NOIP提高组复赛模拟（简单）

一、题目概况

|  |  |  |  |
| --- | --- | --- | --- |
| 中文题目名称 | 新的世界 | 邻面合并 | 光线追踪 |
| 英文题目与子目录名 | neworld | merging | raytracing |
| 可执行文件名 | neworld | merging | raytracing |
| 输入文件名 | neworld.in | merging.in | raytracing.in |
| 输出文件名 | neworld.out | merging.out | raytracing.out |
| 每个测试点时限 | 1秒 | 2秒 | 2秒 |
| 测试点数目 | 10 | 10 | 10 |
| 每个测试点分值 | 10 | 10 | 10 |
| 附加样例文件 | 无 | 无 | 有 |
| 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 |
| 运行内存上限 | 256M | 256M | 256M |

二、提交源程序文件名

|  |  |  |  |
| --- | --- | --- | --- |
| 对于C++ 语言 | neworld.cpp | merging.cpp | raytracing.cpp |
| 对于C语言 | neworld.c | merging.c | raytracing.c |
| 对于Pascal语言 | neworld.pas | merging.pas | raytracing.pas |

三、编译命令（不包含任何优化开关）

|  |  |  |  |
| --- | --- | --- | --- |
| 对于C++ 语言 | g++ -o neworld neworld.cpp -lm | g++ -o merging merging.cpp -lm | g++ -o raytracing raytracing.cpp -lm |
| 对于C语言 | gcc -o neworld neworld.c -lm | gcc -o merging merging.c -lm | gcc -o raytracing raytracing.c -lm |
| 对于Pascal语言 | fpc neworld.pas | fpc merging.pas | fpc raytracing.pas |

注意事项：

1、文件名（程序名和输入输出文件名）必须使用英文小写。

2、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。

3、只提供 Linux 格式附加样例文件。

4、提交的程序代码文件的放置位置请参照各省的具体要求。

5、特别提醒：评测在当前最新公布的 NOI Linux 下进行，各语言的编译器版本以其为准。

[BGM: With An Orchid - Yanni]

1. **新的世界**

**(neworld.cpp/c/pas)**

**【题目背景】**

小学五六年级的乔猫是一个喜欢不务正业写游戏的孩纸……他曾经模仿著名的沙盒游戏《Minecraft》做过一个自己的游戏“NEWorld”。这两个游戏有着相同的规则，都是通过在一个满是方块组成的3D世界中，放置不同的方块来建造各种各样的东西。对了，游戏中还有一个独特的“近似全局光照”的亮度系统……为了简单，我们只考虑二维的情况吧。

**【问题描述】**

在一个行列的网格中，第行列的格子有一个可变的“亮度” （初始时都为0）和一个固定的“不透光度”。现在在行列放入一个亮度为的光源，NEWorld游戏引擎会根据以下逻辑，让光源逐步“照亮”附近的方格：

先将光源所在方格的亮度赋值为。而对于行列一个**不是光源**的方格，它的亮度由和四周方格的亮度所确定。定义（**此处当不成立或不成立时，被看作是0**），我们称方格的亮度 是“有效”的，当且仅当 。显然初始时所有亮度都是“有效”的，而放入光源后则可能存在亮度“无效”的方格。

现在引擎会循环执行操作，每一步找出当前所有亮度“无效”（不包括光源）的方格中，行数最小的那一个（如果有多个行数最小的，就选择其中列数最小的方格），然后计算的值，将其赋值给。操作会不停地执行，直到所有亮度都“有效”为止（请参考样例，循环一定会在有限步操作后结束）。请问最后行列的方格亮度值是多少？

注：表示取中最大的值。

**【输入格式】**

输入文件名为neworld.in。

输入文件第一行两个正整数，表示网格大小为行列。

接下来的行，每行个正整数，其中第行列的正整数为。

最后一行包含五个正整数，表示在行列放入亮度为的光源，需要查询的是亮度计算完成后行列的亮度值。

**【输出格式】**

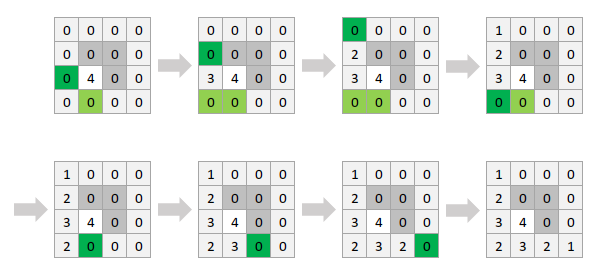
输出文件名为neworld.out。

输出文件包含一行一个正整数，表示最后的值。

**【输入输出样例】**

|  |  |
| --- | --- |
| **neworld.in** | **neworld.out** |
| 4 4  1 1 1 1  1 4 4 1  1 1 4 1  1 1 1 1  3 2 4 1 1 | 1 |

**【输入输出样例说明】**

这张图展示了亮度重新计算的过程。数字表示方格亮度，白色方格为光源，灰色方格的灰度表示该方格的不透光度（浅灰为1深灰为4），绿色方格是亮度“无效”的方格，其中深绿色是即将被重新计算亮度的方格。

**【数据规模与约定】**

对于60% 的数据：。

对于100% 的数据：。

1. **邻面合并**

**(merging.cpp/c/pas)**

**【题目背景】**

NEWorld作为一个3D游戏，对渲染（图形绘制）的效率要求极高。当玩家扩大视野范围时，可见的方块面数量将会迅速增多，以至于大量的顶点处理很快就成为了图形管线中的瓶颈。乔猫想了想，决定在大量绘制前，预处理一些相邻且有着相同材质的方块面——将许多小的面合成一个大的面，便可以在不改变渲染结果的同时减少很多顶点数量了吧……

**【问题描述】**

给定一个的网格，每个格子上写有0或1。现在用一些长方形覆盖其中写有1的格子，长方形的每条边都要与坐标轴平行。要求：每个写着1的格子都要被覆盖，长方形不可以重叠（重复绘制也多少会增加性能开销），也不能覆盖到任何一个写着0的格子（不然绘制结果就不正确了）。请问最少需要多少长方形？

**【输入格式】**

输入文件名为merging.in。

输入文件第一行两个正整数，表示网格大小为行列。

接下来的行，每行个正整数（保证均为0或1），其中第行列的正整数表示网格行列里填的数。

**【输出格式】**

输出文件名为merging.out。

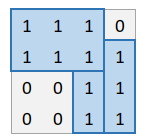
输出文件包含一行一个正整数，表示最少需要的长方形数量。

**【输入输出样例】**

|  |  |
| --- | --- |
| **merging.in** | **merging.out** |
| 4 4  1 1 1 0  1 1 1 1  0 0 1 1  0 0 1 1 | 3 |

**【输入输出样例说明】**

一种可行的覆盖方案（粗线表示分割线）：



**【数据规模与约定】**

对于30% 的数据：。

对于100% 的数据：。

1. **光线追踪**

**(raytracing.cpp/c/pas)**

**【题目背景】**

初中时的乔猫试着组建了NEWorld开发组，可是不久之后却因为合作上的问题（和乔猫工程水平差，代码混乱的问题），开发组成员之间常常产生矛盾，关系越来越不如以前……一年下来，受到长期挫折的乔猫最终放弃了NEWorld，决定在信息竞赛方面努力奋斗……

又是一年过去，上了高中的乔猫突发奇想，决定自己尝试写一个基于八叉树BVH（空间细分）的光线追踪渲染器。为了向自己的中二时代致敬，渲染的模型也是一个“方块组成的世界”……同样，为了简化，这里只考虑二维的情况……（貌似简化太多了吧233）

**【问题描述】**

考虑一个二维平面，摄像机在的位置，初始时平面上没有障碍物。现在执行次操作，操作有两种（假设这是第次操作，）：

1. 给定，创建一个每条边与坐标轴平行的长方形障碍物，包含所有满足且的点（如果这个区域的某一部分已经存在障碍，则直接覆盖掉它，具体请看样例）。这个障碍物的编号为。
2. 给定向量，会有一个动点从摄像机所在的位置出发，以所指的方向前进，直到碰到第一个障碍物为止。

对于第2种操作，输出最先碰到的障碍物的编号。若不会碰到任何障碍物，输出0。

**【输入格式】**

输入文件名为raytracing.in。

输入文件第一行一个正整数，表示操作总数。

接下来的行，每行第一个正整数为操作种类（保证为1或2）。如果为1，则接下来四个正整数表示障碍的位置；如果为2，则接下来两个正整数表示前进方向。

**【输出格式】**

输出文件名为raytracing.out。

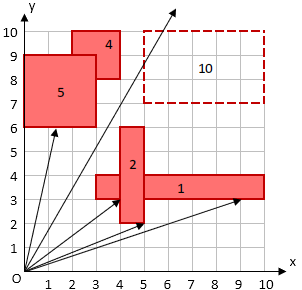
输出文件包含行（为第2种操作的总数），每行一个正整数，表示第一个碰到的障碍物编号。

**【输入输出样例1】**

|  |  |
| --- | --- |
| **raytracing.in** | **raytracing.out** |
| 10  1 3 3 10 4  1 4 2 5 6  2 6 2  1 2 8 4 10  1 0 6 3 9  2 5 2  2 8 6  2 2 9  2 4 7  1 5 7 10 10 | 1  2  2  5  0 |

**【输入输出样例1说明】**

在9次操作之后，平面的一部分如图所示（箭头为所有第2种操作询问的路线）。

****

**【输入输出样例2】**见选手目录下的

raytracing/raytracing2.in和raytracing/raytracing2.ans。

**【数据规模与约定】**

对于30% 的数据：。

对于另外30% 的数据：。

对于100% 的数据：和和