果冻的果树

相信对大二或以上的选手这是非常简单的送分分签到到题了 awa

设二叉树有 n 个节点，叶子节点有 n_0 个，只有左儿子或右儿子的节点有 n_1 个，既有左儿子又有右儿子的节点题给为 m 个，那么节点总和为 n ，有恒等式：

$$n = n_0 + n_1 + m \quad ①$$

由于这是一棵树，所以有且仅有 $n-1$ 条边，其中 n_0 的儿子节点贡献 0 条边， n_1 的儿子节点贡献 $1 \times n_1$ 条边， m 的儿子节点贡献 $2 \times m$ 条边，有恒等式：

$$n - 1 = n_1 + 2m \quad ②$$

$① - ②$ 得， $1 = n_0 - m$ 即 $n_0 = m + 1$ ，所以 n_0 固定为 $m+1$ ，即 $l = r = m + 1$

拓展地说，当最少总节点 n 时， $n_1 = 0$ ，当最多总节点时 $n_1 \to \infty$ ，可以构造无限延伸的斜树，但是都不会改变 $n_0 = m + 1$ 的结论

这题过题率还是蛮高的，WA题的绝大多数都是没有换行或少输出而挂的

C 参考代码

```c
#include <stdio.h>
signed main()
{
    int t, m;
    for (scanf("%d", &t); t; --t)
    {
        scanf("%d", &m);
        printf("%d %d\n", m + 1, m + 1);
    }
}
```

C++ 参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
int t, m;
signed main()
{
    for (cin >> t; t; --t)
    {
        cin >> m;
        cout << (m + 1) << ' ' << (m + 1) << '\n';
    }
}
```

Python 参考代码

```python
for _ in range(int(input())):
    m = int(input())
    print(m+1, m+1)
```

Java 参考代码

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int t = in.nextInt(), m;
        for (int i = 0; i < t; ++i) {
            m = in.nextInt();
            System.out.println((m + 1) + " " + (m + 1));
        }
    }
}
```

云烟的正方体

这是一道小模拟题。解法不唯一，下面仅展示其中一种实现方法。

整体思路：开一个长至少为 m 的数组 z 存每一列的最值，$h = \max z$，那么第 i 列应当由下往上输出 z_i 个 `*` 和 $h - z_i$ 个空格

实现细节：

- 为与题意对应，可以把下标均从 1 开始
- 事实上输入量不超过 10^4，可以不存 a 矩阵，只需要每次读入后马上与 z 比较并更新 z 即可

  如果存 a 矩阵，可以开二维动态数组或 vector，空间复杂度为 $O(\max(n, m))$；如果开二维静态数组，空间复杂度为 $O(nm)$，计算得对 int，需要 $10^4 \times 10^4 \times 4 \div 1024 \div 1024 \approx 381MB$，会超内存，但是考虑到 $a \leq 100$，不超过 $char$ 范围，如果开 $char$ 数组来存，内存为 $381 \div 4 = 95MB$，不会超内存
- 可以初始化一个至少 h 行至少 m 列的二维字符数组，初始值为空格，然后往上面填 `*`，最后逐行输出这个数组即可

  特别地，OJ判题原理会自动忽略行末空格，所以行末多余的空格不会影响判题，直接在第 $m + 1$ 个输出字符的位置填上 `\0` 即可

容易发现，总时空复杂度为 $O(\max(n, m) + hm)$

C++ 参考代码(不存矩阵)

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll n, m, h, z[10010], v;
char ans[102][10010];
signed main()
{
    cin >> n >> m;
    memset(ans, ' ', sizeof ans);
    for (ll i = 1; i <= n; ++i)
    {
        for (ll j = 1; j <= m; ++j)
        {
            cin >> v;
```

```
            z[j] = max(z[j], v);
            h = max(h, z[j]);
        }
    }
    for (ll i = 1; i <= m; ++i)
    {
        for (ll j = h; j > h - z[i]; --j)
        {
            ans[j][i] = '*';
        }
    }
    for (ll i = 1; i <= h; ++i)
    {
        ans[i][m + 1] = '\0';
        printf("%s\n", ans[i] + 1);
    }
    return 0;
}
```

C++ 参考代码(存矩阵)

> 这是一种未优化代码，事实上 `a[maxn][maxn]` 没有存在的必要，可以优化掉，展示该参考代码只为了证明 $O(nm)$ 空间复杂度不会 MLE

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define maxn 10010
ll n, m, h, z[maxn], v;
char a[maxn][maxn], ans[101][maxn];
signed main()
{
    cin >> n >> m;
    memset(ans, ' ', sizeof ans);
    // memset(a, 0, sizeof a); 使得接近MLE
    for (ll i = 1; i <= n; ++i)
    {
        for (ll j = 1; j <= m; ++j)
        {
            cin >> v; //小心不要读字符
            a[i][j] = (char)v;
            z[j] = max(z[j], (ll)a[i][j]);
            h = max(h, z[j]);
        }
    }
    for (ll i = 1; i <= m; ++i)
    {
        for (ll j = h; j > h - z[i]; --j)
        {
            ans[j][i] = '*';
        }
    }
    for (ll i = 1, j; i <= h; ++i)
    {
        ans[i][m + 1] = '\0';
```

```
        printf("%s\n", ans[i] + 1);
    }
    return 0;
}
```

Python 参考代码

```
n, m = [int(i) for i in input().split()]
a = []
z = [0 for i in range(m)]
h = 0
for i in range(n):
    a.append([])
    row = [int(i) for i in input().split()]
    for j in range(m):
        a[-1].append(row[j])
        z[j] = max(z[j], row[j])
        h = max(h, z[j])
ans = [[' ' for j in range(m)] for i in range(h)]  #' ' * m 不行
for i in range(m):
    for j in range(h - 1, h - z[i] - 1, -1):
        ans[j][i] = '*'
for i in ans:
    for j in i:
        print(j, end='')
    print()
```

弥明的伤害分摊

题目所求即在约束条件 $x+y=k$ 下，使得函数 $\sqrt{p-a+x^2}+\sqrt{p-b+y^2}$ 取得最小值的 x,y

本题有多种解法，最简单的解法为数学推导法，有几何法(使用初中知识即可)和代数法(使用高等数学)，推导后能够直接得出答案，然后 $O(1)$ 计算答案即可(开方函数是 $O(1)$ 的)

本题还有其他解法。这些解法建立在如下结论的基础上：用 $x+y=k$ 消元 y 后，该函数是单峰凹函数，且最值必然在 $[0,k]$ 内取得

> (可以看几何法证明的前半段看证明过程证明最值取值范围，而单峰凹函数的结论可以拆分函数为两部分，一部分单调递增、一部分单调递减，不难得出)

求单峰函数最值，可以直接用三分法来做，也可以用爬山算法来做；本题没有必要用模拟退火来做。三分和爬山都是比较板子的，直接看代码实现细节即可，不具体细述算法思路了，可自行查阅网络

三分法和爬山算法在本题既可以预处理也可以每次输入一次现推一次。预处理是基于一个结论： k 输入的不同，答案会成比例发生变化。这点可以由下述严格数学证明得知。

假定精度为 $\epsilon=10^{-12}$，那么三分法预处理时间复杂度为 $\mathrm{O}(k+2\log_3\epsilon^{-1})$，非预处理为 $O(2k\log_3\epsilon^{-1})$

对爬山算法，若设定初温为 $t=1$，精度为 $\epsilon=10^{-15}$，降温系数为 $r=0.996$，那么预处理时间复杂度为 $O(\frac{\ln\frac{\epsilon}{t}}{\ln r}+k)$，非预处理时间复杂度为 $O(\frac{\ln\frac{\epsilon}{t}}{\ln r}k)$

严格数学证明(几何法)

首先需要理清一个结论：$x<0$ 或 $y<0$ 下，答案一定不会比 $x>0$ 且 $y>0$ 更优

这是因为最后贡献答案的部分为 x^2,y^2 ，当一方小于零时，平方是大于零的。具体而言，不妨设 $x<0$ ，则 $y>k$ ，那么必然有放缩不等式如下：

$$\sqrt{p-a+x^2}+\sqrt{p-b+y^2}>\sqrt{p-a+x^2}+\sqrt{p-b+k^2}>\sqrt{p-b+k^2}$$

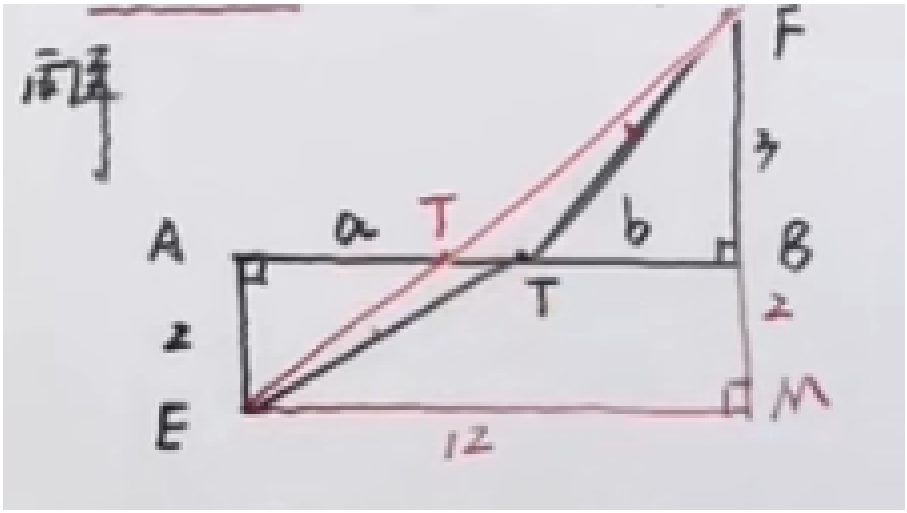
即 $x=0,y=k$ 优于一切 $x<0,y>k$ 的情况；同理 $x=k,y=0$ 由于一切 $y<0,x>k$

也就是说取得最小值必然取值范围为 $x>0,y>0$ ；事实上不难发现，这是一个单峰函数

不妨设线段 $|AB|=x+y=k,|AT|=x,|BT|=y,|AE|=\sqrt{p-a},|AF|=\sqrt{p-b}$

由于 $x>0,y>0$ ，所以他们在上述构造是有意义的(即不会出现负长度的边)

作 $AB\perp AE,AB\perp BF$ ，如图所示(黑色部分)：



根据勾股定理， $|ET|=\sqrt{p-a+x^2},|FT|=\sqrt{p-b+y^2}$ ，故所求为 $ET+FT$

在 ΔEFT 中，两边之和大于第三边，有：$|ET|+|FT|>|EF|$

延长 BF ，作 $EM//AB$ 交 BF 于点 M ，如上图所示(红色部分)

在矩形 $ABME$ 中， $|EM|=|AB|,|BM|=|AE|$ ，在 $Rt\Delta EMF$ 中，最小值为得：
$|EF|=\sqrt{k^2+(2p-a-b)^2}$

取得最小值时，在相似三角形 $\Delta AET,\Delta BFT$ 中，有 $\frac{|AT|}{|BT|}=\frac{|AE|}{|BF|}$ ，即 $\frac{x}{y}=\frac{\sqrt{p-a}}{\sqrt{p-b}}$ ①

联立 $x+y=k$ ②，代入消元法，可得：

$$\begin{cases}x=\dfrac{k\sqrt{p-a}}{\sqrt{p-a}+\sqrt{p-b}}\\ y=\dfrac{k\sqrt{p-b}}{\sqrt{p-a}+\sqrt{p-b}}\end{cases}$$

严格数学证明(代数法)

直接用高中数学方法求一元函数导数来求极值比较困难，考虑使用更简便的高等数学的多元函数求条件极值的拉格朗日乘数法

原函数为 $f(x,y)=\sqrt{p-a+x^2}+\sqrt{p-b+y^2}$，附加条件为 $x+y=k$ 即 $x+y-k=0$

设辅助函数 $L(x,y,\lambda)=\sqrt{p-a+x^2}+\sqrt{p-b+y^2}+\lambda(x+y-k)$

辅助函数 $L(x,y,\lambda)$ 的三个偏导数分别为：

$$\begin{aligned}\frac{\partial L}{\partial x}&=\frac{2x}{\sqrt{p-a+x^2}}+\lambda\\\frac{\partial L}{\partial y}&=\frac{2y}{\sqrt{p-b+y^2}}+\lambda\\\frac{\partial L}{\partial \lambda}&=x+y-k\end{aligned}$$

联立以 x,y,λ 为未知数的三元方程：

$$\begin{cases}\dfrac{\partial L}{\partial x}=0 & ①\\\dfrac{\partial L}{\partial y}=0 & ②\\\dfrac{\partial L}{\partial \lambda}=0 & ③\end{cases}$$

$①-②$，移项得：

$$\frac{x}{\sqrt{p-a+x^2}}=\frac{y}{\sqrt{p-b+y^2}}$$

两边平方，分母移项，得：

$$x^2(p-b+y^2)=y^2(p-a+x^2)$$

两边减去 x^2y^2，然后移项，两边开平方，得：

$$\frac{x}{y}=\frac{\sqrt{p-a}}{\sqrt{p-b}}\quad④$$

由 ③ 即约束条件，得 $x+y=k$，代入消元法联立 ③④ 不难可得：

$$\begin{cases}x=\dfrac{k\sqrt{p-a}}{\sqrt{p-a}+\sqrt{p-b}}\\y=\dfrac{k\sqrt{p-b}}{\sqrt{p-a}+\sqrt{p-b}}\end{cases}$$

由于极值有且仅有一个，根据题意可知，极值必然是极大值，也是全局最大值

> P.S. 如果你有更好的证明方法，欢迎分享 =w=

参考代码

C++ 数学推导法

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
#define sc(x) scanf("%lld", &x)
db p, a, b, m, n, k, s;
ll t;
signed main()
{
    sc(p), sc(a), sc(b);
    s = sqrt(p - a) + sqrt(p - b);
    m = sqrt(p - a) / s;
    n = sqrt(p - b) / s;
    sc(t);
    while (t--)
    {
        scanf("%lf", &k);
        printf("%.12lf %.12lf\n", m * k, n * k);
    }
    return 0;
}
```

C++ 三分法

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
ll p, a, b, t;
db k, lf, rf = 1, lc, rc;
db f(db x) //assume k=1
{
    return sqrt(p - a + x * x) + sqrt(p - b + (1 - x) * (1 - x));
}
signed main()
{
    scanf("%lld%lld%lld%lld", &p, &a, &b, &t);
    while (rf - lf > 1e-12)
    {
        lc = lf + (rf - lf) / 3;
        rc = rf - (rf - lf) / 3;
        if (f(lc) > f(rc))
        {
            lf = lc;
        }
        else
        {
            rf = rc;
        }
    }
    while (t--)
```

```
    {
        scanf("%lf", &k);
        printf("%lf %lf\n", k * lf, k * (1 - lf));
    }
    return 0;
}
```

C++ 爬山算法

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
ll p, a, b, q;
db k, t = 1.0, d, minx = 0.5, nowx, nowe, mine = 1e18;
db f(db x) //assume k=1
{
    return sqrt(p - a + x * x) + sqrt(p - b + (1 - x) * (1 - x));
}
signed main()
{
    scanf("%lld%lld%lld%lld", &p, &a, &b, &q);
    while (t > 1e-15) //t是温度参数
    {
        nowx = minx + (2.0 * rand() - RAND_MAX) / RAND_MAX * t;
        nowe = f(nowx);
        d = nowe - mine;
        if (d < 0) // || exp(-d / t) * RAND_MAX > rand()加上注释是模拟退火
        {          //不加是爬山算法
            minx = nowx;
        }
        if (d < 0)
        {
            mine = nowe;
        }
        t *= 0.996; //0.996是降温系数
    }
    while (q--)
    {
        scanf("%lf", &k);
        printf("%lf %lf\n", minx * k, (1 - minx) * k);
    }
    return 0;
}
```

桑泽的质监

如果暴力实现，可能会出现两个问题：

- 删除动态数组的复杂度为 $O(n)$，删除 m 次复杂度为 $O(nm)$
- 计算一次均值复杂度为 $O(n)$，计算方差为 $O(n)$，m 次为 $O(nm)$

所以要对这两个问题都进行优化，先考虑前者：

我们需要维护的数据结构需要满足三种操作：尾部插入、头部删除、随机修改

能够以较高复杂度满足条件的结构为：静态队列，能够以 $O(1)$ 完成三种操作，用静态数组+首尾指针实现即可，相信这一点不难

需要特别注意，由于可能会插入 m 次，所以数组长度需要开到 $n+m$

重点是第二个问题，下面对标准差进行化简：

$$s^2=\frac{\sum_{i=l}^{r}(x_i-\overline{x})^2}{r-l+1}$$

$$s^2=\frac{\sum_{i=l}^{r}(x_i^2-2x_i\overline{x}+\overline{x}^2)}{r-l+1}$$

$$s^2=\frac{\sum_{i=l}^{r}x_i^2-2(r-l+1)\overline{x}\sum_{i=l}^{r}x_i+(r-l+1)\overline{x}^2}{r-l+1}$$

$$s^2=\frac{\sum_{i=l}^{r}x_i^2-2\overline{x}\sum_{i=l}^{r}x_i+(r-l+1)\frac{\sum_{i=l}^{r}x_i}{r-l+1}\overline{x}}{r-l+1}$$

$$s^2=\frac{\sum_{i=l}^{r}x_i^2-2\overline{x}\sum_{i=l}^{r}x_i+(r-l+1)\frac{\sum_{i=l}^{r}x_i}{r-l+1}\overline{x}}{r-l+1}$$

$$s^2=\frac{\sum_{i=l}^{r}x_i^2-2\overline{x}\sum_{i=l}^{r}x_i+\overline{x}\sum_{i=l}^{r}x_i}{r-l+1}$$

$$s^2=\frac{\sum_{i=l}^{r}x_i^2-\overline{x}\sum_{i=l}^{r}x_i}{r-l+1}$$

$$s^2=\frac{\sum_{i=l}^{r}x_i^2-\frac{\sum_{i=l}^{r}x_i}{r-l+1}\times\sum_{i=l}^{r}x_i}{r-l+1}$$

设 $s_1=\sum_{i=l}^{r}x_i,s_2=\sum_{i=l}^{r}x_i^2,n=r-l+1$，有：

$$s^2=\frac{s_2}{n}-\frac{s_1^2}{n^2}$$

我们可以维护这三个变量，每次插入一个数 w 时，将 s_1 增加 w，将 s_2 增加 w^2，将 n 加一

每次删除一个数时，先在静态队列里取该数，然后将 s_1 减少 w，将 s_2 减少 2^2，将 n 减一

每次修改一个数时，等效于先删除再增加

于是每次计算标准差的复杂度为 $O(1)$，总时间复杂度为 $O(n+m)$，空间复杂度也为 $O(n+m)$

注意实现时避免零除错误。通常更推荐用 `/(n*n)` 而不是 `/n/n`；而且需要注意 p_i^2 必然会爆 `int`，建议开 `long long`，本题最大值为 $(n+m)(10^5)^2\approx10^{15}$，不会爆 `long long`

C++ 参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
#define sc(x) scanf("%lld", &x)
#define mn 200010
ll n, m, p[mn], lf = 1, rf, s1, s2, cmd, x, y; //有效编号范围[lf,rf]
signed main()
{
    sc(n), sc(m), rf = n;
    for (ll i = 1; i <= n; ++i)
    {
        sc(p[i]);
        s1 += p[i], s2 += p[i] * p[i];
    }
    while (m--)
    {
        sc(cmd);
        if (cmd == 1)
        {
            sc(x);
            p[++rf] = x;
            s1 += x, s2 += x * x;
        }
        else if (cmd == 2)
        {
            s1 -= p[lf], s2 -= p[lf] * p[lf];
            ++lf;
        }
        else if (cmd == 3)
        {
            sc(x), sc(y);
            s1 -= p[x], s2 -= p[x] * p[x];
            p[x] = y;
            s1 += y, s2 += y * y;
        }
        n = rf - lf + 1;
        printf("%lf\n", sqrt(1.0 * s2 / n - 1.0 * s1 * s1 / (n * n)));
    }
    return 0;
}
```

锦乐的记忆流

直接暴力来的时间复杂度显然是 $O(nq)$，会超时，考虑优化

可以使用前缀和进行优化，构造一个编号为 0 的桩子，设 $p_0 = 0$，记数组 $s1[i]$ 代表从编号为 0 的桩子顺着走到编号为 i 的桩子的总用时，那么不难得到递推等式：

$$s1[i] = s1[i-1] + \max(a[i] - a[i-1], 0)$$

含义是先从 0 走到 $i-1$，然后再从 $i-1$ 走到 i

相似地，记 $s2[i]$ 代表从编号为 i 的桩子逆着走到编号为 0 的桩子的总用时，同理有：

$$s2[i] = s2[i-1] + \max(a[i-1] - a[i], 0)$$

那么，根据前缀和的性质，从 x 顺着走到 $y(x < y)$ 的耗时为：

$$s1[y] - s1[x-1]$$

即从 0 到 y 减去从 0 到 x 的前一个桩子 $x-1$ 的这一段用时。若无法理解，不妨类比高等数学的积分性质或概率论的密度函数性质来理解

同理，从 x 逆着走到 $y(x > y)$ 耗时为：

$$s2[x] - s2[y-1]$$

题目是一个环，通常而言处理环习惯上可以转化为处理一个二倍长度的线段，即把环复制一份，首尾相连，形成长为 $2n$ 的链，原题转化为有一个长为 $2n$ 的桩子链，那么从环上 u 到 $v(u < v)$ 可以转化为 $u \leftrightarrow v$ 和 $v \leftrightarrow u + n$ 两条链

因此答案为：

$$\begin{cases} \min(s1[v] - s1[u-1], s2[u+n] - s2[v]) & ,u < v \\ \min(s1[v+n] - s1[u], s2[u] - s2[v-1]) & ,u > v \end{cases}$$

查询前预处理好 $s1, s2$，则每次查询可以在 $O(1)$ 完成，时间复杂度为 $O(n+q)$，空间复杂度为 $O(2n)$

C++ 参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
#define sc(x) scanf("%lld", &x)
#define mn 200010
ll n, q, a[mn], u, v, s1[mn], s2[mn], lr, rl;
signed main()
{
    sc(n), sc(q);
    for (ll i = 1; i <= n; ++i)
    {
        sc(a[i]), a[i + n] = a[i];
    }
    for (ll i = 1; i <= 2 * n; ++i)
    {
        s1[i] = s1[i - 1] + max(a[i] - a[i - 1], 0LL); //从0到i
        s2[i] = s2[i - 1] + max(a[i - 1] - a[i], 0LL); //从i到0
    }
    while (q--)
    {
        sc(u), sc(v);
        if (u < v)
        {
            lr = s1[v] - s1[u - 1];
            rl = s2[u + n] - s2[v];
        }
        else
        {
            lr = s1[v + n] - s1[u];
```

```
            rl = s2[u] - s2[v - 1];
        }
        printf("%lld\n", min(lr, rl));
    }
    return 0;
}
```

白茶的猫猫自动机(Easy Version)

不难发现，自动机是一个有向图结构，所以本题可以建图，特别注意所有不在信号集合的边都会连向起点，所以该图是一个有向完全图。建图后从起点开始沿着图的边走 q 次即可。下面寻找合适的数据结构建图

题给条件保证了没有编号相同的边，但是没有保证没有重边，并且图的大小可能到达 2×10^4，因此邻接矩阵不可用(空间复杂度为 $O(n^2)$)

本题边带编号，要根据编号找到对应的边，注意到编号范围为 $t\in[1,10^5]$，那么假设开静态数组，可能需要 $O(nt)$ 的内存，所以不可以开静态数组

考虑使用普通的邻接矩阵，那么由题给条件，总边数不超过 10^5，所以空间复杂度可行；但是假设这些边它们都在一个点上(即形成菊花图)，要找到编号对应边需要 $O(10^5)$ 的复杂度，q 次询问需要 $O(10^5q)$，显然会超时

为此，考虑使用哈希表存边，在空间复杂度不变的情况下，若使用 `std::map`，可以用对数复杂度找出编号，总时间复杂度为 $O(q\log\sum m)$，可以过题(事实上本题是套了个图论皮的哈希表题目)

> 注：`std::map` 的本质是红黑树，哈希表实现的应该为 `std::unordered_map`，实际使用时二者效率相近

C++ 参考代码

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
#define sc(x) scanf("%lld", &x)
#define maxn 20010
ll n, q, m, t, s, a, now = 1;
map<ll, ll> e[maxn];
signed main()
{
    cin >> n >> q;
    for (ll i = 1; i <= n; ++i)
    {
        cin >> m;
        for (ll j = 1; j <= m; ++j)
        {
            cin >> s >> t;
            e[i][s] = t;
        }
    }
    while (q--)
    {
        cin >> a;
```

```
        now = e[now][a] ? e[now][a] : 1;
        printf("%lld ", now);
    }
    return 0;
}
```

白茶的猫猫自动机(Hard Version)

如果还像 Easy Version 一样一条一条边地遍历，显然不可行，因为随便一个行程长度就有 10^9 之高，一下子就 T 掉了，考虑如何优化

可以使用倍增的思想，维护一个 ST 表，记 $e[h][i][j]$ 表示从编号为 i 的状态为起点，连续接收 2^h 个编号为 j 聊天信号之后位于的状态是 $e[h][i][j]$

很显然， $h=0$ 时，$e[0][i][j]$ 存所有邻边(不存在该状态时初始化为 1)

那么递推状态为：$e[h][i][j]=e[h-1][e[h-1][i][j]][j]$ ，可以理解为先从 i 出发走 2^{h-1}步，得到状态 $e[h-1][i][j]$ ，再从这个状态出发再走 2^{h-1} 步，就得到从 i 出发走 2^h 步的答案

由于这个递推是倍增的，所以对 10^9 步，只需要倍增地处理 $h=\lceil \log_2 10^9 \rceil$ 即可，故时空复杂度为 $O(nt\log_2 v)$

处理出来后，对于任意想要走的步数 $v(1\le v\le 10^9)$ ，都可以把 v 拆分成若干 2 的幂的和，可以证明必然能够拆分，且拆分结果即为 v 的二进制表示，然后依次走这些 2 的幂次步即可，这个过程的时间复杂度为 $O(\log_2 v)$

故总时间复杂度为 $O(nt\log_2 v+q\log_2 v)$

C++ 参考代码

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
#define mn 102
#define lg 32
ll n, q, e[lg][mn][mn]; //e[h][i][j] 状态i接受2^h个信号j到达什么状态
signed main()
{
    sc(n), sc(q);
    for (ll i = 1; i <= n; ++i)
    {
        for (ll j = 1; j <= 100; ++j)
        {
            e[0][i][j] = 1;
        }
    }
    for (ll i = 1, m; i <= n; ++i)
    {
        sc(m);
        for (ll j = 1, s, t; j <= m; ++j)
        {
            sc(s), sc(t);
            e[0][i][s] = t;
        }
```

```
    }
    for (ll h = 1; h < lg; ++h)
    {
        for (ll i = 1; i <= n; ++i)
        {
            for (ll j = 1; j <= 100; ++j)
            {
                e[h][i][j] = e[h - 1][e[h - 1][i][j]][j];
            }
        }
    }
    ll now = 1, u, v;
    while (q--)
    {
        sc(u), sc(v);
        for (ll i = 0; v > 0; ++i, v >>= 1)
        {
            if (v & 1) //跳2^i步
            {
                now = e[i][now][u];
            }
        }
    }
    printf("%lld", now);
    return 0;
}
```

禾枫的仙人掌

删边时要维持原图是连通分量，那么可以发现结论为对于每个环，只能删除环上的一条边或不操作；对于非环部分，不可以删除。理由如下：

- 如果不操作一个环，显然不影响连通性
- 如果删掉环上任一条边，就会破环，此时环上其余边都将成为桥，显然桥是不能够删除的
- 非环部分每条边都是桥，不可以删除

那么假设有 k 个环，第 i 个环上有 a_i 条边，那么根据组合数学的加法原理，对于每个环，可以选择删除 (a_i 种方案) 或者不删除 (1 种方案)，共有 a_i+1 个方案；然后对每个环，根据乘法原理，将其乘起来即可。特别注意不可以一个环也不删，所有环都不删有且仅有一个方案，最后减去该方案即可，那么答案为：

$$-1+\prod_{i=1}^{k}(a_i+1)$$

找仙人掌的环及其统计环上边数，一种较为简单的解法是可以使用 DFS

由于仙人掌图是连通的，所以可以从任一点开始 DFS ，并在搜索过程标记搜索深度 d ，如果搜索时发现某个点 v 标记过深度，那么证明 v 访问过，且从当前搜索的 u 可以再次访问 v ，即形成了环。根据仙人掌的特性，在搜索 v 到 u 的过程沿途的所有点都是该环上的(仙人掌的定义为每条边最多在一个环内的无向连通图)，那么此时只需要逆着搜索过程来计数 (可以开一个父亲节点数组 fa 实现逆过程)，记录从 v 到 u 经过了多少个点即可

如果不理解上述过程的，可以结合代码，造例子作草稿推导一下过程，应该不难理解

深度 d 相当于 `vis` 标记，所以不走重复路，那么搜索的复杂度为 $O(m)$ ，故本题时间复杂度为 $O(m)$

C++ 参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
#define mn 100010
struct edge
{
    ll to, nx;
} e[mn * 4];
ll hd[mn], cnt, n, m, fa[mn], d[mn], ans = 1, mod = 1e9 + 7;
void adde(ll &u, ll &v)
{
    e[++cnt] = {v, hd[u]};
    hd[u] = cnt;
}
void dfs(ll u, ll f)
{
    fa[u] = f, d[u] = d[f] + 1;
    for (ll i = hd[u], v; i; i = e[i].nx)
    {
        v = e[i].to;
        if (v == f)
        {
            continue;
        }
        if (d[v])
        {
            if (d[u] < d[v])
            {
                continue;
            }
            ll num = 1;
            for (ll now = u; now != v; now = fa[now])
            {
                ++num;
            }
            ans = ans * (num + 1) % mod;
        }
        else
        {
            dfs(v, u);
        }
    }
}
signed main()
{
    sc(n), sc(m);
    for (ll i = 1, u, v; i <= m; ++i)
    {
        sc(u), sc(v), adde(u, v), adde(v, u);
    }
    dfs(1, 0);
    printf("%lld", (ans - 1 + mod) % mod);
```

```
    return 0;
}
```