

2018 年全国青少年信息学奥林匹克 浙江省队选拔赛第二试

竞赛时间：4 月 26 日 8:00 – 13:00

题目名称	树	胖	保镖
目录	tree	joke	guard
可执行文件名	tree	joke	guard
输入文件名	tree.in	joke.in	guard.in
输出文件名	tree.out	joke.out	guard.out
每个测试点时限	3s	3s	4s
内存限制	512MB	512MB	512MB
测试点数目	10	10	10
每个测试点分值	10	10	10
是否有部分分	否	否	否
题目类型	传统型	传统型	传统型
是否有附加文件	是	是	是
是否有 Special Judge	否	否	是

提交源程序必须加后缀

对于 C++ 语言	tree.cpp	joke.cpp	guard.cpp
对于 C 语言	tree.c	joke.c	guard.c
对于 Pascal 语言	tree.pas	joke.pas	guard.pas

编译开关

对于 C++ 语言	-O2 -lm	-O2 -lm	-O2 -lm
对于 C 语言	-O2 -lm	-O2 -lm	-O2 -lm
对于 Pascal 语言	-O2	-O2	-O2

1 树

1.1 题目描述

九条可怜是一个热爱出题的女孩子。

虽然出题本身是一件非常有趣的事情，但是要把题目给出成正式比赛，就不是那么有趣了：造数据总是一件让人心力憔悴的事情。

在 ZJOI2018 Day 1 中，可怜出了一道和树相关的非常有趣的题，她打算采用一种常用的方式随机生成一棵 n 个节点的有根树：

- 节点 1 作为树的根。
- 对于 $i \in [2, n]$ ，独立地从 $[1, i]$ 中等概率随机选取一个节点作为 i 的父亲。

可怜不是很想考虑这样随机出来的数据能不能卡掉暴力，毕竟乱搞也是 OI 比赛的一部分。可怜比较在意的是题目的区分度，以及是不是所有可能的分数都出现了。因此，可怜希望任何两个测试点的树是有区别的：这样就可能会有错误的程序能只通过其中一个点。

因此，可怜想要计算，通过上面的方法独立的随机生成 k 棵 n 个节点的有根树 T_1 至 T_k ，他们两两同构的概率是多少。

两棵 n 个节点的有根树 T_1 和 T_2 同构当且仅当存在长度为 n 的排列 p ，满足 $p_1 = 1$ ，且对于 $\forall i \in [2, n]$ ，若 i 在 T_1 的父亲是 f ，则 p_i 在 T_2 的父亲是 p_f 。

1.2 输入格式

第一行输入三个整数 n, k, p ，表示节点个数，树的个数以及模数。输入保证 $10^8 \leq p \leq 10^9$ 且 p 是质数。

1.3 输出格式

输出一行一个整数，表示答案对 p 取模后的值。即如果答案的最简分数表示为 $\frac{a}{b}$ ，输出 $a \times b^{-1} \bmod p$ 。

1.4 样例输入

```
2 2 998244353
3 2 998244353
4 2 998244353
10 2 998244353
50 233 998244353
```

1.5 样例输出

1
499122177
332748118
113919852
634280054

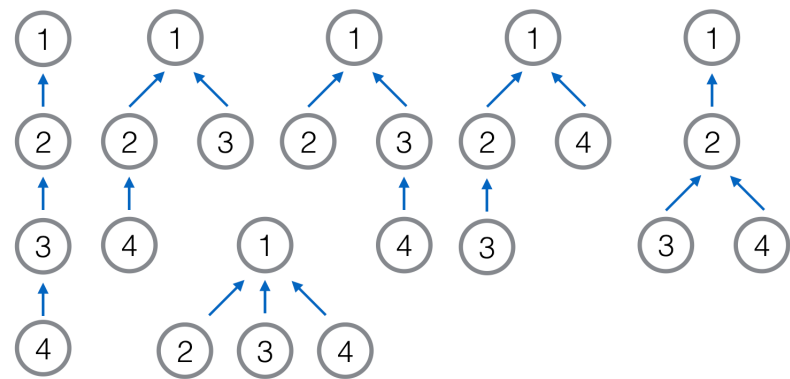
1.6 样例解释

样例中有五组不同的数据，所以输入格式略有不同。在实际的测试数据中，输入只有一行。

在第一组数据中，能够生成的树是唯一的，因此生成的两棵树必定相同。

在第二组数据中，能够生成的树只有两种，他们是不同构的。因此生成的两棵树同构的概率为 $\frac{1}{2}$ ，在模 998244353 意义下为 499122177。

在第三组数据中，能够生成的树有 6 种，如下图所示。其中第二、三、四棵（第一排中间三棵）是同构的，其余两两不同构。因此生成的两棵树同构的概率为 $\frac{1}{3}$ ，在模 998244353 意义下为 332748118。



1.7 数据范围与约定

测试点	n	k	测试点	n	k
1	≤ 5	$=2$	6	≤ 50	$\leq 10^9$
2	≤ 10		7	≤ 200	
3	≤ 20		8	≤ 500	
4	≤ 50		9	≤ 1000	
5			10	≤ 2000	

对于 100% 的数据，保证 p 是质数且 $10^8 \leq p \leq 10^9$ 。

2 胖

2.1 题目描述

Cedyks 是九条可怜的好朋友（可能这场比赛公开以后就不是了），也是这题的主人公。

Cedyks 是一个富有的男孩子。他住在著名的 The Place（宫殿）中。

Cedyks 是一个努力的男孩子。他每天都做着不一样的题来锻炼他的 The Salt（灵魂）。

这天，他打算在他的宫殿外围修筑一道城墙，城墙上有 n 座瞭望塔。你可以把城墙看做一条线段，瞭望塔是线段上的 n 个点，其中 1 和 n 分别为城墙的两个端点。其中第 i 座瞭望塔和第 $i + 1$ 座瞭望塔的距离为 w_i ，他们之间的道路是双向的。

城墙很快就修建好了，现在 Cedyks 开始计划修筑他的宫殿到城墙的道路。因为这题的题目名称，Cedyks 打算用他的宫殿到每一个瞭望塔的最短道路之和来衡量一个修建计划。

现在 Cedyks 手上有 m 个设计方案，第 k 个设计方案会在宫殿和瞭望塔之间修建 T_k 条双向道路，第 i 条道路连接着瞭望塔 a_i ，长度为 l_i 。

计算到每一个瞭望塔的最短路之和是一个繁重的工程，本来 Cedyks 想用广为流传的 SPFA 算法来求解，但是因为他的 butter（缓冲区）实在是太小了，他只能转而用原始的贝尔福特曼算法来计算，算法的流程大概如下：

1. 定义宫殿是 0 号点，第 i 个瞭望塔是 i 号点，双向边 (u_i, v_i, l_i) 为一条连接 u_i 和 v_i 的双向道路。令 d 为距离数组，最开始 $d_0 = 0, d_i = 10^{18} (i \in [1, n])$ 。
2. 令辅助数组 $c = d$ 。依次对于每一条边 (u_i, v_i, w_i) 进行增广， $c_{u_i} = \min(c_{u_i}, d_{v_i} + w_i)$ ， $c_{v_i} = \min(c_{v_i}, d_{u_i} + w_i)$ 。
3. 令 t 为 c 和 d 中不一样的位置个数，即令 $S = \{i | c_i \neq d_i\}$ ，则 $t = |S|$ 。若 $t = 0$ ，说明 d 就是最终的最短路，算法结束。否则令 $d = c$ ，回到第二步。

因为需要计算的设计方案实在是太多了，所以 Cedyks 雇佣了一些人来帮他进行计算。为了避免这些人用捏造出来的数据偷懒，他定义一个设计方案的校验值为在这个方案上运行贝尔福特曼算法每一次进入第三步 t 的和。他会让好几个雇佣来的人计算同样的设计方案，并比对每一个人给出的校验值。

你是 Cedyks 雇佣来的苦力之一，聪明的你发现在这个情形下计算最短路的长度的和是一件非常简单的事情。但是寄人篱下不得不低头，你不得不再计算出每一个方案的校验值来交差。

2.2 输入格式

第一行输入两个整数 n, m ，表示瞭望塔个数和设计方案个数。

接下来一行 $n - 1$ 个数 w_i ，表示瞭望塔 i 和 $i + 1$ 之间道路的长度。

接下来 m 行，每行描述一个设计方案。第一个整数 K 表示设计方案中的道路数量，接下来 K 个数对 (a_i, l_i) 为一条宫殿到瞭望塔的边。

2.3 输出格式

对于每一个设计方案，输出一行一个整数表示校验值。

2.4 样例输入

```
5 5
2 3 1 4
1 2 2
2 1 1 4 10
3 1 1 3 1 5 1
3 1 10 2 100 5 1
5 1 1 2 1 3 1 4 1 5 1
```

2.5 样例输出

```
5
8
5
8
5
```

2.6 样例解释

- 对于第一个设计方案，每一个阶段 d 的变化为：

– $[0, 10^{18}, 10^{18}, 10^{18}, 10^{18}, 10^{18}] \rightarrow [0, 10^{18}, 2, 10^{18}, 10^{18}, 10^{18}] \rightarrow$
– $[0, 4, 2, 5, 10^{18}, 10^{18}] \rightarrow [0, 4, 2, 5, 6, 10^{18}] \rightarrow [0, 4, 2, 5, 6, 10]$ 。

因此校验值为 $1 + 2 + 1 + 1 = 5$ 。

- 对于第二个设计方案，每一个阶段 d 的变化为：

– $[0, 10^{18}, 10^{18}, 10^{18}, 10^{18}, 10^{18}] \rightarrow [0, 1, 10^{18}, 10^{18}, 10, 10^{18}] \rightarrow$
– $[0, 1, 3, 11, 10, 14] \rightarrow [0, 1, 3, 6, 10, 14] \rightarrow [0, 1, 3, 6, 7, 14] \rightarrow [0, 1, 3, 6, 7, 11]$ 。

因此校验值为 $2 + 3 + 1 + 1 + 1 = 8$ 。

- 对于第三个设计方案，每一个阶段 d 的变化为：

– $[0, 10^{18}, 10^{18}, 10^{18}, 10^{18}, 10^{18}] \rightarrow [0, 1, 10^{18}, 1, 10^{18}, 1] \rightarrow [0, 1, 3, 1, 2, 1]$ 。

因此校验值为 $3 + 1 + 1 = 5$ 。

- 对于第四个设计方案，每一个阶段 d 的变化为：

– $[0, 10^{18}, 10^{18}, 10^{18}, 10^{18}, 10^{18}] \rightarrow [0, 10, 100, 10^{18}, 10^{18}, 1] \rightarrow$
– $[0, 10, 12, 103, 5, 1] \rightarrow [0, 10, 12, 6, 5, 1] \rightarrow [0, 10, 9, 5, 1]$ 。

因此校验值为 $3 + 3 + 1 + 1 = 8$ 。

- 对于第五个设计方案，每一个阶段 d 的变化为：
 $[0, 10^{18}, 10^{18}, 10^{18}, 10^{18}, 10^{18}] \rightarrow [0, 1, 1, 1, 1, 1]$

因此校验值为 5。

2.7 数据范围与约定

测试点	n	m	K	其他约定
1	≤ 1000	≤ 1000	≤ 100	无
2				
3	$\leq 2 \times 10^5$	$\leq 2 \times 10^5$	$\leq 2 \times 10^5$	$1 \leq w_i, l_i \leq 50$
4				无
5				
6				
7				
8				
9				
10				

对于 100% 的数据，保证每个设计方案 a_i 两两不同且 $1 \leq a_i \leq n$ 。

对于 100% 的数据，保证 $1 \leq w_i, l_i \leq 10^9, 1 \leq \sum K \leq 2 \times 10^5$ 。

3 保镖

3.1 题目描述

九条可怜是一个贪玩的女孩子。

一个可爱的女孩子出门在外和朋友玩，难免会遇到危险，于是可怜的爸爸悄悄的安插了 n 位保镖在暗中保护她。因为可怜是一个要强的女孩子，所以她的爸爸不想让她知道她的身边有这么多的保镖，因此保镖们必须通过定期的换岗来避免被可怜发现。

一次理想的换岗是：距离可怜远的保镖被换到了较近的位置，距离可怜近的保镖被换到了较远的位置。经过一系列安排，保镖们会按照如下的方式进行换岗：

- 换岗时，可怜和保镖们的位置可以被抽象成二维平面直角坐标系上的点。设可怜的位置为 O ，保镖的位置为 P_i ，换岗后的位置为 P'_i
- 在换岗前后，保镖和可怜的相对位置不变。即对于 $\forall i \in [1, n]$ ， P'_i 在射线 OP_i 上。
- 在换岗前后，保镖和可怜的距离变成了原来的倒数。即对于 $\forall i \in [1, n]$ ， $|OP_i||OP'_i| = 1$ 。

同时，保镖的位置决定了可怜的安全度。如果在外围的保镖越多，那么他们能观察到的信息就越多，可怜就越安全。因此，我们定义这些保镖的位置的凸包的**顶点个数**为可怜的安全度。

然而，可怜的行踪总是神出鬼没的。这一天，保镖们跟丢了可怜，只知道可怜下次会在以 (x_1, y_1) 为左下角，以 (x_2, y_2) 为右上角的矩形区域内出现，具体的位置服从这个矩形区域内的均匀分布（可以理解为等概率随机）。因为保镖们已经有很长时间没有换岗了，于是他们打算在可怜下次出现的时候换岗。同时，在极小的概率下，可怜可能会出现在和某个保镖相同的位置上。这时这个保镖会被发现，从而他会保持位置不动，而其他的保镖还是会按照上述规则进行换岗。

现在，可怜的爸爸想要计算，保镖们在换岗后，可怜的安全度的期望是多少。

如果你对凸包不太熟悉，这儿给出凸包形式化的定义：

- 对于一个简单多边形，它是凸多边形当且仅当它内部任意两点的连线在它的内部。
- 对于一个点集 P ，它的凸包为包含所有点面积最小的没有三个连续顶点共线的凸多边形。

3.2 输入格式

第一行输入一个整数 n ，表示保镖的数量。

第二行输入四个整数 x_1, y_1, x_2, y_2 ，表示可怜可能出现的矩形区域。

接下来 n 行每行两个整数 a, b ，表示一个保镖的坐标。保证保镖的坐标两两不同。

3.3 输出格式

输出一行一个实数表示凸包节点数的期望。

当你的答案与标准输出的绝对误差或相对误差在 10^{-7} 内时，就会被视为正确。

3.4 样例输入

```
4
0 0 1 1
0 0
2 0
0 1
1 1
```

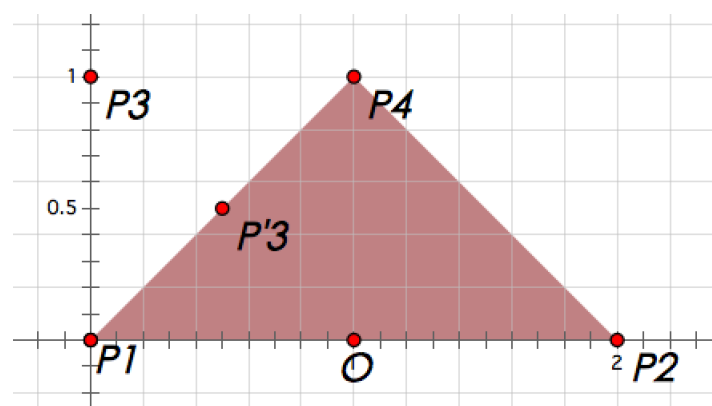
3.5 样例输出

```
3.7853981633974474
```

3.6 样例解释

这儿画出可怜出现在 $(1, 0)$ 时的情况，如下图所示，1, 2, 4 号保镖的位置保持不变，分别为 P_1, P_2, P_4 ，3 号保镖从 P_3 变到了 P'_3 ，坐标为 $(\frac{1}{2}, \frac{1}{2})$ 。

这时四个保镖的位置 P_1, P_2, P'_3, P_4 的凸包为三角形 P_1, P_4, P_2 ，因此可怜的安全度为 3。注意这时 P'_3 正好落在边 P_1P_4 上，但是根据凸包的定义，它不是顶点。



3.7 数据范围与约定

测试点	n	测试点	n
1	≤ 3	6	≤ 50
2	≤ 4	7	≤ 350
3		8	
4	≤ 50	9	≤ 2000
5		10	

对于 100% 的数据，保证 $0 \leq a, b, x_0, x_1, y_0, y_1 \leq 10^5, x_0 < x_1, y_0 < y_1, n \geq 3$ 。

对于 100% 的数据，保证保镖的位置两两不同。

同时为了避免可能出现的**精度误差**，在所有实际的测试数据以及大样例中，保证可能出现的矩形区域的长宽都不小于 10^3 ，即 $x_1 - x_0, y_1 - y_0 \geq 10^3$.