《算法设计与分析》课程作业二

尹达恒

2020/10/27

1 钢条切割

1.1 问题描述

Description

给定一根长度为 $n(n \leq 10000)$ 的钢条以及一张价格表，请计算这根钢条能卖出的最大总收益。价格表表示为 $(l_i, p_i), 1 \leq i \leq k$。不在价格表中的钢条可卖出价格为 0。

Input

第一行输入 `m(m≤10)` 表示有 M 组数据。每组数据第一行输入两个 `int` 型整数 `n` 和 `k`，分别表示钢条长度以及价格表中不同价格数量。接下来一行输入 `k` 个价格的表示 (l_i, p_i)，均为整数，l_i 可能大于 `n`。

Output

输出 `m` 行整数，第 `i` 行表示第 `i` 组数据的最大总收益。

Sample Input

```
2
27 3
35 41 61 49 73 74
94 2
21 55 88 64
```

Sample Output

```
0
220
```

1.2 算法思路

令 $S_L = \{L_i | i \in [1, N]\}$ 表示长 L 的钢条在价格表 $P = \{(l, p_l) | l \in \mathbb{N}, p \in \mathbb{R}\}$ 下的最优切割方案，其价格为 $P(S)$，显然最优切割方案有如下性质：

1. 最优子结构：

$$(\forall S \subseteq S_L) S = S_{\sum_{l \in S} l}$$

2. 重叠子问题：

$$P(S_L) = max\{P(S_{L-l}) + p_l | (l, p_l) \in P\}$$

因此可以采用查表法，求解长 L 的钢条的最优切割方案时，使用数组存储 $P(S_i)$，从 $i = 0$ 开始依次计算 $P(S_L) = max\{P(S_{i-l}) + p_l | (l, p_l) \in P\}$ 直到 $i = L$ 即为要求的 $P(S_L)$。

1.3 算法伪代码

见算法 1。

Algorithm 1: 钢条切割算法伪代码

```
1  Function SteelCut(L, P) begin
       Input: 钢条长度 L ∈ ℕ+、价格表 P = {(l, p_l)|l ∈ ℕ, p ∈ ℝ}
       Output: 最佳切割方案价格 P(S_L)
2      for i ∈ {1, 2, 3, ..., L} do
3          P_i = 0;
4          for (l, p_l) ∈ P do
5              if l ≤ i then
6                  P_i = max(P_i, l + P_{i-l});
7              else
8                  break;
9              end
10         end
11         记下 P_i;
12     end
13     return P_L
14 end
```

2 最长公共子序列

2.1 问题描述

Description

给定两个字符串 A 和 B，请计算这两个字符串的最长公共子序列长度。

Input

第一行输入 M(M≤10) 表示有 M 组数据。每组数据输入两行字符串，字符串的长度不长于 500。

Output

输出 M 行正整数，第 i 行表示第 i 组数据的最长公共子序列长度。

Sample Input

```
2
abcdefg
cemg
abcdefgh
ceaaegh
```

Sample Output

```
3
4
```

2.2 算法思路

设字符串 $A = a_1a_2 \dots a_m \dots a_M$ 和字符串 $B = b_1b_2 \dots b_n \dots b_N$ 的最长公共子序列为 $C = C(A,B) = c_1c_2 \dots c_k \dots c_K, c_k = a_{m_k} = b_{n_k}, m_k < m_{k+1}, n_k < n_{k+1}$。定义字符串前缀 $A_i = a_1a_2 \dots a_i, B_i = b_1b_2 \dots b_i$，易得 $C = C(A,B)$ 具有如下性质：

- 最优子结构：

$$C_i = C(A_{m_i}, B_{n_i})$$

- 重叠子问题：

$$|C(A_m, B_n)| = \begin{cases} max\left(|C(A_{m-1}, B_n)|, |C(A_m, B_{n-1})|\right) & a_m \neq b_n \\ |C(A_{m-1}, B_{n-1})| + 1 & a_m = b_n \end{cases}$$

因此可以采用查表法，求解字符串 A 和 B 的最长公共子序列长度时，使用矩阵存储 $|C(A_m, B_n)|$，按重叠子问题公式从 $m = n = 0$ 开始依次计算 $|C(A_m, B_n)|$，直到 $m = M, n = N$ 即得到所需结果 $|C(A,B)|$。

2.3 算法伪代码

见算法 2。

3 最低票价

3.1 问题描述

Description

在一个火车旅行很受欢迎的国度，你提前一年计划了一些火车旅行。在接下来的一年里，你要旅行的日子将以一个名为 `days` 的数组给出。每一项是一个从 `1` 到 `365` 的整数。

火车票有三种不同的销售方式：

- 一张为期一天的通行证售价为 `costs[0]` 美元；
- 一张为期七天的通行证售价为 `costs[1]` 美元；
- 一张为期三十天的通行证售价为 `costs[2]` 美元。

通行证允许数天无限制的旅行。例如，如果我们在第 2 天获得一张为期 7 天的通行证，那么我们可以连着旅行 7 天：第 2 天、第 3 天、第 4 天、第 5 天、第 6 天、第 7 天和第 8 天。

返回你想要完成在给定的列表 `days` 中列出的每一天的旅行所需要的最低消费。

Input

第一行输入 `nums` 表示有 `nums` 组测试

对每组测试用例

第一行输入 `m`

第二行输入具有 `m` 个元素的 `days` 数组,`days[i]` 表示你将在 `days[i]` 这天旅行

第三行输入具有 3 个元素的 `costs` 数组，具体释义见 Description

Output

对每组测试数据，输出你想要完成在给定的 `days` 数组中列出的每一天的旅行所需要的最低消费。

Algorithm 2: 最长公共子序列算法伪代码

Function *CommonLongest(A, B)* **begin**

Input: 字符串 $A = a_1a_2 \dots a_m \dots a_M$、$B = b_1b_2 \dots b_n \dots b_N$

Output: 最长公共子序列长度 $|C| = |C(A_M, B_N)|$

$$|C(A_1, B_1)| = \begin{cases} 0 & a_1 \neq b_1 \\ 1 & a_1 = b_1 \end{cases}$$

for $m \in \{2, 3, ..., M\}$ **do**

$$|C(A_m, B_1)| = \begin{cases} 0 & a_m \neq b_n \wedge |C(A_{m-1}, B_1)| = 0 \\ 1 & a_m = b_n \vee |C(A_{m-1}, B_1)| = 1 \end{cases}$$

end

for $n \in \{2, 3, ..., N\}$ **do**

$$|C(A_1, B_n)| = \begin{cases} 0 & a_m \neq b_n \wedge |C(A_1, B_{n-1})| = 0 \\ 1 & a_m = b_n \vee |C(A_1, B_{n-1})| = 1 \end{cases}$$

end

for $m \in \{2, 3, ..., M\}$ **do**

for $n \in \{2, 3, ..., N\}$ **do**

$$|C(A_m, B_n)| = \begin{cases} max\left(|C(A_{m-1}, B_n)|, |C(A_m, B_{n-1})|\right) & a_m \neq b_n \\ |C(A_{m-1}, B_{n-1})| + 1 & a_m = b_n \end{cases}$$

end

end

return $|C(A_M, B_N)|$;

end

Sample Input

```
2
6
1 4 6 7 8 20
2 7 15
12
1 2 3 4 5 6 7 8 9 10 30 31
2 7 15
```

Sample Output

```
11
17
```

提示

`1 ≤ days.length ≤ 365`
`1 ≤ days[i] ≤ 365`
`days` 按顺序严格递增
`costs.length ≡ 3`
`1 ≤ costs[i] ≤ 1000`

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：

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

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 $\mathbb{M}$ 和 $\mathbb{N}$ 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 $\mathbb{U}$ 中第 $2k$ 大数 U_{2k} 的算法：

1. 若两数组长度和为偶数，则 $2k = (|\mathbb{M}| + |\mathbb{N}|)/2$，查找 U_{2k} 和 U_{2k+1} 取平均值；
2. 若两数组长度和为奇数，则 $2k = (|\mathbb{M}| + |\mathbb{N}| + 1)/2$，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 $\{M_i \in \mathbb{M} | 1 \leq i \leq k\}$ 中，因此可以直接舍弃该部分，在 $\{M_i \in \mathbb{M} | k+1 \leq i \leq |\mathbb{M}|\}$ 和 $\mathbb{N}$ 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 $\mathbb{M}$ 或 $\mathbb{N}$ 为空。算法的时间复杂度为 $O(log|\mathbb{M}| + log|\mathbb{N}|)$。

3.3 算法伪代码

见算法 3和算法 4。

4 鸡蛋掉落

4.1 问题描述

Description

你将获得 K 个鸡蛋，并可以使用一栋从 1 到 N 共有 N 层楼的建筑。

每个蛋的功能都是一样的，如果一个蛋碎了，你就不能再把它掉下去。

你知道存在楼层 F，满足 $0 \leq F \leq N$ 任何从高于 F 的楼层落下的鸡蛋都会碎，从 F 楼层或比它低的楼层落下的鸡蛋都不会破。

每次移动，你可以取一个鸡蛋（如果你有完整的鸡蛋）并把它从任一楼层 X 扔下（满足 $1 \leq X \leq N$）。

你的目标是确切地知道 F 的值是多少。

无论 F 的初始值如何，你确定 F 的值的最小移动次数是多少？

Algorithm 3: 在两个正序数组中找第 k 大数

```
1  Function FindK(𝕄, ℕ, k) begin
       Input: 数组 𝕄 和 ℕ、整数 k
       Output: 数组 𝕄 和 ℕ 中第 k 大的数
2      if 𝕄 = ∅ then return N_k;
3      if ℕ = ∅ then return M_k;
4      if k = 1 then return min(M_1, N_1);
5      d = ⌊k/2⌋;
6      if d > |𝕄| then
7          if M_|𝕄| ≤ N_d then
8              return FindK(∅, ℕ, k − |𝕄|);
9          else
10             return FindK(𝕄, {N_i ∈ ℕ | i > d}, k − d);
11         end
12     end
13     if d > |ℕ| then
14         if N_|ℕ| ≤ M_d then
15             return FindK(𝕄, ∅, k − |ℕ|);
16         else
17             return FindK({M_i ∈ 𝕄 | i > d}, 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 ∈ 𝕄 | i > d}, ℕ, k − d);
26     else
27         return FindK(𝕄, {N_i ∈ 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(𝕄, ℕ) begin
       Input: 正序数组 𝕄、ℕ
       Output: 数组 𝕄 和 ℕ 的中位数
2      if |𝕄| + |ℕ| 为奇数 then
3          if 𝕄 = ∅ then return N_{(|𝕄|+|ℕ|+1)/2};
4          if ℕ = ∅ then return M_{(|𝕄|+|ℕ|+1)/2};
5          return FindK(𝕄, ℕ, (|𝕄|+|ℕ|+1)/2);
6      else
7          if 𝕄 = ∅ then return (N_{(|𝕄|+|ℕ|)/2} + N_{(|𝕄|+|ℕ|)/2+1})/2;
8          if ℕ = ∅ then return (M_{(|𝕄|+|ℕ|)/2} + M_{(|𝕄|+|ℕ|)/2+1})/2;
9          return
             (FindK(𝕄, ℕ, (|𝕄|+|ℕ|)/2) + FindK(𝕄, ℕ, (|𝕄|+|ℕ|)/2 + 1))/2;
10     end
11 end
```

Input

第一行输入 `nums` 表示有 `nums` 组测试
每组测试输入 `K`，`N`，表示有 K 个鸡蛋，N 层楼

Output

对每组测试数据，输出确定 F 的最小移动次数

Sample Input

```
3
1 2
2 6
3 14
```

Sample Output

```
2
3
4
```

提示

$1 <= \texttt{K} <= 1000$
$1 <= \texttt{N} <= 10000$

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：

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

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。

5 平台截图

Algorithm 5: 搜索二维矩阵算法伪代码

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 $a_{i,j} = x$
  if $m = 0 \vee n = 0$ **then return** false;
  if $x = a_{1,n}$ **then return** true;
  if $x > a_{1,n}$ **then**
    二分查找满足条件的 k：$k > 1 \wedge a_{k,n} \geq x \wedge a_{k-1,n} < x$;
    return $FindNumber(A'_{k,n} = (a_{i,j})(i \in [k, n], j \in [1, n]), x)$;
  else
    二分查找满足条件的 k：$k > 1 \wedge a_{1,k} \leq x \wedge a_{1,k+1} > x$;
    return $FindNumber(A'_{m,k} = (a_{i,j})(i \in [1, n], j \in [1, k]), x)$;
  end
end

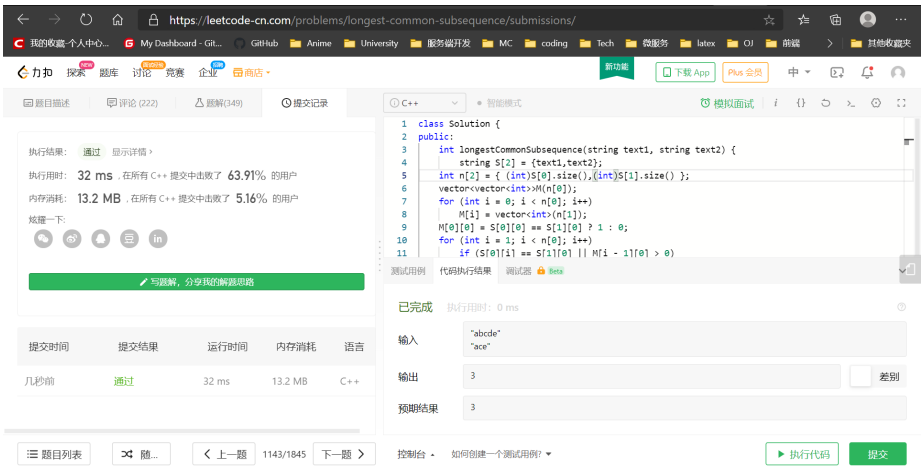

图 1: LeetCode 4 Accepted 截图

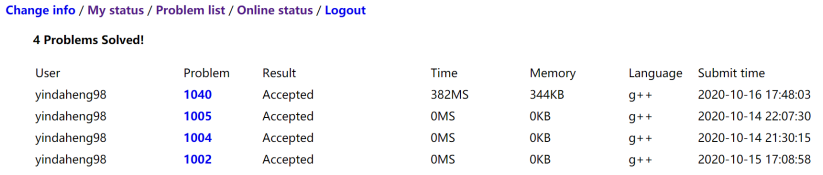

图 2: 47.99.179.148 Accepted 截图