NOIP 模拟赛

day4

题目名称	统计数列	魔法阵	软件包管理器
题目类型	传统型	传统型	传统型
可执行文件名	count.exe	magic.exe	manager.exe
输入文件名	count.in	magic.in	manager.in
输出文件名	count.out	magic.out	manager.out
每个测试点时限	1.0 秒	1.0 秒	2.0 秒
内存限制	512 MB	512 MB	512 MB
测试点/包数目	20	10	20
测试点是否等分	是	是	是

提交源程序文件名

对于 C++ 语言	count.cpp	magic.cpp	manager.cpp
对于 C 语言	count.c	magic.c	manager.c
对于 Pascal 语言	count.pas	magic.pas	manager.pas

编译选项

对于 C++ 语言	-O2 -std=c++14 -Wl,--stack=536870912
对于 C 语言	-O2 -std=c14 -Wl,--stack=536870912
对于 Pascal 语言	-O2

统计数列（count）

【题目描述】

xhl 最近迷上了全排列，他经常会将 1~n 的全排列写出来，并统计它们逆序对的个数。但他的耐心是有限的，于是他找到了你，希望知道 n 个数组成的所有排列中，逆序对数为 k 的序列个数。答案可能很大，只需输出答案除以 1000000007 的余数。

对于排列 p，逆序对数即满足 i<j 且 p[i]>p[j] 的二元组（i，j）数量。

【输入格式】

从文件 ***count.in*** 中读入数据。

一行两个正整数 n，k。

【输出格式】

输出到文件 ***count.out*** 中。

一行，表示答案。

【样例 1 输入】

```
3 1
```

【样例 1 输出】

```
2
```

【样例 2 输入】

```
7 12
```

【样例 2 输出】

```
531
```

【样例 3 输入】

```
4321 1234
```

【样例 3 输出】

```
123042437
```

【子任务】

对于 20% 数据，n,k<=5.
对于 40% 数据，n,k<=20.
对于 60% 数据，n,k<=100.
对于 80% 数据，n,k<=6000.
对于另 20% 数据，n<=100000,k<=50.

魔法阵（magic）

【题目描述】

“这样的真的可以吗...”

小 X 拿着那本魔法书，犹豫着。

尽管她早已成为 H 国最强的魔法师，但她对力量的热情从未减少，直到那天，小 L 给了她那本魔法书。

“这本书记载着魔法的最高境界，但只有最虔诚的人才能领悟。”

“虔诚的人...”

她打开那本魔法书，咏唱着那一串串神秘的咒语，接着她拿出水晶，按书中的指示排列好。那水晶是能力的象征，只有最强的她才拥有如此多的水晶。

她再一次咏唱咒语，那水晶绽放出从未有过的能量，彼此由光线相连，连成符文一样的图案。

“虔诚的人...”

她用手触摸那些水晶，那些被触摸的水晶连同它们之间的光线变得如此耀眼，而她感受到了自己内心的力量。

“太好了，只要将它们全都点亮就能得到真正的力量了!”

但是，事与愿违，随着水晶的点亮，那份力量却在逐渐消退。

“为什么...”

她将那些水晶反复点亮又熄灭，慢慢地，她意识到，每颗水晶拥有的能量是不同的，而不同的水晶之间连接的光线只会白白消耗它们的能量。

“原来如此!”

她拿出了最亮的水晶，但是她没有感受到任何力量。

“虔诚的人!”

她终于明白了，她必须**点亮至少一条光线**，否则魔法是不会有任何效果的。而她最终获得的力量值就是**她点亮的水晶的能量值和/连接水晶的光线的亮度和**。

“我该怎么办...”

【输入格式】

从文件 ***magic.in*** 中读入数据。

第一行包含两个正整数 n,m, 表示水晶个数和有光线相连的水晶对数。

第二行 n 个正整数，第 i 个数表示第 i 个水晶的能量值。

接下来 m 行，每行 3 个正整数，第 i 行表示连接的两个水晶的编号和这条光线的亮度。

【输出格式】

输出到文件 ***magic.out*** 中。
输出一个小数，表示小 X 获得的力量值的最大值。保留两位小数。

【样例 1 输入】

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

【样例 1 输出】

```
5.00
```

【样例 1 解释】

小 X 点亮了 2 号水晶和 3 号水晶，它们之间有一条光线，获得的力量值是 5。

【子任务】

对 30% 数据，$n \leq 10, m \leq 20$。
对 50% 数据，$n \leq 20, m \leq 190$。
对另 10% 数据，光线的亮度都是 1。
对 100% 数据，$n \leq 100000, m \leq 200000$, 水晶的能量值，光线的亮度 $\leq 10^9$。

软件包管理器（manager）

【题目描述】

Linux 用户和 OSX 用户一定对软件包管理器不会陌生。通过软件包管理器，你可以通过一行命令安装某一个软件包，然后软件包管理器会帮助你从软件源下载软件包，同时自动解决所有的依赖（即下载安装这个软件包的安装所依赖的其它软件包），完成所有的配置。Debian/Ubuntu 使用的 apt-get，Fedora/CentOS 使用的 yum，以及 OSX 下可用的 homebrew 都是优秀的软件包管理器。

你决定设计你自己的软件包管理器。不可避免地，你要解决软件包之间的依赖问题。如果软件包 A 依赖软件包 B，那么安装软件包 A 以前，必须先安装软件包 B。同时，如果想要卸载软件包 B，则必须卸载软件包 A。现在你已经获得了所有的软件包之间的依赖关系。而且，由于你之前的工作，除 0 号软件包以外，在你的管理器当中的软件包都会依赖一个且仅一个软件包，而 0 号软件包不依赖任何一个软件包。依赖关系不存在环（若有 $m(m \geq 2)$ 个软件包 $A_1, A_2, A_3, , A_m$，其中 A_1 依赖 A_2，A_2 依赖 A_3，A_3 依赖 A_4，……，A_{m-1} 依赖 A_m，而 A_m 依赖 A_1，则称这 m 个软件包的依赖关系构成环），当然也不会有一个软件包依赖自己。

现在你要为你的软件包管理器写一个依赖解决程序。根据反馈，用户希望在安装和卸载某个软件包时，快速地知道这个操作实际上会改变多少个软件包的安装状态（即安装操作会安装多少个未安装的软件包，或卸载操作会卸载多少个已安装的软件包），你的任务就是实现这个部分。**注意，安装一个已安装的软件包，或卸载一个未安装的软件包，都不会改变任何软件包的安装状态，即在此情况下，改变安装状态的软件包数为 0。**

【输入格式】

从文件 ***manager.in*** 中读入数据。

输入文件的第 1 行包含 1 个整数 n，表示软件包的总数。软件包从 0 开始编号。

随后一行包含 $n-1$ 个整数，相邻整数之间用单个空格隔开，分别表示 $1, 2, 3, , n-2, n-1$ 号软件包依赖的软件包的编号。

接下来一行包含 1 个整数 q，表示询问的总数。之后 q 行，每行 1 个询问。询问分为两种：

- install x：表示安装软件包 x
- uninstall x：表示卸载软件包 x

你需要维护每个软件包的安装状态，一开始所有的软件包都处于未安装状态。

对于每个操作，你需要输出这步操作会改变多少个软件包的安装状态，随后应用这个操作（即改变你维护的安装状态）。

【输出格式】

输出到文件 ***manager.out*** 中。
输出文件包括 q 行。
输出文件的第 i 行输出 1 个整数，为第 i 步操作中改变安装状态的软件包数。

【样例 1 输入】

```
7
0 0 0 1 1 5
5
install 5
install 6
uninstall 1
install 4
uninstall 0
```

【样例 1 输出】

```
3
1
3
2
3
```

【样例 1 解释】

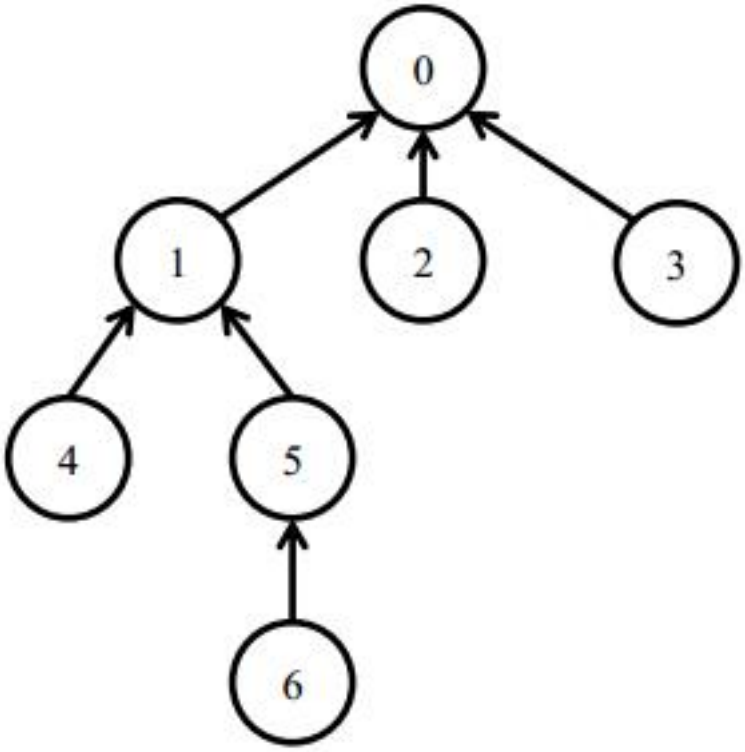
一开始所有的软件包都处于未安装状态。

安装 5 号软件包，需要安装 0,1,5 三个软件包。

之后安装 6 号软件包，只需要安装 6 号软件包。此时安装了 0,1,5,6 四个软件包。

卸载 1 号软件包需要卸载 1,5,6 三个软件包。此时只有 0 号软件包还处于安装状态。

之后安装 4 号软件包，需要安装 1,4 两个软件包。此时 0,1,4 处在安装状态。最后，卸载 0 号软件包会卸载所有的软件包。



【样例 2 输入】

```
10
0 1 2 1 3 0 0 3 2
10
install 0
install 3
uninstall 2
install 7
install 5
install 9
uninstall 9
```

```
install 4
install 1
install 9
```

【样例 2 输出】

```
1
3
2
1
3
1
1
1
0
1
```

【子任务】

【数据规模与约定】

测试点编号	n 的规模	q 的规模	备注
1	$n = 5{,}000$	$q = 5{,}000$	
2			
3	$n = 100{,}000$	$q = 100{,}000$	数据不包含卸载操作
4			
5	$n = 100{,}000$	$q = 100{,}000$	编号为 i 的软件包所依赖的软件包编号在 $[0, i-1]$ 内均匀随机 每次执行操作的软件包编号在 $[0, n-1]$ 内均匀随机
6			
7			
8			
9	$n = 100{,}000$	$q = 100{,}000$	
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			