计算机学院《算法设计与分析》
(2022 年秋季学期)

第二次作业参考答案

1 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，写出伪代码并分析算法的时间复杂度。

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 1。

Algorithm 1 $jump(\{h_n\})$

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]
```

2 二进制串变换问题 (20 分)

给定两个长度均为 n 的仅由 0 和 1 组成的字符串 a 和 b，你可以对串 a 进行如下操作：

1. 对任意 $i,j(1 \le i,j \le n)$，交换 a_i 和 a_j，操作代价为 $|i-j|$；
2. 对任意 $i(1 \le i \le n)$，取反 a_i，操作代价为 1；

请你设计算法计算将串 a 变为串 b 所需的最小代价（只能对串 a 进行操作），写出伪代码并分析算法的时间复杂度。

解：

1. 决策选择

很容易发现，仅在存在连续两位需要取反时才会选择交换操作，其他情况下使用取反操作即可。

2. 状态设计

定义状态 $C[i]$ 表示将串 a 的前 i 位变成 b 所需的最小代价

3. 状态转移

采用如下转移：

$$C[i] = \begin{cases} C[i-1] & \text{若} a[i] = b[i] \\ C[i-1]+1 & \text{若} a[i]! = b[i] \text{且} a[i-1] = b[i-1] \\ C[i-1]+1 & \text{若} a[i]! = b[i] \text{且} a[i-1]! = b[i-1] \text{且} b[i] = b[i-1] \\ C[i-2]+1 & \text{若} a[i]! = b[i] \text{且} a[i-1]! = b[i-1] \text{且} b[i]! = b[i-1] \end{cases}$$

其边界条件为 $C[0]=0$，若 $a[1]=b[1]$，则 $C[1]=0$，否则 $C[1]=1$

4. 时间复杂度分析

状态数为 $O(n)$ 级别，每个状态的转移是 $O(1)$ 级别，故总时间复杂度为 $O(n)$。

算法伪代码如 Algorithm 2 所示。

Algorithm 2 $binary(n, a[1..n], b[1..n])$

Input:

两个长度均为 n 的 01 字符串 $a[1..n], b[1..n]$。

Output:

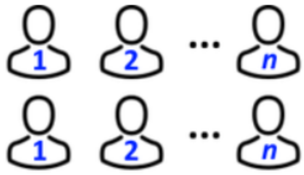
$C[0] \leftarrow 0$
if $a[1] = b[1]$ **then**
  $C[1] \leftarrow 0$
else
  $C[1] \leftarrow 1$
end if
for $i \leftarrow 1$ **to** n **do**
  if $a[i] = b[i]$ **then**
    $C[i] \leftarrow C[i-1]$
  else if $(a[i]! = b[i]$**and**$a[i-1] = b[i-1])$**or**$(a[i]! = b[i]$**and**$a[i-1]! = b[i-1]$**and**$b[i] = b[i-1])$ **then**
    $C[i] \leftarrow C[i-1]+1$
  else if $a[i]! = b[i]$**and**$a[i-1]! = b[i-1]$**and**$b[i]! = b[i-1]$ **then**
    $C[i] \leftarrow C[i-2]+1$
  end if
end for
return $C[n]$

---

3 球队组建问题 (20 分)

有 $2n$ 个学生分为两排，每排有 n 个人，由左至右分别编号为 $1, 2, \cdots, n$，如图所示。现在请你在这两排学生中挑选出一些学生组成一支球队，挑选出的学生编号必须是严格递增的（编号相同的两名学生最多只能取其中一个）。此外，为避免球队中的队员都来自同一排，不能同时

选择同一排相邻的两名学生（例如，若选择第一排的 5 号同学，就不能再选择第一排的 4 号和 6 号同学）。组建队伍的总人数没有限制。



给出同学们的身高数据 $h_{i,j}$，$h_{1,k}(1 \le k \le n)$ 表示第一排同学的身高，$h_{2,k}(1 \le k \le n)$ 表示第二排同学的身高。请你设计算法使组建成的球队中队员的身高之和最大，写出伪代码并分析算法的时间复杂度。

解：

这是一道非常标准的二维动态规划题目。定义状态 $MH[i][0..2]$，表示考虑编号小于等于 i 的队员时可以取得的最大身高。其中，$MH[i][0]$ 表示编号为 i 的两名队员均未被选进球队；$MH[i][1]$ 表示在第一列中编号为 i 的队员被选入球队；$MH[i][2]$ 表示在第二列中编号为 i 的队员被选入球队。

很容易得出如下所述递归式：

$MH[i,0] = \max\{MH[i-1,1], MH[i-1,2]\}$ 请思考这里为什么不需要考虑$MH[i-1,0]$

$MH[i,1] = \max\{MH[i-1,0] + h_{1,i}, MH[i-1,2] + h_{1,i}\}$

$MH[i,2] = \max\{MH[i-1,0] + h_{2,i}, MH[i-1,1] + h_{2,i}\}$

边界条件为 $MF[0,0] = MH[0,1] = MH[0,2] = 0$，能够组建成的球队中队员身高之和的最大值即为 $MH[n,0], MH[n,1], MH[n,2]$ 中的最大值。

事实上，可以通过进一步修改该状态定义得到更加精炼的递归式。定义状态 $MH[i][1]$ 表示最后一名队员是从第一排中选出，且编号小于等于 i；$MH[i][2]$ 表示最后一名队员是从第二排中选出，且编号小于等于 i。这样，原本的三种递归情况可减少为如下两种：

$MH[i,1] = \max\{MH[i-1,1], MH[i-1,2] + h_{1,i}\}$

$MH[i,2] = \max\{MH[i-1,1] + h_{2,i}, MH[i-1,2]\}$

该算法的伪代码如 Algorithm 3 所示。

时间复杂度分析：该算法共有 $O(n)$ 种状态，每种状态仅需要 $O(1)$ 的时间进行转移，因此总的时间复杂度为 $O(n)$。

4 括号匹配问题 (20 分)

定义合法的括号串如下：

1. 空串是合法的括号串；
2. 若串 s 是合法的，则 （s） 和 ［s］ 也是合法的；
3. 若串 a, b 均是合法的，则 ab 也是合法的。

现在给定由‘[’,‘]’和‘(’,‘)’构成的字符串，请你设计算法计算该串中合法的子序列的最大长度，写出伪代码并分析算法的时间复杂度。例如字符串“([(])])”，最长的合法子序列“([()])”长度为 6。

解

1. 状态设计

定义 $D[i,j]$ 表示子串 $S[i..j]$ 的最长合法子序列的长度。

2. 状态转移

状态转移递归式如下：

$$D[i,j] = \max \begin{cases} D[i+1, j-1] + 2 & \text{若S[i]='(' 且 S[j]=')' 或 S[i]='[' 且 S[j]=']'} \\ D[i,k] + D[k+1, j] & \text{对所有} k = \{i, i+1, \cdots, j-1\} \end{cases}$$

和矩阵链乘问题类似，我们按照 |j-i| 递增的顺序来进行转移。

其边界条件为 $D[i,i] = 0$，其中 $i = \{1, 2, .., n\}$

Algorithm 3 $MaxHeight(h_{1..2,1..n})$

Input:

同学们的身高数组 $h_{1..2,1..n}$

Output:

最大身高和、队员选择方案数组

新建二维数组 $MH[0..n, 1..2], REC[0..n, 1..2]$
$MH[0,1], MH[0,2] \leftarrow 0, 0$
$REC[0,1], REC[0,2] \leftarrow \{\}, \{\}$
for $i \leftarrow 1$ **to** n **do**
  $MH[i,1] \leftarrow \max\{MH[i-1,1], MH[i-1,2] + h_{1,i}\}$
  if $MH[i-1,1] > MH[i-1,2] + h_{1,i}$ **then**
    $REC[i,1] \leftarrow REC[i-1,1]$
  else
    $REC[i,1] \leftarrow REC[i-1,2] \cup \{(i,1)\}$
  end if
  $MH[i,2] \leftarrow \max\{MH[i-1,1] + h_{2,i}, MH[i-1,2]\}$
  if $MH[i-1,1] + h_{2,i} > MH[i-1,2]]$ **then**
    $REC[i,2] \leftarrow REC[i-1,1] \cup \{(i,2)\}$
  else
    $REC[i,2] \leftarrow REC[i-1,2]$
  end if
end for
if $MH[n,1] > MH[n,2]$ **then**
  return $MH[n,1], REC[n,1]$
else
  return $MH[n,2], REC[n,2]$
end if

3. 时间复杂度分析

状态数为 $O(n^2)$ 级别，每个状态的转移要考虑之前的 $O(n)$ 级别个状态，故总时间复杂度为 $O(n^3)$。

算法伪代码如 Algorithm 4 所示。

Algorithm 4 $MaxLength(S[1..n])$

Input:

字符串 $S[1..n]$。

Output:

最长合法子序列长度

新建二维数组 $D[1..n, 1..n]$
for $i \leftarrow 1$ *to* n **do**
    $D[i,i] \leftarrow 0$
end for
for $l \leftarrow 2$ *to* n **do**
    for $i \leftarrow 1$ *to* $n - l + 1$ **do**
        $j \leftarrow i + l - 1$
        if $S[i] =' ('$ **and** $S[j] =')'$ **or** $S[i] =' ['$ **and** $S[j] =']'$ **then**
            if $l = 2$ **then**
                $D[i,j] \leftarrow 2$
            else
                $D[i,j] \leftarrow D[i+1, j-1] + 2$
            end if
        end if
        for $k \leftarrow i$ *to* $j - 1$ **do**
            $D[i,j] \leftarrow \max(D[i,j], D[i,k] + D[k+1,j])$
        end for
    end for
end for
return $D[1,n]$

5 箱子问题 (20 分)

给定 n 种箱子 $a_1, \cdots, a_n$，第 i 种箱子 a_i 可表示为 $h_i \times w_i \times d_i$ 的长方体。请用这些箱子搭建一个尽可能高的塔：如果一个箱子 A 要水平的放在另一个箱子 B 上，那么要求箱子 A 底面的长和宽都严格小于箱子 B。可以任意旋转箱子，每种箱子可以用任意次。

设计一个算法求出一个建塔方案使得该塔的高度最高，写出伪代码并分析算法的时间复杂度。

例如给定 $n = 1$ 种箱子，其可表示为 $3 \times 4 \times 5$ 的长方体，建塔方案如下：

1. 最底层，放置一个以 4×5 为底面的箱子，该箱子高度为 3;
2. 第二层，放置一个以 3×4 为底面的箱子，该箱子高度为 5。

此时该塔高度最高，为 $3 + 5 = 8$。

如下的建塔方案不合法：

1. 最底层，放置一个以 4×5 为底面的箱子，该箱子高度为 3;
2. 第二层，放置一个以 3×5 为底面的箱子，此时底面的长为 5，不满足条件。

解：

1. 确定决策顺序

此题可以看做是最长上升子序列问题的一个变形，先要注意到如下两个事实：

1. 每种箱子最多使用 3 次，分别是 $h \times w \times d, w \times h \times d, d \times h \times w$ 即以 $w \times d, h \times d, h \times w$ 作为底面各尝试一次。
2. 只有底面积比当前箱子大的箱子，才可能允许在其之上放置当前箱子。

2. 状态设计

故我们，可以先将箱子个数按照三种底面的情形扩充至 $3n$ 的情形，再按照底面积大小从小到大排序为 $b_1, \cdots b_{3n}$(b_i 箱子对应的长、宽、高分别记做 d'_i, w'_i, h'_i)。$dp[i]$ 表示最后一个选择的箱子为 b_i 的最高塔高。

3. 状态转移

采用如下转移:

$$dp[i] = \max \begin{cases} h'_i \\ dp[j] + h'_i \quad (d'_j > d'_i \cap w'_j > w'_i) \end{cases}$$

这是因为在考虑阶段 i 时，可以枚举之前所有考虑过的阶段 $j(j < i)$，只要其满足摆放条件就可以转移给状态 i。

而状态 i 的边界条件就是仅放了 b_i 这一个箱子的情形。

4. 时间复杂度分析

故状态数为 $O(n)$ 级别，每个状态的转移要考虑其之前的所有状态，故时间复杂度为 $T(n) = \sum_{i=1}^{3n} O(i) = O(n^2)$。

算法伪代码如 Algorithm 5 所示。

Algorithm 5 $boxing(n, h[1..n], w[1..n], d[1..n])$

Input:
　n 种箱子，第 i 种的规格为 $h_i \times w_i \times d_i$

Output:
　使得塔最高的建塔方案

将 $w_i \times d_i, h_i \times d_i, h_i \times w_i$ 分别作为底面扩充成数组 $b[1..3n]$（并约束每个箱子的长大于等于宽）
按照底面积从大到小对 $b[1..3n]$ 数组排序
for $i : 1 \to 3 \times n$ **do**
　$dp[i] \leftarrow h'_i$
　$rule[i] = 0$
　for $j : 1 \to i - 1$ **do**
　　if $d'_j > d'_i \cap w'_j > w'_i \cap dp[j] + h'_i > dp[i]$ **then**
　　　$dp[i] \leftarrow dp[j] + h'_i$
　　　$rule[i] = j$
　　end if
　end for
end for
$mi \leftarrow \arg\max_i dp[i]$
$plan \leftarrow \varphi$
$maxheight \leftarrow dp[mi]$
for $mi \neq 0$ **do**
　add mi into $plan$
end for
return $plan, maxheight$
