P1065 [NOIP2006 提高组] 作业调度方案

题目描述

我们现在要利用m台机器加工n个工件，每个工件都有mm道工序，每道工序都在不同的指定的机器上完成。每个工件的每道工序都有指定的加工时间。

每个工件的每个工序称为一个操作，我们用记号j-k表示一个操作，其中j为1到n中的某个数字，为工件号；k为1到m中的某个数字，为工序号，例如2-4表示第2个工件第4道工序的这个操作。在本题中，我们还给定对于各操作的一个安排顺序。

例如，当n=3，m=2时，1-1,1-2,2-1,3-1,3-2,2-2 就是一个给定的安排顺序，即先安排第1个工件的第1个工序，再安排第1个工件的第2个工序，然后再安排第2个工件的第1个工序，等等。

一方面，每个操作的安排都要满足以下的两个约束条件。

1、对同一个工件，每道工序必须在它前面的工序完成后才能开始；

2、同一时刻每一台机器至多只能加工一个工件。

另一方面，在安排后面的操作时，不能改动前面已安排的操作的工作状态。

由于同一工件都是按工序的顺序安排的，因此，只按原顺序给出工件号，仍可得到同样的安排顺序，于是，在输入数据中，我们将这个安排顺序简写为“112332”。

还要注意，“安排顺序”只要求按照给定的顺序安排每个操作。不一定是各机器上的实际操作顺序。在具体实施时，有可能排在后面的某个操作比前面的某个操作先完成。

例如，取n=3,m=2，已知数据如下（机器号/加工时间）：

工件号	工序1	工序2
1	1/3	2/2
2	1/2	2/5
3	2/2	1/4

则对于安排顺序“112332”，下图中的两个实施方案都是正确的。但所需要的总时间分别是10与12。

当一个操作插入到某台机器的某个空档时（机器上最后的尚未安排操作的部分也可以看作一个空档），可以靠前插入，也可以靠后或居中插入。为了使问题简单一些，我们约定：在保证约束条件（1）（2）的条件下，尽量靠前插入。并且，我们还约定，如果有多个空档可以插入，就在保证约束条件（1）（2）的条件下，插入到最前面的一个空档。于是，在这些约定下，上例中的方案一是正确的，而方案二是不正确的。

显然，在这些约定下，对于给定的安排顺序，符合该安排顺序的实施方案是唯一的，请你计算出该方案完成全部任务所需的总时间。

输入格式

第1行为两个正整数 m，n，用一个空格隔开，（其中m(<20)表示机器数，n(<20)表示工件数）

第2行：m×n 个用空格隔开的数，为给定的安排顺序。

接下来的2n行，每行都是用空格隔开的m个正整数，每个数不超过20。

其中前n行依次表示每个工件的每个工序所使用的机器号，第1个数为第1个工序的机器号，第2个数为第2个工序机器号，等等。

后n行依次表示每个工件的每个工序的加工时间。

可以保证，以上各数据都是正确的，不必检验。

输出格式

1个正整数，为最少的加工时间。

输入样例

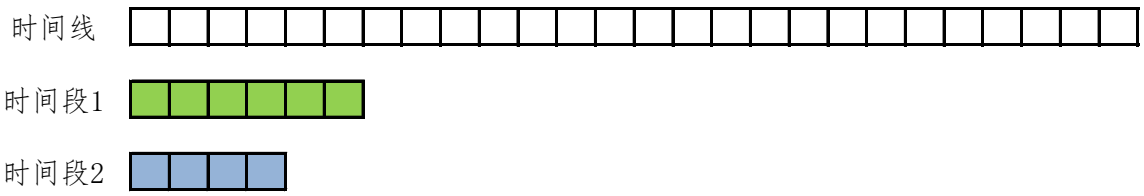
```
2 3
1 1 2 3 3 2
1 2
1 2
2 1
3 2
2 5
2 4
```

输出样例

```
10
```

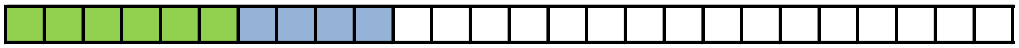
解析

本题的难点在于时间是如何插入的。现给定一段空的时间线，两段时间，我们该如何将其置入呢？如图所示：

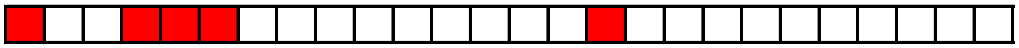


对于时间段1很容易，直接将其放入空时间线即可。那么时间线2呢？

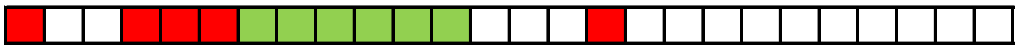
思路很简单，我们只要在时间线上开始搜索，直到找到第一个能容纳的下的即可。

时间线 

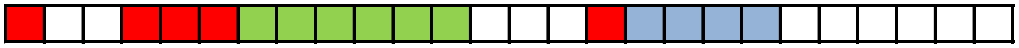
现在我们开始将初始的时间线变得混乱一些，看看如何将时间段1进行放置。其中红色代表障碍点。

时间线 

很明显，我们应该将时间段1存入如下位置：

时间线 

继续在时间线上搜索空余空间，将时间段2置入。

时间线 

通过以上操作，我们可以把一条时间线推广到多条时间线上。每一条时间线就是一个机器。另外一个问题就是时间段的起点，这里我们只需要保证后一个时间段的起点不小于前一个时间段的终点即可。

编码

```
#include<bits/stdc++.h>

using namespace std;

const int MAX = 21;

struct item {//工件
    int number[MAX];//每个工序对应的机器号
    int time[MAX];//每个工序对应的加工时间
    int order;//下一个将执行的工序
    int start;//下一个将执行的工序的最早开始时间
};
int arrange[400];
item items[MAX];//工件数组
bool machines[MAX][100005];//m个机器，及对应的时间安排

//机器号，耗时，起始时间
int find(int m, int t, int start) {
     int sum = 0;//有空闲的时间格数
    int i = start;//当前访问的时间格
    while (sum < t) {
         //判断m号机器的第i时间作为起点是否可行
        if (!machines[m][i]) {
             sum++;
         } else {
             sum = 0;
         }
         i++;
```

```
    }
    //到达时间以后，我们将所有格子进行填充
    int res = i;
    for (i = i - 1; sum > 0; i--) {
        machines[m][i] = true;
        sum--;
    }
    //返回最大时间
    return res;
}

int solve(int n) {
    //n为工件序号
    int k = items[n].order;//k为该工件现在要执行的工序
    int m = items[n].number[k];//m为该工件现在要执行的工序，所对应的机器号
    int t = items[n].time[k];//t为该工件现在要执行的工序，所对应的加工时间

    int res = find(m, t, items[n].start);//找到空挡并执行
    items[n].order++;//准备执行下一个工序
    items[n].start = res;
    return res;
}

int main() {
    int m, n;
    cin >> m >> n;//m：机器数/每个工件的工序数；n：工件数
    for (int i = 0; i < m * n; i++) {//记录安排顺序
       cin >> arrange[i];
    }
    for (int i = 1; i <= n; i++) {
        items[i].order = 1;
        items[i].start = 0;
        for (int j = 1; j <= m; j++) {
            cin >> items[i].number[j];
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> items[i].time[j];
        }
    }
    //录入数据结束
    //开始处理安排
    int res = 0;
    for (int i = 0; i < m * n; i++) {
        res = max(solve(arrange[i]), res);
    }
    cout << res;
    return 0;
```

```
}
```