算法设计与分析第二次作业

姓名：王泽黎
学号：2022K8009929011

1. Step Problem

1.1 Modeling

核心要求：设计一个算法来计算青蛙跳到一个 n 级台阶的不同方法总数，青蛙每次可以选择跳 1 步或 2 步。

关键变量：
1. n: 台阶的级数
2. dp[i]：跳到第 i 级台阶的不同方法总数

分析思路：
1. 观察到青蛙跳到第 i 级台阶的方式与其前两级台阶的跳法有关，即得到递推公式 dp[i] = dp[i-1] + dp[i-2]。
2. 显然 dp[1] = 1，dp[2] = 2。

1.2 Algorithm description

1. 初始化变量：
    - 定义数组 dp，长度为 n+1，用于记录跳到每级台阶的不同方法总数。
    - 初始化 dp[1] = 1，dp[2] = 2。
2. 计算不同方法总数：
    - 使用循环计算 dp[i]，其中 i 从 3 到 n。
    - 计算 dp[i] = dp[i-1] + dp[i-2]。
3. 返回 dp[n]。

1.3 Complexity analysis

- 时间复杂度：O(n)，时间复杂度主要需要考虑计算 dp 数组的过程，需要遍历 n 次。
- 空间复杂度：O(n)，空间复杂度主要需要考虑 dp 数组的空间占用。

2. Buy

2.1 Modeling

核心要求:
1. 预算限制：所选物品总价格不能超过 n。
2. 最大化价值

关键变量:
1. 预算 n：可用于购买物品的总金额。
2. 物品数量 m：可供选择的物品数量。
3. 物品价格 vj：每个物品的价格。
4. 物品重要性 wj：每个物品的重要性等级（1 到 5）。
5. 总价值：所选物品的总价值。

分析思路:
1. 该问题是一个 0-1 背包问题，需要考虑物品的价格和重要性。
2. 考虑使用动态规划的方法，定义状态 dp[i][j] 表示在前 i 个物品中，总价格不超过 j 的情况下，所选物品的最大价值。
3. 根据物品的价格和重要性，更新状态 dp[i][j]。

2.2 Algorithm description

1. 初始化变量:
   - 定义二维数组 dp[m+1][n+1]，其中 dp[i][j] 表示在前 i 个物品中，总价格不超过 j 的情况下，所选物品的最大价值。
   - 初始化 dp[0][j] = 0，dp[i][0] = 0，即为当没有物品或者预算为 0 时，最大价值为 0。
2. 状态转移:
   - 对每个物品 i(从 1 到 m) 和每个预算 j(从 1 到 n)，进行如下计算:
     - 检查当前物品的价格 vi 是否小于等于当前预算 j:
       - 如果可以选择当前物品
         - 选择当前物品的价值为 dp[i-1][j-vi] + vi × wi，其中 vi 为当前物品的价格，wi 为当前物品的重要性。
         - 不选择当前物品的价值为 dp[i-1][j]。
         - 更新 dp[i][j] 为两者中的最大值。
       - 如果不能选择当前物品，则 dp[i][j] = dp[i-1][j]。
3. 返回 dp[m][n]。

2.3 Complexity analysis

- 时间复杂度：O(mn)，时间复杂度主要需要考虑计算 dp 数组的过程，需要遍历 m 个物品和 n 个预算。
- 空间复杂度：O(mn)，空间复杂度主要需要考虑 dp 数组的空间占用。

3. Counting

3.1 Modeling

核心要求：
1. 好数组(Nice Array)：给定一个长度为 n 的数组 a，如果能够将其划分为若干个区间，使得每个区间的最小值等于该区间的长度，则称该数组为好数组。

关键变量：
1. n：数组 a 的长度。
2. S：包含 m 个正整数的升序序列。
3. dp[i]：以 a[i] 为结尾的好数组的个数。

分析思路：
1. 对于每个长度 i(从 1 到 n)，考虑所有可能的区间长度 j(从 1 到 i)。
2. 如果 k 在集合 S 中，则将 dp[i-k] 添加到 dp[i] 中。

3.2 Algorithm description

1. 初始化变量：
    - 定义数组 dp，长度为 n+1，用于记录以 a[i] 为结尾的好数组的个数。
    - 初始化 dp[0] = 1。
2. 状态转移：
    - 对于每个长度 i(从 1 到 n)，计算 dp[i]：
        - 对于每个区间长度 j(从 1 到 i)，如果 j 在集合 S 中，则将 dp[i-j] 添加到 dp[i] 中。
3. 返回 dp[n]。

3.3 Complexity analysis

- 时间复杂度：O(n^2)，时间复杂度主要需要考虑计算 dp 数组的过程，需要遍历 n 个长度和 n 个区间长度。
- 空间复杂度：O(n)，空间复杂度主要需要考虑 dp 数组的空间占用。

4. Buy!Buy!Buy

4.1 Modeling

核心要求：
1. 预算限制：所选物品总价格不能超过 n。
2. 每个主物品可以选择0、1或2个配件，但必须先购买相应的主物品。
3. 每个物品都有价格和重要性等级，选择的物品组合应使得总价值最大。

关键变量：
1. 预算 n：可用于购买物品的总金额。
2. 主物品数量 m：可供选择的主物品数量。
3. 物品价格 vj：每个物品的价格。
4. 物品重要性 wj：每个物品的重要性等级（1 到 5）。
5. 总价值：所选物品的总价值。

分析思路：
1. 与第二题类似，该问题也是一个 0-1 背包问题，需要考虑物品的价格和重要性。
2. 与第二题的区别是，该问题在更新主物品价值后，还需要考虑配件的价值。

4.2 Algorithm description

1. 初始化变量：
    - 定义二维数组 dp[m+1][n+1]，其中 dp[i][j] 表示在前 i 个主物品中，总价格不超过 j 的情况下，所选物品的最大价值。
    - 初始化 dp[0][j] = 0，dp[i][0] = 0，即为当没有物品或者预算为 0 时，最大价值为 0。
2. 状态转移：
    - 对每个物品 i(从 1 到 m) 和每个预算 j(从 1 到 n)，进行如下计算：
        - 检查当前物品的价格 vi 是否小于等于当前预算 j：
            - 如果可以选择当前物品
                - 分别计算选择可能配件的价值。
                - 不选择当前物品的价值为 dp[i-1][j]。
                - 更新 dp[i][j] 为以上价值中的最大值。
            - 如果不能选择当前物品，则 dp[i][j] = dp[i-1][j]。
3. 返回 dp[m][n]。

4.3 Complexity analysis

- 时间复杂度：O(mn)，时间复杂度主要需要考虑计算 dp 数组的过程，需要遍历 m 个物品和 n 个预算。
- 空间复杂度：O(mn)，空间复杂度主要需要考虑 dp 数组的空间占用。