# 5-3 最小重量机器设计问题

## 问题描述

设某一机器由n个部件组成，每一种部件都可以从m个不同的供应商处购得。设wij是从供应商j处购来的部件i的重量，cij是相应的价格。

试设计一个算法，给出总价格不超过c的最小重量机器设计。对于给定的机器部件重量和机器部件价格，计算总价值不超过d的最小重量机器设计。

数据输入：由文件input.txt给出。第一行由3个正整数n，m，d。接下来的2n行，每行m个数。前n行是c，后n行是w。

结果输出:将计算的最小重量及每个部件的供应商输出到output.txt。

## 算法分析

需要实现的是所有部件都有厂商提供，并且价格和和重量都有限制，所以做回溯剪枝就好

## 算法实现

#include <iostream>

#include <fstream>

#include <string>

#include <cmath>

#include <vector>

using namespace std;

int n = 0;

int k = 0;

int d = 0;//最低价格

int minW = INT\_MAX;//最轻重量

vector<int> result;

vector<vector<int>> w;//从供应商j处购得部件i的重量

vector<vector<int>> c;//从供应商j处购得部件i的价格

//0-n的部件每样都需要，只能访问一次

void readFile()

{

fstream in;

in.open("input.txt", ios::in);

if (!in)

{

return;

}

string temp;

getline(in, temp, ' ');

n = stoi(temp);

getline(in, temp, ' ');

k = stoi(temp);

getline(in, temp);

d = stoi(temp);

w.resize(n);

c.resize(n);

for (int i = 0; i < n; i++)

{

c[i].resize(k);

for (int j = 0; j < k - 1; j++)

{

getline(in, temp, ' ');

c[i][j] = stoi(temp);

}

getline(in, temp);

c[i][k - 1] = stoi(temp);

}

for (int i = 0; i < n; i++)

{

w[i].resize(k);

for (int j = 0; j < k - 1; j++)

{

getline(in, temp, ' ');

w[i][j] = stoi(temp);

}

getline(in, temp);

w[i][k - 1] = stoi(temp);

}

result.resize(n);

in.close();

}

void backTrace(int money, int weight, int index, vector<int> &visited)

{

if (index == n)

{

if (minW > weight)

{

minW = weight;

result = visited;

}

minW = min(weight, minW);

return;

}

for (int j = 0; j < k; j++)

{

if (money + c[index][j] <= d && weight < minW)

{

visited[index] = j + 1;

backTrace(money + c[index][j], weight + w[index][j], index + 1, visited);

}

}

}

void writeFile()

{

fstream in;

in.open("output.txt", ios::out);

in << minW << "\n";

for (int i = 0; i < n; i++)

in << result[i] << " ";

in.close();

}

int main()

{

readFile();

vector<int> visited (n, 0);//访问路径

backTrace(0,0,0, visited);

writeFile();

}

## 结果分析

### 测试文档1：

**input.txt**

8 18 14

18 15 20 5 15 10 16 6 1 6 17 6 1 2 17 15 13 17

16 6 7 4 7 2 11 6 18 4 13 12 8 5 2 8 15 14

12 6 19 10 13 8 2 10 16 4 15 15 16 13 17 12 14 4

18 18 2 13 15 19 5 12 18 7 13 9 8 17 10 13 15 11

8 5 14 11 18 20 17 3 11 17 13 11 4 9 17 14 19 1

10 7 8 11 13 3 19 3 12 11 12 14 4 2 12 10 14 15

12 9 13 9 16 17 12 15 6 3 11 17 13 17 14 13 4 4

19 12 3 19 3 20 19 12 8 19 8 10 19 20 3 1 7 1

16 12 4 16 2 6 15 1 13 3 7 16 5 3 16 16 14 19

12 14 6 2 11 15 9 17 15 16 19 20 14 14 20 9 4 4

6 13 16 6 3 12 12 19 11 20 4 13 9 18 7 17 8 1

4 17 3 20 3 8 12 7 4 12 6 12 1 18 13 20 20 8

4 15 1 10 2 12 8 11 5 4 20 13 12 20 1 3 3 11

1 9 2 1 16 1 12 4 5 2 7 15 12 3 9 4 13 6

13 1 10 8 5 13 20 10 6 4 8 15 8 8 20 11 9 9

2 10 11 1 18 8 20 11 18 2 3 6 14 16 19 4 3 15

**output.txt**

57

13 6 7 3 18 14 10 16

### 测试文档2：

**input.txt**

3 3 4

1 2 3

3 2 1

2 2 2

1 2 3

3 2 1

2 2 2

**output.tx**t

4

1 3 1

# 5-15 最佳调度问题

## 问题描述

假设有n（n<=20）个任务由k（k<=20）个可并行工作的机器完成。完成任务i需要的时间为ti。 试设计一个算法，对任意给定的整数n和k，以及完成任务i 需要的时间为ti ，i=1~n。计算完成这n个任务的最佳调度，使得完成全部任务的时间最早。

输入格式:

由文件input.txt给出。输入数据的第一行有2 个正整数n和k。第2 行的n个正整数是完成n个任务需要的时间。

输出格式:

将计算出的完成全部任务的最早时间输出到output.txt。

## 算法分析

仍然是回溯，不过变成了3个节点，读到叶子节点回退就好

## 算法实现

#include <iostream>

#include <fstream>

#include <string>

#include <cmath>

#include <vector>

using namespace std;

int n = 0;

int k = 0;

int bestTime = 0;

vector<int> t;

vector<int> mech;

void readFile()

{

fstream in;

in.open("input.txt", ios::in);

if (!in)

{

return;

}

string temp;

getline(in, temp, ' ');

n = stoi(temp);

getline(in, temp);

k = stoi(temp);

t.resize(n + 1);

for (int i = 1; i < n; i++)

{

getline(in, temp, ' ');

t[i] = stoi(temp);

bestTime += t[i];

}

getline(in, temp);

t[n] = stoi(temp);

bestTime += t[n];

in.close();

}

int timeGet()

{

int maxtime = 0;

for (int i = 1; i <= k; i++)

{

maxtime = max(mech[i], maxtime);

}

return maxtime;

}

void backTrace(int level)

{

if (level == n + 1)//走到叶子节点

{

bestTime = min(timeGet(), bestTime);

return;

}

for (int i = 1; i <= k; i++)

{

mech[i] += t[level];//一直往一台机器里加任务

if (mech[i] < bestTime)//当前调度时间仍然最小于最优，可以继续向下

backTrace(level + 1);

mech[i] -= t[level];//撤销做法

}

}

void writeFile()

{

fstream in;

in.open("output.txt", ios::out);

if (!in)

{

cout << "error";

}

in << bestTime;

in.close();

}

int main()

{

readFile();

mech.resize(k + 1);//每台机器完成任务所需时间

backTrace(1);

writeFile();

}

## 结果分析

### 测试文档1：

**input.txt**

7 3

2 14 4 16 6 5 3

**output.txt**

17

### 测试文档2：

**input.txt**

8 9

1 2 3 4 5 1 6 6

**output.txt**

6