实验五　0-1 背包问题的算法设计

一、实验原理

1. 背包问题

背包问题已经是一个很经典而且讨论很广泛的算法问题了。

背包问题泛指这类种问题：给定一组有固定价值和固定重量的物品，以及一个已知最大承重量的背包，求在不超过背包最大承重量的前提下，能放进背包里面的物品的最大总价值。具体各类背包问题可以分成以下 3 种不同的子问题。

1.1 0-1 背包问题

问题描述：有编号分别为 a,b,c,d,e 的五件物品，它们的重量分别是 2,2,6,5,4，它们的价值分别是 6,3,5,4,6，每件物品数量只有一个，现在给你个承重为 10 的背包，如何让背包里装入的物品具有最大的价值总和？

特点:每个物品只有一件，选择放或者不放。

1.2 完全背包问题

问题描述：有编号分别为 a,b,c,d 的四件物品，它们的重量分别是 2,3,4,7，它们的价值分别是 1,3,5,9，每件物品数量无限个，现在给你个承重为 10 的背包，如何让背包里装入的物品具有最大的价值总和？

特点：每个物品可以无限选用。

1.3 多重背包问题

问题描述：有编号分别为 a,b,c 的三件物品，它们的重量分别是 1，2，2，它们的价值分别是 6，10，20，他们的数目分别是 10，5，2，现在给你个承重为 8 的背包，如何让背包里装入的物品具有最大的价值总和？

特点 ：每个物品都有**一定的数量**。

2. 解决算法

2.1 动态规划算法

动态规划原理：动态规划是一种将问题实例分解为更小的、相似的子问题，并存储子问题的解而避免计算重复的子问题，以解决最优化问题的算法策略。

动态规划法所针对的问题有一个显著的特征，即它所对应的子问题树中的子问题呈现大量的重复。动态规划法的关键就在于，对于重复出现的子问题，只在第一次遇到时加以求解，并把答案保存起来，让以后再遇到时直接引用，不必重新求解。

用动态规划方法解决 0-1 背包问题：

步骤 1-找子问题：子问题必然是和物品有关的，对于每一个物品，有两种结果：能装下或者不能装下。第一，包的容量比物品体积小，装不下，这时的最大价值和前 i-1 个物品的最大价值是一样的。第二，还有足够的容量装下该物品，但是装了不一定大于当前相同体积的最优价值，所以要进行比较。由上述分析，子问题中物品数和背包容量都应当作为变量。因此子问题确定为背包容量为 j 时，求前 i 个物品所能达到最大价值。

步骤 2-确定状态：由上述分析，“状态”对应的“值”即为背包容量为 j 时，求前 i 个物品所能达到最大价值，设为 dp[i][j]。初始时，dp[0][j](0<=j<=V)为 0，没有物品也就没有价值。

步骤 3-确定状态转移方程：由上述分析，第 i 个物品的体积为 w,价值为 v，则状态转移方程为

- **j<w , dp[i][j] = dp[i-1][j]** //背包装不下该物品，最大价值不变
- **j>=w, dp[i][j] = max{ dp[i-1][j-list[i].w] + v, dp[i-1][j] }** //和不放入该物品时同样达到该体积的最大价值比较

2.2 贪婪算法

贪心法把一个复杂问题分解为一系列较为简单的局部最优选择，每一步选择都是对当前的一个扩展，直到获得问题的完整解。

k-optimal 算法用来解决 0-1 背包问题：

步骤 1-计算每种物品单位重量的价值 Vi/Wi ；

步骤 2-依贪心选择策略，将尽可能多的单位重量价值最高的物品装入背包。

步骤 3-若将这种物品全部装入背包后，背包内的物品总重量未超过 C，则选择单位重量价值次高的物品并尽可能多地装入背包。

依此策略一直地进行下去，直到背包装满为止。

2.3 回溯法

回溯法先确定解空间的结构，使用深度优先搜索，搜索路径一般沿树形结构进行，在搜索过程中，首先会判断所搜索的树结点是否包含问题的解，如果肯定不包含，则不再搜索以该结点为根的树结点，而向其祖先结点回溯；否则进入该子树，继续按深度优先策略搜索。

运用回溯法解题通常包含以下三个步骤：

a. 针对所给问题，定义问题的解空间；

b. 确定易于搜索的解空间结构；

c. 以深度优先的方式搜索解空间，并且在搜索过程中用剪枝函数避免无效搜索；

2.4 分支定界法

分支限界法类似于回溯法，也是在问题的解空间上搜索问题解的算法。

分支限界法首先要确定一个合理的限界函数（bound funciton），并根据限界函数确定目标函数的界[down，up]，按照广度优先策略或以最小耗费优先搜索问题的解空间树，在分直结点上依次扩展该结点的孩子结点，分别估算孩子结点的目标函数可能值，如果某孩子结点的目标函数可能超出目标函数的界，则将其丢弃；否则将其加入待处理结点表（简称 PT 表），依次从表 PT 中选取使目标函数取得极值的结点成为当前扩展结点，重复上述过程，直到得到最优解。

常见的两种分枝限界法包含队列式(FIFO)分支限界法和优先队列式分支限界法。

分支限界法解决 0-1 背包问题

- 按价值重量比 递减 的顺序，对n个商品进行排序
  排序后商品序号的结合为S = {0, 1, ..., n-1}
- 将这些商品分为3个集合：
  S_1——选择装入背包的商品集合
  S_2——不选择装入背包的商品集合
  S_3——尚待选择的商品集合
- $S_1(k)$、$S_2(k)$、$S_3(k)$分别表示在搜索深度为k时的3个集合中的商品。开始时有：
  $S_1(0) = \varnothing$
  $S_2(0) = \varnothing$
  $S_3(0) = \{0, 1, ..., n-1\}$

假设比值 pi/wi 最大的物品序号为 s（s ∈ S3），按照价值重量比递减排序后，s 就是集合 S3(k)中的第一个元素。用 s 进行分支，一个分支结点表示把 s 装入背包，另一个分支结点表示不把 s 装入背包。

- **把商品s装入背包的分支结点**

  $S_1(k+1) = S_1(k) \cup \{k\}$

  $S_2(k+1) = S_2(k)$

  $S_3(k+1) = S_3(k) - \{k\}$

- **不把商品s装入背包的分支结点**

  $S_1(k+1) = S_1(k)$

  $S_2(k+1) = S_2(k) \cup \{k\}$

  $S_3(k+1) = S_3(k) - \{k\}$

① **设置上界估算方法 b(k)**

假定 b(k)表示在搜索深度为 k 时，某个分支结点的背包中商品的价值上界。此时 S3(k) = {k, k+1, ..., n-1}。

$$\text{if} \quad M < \sum_{i \in S_1(k)} w_i$$
$$\text{then} \quad b(k) = 0$$

$$\text{if} \quad M = \sum_{i \in S_1(k)} w_i + \sum_{i=k}^{l-1} w_i + x \cdot w_l \qquad 0 \le x < 1, k < l, k, \cdots, l \in S_3(k)$$
$$\text{then} \quad b(k) = \sum_{i \in S_1(k)} p_i + \sum_{i=k}^{l-1} p_i + x \cdot p_l$$

（参考来自：https://www.jianshu.com/p/c738c8262087）

② **利用分支限界法求解：**

第一步，初始化 bound = 0，把物品按价值重量比递减排序，建立根节点 X；

第二步，建立新结点，计算新结点的上界，与 bound 进行比较，据此判定是否插入优先队列，直到当前尚待选择的物品集合为空时，找到一个可行解，判定是否更新 bound；

第三步，同样操作建立另外一个新结点 Z；第四步，取出优先队列首元素作为根结点 X，

第四步，如此往复直到搜索深度为所有物品数量为止。

二、实验要求

算法设计：

输入物品数 n，背包容量 c，输入 n 个物品的重量、价值，在以上算法中任选两个实现最优解决 0-1 背包问题。

请问：所选算法的实现流程图或者伪代码是什么？比较时间复杂度和空间复杂度，得出什么结论？

注意：实验结果当节课找实验老师检查，无需提交实验报告。