**Report For Scheduling Problem2**

09019204

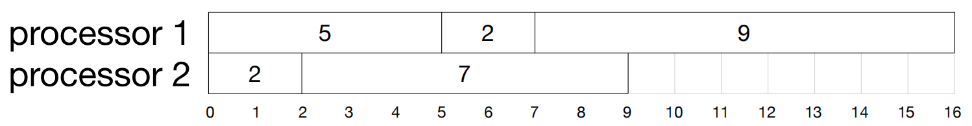
曹邹颖

1. **Problem description / demand analysis问题描述**

Given a set of *n* jobs. Each job i has processing time *pi*, all available at the beginning.

The goal is to assign jobs on two processors such that the maximum load is minimized.

Example of 5 jobs with processing times {5,2,7,2,9}



maximum load=16

1. Design a greedy algorithm to assign jobs to processors

2. Is this algorithm optimal? If it is not, find a counter example (bonus: find the

worst case of the algorithm)

3. Try different order of the inputs and run the greedy algorithm again

4. Design a dynamic programming algorithm to solve this problem

5. Suppose now we have 3 processors. Answer questions 1-4 again

1. **System structure / algorithm idea算法设计**
2. **basic idea基本思想**

**贪心算法：**对于输入的jobs序列，依次安排，查找当前load最小的processor让它处理下一个待处理的job。

**动态规划：**该工作安排问题可转化为将n个jobs安排给m个processor，使得这m个processor的load之差最小，则可以看作求jobs的一个子集，使得这个子集中的processing time和尽可能接近sum/m，其中sum为jobs的processing time总和。这样就可以使用动态规划的方式来求得其中的一个jobs子集，其重量(processing time)不超过sum/m，但重量达到最大。剩下几个子集可以将已安排好的jobs排除掉，processor个数减一，继续按此方式求解。

1. **system framework代码框架**

**贪心算法：**由题意，无需对输入的jobs序列进行排序，依次安排即可。主要代码核心便是查找当前load最小的processor，构造一个int findMinPtr(int\*, int);函数，传入记录每个processor已安排jobs的processing time总和，O(n)遍历，查找最大值返回下标k即可，然后让processor k处理下一个待处理的job。

**动态规划：**将每次安排的processor看作一个重量为sum/m的背包，将jobs看作物品，其processing time看作价值，从n个物品中选取若干个使得价值达到最大。从而该工作安排问题转化为做m-1次01背包问题，其代码核心即为状态转换方程：load[j] = max(load[j],load[j - times[i]] + times[i]);load[i]表示重量为i的背包选取物品放入的最大价值。安排jobs结果打印部分，用状态矩阵path[i][j]记录状态j下物品i是否被选中。path[i][j] = 1表示被选，初始化为全0。打印时，从最后一个状态往前找，这样就可以得到每个processor所安排的jobs序号将添加入每个对应的processor vector中记录。

1. **the functions and relationships of each module模块的功能和关系**

**贪心算法：**可以按序输入job的processing time时，便安排该job，找到当前load最小的processor，即可让它处理该job；

**动态规划：**安排的jobs需要在每个processor安排jobs部分记录下来，即当load[j] < load[j - times[i]] + times[i]时，置path[i][j]=1，而打印安排jobs结果部分是与每个processor安排jobs模块分开的。

1. **Function module design功能模块设计**
2. **function module design idea**

**贪心算法：**

for 所有待安排的job i

按序输入job i的processing time (times[i])

找到当前load最小的processor p

load[p] += times[i];让processor p加工job i

processor[p].push\_back(i)；记录该processor p处理的job

**动态规划：**

While m>1:

processor安排jobs部分(包含记录安排jobs结果)：

for 所有的未安排的job i

for j=sum/m... times[i]

if load[j] < load[j - times[i]] + times[i]:

load[j] = load[j - times[i]] + times[i];

path[i][j]=1

安排jobs结果打印部分

(用processor vector记录每个processor安排的jobs序号)：

初始i为当前待分配的jobs个数，j为sum/m；

while i，j均不为0

i--;

if path[i][j]==1: j -=time[i];processor.push\_back(i);

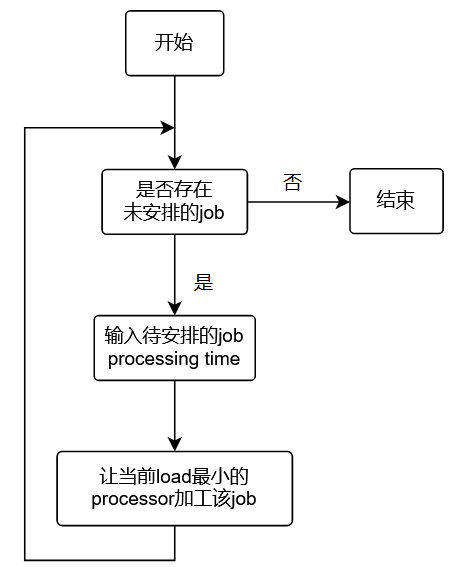
m--;

当m=1时，

将剩余未安排的工作都安排给剩下的最后一个processor

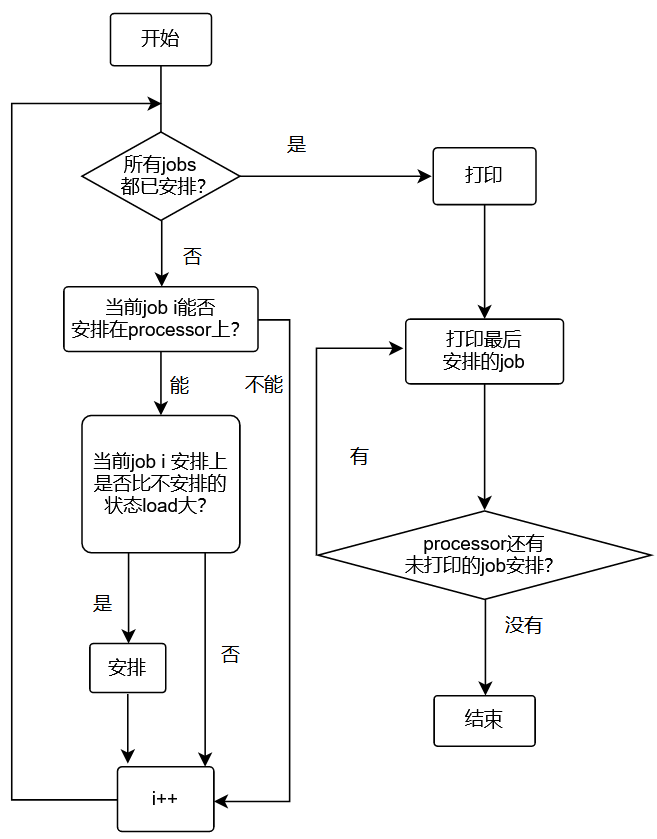
1. **flow chart**

**贪心算法：**



**动态规划：**

While m>1:（01背包流程图）



1. **algorithm complexity analysis**

**贪心算法：**O(mn)，m:processors个数，n:jobs个数；

**动态规划：**), :还剩个jobs未安排，:未安排工作的processors个数，:当前未安排jobs的processing times总和，sum:所有jobs的processing times总和。

1. **Test results and exercises answers测试结果和练习回答**
2. **test data selection or generation method测试数据选择/生成方法**

Data set

• E = {1,1,2,3,7,10,12}

• E = {5,8,3,1,8,20}

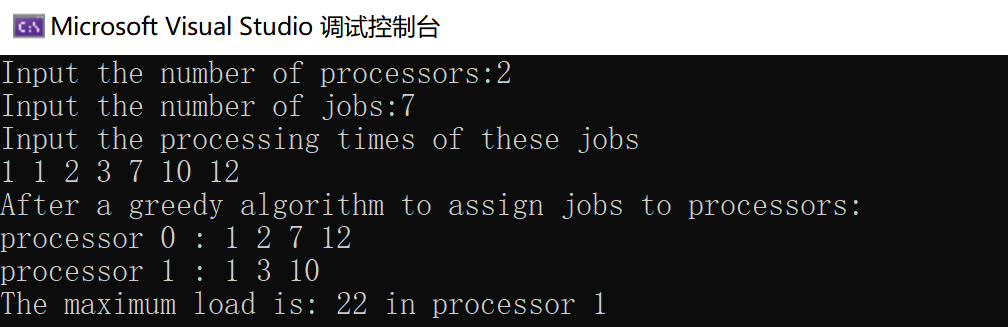
• E = {2,3,2,3,2}

1. **operation result screenshot运行结果截屏**

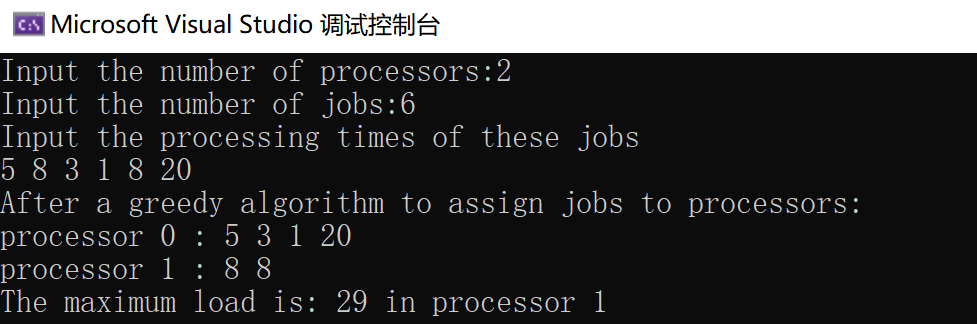
**贪心算法：**

m=2时，

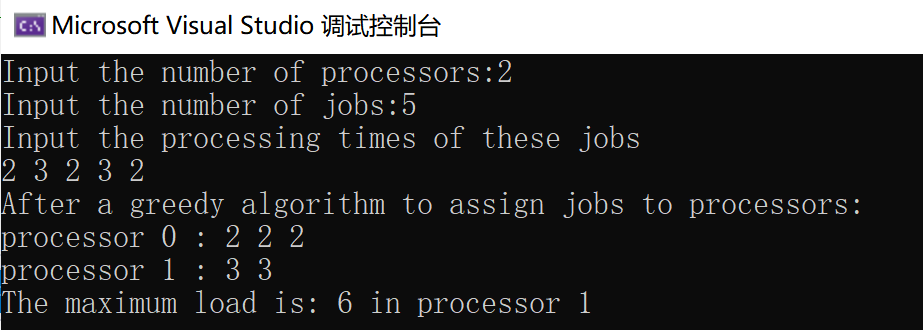
• E = {1,1,2,3,7,10,12}



• E = {5,8,3,1,8,20}

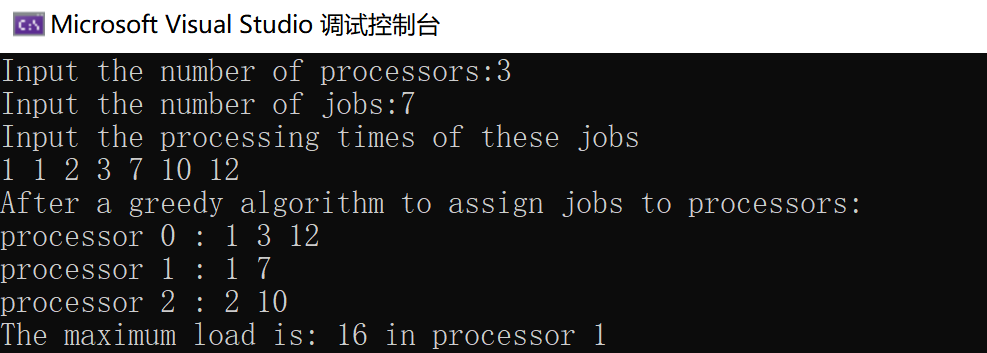


• E = {2,3,2,3,2}

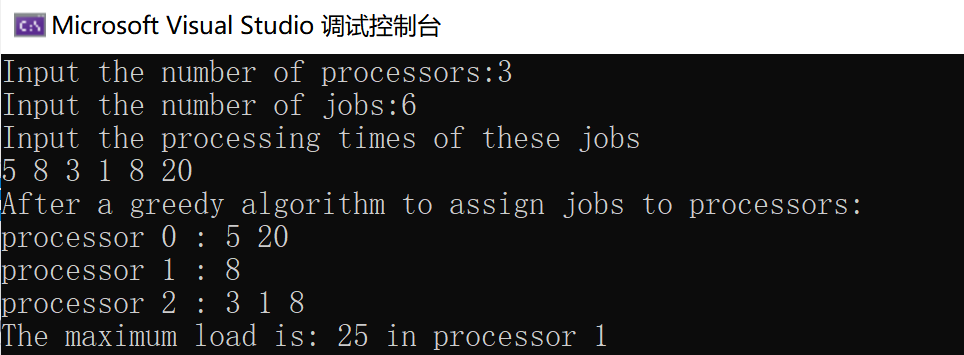


m=3时，

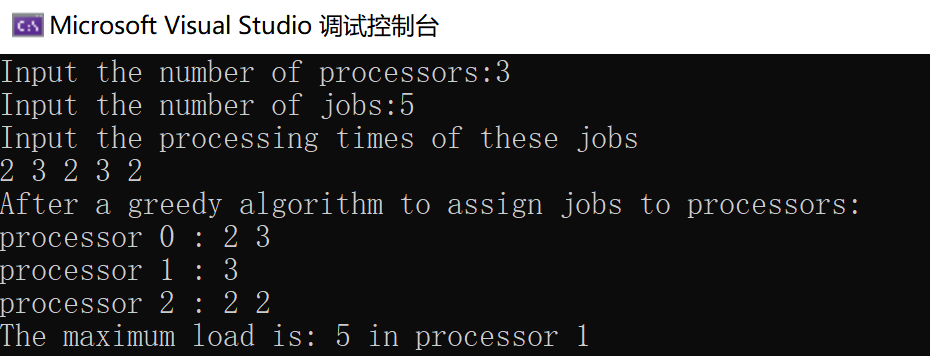
• E = {1,1,2,3,7,10,12}



• E = {5,8,3,1,8,20}



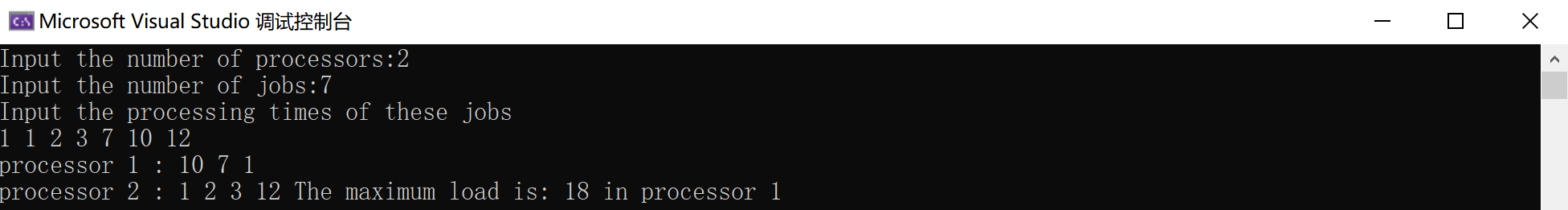
• E = {2,3,2,3,2}



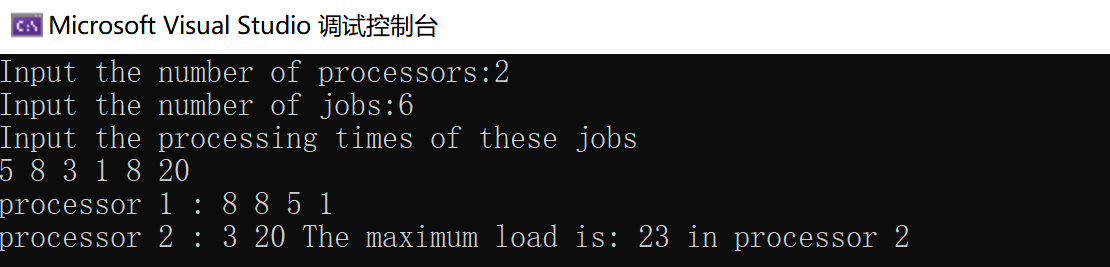
**动态规划：**

m=2时，

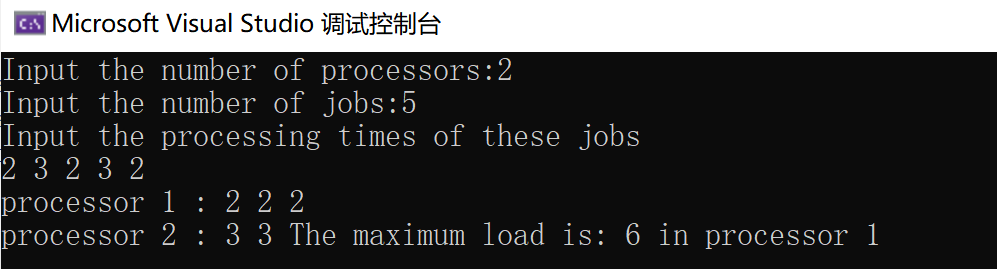
• E = {1,1,2,3,7,10,12}



• E = {5,8,3,1,8,20}

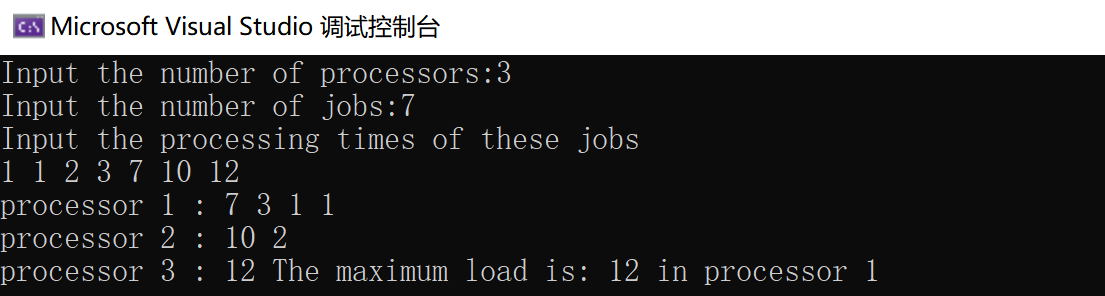


• E = {2,3,2,3,2}

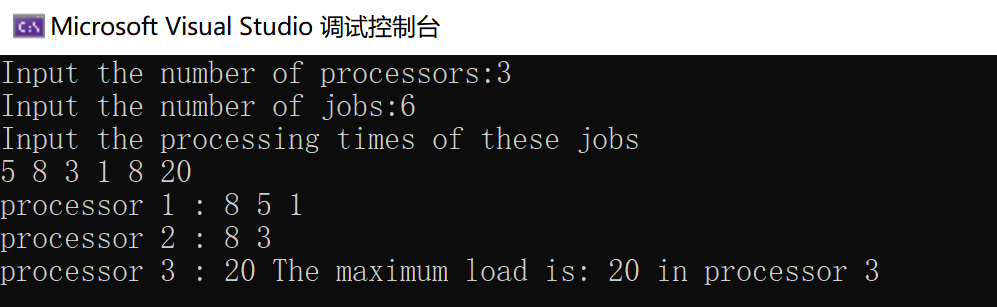


m=3时，

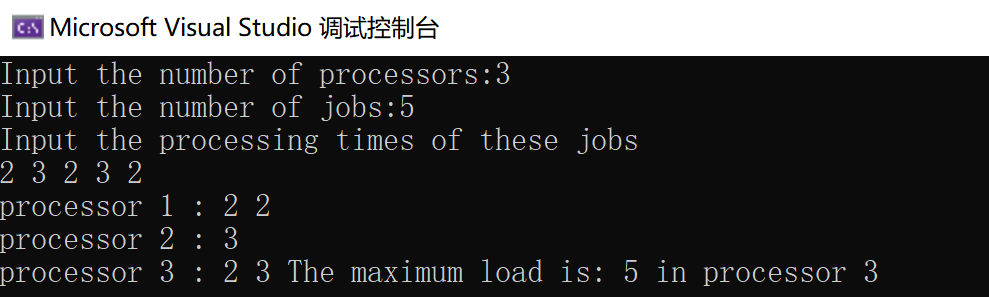
• E = {1,1,2,3,7,10,12}



• E = {5,8,3,1,8,20}



• E = {2,3,2,3,2}



1. **answers to the questions问题的回答**

**m=2时，**

1. 设计的贪心算法见Report 6.

2. 贪心算法解决这问题不是最优的。

反例：E = {1,1,2,3,7,10,12}

贪心算法求得：

Processor 1：1 2 7 12

Processor 2：1 3 10

maximum load=22

而最佳的解决方案为：

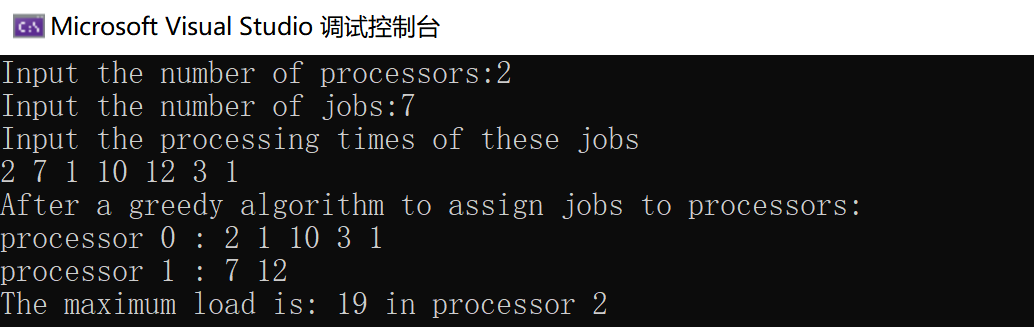
Processor 1：10 7 1

Processor 2：1 2 3 12

maximum load=18

3. 针对E = {1,1,2,3,7,10,12}，m=2

改变输入顺序为2 7 1 10 12 3 1，便得到不一样的解决方案如下：



4. 设计的动态规划算法见Report 6.

**m=3时，**

1. 设计的贪心算法见Report 6.

2. 贪心算法解决这问题不是最优的。

反例：

E = {1,1,2,3,7,10,12}

贪心算法求得：

Processor 1：1 3 12

Processor 2：1 7

Processor 3：2 10

maximum load=16

而最佳的解决方案为：

Processor 1：7 3 1 1

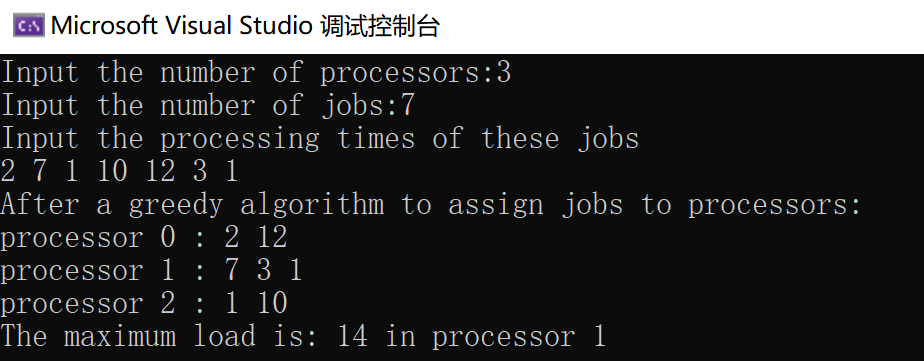
Processor 2：2 10

Processor 3：12

maximum load=12

1. 针对E = {1,1,2,3,7,10,12}，m=3

改变输入顺序为2 7 1 10 12 3 1，便得到不一样的解决方案如下：



4. 设计的动态规划算法见Report 6.

1. **Experimental summary实验总结**
2. **the problems encountered遇到的问题**

问题：

因为我设计的算法是从一般性出发，processor的个数不定，由键盘输入，那么如何设计动态规划算法来得到最佳工作安排方案是一个难题。

1. **the problem-solving process解决方案**

解决方案：

将每次安排的processor看作一个重量为sum/m的背包，将jobs看作物品，其processing time看作价值，从n个物品中选取若干个使得价值达到最大。从而该工作安排问题转化为做m-1次01背包问题，具体算法设计见Report 6.

1. **summarize the experimental experience 实验总结**

实验中，体会了贪心算法总是选择当前最优的，期望通过局部最优选择来得到一个全局最优解，但是对于某些问题来说，一些贪心的选择并不能得到最优解，只有考虑动态规划算法才能得出最佳解决方案。

1. **Source code源代码**

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int findMinPtr(int\*, int);

void greedAssign();

int main()

{

int m, n; // m:processors个数,n:jobs个数

cout << "Input the number of processors:"; cin >> m;

cout << "Input the number of jobs:"; cin >> n;

vector<int>times; // 记录processing times

cout << "Input the processing times of these jobs" << endl;

int temp, sum, max = 0, key = 0; // max:maximum load,key:最大负载 对应的processor(若多个选其一)

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

cin >> temp;

times.push\_back(temp);

}

temp = m; // 记录最初的processors个数

while (m>1) { // 等价：从n个物品中选取若干个， 其重量不超过sum/n，且重量达到最大

sum = 0; // 计算jobs耗时总和

n = times.size(); // 当前待分配的jobs个数

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

sum += times[i];

int\* load = new int[sum / m]; // load[i]:重量不超过i的物品放入 背包的最大利润

memset(load, 0, sizeof(int) \* (sum / m));

int\*\* path = new int\* [n]; // 该processor分配的job记录

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

path[i] = new int[sum / m];

memset(path[i], 0, sizeof(int) \* (sum / m));

}

sort(times.begin(), times.end()); // processing times从小到大排序

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

for (int j = sum / m; j >= times[i]; j--)

if (load[j] < load[j - times[i]] + times[i]) {

load[j] = load[j - times[i]] + times[i];

path[i][j] = 1;

}

}

int i = n, j = sum / m;

vector<int>processor; // 记录该processor安排的job序号

cout << "processor " << temp - m + 1 << " : ";

sum = 0;

while (j&&i) {

i--;

if (path[i][j] == 1) {

cout << times[i] << " "; // 打印安排的job processing times

j -= times[i];

sum += times[i]; // 记录load总和

processor.push\_back(i); // jobs中第i+1个被安排

}

}

cout << endl;

if (sum > max) { // 记录maximum load

max = sum;

key = temp - m + 1;

}

for (int i = 0; i < processor.size(); i++)

times.erase(times.begin() + processor[i]); // 将安排好的jobs剔除

m--; // 未安排的processor个数减一

}

cout << "processor " << temp << " : ";

sum = 0;

for (int i = 0; i < times.size(); i++) {

cout << times[i] << " ";

sum += times[i];

}

if (sum > max) { // 记录maximum load

max = sum;

key = temp;

}

cout << "The maximum load is: " << max << " in processor " << key << endl;

return 0;

}

void greedAssign()

{

int m,n; // m:processors个数,n:jobs个数

cout << "Input the number of processors:";cin >> m;

cout << "Input the number of jobs:"; cin >> n;

cout << "Input the processing times of these jobs" << endl;

int\* load = new int[m];

memset(load, 0, sizeof(int) \* m);

vector<vector<int> >processor; // 记录processor处理的job

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

processor.push\_back(vector<int>());

int\* times = new int[n];

int p;

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

cin >> times[i];

p = findMinPtr(load, m); // 找到当前load最小的processor p

load[p] += times[i]; // 贪心:让processor p加工该job

processor[p].push\_back(i); // 记录该processor p处理的job

}

cout << "After a greedy algorithm to assign jobs to processors:" << endl;

int max = 0, k = 0;

for (int i = 0; i < m; i++){

if (load[i] > max) // 记录maximum load

{

max = load[i];

k = i + 1;

}

cout << "processor " << i << " : ";

for (int j = 0; j < processor[i].size(); j++)

cout << times[processor[i][j]] << " ";

cout << endl;

}

cout << "The maximum load is: " << max <<" in processor "<< k<< endl;

delete[] load;

}

int findMinPtr(int\* a,int m) // 找到当前load最小的processor

{

int min = a[0], key = 0;

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

if (a[i] < min) {

min = a[i], key = i;

}

return key;

}