一起写码吧

题目名称	源文件	标准输入文件	标准输出文件	测试点数量	单个测试点分值	时间限制	空间限制
单源次短路	second.cpp	second.in	second.out	10	10	1S	512MB
直径	longest.cpp	longest.in	longest.out	20	5	1S	512MB
软件包管理器	manager.cpp	manager.in	manager.out	20	5	1S	512MB
时间复杂度	complexity.cpp	complexity.in	complexity.out	10	10	1S	512MB
染色	coloring.cpp	coloring.in	coloring.out	20	5	1S	512MB

基本按难度排序

单源次短路

题目描述

给定一个 n 个点，m 条无向边的带非负权图，请你计算从 1 出发，到n的次短距离。

输入格式

第一行为三个正整数 n, m。
第二行起 m 行，每行三个非负整数 u_i, v_i, w_i，表示连接u_i和v_i有一条权值为 w_i 的边。

输出格式

输出一行 1 个非负整数。

样例 #1

样例输入 #1

```
4 6
1 2 2
2 3 2
2 4 1
1 3 5
3 4 3
1 4 4
```

样例输出 #1

```
4
```

提示

- 对于 30% 的数据：$1 \leq n \leq 100$，$1 \leq m \leq 10^3$；
- 对于 100% 的数据：$1 \leq n \leq 10^5$，$1 \leq m \leq 2 \times 10^5$，$1 \leq u, v \leq n$，$1 \leq w \leq 1000$。

直径(longest)

题目描述

给定一颗有 n 个节点的树, 输出直径长度.

输入格式

输入共 n 行,

第 1 行有两个正整数 n,

第 $2 \sim n$ 行每行三个正整数 u, v, w, 表示 u, v 之间有一条边, 边权为 w.

输出格式

输出共 1 行, 表示直径长度.

输入输出样例

输入 # 1

```
10
1 2 254
1 3 65
2 5 144
3 4 835
3 6 730
3 7 931
3 8 43
3 10 504
6 9 92
```

输出 # 1

```
1766
```

说明/提示

数据范围

- 对于 15% 的数据，$1 \leq n \leq 10$
- 对于 35% 的数据，$1 \leq n \leq 100$
- 对于 100% 的数据，$1 \leq n \leq 10^5, 1 \leq w \leq 10^3$

软件包管理器

题目背景

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。

题目描述

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。且依赖关系不存在环（即不会存在 m 个软件包 $a_1, a_2, \ldots, a_m$，对于 $i < m$，a_i 依赖 a_{i+1}，而 a_m 依赖 a_1 的情况）。

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

第一行一个正整数 n，表示软件包个数，从 0 开始编号。
第二行有 $n-1$ 个整数，第 i 个表示 i 号软件包依赖的软件包编号。
然后一行一个正整数 q，表示操作个数，格式如下：

- `install x` 表示安装 x 号软件包
- `uninstall x` 表示卸载 x 号软件包

一开始所有软件包都是未安装的。

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 q 行，每行一个整数，表示每次询问的答案。

样例 #1

样例输入 #1

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

样例输出 #1

```
3
1
3
2
3
```

样例 #2

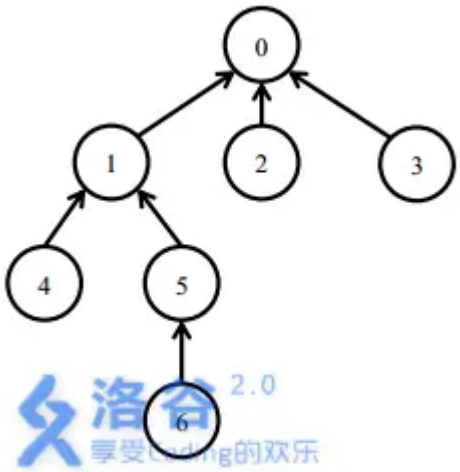
样例输入 #2

```
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
```

样例输出 #2

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

提示



一开始所有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。

【数据范围】

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			

时间复杂度

题目描述

小明正在学习一种新的编程语言 A++，刚学会循环语句的他激动地写了好多程序并 给出了他自己算出的时间复杂度，可他的编程老师实在不想一个一个检查小明的程序， 于是你的机会来啦！下面请你编写程序来判断小明对他的每个程序给出的时间复杂度是否正确。

A++语言的循环结构如下：

```
F i x y
    循环体
E
```

其中`F i x y`表示新建变量 i（变量 i 不可与未被销毁的变量重名）并初始化为 x，然后判断 i 和 y 的大小关系，若 i 小于等于 y 则进入循环，否则不进入。每次循环结束后 i 都会被修改成 $i+1$，一旦 i 大于 y 终止循环。

x 和 y 可以是正整数（x 和 y 的大小关系不定）或变量 n。n 是一个表示数据规模的变量，在时间复杂度计算中需保留该变量而不能将其视为常数，该数远大于 100。

`E` 表示循环体结束。循环体结束时，这个循环体新建的变量也被销毁。

注：本题中为了书写方便，在描述复杂度时，使用大写英文字母 O 表示通常意义下 Θ 的概念。

输入格式

输入文件第一行一个正整数 t，表示有 t（$t \leq 10$）个程序需要计算时间复杂度。 每个程序我们只需抽取其中 `F i x y` 和 `E` 即可计算时间复杂度。注意：循环结构允许嵌套。

接下来每个程序的第一行包含一个正整数 L 和一个字符串，L 代表程序行数，字符串表示这个程序的复杂度，`O(1)` 表示常数复杂度，`O(n^w)` 表示复杂度为 n^w，其中 w 是一个小于 100 的正整数，输入保证复杂度只有 `O(1)` 和 `O(n^w)` 两种类型。

接下来 L 行代表程序中循环结构中的`F i x y`或者 `E`。 程序行若以`F`开头，表示进入一个循环，之后有空格分离的三个字符（串）`i x y`， 其中 i 是一个小写字母（保证不为n），表示新建的变量名，x 和 y 可能是正整数或 n ，已知若为正整数则一定小于 100。

程序行若以`E`开头，则表示循环体结束。

输出格式

输出文件共 t 行，对应输入的 t 个程序，每行输出 `Yes` 或 `No` 或者 `ERR`，若程序实际复杂度与输入给出的复杂度一致则输出 `Yes`，不一致则输出 `No`，若程序有语法错误（其中语法错误只有: ① `F` 和 `E` 不匹配 ②新建的变量与已经存在但未被销毁的变量重复两种情况），则输出 `ERR`。

注意：即使在程序不会执行的循环体中出现了语法错误也会编译错误，要输出 `ERR`。

样例 #1

样例输入 #1

```
8
2 O(1)
F i 1 1
E
2 O(n^1)
F x 1 n
E
1 O(1)
F x 1 n
4 O(n^2)
F x 5 n
F y 10 n
E
E
4 O(n^2)
F x 9 n
E
F y 2 n
E
4 O(n^1)
F x 9 n
F y n 4
E
E
```

```
4 O(1)
F y n 4
F x 9 n
E
E
4 O(n^2)
F x 1 n
F x 1 10
E
E
```

样例输出 #1

```
Yes
Yes
ERR
Yes
No
Yes
Yes
ERR
```

提示

【输入输出样例解释 1】

第一个程序 i 从 1 到 1 是常数复杂度。

第二个程序 x 从 1 到 n 是 n 的一次方的复杂度。

第三个程序有一个 `F` 开启循环却没有 `E` 结束，语法错误。

第四个程序二重循环，n 的平方的复杂度。

第五个程序两个一重循环，n 的一次方的复杂度。

第六个程序第一重循环正常，但第二重循环开始即终止（因为 n 远大于 100，100 大于 4）。

第七个程序第一重循环无法进入，故为常数复杂度。

第八个程序第二重循环中的变量 x 与第一重循环中的变量重复，出现语法错误②，输出 `ERR`。

【数据规模与约定】

对于 30% 的数据：不存在语法错误，数据保证小明给出的每个程序的前 $L/2$ 行一定为以 `F` 开头的语句，第 $L/2+1$ 行至第 L 行一定为以 `E` 开头的语句，$L \leq 10$，若 x、y 均为整数，x 一定小于 y，且只有 y 有可能为 n。

对于 50% 的数据：不存在语法错误，$L \leq 100$，且若 x、y 均为整数，x 一定小于 y，且只有 y 有可能为 n。

对于 70% 的数据：不存在语法错误，$L \leq 100$。

对于 100% 的数据：$L \leq 100$。

染色

题目描述

给定一棵 n 个节点的无根树，共有 m 个操作，操作分为两种：

1. 将节点 a 到节点 b 的路径上的所有点（包括 a 和 b）都染成颜色 c。
2. 询问节点 a 到节点 b 的路径上的颜色段数量。

颜色段的定义是极长的连续相同颜色被认为是一段。例如 `112221` 由三段组成：`11`、`222`、`1`。

输入格式

输入的第一行是用空格隔开的两个整数，分别代表树的节点个数 n 和操作个数 m。

第二行有 n 个用空格隔开的整数，第 i 个整数 w_i 代表结点 i 的初始颜色。

第 3 到第 $(n+1)$ 行，每行两个用空格隔开的整数 u, v，代表树上存在一条连结节点 u 和节点 v 的边。

第 $(n+2)$ 到第 $(n+m+1)$ 行，每行描述一个操作，其格式为：

每行首先有一个字符 op，代表本次操作的类型。

- 若 op 为 `C`，则代表本次操作是一次染色操作，在一个空格后有三个用空格隔开的整数 a, b, c，代表将 a 到 b 的路径上所有点都染成颜色 c。
- 若 op 为 `Q`，则代表本次操作是一次查询操作，在一个空格后有两个用空格隔开的整数 a, b，表示查询 a 到 b 路径上的颜色段数量。

输出格式

对于每次查询操作，输出一行一个整数代表答案。

样例 #1

样例输入 #1

```
6 5
2 2 1 2 1 1
1 2
1 3
2 4
2 5
2 6
Q 3 5
C 2 1 1
Q 3 5
C 5 1 2
Q 3 5
```

样例输出 #1

```
3
1
2
```

提示

数据规模与约定

对于 100% 的数据，$1 \leq n, m \leq 10^5$，$1 \leq w_i, c \leq 10^9$，$1 \leq a, b, u, v \leq n$，op 一定为 `C` 或 `Q`，保证给出的图是一棵树。