Report For Scheduling Problem2

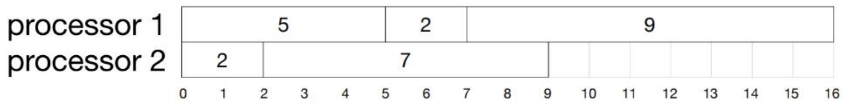
09019204
曹邹颖

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

2. System structure / algorithm idea 算法设计

(1) 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<u>动态规划</u>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。

(2) 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。

(3) 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。

3. Function module design 功能模块设计

(1) 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
```

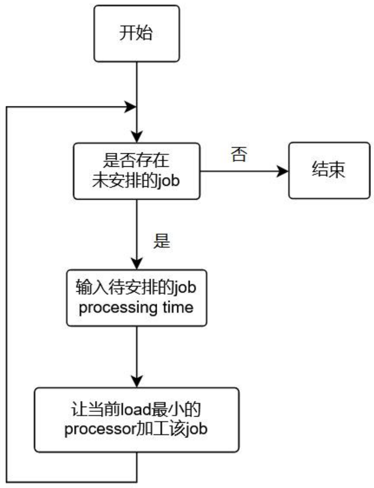
动态规划：

```
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
```

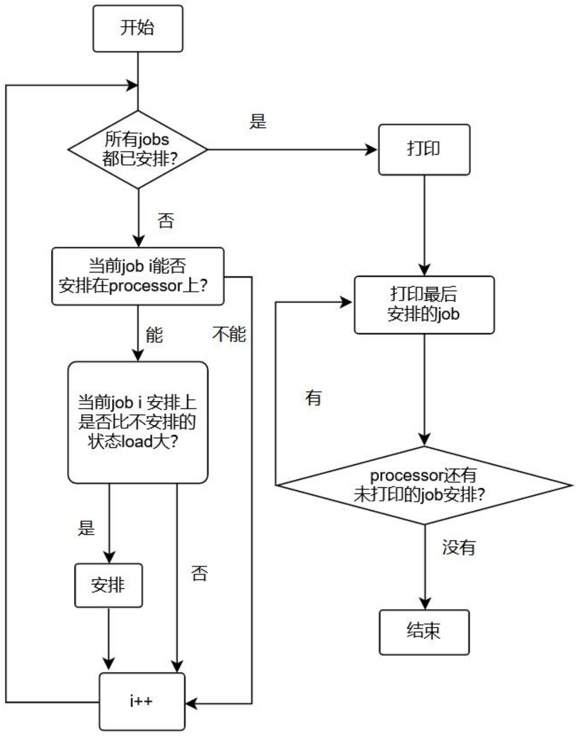
(2) flow chart

贪心算法：



动态规划：

While m>1:（01 背包流程图）



(3) algorithm complexity analysis

贪心算法：O(mn)，m:processors 个数，n:jobs 个数；

动态规划：$O(\sum_{i=1}^{i=m-1} n_i \cdot \frac{sum_i}{k_i}) = O(\frac{sum}{m} \cdot \sum_{i=1}^{i=m-1} n_i)$，$n_i$:还剩$n_i$个 jobs 未安排，$k_i$:未安排工作的 processors 个数，$sum_i$:当前未安排 jobs 的 processing times 总和，sum:所有 jobs 的 processing times 总和。

4. Test results and exercises answers 测试结果和练习回答

(1) 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}

(2) operation result screenshot 运行结果截屏

贪心算法：

m=2 时，

• E = {1,1,2,3,7,10,12}

Microsoft Visual Studio 调试控制台

```
Input the number of processors:2
Input the number of jobs:7
Input the processing times of these jobs
1 1 2 3 7 10 12
After a greedy algorithm to assign jobs to processors:
processor 0 : 1 2 7 12
processor 1 : 1 3 10
The maximum load is: 22 in processor 1
```

• E = {5,8,3,1,8,20}

Microsoft Visual Studio 调试控制台

```
Input the number of processors:2
Input the number of jobs:6
Input the processing times of these jobs
5 8 3 1 8 20
After a greedy algorithm to assign jobs to processors:
processor 0 : 5 3 1 20
processor 1 : 8 8
The maximum load is: 29 in processor 1
```

• E = {2,3,2,3,2}

Microsoft Visual Studio 调试控制台

```
Input the number of processors:2
Input the number of jobs:5
Input the processing times of these jobs
2 3 2 3 2
After a greedy algorithm to assign jobs to processors:
processor 0 : 2 2 2
processor 1 : 3 3
The maximum load is: 6 in processor 1
```

m=3 时，

• E = {1,1,2,3,7,10,12}

Microsoft Visual Studio 调试控制台

```
Input the number of processors:3
Input the number of jobs:7
Input the processing times of these jobs
1 1 2 3 7 10 12
After a greedy algorithm to assign jobs to processors:
processor 0 : 1 3 12
processor 1 : 1 7
processor 2 : 2 10
The maximum load is: 16 in processor 1
```

• E = {5,8,3,1,8,20}

Microsoft Visual Studio 调试控制台

```
Input the number of processors:3
Input the number of jobs:6
Input the processing times of these jobs
5 8 3 1 8 20
After a greedy algorithm to assign jobs to processors:
processor 0 : 5 20
processor 1 : 8
processor 2 : 3 1 8
The maximum load is: 25 in processor 1
```

• E = {2,3,2,3,2}

Microsoft Visual Studio 调试控制台

```
Input the number of processors:3
Input the number of jobs:5
Input the processing times of these jobs
2 3 2 3 2
After a greedy algorithm to assign jobs to processors:
processor 0 : 2 3
processor 1 : 3
processor 2 : 2 2
The maximum load is: 5 in processor 1
```

动态规划：

m=2 时，

• E = {1,1,2,3,7,10,12}

Microsoft Visual Studio 调试控制台

```
Input the number of processors:2
Input the number of jobs:7
Input the processing times of these jobs
1 1 2 3 7 10 12
processor 1 : 10 7 1
processor 2 : 1 2 3 12 The maximum load is: 18 in processor 1
```

• E = {5,8,3,1,8,20}

Microsoft Visual Studio 调试控制台

```
Input the number of processors:2
Input the number of jobs:6
Input the processing times of these jobs
5 8 3 1 8 20
processor 1 : 8 8 5 1
processor 2 : 3 20 The maximum load is: 23 in processor 2
```

• E = {2,3,2,3,2}

Microsoft Visual Studio 调试控制台

```
Input the number of processors:2
Input the number of jobs:5
Input the processing times of these jobs
2 3 2 3 2
processor 1 : 2 2 2
processor 2 : 3 3 The maximum load is: 6 in processor 1
```

m=3 时，

• E = {1,1,2,3,7,10,12}

Microsoft Visual Studio 调试控制台

```
Input the number of processors:3
Input the number of jobs:7
Input the processing times of these jobs
1 1 2 3 7 10 12
processor 1 : 7 3 1 1
processor 2 : 10 2
processor 3 : 12 The maximum load is: 12 in processor 1
```

• E = {5,8,3,1,8,20}

Microsoft Visual Studio 调试控制台

```
Input the number of processors:3
Input the number of jobs:6
Input the processing times of these jobs
5 8 3 1 8 20
processor 1 : 8 5 1
processor 2 : 8 3
processor 3 : 20 The maximum load is: 20 in processor 3
```

• E = {2,3,2,3,2}

Microsoft Visual Studio 调试控制台

```
Input the number of processors:3
Input the number of jobs:5
Input the processing times of these jobs
2 3 2 3 2
processor 1 : 2 2
processor 2 : 3
processor 3 : 2 3 The maximum load is: 5 in processor 3
```

(3) 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：

   Microsoft Visual Studio 调试控制台

```
Input the number of processors:2
Input the number of jobs:7
Input the processing times of these jobs
2 7 1 10 12 3 1
After a greedy algorithm to assign jobs to processors:
processor 0 : 2 1 10 3 1
processor 1 : 7 12
The maximum load is: 19 in processor 2
```

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

3. 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：

Microsoft Visual Studio 调试控制台

```
Input the number of processors:3
Input the number of jobs:7
Input the processing times of these jobs
2 7 1 10 12 3 1
After a greedy algorithm to assign jobs to processors:
processor 0 : 2 12
processor 1 : 7 3 1
processor 2 : 1 10
The maximum load is: 14 in processor 1
```

4. 设计的动态规划算法见 Report 6.

5. Experimental summary 实验总结

(1) 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。

(2) 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.

(3) 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。

6. 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;
}
```