《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

Function *SteelCut(L, P)* **begin**
    Input: 钢条长度 $L \in \mathbb{N}_+$、价格表 $P = \{(l, p_l) | l \in \mathbb{N}, p \in \mathbb{R}\}$
    Output: 最佳切割方案价格 $P(S_L)$
    for $i \in \{1, 2, 3, ..., L\}$ **do**
        $P_i = 0$;
        for $(l, p_l) \in P$ **do**
            if $l \leq i$ **then**
                $P_i = max(P_i, l + P_{i-l})$;
            else
                break;
            end
        end
        记下 P_i;
    end
    return P_L
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 $A = a_1a_2 \ldots a_m \ldots a_M$ 和字符串 $B = b_1b_2 \ldots b_n \ldots b_N$ 的最长公共子序列为 $C = C(A, B) = c_1c_2 \ldots c_k \ldots c_K, c_k = a_{m_k} = b_{n_k}, m_k < m_{k+1}, n_k < n_{k+1}$。定义字符串前缀 $A_i = a_1a_2 \ldots a_i, B_i = b_1b_2 \ldots b_i$，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。

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

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。

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 $\leq$ `days.length` $\leq$ 365

1 $\leq$ `days[i]` $\leq$ 365

`days` 按顺序严格递增

`costs.length` $\equiv$ 3

1 $\leq$ `costs[i]` $\leq$ 1000

3.2 算法思路

对于要旅行日集合 S 和价目表 $P = \{(d, p)\}$，令 x 天内的最低消费方案表示为 $F_x = \{(x, d)|$第 x 天购买了时长为 d 天的车票$\}$，花费的票价为 $f(x) = \sum_{(x,d)\in F_x \wedge (d,p)\in P} p$，易得最低消费有如下性质：

- 最优子结构：

$$(\forall x' \leq x) F_{x'} \subseteq F_x$$

- 重叠子问题：

$$f(x) = \begin{cases} min\{f(x-d)+p|(d,p)\in P\} & x \in S \\ f(x) = f(x-1) & x \notin S \end{cases}$$

因此可以使用查表法，在求解 $f(X)$ 时，使用数组存储 $f(x)$，从 0 开始，按照重叠子问题公式依次计算 $f(x)$，直到 $f(X)$ 即为所需结果。

3.3 算法伪代码

见算法 3。

Algorithm 3: 最低票价

Function *FindK(X, S, P)* **begin**

Input: 需要安排购票方案的总天数 X、旅行日集合 S、价目表 $P = \{(d, p)\}$

Output: 最低消费方案花费的票价 $f(X)$

令 $(\forall x \leq 0) f(x) = 0$;

for $x \in \{1, 2, ..., X\}$ **do**

if $x \in S$ **then**

$$f(x) = \begin{cases} min\{f(x-d) + p|(d,p) \in P\} & x \in S \\ f(x) = f(x-1) & x \notin S \end{cases}$$

end

end

return $f(X)$

end

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？

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 <= `K` <= 1000

1 <= `N` <= 10000

4.2 算法思路

设满足条件的 F 最小移动次数为 $M = f(K, N)$，令 $n = g(k, m)$ 表示使用 k 个鸡蛋移动 m 次可以确定 F 的最大楼层数。对于此函数，可以推理出如下性质：

1. 鸡蛋数量和移动次数至少为 1，即：

$$k > 1, m > 1$$

2. 如果鸡蛋数量大于移动次数，则多出来的鸡蛋必定不会被用到，因此可以确定的楼层数和鸡蛋数量等于移动次数的情况相同，即：

$$(\forall k < m) g(k, m) = g(m, m)$$

3. 对于只有一个鸡蛋和一次移动的情况（$k = m = 1$），如果把鸡蛋从 1 楼扔下，若碎了，则确定 $F = 0$，若没碎，则无法确定 F；但如果把它从较高楼层上扔下，不管鸡蛋碎没碎，都无法确定 F，因此有这种情况最多只能确定 $F = 0$ 的情况，即：

$$g(1, 1) = 0$$

4. 进一步，对于有一个鸡蛋和 m 次移动的情况，显然只能从 $1 \sim m$ 楼依次扔鸡蛋，因为如果从较高层扔鸡蛋碎了，就无法确定 F 的准确值。这时最大的 F 出现于最后一次从 m 层扔下时鸡蛋碎了的情况，即 $F = m - 1$ 的情况，因此有：

$$g(1, m) = m - 1$$

5. 更进一步，对于有两个鸡蛋和 m 次移动的情况，显然，可以一开始就将鸡蛋从第 m 层扔下，如果碎了，说明 $F \leq m - 1$，剩下的 $m - 1$ 个鸡蛋可以保证在 $m - 1$ 层之内找到 F，可以确定 F 的楼层数最高为 $F = m - 1$ 层；如果没碎，说明 $F \geq m$，那么剩下的两个鸡蛋和 $m - 1$ 次就是一个从 m 层开始的 $g(2, m - 1)$ 问题，即最大可确定 F 的层数为：

$$g(2, m) = m + g(2, m - 1)$$

6. 同理可知，对于 k 个鸡蛋和 m 次移动的情况，可以一开始就将鸡蛋从第 $g(k - 1, m - 1) + 1$ 层扔下，如果碎了，剩下的 $k - 1$ 个鸡蛋和 $m - 1$ 次移动可以保证在 $g(k - 1, m - 1)$ 层内找到 F；如果没碎，那

么剩下的 k 个鸡蛋和 $m-1$ 次移动就是一个从 $g(k-1,m-1)+1$ 层开始的 $g(k,m-1)$ 问题，即：

$$g(k,m) = g(k-1,m-1) + 1 + g(k,m-1)$$

综上所述，求解 $N = g(K,M)$ 可以使用递归，按照 $g(k,m) = g(k-1,m-1)+1+g(k,m-1)$ 进行递归计算即可。进而，对 $M = f(K,N)$ 的求解可以转化为一系列 $N = g(K,m)$ 的问题，即令 m 从 1 开始，依次计算 $n = g(K,m)$ 直到 $n = g(K,m) \geq N$，此时的 m 即为所求值。

4.3 算法伪代码

见算法 4。

Algorithm 4: 鸡蛋掉落算法伪代码

Function *f(K, N)* **begin**
  Input: 鸡蛋个数 K、楼层高度 N
  Output: 确定 F 的值的最小移动次数 M
  Function *g(k,m)* **begin**
    Input: 鸡蛋个数 k、移动次数 m
    Output: k 个鸡蛋移动 m 次可以确定 F 的最大楼层数 n
    if $k = 1 \vee m = 1$ **then return** 1;
    return $g(k,m) = g(k-1,m-1)+1+g(k,m-1)$
  end
  $m = 1$;
  for $m \in \{m | g(k,m) < N\}$ **do**
    $m = m + 1$;
  end
  return m
end

5 平台截图

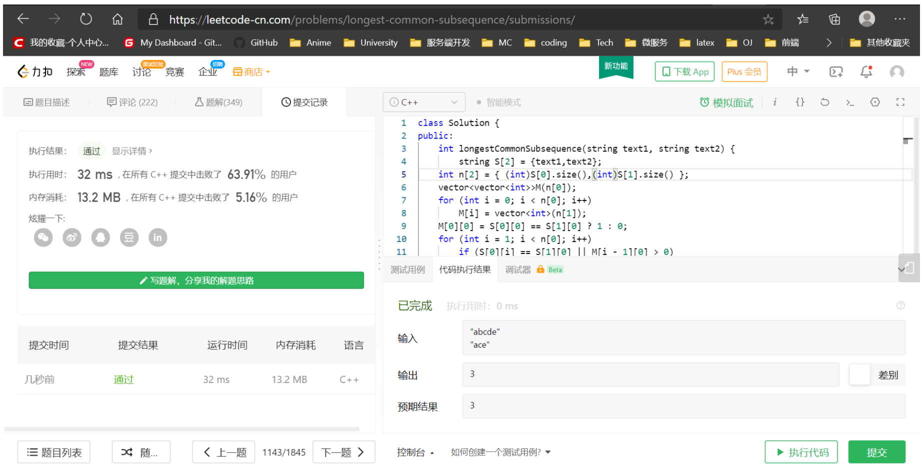

图 1: LeetCode 4 Accepted 截图

Change info / **My status** / **Problem list** / **Online status** / **Logout**

4 Problems Solved!

User	Problem	Result	Time	Memory	Language	Submit time
yindaheng98	**1040**	Accepted	382MS	344KB	g++	2020-10-16 17:48:03
yindaheng98	**1005**	Accepted	0MS	0KB	g++	2020-10-14 22:07:30
yindaheng98	**1004**	Accepted	0MS	0KB	g++	2020-10-14 21:30:15
yindaheng98	**1002**	Accepted	0MS	0KB	g++	2020-10-15 17:08:58

图 2: 47.99.179.148 Accepted 截图