《算法设计与分析》第二次手写作业

2024.10.29 唐治江

《算法设计与分析》第二次手写作业

2024.10.29 唐治江

1.Step Problem

2. Buy!

3.Counting

4. Ex. Buy! Buy! Buy!

1.Step Problem

1. **建模**:

- **输入：** 台阶数n
- **输出：** 跳法数目
- **思路：**

  我们可以将问题建模为一个递归关系。设$opt(n)$表示到达第 n 级台阶的方法总数。青蛙可以通过两种方式到达第 n 级台阶：

  - 从第 n−1级跳 1 步。
  - 从第 n−2级跳 2 步。

  因此，我们可以得到递归关系：

  $opt(n) = opt(n-1) + opt(n-2)$

  其中opt(1)=1，opt(2) = 2。

2. **算法描述**:

- 设置 opt(1)=1，opt(2) = 2。
- 从 3到 n 进行迭代i，计算当前的跳跃方式数$opt(i) = opt(i-1) + opt(i-2)$ 。
- 迭代结束后，返回$opt(n)$。

3. **时间复杂性**：时间复杂度是 $O(n)$ 。我们需要从 2 迭代到 n ，每次迭代执行常数时间的计算。
4. **空间复杂性**：空间复杂度是 $O(1)$。我们只使用了常量空间来存储，不需要额外的数组来存储中间结果。

2. Buy!

1. **建模**

- **输入**:
  - v_j 为第 j 个物品的价格，假设共有k种物品，假设每种商品只能购买一次
  - w_j 为第 j 个物品的重要性，
  - n为预算上限。
- **输出**：预算限制内，最大的得分：$v_{j_1} \times w_{j_1} + v_{j_2} \times w_{j_2} + \cdots + v_{j_k} \times w_{j_k}$。
- **思路：**

  $opt(i, b)$ 表示在前i个物品中选择，总预算不超过 b 的情况下所能获得的最大总价值。Bellman方程如下：

  $opt(i, b) = \max(opt(i-1, b), opt(i-1, b-v_i) + v_i \times w_i)$。

最终，$opt(k,n)$ 就是最大可能的总价值，其中 k 是所有物品的数量。

2. **算法描述**

- 初始化一个二维数组 opt，其大小为$(k+1)\times(n+1)$，每个元素初始化为 0。
- 对于每个物品 i，从预算 $b=n$ 递减到v_i，更新 $opt(i,b)$ 为 $\max(f(i-1,b),f(i-1,b-v_i)+v_i\times w_i)$。
- 完成迭代后，$opt(k,n)$为结果。

3. **时间复杂性**：时间复杂度为 $O(kn)$ ，其中k 品的数量，n 是预算上限。因为每个物品需要针对预算的所有可能值进行计算。
4. **空间复杂性**：空间复杂度为 $O(kn)$ ，使用二维数组opt。

3.Counting

1. **建模**

- **输入**：给定数组长度n以及集合S。
- **输出**：完美数组的个数
- **思路:**

  可以将问题考虑为将S中的若干个元素拼成长度为n的数组，最终完美数组的个数等于拼接的方法数和长度为S_k数组个数控制

  - 首先计算长度为S_k数组总共的个数（该数组最小的值必须是S_k，所以$S_{k+1}\dots S_m$都是可以作为数组中的数）。
    - 可以考虑当该数组有1个S_k时，总共应该有$S_k(m-k)^{S_k-1}$种数组，当数组有两个S_k时，总共应该有$C_{S_k}^2(m-k)^{S_k-2}$。所以长度为S_k数组总共的个数$N(S_k)$为：
    - $N(S_k)=S_k(m-k)^{S_k-1}+C_{S_k}^2(m-k)^{S_k-2}+\cdots+C_{S_k}^{S_k-1}(m-k)+1$，利用二项式定理化简得到$N(S_k)=(m-k+1)^{S_k}-(m-k)^{S_k}$
  - 然后将找拼接方法的过程看作分步决策过程，长度为n的数组的形成方式有$opt[n]$，其等于：
    - $opt[n]=\Sigma_{i=1}^{m}N(S_i)opt(N-S_i)$
    - 其中要求$S_i\leq N$

2. **算法描述**

- 从1到m，计算长度为S_k的数组可能的个数，储存在数组N中，$N[k]=(m-k+1)^{S_k}-(m-k)^{S_k}$
- i从1到n，计算$opt[i]$
  - k从1到m，如果$S_k\leq i$，
    - $opt[i]+=N[k]opt[i-S_k]$
- $opt[n]$为最终答案

3. **时间复杂性**：时间复杂度为 ，其中n是数组的长度，m是集合S $的大小。
4. **空间复杂性**：空间复杂度为 $O(n)$。

4. Ex. Buy! Buy! Buy!

1. **建模**
   - **输入**：
     - v_j：第 j 个物品的价格。

        - w_j：第 j 个物品的重要性，范围为 1 到 5。
        - n预算上限。
        - 主物品和附属品的从属关系。
    - **输出**： 在预算限制内，最大化得分

2. **思路**：

   定义状态 $opt(i,b)$表示在前i个主物品中选择，总预算不超过 b时所能获得的最大价值。对于每个主物品，可以选择0个、1个或2个附属品，因此状态转移方程根据是否选择附属品而有所不同。

   若主物品i的附属品分别为a_1和 a_2，则状态转移方程为：
   $$\text{opt}(i,b)=\max(\text{opt}(i-1,b),\text{opt}(i-1,b-v_i)+v_i\times w_i,\text{opt}(i-1,b-v_i-v_{a_1})+(v_i\times w_i+v_{a_1}$$
   $$\text{opt}(i-1,b-v_i-v_{a_1}-v_{a_2})+(v_i\times w_i+v_{a_1}\times w_{a_1}+v_{a_2}\times w_{a_2}))$$

3. **算法描述**
    - 初始化一个二维数组 opt，大小为$(k+1)\times(n+1)$，每个元素初始化为 0。
    - 对于每个主物品 i，逐一更新每个预算b。
    - 完成所有更新后，$opt(k,n)$ 为最终结果。

4. **时间复杂性**：
   时间复杂度为 $O(kn)$，其中k 为主物品数量，n 为预算上限。

5. **空间复杂性**：
   空间复杂度为$O(kn)$，需要二维数组来保存状态。