计算机学院《算法设计与分析》
(2023 年秋季学期)

第二次作业参考答案

1 假日愉悦值问题 (20 分)

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 a_i 点愉悦值；
2. 去野外爬山。可以获得 b_i 点愉悦值；
3. 在家里学习。可以获得 c_i 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。

解：

1. 状态设计

设 $f[i][j]$ 表示在第 i 天进行的活动为第 j 种的条件下前 i 天进行活动后获得的最大愉悦值。

2. 状态转移

考虑到连续两天不能进行相同种类的活动，

1. 第 i 天选择去海边游泳，则前一天可以选择 2、3 两种活动，可能获得的最大预约值为 $a_i + \max\{f[i-1][2], f[i-1][3]\}$
2. 第 i 天选择去野外爬山，则前一天可以选择 1、3 两种活动，可能获得的最大预约值为 $b_i + \max\{f[i-1][1], f[i-1][3]\}$
3. 第 i 天选择在家里学习，则前一天可以选择 1、2 两种活动，可能获得的最大预约值为 $c_i + \max\{f[i-1][1], f[i-1][2]\}$

故有如下递推式

$$f[i][j] = \max_{j' \in \{1,2,3\} \cap j' \neq j} \{f[i-1][j']\} + \begin{cases} a_i & \text{if } j = 1 \\ b_i & \text{if } j = 2 \\ c_i & \text{if } j = 3 \end{cases}$$

3. 记录决策方案

为了记录决策方案，可以记 $pa[i][j]$ 表示 $f[i][j]$ 状态是由 $f[i-1][pa[i][j]]$ 状态转移而来的。那么可以利用 pa 逐次得到在第 n 天，第 $n-1$ 天，$\cdots$，第 1 天选择的最优活动。

4. 边界条件

第一天选择任何活动没有前置约束，故对应的愉悦值 $f[1][1] = a_1, f[1][2] = b_1, f[1][3] = c_1$，且没有对应 $pa[1][1], pa[1][2], pa[1][3]$ 无意义。

5. 目标状态

则情形下最大愉悦值对应的状态为 $f[n][k]$，对应的最优安排为 $pa[n][k]$ 及其前序安排，其中 $k = \arg\max_{i=1,2,3} f[n][i]$。

6. 时间复杂度分析

故总状态是 $O(n)$ 级别的 (注意到活动仅有 3 种)，每个状态的转移是 $O(1)$ 的，故总的时间复杂度为 $T(n) = O(n)$。伪代码如 Algorithm 1。

Algorithm 1 $fun(a[1..n], b[1..n], c[1..n])$

Input:

三个数组 $a[1..n], b[1..n], c[1..n]$ 表示每天完成每种活动的愉悦值。

Output:

n 天可获得的最大愉悦值，及其假日安排。

初始化数组 $f[1..n][1..3]$
$f[1][1] \leftarrow a_1, f[1][2] \leftarrow b_1, f[1][3] \leftarrow c_1$
for $i : 2 \to n$ **do**
  $pa[i][1] \leftarrow \arg\max_{k=2,3} f[i-1][k]$
  $f[i][1] \leftarrow a_i + \max_{k=2,3} f[i-1][k]$
  $pa[i][2] \leftarrow \arg\max_{k=1,3} f[i-1][k]$
  $f[i][2] \leftarrow b_i + \max_{k=1,3} f[i-1][k]$
  $pa[i][3] \leftarrow \arg\max_{k=1,2} f[i-1][k]$
  $f[i][3] \leftarrow c_i + \max_{k=1,2} f[i-1][k]$
end for
$plan \leftarrow \emptyset$
$now \leftarrow \arg\max_{k=1,2,3} f[n][k]$
for $i : n \to 1$ **do**
  add the i-th day do the kind-now activity in front of $plan$.
  $now \leftarrow pa[i][now]$
end for
return $\max_{k=1,2,3} f[n][k], plan$

2 倍序数组问题 (20 分)

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 $1 \le i \le n-1$，都有 $a_i | a_{i+1}$（其中”|“表示整除，即存在一个正整数 s 使得 $a_{i+1} = s \times a_i$）。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 $1 \le a_i \le k$，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。

例如，当 $k = 3, n = 2$ 时，满足条件的数组有 5 个，分别是 $\{1,1\}$，$\{1,2\}$，$\{1,3\}$，$\{2,2\}$ 和 $\{3,3\}$。

解：

1. 状态设计

设 $f[i][j]$ 表示长度为 i，以正整数 j 结尾的倍序数组数量。

2. 状态转移

对于当前状态 $f[i][j]$，考虑所有长度为 $i-1$、以正整数 s 结尾的倍序数组，若在其末尾加上数字 j 后仍为倍序数组，需要满足 $s|j$，即转移方程为：

$$f[i][j] = \sum_{s|j} f[i-1][s]$$

3. 边界条件与目标状态

起始条件为 $f[0][j] = 0 (1 \le j \le k)$。

最终答案为 $\sum_{j=1}^{k} f[n][j]$，即长度为 n 的倍序数组数量。由于倍序数组的末尾元素 j 可能是 $[1,k]$ 中的任意正整数，因此进行了求和。

4. 时间复杂度分析

每次转移中，若对当前末尾元素 j 进行因式分解、进而枚举 j 的所有因子 s，复杂度为 $O(\sqrt{k})$，状态数为 $n \times k$，因此复杂度为 $O(nk\sqrt{k})$。

5. 状态转移的改进

考虑到对于以因子 s 结尾的倍序数组，唯有后续元素为 s 的倍数时，才能继续构成倍序数组，因此可以通过枚举因子优化状态转移的计算。

基于上述思路，对于状态 $f[i-1][s]$，其本质上只对 $f[i][s]$、$f[i][2*s]$、......、$f[i][\lfloor k/s \rfloor * s]$ 产生贡献。因此在求解出 $f[i-1][s]$ 后，只需枚举小于等于 k 的 s 的倍数 z，把 $f[i-1][s]$ 加到 $f[i][z]$ 上。

固定数组长度，枚举所有的因子 s，每个因子的倍数数量为 k/s，因此这部分的复杂度为 $\sum_{s=1}^{k} k/s = k + k/2 + k/3 + ... + 1 = k \log k$。对数组长度的枚举复杂度为 $O(n)$，因此总复杂度为 $O(nk \log k)$。

伪代码见 Algorithm2。

Algorithm 2 $multipliedArray(n, k)$

Input:
数组长度 n，元素最大值 k。
Output:
满足要求的倍序数组数量。

初始化数组 $f[1..n][1..k]$
for $i : 0 \to n$ **do**
  for $j : 1 \to k$ **do**
    $f[i][j] \leftarrow 0$
  end for
end for
for $i : 1 \to n$ **do**
  for $j : 1 \to k$ **do**
    for $z : j \to k, z \leftarrow z + j$ **do**
      $f[i][z] \leftarrow f[i][z] + f[i-1][j]$
    end for
  end for
end for
return $\sum_{j=1}^{k} f[n][j]$

3 鲜花组合问题 (20 分)

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)

解：

1. 状态设计

该问题类似背包问题，考虑每种花为一个物品，最多可以购买 a_i 件，问题转化为总共购买 m 件的前提下，有多少种购买方案。

状态 $dp[i][j]$ 表示考虑前 i 种花，已经购买了 j 枝花的方案数。

2. 状态转移

在上一状态，即只考虑了前 $i-1$ 种花的情况下，枚举购买多少枝当前种类的花，进行转移。转移方程为：

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

3. 边界条件与目标状态

起始条件为 $dp[0][0] = 1$。

最终答案为 $dp[n][m]$，考虑全部 n 种花，恰好购买了 m 支的方案数量。

4. 时间复杂度分析

每次转移需要枚举当前种类的花购买的枝数，复杂度为 $O(m)$，状态数为 $n \times m$，因此复杂度为 $O(nm^2)$。

5. DP 转移优化

观察转移方程可以发现，其形式与前缀和相似，因此可以使用前缀和优化转移方程。

在转移之前计算

$$sum[j] = dp[i-1][j] + sum[j-1]$$

转移方程可以改写为

$$dp[i][j] = sum[j] - sum[j - \min(a_i, j) - 1]$$

每次转移复杂度为 $O(1)$，因此总复杂度降低为 $O(nm)$。

伪代码见 Algorithm3。

Algorithm 3 $flower(A[1..n])$

Input:

鲜花库存数量 $A[1..n]$，选择数量 m。

Output:

鲜花组合方案数量。

$dp[1..n][1..m]$
$sum[1..m]$
$dp[0][0] \leftarrow 1$
for $i : 1 \to n$ **do**
  $sum[0] \leftarrow dp[i-1][0]$
  for $j : 1 \to m$ **do**
    $sum[j] \leftarrow sum[j-1] + dp[i-1][j]$
  end for
  for $j : 0 \to m$ **do**
    $dp[i][j] \leftarrow sum[j] - sum[j - min(a[i], j) - 1]$
  end for
end for
return $dp[n][m]$

4 叠塔问题 (20 分)

给定 n 块积木，编号为 1 到 n。第 i 块积木的重量为 w_i(w_i 为整数)，硬度为 s_i，价值为 v_i。现要从中选择部分积木垂直摞成一座塔，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。

解:

1. 确定决策顺序

在确定 dp 状态前，我们先确定积木选择的顺序，思路如下:

假定目前已经放了重量为 W 的方块，考虑在底部放 i, j 两个积木，假定最优策略是 i 在 j 的上面，则有:

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

故有 $s_i + w_i < s_j + w_j$。如此可知在决策时按照 $s_i + w_i$ 从小到大的顺序决策正好能决策出一个自顶至底的最优策略。

2. 状态设计

故我们将原积木按照 $s_i + w_i$ 排序，设 $dp[i][W]$ 表示，当前已经考虑了前 i 个积木，搭建了重量为 W 的积木塔的最大价值。

3. 状态转移

状态转移类似背包问题:

考虑 $dp[i][W]$ 从哪些状态转移而来:

1. $W - w_i \leq s_i$，此时可以将 (s_i, w_i, v_i) 这块积木放置到塔的最底下，故可以选择状态 $dp[i-1][W - w_i]$ 对应的塔放在 (s_i, w_i, v_i) 这块积木之上。
2. $W - w_i > s_i$，此时不满足放置条件，只能考虑舍弃这块积木无法放置，即选择状态 $dp[i-1][W]$。

故转移方程为:

$$dp[i][W]=\begin{cases}dp[i-1][W] & W-w_i>s_i\\ \max\{dp[i-1][W],dp[i-1][W-w_i]+v_i\} & W-w_i\le s_i\end{cases}$$

故答案为 $\max_w dp[n][w]$。

4. 记录决策方案

为了求出满足条件的最大价值积木塔，还需要记录哪些积木块被选择了。

注意到仅有在 $W-w_i\le s_i$ 的条件下，且选择了决策 $dp[i-1][W-w_i]+v_i$ 才会选择积木块 (s_i,w_i,v_i)。

故记录 $elect[i][w]$ 表示 $dp[i][w]$ 状态时有哪些积木块被使用了。

则在转移选了决策 $dp[i-1][W-w_i]+v_i$ 时有 $elect[i][W]=elect[i-1][W-w_i]\cup\{(s_i,w_i,v_i)\}$，否则 $elect[i][W]$ 与 $elect[i-1][W]$ 相同。

5. 边界条件

对于没有考虑任何积木时，即 $dp[0][0\sim\sum_i w_i]$，获得积木塔的价值都是 0，且 $elect[0][0\sim\sum_i w_i]$ 均为 $\emptyset$。

6. 目标状态

类似背包问题，最优方案为 $elect[n][mw]$，对应的最大价值为 $dp[n][mw]$，其中 $mw=\arg\max_w dp[n][w]$。

7. 时间复杂度分析

考虑至少有 $O(n\times\sum_i w_i)$ 的状态，每个状态需要 $O(1)$ 的时间转移，排序时间复杂度为 $O(n\log n)$，因为 $\sum_i w_i>n$，故总的复杂度为 $O(n\times\sum_i w_i)$，伪代码如 Algorithm 4。

Algorithm 4 $tower(n,\{w_n\},\{s_n\},\{v_n\})$

Input:

n 块积木，第 i 块积木的重量为 w_i，硬度为 s_i，价值为 v_i。

Output:

积木塔最大价值和以及对应的摆放方案。

初始化 $dp[0..n][0..\sum_i w_i]$ 全部为 0，$elect[0..n][0..\sum_i w_i]$ 全部为空集
sort (w_i,s_i,v_i) tuples by s_i+w_i in ascending order
$sum\leftarrow 0$
for $i:1\to n$ **do**
  for $w:w_i\to sum$ **do**
    if $w-w_i\le s_i\cap dp[i-1][w-w_i]+v_i>dp[i-1][w]$ **then**
      $dp[i][w]\leftarrow dp[i-1][w-w_i]+v_i$
      $elect[i][w]\leftarrow\{(s_i,w_i,v_i)\}\cup elect[i-1][w-w_i]$
    else
      $dp[i][w]\leftarrow dp[i-1][w]$
      $elect[i][w]\leftarrow elect[i-1][w]$
    end if
  end for
  $sum\leftarrow sum+w_i$
end for
$mw\leftarrow\arg\max_w dp[n][w]$
return $dp[n][mw],elect[n][w]$

---

5 最大分值问题 (20 分)

给定一个包含 n 个整数的序列 $a_1,a_2,\ldots,a_n$，对其中任意一段连续区间 $a_i..a_j$，其分值为

$$(\sum_{t=i}^{j}a_t)\%p$$

符号 % 表示取余运算符，可以认为 p 远小于 n。

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 $(3+4)\%10+7\%10+2\%10=16$。

解：

1. 状态设计

记 $val(i,j)=(\sum_{t=i}^{j}a_t)\%p$，令 $f[i][j]$ 表示将前 i 个数分为 j 段可获得的最大分值。

2. 状态转移

枚举第 j 段的起始位置 $t+1$，若最后一段是由 $a[t+1..i]$ 构成，则这一段对答案的贡献为 $val(t+1,i)$，而 $a[1..t]$ 应被分为 $j-1$ 段，其最大分值为 $f[t][j-1]$。

故递归式如下：

$$f[i][j]=\max_{t=0}^{i-1}\{f[t][j-1]+val(t+1,i)\}$$

3. 边界条件与目标状态

初始化仅需将所有的 $f[i][j]$ 置为 0。

目标状态为 $f[n][k]$。

4. 时间复杂度分析

其状态数为 $O(nk)$，每次转移的复杂度为 $O(n)$，故总的时间复杂度为 $O(n^2k)$。

$\cdots\cdots$（此为暴力算法总计 10 分）

5. 状态转移的改进

考虑如何进行优化，通过观察可发现，上述公式中，$val(t+1,j)$ 可能的取值只有 p 种。这意味着我们可以将 $f[t][j-1]$ 按照其对应的 $val(t+1,i)$ 的不同取值进行分组，对每一组预统计出其最大值，之后仅需花费 $O(p)$ 的时间进行转移。

具体来说，记 $sum[i]=\sum_{t=1}^{i}a_t$。则 $val(t+1,i)$ 可改写为

$$val(t+1,i)=(sum[i]-sum[t])\%p$$

由此可看出，在计算状态 $f[i][j]$ 时，i 已经确定，那么 $val(t+1,i)$ 的取值仅和 $sum[t]\%p$ 相关。因此，可将所有 $f[t][j-1]$ 根据 $sum[t]\%p$ 分组，并统计每组的最大值 (记 $g[x][j-1]$ 表示所有满足 $sum[t]\%p=x$ 的状态 $f[t][j-1]$ 的最大值)。之后需将每一组的最大值 $g[x][j-1]$ 加上这一组对应的 val 值。根据公式

$$val(t+1,i)=(sum[i]-sum[t])\%p$$

可知，对所有满足 $sum[t]\%p=x$ 的 t，其对应的 $val(t+1,i)$ 均为 $(sum[i]-x)\%p$。

根据上述分析，递归式可写为：

$$f[i][j]=\max_{x=0}^{p-1}\{g[x][j-1]+(sum[i]-x)\%p\}$$

其中，

$$g[x][j-1]=\max_{t<i,sum[t]=x}f[t][j-1]$$

6. 改进后的时间复杂度分析

其状态数为 $O(nk)$，每次转移的复杂度为 $O(p)$，故总的时间复杂度为 $O(npk)$。

算法伪代码如 Algorithm 5 所示。

Algorithm 5 $Feasible(a[1..n], k, p)$

初始化数组 $sum[0..n], f[1..n][1..k], g[0..p-1][0..k]$ 全部元素为 0
for $i \leftarrow 1$ **to** n **do**
  $sum[i] \leftarrow sum[i-1] + a[i]$
end for
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** k **do**
    for $t \leftarrow 0$ **to** $p-1$ **do**
      $f[i][j] \leftarrow \max\{f[i][j], g[t][j-1] + (sum[i]-t)\%p\}$
      $g[sum[i]\%p][j] \leftarrow \max\{g[sum[i]\%p][j], f[i][j]\}$
    end for
  end for
end for
return $f[n][k]$