先说几句前面的话

开场

- 给定一个由n个小写字母组成的字符串s
- 有m次形如：x L R的操作，如果x=1，则将s[L]~s[R]升序排序，如果x=0，则将s[L]~s[R]降序排序。
- 求最终序列，n,m≤100,000

- 用线段树维护区间内a~z的个数，每次询问拆成26个区间修改操作

模拟题解析与训练

湖南师大附中 许力

什么是模拟题

- 用代码模拟题意描述的全过程，最终得出结果的一类题
- 不涉及复杂的算法或数据结构
- 模拟题对算法的设计能力要求比较低，但是可能会使用一些简单的数据结构，主要是考察代码能力

A+B problem（普及P1难度）

计算两个正整数相加的和：a+b，结果按标准格式输出

-10^6≤a，b≤10^6

Sample Input	Sample Output
-100000 9	-99,991

分析

- 根据数据范围，至多会出现两个“,”符号间隔，那么分三段处理和即可

参考代码

```
int sum = a + b;
if (sum < 0)
    sum = -sum, printf("-");
if (sum >= 1000000) printf("%d,%03d,%03d", sum/1000000, (sum%1000000)/1000, sum%1000);
    else if (sum >= 1000) printf("%d,%03d", sum/1000, sum%1000);
        else printf("%d", sum);
```

模拟题

- 模拟题往往要特别注意细节，以及考虑特殊情况。考场上可以自己多找反例，以免拿不到满分

A+B problem II（普及P2/提高P1难度）

计算A、B两个多项式相加的和：

$(a_1x^{k-1}+a_2x^{k-2}+\cdots\cdots+a_kx^0) + (b_1x^{k'-1}+b_2x^{k'-2}+\cdots\cdots+b_{k'}x^0)$

$1 \leqslant k \leqslant 10$，$0 \leqslant a, b \leqslant 1{,}000$（a、b保留1位小数）

Sample Input	Sample Output
2 1 2.4 0 3.2 2 2 1.5 1 0.5	3 2 1.5 1 2.9 0 3.2

分析

- 多项式相加，实际上是对应幂次的系数相加
- 这样我们可以用数组来模拟，用数组元素下标记录幂次，系数存入下标对应位置

$$2.4x^1+3.2x^0$$
$$1.5x^2+0.5x^1$$
$$1.5x^2+2.9x^1+3.2x^0$$

参考代码

```
scanf("%d", &k);
for (int i = 0; i < k; ++ i)
    scanf("%d%f", &n, &m), a[n] = m;
scanf("%d", &k);
for (int i = 0; i < k; ++ i)
    scanf("%d%f", &n, &m), b[n] = m;

for (int i = 0; i < 20; ++ i)
    c[i] = a[i] + b[i];    //同次系数相加
for (int i = 20-1; i >= 0; -- i)
    if (c[i] != 0) printf("%d %.1f ", i, c[i]);
```

A*B problem（普及P2+/提高P1+难度）

计算A、B两个多项式相乘的积:

$(a_1x^{k-1}+a_2x^{k-2}+\cdots\cdots+a_kx^0) \times (b_1x^{k'-1}+b_2x^{k'-2}+\cdots\cdots+b_{k'}x^0)$

$1 \leqslant k \leqslant 10$，$0 \leqslant a$，$b \leqslant 1{,}000$（a、b保留1位小数）

Sample Input	Sample Output
2 1 2.4 0 3.2 2 2 1.5 1 0.5	3 3 3.6 2 6.0 1 1.6

分析

- 多项式相乘，即一个多项式和另一个多项式的每一项相乘，然后再相加

买铅笔（普及P1难度）

P 老师需要去商店买 n 支铅笔作为小朋友们参加NOIP 的礼物。她发现商店一共有3 种包装的铅笔，不同包装内的铅笔数量有可能不同，价格也有可能不同。为了公平起见，P 老师决定只买同一种包装的铅笔。

商店不允许将铅笔的包装拆开，因此P老师可能需要购买超过 n 支铅笔才够给小朋友们发礼物。

现在P 老师想知道，在商店每种包装的数量都足够的情况下，要买够至少 n 支铅笔**最少**需要花费多少钱。

Sample Input	Sample Output
57 2 2 50 30 30 27	54

分析

- 一共只有3种包装，因此模拟就可以了

作用	函数名	举例
向下取整	**floor**	floor(1.732)=1
向上取整	**ceil**	ceil(3.14)=4
整数绝对值	**abs**	abs(-9)=9
浮点数绝对值	**fabs**	fabs(-9.21)=9.21
算术平方根	**sqrt**	sqrt(9.9)=3.14
取最小值	**min**	min(3,5)=3
取最大值	**max**	max(3,5)=5
交换变量值	**swap**	swap(a,b)

参考代码

```
scanf("%d", &m);
for (int i = 1; i <= 3; ++i)
{
    scanf("%d%d", &a, &b);
    int k = ceil(m * 1.0 / a);
    // ceil向上取整
    ans = min(ans, k * b);
}
printf("%d", ans);
```

改编一下

- 如果是n种不同的包装呢?
- 背包DP

金币（普及P1难度）

国王将金币作为工资，发放给忠诚的骑士。第一天，骑士收到一枚金币；之后两天（第二天和第三天），每天收到两枚金币；之后三天（第四、五、六天），每天收到三枚金币；之后四天（第七、八、九、十天），每天收到四枚金币……；这种工资发放模式会一直这样延续下去：当连续 N 天每天收到 N 枚金币后，骑士会在之后的连续 N+1 天里，每天收到 N+1 枚金币。

请计算在前 K 天里，骑士一共获得了多少金币。

$1 \leqslant K \leqslant 10,000$

分析

- 注意每到发放当前标准的工资天数与工资标准相等时，就涨一个金币的工资即可

参考代码

```
int K, ans = 0, day = 1, j = 0;
// day记录每天领到的金币，j记录当前金币标准持续了几天
scanf("%d", &K);
for (int i = 1; i <= K; ++ i)
{
    ++ j;
    ans += day;
    if (j == day)    //要加金币了
        ++ day, j = 0;
 }
```

改编一下

- 如果将问题变为：已知国王有n枚金币，够发放几天的工资（傲娇的骑士每天收到的金币要足量，不然会拒收）

Sample Input	Sample Output
10	4
100000	2240

分析

- 实际上就是1^2+2^2+3^2+4^2+……≤n这个式子，求最大的 i

计数问题（普及P1难度）

试计算在区间 1 到 n 的所有整数中，数字 x（$0 \le x \le 9$）共出现了多少次？例如，在 1 到 11 中，即在 1、2、3、4、5、6、7、8、9、10、11 中，数字 1 出现了 4 次。

$1 \le n \le 1{,}000{,}000$

Sample Input	Sample Output
11 1	4

分析

- 直接1~n范围循环，逐个分离数字，复杂度≈O(n)
- 需要注意的是循环体内不要修改循环变量

参考代码

```
scanf("%d%d", &n, &x);
for (int i = 1; i <= n; ++ i)
{
    int m = i;  //循环体中不要修改循环变量
    while (m > 0)
    {
        int k = m % 10;
        if (k == x) ++ cnt;
        m /= 10;
     }
}
printf("%d", cnt);
```

排序函数 sort

```
scanf("%d",&n);
for (int i = 0; i < n; ++i)
    scanf("%d",&a[i]);

sort(a,a + n);     // a数组a[0]~a[n-1]元素从小到大排序
```

从大到小

```
bool cmp(int a, int b)
{
    return a > b;
}


sort(a, a + n, cmp);
```

多关键字排序

```
struct pack
{
    int weight;
    int value;
} p[maxn];

bool cmp(pack a, pack b)
{
    if (a.weight < b.weight) return true;
    else if (a.weight == b.weight)
    {
        if (a.value < b.value) return true;
        return false;
    }
    return false;
}
// weight为排序第一关键字，value为第二关键字

sort(p, p + n, cmp);
```

去重函数 unique

```
scanf("%d", &n);
for (int i = 0; i < n; ++ i)
    scanf("%d", &a[i]);
int len = unique(a, a + n) - a;
//去重并获取去重后的实际长度
```

珠心算测验（普及P1难度）

珠心算是一种通过在脑中模拟算盘变化来完成快速运算的一种计算技术。珠心算训练，既能够开发智力，又能够为日常生活带来很多便利，因而在很多学校得到普及。

某学校的珠心算老师采用一种快速考察珠心算加法能力的测验方法。他随机生成一个正整数集合，集合中的数各不相同，然后要求学生回答：其中有多少个数，恰好等于集合中另外两个（不同的）数之和？

最近老师出了一些测验题，请你帮忙求出答案。

$3 \leqslant n \leqslant 100$

Sample Input	Sample Output
4 1 2 3 4	2

分析

- n的范围很小，支持n^3，因此排序后扫一遍就可以了

参考代码

```
sort(a, a + n);
for (int i = 0; i < n; ++ i)
    for (int j = i + 1; j < n; ++ j)
        for (int k = j + 1; k < n; ++ k)
            if (a[i] + a[j] == a[k] && flag[k] == 0)
            {
                ans ++;
                flag[k] = 1;
            }
```

继续分析

- 如果数据范围扩大，n^3过不了，还能做吗？
- 排序后枚举和sum，L、R从两端向中间扫一遍，复杂度O(n^2)

参考代码

```
sort(a, a + n);
for (int sum = n - 1; sum >= 2; -- sum)
{
    int L = 1, R = sum - 1;
    while (L < R)
    {
        if (a[L] + a[R] < a[sum]) ++ L;
        else if (a[L] + a[R] > a[sum]) -- R;
        else {++ ans; break;}
    }
}
```

二维数组

- 计算空间开销

```
381
--------------------------------
Process exited after 0.02165 seconds
with return value 0
请按任意键继续. . .
```

```
#include<iostream>
using namespace std;
#define maxn 10000
int a[maxn][maxn];
int main()
{
    cout << sizeof(int) * maxn * maxn / (1024 * 1024);
    return 0;
 }
```

扫雷游戏（普及P2-/提高P1-难度）

扫雷游戏是一款十分经典的单机小游戏。在 n 行 m 列的雷区中有一些格子含有地雷（称之为地雷格），其他格子不含地雷（称之为非地雷格）。玩家翻开一个非地雷格时，该格将会出现一个数字——提示周围格子中有多少个是地雷格。游戏的目标是在不翻出任何地雷格的条件下，找出所有的非地雷格。

现在给出n行m列的雷区中的地雷分布，要求计算出每个非地雷格周围的地雷格数。

注：一个格子的周围格子包括其上、下、左、右、左上、右上、左下、右下八个方向上与之直接相邻的格子。

$1 \leqslant n \leqslant 100$，$1 \leqslant m \leqslant 100$

分析

- 数据范围可以支持n^2，那么直接把地雷分布用二维数组存起来，然后扫一遍即可

- 无需开字符数组，直接开 int 数组存储0/1，更方便统计

Sample Input	Sample Output
3 3 *?? ??? ?*?	*10 221 1*1
2 3 ?*? *??	2*1 *21

参考代码

```
char x;
for (int i = 1; i <= n; ++ i)
    for (int j = 1; j <= m; ++ j)
    {
        cin >> x;
        if (x == '*') a[i][j] = 1;
    }
for (int i = 1; i <= n; ++ i)
    for (int j = 1; j <= m; ++ j)
        ans[i][j] = a[i-1][j-1] + a[i-1][j] + a[i-1][j+1] + a[i][j-1] + a[i][j+1] + a[i+1][j-1] + a[i+1][j] + a[i+1][j+1];
//把八个方向的地雷数加起来
```

边界

- 这种二维数组模拟的题，常常会面临处理边界的问题，一般修改循环范围就可以解决
- 必要时还可以根据题意在边界外围做一圈预处理

神奇的幻方（普及P2/提高P1难度）

幻方是一种很神奇的 $N*N$ 矩阵：它由数字 $1,2,3,\ldots\ldots,N*N$ 构成，且每行、每列及两条对角线上的数字之和都相同。

当 N 为奇数时，我们可以通过以下方法构建一个幻方：

首先将 1 写在第一行的中间。

之后，按如下方式从小到大依次填写每个数 $K(K=2,3,\ldots,N*N)$：

1. 若 $(K-1)$ 在第一行但不在最后一列，则将 K 填在最后一行，$(K-1)$ 所在列的右一列；
2. 若 $(K-1)$ 在最后一列但不在第一行，则将 K 填在第一列，$(K-1)$ 所在行的上一行；
3. 若 $(K-1)$ 在第一行最后一列，则将 K 填在 $(K-1)$ 的正下方；
4. 若 $(K-1)$ 既不在第一行，也不在最后一列，如果 $(K-1)$ 的右上方还未填数，则将 K 填在$(K-1)$的右上方，否则将 K 填在 $(K-1)$ 的正下方。

现给定 N，请按上述方法构造 $N*N$ 的幻方。

$1 \le N \le 39$ 且 N 为奇数.

分析

- n的范围仅有39，直接按题意模拟

Sample Input	Sample Output
3	8 1 6 3 5 7 4 9 2

参考代码

```
a[i][j] = 1;
for (num = 2; num <= n * n; ++ num)
{
    if (i == 1 && j != n)
        i = n, ++ j;
    else if (i != 1 && j == n)
            -- i, j = 1;
    else if (i == 1 && j == n)
            ++ i;
    else if (! a[i-1][j+1])
            -- i, ++ j;
    else i ++;
    a[i][j] = num;
}
```

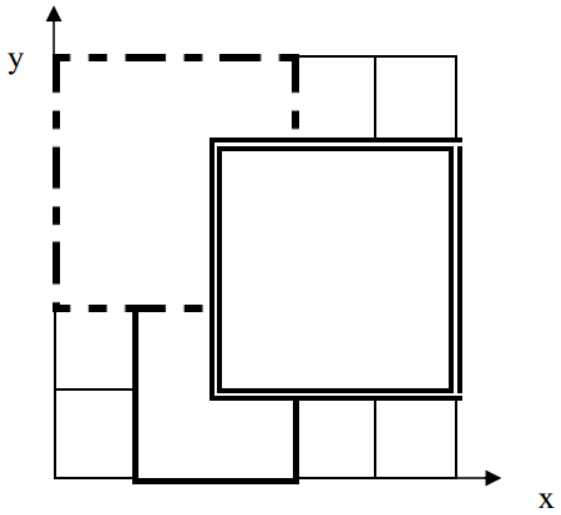
关于 else if

- 多重细分判断时，推荐多用 else if，比平行 if 安全
- switch也可以用，但不如 else if 清晰易读

铺地毯（普及P2/提高P1难度）

为了准备一个独特的颁奖典礼，组织者在会场的一片矩形区域（可看做是平面直角坐标系的第一象限）铺上一些矩形地毯。一共有 n 张地毯，编号从 1 到 n。现在将这些地毯按照编号从小到大的顺序平行于坐标轴先后铺设，后铺的地毯覆盖在前面已经铺好的地毯之上。地毯铺设完成后，组织者想知道覆盖地面某个点的最上面的那张地毯的编号。注意：在矩形地毯边界和四个顶点上的点也算被地毯覆盖。

$0 \leqslant n \leqslant 10{,}000$，$0 \leqslant a, b, g, k \leqslant 100{,}000$



分析

- 考虑到地毯会被相互覆盖，因此从后往前扫会更优
- 那么剩下的就只是根据点的坐标来判断了

- 因为每张地毯只给四个数据，所以开 n*4 的二维数组即可

Sample Input	Sample Output
3 1 0 2 3 0 2 3 3 2 1 3 3 2 2	3
3 1 0 2 3 0 2 3 3 2 1 3 3 4 5	-1

参考代码

```
for (int i = 1; i <= n; ++ i)
    for (int j = 0; j < 4; ++ j)
        scanf("%d", &a[i][j]);    //把地毯数据读入到一个 n × 4的二维数组
int x, y;
scanf("%d%d", &x, &y);
for (int i = n; i >= 1; -- i)
    if (a[i][0] <= x && a[i][0] + a[i][2] >= x && a[i][1] <= y && a[i][1] + a[i][3] >= y)
    {
        printf("%d", i);     //如果地毯四个点把目标点覆盖住
        return 0;
    }
printf("-1");
```

生活大爆炸版石头剪刀布（普及P2/提高P1难度）

表一 石头剪刀布升级版胜负关系

甲 \ 乙 / 甲对乙的结果	剪刀	石头	布	蜥蜴人	斯波克
剪刀	平	输	赢	赢	输
石头		平	输	赢	输
布			平	输	赢
蜥蜴人				平	赢
斯波克					平

现在，小 A 和小 B 尝试玩这种升级版的猜拳游戏。已知他们的出拳都是有周期性规律的，但周期长度不一定相等。例如：如果小 A 以“石头-布-石头-剪刀-蜥蜴人-斯波克”长度为 6 的周期出拳，那么他的出拳序列就是“石头-布-石头-剪刀-蜥蜴人-斯波克-石头-布-石头-剪刀-蜥蜴人-斯波克-……”，而如果小 B 以“剪刀-石头-布-斯波克-蜥蜴人”长度为 5 的周期出拳，那么他出拳的序列就是“剪刀-石头-布-斯波克-蜥蜴人-剪刀-石头-布-斯波克-蜥蜴人-……”

已知小 A 和小 B 一共进行 N 次猜拳。每一次赢的人得 1 分，输的得 0 分；平局两人都得 0 分。现请你统计 N 次猜拳结束之后两人的得分。

$0 < N \leq 200$

分析

- 这题的关键：

1. 把胜负关系那张表用二维数组预处理一下
2. 第 i 次出招和 第 i % na、i % nb 次出招是一样的，这样就完全不用考虑一轮出招周期结束后重新出招以及两人周期不等长等问题

Sample Input	Sample Output
10 5 6 0 1 2 3 4 0 3 4 2 1 0	6 2
9 5 5 0 1 2 3 4 1 0 3 2 4	4 4

参考代码

```
int score[5][5] = {{0,0,1,1,0},
                   {1,0,0,1,0},
                   {0,1,0,0,1},
                   {0,0,1,0,1},
                   {1,1,0,0,0}};

for (int i = 0; i < n; ++ i)
{
    ansa += score[a[i % na]][b[i % nb]];
    ansb += score[b[i % nb]][a[i % na]];
}
```

关于 % 运算

- 在带周期性质的运算中，% 运算往往能起妙用，考场上不妨往这个方向思考

- 另一个常有妙用的是异或运算^

开灯

在一条无限长的路上，有一排无限长的路灯，编号为1,2,3,4…

每一盏灯只有两种可能的状态，开或者关。如果按一下某一盏灯的开关，那么这盏灯的状态将发生改变。如果原来是开，将变成关。如果原来是关，将变成开

在刚开始的时候，所有的灯都是关的。小明每次可以进行如下的操作:

指定两个数：a,t（a为实数，t为正整数），将编号为[a],[2×a],…,[t×a]的灯各按一次。其中[k]表示实数k的整数部分

进行了n次操作后，小明突然发现，这个时候只有一盏灯是开的，小明很想知道这盏灯的编号

分析

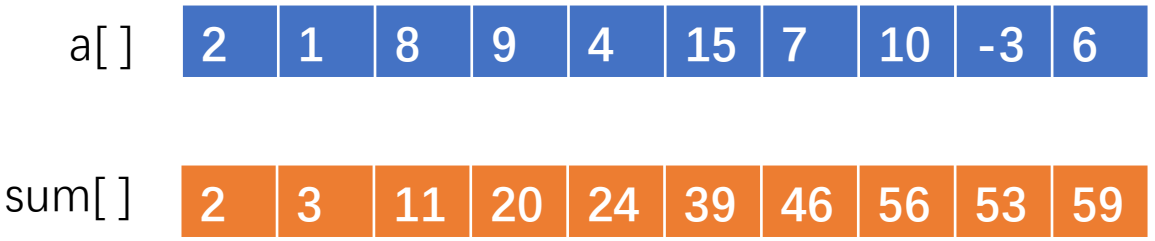
- 异或运算的特性是一个数异或它自己为0，与0异或得它自己
- 因为只有一盏灯是开的，那就表示对其他的灯做的开关操作都是成对的，因此把所有的操作异或起来就是答案

参考代码

```
for (int i = 1; i <= n; ++ i)
{
    scanf("%lf%d", &a, &t);
    for (int j = 1; j <= t; ++ j)
        ans ^= int (j * a);
        //利用异或运算特性，一个数与0异或得它自己
}
```

前缀和与二维前缀和

- 前缀和，O(n)预处理，O(1)查询




```
for (int i = 1; i <= n; ++ i)
    scanf("%d", &a[i]), sum[i] = sum[i-1] + a[i];
```

前缀和与二维前缀和

- 这个思想推广到二维

a[ ][ ]

2	1	8	9	4
0	4	-2	7	10
3	11	2	3	4
4	6	5	-3	1
15	2	4	5	-8

sum[ ][ ]

2	3	11	20	24
2	7	13	29	43
5	21	29	48	66
9	31	44	60	79
24	48	65	86	97

```
for (int i = 1; i <= n; ++ i)
    for (int j = 1; j <= m; ++ j)
        scanf("%d", &a[i][j]), sum[i][j] = sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1] + a[i];
```

无线网路发射器选址（普及P2/提高P1难度）

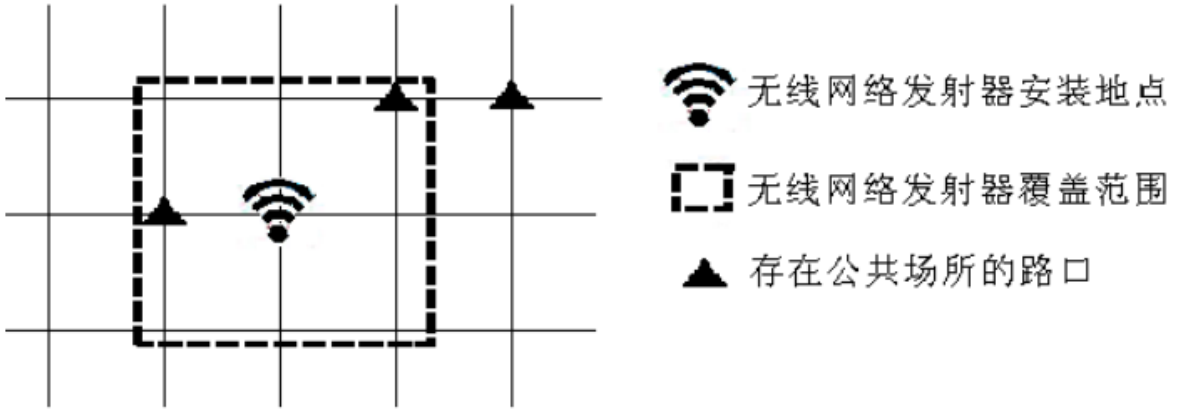
随着智能手机的日益普及，人们对无线网的需求日益增大。某城市决定对城市内的公共场所覆盖无线网。

假设该城市的布局为由严格平行的129条东西向街道和129条南北向街道所形成的网格状，并且相邻的平行街道之间的距离都是恒定值 1。东西向街道从北到南依次编号为0,1,2…128,南北向街道从西到东依次编号为0,1,2…128。

东西向街道和南北向街道相交形成路口，规定编号为 x 的南北向街道和编号为 y 的东西向街道形成的路口的坐标是（x, y）。在某些路口存在一定数量的公共场所。

由于政府财政问题，只能安装一个大型无线网络发射器。该无线网络发射器的传播范围是一个以该点为中心，边长为 $2*d$ 的正方形。传播范围包括正方形边界。

例如下图是一个 $d = 1$ 的无线网络发射器的覆盖范围示意图。



现在政府有关部门准备安装一个传播参数为 d 的无线网络发射器，希望你帮助他们在城市内找出合适的安装地点，使得覆盖的公共场所最多。

$1 \le d \le 20, 1 \le n \le 20$

分析

- 横纵都是[0,128]的矩阵，每个点都有个值，找一个点，使[x-d][y-d]~[x+d][y+d]范围内的点值和最大
- 考虑到 d 最大到20，因此当扫到<20范围时，边界需要处理：直接把边界扩大20，这样就避免了讨论出界的问题
- 然后直接扫一遍即可，注意还要统计方案数

Sample Input	Sample Output
1 2 4 4 10 6 6 20	1 30

参考代码

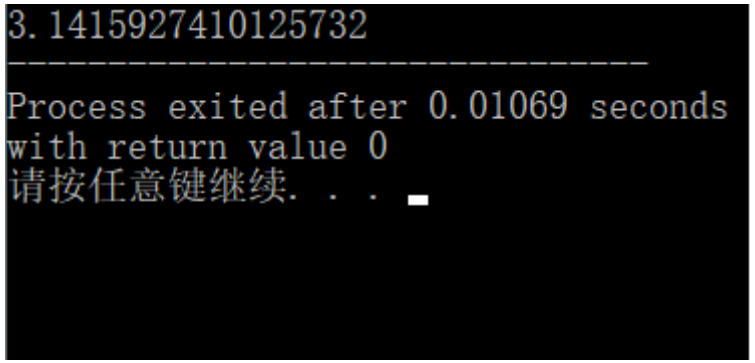
```
for (int i = 1; i <= n; ++ i)
{
    scanf("%d%d", &x, &y);
    scanf("%d", &k[x + 20][y + 20]);
    //读入矩阵时直接横、纵都平移 20以处理边界
}
for (int i = 20; i <= 128 + 20; ++ i)
    for (int j = 20; j <= 128 + 20; ++ j)
    {
        int sum = 0;
        for (int u = i - d; u <= i + d; ++ u)
            for (int v = j - d; v <= j + d; ++ v)
                sum += k[u][v];
        if (sum == ans) ++ cnt;     //统计方案数
        if (sum > ans) {ans = sum; cnt = 1;}    //更新最值
    }
```

浮点数处理

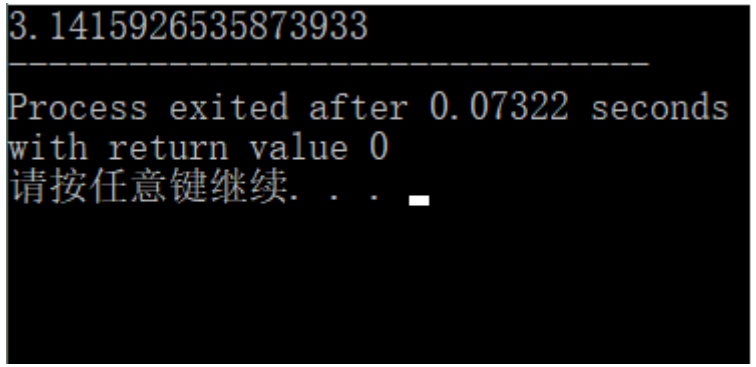
- 一般题目会要求浮点数精确到4位或者6位
- 但需要注意在类型转换过程中会损失精度

float 和 double

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    float x = 3.1415926535873932;
    printf("%.16f", x);
    return 0;
 }
```




```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    double x = 3.1415926535873932;
    printf("%.16lf", x);
    return 0;
 }
```



浮点数精度误差

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    float x = 1.3;
    x -= int(x);
    printf("%d", int(x * 10));
    return 0;
}
```

```
2
--------------------------------

Process exited after 0.1985 seconds w
ith return value 0
请按任意键继续. . .
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    double x = 1.3;
    x -= int(x);
    printf("%d", int(x * 10));
    return 0;
}
```

```
3
--------------------------------

Process exited after 0.01186 seconds
with return value 0
请按任意键继续. . .
```

字符处理

- 字符处理的主要意义，在于突破数据类型的范围限制，因此大数处理常需要借助于字符

- 字符有两种常见用法:
  - 字符数组
  - 字符串

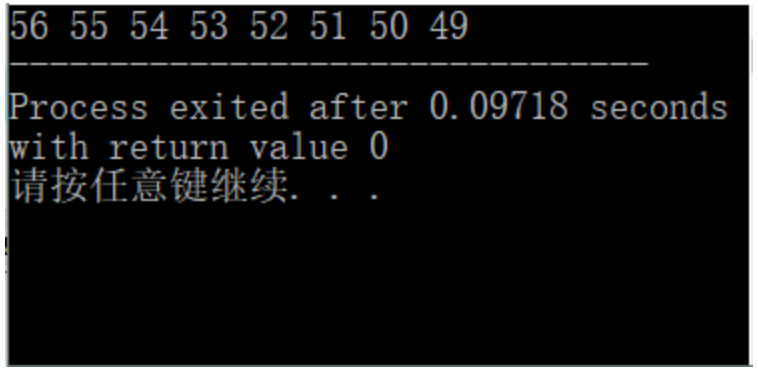
字符数组和字符串

- 字符数组实际上就是数组，因此数组支持的它都支持
- 字符串相对于字符数组整体性更好，也支持比较多的STL用法

字符数组和字符串

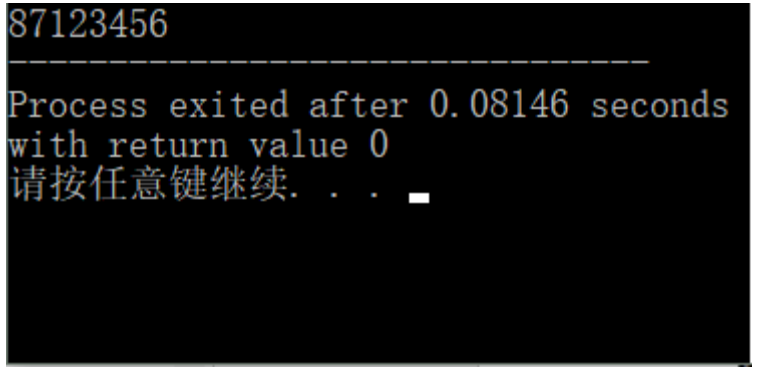
	字符数组 char s[ ]	字符串 string s
成行读入	gets(s)	getline(cin,s)
成行输出	puts(s)	
读入单个字符	c=getchar()	
输出单个字符	putchar(c)	
长度	strlen(s)	s.size() / s.length()
判等	strcmp(s1,s2) == 0	s1 == s2
复制	memcpy(s1,s2,streln(s2))	s1 = s2
连接	strcat(s1,s2)	s1 + s2
整型转换字符	sprintf	

字符数组转换函数 sprintf



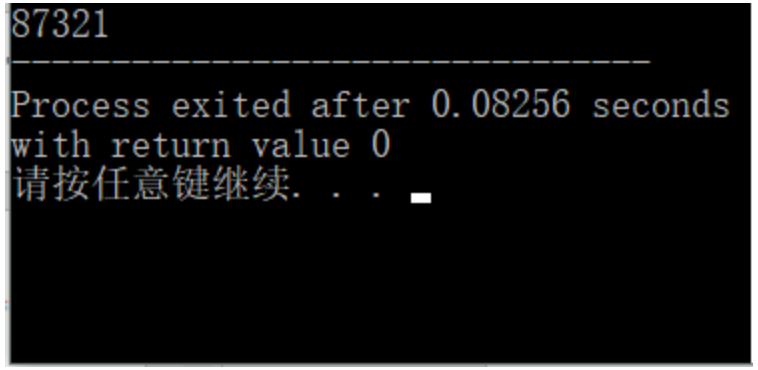

```
char s[10];
int x = 87654321;
sprintf(s, "%d", x);
for (int i = 0; i < strlen(s); ++ i)
    printf("%d ", s[i]);
```

字符串翻转函数 reverse




```
string n = "87654321";
reverse(n.begin() + 2, n.end());
cout << n;
```

字符串删除函数 erase




```
string n = "87654321";
n.erase(2, 3);
cout << n;
```

数字反转（普及P1难度）

- 给定一个整数，请将该数各位上数字反转得到一个新数

$$-1{,}000{,}000{,}000 \leq N \leq 1{,}000{,}000{,}000$$

Sample Input	Sample Output
-380	-83

怎样清理高位零

- 数字形态

```
int i = 0;
while (a[i] == 0) ++ i;
// i即为首个非零数字的位置
```

- 字符形态

```
while(*a.begin() == 0)
    a.erase(a.begin());
//直接删除高位零
```

参考代码

```
string n; cin >> n;
if (n[0] == '-')
    cout << "-", n.erase(n.begin());
reverse(n.begin(), n.end());
while (n[0] == '0') n.erase(n.begin());
cout << n;
```

*Ps：其中 n[0] 也可以写成 *n.begin()*

分析

- 这数据范围，也可以不用字符
- 负数预处理一下
- 另外这题带有明显的先入后出特征，因此也可以用栈来模拟

参考代码

```
if (n < 0)
    n = -n, printf("-");
while (n >= 1)
{
    m = m * 10 + n % 10;
    n /= 10;
}
printf("%d", m);
```

图书管理员（普及P2-/提高P1-难度）

图书馆中每本书都有一个图书编码，可以用于快速检索图书，这个图书编码是一个正整数。

每位借书的读者手中有一个需求码，这个需求码也是一个正整数。如果一本书的图书编码恰好以读者的需求码结尾，那么这本书就是这位读者所需要的。

小 D 刚刚当上图书馆的管理员，她知道图书馆里所有书的图书编码，她请你帮她写一个程序，对于每一位读者，求出他所需要的书中图书编码最小的那本书，如果没有他需要的书，请输出-1。

$1 \leqslant n \leqslant 1{,}000$，$1 \leqslant q \leqslant 1{,}000$

分析

- 这题也可以不需要处理字符，只要根据读者手中需求码的长度取模就好，因此可以预处理一个 mod[ ] 数组

- 因为求最小，所以 sort 一下

Sample Input	Sample Output
5 5 2123 1123 23 24 24 2 23 3 123 3 124 2 12 2 12	23 1123 -1 -1 -1

参考代码

```
sort(a, a + n);
while (q--)
{
    int L, m, ans = -1;
    scanf("%d%d", &L, &m);
    for (int i = 0; i < n; ++ i)
        if (a[i] % mod[L] == m) {ans = a[i]; break;}
    printf("%d\n", ans);
}
```

回文日期（普及P2/提高P1难度）

在日常生活中，通过年、月、日这三个要素可以表示出一个唯一确定的日期。

牛牛习惯用 8 位数字表示一个日期，其中，前 4 位代表年份，接下来 2 位代表月份，最后 2 位代表日期。显然：一个日期只有一种表示方法，而两个不同的日期的表示方法不会相同。

牛牛认为，一个日期是回文的，当且仅当表示这个日期的 8 位数字是回文的。现在，牛牛想知道：在他指定的两个日期之间（包含这两个日期本身），有多少个真实存在的日期是回文的。

Sample Input	Sample Output
20110101 20111231	1
20000101 20101231	2

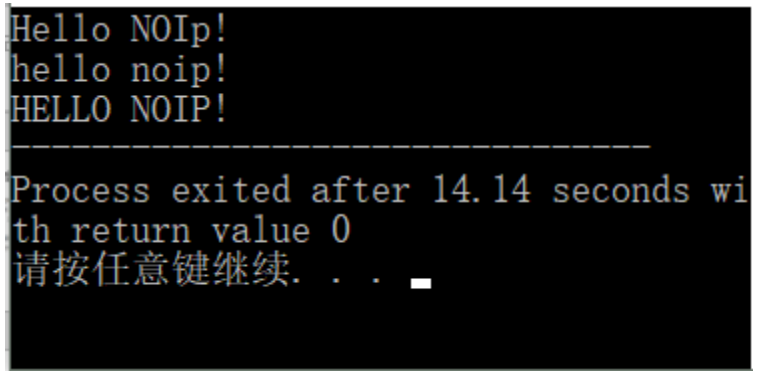
分析

- 这题没有必要逐天判断回文，因此可以构造回文后再判断是否合法
- 每月的天数没有规律，可以预处理一个 month[ ] 数组
- 最后闰年其实是无需判断的，因为0229是固定的

参考代码

```
scanf("%d%d", &d1, &d2);
for (int i = 1; i <= 12; i ++)
{
    for (int j = 1; j <= month[i]; j ++)
    {
        int temp = i * 100 + j;
        int date = (temp % 10 * 1000 + temp / 10 % 10 * 100 + temp / 100 % 10 * 10 + temp / 1000 % 10) * 10000 + temp;
        if (date >= d1 && date <= d2) ans ++;
    }
}
printf("%d", ans);
```

字符大小写转换函数 tolower / toupper




```
string s; getline(cin, s);
for(int i = 0; i < s.length(); i ++)
{
    char a = tolower(s[i]);
    printf("%c", a);
}
printf("\n");
for(int i = 0; i < s.length(); i ++)
{
    char b = toupper(s[i]);
    printf("%c", b);
}
```

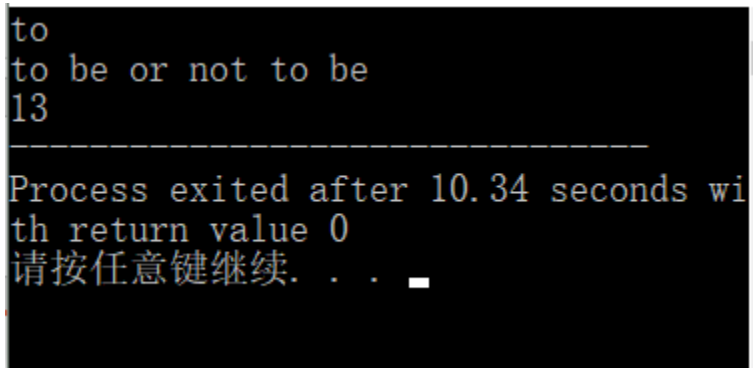
字符串定位函数 find

- 注意空格也占位置
- 没找到则返回-1


```
string a, b;
getline(cin, a);
getline(cin, b);
int pos = b.find(a);
cout << pos;
```

字符串定位函数 find

- 从串 b 的第2个位置开始定位串 a




```
string a, b;
getline(cin, a);
getline(cin, b);
int pos = b.find(a, 2);
cout << pos;
```

统计单词数（普及P2/提高P1难度）

一般的文本编辑器都有查找单词的功能，该功能可以快速定位特定单词在文章中的位置，有的还能统计出特定单词在文章中出现的次数。

现在，请你编程实现这一功能，具体要求是：给定一个单词，请你输出它在给定的文章中出现的次数和第一次出现的位置。**注意：匹配单词时，不区分大小写，但要求完全匹配，即给定单词必须与文章中的某一独立单词在不区分大小写的情况下完全相同（参见样例 1），如果给定单词仅是文章中某一单词的一部分则不算匹配（参见样例 2）。**

$1 \le$ 文章长度 $\le 1{,}000{,}000$.

Sample Input	Sample Output
To to be or not to be is a question	2 0
to Did the Ottoman Empire lose its power at that time	-1

分析

- 这题主要的细节在于：暴力匹配容易出现出现 to 匹配 toto 的情况
- 因此要么每次读入一个单词，要么把读入的单词前后加上空格

参考代码

```
getline(cin, word);
getline(cin, text);
for (int i = 0; i < word.size(); i ++)
    word[i] = tolower(word[i]);
for (int i = 0; i < text.size(); i ++)
    text[i] = tolower(text[i]);
word = ' ' + word + ' ';
text = ' ' + text + ' ';
int first = text.find(word);
for (int k = 0; (k = text.find(word, k)) >= 0; k += word.size() - 1)
    ans ++;
if (ans != 0) printf("%d %d", ans, first);
    else printf("-1");
```

Vigenère 密码（普及P2/提高P1难度）

16 世纪法国外交家 Blaise de Vigenère 设计了一种多表密码加密算法——Vigenère 密码。Vigenère 密码的加密解密算法简单易用，且破译难度比较高，曾在美国南北战争中为南军所广泛使用。

在密码学中，我们称需要加密的信息为明文，用 M 表示；称加密后的信息为密文，用 C 表示；而密钥是一种参数，是将明文转换为密文或将密文转换为明文的算法中输入的数据，记为 k。 在 Vigenère 密码中，密钥 k 是一个字母串，$k=k_1k_2\cdots k_n$。当明文 $M=m_1m_2\cdots m_n$ 时，得到的密文 $C=c_1c_2\cdots c_n$，其中 $c_i=m_i®k_i$，运算®的规则如下表所示：

Vigenère 密码（普及P2/提高P1难度）

®	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Vigenère 密码（普及P2/提高P1难度）

Vigenère 加密在操作时需要注意：

1. ®运算忽略参与运算的字母的大小写，并保持字母在明文 M 中的大小写形式；
2. 当明文 M 的长度大于密钥 k 的长度时，将密钥 k 重复使用。

例如，明文 M=Helloworld，密钥 k=abc 时，密文 C=Hfnlpyosnd。

明文	H	e	l	l	o	w	o	r	l	d
密钥	a	b	c	a	b	c	a	b	c	a
密文	H	f	n	l	p	y	o	s	n	d

Sample Input	Sample Output
CompleteVictory Yvqgpxaimmklongnzfwpvxmniytm	Wherethereisawillthereisaway

分析

- 那张庞大的表其实是迷惑人的，只要知道明文和密文之间其实就是按照密钥往后递推顺延就够了
- 然后密钥长度不够就需要拼接
- 计算差值的时候，如果小于26不够减，则需要“向高位借位”，否则会输出 “a” 或者 “A” 前面的字符

- 注意明文和密文不要弄反

参考代码

```
int temp;  // temp记录密文和密钥间的差值
scanf("%s%s", k, c);
int lenk = strlen(k); int lenc = strlen(c);
if (lenk < lenc)  //若密钥长度不够，则需要拼接
    for (int i = lenk; i < lenc; ++ i)
        k[i] = k[i - lenk];
for (int i = 0; i < lenc; ++ i)
{
    if (k[i] >= 'A' && k[i] <= 'Z') temp = k[i] - 'A';
    if (k[i] >= 'a' && k[i] <= 'z') temp = k[i] - 'a';
    //大小写分开处理，保持大小写不变
    m[i] = c[i] - temp;  // 取得密文对应的明文
    if (c[i] >= 'A' && c[i] <= 'Z')
        if (m[i] < 'A') m[i] += 26;
    //如果密文减去密钥得到的数字已经小于A，就需要借26过来
    if (c[i] >= 'a' && c[i] <= 'z')
        if (m[i] < 'a') m[i] += 26;
    //同上
}
```

玩具谜题（普及P2/提高P1难度）

小南有一套可爱的玩具小人，它们各有不同的职业。

有一天，这些玩具小人把小南的眼镜藏了起来。小南发现玩具小人们围成了一个圈，它们有的面朝圈内，有的面朝圈外。如下图：

玩具谜题（普及P2/提高P1难度）

这时singer 告诉小南一个谜题："眼镜藏在我左数第 3 个玩具小人的右数第 1 个玩具小人的左数第 2 个玩具小人那里。"

小南发现，这个谜题中玩具小人的朝向非常关键，因为朝内和朝外的玩具小人的左右方向是相反的：面朝圈内的玩具小人，它的左边是顺时针方向，右边是逆时针方向；而面向圈外的玩具小人，它的左边是逆时针方向，右边是顺时针方向。

小南一边艰难地辨认着玩具小人，一边数着：

"singer 朝内，左数第 3 个是archer。

"archer 朝外，右数第 1 个是thinker。

"thinker 朝外，左数第 2 个是writer。

"所以眼镜藏在writer 这里！"

虽然成功找回了眼镜，但小南并没有放心。如果下次有更多的玩具小人藏他的眼镜，或是谜题的长度更长，他可能就无法找到眼镜了。所以小南希望你写程序帮他解决类似的谜题。这样的谜题具体可以描述为：

有 n 个玩具小人围成一圈，已知它们的职业和朝向。现在第 1 个玩具小人告诉小南一个包含 m 条指令的谜题，其中第 i 条指令形如"左数/右数第 s_i 个玩具小人"。你需要输出依次数完这些指令后，到达的玩具小人的职业。

n	m
$= 10^5$	$= 10^5$

分析

- 其实是一个普通的从1号点开始绕环问题，唯一的细节在于面向圈内还是圈外，那么只需要统一朝内方向后，原来朝外的反向处理即可

- 环的处理也可以用取模运算来做，方向加朝向的叠加处理也可以用异或运算来做

Sample Input	Sample Output
7 3 0 singer 0 reader 0 mengbier 1 thinker 1 archer 0 writer 1 mogician 0 3 1 1 0 2	writer

参考代码

```
for (int i = 1; i <= m; i ++)
{
    scanf("%d%d", &a, &s);
    if (p[pos].face) a = 1 - a;
    //如果朝外，就变成和朝内方向相反
    if (a == 0) {pos -= s; if (pos <= 0) pos += n;}
                                    //环如果小于等于零就变正
    else {pos += s; if (pos > n) pos -= n;}
                                //环如果超过了就缩小
}
```

寻宝（普及P3/提高P2难度）

传说很遥远的藏宝楼顶层藏着诱人的宝藏。小明历尽千辛万苦终于找到传说中的这个藏宝楼，藏宝楼的门口竖着一个木板，上面写有几个大字：寻宝说明书。说明书的内容如下：

藏宝楼共有 N+1 层，最上面一层是顶层，顶层有一个房间里面藏着宝藏。除了顶层外，藏宝楼另有 N 层，每层 M 个房间，这 M 个房间围成一圈并按逆时针方向依次编号为 0，…，M-1。其中一些房间有通往上一层的楼梯，每层楼的楼梯设计可能不同。每个房间里有一个指示牌，指示牌上有一个数字 x，表示从这个房间开始按逆时针方向选择第 x 个有楼梯的房间（假定该房间的编号为 k），从该房间上楼，上楼后到达上一层的 k 号房间。比如当前房间的指示牌上写着 2，则按逆时针方向开始尝试，找到第 2 个有楼梯的房间，从该房间上楼。如果当前房间本身就有楼梯通向上层，该房间作为第一个有楼梯的房间。

寻宝说明书的最后用红色大号字体写着：“寻宝须知：**帮助你找到每层上楼房间的指示牌上的数字（即每层第一个进入的房间内指示牌上的数字）总和为打开宝箱的密钥**”。

请帮助小明算出这个打开宝箱的密钥。

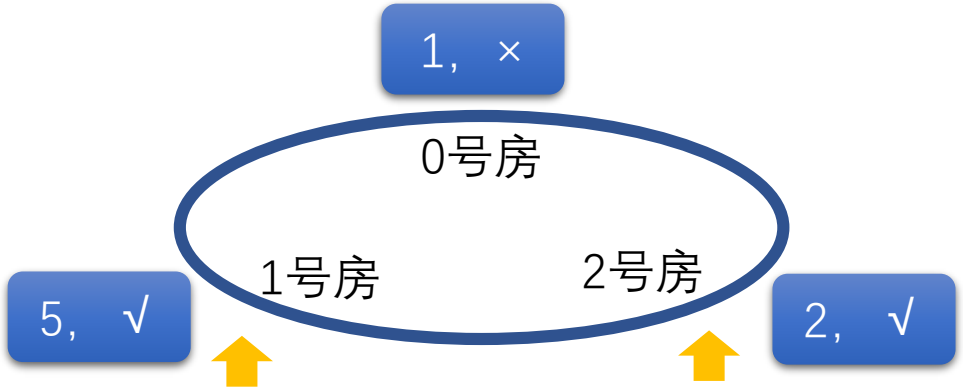
$0<N\leq 10000$，$0<M\leq 100$，$0<x\leq 1{,}000{,}000$

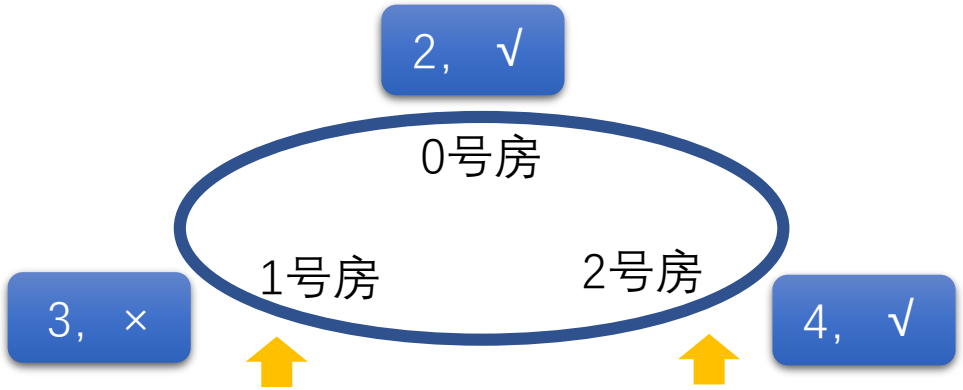
分析

- 大意是至多有10000个环，每个环至多100
- 然后在每圈里去数 x 个1

Sample Input	Sample Output
2 3 1 2 0 3 1 4 0 1 1 5 1 2 1	5

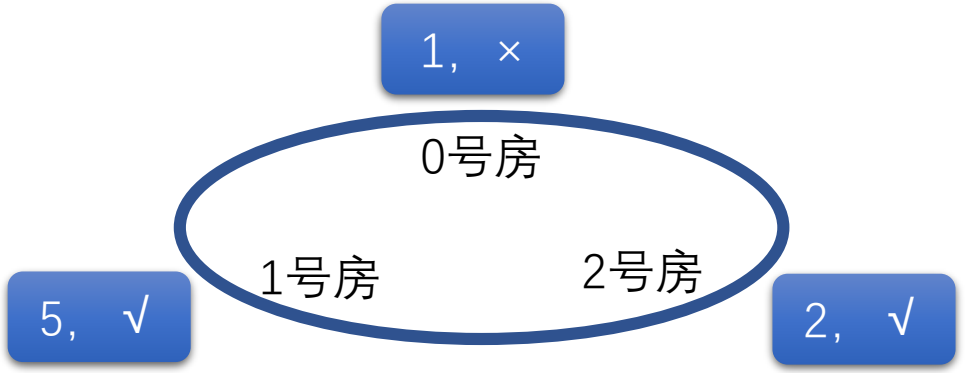
3 + 2

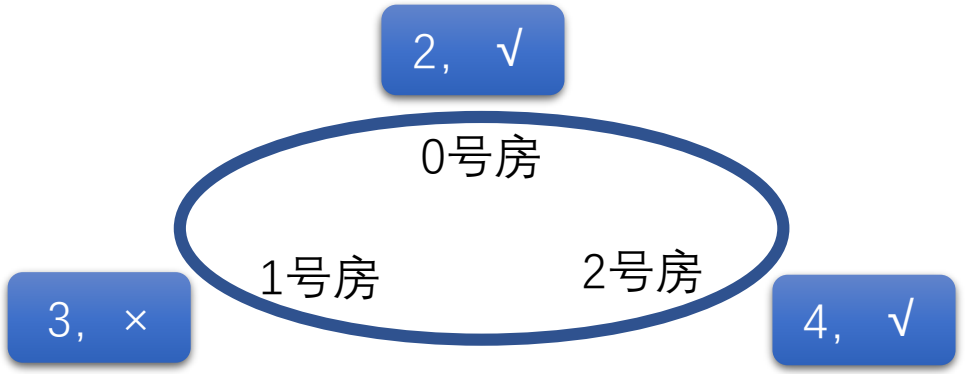




分析

- 这题 x 的数据范围巨大（10^6），再乘上层数（10^4），必定超时
- 因此每一层都要考虑取模，这样每层只要走一圈（10^2）就够了





参考代码

```
scanf("%d%d", &n, &m);
for (i = 1; i <= n; i ++)  // 第 i层
    for (j = 0; j < m; j ++) // 每层房间编号
    {
        scanf("%d%d", &f[i][j], &a[i][j]);
        // f记录是否有楼梯，a记录指示牌数字
        if (f[i][j]) one[i] ++;
        // one记录一层中 1的数量（有楼梯的房间）
    }
scanf("%d", &start);
```

参考

```
for (i = 1; i <= n; i ++)
{
    ans += a[i][start]; ans %= mod;
    if (a[i][start] % one[i] == 0) a[i][start] = one[i];
        else a[i][start] = a[i][start] % one[i];
    //取模用于防止循环太多圈
    int temp = 0, j = start;
    if (f[i][start]) temp ++;
    //如果起点房间有楼梯，计数加 1
    while (temp < a[i][start])
    //寻找上一层的起始位置
    {
        j ++;
        if (j > m) j = 0;
        if (f[i][j]) temp ++;
    }
    start = j;  //获得上一层的起始位置
}
```

快速读入优化

```
#include<bits/stdc++.h>
using namespace std;

long long read()
{
    char ch = getchar(); long long num = 0, f = 1;
    while (ch < '0' || ch > '9') {if (ch == '-') f = -1; ch = getchar();}
    while (ch >= '0' && ch <= '9'){num = num * 10 + ch - '0'; ch = getchar();}
    return num * f;
}

int main()
{
    long long n;
    n = read();
    printf("%lld", n);
    return 0;
}
```