\le 1+\int_1^{n}\frac{1}{x}\mathrm{d}x \tag{1}$$

$$\ln(n+1)-\ln 1\le\sum_{k=1}^{n}\frac{1}{k}\le 1+\ln n-\ln 1 \tag{2}$$

$$\ln n<\ln(n+1)\le\sum_{k=1}^{n}\frac{1}{k}\le 1+\ln n \tag{3}$$

$$\Omega(\log n)=\sum_{k=1}^{n}\frac{1}{k}=O(\log n) \tag{4}$$

即

$$\sum_{k=1}^{n}\frac{1}{k}=\Theta(\log n)$$

原式得证，证毕

2. 求解递推关系

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

其中 $T(1)=1$

解 令 $S(n)=\sum_{i=1}^{n}T(i)$，有

$$S(n)=S(n-1)+T(n)=\sum_{1}^{n-1}T(i)+T(n) \tag{5}$$

$$=2\sum_{1}^{n-1}T(i)+n=2S(n-1)+n \tag{6}$$

可得

$$S(n)+n+2=2(S(n-1)+(n-1)+2) \tag{7}$$

设 $b(n)=S(n)+n+2$, 有 $b(n)=2b(n-1)$。代入 $S(1)=T(1)=1$ 得 $b(1)=4$, $b(n)=2^{n+1}$
反解 $T(n)$，可得

$$T(n)=S(n)-S(n-1)=2^n-1 \tag{8}$$

经检验，$T(1)=1$ 符合上式，原式得解。

3. 设有如下递推关系

$$T(n)=\begin{cases}T(\frac{n}{2})+1 & ,n\text{为偶数}\\ 2T(\frac{n-1}{2}) & ,n\text{为奇数}\end{cases}$$

其中 $T(1)=1$

1. 证明当 $n=2^k$ 时，$T(n)=O(\log n)$
2. 证明存在无穷集合 X，当 $n\in X$ 时，$T(n)=\Omega(n)$
3. 以上两个结论说明了什么?

解 1.

$$T(n)=T(2^k)=T(\frac{n}{2})+1=T(2^{k-1})+1=T(2^{k-2})+2=\dots \tag{9}$$

$$=T(1)+k-1=k=\log n \tag{10}$$

得证，当 $n=2^k$ 时

$$T(n)=O(\log n)$$

2. 可举例，如无穷集合 $X=\left\{2^k-1\mid k\in\mathbb{N}^*\right\}$。证明如下:
易得 X 中元素均为奇数，则

$$T(n)=T(2^k-1)=2\cdot T(\frac{n-1}{2})=2\cdot T(2^{k-1}-1) \tag{11}$$

$$=4\cdot T(2^{k-2}-1)=\dots \tag{12}$$

$$=2^{k-1}\cdot T(2-1)=2^{k-1} \tag{13}$$

$$>2^{k-1}-\frac{1}{2}=\frac{2^k-1}{2}=\frac{n}{2} \tag{14}$$

所以

$$T(n)=\Omega(n)$$

3. 说明同一个算法在不同的特定输入下会表现出不同的时间复杂度特征。

4. 在寻找一对一映射问题中，算法结束时集合 S 是否可能为空集？请证明你的结论

解 不可能为空集。证明如下：

对于 A 中某一个元素 x，取 $f(x)$、$f(f(x))$... 据此嵌套，可以得到一个无限的序列 $b_i = f(x), f(f(x)), f(f(f(x)))$...，其中第 i 项表示嵌套 i 层。

由于 f 是从 $A \to A$ 的一对一映射，可知序列中的所有元素都在 A 中，但已知 A 是有限集，则必存在 $b_j = b_k$ 的循环。

存在循环，S 就至少可以包含这一循环间所有元素，可能出现的最小循环是 $f(x) = x$, 即 S 内元素的最小个数为 1。已证循环必然存在，所以 S 不可能为空集。

5. 课上的最大连续子序列是指和最大，如果要求是积最大呢？设空序列的积为 1，只需求出最大值即可，不需要知道是哪个序列

解 此时除了存储最大后缀外，还需存储最小后缀（因为负的最小后缀遇见新的负数后，负负得正可能得到更大的连续子序列）。**伪代码见算法1**

Algorithm 1 寻找积最大连续子序列

输入: $X[1..n]$，大小为 n 的数组
输出: 积最大连续子序列 $GlobalMax$

```
GlobalMax ← 1, SuffixMax ← 1, SuffixMin ← 1
for i ← 1 to n do
    SuffixMax ← SuffixMax · X_i
    SuffixMin ← SuffixMin · X_i
    if SuffixMax < 1 then
        SuffixMax ← 1
    end if
    if SuffixMax < SuffixMin then
        SuffixMax ← SuffixMin ▷ 负负得正，最小负乘积后缀变为更大的正乘积子序列
    end if
    if SuffixMin ≥ 0 then
        SuffixMin ← SuffixMax
    end if
    if SuffixMax > GlobalMax then
        GlobalMax ← SuffixMax
    end if
end for
return GlobalMax
```

6. 有 A、B、C 三个桩子，A 上放有 n 个不同大小的圆盘，按大小递减顺序自下而上

    1. 设计算法，将 A 上的圆盘都移到 C，每次只能移一个，且任一桩子上的圆盘都必须满足圆盘大小自下而上递减的顺序。给出算法即可，不用证明。
    2. 给定数组 p，其元素取值只有 1，2，3 三种，代表了 n 个圆盘目前的位置，1 代表 A，2 代表 B，3 代表 C，p[i] 的值为第 i 个圆盘的位置（i=1 代表最小的盘）。例如 p=3,3,2,1 表示共有 4 个圆盘，第一个和第二个都在 C，第三个在 B，第 4 个在 A。如果 p 代表了 1）中

的移动过程中某一步的状态，则求出这是第几步。注意诸如 p=[2,2] 是不可能出现的，此时算法得到-1

解 1. 运用递归，**伪代码见算法2**

2. 移动步数满足

$$T(n) = \begin{cases} 2 \times T(n-1) + T(1) & , n \geq 2 \\ 1 & , n = 1 \end{cases} \tag{15}$$

解得 $T(n) = 2^n - 1$。

现考虑存储各圆盘位置的数 $p[i]$，当问题规模为 n 时，我们考察 $p[n]$。

- 当 $p[n] = start$ 时，表示当前处于将 n-1 个圆盘从其起始柱移动到过渡柱的过程，记 $f[n] = 0$
- 当 $p[n] = end$ 时，表示当前处于将 n-1 个圆盘从其过渡柱移动到目标柱的过程，记 $f[n] = 1$

迭代进行上述过程，则可得到序列 $\{f[i]\}_{i \in [1,n]}$。

则 $N = \sum_{i=1}^{n} f[i] \cdot 2^{i-1}$ 可表示到此状态时已经经过的步数。**伪代码见算法3**

Algorithm 2 汉诺塔问题

输入: 三个桩 A、B、C ▷ 可以用栈存储，初始时栈 A 从栈顶到栈底依次是从小到大（标号从 0 到 n）的 n 个元素，B 和 C 初始为空

```
1: procedure MAIN
2:     HANOI(A, B, C, n)
3: end procedure

4: procedure HANOI(start, via, end, k)
                ▷ 求解将 k 个圆盘从 start 柱经由 via 柱，以汉诺塔规则送到 end 柱的子问题
5:     if k = 1 then
6:         ▷ 将 start 柱最顶端元素移动到 end 柱，若用栈存储，此处为 end.push(start.pop())
7:     else
8:         HANOI(start, finish, via, k − 1)
9:         HANOI(start, via, finish, 1)
10:        HANOI(via, start, finish, k − 1)
11:    end if
12: end procedure
```

Algorithm 3 检查汉诺塔状态

输入: 数组 $p[1..n]$ 表示用上述算法解决 n 个圆盘的汉诺塔问题的一个中间状态

输出: 到达这一状态经过的步数 N，如果该状态非法则为 -1

```
1: start ← 1, via ← 2, finish ← 3
2: N ← 0, f[1..n] ← 0
3: for i ← n downto 1 do
```

```
    if p[i] = start then
        SWAP(finish, via)
    else if p[i] = finish then
        N ← N + 2^(i−1)
        SWAP(start, via)
    else
        return −1
    end if
end for
return N
```

7. 背包问题中如果各种大小的物品的数量不限，那么如何知道背包是否能够恰好放满?

解 类似硬币找零问题。设背包大小为 K，n 种物品的大小分别为 s_i。

设布尔型变量 $P(n, K)$ 表示使用前 n 种物品填充 K 容量的背包是否能够恰好放满，则有状态转移方程

$$P(n, K) = P(n-1, K)||P(n, K - s_n) \tag{16}$$

伪代码见算法4

Algorithm 4 物品数量不限的背包问题

输入: $K, S[1..n]$
输出: 是否有解

```
P[0..n][0..K] ← false
P[0][0] ← true
for k = 0 to K do
    for i = 1 to n do
        if P[i − 1][k] = true then
            P[i][k] ← true
        else if k − S[i] ≥ 0 and P[i][k − S[i]] = true then
            P[i][k] ← true
        end if
    end for
end for
```

8. 如果背包问题每个物品有价值 vi，那么在不超过背包容量的情况下，如何使背包中物品价值最高？只需输出最大价值即可

解 设背包大小为 K，n 种物品的大小和价值分别为 s_i、v_n。

设 $P(n, K)$ 表示使用前 n 种物品填充 K 容量的背包的最大价值，则有状态转移方程

$$P(n, K) = max(P(n-1, K), P(n-1, K - s_n) + v_n) \tag{17}$$

特别地，当小规模问题的容量 K 放不下第 n 件物品（即 $K < s_n$）时, 有

$$P(n, K) = P(n-1, K) \tag{18}$$

算法设计类似4

9. 给定整数数组 A[1..n]，相邻两个元素的值最多相差 1。设 A[1]=x，A[n]=y，并且 $x < y$，输入 z，$x \le z \le y$，判断 z 在数组 A 中出现的位置。给出算法及时间复杂性（不得穷举）

解 相邻两个元素的值最多相差 1，即 $A[i+1] - A[i] \in \{-1, 0, 1\}$。

类比连续函数上的零点存在性定理，即

$$\text{若 } A[i] \le z \le A[j], (i < j)\text{，则一定存在一个 } k : i \le k \le j \text{ 使 } A[k] = z$$

注，上述结论只能保证找到解 k，并不保证找到所有解（类比函数不单调）。

题目要求找到一个 z 即可，因此可用二分法，时间复杂度为 $O(\log n)$，**伪代码见算法5**

Algorithm 5 二分法寻找函数

输入: 整数数组 $A[1..n]$, A 中的一个整数 z
输出: 元素 z 在数组 A 中的位置

```
left ← 1, rignt ← n
mid ← ⌊(left + right)/2⌋
while A[mid] ≠ z do
    if A[mid] < z then
        left ← mid
    else
        right ← mid
    end if
    mid ← ⌊(left + right)/2⌋
end while
return mid
```

10. 每个螺母需要一个螺栓配套使用，现有 n 个不同尺寸的螺母和相应的 n 个螺栓，如何快速地为为每一个螺母找到对应的螺栓？只允许将一个螺母与一个螺栓进行匹配尝试，从而知道相互之间的大小关系，不能够比较两个螺母的大小，也不能比较两个螺栓的大小。给出算法及相应的时间复杂性

解 首先从螺母中随机取一个数（默认第一个元素），以该数作为螺钉的基数去进行一次快速排序：

1. 假设该基准元素位于最后一位，i j 同时从首个元素出发，j 从第一个元素开始向后寻找比基准小或基准元素

   (1) 当 j 位置的元素小于基准元素，则 i j 的元素交换，i 往后移动一位，j 继续往后寻找;

   (2) 当 j 位置的元素等于基准元素，则 j 所在的与最后一位元素进行交换，j 从当前位置继续往后寻找;

   (3) 当 j 位置的元素大于基准元素，则 j 继续往后寻找;

2. 直到 j == right 时，交换 i j 所在的元素，返回 i ，一次排序结束。

从螺钉中选择进行一次排序返回的基准元素，作为螺母的基准元素去对螺母进行一次快速排序：(同以上步骤)

1. 假设该基准元素位于最后一位，i j 同时从首个元素出发，j 从第一个元素开始向后寻找比基准小或基准元素：

   (1) 当 j 位置的元素小于基准元素，则 i j 的元素交换，i 往后移动一位，j 继续往后寻找;

   (2) 当 j 位置的元素等于基准元素，则 j 所在的与最后一位元素进行交换，j 从当前位置继续往后寻找;

   (3) 当 j 位置的元素大于基准元素，则 j 继续往后寻找;

2. 直到 j == right 时，交换 i j 所在的元素，返回 i ，一次排序结束。

根据以上步骤，将基准元素左、右的部分数组再次进行排序，直到数组有序

由于两个数组中的元素是 1:1 匹配的，所以当两个数组有序之后，螺母与螺钉的匹配问题也就解决了。

本算法借助了快速排序的思想，算法复杂度在平均情况下为 $O(n \log n)$。

11. 设 A[1..n] 是一个包含 n 个不同数的数组。如果在 i<j 的情况下有 A[i]>A[j]，则 (i,j) 为 A 中的一个逆序对。1）若 A= {2，3，8，6，1}，列出其中所有的逆序对；2）若数组元素取自集合 1,2,⋯,n，那么怎样的数组含有最多的逆序对？它包含多少个逆序对？3）求任意数组 A 中逆序对的数目，并证明算法的时间复杂性为 $\Theta(nlogn)$

解 1. 逆序对有 5 个，分别是 (2,1)、(3,1)、(8,6)、(8,1)、(6,1)

2. 含有最多逆序对的数组是把集合中元素逆序排放，A[1..n]=[n,n-1,...,1]，每个元素和其之后所有元素构成逆序对，总共含 $\sum_{i=1}^{n}(i-1) = \frac{n(n-1)}{2}$ 个逆序对

3. 可以在归并排序的基础上寻找逆序对。将数组二分成左子序列和右子序列，分别求子序列的逆序对数相加，并加上在归并的过程中产生的新逆序对数量。

$$T(n) = 2T(n/2) + bn \tag{19}$$

其中，b 是任意常数，可知时间复杂度是 $O(n \log n)$

伪代码见算法6

Algorithm 6 求逆序对数

输入: 数组 $A[1..n]$

输出: A 中的逆序对数 m

```
1: procedure MERGE(A[1..n])
                          ▷ 输入一个数组 A，输出该数组中的逆序对数和该数组排序后结果
2:     if Len(A) = 1 then
```

```
        return (0, A)
    else if Len(A) = 2 then
        if A[1] > A[2] then
            return (1, [A[2], A[1]])
        else
            return (0, A)
        end if
    else                                                    ▷ 分治
        (nL, L) ← MERGE(A[1.. ⌊n/2⌋])
        (nR, R) ← MERGE(A[⌊n/2⌋ + 1..n])
        k ← 1, cnt ← 0                                      ▷ 归并
        while ¬L.isEmpty ∧ ¬R.isEmpty do
            if L.front ≤ R.front then
                A[k] = L.Pop()
            else
                A[k] = R.Pop()
                cnt ← cnt + Len(R)
            end if
            k ← k + 1
        end while
        if ¬L.isEmpty then                                  ▷ 填充剩余元素
            A[k..n] = L
        else if ¬R.isEmpty then
            A[k..n] = R
        end if
        return (cnt, A)
    end if
end procedure
```