算法设计与分析第二次作业

杨博文 21373037

2023 年 10 月 22 日

1. 假日愉悦值问题

F 同学开始计划他的 N 天长假，第 i 天他可以从以下三种活动中选择一种进行。

1. 去海边游泳。可以获得 ai 点愉悦值；
2. 去野外爬山。可以获得 bi 点愉悦值；
3. 在家里学习。可以获得 ci 点愉悦值。

由于他希望自己的假日丰富多彩，他并不希望连续两天（或者两天以上）进行相同类型的活动。试设计算法制定一个假日安排，使得在满足 F 同学要求的情形下所获得的愉悦值的和最大，输出这个假日安排以及最大愉悦和，请描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度。

1. 状态设计

我们令游泳是第 0 项，爬山是第 1 项，学习是第 2 项活动，创建二维数组 dp[3][n], dp[i][j] 代表在第 j 天选择第 i 项活动时的最大愉悦值。

2. 状态转移

题目中要求不能连续两天选择同样的活动，对于 dp[i][j]，有三种情况：

1. i=0, 即第 j 天选择游泳活动, 有两种情况, j-1 天选择爬山或学习, 从中选择价值最大的, 即 $dp[0][j] = max(dp[1][j-1] + a_j, dp[2][j-1] + a_j)$
2. i=1, 即第 j 天选择爬山活动, 有两种情况, j-1 天选择游泳或学习, 从中选择价值最大的, 即 $dp[1][j] = max(dp[0][j-1] + b_j, dp[2][j-1] + b_j)$
3. i=2, 即第 j 天选择学习活动, 有两种情况, j-1 天选择游泳或爬山, 从中选择价值最大的, 即 $dp[2][j] = max(dp[0][j-1] + c_j, dp[1][j-1] + c_j)$

那么截至第 j 天的假期愉悦最大值为 $max(dp[0][j], dp[1][j], dp[2][j])$

3. 决策方案

在计算 dp 数组的同时维护一个 path[3][n] 数组，path[i][j] 代表第 j 天选择第 i 种活动获得最大愉悦值时前一天选择了哪个活动，如此便可在正向遍历完 dp 数组后反向逆推出每一天的选择。具体如下：

1. i=0, dp[1][j-1] > dp[2][j-1], 则 path[0][j] = 1, 否则 path[0][j] = 2

2. i=1, dp[0][j-1] > dp[2][j-1], 则 path[1][j] = 0, 否则 path[1][j] = 2

3. i=2, dp[0][j-1] > dp[1][j-1], 则 path[2][j] = 0, 否则 path[2][j] = 1

4. 起始条件

针对第一天决策初始化，$dp[0][1] = a_1, dp[1][1] = b_1, dp[2][1] = c_1$。

5. 遍历顺序

由于 dp[0..2][i] 依赖于 dp[0..2][i-1]，因此**从小到大遍历天数 n**。

5. 伪代码

Algorithm 1 假日愉悦值问题

Input: $a[1..n], b[1..n], c[1..n]$

Output: $maxValue, plan[1..n]$

$dp[0][1] \leftarrow a[1], dp[1][1] \leftarrow b[1], dp[2][1] \leftarrow c[1]$

for $i : 2 \rightarrow n$ **do**

  $dp[0][i] \leftarrow a[i] + max(dp[1][i-1], dp[2][i-1])$

  $dp[1][i] \leftarrow b[i] + max(dp[0][i-1], dp[2][i-1])$

  $dp[2][i] \leftarrow c[i] + max(dp[0][i-1], dp[1][i-1])$

  $path[0][i] \leftarrow (dp[1][i-1] > dp[2][i-1])?1 : 2$

  $path[1][i] \leftarrow (dp[0][i-1] > dp[2][i-1])?0 : 2$

  $path[2][i] \leftarrow (dp[0][i-1] > dp[1][i-1])?0 : 1$

end

$pos \leftarrow argmax_{k=0,1,2} dp[k][n], maxValue \leftarrow dp[pos][n], plan[n] \leftarrow pos$

for $i : n \rightarrow 2$ **do**

  $plan[n-1], pos \leftarrow path[pos][n]$

end

6. 时间复杂度分析

动规遍历天数共 n 个循环，每个循环内 $O(1)$ 的时间复杂度，动规时间复杂度为 $O(n)$；输出方案时遍历反推 path 数组，共循环 n 次，时间复杂度为 $O(n)$，因此该算法时间复杂度为 $O(n)$。

2. 倍序数组问题

一个长度为 n 的正整数数组 a，被称为“倍序数组”当且仅当对于任意 $1 \leq i \leq n-1$，都有 $a_i | a_i + 1$（其中” | “表示整除，即存在一个正整数 s 使得 $ai + 1 = s \times ai$）。请设计算法求出长度为 n 的倍序数组 a 的个数，且数组中每个元素满足 $1 \leq a_i \leq k$)，请描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度。

1. 状态设计

该数组中最大值为 k，长度为 n，我们考虑如何转化问题。易得该数组为递增序列，最大值一定出现在数组最后一位，由于最大值为 k，代表着该数组末位可能为 1,2...k 共 k 种情况，我们只需要将这 k 种情况各自的数组数求

和，即可得到解。因此问题转化为: 求数组长度为 n，末位为 k 的数组数。创建二维数组 dp[n][k]，其中 dp[i][j] 表示数组长度为 i，末位为 j(也为该数组最大值) 的数组个数。

2. 状态转移

考虑如何将原问题拆解为子问题。长度 i，末位 j 的数组数是由长度 i-1，末位 k(1<=k<=j) 的 j 个子问题叠加而来的。我们还应加上限制，当前的末位 j 与子问题的末位，即第 i-1 位数 k 应满足整除关系，即当 $j \mod k = 0$ 时该子问题才有效，形式化表述如下。

$dp[i][j] = \sum_{k=1}^{j} f(k)$

$$f(k) = \begin{cases} dp[i-1][k] & j \mod k \equiv 0 \\ 0 & j \mod k \neq 0 \end{cases}$$

3. 起始条件

末位为 1 时，任意长度只有一种 (1...1) 情况；长度为 1 时，任意末位只有一种 (k) 情况。因此 dp[1][1..k]=1, dp[1..n][1]=1。

4. 遍历顺序

根据递推公式可以看出，dp[i][j] 依赖于 dp[i-1][1..j]，因此可以**由小到大遍历长度 n 作为外层循环，由小到大遍历末位 k 作为内层循环**。

5. 伪代码

Algorithm 2 倍序数组问题

Input: n, k

Output: count

```
dp[1][1..k] ← 1, dp[1..n][1] ← 1, count ← 0
for i : 2 → n do
    for j : 2 → k do
        for m : 1 → j do
            if (j mod m) ≡ 0 then
                dp[i][j] ← dp[i][j] + dp[i − 1][m]
            end
        end
    end
end
for i : 1 → k do
    count ← count + dp[n][i]
end
```

6. 时间复杂度分析

动态规划用到三层循环，最外层循环 n 次，中层循环 k 次，内层循环依赖于中层循环，循环 j 次，不难计算得到动规时间复杂度为 $O(nk^2)$；计算结果循环 k 次，时间复杂度为 $O(k)$。综上本问题时间复杂度为 $O(nk^2)$。

3. 鲜花组合问题

花店共有 n 种不同颜色的花，其中第 i 种库存有 a_i 枝，现要从中选出 m 枝花组成一束鲜花。请设计算法计算有多少种组合一束花的方案，请描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度。(两种方案不同当且仅当存在一个花的种类 i，两种方案中第 i 种花的数量不同)

1. 状态设计

首先分析该问题，类似于背包问题，将花的种类看作是背包问题中的物品，而取的枝数看作背包容量。与背包问题不同的是，我们要取满 m 枝，而背包只需小于等于 m，而每种花不再代表取或不取，而是代表取几枝。我们创建二维数组 dp[n][m]，其种 dp[i][j] 表示在前 i 种花中取 j 枝的方案数，因此该问题的解为 dp[n][m]。

2. 状态转移

考虑将原问题拆解为子问题。dp[i][j] 为前 i 种花取 j 枝的方案数，而第 i 种花有 a_i 枝，即可能存在 $a_i + 1$ 种取法，设第 i 种花取 k 枝，则子问题为 dp[i-1][j-k]，即前 i-1 种花取 j-k 枝的方案数。我们将 a_i 种取法得到的子问题解相加，得到该层解。形式化表达如下。

$dp[i][j] \leftarrow \sum_{k=0}^{min(j,a_i)} dp[i-1][j-k]$

3. 起始条件

本问题起始条件设定如下。

1. $dp[1..n][0] = 1$，即无论几种花取 0 枝共有一种取法。
2. $dp[1][1..a_1] = 1$，即第一种花在不超出库存的情况下无论取几枝只有一种取法。
3. $dp[1][a_1 + 1...m] = 0$，即第一种花取超出库存的枝数无解。

4. 遍历顺序

根据递推公式看出，dp[i][j] 依赖于 dp[i-1][j-k]，在遍历到 i,j 时其左上角元素 (小于 i, 小于 j) 应都已经计算出来。因此可以**从小到大遍历种类 n 作为外循环，从小到大遍历枝数 m 作为内循环**。

5. 伪代码

Algorithm 3 鲜花组合问题

```
Input: a[1..n], m
Output: count
dp[1..n][0] ← 1
for i : 1 → m do
  | if a_i i then
  |   | dp[1][i] ← 1
  | else
  |   | dp[1][i] ← 0
  | end
end
for i : 2 → n do
  | for j : 0 → m do
  |   | dp[i][j] ← Σ_{k=0}^{min(j,a_i)} dp[i−1][j−k]
  | end
end
```

6. 时间复杂度分析

动态规划外层循环遍历鲜花种类 n 次，内层循环遍历取花数量 m 次，状态转移时，求和共需 $min(j, a_i)$ 次，此处情况不固定，假定每次求和 a_i 次，放大了时间复杂度，则动规最坏时间复杂度为 $O(m\sum_{i=1}^{n} a_i)$；求解结果 dp[n][m] 时间复杂度为 $O(1)$。因此该算法时间复杂度为 $O(m\sum_{i=1}^{n} a_i)$。

4. 叠塔问题

给定 n 块积木，编号为 1 到 n。第 i 块积木的重量为 w_i，硬度为 s_i，价值为 v_i。现要从中选择部分积木垂直摆成一座塔，要求每块积木满足如下条件:

若第 i 块积木在积木塔中，那么在其之上摆放的所有积木的重量之和不能超过第 i 块积木的硬度。

试设计算法求出满足上述条件的价值和最大的积木塔，输出摆放方案和最大价值和。请描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度。

0. 题目分析与转化

该问题类似于背包问题，可以将重量看作是背包容量，积木同物品，总重量和硬度间满足某些条件才允许放入。不同处在于该问题不只需要考虑要不要将积木放进背包，还要考虑该积木所在的位置，而输入的积木序列并不包含位置顺序信息，我们尝试转化问题。

假定当前放置了重量为 W 的积木，此时下面有 (w_i, s_i) 的 i 积木和 (w_j, s_j) 的 j 积木 (i 和 j 一定能想办法都放在下面)，讨论 i 和 j 的先后顺序。

设最优解为 i 在 j 的上面，考虑极端情况 i 和 j 调换顺序后无解，则有:

$s_j \geq W + w_i, s_i < W + w_j$

可得，$s_j + w_j > s_i + w_i$，即 $s_i + w_i$ 越小，越往高层放，反之越大，越往底层放。将积木根据 $s + w$ 进行升序排序，得到积木块的相对位置信息，即若积木块 $a_i, a_j (i < j)$ 都出现在积木塔中，一定有 a_i 在 a_j 的上面，由此将积木信息由无序变为有序。

1. 状态设计

经过上述转化，该问题演变成背包问题，创建二维数组 dp[n][m]，其中 dp[i][w] 表示前 i 块积木总重量为 w 时的最大价值，取 m 为所有积木块重量和 $\sum_{i=1}^{n} w_i$，则题目所求应该为 $max(dp[n][1...m])$

2. 状态转移

尝试将问题拆解为子问题，与 01 背包同理，dp[i][w] 要考虑选取积木 i 和不选取积木 i 两种情况，然而由于硬度和当前重量的关系，在选取积木 i 时存在约束，若积木 i 无法承重当前重量减自身重量则一定无法选择，形式化表示如下。

$$dp[i][w] = \begin{cases} dp[i-1][w] & w - w_i > s_i \\ max(dp[i-1][w], dp[i-1][w-w_i] + v_i) & w - w_i \le s_i \\ -\infty & w \le 0 \end{cases}$$

eg: w 小于 0 就越界了，只是提醒若 $w - w_i \le 0$ 时应当作 $-\infty$ 处理。

3. 起始条件

由于存在一些重量是无法达到的，我们令无法达到重量的相应位置为 $-\infty$，因此 $dp[1][w_1] = v_1$, 而 $dp[1][w \ne w_1] = -\infty$。总重量为 0 时价值一定为 0，即 $dp[1..n][0] = 0$。特别的 dp[1][0]=0。

4. 遍历顺序

根据递推公式看出，$dp[i][w]$ 依赖于 $dp[i-1][w-w_i]$。因此可以**从小到大遍历积木块 n 作为外循环，从小到大遍历总重量 m 作为内循环**。

5. 摆放方案

同 01 背包，开同样大小的二维数组 path[n][m]，遍历到 dp[i][w] 时若最优结果选择第 i 块则对应位置置 1，否则置 0。遍历结束后逆推 path 数组得到积木块的选择方案。

5. 伪代码

Algorithm 4 叠塔问题

Input: $w[1..n], s[1..n], v[1..n]$

Output: $plan[1..n], maxValue$

```
依据 s_i + w_i 升序对积木排序，用排序后的结果替换输入
dp[1][w_1] = v_1, dp[1][w ≠ w_1] = −∞, m ← ∑_{i=1}^{n} w_i
for i : 2 → n do
    for j : 1 → m do
        if j − w_i ≤ s_i then
            dp[i][j] ← max(dp[i−1][j], dp[i−1][j − w_i] + v_i)
            path[i][j] ← (dp[i−1][j] > dp[i−1][j − w_i] + v_i)?0 : 1
            // j − w_i < 0 时当负无穷处理，两个负无穷取最大值仍是负无穷
        else
            dp[i][j] ← dp[i−1][j]
            path[i][j] ← 0
        end
    end
end
maxValue ← max(dp[n][1..m]), w ← argmax_{k=1,2..m} dp[n][k], plan[n] ← path[n][w]
for i : n − 1 → 1 do
    if plan[i+1] ≡ 1 then
        w ← w − w_{i+1}
    end
    plan[i] ← path[i][w]
end
for i : 1 → n do
    plan[i] 若为 1，则代表挑选了对应的积木
end
```

6. 时间复杂度分析

将积木信息重新排序，所用时间复杂度为 $O(n\log n)$；动态规划中，外层循环遍历 n 次，内层循环遍历 m 次，状态转移时间复杂度为 $O(1)$，动规部分时间复杂度为 $O(n\sum_{i=1}^{n} w_i)$；输出堆积木方案需遍历 n 次，时间复杂度为 $O(n)$。由于 $\sum_{i=1}^{n} w_i > n > \log n$, 综上该算法时间复杂度为 $O(n\sum_{i=1}^{n} w_i)$。

5. 最大分值问题

给定一个包含 n 个整数的序列 $a_1, a_2, ..., a_n$，对其中任意一段连续区间 $a_i..a_j$，其分值为 $(\sum_{t=i}^{j} a_t) \mod p$。

现请你设计算法计算将其分为 k 段 (每段至少包含 1 个元素) 后分值和的最大值，请描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度。

例如，将 3, 4, 7, 2 分为 3 段，模数为 p = 10，则可将其分为 (3, 4),(7),(2) 这三段，其分值和为 $(3+4) \mod 10 + 7 \mod 10 + 2 \mod 10 = 16$。

1. 状态设计

分析题目是 n 个数分成 k 段的经典问题，我们创建二维数组 dp[k][n]，其中 dp[i][j] 代表前 j 个数分成 k 段的最大分值。题目所求解为 dp[k][n]。

2. 状态转移

尝试拆解子问题，计算 dp[i][j] 时，考虑第 j 个数 a_j，它一定会被分到一组，组内元素可能只有自己，可能是 $\{a_j, a_{j-1}\}$，$\{a_j, a_{j-1}, a_{j-2}\}$... 剩余的数分到 i-1 组中，若剩余的数量小于要分的组数，则置为负无穷不予考虑，递推公式形式化表达如下。

$$dp[i][j] = \begin{cases} -\infty & i > j \\ max(dp[i-1][j-k] + (\sum_{m=j-k+1}^{j} a_m) \mod p)(k = 1, 2..j) & i \le j \end{cases}$$

3. 起始条件

前任意个数分为一段时，其最大分值均为全部相加再求模，即 $dp[1][i] = (\sum_{j=1}^{i} a_j) \mod p$。

4. 遍历顺序

根据递推公式看出，$dp[i][j]$ 依赖于 $dp[i-1][j-k]$。因此可以**从小到大遍历分段数 k 作为外循环，从小到大遍历数组长度 n 作为内循环**。

5. 伪代码

Algorithm 5 最大分值问题

Input: $k, p, a[1..n]$

Output: $maxValue$

for $i : 1 \to n$ **do**

$\quad dp[1][i] \leftarrow (\sum_{j=1}^{i} a_j) \mod p$

end

for $i : 2 \to k$ **do**

$\quad$ **for** $j : 1 \to n$ **do**

$$dp[i][j] = \begin{cases} -\infty & i > j \\ max(dp[i-1][j-k] + (\sum_{m=j-k+1}^{j} a_m) \mod p)(k = 1, 2..j) & i \le j \end{cases}$$

$\quad$ **end**

end

6. 时间复杂度分析

算法中数组求和部分可优化为 $O(1)$ 复杂度。dp 数组初始化时间复杂度为 $O(n)$；动态规划中外层循环遍历 k 次，内层循环遍历 n 次，状态转移时需要遍历 j 次，动规部分总时间复杂度为 $O(n^2k)$。因此该算法总时间复杂度为 $O(n^2k)$。