# CSP2019-S 模拟题（第二试）

（请选手务必仔细阅读此页内容）

## 一．题目概况

| 中文题目名称 | 矩阵 | 序列 | 二分图 |
|---|---|---|---|
| 英文题目名称 | matrix | sequence | grape |
| 可执行文件名 | matrix | sequence | grape |
| 输入文件名 | matrix.in | sequence.in | grape.in |
| 输出文件名 | matrix.out | sequence.out | grape.out |
| 每个测试点时限 | 1 秒 | 1 秒 | 1 秒 |
| 内存上限 | 256MB | 256MB | 256MB |
| 测试点数目 | 10 | 20 | 20 |
| 每个测试点分值 | 10 | 5 | 5 |
| 结果比较方式 | 全文比较（过滤行末空格及文末回车） | | |
| 题目类型 | 传统 | 传统 | 传统 |

## 二．提交源程序文件名

| 对于 Pascal 语言 | matrix.pas | sequence.pas | grape.pas |
|---|---|---|---|
| 对于 C 语言 | matrix.c | sequence.c | grape.c |
| 对于 C++语言 | matrix.cpp | sequence.cpp | grape.cpp |

## 三．编译命令

| 对于 Pascal 语言 | fpc matrix.pas | fpc sequence.pas | fpc grape.pas |
|---|---|---|---|
| 对于 C 语言 | gcc -o matrix matrix.c -lm | gcc -o sequence sequence.c –lm | gcc -o grape grape.c –lm |
| 对于 C++语言 | g++ -o matrix matrix.cpp -lm | g++ -o sequence sequence.cpp –lm | g++ -o grape grape.cpp -lm |

## 四．注意事项：

1． 文件名（程序名和输入输出文件名）必须使用小写。

2． 选手提交以自己编号命名的文件夹，文件夹内包含 3 个源文件(.c, .cpp, .pas,)，并在文件夹下建立 3 个相应的子目录，并将 3 个对应的源程序分别放入对应的子文件夹中，所有名字必须使用小写；例如：

```
001          编号
|---matrix
      |---matrix.cpp
|---sequence
      |---sequence.cpp
|---grape
      |---grape.cpp
|---matrix.cpp
|---sequence.cpp
|---grape.cpp
```

3. C/C++中函数 main()的返回值类型必须是 int，程序正常结束时的返回值必须是 0。
4. 题目简单，请认真对待，争取三位数.
5. 每道题源代码长度限制均为 50KB。
6. 每道题的数据都有一定梯度。请尽量优化算法，争取拿高分。
7. 编译时不打开任何优化选项。
8. 建议最后 10 分钟不要再编程，检查一下提交的文件夹中的代码是否符合要求，检查文件名，输入输出文件名，数据类型，数据精度，空间限制，赋初值等是否按试卷上的要求来做的，一定要杜绝一切的不小心的人为错误，显然这种错误是致命的。
9. 做题时，审题是关键，必须深入与全面，学过的知识与做过的题都是分析问题的有利武器；编写代码要细致，多写函数，便于调试，只有这样，才能达到你的期望。
10. 不管第一天考得如何，你再去想，已经没用了，希望的就是现在，所以必须静下心来全力去完成好今天的考试，发挥出自己的水平，成功的期望值就是最大，因为这是一个系列赛，是两天考试的总和。

# 必须杜绝文件操作错误！

# 1. 矩阵

## (matrix.pas/c/cpp)

**【问题描述】**

小 $w$ 是个热爱数学的好少年。最近他上了几堂线性代数课，便深深地爱上了矩阵。他仔细研究了矩阵后，被矩阵乘法优美的性质所折服。在接下来的几天里，他找出了许许多多的矩阵乘法题，统统刷掉。

所谓矩阵乘法，就是把一个 $n$ 行 $r$ 列的矩阵 $A$ 和一个 $r$ 行 $m$ 列的矩阵 $B$，相乘后得到一个 $n$ 行 $m$ 列的矩阵 $C$，若对于矩阵 $N$，$N_{i,j}$表示矩阵 $N$ 中第 $i$ 行第 $j$ 列的元素，则有：

$$C_{i,j} = \sum_{k=1}^{r} A_{i,k} \times B_{k,j}$$

众所周知，矩阵乘法的复杂度是 $O(nmr)$的。但是，小 $w$ 在刷了大量的题目之后，决定优化它。于是，他构造了两个 $n*n$ 的矩阵 $A$ 和 $B$，并试图计算出这两个矩阵相乘所得到的矩阵 $C$。由于 $n$ 大了点，他发现自己不会做了，就将问题改为了每次询问矩阵 $C$ 中一个子矩阵的权值和，共询问 $m$ 次。但是，他还是不会做。你能帮帮他吗？

**【输入】**

输入文件 matrix.in

第一行两个数 $n,m$ 表示矩阵大小为 $n$，共有 $m$ 次询问。

接下来 $2*n$ 行描述两个 $n*n$ 的矩阵，按由第 1 行到第 $n$ 行、第 1 列到第 $n$ 列的顺序输入。

接下来 $m$ 行，每行四个数 $a,b,c,d$，表示询问矩阵 $C$ 中以第 $a$、c 行为上下边界，以 $b$、$d$ 为左右边界的子矩阵权值和。

**【输出】**

对于每次询问，输出该子矩阵权值和。

**【输入输出样例】**

| matrix.in | matrix.out |
|---|---|
| 3 2<br>1 9 8<br>3 2 0<br>1 8 3<br>9 8 4<br>0 5 15<br>1 9 6<br>1 1 3 3<br>2 3 1 2 | 661<br>388 |

**【输入输出样例说明】**

相乘得到的矩阵为：
17 125 187
27 34 42

12 75 142

**【数据说明】**

读入数据较大，请优化读入方式。

保证矩阵中的每个元素为不超过 100 的自然数。

数据范围如下表所示：

| 测试点编号 | $n \leq$ | $m \leq$ | 测试点编号 | $n \leq$ | $m \leq$ |
|---|---|---|---|---|---|
| 1 | 200 | 200 | 6 | 800 | 10 |
| 2 | 500 | 500 | 7 | 800 | 1000 |
| 3 | 500 | 500 | 8 | 800 | 5000 |
| 4 | 800 | 1 | 9 | 800 | 10000 |
| 5 | 800 | 5 | 10 | 800 | 10000 |

# 2. 序列

## (sequence.pas/c/cpp)

**【问题描述】**

小 $w$ 是个热爱数学的好少年。最近他学会了求最大连续子段和和多种方法，形如线段树、动态规划、分治等。小 $w$ 仍不满足，他还想求出第 $k$ 大的连续子段和，但是他发现自己不会做了。你能帮帮他吗？

**【输入】**

输入文件 sequence.in

第一行两个数 $n,k$，表示序列长度为 $n$，你需要求第 $k$ 大的连续区间和。

**注意：空序列不被计算在内！**

**【输出】**

仅一个数，表示第 $k$ 大的连续区间和。

**【输入输出样例 1】**

| sequence.in | sequence.out |
|---|---|
| 3 4<br>1 4 2 | 4 |

**【输入输出样例说明】**

最大子段和为 7，次大子段和为 6，第三大子段和为 5，第四大子段和为 4。

**【输入输出样例 2】**

| sequence.in | sequence.out |
|---|---|
| 10 4<br>-81 90 -76 66 -47 59 37 -60 97 60 | 166 |

**【数据说明】**

| 测试点编号 | $n \leq$ | 数据特点 | 测试点编号 | $n \leq$ | 数据特点 |
|---|---|---|---|---|---|
| 1 | 10 | $k \leq \frac{n(n+1)}{2}$ | 11 | 100000 | $k \leq \frac{n(n+1)}{2}$ |
| 2 | 100 | | 12 | | |
| 3 | 1000 | | 13 | | |
| 4 | | | 14 | | |
| 5 | | | 15 | | |
| 6 | | $k = 1$ | 16 | | |
| 7 | 10000 | | 17 | | |
| 8 | | | 18 | | |
| 9 | | $k = 10$ | 19 | | |
| 10 | | | 20 | | |

# 3. 二分图

## (grape.pas/c/cpp)

**【问题描述】**

小 $e$ 是一名热爱学习的 $oier$。最近他学习了一个算法，叫做匈牙利算法。他觉得这个算法非常神奇，因为它可以在 $O(nm)$的时间内求出一张二分图的最大匹配。于是，小 $e$ 开心地写了许多二分图的题目。

然后，神犇 $lg$ 路过，就顺手给小 $e$ 出了一道题：给出一张无向图，求这张图的最大匹配。即选出尽量多的边，使得任意两条边没有公共点。虽然这不是一张二分图，但是小 $e$ 还是写了匈牙利算法。当然，他发现自己 $Wa$ 了。

神犇 $lg$ 发现小 $e$ 居然用错误的算法过掉了一些点，非常生气，觉得自己的题目出水了。于是，给图中的每条边增加了两个值 $start,end$，表示这条边在时刻$(start,end]$中会出现在图中。问题也变为了求每个时刻图的最大匹配。

到这里小 $e$ 已经不会做了，但是他还是会用匈牙利算法打暴力，对每个时刻的图跑一遍匈牙利算法。$lg$ 知道，匈牙利算法只有在图是二分图的情况下得到的才一定是正确的结果。于是，他想知道，这张图在哪些时刻是二分图。

这个问题 $lg$ 当然会做了，但是这么简单的问题他也不屑于去写。于是，这个任务就落到你头上了。

注：若一个图为二分图，则存在一种染色方案，使得每个点不是黑色就是白色，并且每条边的两个端点颜色不同。

**【输入】**

输入文件 grape.in

第一行三个数 $n,m,T$，表示这张图有 $n$ 个点，$m$ 条边，共有 $T$ 个时刻，即最大时刻$\leq T$。

接下来 $m$ 行，每行四个数 $u,v,start,end$，表示这条边连接 $u$ 个 $v$，在时刻$(start,end]$中出现。

**【输出】**

共 $T$ 行。若时刻 $i$ 这张图是二分图，则在第 $i$ 行输出 $Yes$；否则在第 $i$ 行输出 $No$。

**【输入输出样例】**

| grape.in | grape.out |
|---|---|
| 3 3 3<br>1 2 0 2<br>2 3 0 3<br>1 3 1 2 | Yes<br>No<br>Yes |

**【输入输出样例说明】**

在 1 时刻，有两条边(1,2),(2,3)，是二分图；

在 2 时刻，多了一条边(1,3)，不是二分图；

在 3 时刻，只剩一条边(2,3)，是二分图。

**【数据说明】**

注意：可能有重边和自环。

| 测试点 | $n \leq$ | $m \leq$ | $T \leq$ | 测试点 | $n \leq$ | $m \leq$ | $T \leq$ |
|---|---|---|---|---|---|---|---|
| 1 | 10 | 10 | 10 | 11 | 10000 | 20000 | 50000 |
| 2 | 50 | 50 | 50 | 12 | 20000 | 50000 | 50000 |
| 3 | 100 | 100 | 100 | 13 | 50000 | 100000 | 100000 |
| 4 | 1000 | 1000 | 1000 | 14 | 100000 | 200000 | 100000 |
| 5 | 1000 | 1000 | 1000 | 15 | 100000 | 200000 | 150000 |
| 6 | 1000 | 1000 | 1500 | 16 | 150000 | 200000 | 100000 |
| 7 | 5000 | 10000 | 10000 | 17 | 150000 | 300000 | 100000 |
| 8 | 10000 | 10000 | 10000 | 18 | 150000 | 300000 | 200000 |
| 9 | 10000 | 20000 | 10000 | 19 | 200000 | 400000 | 200000 |
| 10 | 20000 | 20000 | 20000 | 20 | 200000 | 400000 | 200000 |

注：请注意优化常数