高等理工学院《算法设计与分析》
(2021 年秋季学期)

第二次作业参考答案

1 假日愉悦值问题 (20 分)

Ange 开始计划她的 N 天长假，第 i 天她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。

由于她希望自己的假日丰富多彩，她并不希望连续两天（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 Ange 要求的情形下所获得的愉悦值的和最大，并分析该算法的时间复杂度。

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 $a_i + \max\{f[i-1][1], f[i-1][2]\}$

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。

$f[1][1] \leftarrow a_1, f[1][2] \leftarrow b_1, f[1][3] \leftarrow c_i$
for $i : 2 \rightarrow n$ **do**
  $pa[i][1] \leftarrow \arg\max_{k=2,3} f[i-1][k]$
  $f[i][1] \leftarrow \max_{k=2,3} f[i-1][k]$
  $pa[i][2] \leftarrow \arg\max_{k=1,3} f[i-1][k]$
  $f[i][2] \leftarrow \max_{k=1,3} f[i-1][k]$
  $pa[i][3] \leftarrow \arg\max_{k=1,2} f[i-1][k]$
  $f[i][3] \leftarrow \max_{k=1,2} f[i-1][k]$
end for
$plan \leftarrow \emptyset$
$now \leftarrow \arg\max_{k=1,2,3} f[n][k]$
for $i : n \rightarrow 1$ **do**
  add the i-th day do the kind-now activity in front of $plan$.
  $now \leftarrow pa[i][now]$
end for

2 小跳蛙问题 (20 分)

给定 n 块石头，依次编号为 1 到 n，第 i 块石头的高度是 h_i，青蛙最远跳跃距离 k。

现有一只小跳蛙在第 1 块石头上，它重复以下操作，直到它到达第 n 块石头：

若它当前在第 i 块石头上，则可跳到第 $j(i+1 \le j \le \min(i+k, n))$ 块石头上，耗费的体力为 $|h_i - h_j|$。

试设计算法求它最少耗费多少体力可以到达第 n 块石头，并分析该算法的时间复杂度。

解：

1. 状态设计

记 $f[i]$ 表示小跳蛙跳到第 i 号石头上时的最小代价。

2. 状态转移

因为只可以从第 $i-1$ 到 $i-k$ 块石头跳到第 i 块石头，而跳到第 i 块石头的代价为上一块和第 i 块的高度差

故转移描述如下:

$$f[i] = \min_{j=i-k}^{i-1} \{f[j] + |h_i - h_j|\}$$

3. 边界条件

临界状态即 $f[1] = 0$。因为小跳蛙初始在第 1 块石头上。

4. 目标状态

由状态含义可知，到达第 n 块石头的最小代价为 $f[n]$。

5. 时间复杂度分析

故总状态是 $O(n)$ 级别的，而每个状态的转移是 $O(k)$ 时间的。故总的时间复杂度为 $O(nk)$，参考伪代码如 Algorithm 2。

Algorithm 2 $jump(\{h_n\})$

Input:

1 到 n 每块石头的高度 $\{h_n\}$。

Output:

最少需要耗费多少体力。

```
f[1] ← 0
for i : 2 → n do
    f[i] ← ∞
    for j : max{i − k, 1} → i − 1 do
        f[i] ← min{f[i], f[j] + |h_i + h_j|}
    end for
end for
return f[n]
```

3 好序列问题 (20 分)

一个长度为 n 的序列 $A = [a_1, \cdots, a_n]$ 满足以下任一条件时，我们称序列 A 为好序列：

1. 序列第一个元素等于序列长度减 1，并且 a_1 不为 0，即 $a_1 = n - 1$ 并且 $a_1 > 0$；
2. 序列 A 可以拆分为 $A_1 = [a_1, \cdots, a_k], A_2 = [a_{k+1}, \cdots, a_n](1 \le k < n)$，并且 A_1 和 A_2 都是好序列。

注：一个序列的子序列是指从原始序列中去除某些元素但不破坏剩余元素相对顺序而产生的新序列。

例如 $[3, 1, -5, 6], [1, 5]$ 是好序列（满足规则 1），$[3, 1, -5, 6, 1, 5]$ 是好序列（满足规则 2），$[0]$, $[1, 2, 3]$ 则不是好序列。

给定一个长度为 n 的序列 A，请设计一个算法，计算该序列有多少**子序列**是好序列，并分析该算法的时间复杂度。

例如，对序列 $A = [1, 1, 1, 1]$，其中有 7 个子序列为好序列，分别为：$[a_1, a_2, a_3, a_4]$, $[a_1, a_2]$, $[a_1, a_3]$, $[a_1, a_4]$, $[a_2, a_3]$, $[a_2, a_4]$, $[a_3, a_4]$ 都是好序列。

解：

1. 状态设计

一个序列是否满足规则 1 可以通过第一个元素判断，与之后的元素无关，因此可以逆向进行动态规划，状态 $dp[i]$ 表示考虑 $[i, n]$ 区间内，必须包含元素 a_i，且是好序列的子序列的个数。

2. 状态转移

序列 $[i, n]$ 的答案计算通过枚举切分点，使用规则 2 拆分为两部分，分别是 $[i, k]$ 和 $[k+1, n]$，需满足 $[i, k]$ 的子序列满足规则 1，$[k+1, n]$ 的子序列满足规则 1 或 2，$[k+1, n]$ 的答案为 $dp[k+1]$。因为必须包含 a_i，所以 $[i, k]$ 的答案为 $[i+1, k]$ 中选择 a_i 个元素，即 $C_{k-i}^{a_i}$，使得 $[i, k]$ 的子序列满足规则 1。

转移方程为：

$$dp[i] = \sum_{k=a_i+1}^{n} C_{k-i}^{a_i} \times dp[k+1]$$

3. 边界条件

假设 $dp[n+1]$ 代表空序列，将该序列附加在任意好序列末尾并不会产生影响，因此边界条件为 $dp[n+1] = 1$。

4. 目标状态

最终答案可以从任意元素开始，因此 DP 数组求和为答案，$ans = \sum_{i=1}^{n} dp[i]$。

5. 时间复杂度分析

每次转移需要遍历切分点，复杂度为 $O(n)$，状态数量为 n，因此总复杂度为 $O(n^2)$。参考伪代码如 Algorithm 3。

Algorithm 3 $good\_seq(n, \{a_n\})$

Input:

序列长度 n，长度为 n 的序列 A。

Output:

好序列个数。

```
dp[n+1] ← 1
sum ← 0
for i : n → 1 do
    dp[i] ← 0
    for j : i + a_i → n do
        dp[i] ← dp[i] + C(a_i, j−i) × dp[j+1]
    end for
    sum ← sum + dp[i]
end for
return sum
```

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，那么在其之上的积木的重量和不能超过其硬度。
试设计算法求出满足上述条件的价值和最大的积木塔，并分析该算法的时间复杂度。

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

故我们想将原积木按照 $s_i + w_i$ 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

则转移类似背包问题:

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:

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

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 $W - w_i \leq s_i$ 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 $elect[n][mw]$，对应的最大高度为 $dp[n][mw]$，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:

价值和最大的积木塔选择了哪些积木块。

```
sort (w_i, s_i, v_i) tuples by s_i + w_i in ascending order
sum ← 0
for i : 1 → n do
    for w : 0 → sum do
        if w − w_i ≤ s_i ∩ dp[i − 1][w − w_i] + v_i > dp[i − 1][w] then
            dp[i][w] ← dp[i − 1][w − w_i] + v_i
            elect[i][w] ← {(s_i, w_i, v_i)} ∪ elect[i − 1][w − w_i]
        else
            dp[i][w] ← dp[i − 1][w]
            elect[i][w] ← elect[i − 1][w]
        end if
    end for
    sum ← sum + w_i
end for
mw ← arg max_w dp[n][w]
return elect[n][w]
```

5 戳气球问题 (20 分)

给定 n 个气球，第 i 个气球上都标有数字 a_i。现在要求你按照一定顺序戳破第 $2\sim n-1$ 个气球。若戳破气球 i，则可以获得 $a_{left}\times a_i\times a_{right}$ 的得分，这里 $left$ 和 $right$ 表示当前与第 i 个气球相邻的左右两个气球的序号。（注意，当你戳破了气球 i 之后，气球 $left$ 和 $right$ 就变成了相邻的气球。）请设计一个高效的算法，求出所能获得的最大得分，并分析该算法的时间复杂度。

解：

时间复杂度为 $O(n^3)$ 的算法可得满分，复杂度高于 $O(n^3)$ 的算法也可得分。

1. 状态设计

设计状态 $dp[i][j]$ 表示开区间 (i,j) 内所有气球都戳破可以得到的最大得分。

2. 状态转移

考虑开区间 (i,j) 中最后戳破的气球编号为 k，戳破该区间中所有气球的得分分为三部分：

1. 戳破开区间 (i,k) 内所有气球的得分，$dp[i][k]$；
2. 戳破开区间 (k,j) 内所有气球的得分，$dp[k][j]$；
3. 戳破气球 k 的得分 $a_i\times a_k\times a_j$。

转移方程如下

$$dp[i][j]=\max_{k\in(i,j)}\{dp[i][k]+dp[k][j]+a_i\times a_k\times a_j\}$$

3. 边界条件

若开区间长度为 0，不包含任何气球，则得分为 0，即 $dp[i][j] = 0\ (j - i \leq 1)$

4. 目标状态

目标状态为 $dp[1][n]$，即戳破 $(1, n)$ 内所有气球的最高得分。

5. 时间复杂度分析

总的状态数为 $O(n^2)$，而每次转移需要遍历上一次戳破的气球，假设当前状态为 $[i, j]$，则需要遍历 $j - i$ 次，故总的时间复杂度为 $O(n^3)$。伪代码如 Algorithm 5。

Algorithm 5 $balloon(n, \{a_n\})$

Input:

气球数量 n，第 i 个气球标号为 a_i。

Output:

戳破开区间 $(1, n)$ 所有气球的最大得分。

```
for len : 3 → n do
    for i : 1 → n − len + 1 do
        j ← i + len − 1
        dp[i][j] ← −∞
        for k : i + 1 → j − 1 do
            l_ans ← 0
            r_ans ← 0
            if k − i > 1 then
                l_ans ← dp[i][k]
            end if
            if j − k > 1 then
                r_ans ← dp[k][j]
            end if
            if dp[i][j] < l_ans + r_ans + a_i × a_j × a_k then
                dp[i][j] ← l_ans + r_ans + a_i × a_j × a_k
            end if
        end for
    end for
end for
return dp[1][n]
```