运筹学课程实验实验报告

一、最小成本循环流的强多项式算法

二、动态规划算法

（一）实验要求

实现动态规划的一种快速算法。

1. 调研一种动态规划的快速算法，在报告中详细写出算法迭代过程；
2. 本作业需至少构造一个实际案例（多重背包问题/凑零钱问题/投资问题/排序问题等）；
3. 算法程序应至少在该案例上进行测试，鼓励构造更多实例，充分测试。

（二）问题描述

本实验选取多重背包问题作为动态规划算法的实例。

多重背包问题：一位旅行者能承受的背包最大重量是 b 千克，现有 n 种物品供他选择装入背包，第 i 种物品单件重量为 $w[i]$ 千克，最多有 $p[i]$ 件，每件的价值为 $v[i]$ ，$1 \leq i \leq n$ 。设第 i 种物品装载数量是 $x[i]$，问旅行者应该如何选择所携带的物品件数，以使得总装载价值最大[1]。

（三）算法原理

1. 理论说明

首先寻找问题的状态转移方程：

1. 定义$F[i,v]$为前 i 种物品恰放入一个容量为 V 的背包的最大价值。
2. 于是可以根据最终的背包方案是否取第 i 个元素分为如下两种情况：

$$F[i,v]=\begin{cases} F[i-1,V], & \text{背包方案不取第}i\text{个元素} \\ \max\{F[i-1,V-k*v[i]]+k*w[i] \mid 0<k\leq p[i]\}, & \text{背包方案包含第}i\text{个元素} \end{cases}$$

第一种情况显然成立。第二种情况选取第 i 个背包 k 个，于是最大价值可以转变为 $F[i-1,V-k*v[i]]+k*w[i]$ ，而动态规划算法选取最大的作为最优解。

3. 精简结构，发现第一种情况恰为 $k=0$ 的情况。于是精简结构为：

$$F[i,v]=\max\{F[i-1,V-k*v[i]]+k*w[i] \mid 0\leq k\leq p[i]\}$$

之后可以根据状态转移方程编程实现。

2. 编程实现

代码的主要部分如下所示：

```
1  def mul_pack_dp(V, w, v, p):
2      n = len(w)
3      dp = [[0] * (V + 1) for _ in range(n + 1)]
4      min_cost = [[0] * (V + 1) for _ in range(n + 1)]
5
6      for i in range(1, n + 1):
7          x = i - 1  # 第i件物品的索引
8          for j in range(V + 1):
9              for k in range(min(p[x], j // w[x]) + 1):
10                 if dp[i - 1][j - k * w[x]] + k * v[x] > dp[i][j]:
11                     dp[i][j] = dp[i - 1][j - k * w[x]] + k * v[x]
12                     min_cost[i][j] = k
```

该函数负责处理多重背包问题，采用bottle-up方法实现，下面对程序进行几点说明：

1. 函数的参数：V 是多重背包问题的最大背包重量，w 是一个记录各个物品重量的列表，v 是一个记录各个物品价值的列表，p 是一个记录各个物品最多的数量的列表。
2. 函数的3~4行，进行初始化，初始化 dp（即为状态转移方程的F）为 $(n+1)\times(V+1)$维的二维列表，初始化 min_cost 为记录 $F[i,v]$ 状态的辅助列表，也为 $(n+1)\times(V+1)$维。
3. 函数的6~12行为主要的循环部分，实现了状态转移方程：
    1. 6、8行的循环是对$F[i,v]$的循环，意在自底向上求出$F[i,v]$的值；
    2. 9行的循环为状态转移方程中 $\max$ 函数的求解，限制 $0\le k\le p[i]$，且 k 也不能超过 $\frac{V}{w[x]}$ ，即为背包所能承载的最多数量；
    3. 11、12行记录最优的 $\max$ 值，并记录最优的 $\max$ 值所需要的物品数量 k 。

（四）数据集说明

1. 数据集包含如下三个文件夹：little.txt、normal.txt、large.txt。
2. 数据集txt的格式为：
    1. 首先一行为 V 的值；
    2. 第二行为物品的个数 n ；
    3. 最后三行依次为 $w[i]$ 、 $v[i]$ 、 $p[i]$,中间用逗号分隔。
3. 三个数据集的规模 n 依次为3，30,300。
4. 三个数据集的目标 V 依次为15，500，5000。

5. 三个数据集的物品重量依次为 $1 \leq w[i] \leq 5$、$1 \leq w[i] \leq 100$、$1 \leq w[i] \leq 100$。
6. 三个数据集的物品价值依次为 $1 \leq v[i] \leq 5$、$1 \leq v[i] \leq 100$、$1 \leq v[i] \leq 100$。
7. 三个数据集的物品最大数量依次为 $1 \leq p[i] \leq 5$、$1 \leq p[i] \leq 30$、$1 \leq p[i] \leq 30$。

（五）程序输入、输出

1. 程序输入

在“多重背包.py”文件下，建立新的文件夹“dataset”,依次放入三个数据集即可。

2. 程序输出

程序输出结果如下所示：

```
11
2(3个) 1(1个)
2602
13(16个) 9(17个) 8(3个) 7(1个)
38339
292(13个) 289(23个) 285(9个) 281(7个) 280(10个) 270(22个) …………
```

依次为little.txt、normal.txt、large.txt的输出结果，每个两行

具体如下：

1. 该多重背包问题的最优解，即最大价值；
2. 该多重背包问题的最优解结构，即选取的物品和选取该物品的数量。

（六）程序测试结果

程序测试结果如下所示：

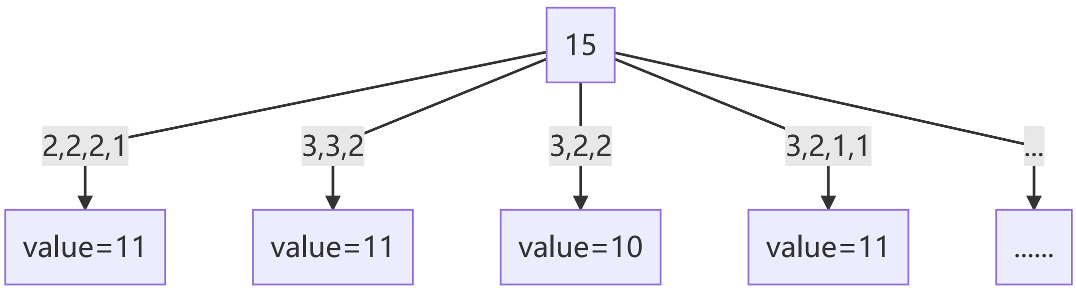
```
11
2(3个) 1(1个)
2602
13(16个) 9(17个) 8(3个) 7(1个)
38339
292(13个) 289(23个) 285(9个) 281(7个) 280(10个) 270(22个) …………
```

依次为little.txt、normal.txt、large.txt的输出结果。

以small.txt为例，数据为：

```
15
3
3, 4, 5
2, 3, 4
4, 3, 2
```

简单验证几种可能情况可得



程序得到的结果确实是一种最优解。

（七）分析总结

关于动态规划算法的多重背包问题，有以下分析：

1. 该问题利用了动态规划算法的精髓，即状态转移方程来构建问题的解；
2. 该程序的算法时间复杂度为 $O(n^3)$ ，很显然由程序的三重循环可以得到；
3. 该程序输出的结果只选取了一种可能的最优解，并不能输出问题的所有最优解，通过修改输出函数，也可以输出所有的最优解。

三、非精确一维线搜索Wolfe-Powell算法

[1] 运筹学课程实验PPT14页。 ↩