【2025SD省队第三轮(济南)集训第2场】

一、题目概览

中文题目名称	哩哩哩啦哩啦	邦霸迪咯克洛克迪咯	通通通萨湖啦
英文题目名称	lirililarila	bombardirocrocodilo	tungtungtungsahur
可执行文件名	lirililarila	bombardirocrocodilo.exe	tungtungtungsahur.exe
输入文件名	lirililarila.in	bombardirocrocodilo.in	tungtungtungsahur.in
输出文件名	lirililarila.out	bombardirocrocodilo.out	tungtungtungsahur.out
时间限制	2s	8s	1s
空间限制	512MB	512MB	512MB
测试点数目	子任务	子任务	1
测试点分值	子任务	子任务	100
题目类型	传统	传统	传统
比较方式	SPJ	全文比较	SPJ
是否有部分分	是	是	是

二、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.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.评测环境为Windows，使用lemon进行评测。

4.选手不得使用SSH等命令。

5.选手不得使用内嵌汇编，#pragma等指令。

6.评测时使用 C++17 环境，同时开启 O2 优化，栈空间和内存限制相同。

A. 哩哩哩啦哩啦（lirililarila）

题目描述

某池塘共有 n 只青蛙，每只青蛙都有一个唯一的编号，编号从 1 到 n。编号为 1 的青蛙是池塘总呱呱，除了总呱呱之外，每只青蛙都有一个直接妈妈，青蛙 i 的直接妈妈编号为 p_i，且有 $p_i < i$。

若 $p_x = y$，则青蛙 x 被称为青蛙 y 的第 1 层蝌蚪；若青蛙 p_x 是青蛙 y 的第 $k-1$ 层蝌蚪，则青蛙 x 被称为青蛙 y 的第 k 层蝌蚪。

现在，公司总呱呱可以选择一些青蛙参加火锅。他决定选择两个数字 L 和 R，将所有编号满足 $L \leq i \leq R$ 的青蛙送去火锅。

在确定 L 和 R 之前，总呱呱收到了 m 条青蛙的要求，第 j 条要求用两个数字 u_j 和 k_j 表示，这代表青蛙 u_j 希望其第 k_j 层蝌蚪中**至少有一只**被送去火锅。为了节省费用，总呱呱希望确定 L 和 R 的值使得送去培训的青蛙蛙数最少，并且所有青蛙的要求都能满足。

需要编写程序，根据池塘内部的母子级关系和青蛙的要求，找出一对 L 和 R，使得所有要求都能得到满足，并且送去火锅的青蛙蛙数最少。如果有多个满足条件的 (L, R) 对，选择其中 L 最小的那个。

输入格式

第一行输入一个整数 n，表示池塘青蛙的总数（$2 \leq n \leq 200000$）。

第二行输入 $n-1$ 个整数 $p_2, p_3, \ldots, p_n$，其中 p_i 表示青蛙 i 的直接妈妈编号，且 $1 \leq p_i < i$。

第三行输入一个整数 m，表示青蛙的要求数量。

接下来 m 行，每行包含两个整数 u_j 和 k_j，表示第 j 条要求。

输出格式

输出两个整数 L 和 R，表示选择的青蛙编号区间。如果存在多个符合条件的 (L, R) 对，输出其中 L 最小的那一对。

输入输出样例 #1

输入 #1

```
7
1 1 2 2 3 3
3
1 1
3 1
1 2
```

输出 #1

```
3 6
```

说明/提示

样例解释

青蛙编号为 $3, 4, 5, 6$ 的青蛙将被送去火锅。这满足所有要求，因为青蛙 3 是青蛙 1 的第 1 层蝌蚪，青蛙 4 是青蛙 1 的第 2 层蝌蚪，青蛙 6 是青蛙 3 的第 1 层蝌蚪。

数据范围

子任务	分值	$2 \leq n \leq$	$1 \leq m \leq$	其它特殊性质
1	19	50	50	
2	25	3000	3000	
3	21	200000	200000	$p_i = i - 1$
4	35	200000	200000	

B. 邦霸迪咯克洛克迪咯（bombardirocrocodilo)

题目描述

给定 n 个点 (x_i, y_i, c_i)，$i = 1, 2, \ldots, n$，共有 m 次查询操作，每次查询给定 A, B, C，问满足 $Ax_i + By_i + C < 0$，$Ax_j + By_j + C < 0$，$c_i = c_j$ 的二元组 (i, j) 的个数。

输入格式

第一行两个整数 n m;

接下来 n 行，每行三个整数 x_i y_i c_i，$i = 1, 2, \ldots, n$;

接下来 m 行，每行三个整数 A B C。

输出格式

共 m 行，每行一个整数，表示答案。

输入输出样例 #1

输入 #1

```
5 2
2 -1 1
0 -3 5
1 -3 2
1 3 5
3 2 2
1 2 4
1 -2 -9
```

输出 #1

```
2
9
```

说明/提示

样例解释：

第一个查询对应 $(2,2)(3,3)$；

第二个查询对应 $(1,1)(2,2)(2,4)(3,3)(3,5)(4,2)(4,4)(5,3)(5,5)$。

数据范围：

对 5% 的数据，$n,m\leq 10^3$；

对另外 10% 的数据，$c_i\leq 2$；

对另外 15% 的数据，$c_i\leq 100$；

对另外 15% 的数据，$\max(|x_i|,|y_i|)=10^6$；

对另外 15% 的数据，$|A|=|B|=1$；

对另外 10% 的数据，$n\leq 20000$，$m\leq 200000$；

对于其余数据，无特殊约束。

每部分数据构成子任务，无依赖关系。

所有数据满足：

$1\leq n\leq 50000$；

$1\leq m\leq 500000$；

$A^2+B^2>0$；

$-10^9\leq x_i,y_i,A,B,C\leq 10^9$；

$1\leq c_i\leq n$；

所有数值为整数；

当 $i\neq j$ 时，$x_i\neq x_j$ 或 $y_i\neq y_j$。

对于除了子任务 4 以外的数据，满足 n 个点的 x,y 坐标分别在某个预设的区间内均匀随机选取，并保证没有重复的点，且对于第 i 个点，c_i 和 x_i,y_i 是分别独立地随机选取的，但 c_i 的分布没有特殊限制。

C. 通通通萨湖啦（tungtungtungsahur）

题目描述

定义： 第六分块

这是一个数据结构问题。

给定整数 n, m ， 以及长度 n 的序列 $a_1, \ldots, a_n$ ， 共 m 次操作。

每次操作可以是 $1, l, r, x$ 表示将 $a_l, a_{l+1}, \ldots, a_r$ 的值增加 x，或 $2, l, r$ 表示查询 $a_l, a_{l+1}, \ldots, a_r$ 的最大子段和（即在序列中选取相邻的若干个（可以是 0 个）元素求和的最大值）。

要求 $1 \leq n \leq 10^5,\ 1 \leq m \leq 2 \times 10^5,\ 1 \leq l \leq r \leq n,\ |x| \leq 10^6$ ， 初始情况下 $|a_i| \leq 10^{12}$ ， 所有数值为整数。

输入格式为依次输入 $n, m, a_1, \ldots, a_n$ ， 然后依次输入每个操作。

输出格式为对每个操作 $2, l, r$ 输出一个整数表示答案。

这个问题存在时间复杂度为 $O((n+m)\sqrt{n})$ 的算法。

定义： (max,+)矩阵乘法

a, b, c 分别是 $n_1 \times n_2,\ n_2 \times n_3,\ n_1 \times n_3$ 规模的由 $[0, 1000]$ 范围内的整数构成的矩阵，满足：

$c_{i,k} = \max\limits_{j=1}^{n_2} a_{i,j} + b_{j,k}$ 。

a, b 矩阵中的元素在范围内独立均匀随机选取。

输入格式为依次输入 n_1, n_2, n_3 ， 然后依次输入 a 和 b 矩阵。

输出格式为输出 c 矩阵。

输入（输出）一个 $A \times B$ 规模的矩阵 v 时，需要对 $i = 1, \ldots, A$ 依次输入（输出）$v_{i,1}, \ldots, v_{i,B}$ 。

你需要解决的问题

你需要编写一个程序，这个程序的输入为空，输出是一个由 **指令** 构成的程序，这个程序用于求解 **(max,+)矩阵乘法** （需要按输入格式读若干个整数，最后按输出格式写若干个整数），且可以调用一次对 **第六分块** 的求解（对应的指令是特殊函数调用3，需要先按输入格式写若干个整数表示问题描述，调用后按输出格式读若干个整数得到答案）和若干次读写整数（对应的指令是特殊函数调用1,2）。

下发的库 （`std.cpp`）可以简化生成指令的过程，你可以基于下发的库编写程序。

下发的 **模拟器** （`chk.cpp`）可以执行由指令构成的程序，并给出得分。

模拟器的编译运行方式可以参考下发的 `makefile` 。

评分方式

如果你的程序不能正常结束，或输出的指令格式错误，或输出行数超过1024，或输出的一行中超过1024个字符，则不能得到任何分数。

输出的指令构成的程序格式正确时，模拟器将对输出的程序进行评测。

输出的程序将被执行4次，依次对应 $n_1 = n_2 = n_3 = 50, 100, 150, 200$ 。如果某次执行出现运行时错误（例如），则本题得0分；否则每次执行如果输出了正确的答案并正常结束，则可以得到 $25\left(\min(1, \max(0, 2 - \frac{M}{M_0})) \cdot \min(1, \max(0, 2 - \frac{T}{T_0}))\right)$ 的分数。总分为每次执行的分数之和。

其中 M 为内存访问次数，T 为指令执行条数，在 **指令和模拟器** 一节定义。

$M_0 = 150000$

$T_0 = 3000000$

指令和模拟器

以下简单介绍评测用的模拟器支持的指令，更详细的指令行为可以查看模拟器的源代码。

内存模型和计算代价统计：

可以读写的内存被表示为由int64_t整数组成的数组 `M` ，大小为 2^{24} ，初值为0；

指针 `S` 初始值为 `M` ，可能被 `call` 和 `ret` 指令修改；

指令的操作数表示为 `M[S[<Int32>]]` 或 `S[<Int32>]` ，用于读写内存；

指令不保存在 `M` 中，不能读写。

模拟器统计的计算代价包括 **指令执行条数** 和 **内存访问次数。**

一条指令（不包括 `<Label>:` ）被执行时， **指令执行条数增加1** ，指令中出现的每个 `M[S[<Int>]]` 导致 **内存访问次数增加1** 。

模拟器从第一条指令开始执行，执行 `exit` 后结束执行。

指令默认顺序执行，但对于指令说明中要求跳转的情况，执行完当前指令后调整到相应位置继续执行。

指令中需要用到以下语法元素：

语法元素	说明
`<A>`	可以是 `M[S[<Int>]]` 或 `S[<Int>]` ，表示指令中需要读写的数据的来源
`<UOP>`	可以是 `- !`
`<BOP>`	可以是 `+ - * / % ^ & | << >> < <= > >= == !=`
`<Label>`	64位有符号整数常数，十进制，表示可以跳转到的标签
`<Int>`	64位有符号整数常数，十进制
`<Int32>`	32位有符号整数常数，十进制

支持的指令如下（需要严格满足描述的语法格式，不允许出现空格等）：

指令	说明
`<A>=<A>`	赋值，读取右边的 `<A>` 写到左边的 `<A>`
`<A>=<UOP><A>`	一元运算，语义和c++对int64_t的相应运算相同，运算结果写到左边的 `<A>`
`<A>=<A><BOP><A>`	二元运算，语义和c++对int64_t的相应运算相同，运算结果写到左边的 `<A>`
`<A>=<Int>`	赋值为常数
`<A>=&S[<Int>]`	取地址，相当于计算 `S-M+<Int>` 保存到 `<A>`
`if <A> goto <Label>`	条件跳转，如果 `<A>` 非零则跳转到 `<Label>`
`goto <Label>`	无条件跳转到 `<Label>`
`ret`	跳转到 `S[0]` 保存的指令地址，将 `S` 修改为 `M+S[1]`
`call <Label> <Int>`	保存下一条指令的地址到 `S[<Int>]` ，保存 `S-M` 到 `S[<Int>+1]` ，将 `S` 修改为 `S+<Int>` ，跳转到 `<Label>`
`exit`	结束执行
`ncall <Int> <A>`	特殊函数调用，`<Int>` 表示特殊函数的类型，可能读写 `<A>`
`<Label>:`	标签，同个标签只能出现一次，不计入指令执行条数，用于表示跳转到的指令地址

特殊函数调用的规定：

`<Int>`	`<A>`	说明
1	读的整数的地址	读一个整数
2	写的整数	写一个整数
3	占位	只能调用一次，功能在题目描述中指定

输入输出模型：

输入输出被保存在一个初始为空的由整数组成的队列中。

特殊函数1从队列开头中取出一个整数，作为读取的结果；

特殊函数2将待写的整数放到队列末尾；

特殊函数3按指定输入输出格式调用特殊函数1,2读写整数。

使用下发的库生成指令

为了方便地生成指令，你可以使用下发的基于宏定义的库作为你提交的源代码的一部分，编译运行后可以输出生成的指令到标准输出。

`<Type>` 可以为 `Int` 表示int64_t，或者 `Ptr<T>` 表示类型 `T` 的指针，或者 `Array<T,n>` 表示类型 `T` 的大小 `n` 的数组。要求 `T` 是 `<Type>` 。

在表达式中， `Int` 之间可以使用二元运算符 `+ - * / % ^ & | << >> < <= > >= == != += -= *= /= %= ^= &= |= <<= >>=` 和一元运算符 `+ - !` ，计算结果的表达式类型为 `Int`。

`Ptr<T>` 或 `Array<T,n>` 和 `Int` 之间可以使用二元运算符 `+ -` ，计算结果的类型为 `Ptr<T>`。

`Ptr<T>` 或 `Array<T,n>` 可以使用一元运算符 `*` ，计算结果的类型为 `T` 。

表达式中可以出现数组访问运算符 `<Expr>[<Expr>]` ，要求左边为 `Array<T,n>` 或 `Ptr<T>` ，右边为 `Int` ，计算结果的类型为 `T`。

表达式中可以出现整数常量，可以被隐式转化为 `Int` 处理。

表达式中可以使用 `()` 改变运算符优先级。

表达式中可以出现类型转化 `<Type>(<Expr>)` ，仅限于 `Array<T,n>` 转为 `Ptr<T>` 或同个类型内的转化，以及 `T` 转为 `Ptr<T>` 表示c++中的 `&`。

表达式中可以出现赋值 `<Expr>=<Expr>` ，赋值时需要满足右侧类型能转化为左侧类型。

`<Expr>` 是表达式，只能包含整数或类型为某种 `<Type>` 的变量，变量的作用域规则和c++相同。

`If(<Expr>) ... End` 类似于c++中的if语句。

`If(<Expr>) ... Else ... End` 对应于c++中的if-else语句。

`While(<Expr>) ... End` 对应于c++中的while语句。

`For(A,B,C) ... End` 对应于c++中的for语句，其中 A,B,C为 `<Expr>` 。

`Break;` `Continue;` `Return;` 对应c++中相应的关键字。

`<Expr>;` 用于计算表达式，并处理表达式中可能出现的赋值。

`<Type> x;` 或 `<Type> x=<Expr>;` 用于在栈上定义变量。特别地，如果 `<Expr>` 的形式为 `<Expr>[<Expr>]` 或 `<Expr>` ，则 `x` 是右侧表达式的引用，而不是一个新的变量。

`Func(name,args) ... End` 用于定义无返回值的函数， `args` 为参数（可以为空，或包含形如 `<Type> x;` 的若干个定义），从第一个被定义的函数开始执行。函数定义不能嵌套，参数类型必须是 `Int` 或 `Ptr<T>`，语句必须出现在函数定义内。

`Read(<Expr>);` ， `Write(<Expr>);` ， `Solve();` 分别表示特殊函数调用1,2,3。

未说明的用法可能能通过编译，但不一定能正确翻译为指令。

框架没有进行过多的优化，因此生成的指令可能有较高的指令执行条数。内存访问次数对应于形如 `<Expr>[<Expr>]` 或 `*<Expr>` 且类型不是数组的表达式的执行次数。