算法设计与分析作业 3

杨树鑫 2019E8013261016

October 2th , 2019

本次作业选做 3、4 题。

1 第 3 题 Cut Rope

1.1 算法思路描述

1. 当 $n = 1, 2, 3, 4$ 时，直接给出最优解；

2. 当 $n > 4$ 时，长度为 n 的绳子可以从位置 k 被分为乘积最大的两段 a 和 b，a 和 b 又可以按此方法继续各自被分为乘积最大的两段，直到长度小于等于 4。(在算法正确性部分将会证明 $a > 2, b > 2, a + b = n$ 时，有 $ab > a + b$ 成立，因此，对于一个长为 $n(n > 4)$ 的绳子，我们总是可以将其继续分为更小的两段，使其乘积比和更大)

再反推，由绳长为 $1, 2, 3, 4$ 只能组成的最大长度为 7，观察 $5, 6, 7$ 这三种情况的最大切法都包含 3，而更大的数又可以分成这几个数，所以只需将原绳子先按长度 3 进行切分，切分到长度小于等于 4 时停止, 即可得到原问题的解。

1.2 伪代码

伪代码如算法 1 所示

1.3 Greedy 选择性质和最优子结构

对一根长为 $n(n > 4)$ 的绳子，我们总是可以将其继续分为 3 和 $OPT(n-3)$ 的两段 ($OPT(n-3)$ 表示如果 $n - 3 > 4$，那么还继续分割，否则按平凡解给出结果)，使其乘积更大，当 $n = 1, 2, 3, 4$ 时，可以直接给出最优解。

$$OPT(i) = \begin{cases} 3 * OPT(i-3) & \text{i>4} \\ 4 & \text{i=4} \\ 2 & \text{i=3} \\ 1 & \text{i=1or2} \end{cases}$$

算法 1 切绳子

输入: n 绳子的长度

输出: 切割的最大乘积

```
function CutRope(n)
    if n == 1or2 then return 1
    end if
    if n == 3 then return 2
    end if
    if n == 4 then return 4
    end if
    k ← 0
    for i = 5 to n do
        OPT(i) ← 3 * OPT(i − 3)
    end for
    return OPT(n)
end function
```

1.4 算法正确性

首先证明当 $a > 2, b > 2$ $a + b = n$ 时，有 $ab > a + b$ 成立

证明:

$$\because a + b = n$$
$$\therefore b = n - a > 2$$
$$\therefore ab = a(n - a), n > a + 2$$
$$\because ab - (a + b)$$
$$\begin{aligned} &= a(n-a) - n \\ &= na - a^2 - n \\ &= (a-1)n - a^2 \\ &> (a-1)(a+2) - a^2 \\ &= a^2 + a - 2 - a^2 \\ &= a - 2 \end{aligned}$$
$$\because a > 2$$
$$\therefore ab - (a + b) > a - 2 > 0$$
$$\therefore ab > a + b$$

根据以上证明，当 $n = 1, 2, 3, 4$ 时，是平凡的情况，直接给出最优解。

当 $n > 4$ 时，长度为 n 的绳子可以从位置 k 被分为乘积最大的两段 a 和 b，a 和 b 又可以按此方法继续各自被分为乘积最大的两段，直到长度小于等于 4。再反推，由绳长为 $1, 2, 3, 4$ 只能组成的最大长度为 7，观察 $5, 6, 7$ 这三种情况的最大切法都包含 3，而更大的数又可以分成这几个数，所以只需将原绳子先按长度 3 进行切分，切分到长度小于等于 4 时停止, 即可得到原问题的解。

因此，对一根长为 $n(n>4)$ 的绳子，我们总是可以将其继续分为 3 和 $OPT(n-3)$ 的两段，使其乘积更大

综上，算法正确

1.5 算法复杂度

原问题需要计算 $OPT(4) \sim OPT(n)$ 一遍，所以原问题的时间复杂度为

$$T(n)=O(n)$$

由于开了一个 n 的数组存储子问题的最优解，因此原问题的空间复杂度为

$$O(n)$$

2 第 4 题 Cross River

2.1 算法思路描述

先将整个数组按升序排序，用 i 表示当前未被选取的体重最小的人的索引，用 j 表示当前未被选取的体重最大的人的索引，然后从最大的数开始选取，直到 $i=j$:

1. 如果 $w[j]+w[i]>limit$，则只选择当前体重最大的人，j 减小 1；
2. 如果 $w[j]+w[i]\le limit$，则选择当前体重最大的人和最小的人，j 减小 1，i 增大 1；

2.2 伪代码

伪代码如算法 2 所示

算法 2 过河

输入: *Array* 体重数组 w，*int* 人数 n, *int* 最大限制 *limit*

输出: 最少的船数量

```
function CrossRiver(w, n, limit)
    if n == 1 then return 1
    end if
    sort(w)
    i ← 0, j ← n − 1
    boat ← 0
    while i ≤ j do
        boat ← boat + 1
        if w[j] + w[i] ≤ limit then
            i ← i + 1
        end if
        j ← j − 1
    end while
    return boat
end function
```

2.3 Greedy 选择性质和最优子结构

优先选择当前未被选择的最重的人和最轻的人，如果两人体重之和满足限制，就同时载上，否则只载最重的人

$w[i]$ 表示人的体重数组，且已按非降序排序;

$$OPT(i,j)=\begin{cases}OPT(i+1,j-1)+1 & w[j]+w[i]\leq limit\\ OPT(i,j-1)+1 & w[j]+w[i]>limit\end{cases}$$

2.4 算法正确性

1. 对只能载最重的一个人的情况，显然必须要一条船；

2. 对载了最重的一个人的情况下，只能再载一个最轻的人时，即 $w[j]+w[i]=limit$ 时，那就必须载最轻的人才能使船的数量最小。采用反证法

假设不载最轻的人，那么想载两个人的话，重量就会超过限制，必须要再加一条船，与最少船数量矛盾

因此，必须载最轻那个人。

3. 对载了最重的一个人的情况下，还能再载不一定是最轻的人时，即 $w[j]+w[i]<limit$，其解不唯一，但解中一定含有第 (2) 条中的情况，因此为简单起见，就选择 (2) 的组合方式，即可得到最小船数量。

综上，算法正确

2.5 算法复杂度

原问题将数组从两头往中间遍历一遍，所以原问题的时间复杂度为

$$T(n)=O(n)$$

由于只用了 int 变量存储结果值，没有数组等其他空间分配，因此原问题的空间复杂度为

$$O(1)$$