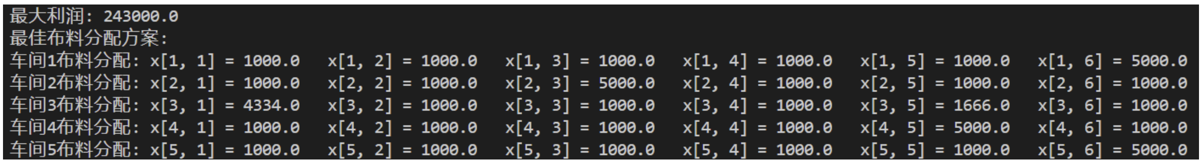
运筹学第二次作业

Q1

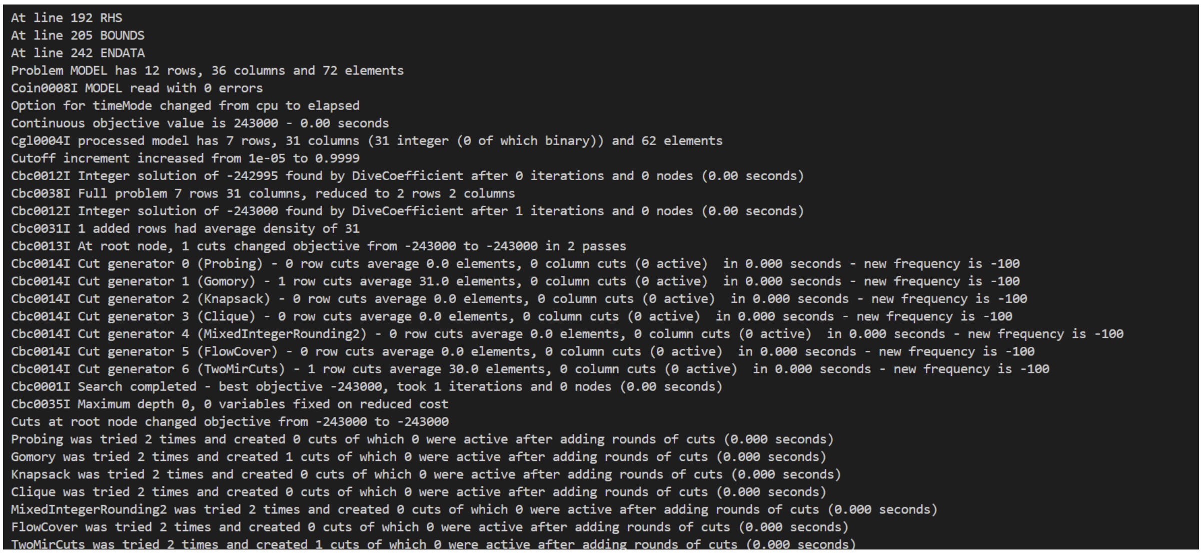
代码如下：

```python
import pulp

# 创建问题变量
prob = pulp.LpProblem("Maximize_Profit", pulp.LpMaximize)

# 定义常量
num_machines = 5
num_fabrics = 6

a = [6, 6, 7, 8, 9, 10]
r = [[4, 3, 4, 4, 5, 6],
     [3, 4, 5, 3, 4, 5],
     [5, 3, 4, 5, 5, 4],
     [3, 3, 4, 4, 6, 6],
     [3, 3, 3, 4, 5, 7]]

# 定义变量
x = pulp.LpVariable.dicts("x", ((i, j) for i in range(num_machines) for j in range(num_fabrics)), lowBound=1000, cat='Integer')
y = pulp.LpVariable.dicts("y", (j for j in range(num_fabrics)), lowBound=0, cat='Integer')

# 目标函数
prob += pulp.lpSum(r[i][j] * x[i, j] for i in range(num_machines) for j in range(num_fabrics)), "Total_Profit"

# 约束条件
for j in range(num_fabrics):
    prob += pulp.lpSum(x[i, j] for i in range(num_machines)) == y[j], f"Fabric_Demand_{j}"

for i in range(num_machines):
    prob += pulp.lpSum(x[i, j] for j in range(num_fabrics)) <= 10000, f"Machine_Capacity_{i}"

prob += pulp.lpSum(a[j] * y[j] for j in range(num_fabrics)) <= 400000, "Total_Fund"

# 求解问题
prob.solve()

# 输出结果
print("Status:", pulp.LpStatus[prob.status])
print("最大利润:", pulp.value(prob.objective))

print("最佳布料分配方案:")
for i in range(num_machines):
    print(f"车间{i+1}布料分配:", end=" ")
    for j in range(num_fabrics):
        print(f"x[{i+1}, {j+1}] = {x[i, j].varValue}", end="    ")
    print()  # 换行

print("各种布料数量:")
for j in range(num_fabrics):
    print(f"y[{j+1}] = {y[j].varValue}")

```

结果如下:



```
最大利润: 243000.0
最佳布料分配方案:
车间1布料分配: x[1, 1] = 1000.0   x[1, 2] = 1000.0   x[1, 3] = 1000.0   x[1, 4] = 1000.0   x[1, 5] = 1000.0   x[1, 6] = 5000.0
车间2布料分配: x[2, 1] = 1000.0   x[2, 2] = 1000.0   x[2, 3] = 5000.0   x[2, 4] = 1000.0   x[2, 5] = 1000.0   x[2, 6] = 1000.0
车间3布料分配: x[3, 1] = 4334.0   x[3, 2] = 1000.0   x[3, 3] = 1000.0   x[3, 4] = 1000.0   x[3, 5] = 1666.0   x[3, 6] = 1000.0
车间4布料分配: x[4, 1] = 1000.0   x[4, 2] = 1000.0   x[4, 3] = 1000.0   x[4, 4] = 1000.0   x[4, 5] = 5000.0   x[4, 6] = 1000.0
车间5布料分配: x[5, 1] = 1000.0   x[5, 2] = 1000.0   x[5, 3] = 1000.0   x[5, 4] = 1000.0   x[5, 5] = 1000.0   x[5, 6] = 5000.0
```



```
At line 192 RHS
At line 205 BOUNDS
At line 242 ENDATA
Problem MODEL has 12 rows, 36 columns and 72 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Continuous objective value is 243000 - 0.00 seconds
Cgl0004I processed model has 7 rows, 31 columns (31 integer (0 of which binary)) and 62 elements
Cutoff increment increased from 1e-05 to 0.9999
Cbc0012I Integer solution of -242995 found by DiveCoefficient after 0 iterations and 0 nodes (0.00 seconds)
Cbc0038I Full problem 7 rows 31 columns, reduced to 2 rows 2 columns
Cbc0012I Integer solution of -243000 found by DiveCoefficient after 1 iterations and 0 nodes (0.00 seconds)
Cbc0031I 1 added rows had average density of 31
Cbc0013I At root node, 1 cuts changed objective from -243000 to -243000 in 2 passes
Cbc0014I Cut generator 0 (Probing) - 0 row cuts average 0.0 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 1 (Gomory) - 1 row cuts average 31.0 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 2 (Knapsack) - 0 row cuts average 0.0 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 3 (Clique) - 0 row cuts average 0.0 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 4 (MixedIntegerRounding2) - 0 row cuts average 0.0 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 5 (FlowCover) - 0 row cuts average 0.0 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0014I Cut generator 6 (TwoMirCuts) - 1 row cuts average 30.0 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0001I Search completed - best objective -243000, took 1 iterations and 0 nodes (0.00 seconds)
Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
Cuts at root node changed objective from -243000 to -243000
Probing was tried 2 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Gomory was tried 2 times and created 1 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Knapsack was tried 2 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Clique was tried 2 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 2 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
FlowCover was tried 2 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
TwoMirCuts was tried 2 times and created 1 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
```

Q2

模型描述

我们有一个容量为 b 的背包，有 n 件物品，每件物品的重量为 a_j，价值为 c_j。我们需要找出一组物品，使得在总重量不超过 b的条件下，总价值最大。

动态规划方程

定义状态 $dp[i][w]$ 表示前 i 件物品中，总重量不超过 w的最大价值。我们需要计算$dp[n][b]$。

状态转移方程如下： $dp[i][w] = max(dp[i-1][w], dp[i-1][w-ai]+ci)$ 其中：

- $dp[i-1][w] \times dp[i-1][w]$表示不选第 i件物品时的最大价值。
- $dp[i-1][w-a_i]+c_i$表示选第 i 件物品时的最大价值。

初始条件： $dp[0][w] = 0$对于所有w

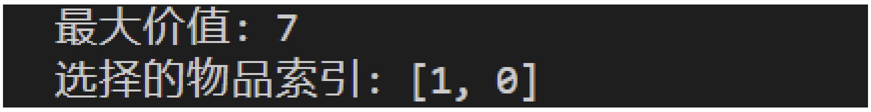
赋值设定

假设有4件物品，重量分别为2，3，4，5，价值分别为3，4，5，6。背包容量为5。

代码如下：

```python
def knapsack(b, weights, values):
    n = len(weights)
    # 创建一个二维数组来存储动态规划的状态
    dp = [[0] * (b + 1) for _ in range(n + 1)]

    # 填充dp数组
    for i in range(1, n + 1):
        for w in range(b + 1):
            if weights[i - 1] <= w:
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1])
            else:
                dp[i][w] = dp[i - 1][w]

    # 回溯找出哪些物品被选择了
    w = b
    items_selected = []
    for i in range(n, 0, -1):
        if dp[i][w] != dp[i - 1][w]:
            items_selected.append(i - 1)
            w -= weights[i - 1]

    return dp[n][b], items_selected

# 定义物品的重量和价值
weights = [2, 3, 4, 5]
values = [3, 4, 5, 6]
b = 5

# 调用函数并输出结果
max_value, items_selected = knapsack(b, weights, values)
print("最大价值:", max_value)
print("选择的物品索引:", items_selected)

```

结果如下：



```
最大价值: 7
选择的物品索引: [1, 0]
```