动态规划算法作业

左逸龙 1120231863 07112303

1 整数线性规划问题

这个问题属于**完全背包问题**，其中各物品重量为 a_i，价值为 c_i，背包容量为 b，每件物品装载的件数 $x_i = 0, 1, 2, \cdots$。令状态 $f[i][j]$ 表示选择前 i 个物品且背包容量为 j 时的最优方案，即所选物品的最大价值，我们可以设计如下状态转移方程：

$$f[i][j] = max(f[i-1][j-ka_i] + kc_i), \quad k = 0, 1, 2, \cdots, \lfloor \frac{j}{a_i} \rfloor$$

我们可以进一步优化该状态转移方程：

$$\begin{aligned}
f[i][j] &= \max(f[i-1][j], f[i-1][j-ka_i] + kc_i), \quad k = 1, 2, \cdots, \lfloor \tfrac{j}{a_i} \rfloor \\
&= \max(f[i-1][j], \max(f[i-1][j-ka_i] + kc_i)) \qquad (1) \\
f[i][j-a_i] &= \max(f[i-1][j-ka_i] + kc_i), \quad k = 1, 2, \cdots, \lfloor \tfrac{j}{a_i} \rfloor \qquad (2)
\end{aligned}$$

将 (2) 代入 (1) 中，可以得到状态转移方程：

$$f[i][j] = \begin{cases} f[i-1][j] & j < a_i \\ \max(f[i-1][j], f[i][j-a_i] + c_i) & j \geq a_i \end{cases}$$

对此，我的理解是若 $f[i][j-a_i]$ 存在，则该状态已经计算了除去 1 个物品 i 时的最优解，在计算 $f[i][j]$ 的时候就不用再取多于 1 个物品了。

据此，给出算法伪代码如下：

算法 1: 动态规划算法解决完全背包问题

输入: 序列 a_i(重量)，序列 c_i(价值)，b(背包容量)

输出: 规划目标 $\max c_1x_1 + c_2x_2 + \cdots + c_nx_n$，其中 x_i 为非负整数

```
初始化 dp[i][0] ← 0, dp[0][j] ← 0，其中 i = 0 ∼ n, j = 0 ∼ b// 边界
for i = 1 to n do // 每一个物品
    for j = 1 to b do // 背包容积
        if j ≥ a_i then dp[i][j] ← max(dp[i−1][j], dp[i][j−a_i] + c_i);
        else dp[i][j] ← dp[i−1][j];
    end
end
return dp[n][b]
```

算法分析： 双层循环，时间复杂度为 $O(bn)$。

2 数字三角形问题

我们假设数字三角形存放于下三角矩阵 dp 中，令状态 $dp[i][j]$ 表示递推至位置 (i,j) 时的最大数字和，于是可以得到如下状态转移方程：

$$dp[i][j] = dp[i][j] + \max(dp[i+1][j], dp[i+1][j+1])$$

据此给出算法伪代码如下：

算法 2: 动态规划算法解决数字三角形问题

输入: 下三角矩阵 dp

输出: 从三角形顶至底的路径经过的数字和的最大值

```
for i = n − 1 to 1 do // 第 i 层
    for j = 1 to i do
        dp[i][j] ← max(dp[i+1][j], dp[i+1][j+1])
    end
end
return dp[1][1]
```

算法分析： 双层循环，时间复杂度为 $O(n^2)$。

3 游艇出租问题

将租金费用保存在一上三角矩阵 dp 当中，其中 $dp[i][j]$ 表示从第 i 个站到第 j 个站的费用；特别的，$dp[i][i]=0$。

一种常见的做法是像**矩阵连乘问题**那样做，令状态 $dp[i][j]$ 表示从第 i 个站到第 j 个站的最少费用，则可以设计如下状态转移方程：

$$dp[i][j] = \min(dp[i][k] + dp[k][j]), \quad k = i, i+1 \cdots j-1$$

代码需要三层循环，时间复杂度为 $O(n^3)$；如果分析该算法的决策树模型，会发现其不完全平衡，因此时间上并未达到最优。以 5 个站为例，所有可能的情况有 $2^3=8$ 种，其决策 (多叉) 树见图 1。

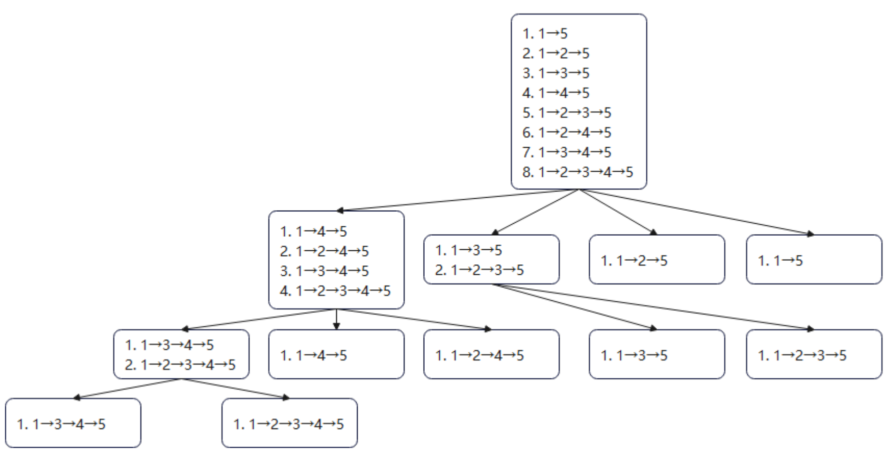

图 1: 矩阵连乘做法的决策树

我们可以修改状态以及状态转移方程来改进时间复杂度。令状态 $dp[i][j]$ 表示考虑在前 i 个站归还游艇时从第 1 个站到达第 j 个站的最少费用，则可以设计如下状态转移方程：

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

$dp[i-1][j]$ 表示不在第 i 个站归还，$dp[i-1][i] + dp[i][j]$ 表示在第 i 个站归还。

据此给出算法伪代码如下：

算法 3: 动态规划算法解决租用游艇问题

输入: 上三角矩阵 dp

输出: 从游艇出租站 1 到游艇出租站 n 所需的最少租金

```
for i = 2 to n − 1 do // 第 i 个站
    for j = i + 1 to n do
        dp[i][j] ← min(dp[i − 1][j], dp[i − 1][i] + dp[i][j])
    end
end
return dp[n − 1][n]
```

算法分析： 双层循环，时间复杂度为 $O(n^2)$

此时的决策树见图 2，可以看到，决策树完全平衡，树高降低，时间复杂度得到了改进。

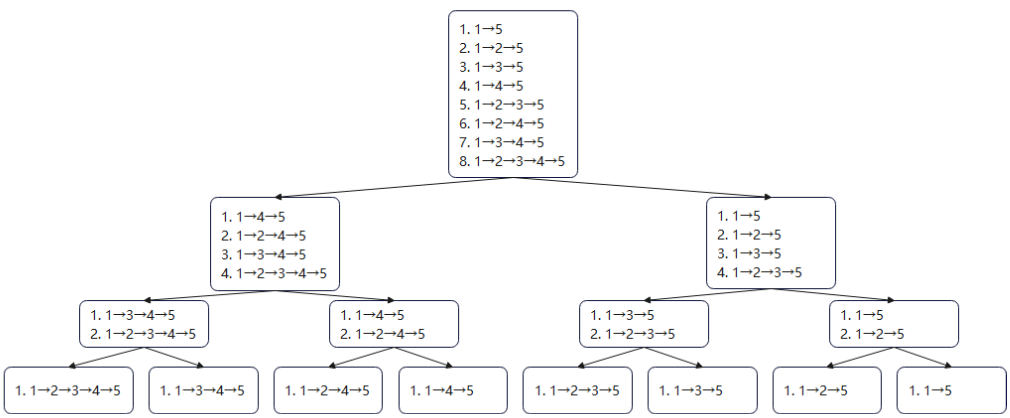

图 2: 改进做法的决策树