assignment3

Xiaoma

2022 年 11 月 20 日

题目 1.

解答. 在没有切割成本时，钢条切割问题的状态转移方程为

$$r_n = \begin{cases} p_1 & n = 1 \\ \max_{1 \le i \le n}(p_i + r_{n-i}) & n > 1 \end{cases}$$

加入切割成本以后，状态转移方程变为

$$r_n = \begin{cases} p_1 & n = 1 \\ \max(p_n + \max_{1 \le i \le n-1}(p_i + r_{n-i} - c)) & n > 1 \end{cases}$$

使用数组将不同长度的最大利润存储下来，自底向上计算钢条切割问题。

题目 2.

解答. 通过动态规划问题求解该问题，我们需要考虑苹果与盘子数量之间的关系对存放方法的影响，设苹果的数量为 i，盘子的数量为 j

- 当苹果的数量小于等于盘子数量时，无论如何放置，相较于苹果的数量和盘子相等时，最终都会多出 $j - i$ 个空盘子，故该种情况下放苹果的方法数等于此时苹果数量与盘子数量相等时的放苹果方法数。

- 当苹果的数量大于盘子数量时，则放苹果的方法数可以分解为两种情况：1. 将新增的盘子作为空盘，则此时放苹果的数量与未新增盘子前相同。2. 将新增的盘子放入苹果，先将每个盘子放入一个苹果，然后剩余的苹果放入所有盘子的方法种类就是该种情况放苹果的种类数。

状态转移方程变为

$$dp[i][j] = \begin{cases} dp[i][i] & j >= i \\ dp[i][j-1] + dp[i-j][j] & j < i \end{cases}$$

Algorithm 1: Put-Apples

Input: The number of apple : m; The number of plates : n;

Output: The number of methods : $methods$;

```
Initialize dp;
for int i = 0; i < m; ++i do
    dp[i][0] = 0;
for int i = 0; i < n; ++i do
    dp[0][i] = 0;
for int i = 0; i < m; ++i do
    for int j = 0; j < n; ++j do
        if j >= i then
            dp[i][j] = dp[i][i];
        else
            dp[i][j] = dp[i][j - 1] + dp[i - j][j];
return dp[m - 1][n - 1]
```

题目 3.

解答.

1. 对点集进行排序
2. 从头开始遍历点集，设区间的左边界为 x_i，然后去掉点集中满足 $x_i \leq x_j \leq x_i+$ 的点 x_j，直至遍历结束。

时间复杂度分析：

- 排序的时间复杂度为 $O(n \log n)$
- 遍历点集的时间复杂度为 $O(n)$
- 算法的总时间复杂度为 $O(n \log n)$

证明:
已知需要建立的区间为单位区间，设每个区间的左边界为 $x_1, x_2, ..., x_m$，则它们所在区间必然不相交，如果要包含点集中的所有点，则单位区间的数量至少要 m 个，所以该算法找到的是满足条件的最小区间。

题目 4.

解答. 贪心算法求解问题的条件：

1. 贪心选择性质
2. 最优子结构

1. 贪心选择性质：
   该问题可以通过局部最优解构造全局最优解，每次为进程安排机器时都会选择最空闲的机器，所以每次选择都会保证任务完成的总时间尽可能的短，尽可能选择空闲的机器的局部最优解在全局最优解序列中。

2. 最优子结构：

   每个问题的最优解都包含组成该问题的子问题的最优解，即 n 个进程的多机调度最优解一定包含着 x(m<x<n) 个进程的多机调度的最优解。

Algorithm 2: Multi-machine Scheduling

Input: The number of processes : n;

The time required for the process : $t[n]$;

The number of machines : m;

Output: The shortest time required : *shortest_time*

Initialize *dp*;

sort(t);

for *int i = 0; i < m; ++i* **do**
    dp[i] = t[i];

for *int i = m; i < n; ++i* **do**
    min_time = findmin(dp);
    dp[min] += t[i];

max_time = findmax(dp);

return *dp[max_time]*

题目 5.

解答.

- 每次尽可能拿最大面额的硬币，直到零钱总额为 n。

证明：

贪心算法求解问题的条件：

1. 贪心选择性质
2. 最优子结构

1. 贪心选择性质：
   该问题可以通过局部最优解构造全局最优解，如果 1 美分的硬币达到 5 个，则可以换成 1 个 5 美分的硬币，其他情况同理，所以每次选择的零钱面额都会保证硬币数量最小，尽可能选择最大面额的硬币的局部最优解在全局最优解序列中。
2. 最优子结构：
   每个问题的最优解都包含组成该问题的子问题的最优解，即大面额零钱兑换问题的最优解一定包含着子面额的零钱兑换问题的最优解。

题目 6.

解答.

- 最优性原理：
  假设 $X_1, X_2, ..., X_n$ 是 0-1 背包问题的最优解，则 $X_1, X_2, ..., X_{n-1}$ 是该问题的子问题，假设 $Y_1, Y_2, ..., Y_n$ 是该问题的最优解，则

  $$(v_1Y_1+v_2Y_2+...+v_{n-1}Y_{n-1})+v_nX_n > (v_1X_1+v_2X_2+...+v_{n-1}X_{n-1})+v_nX_n$$

  则 $Y_1, Y_2, ..., Y_{n-1}, X_n$ 才是问题的最优解，与原假设矛盾。则 0-1 背包问题满足最优性原理。
- 无后小性：
  在任意阶段，只要背包剩余容量和可选物品相同，最优选择相同，不受之前所选择的物品影响。则 0-1 背包问题满足无后效性。

Algorithm 3: 0-1 Knapsack

Input: The number of items : n; The value of items : $v[n]$;

The weight of items : $w[n]$; The capacity of the backpack : $weight$

Output: Maximum value : max_val

```
Initialize dp;
for int i = 0; i < weight; ++i do
    dp[0][j] = 0;
for int i = 0; i < n; ++i do
    dp[i][0] = 0;
for int i = 0; i < n; ++i do
    for int j = 0; j < weight; ++j do
        if w[i] < j then
            dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);
        else
            dp[i][j] = dp[i - 1][j];
return dp[n − 1][weight − 1];
```