R - Solution

类型	题号	考点	难度
代码基础	A	输出字符	1
	B	基本运算	1
	C	二维数组	2
	D	函数，数组	2~3
	E	循环，模运算	2~3
	F	位运算	2~3
	G	数组，散列表	2~3
	H	数组，循环	2~3
	I	循环，结构化编程	3
	J	结构化编程	3~4
	K	sscanf, 字符串，结构化编程	4
复杂模拟	L	模拟，位运算	4
	M	模拟，结构化编程	4
	N	输入，结构化编程	4
	O	结构化编程，输入	4
	P	复杂程度模拟	5
	Q	函数调用	5
	R	复杂模拟	5~6
qsort专题	S	排序一维数组	3
	T	排序指针数组	4
	U	排序下标数组	4
	V	排序二维数组	5
递归专题	W	递推or记忆化递归	4
	X	搜索	4
	Y	递归输出	4~5
	Z	搜索	4~5
	BA	分治	5~6

脑筋急转弯	BB	位运算	3
	BC	杂题	3
	BD	倒推，逻辑	4
思维题or算法题	BE	贪心，区间和	4
	BF	排序，数学	4~5
	BG	组合计数	4~5
	BH	前缀和	5
	BI	递推，动态规划DP	5
	BJ	队列，指针	5~6
	BK	快速幂	5~6
	BL	数论	6
	BM	搜索	6

A 输出字符字符！

难度	考点
1	输入输出

题目分析

注意 `\`, `'`, `"`, `%` 四种字符的输出。

示例代码-1

```
#include<stdio.h>
int main()
{
    puts("\\n");
    puts("%d");
    puts("\\\\");
    puts("\"\\OWA,\\allaccepted\"");
    return 0;
}
```

示例代码-2

```
#include <stdio.h>
int main()
{
    printf("\\n\n");
    printf("%%d\n");
    printf("\\\\\n");
    printf("\"\\OWA,\\allaccepted\"\n");
    return 0;
}
```

B dyc的三角形

难度	考点
1	基本运算

题目分析

设第三边的长度为 c，由三角形三边关系 $|a-b|<c<a+b$，可知 c 可能的取值个数为：

$$a+b-1-|a-b|=2\min(a,b)-1$$

示例代码

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int T, a, b;
    scanf("%d", &T);
    while(~scanf("%d%d", &a, &b))
        printf("%d\n", a + b - 1 - abs(a - b));
    return 0;
}
```

C 计算二次型

难度	考点
2	二维数组，循环

题意分析

设 $x=\begin{bmatrix}x_1\\x_2\\\vdots\\x_n\end{bmatrix}, A=\begin{bmatrix}a_{11}&a_{12}&\cdots&a_{1n}\\a_{21}&a_{22}&\cdots&a_{2n}\\\vdots&\vdots&\ddots&\vdots\\a_{n1}&a_{n2}&\cdots&a_{nn}\end{bmatrix}=[a_{ij}]$，则

$$x^TAx=\sum_{i=1}^{n}\sum_{j=1}^{n}x_ia_{ij}x_j$$

读入之后用两重循环遍历计算即可。

注意最终结果可能会超过 `int` 的范围，因此记录答案的变量要用 `long long` 类型。

当然也可以用计算两次矩阵乘法的方式求得结果。

示例代码

```
#include <stdio.h>
int n, a[100][100], x[100];
long long ans;
int main()
{
    scanf("%d", &n);
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j)
            scanf("%d", &a[i][j]);
    for(int i = 0; i < n; ++i)
        scanf("%d", &x[i]);
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j)
            ans += x[i] * a[i][j] * x[j];
    printf("%lld", ans);
    return 0;
}
```

D 计算混合积

难度	考点
2	函数

题目分析

计算 $(\boldsymbol{a}\times\boldsymbol{b})\cdot\boldsymbol{c}$，先计算 $\boldsymbol{a}\times\boldsymbol{b}$，再与 $\boldsymbol{c}$ 进行点积，最后输出答案。

注意到数据范围 $|x_i|,|y_i|,|z_i|\leq 100$，可以全程使用 int 型变量进行计算。

具体实现见示例代码：

示例代码 1 - 利用全局数组存储计算结果

```
#include <stdio.h>
int x[15], y[15], z[15];
int dot(int i, int j) //计算第i个向量与第j个向量的点积，返回点积结果
{
    return x[i] * x[j] + y[i] * y[j] + z[i] * z[j];
}
void cross(int i, int j, int k) //计算第i个向量叉乘第j个向量，结果存入第k个向量
{
    x[k] = y[i] * z[j] - y[j] * z[i];
    y[k] = z[i] * x[j] - z[j] * x[i];
    z[k] = x[i] * y[j] - x[j] * y[i];
}
int main()
{
    int n, t;
    scanf("%d%d", &n, &t);
    for(int i = 1; i <= n; ++i)
        scanf("%d%d%d", &x[i], &y[i], &z[i]);
    while(t--)
    {
        int i, j, k;
        scanf("%d%d%d", &i, &j, &k);
        cross(i, j, 0);
        printf("%d\n", dot(0, k));
    }
    return 0;
}
```

示例代码 2 - 结构体+函数

```
#include <stdio.h>
typedef struct _v{
    int x, y, z;
} v;
int dot(v a, v b) //计算a与b的点积
{
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
v cross(v a, v b) //计算a与b的叉积
{
    v ret = {a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y *
b.x};
    return ret;
}
int main()
{
    int n, t;
    v a[11];
    scanf("%d%d", &n, &t);
    for(int i = 1; i <= n; ++i)
        scanf("%d%d%d", &a[i].x, &a[i].y, &a[i].z);
    while(t--)
```

```
    {
        int i, j, k;
        scanf("%d%d%d", &i, &j, &k);
        printf("%d\n", dot(cross(a[i], a[j]), a[k]));
    }
    return 0;
}
```

补充

1、用行列式全部展开硬做这道题也完全没有问题，但是多麻烦呀，先计算叉积再计算点积相当于将行列式按最后一行展开，计算量更小。

2、这题不用函数也是完全可以做的，但是在未来，可能需要实现叉积、点积、混合积等功能的地方不只在一处用到，可能会在各种情况下需要用到同一个功能，因此封装成一个函数会使代码更加简洁清晰，且不容易发生错误。

Author: 哪吒

E Triangle：恐怖游轮

难度	考点
3	循环，取模

题目分析

由于直接存储当前数，对于我们学过的数据类型都是不可能存储下来的，所以我们如Hint所示，利用加法和乘法的取模性质，模拟数的生成过程

示例代码

```
#include <stdio.h>

#define maxn 1000005

char s[maxn];
int m;
long long ans = 0;

int main() {
    scanf("%s%d", s, &m);
    for(int i = 0; s[i]; ++i)
        ans = (ans * 10 + s[i] - '0') % m;
    printf("%lld", ans);
    return 0;
}
```

F 奇怪的最小数

难度	考点
3	位运算

题目分析

为了让异或和尽量小，这些 `1` 应当从高位到低位匹配 `num2` 中的 `1`，如果匹配完了还有剩余的 `1`，那么从低位到高位把 `0` 改为 `1` 。

示例代码 - 1

```
#include <stdio.h>

int cal(unsigned int a) {
    int ans = 0;
    while(a) {
        ans += a&1;
        a >>= 1;
    }
    return ans;
}

int main()
{
    unsigned int a, b;
    scanf("%u%u",&a, &b);
    int c1 = a, c2 = cal(b);
    for(;c1 < c2;++c1) b &= (b - 1);
    for(;c1 > c2;--c1) b |= (b + 1);
    printf("%u\n", b);
    return 0;
}
```

示例代码 - 2

```
#include <stdio.h>
int main()
{
    unsigned int a, b, x = 0;
    scanf("%u%u", &a, &b);
    for(int i = 31; a; --i)
        if(i < a || b >> i & 1) x |= 1 << i, --a;
    printf("%u", x);
    return 0;
}
```

G 区间合并 1.0

难度	考点
3	数组，散列表

题目分析

基本思路是用数组存储记录区间，有两种思路来做这道题，具体思路如下：

思路1

初始化数组为 0 ，每次读入区间，左端点为数组元素的下标，右端点为数组元素的值，最后遍历数组进行合并。

在读入时，如果存在左端点相同的区间，则取右端点较大者存入数组。

读入结束后遍历数组，在合并时，如果遍历到的数组元素的值为 0 ，说明没有以该位置为左端点的区间，则应跳过。

如果遇到值不为 0 的数组元素，开始合并该区间：遍历的区间的左端点应该小于等于当前正在合并的区间右端点，每次合并，更新当前正在合并的区间右端点为遍历到的区间右端点和当前正在合并的区间右端点的较大者，遍历到的区间不满足左端点小于当前合并区间右端点时，结束该区间的合并，输出后继续遍历区间。

思路1-示例代码

```
#include <stdio.h>
int range[1000005]; //全局数组初始化均为0
int main() {
    int l, r; //分别表示区间左、右端点
    while(~scanf("%d%d", &l, &r)) //不定组输入
        if(r > range[l]) range[l] = r; //读入时，左端点相同，取右端点较大者
    for(l = 0; l <= 1000000; ++l) {
        if(!range[l]) continue; //数组该位置元素值为0，说明没有以该位置为左端点的区间，跳过
        printf("%d ", l); //输出区间左端点
        for(r = range[l]; l <= r; ++l) //继续遍历数组，合并区间，注意用相同的计数变量，注意循环条件中的r是随着合并过程变化的
            if(range[l] > r) r = range[l]; //更新正在合并的区间右端点
        printf("%d\n", --l); //最后依次++l后不满足l<=r，此时l=r+1，因此应--l，再输出右端点
    }
    return 0;
}
```

思路2

初始化数组为 0 ，每次读入区间，标记数组下标分别为区间左端点和右端点的元素，最后遍历一遍输出区间。

标记方式可以选择标记左端点为 1 ，右端点为 -1 ，由于区间可能存在相同端点，将标记方式改为左端点的元素值自增 1 ，右端点的元素值自减 1 。即，读入之后的数组某位置的元素值表示以该位置为左端点的区间数量减去以该位置为右端点的区间数量。

读入结束后遍历数组，将数组元素值更改为原数组前 i 项值的累和（前缀和），则若在某位值数组元素由 0 变为正数，则说明该位置为合并后区间左端点，在某为值数组元素由正数变为 0 则说明该位置为合并后区间右端点。按输出要求输出即可。

思路2-示例代码

```
#include <stdio.h>
int range[1000005];
int main() {
    int l, r;
    while(~scanf("%d%d", &l, &r)) {
        ++range[l]; //左端点的元素值自增1
        --range[r]; //右端点的元素值自减1
    }
    for(int i = 1; i <= 1000000; ++i) {
        range[i] += range[i - 1]; //更改数组为前i项值的累和
        if(range[i] > 0 && range[i - 1] == 0) printf("%d ", i); //若上一个位置值为0，此位置值大于0，则该位置为合并后区间左端点
        if(range[i] == 0 && range[i - 1] > 0) printf("%d\n", i); //若上一个位置值大于0，此位置值为0，则该位置为合并后区间右端点
    }
    return 0;
}
```

补充

更推荐大家学习理解思路 1 。思路 1 的精髓在于用数组下标存储左端点，自动就将所有的左端点排好序了，仅需按照左端点递增顺序即数组遍历顺序根据右端点情况进行合并。这也启示我们，当无法按照这样存储区间时，可以将区间按照左端点递增进行排序，然后根据右端点情况进行合并。

思路 2 很巧妙，用了类似差分和前缀和的方式来记录区间，但是适用范围可能较小。

Author: 哪吒

H 散列查找-线性探测法

难度	考点
3	散列查找 一维数组

题目分析

根据题目和样例解释，可以得知散列查找（线性探测法）存入数据的基本流程。需要注意的点有：

- 当 $n > m$ 时，只需要 m 次输入即可。

  题目规定，当数组已满，则将接下来要存入的数直接遗弃。因此当 $n > m$ 时，前 m 个数据存入数组后，数组存满。之后的数据不会存在数组中。因此我们只需要循环 m 次将前 m 个数据存入数组后停止输入循环即可。

- 当 $n > m$ 时，输入循环共循环 m 次。当 $n < m$ 时，输入循环共循环 n 次。由此可见输入循环的次数是 m 与 n 的最小值。因此在循环过程中，数组一定没有存满，通过线性查找一定能找到当前输入数据的存入位置。所以无需判定在线性查找过程中是否返回原位置或数组是否存满。

示例代码

```
#include <stdio.h>
int main()
{
    unsigned int hash[1001]={0};
    int n,m;unsigned int x;
    scanf("%d%d",&m,&n);
    int min;
    if (m<n) min=m;
    else min=n;
    for (int i = 0; i < min; ++i) { //以m和n中的最小值作为循环次数
        scanf("%u",&x);
        unsigned int a=x%m;
        while (hash[a]!=0)
        {
            a++;
            a%=m;
        }
        hash[a]=x;
    }
    for (int i = 0; i < m; ++i) {
        printf("%u ",hash[i]);
    }
    return 0;
}
```

I 高端购物

难度	考点
3	循环，结构化编程

题目分析

题意为判断是否存在一种方案使得物品总数为偶数。

由于从每个商店选一个篮子买走，因此一共有三种类型的商店：

1. 商店里有的篮子中物品数量为奇数，有的篮子中物品数量为偶数；
2. 商店里只有物品数量为奇数的篮子；
3. 商店里只有物品数量为偶数的篮子。

统计三种类型商店的数量，进行判断：

- 若存在至少一个第一种商店，则无论在其他商店中购买的物品总数是奇数还是偶数，都可以通过在第一种商店中购买奇数个或偶数个物品使得最终总数为偶数。
- 若第一种商店不存在，则第二种商店数量为偶数时物品总数为偶数，否则物品总数为奇数。

示例代码

```
#include<stdio.h>
int main()
{
    int t, n, m, s1, s2, s3, x;
    scanf("%d", &t);
    while(t--)
    {
        s1 = 0, s2 = 0, s3 = 0;
        scanf("%d", &n);
        while(n--)
        {
            scanf("%d", &m);
            int f1 = 0, f2 = 0;
            for(int i = 1; i <= m; i++)
            {
                scanf("%d", &x);
                if(x % 2 == 1)
                    f1 = 1;
                if(x % 2 == 0)
                    f2 = 1;
            }
            if(f1 == 1 && f2 == 1)
                s1++;
            else if(f1 == 1 && f2 == 0)
                s2++;
            else
                s3++;
        }
        if(s1 >= 1)
            printf("siuuuuu\n");
        else
        {
            if(s2 % 2 == 1)
                printf("bzjt!\n");
            else
                printf("siuuuuu\n");
        }
    }
    return 0;
}
```

J Zuma and Combo

难度	考点
3~4	结构化编程

题目分析

从第一个字符串的尾部和第二个字符串的头部开始寻找相同的连续的字符。对于首尾可分为以下 4 种情况：

1. 首尾不同，直接退出；
2. 一个字符串为空，直接退出；
3. 首尾相同但总数小于3，直接退出；
4. 首尾相同且总数大于3，则开始消除第一个字符串的尾部和第二个字符串的头部的相同元素；

对于消除方法，第一个字符串可以直接将交界处的对应元素置 0 (`'\0'`)，第二个字符串可以使用 `string.h` 中的 `memmove` 函数，也可逐项复制或使用指针，注意逐项复制要先前移前面的元素，防止覆盖时产生错误。

示例代码 - 1

```
#include<stdio.h>
#include<string.h>
char s1[110], s2[110];
int main() {
    scanf("%s%s", s1, s2);
    char *p1 = s1, *p2 = s2;
    while (strlen(p1) != 0 && strlen(p2) != 0) {
        int i1, i2;
        int a1 = 1, a2 = 1;
        int len1 = strlen(p1);
        int len2 = strlen(p2);
        for (i1 = len1 - 2; i1 >= 0; i1--) {
            if (p1[i1] == p1[len1 - 1]) a1++;
            else break;
        }
        for (i2 = 1; i2 <= len2 - 1; i2++) {
            if (p2[i2] == p2[0]) a2++;
            else break;
        }
        if (p2[0] == p1[len1 - 1] && a1 + a2 >= 3) {
            p1[i1 + 1] = 0;
            //改变'\0'的位置即可改变strlen的大小与%s的输出
            p2 = p2 + i2;
        } else break;
    }
    printf("%s%s\n", p1, p2);
    return 0;
}
```

示例代码 - 2

```
#include<stdio.h>
#include<string.h>
char s1[110], s2[110];
int main() {
    scanf("%s%s", s1, s2);
    while (strlen(s1) != 0 && strlen(s2) != 0) {
```

```
        int i1, i2;
        int a1 = 1, a2 = 1;
        int len1 = strlen(s1);
        int len2 = strlen(s2);
        if(s1[len1-1] != s2[0]) break;
        for (i1 = len1 - 2; i1 >= 0; i1--) {
            if (s1[i1] == s1[len1 - 1])a1++;
            else break;
        }
        for (i2 = 1; i2 <= len2 - 1; i2++) {
            if (s2[i2] == s2[0])a2++;
            else break;
        }
        if (a1 + a2 >= 3) {
            s1[i1 + 1] = 0;
            //改变'\0'的位置即可改变strlen的大小与%s的输出
            memmove(s2, s2 + i2, sizeof(s2));
            //memmove函数，移动字符串内部的内容
        } else break;
    }
    printf("%s%s\n", s1, s2);
    return 0;
}
```

K 从科学计数法回归数字！

难度	考点
4	`sscanf`、字符串、结构化编程

题目分析

观察结果的结构，我们不难发现，只需要讨论 `e` 后的数字即可解决相应的问题：

1. `e` 后数字为 0，直接输出 `e` 前内容即可；
2. `e` 后数字为负，直接去除 `e` 前数字的小数点，在这个数字之前加入一个形如 $0.0000\ldots$ 的前缀即可，这个前缀里 0 的个数就是 `e` 后数字的绝对值；
3. `e` 后数字为正，需要讨论小数点的位置。如果 `e` 后的数字大于等于 `e` 前数字的小数位数，则不输出小数点，并且如果位数不够的话用 0 补齐；如果 `e` 后的数字小于 `e` 前数字的小数位数，则输出小数点，记 `e` 后的数字为 n，则小数点应在新数字后第 $n+1$ 位右侧输出。

示例代码

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define min(a,b) ((a)>(b))?(b):(a)
char s[1000005];
char ans[1000006];
int main() {
    scanf("%s", s);
```

```
    int n, j = 0;
    int len = strlen(s);
    for (int i = 0; i <= len - 1; i++) {
        if (s[i] == 'e') {
            sscanf(s + i + 1, "%d", &n);
            break;
        } else if (s[i] != '.') {
            ans[j++] = s[i];
        }
    }
    if (n == 0) {
        printf("%s", ans);
    } else if (n > 0) {
        int len2 = j - 1;
        int less = min(n, len2);
        for (int i = 0; i <= less; i++) {
            printf("%c", ans[i]);
        }
        if (j > less + 1) {
            printf(".");
            for (int i = less + 1; i <= j; i++) {
                printf("%c", ans[i]);
            }
        } else {
            for (int i = less + 1; i <= n; i++) {
                printf("0");
            }
        }
    } else {
        for (int i = 1; i <= abs(n); i++) {
            printf("0");
            if (1 == i) {
                printf(".");
            }
        }
        printf("%s", ans);
    }
}
```

L 均匀量化编码

难度	考点
4	模拟，位运算

题目解析

本题主要是考大家的代码能力的，根据题意模拟即可。

示例代码

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#define MAXN 5+
const double eps = 1e-8;

int main() {
    int v, N;
    scanf("%d%d", &v, &N);
    int L = 1 << N;
    double x;
    double V = v * 1.0;
    while (scanf("%lf", &x) != EOF) {
        if (x > V || fabs(x - V) < eps) {
            x = V - eps;
        } else if (x < -V) {
            x = -V;
        }
        double delta = (2.0 * V) / (1.0 * L);
        int k = (int)floor((x - (-V)) / delta + eps) ; //量化级数
        if (k < (L / 2)) { //负样值
            printf("0");
            for (int i = N - 2; i >= 0; --i) {
                printf("%d", !((k >> i) & 1));
            }
        } else {
            printf("1");
            for (int i = N - 2; i >= 0; --i) {
                printf("%d", ((k >> i) & 1));
            }
        }
        printf("\n");
    }
    return 0;
}
```

M Gino 的音乐练习

难度	考点
4	模拟

题目分析

注意考虑每个情况，保证计算出正确的难度值 w。

这题需要注意的点就是判断是否存在难度最高的谱面同时也是物量最多的谱面，因为物量相同的谱面可能很多，难度相同的谱面也可能很多，因此写这一段代码时需要仔细考虑。

示例代码

```
#include <stdio.h>
#define MAX(a, b) ((a) > (b) ? (a) : (b))
int main(void)
{
    int m[5], a, b, c, d, n, sum, level, maxsum = 0, maxlevel = 0, flag = 0;
    scanf("%d", &n);
    while (n--)
    {
        scanf("%d%d%d%d", &m[1], &m[2], &m[3], &m[4]);
        sum = m[1] + m[2] + m[3] + m[4];
        if (m[1] + m[2] >= 0.9 * sum)
        {
            c = d = 1;
        }
        else
        {
            c = m[3] > m[4] ? 1 : 2;
            d = 3 - c;
        }
        a = 2 * (c + d);
        if (m[2] <= 0.2 * m[1])
        {
            b = a + 3;
        }
        else
        {
            b = a + MAX(c, d);
        }
        level = a * m[1] + b * m[2] + c * m[3] + d * m[4];
        if (sum > 1000 && m[1] * 2 <= sum)
        {
            level += 200;
        }
        if (m[1] > 750)
        {
            level += 500;
        }
        if (maxsum < sum) // 最大物量改变
        {
            if (maxlevel > level) // 只有当这个谱面的难度值不是最大难度值的时候才将flag变量置0
                flag = 0;
            maxsum = sum;
        }
        if (maxlevel < level) // 最大难度值改变
        {
```

```
            maxlevel = level;
            if (maxsum == sum) // 只有当这个谱面的物量不是最大物量的时候才将flag变量置0，否则置1
                flag = 1;
            else
                flag = 0;
        }
        printf("%d\n", level);
    }
    printf(flag ? "Yes" : "No");
    return 0;
}
```

N 孤注一掷：百家乐

难度	考点
4	模拟

题目分析

按照要求模拟运算即可，需要注意的是计算时，变量较多，要注意到底是哪个变量和哪个变量进行运算。本题难度不大，考验同学们仔细程度，并且在发现错误后，要耐心寻找错误点。

示例代码

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
int Max(int a, int b)
{
    return a > b ? a : b;
}
int main()
{
    int money = 500000, bet = 100, max = 0, num = 0;
    int op, x1, x2;
    char a, b, c, d, s[10];
    for(int i = 1; i <= 1000; i++)
    {
        scanf("%s ", s);
        if(strlen(s) == 1) a = s[0];
        else a = '0';
        scanf("%s ", s);
        if(strlen(s) == 1) b = s[0];
        else b = '0';
        scanf("%s ", s);
        if(strlen(s) == 1) c = s[0];
        else c = '0';
        scanf("%s ", s);
        if(strlen(s) == 1) d = s[0];
```

```
        else d = '0';
        scanf("%d ", &op);
        if(!money) break;
        if(bet > money) bet = money;
        max = Max(max, bet);
        num++;
        switch(a)
        {
            case 'A':
                x1 = 1;
                break;
            case 'J':
                x1 = 11;
                break;
            case 'Q':
                x1 = 12;
                break;
            case 'K':
                x1 = 13;
                break;
            default:
                x1 = a - '0';
        }
        switch(b)
        {
            case 'A':
                x1 += 1;
                break;
            case 'J':
                x1 += 11;
                break;
            case 'Q':
                x1 += 12;
                break;
            case 'K':
                x1 += 13;
                break;
            default:
                x1 += b - '0';
        }
        switch(c)
        {
            case 'A':
                x2 = 1;
                break;
            case 'J':
                x2 = 11;
                break;
            case 'Q':
                x2 = 12;
                break;
            case 'K':
                x2 = 13;
                break;
            default:
                x2 = c - '0';
```

```
        }
        switch(d)
        {
            case 'A':
                x2 += 1;
                break;
            case 'J':
                x2 += 11;
                break;
            case 'Q':
                x2 += 12;
                break;
            case 'K':
                x2 += 13;
                break;
            default:
                x2 += d - '0';
        }
        x1 = x1 % 10;
        x2 = x2 % 10;
        if(x1 > x2 && op == 1) money += 0.95 * bet, bet = 100;
        else if(x1 < x2 && op == 2) money += bet, bet = 100;
        else if(x1 == x2 && op == 3) money += 8 * bet, bet = 100;
        else
        {
            money -= bet;
            bet *= 2;
        }
    }
    printf("%d\n%d\n%d", money, num, max);
    return 0;
}
```

拓展思考

如果给你 `n` 个向量坐标，对其中任意两个向量进行操作，你该如何做呢？

O 探险手册

难度	考点
4	模拟、`scanf` 小技巧

题目分析

本题是一个中等难度的模拟，由于机关位置未知，需要看最后一步到哪个位置，但又有可能中途就已经到达机关出口了，那么在按顺序执行操作的过程中就需要记录第一次到达每一个位置时走的步数~~（当然，重定向或者直接读进数组秒了）~~。
那么有两个需要注意的地方：

- 有可能某一次操作步数很多，导致多次撞墙，所以不能只计算撞墙一次的结果。

- 一种极特殊情况：$n=2$ 并且第一次操作是 $+a$ 操作，那么第一步还没走完就已经撞墙了，第一次到达第 2 段的时候走了 0 步，也就是说记录每一段第一次到达时步数的数组不能初始化为 0，可以初始化为 $-1,-2,...$ 这种（或者在这种情况下特判一下也可以）。

为了解决多次撞墙的问题，其实可以在撞墙的时候直接更新当前位置在左/右端，应走的步数为本次撞墙后剩余步数，同时更新方向，然后 **暂停读入** 直接把更新的步数和方向当作下一次循环读入的内容，直到这一次操作完成为止。

一个处理读入的办法：用这种办法可以直接区分开三种读入，并且可以直接读进来整数而不是字符串。

关于 `scanf` 的返回值：
`scanf` 如果成功读取了数据，就会返回读取元素的个数，否则返回 0。
那么我们可以写出如下代码（`dir` 的正负表示方向，大小表示一步走的段数，`a` 表示步数）
写这种表达有三个地方需要注意：

- 由于 `%d` 能跳过空格，但 `+` 和 `-` 本身不能跳过空格，所以前两个判断的读入条件正负号前面务必加上空格，写成 `" +%d"` 和 `" -%d"`。
- `scanf(" -%d",&a)` 的负号是直接吞掉的，并没有被读进 `a` 中，这三种情况下 `a` 始终读进来的是一个正整数。
- `else` 的情况下还什么都没有读，不要忘记在 `else` 后面加上 `scanf("%d",&a);`。

```
if(scanf(" -%d",&a)) dir*=-1;
else if(scanf(" +%d",&a)) dir*=2;
else scanf("%d",&a);
```

示例代码

```
#include <stdio.h>

int ans[1001];//ans[i]表示第一次到第i段走的步数

int main()
{
    int n,m,a=0,now=1,dir=1,cnt=0;//now是所处的段数，cnt是已走的步数
    for(int i=0;i<=1000;i++) ans[i]=-1;//初始化成-1
    scanf("%d%d",&n,&m);
    while(m>0)
    {
        if(a==0)//剩余步数为0，前一次行动已完毕，本次不需要暂停
        {
            if(scanf(" -%d",&a)) dir*=-1;
            else if(scanf(" +%d",&a)) dir*=2;
            else scanf("%d",&a);
        }
        switch(dir)
        {
            case 1://向右正常走
            {
                if(now+a<=n)//不会撞墙
                {
                    cnt+=a;//更新已走步数
                    now+=a;//更新当前位置
```

```
                a=0;//下次不需要暂停
                if(ans[now]==-1) ans[now]=cnt;//还没来过这段
            }
            else//会撞墙
            {
                cnt+=n-now;//更新已走步数
                a-=n-now;//更新剩余步数
                now=n;//更新当前位置
                if(ans[now]==-1) ans[now]=cnt;//还没来过这段
                dir=-1;//撞墙反向
            }
            break;//switych标配
        }
        //需要注意的是，在撞墙与不撞墙两种情况中，cnt、mow、a的更新顺序应当不同。

        case 2://向右信心满满地走，下面同上
        {
            if(now+a*2<=n)
            {
                cnt+=a;
                now+=a*2;
                a=0;
                if(ans[now]==-1) ans[now]=cnt;
                dir=1;
            }
            else
            {
                cnt+=(n-now)/2;
                a-=(n-now)/2;
                now=n;
                if(ans[now]==-1) ans[now]=cnt;
                dir=-1;
            }
            break;
        }
        case -1://向左正常走
        {
            if(now-a>=1)
            {
                cnt+=a;
                now-=a;
                a=0;
                if(ans[now]==-1) ans[now]=cnt;
            }
            else
            {
                cnt+=now-1;
                a-=now-1;
                now=1;
                if(ans[now]==-1) ans[now]=cnt;
                dir=1;
            }
            break;
        }
        case -2://向左信心满满地走
        {
```

```
                if(now-a*2>=1)
                {
                    cnt+=a;
                    now-=a*2;
                    a=0;
                    if(ans[now]==-1) ans[now]=cnt;
                    dir=-1;
                }
                else
                {
                    cnt+=(now-1)/2;
                    a-=(now-1)/2;
                    now=1;
                    if(ans[now]==-1) ans[now]=cnt;
                    dir=1;
                }
                break;
            }
        }
        if(a==0)//下一次可以读入，故用来记读入次数的m可以--；
        {
            m--;
        }
    }
    printf("%d",ans[now]);
    return 0;
}
```

P 选拔

难度	考点
5	模拟

题目解析

本题主要是考大家的代码能力的，没什么好说的，直接看示例代码吧。

提醒一下，本题其实就是北航某学院保研资格分配方案的复述，哈哈哈……

示例代码

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#define MAXN 505
#define MAXM 505

const double eps = 1e-6;
typedef struct {
```

```
    char name[10];//名字
    int m;//所属声部
    double x;//演奏水平
    double a;//综合成绩
    int z;//参赛意愿
    int xRank, aRank; //演奏水平在本声部中的名次，综合成绩在全吹奏乐队中的名次
    int income;//是否入选
} student;

student stu[MAXN];

int cmpByX(const void *a, const void *b) {
    //基于演奏水平从高向低的排序比较函数
    //当 a 对应的同学演奏水平比 b 高时，返回-1，否则返回1
    student A, B;
    A = (*(student *)a);
    B = (*(student *)b);
    if (fabs(A.x - B.x) <= eps)
        return strcmp(A.name, B.name);
    else
        return A.x > B.x ? -1 : 1;
}

int cmpByA(const void *a, const void *b) {
    //基于综合成绩从高向低的排序比较函数
    //当 a 对应的同学综合成绩比 b 高时，返回-1，否则返回1
    student A, B;
    A = (*(student *)a);
    B = (*(student *)b);
    if (fabs(A.a - B.a) <= eps) {
        if (fabs(A.x - B.x) <= eps)
            return strcmp(A.name, B.name);
        else
            return A.x > B.x ? -1 : 1;
    } else
        return A.a > B.a ? -1 : 1;
}

int main() {
    int n, m, b;
    scanf("%d%d%d", &n, &m, &b);
    int d[MAXM];//表示各声部的人数
    memset(d, 0, sizeof(d));
    for (int i = 1; i <= n; ++i) {
        scanf("%s", stu[i].name);
        scanf("%d", &stu[i].m);
        ++d[stu[i].m];
        scanf("%lf", &stu[i].x);
        double y;
        scanf("%lf", &y);
        stu[i].a = stu[i].x + y * 0.15;
        scanf("%d", &stu[i].z);
        stu[i].income = 0;
    }

    int c[MAXM];//各声部的第一轮名额
```

```
    for (int i = 1; i <= m; ++i)
        c[i] = (d[i] * b) / n;

    qsort(stu + 1, n, sizeof(stu[1]), cmpByX);
    int tempcntM[MAXM], lastRank[MAXM];
    double last[MAXM];
    memset(tempcntM, 0, sizeof(tempcntM));//用于计算演奏水平在本声部中的排名的辅助变量
    memset(last, 0, sizeof(last));//声部i里上一个人的演奏水平
    for (int i = 1; i <= n; ++i) {
        int M = stu[i].m;//当前人所属的声部
        ++tempcntM[M];//声部人数+1
        if (fabs(stu[i].x - last[M]) >= eps) //不是并列，更新排名
            lastRank[M] = tempcntM[M];

        stu[i].xRank = lastRank[M];
        last[M] = stu[i].x;
    }

    qsort(stu + 1, n, sizeof(stu[1]), cmpByA);
    int tempcnt = 0, lastRanka = 0;
    double lasta;
    for (int i = 1; i <= n; ++i) {
        ++tempcnt;
        if (fabs(stu[i].a - lasta) >= eps)
            lastRanka = tempcnt;
        stu[i].aRank = lastRanka;
        lasta = stu[i].a;
    }

    int round1 = 0;
    memset(tempcntM, 0, sizeof(tempcntM));//tempcntM存放当前声部已经使用了的拟分配名额
    for (int i = 1; i <= m; ++i) {//对每个声部
        fprintf(stderr, "c[%d]=%d\n", i, c[i]);
        for (int j = 1; j <= n; ++j) {
            if (stu[j].m != i)
                continue;  //对这个声部里的某个学生
            if (tempcntM[i] >= c[i])
                break;
            ++tempcntM[i];//拟分配一个名额
            fprintf(stderr, "round1 asking:%s Rankx=%d Ranka=%d\n", stu[j].name,
stu[j].xRank, stu[j].aRank);
            if (10 * stu[j].xRank <= 3 * d[m] && 2 * stu[j].aRank <= n) { //硬性条
件
                if (stu[j].z == 1) { //接收名额
                    stu[j].income = 1;
                    ++round1;//第一轮分配的名额加一
                    fprintf(stderr, "round1 success:%s\n", stu[j].name);
                } else {
                    ;//丢弃拟分配名额，就是啥也不用干
                }
            }

        }
    }

    int e = b - round1; //第二轮的名额数
```

```
    for (int i = 1; i <= n; ++i) {
        if (e <= 0)
            break;
        if (stu[i].income == 1) //已经有名额了
            continue;
        if (stu[i].z) {
            stu[i].income = 1;
            fprintf(stderr, "round2 success:%s\n", stu[i].name);
            --e;
        }
    }
    for (int i = 1; i <= n; ++i) {
        if (stu[i].income == 1)
            printf("%s %d\n", stu[i].name, stu[i].m);
    }
    return 0;
}
```

Q 加密的线性映射

难度	考点
5	模拟

题目分析

线性映射可以使用矩阵表示，本题使用 `0x3b7u` 进行异或加密。

注意到：

$$f\left(g\left(x\right)+256h\left(x\right)\right)=\left(f\circ\left(g+256h\right)\right)\left(x\right)$$

$$f\left(g\left(y\right)+256h\left(y\right)\right)+f\left(y\right)=\left(f\circ\left(g+256h\right)+f\right)\left(x\right)$$

我们有：

示例代码

```
# include <stdio.h>
# include <stdlib.h>

# define DiM 3
# define CiP 0x3b7u
unsigned int *(*eva(unsigned int *(*(*lin)[DiM])[DiM], unsigned int *(*vec)
[DiM]))[DiM] { unsigned int *(*res)[DiM] = (unsigned int *(*)[DiM])
malloc(sizeof(*res)); for (int j = 0; j < DiM; j++) { (*res)[j] = (unsigned int
*) malloc(sizeof(*(*res)[j])); *(*res)[j] = 0; for (int k = 0; k < DiM; k++) { *
(*res)[j] += (*(*(*lin)[k])[j] ^ CiP) * (*(*vec)[k] ^ CiP); } *(*res)[j] ^= CiP;
} return res; }
```

```c
unsigned int *(*(*plu(unsigned int *(*(*alp)[DiM])[DiM], unsigned int *(*(*bet)
[DiM])[DiM]))[DiM])[DiM] { unsigned int *(*(*res)[DiM])[DiM] = (unsigned int *(*
(*)[DiM])[DiM]) malloc(sizeof(*res)); for (int i = 0; i < DiM; i++) { (*res)[i] =
(unsigned int *(*)[DiM]) malloc(sizeof(*(*res)[i])); for (int j = 0; j < DiM;
j++) { (*(*res)[i])[j] = (unsigned int *) malloc(sizeof(*(*(*res)[i])[j])); *(*
(*res)[i])[j] = ((*(*(*alp)[i])[j] ^ CiP) + (*(*(*bet)[i])[j] ^ CiP)) ^ CiP; } }
return res; }
unsigned int *(*(*com(unsigned int *(*(*alp)[DiM])[DiM], unsigned int *(*(*bet)
[DiM])[DiM]))[DiM])[DiM] { unsigned int *(*(*res)[DiM])[DiM] = (unsigned int *(*
(*)[DiM])[DiM]) malloc(sizeof(*res)); for (int i = 0; i < DiM; i++) { (*res)[i] =
(unsigned int *(*)[DiM]) malloc(sizeof(*(*res)[i])); for (int j = 0; j < DiM;
j++) { (*(*res)[i])[j] = (unsigned int *) malloc(sizeof(*(*(*res)[i])[j])); *(*
(*res)[i])[j] = 0; for (int k = 0; k < DiM; k++) { *(*(*res)[i])[j] += (*(*(*alp)
[k])[j] ^ CiP) * (*(*(*bet)[i])[k] ^ CiP); } *(*(*res)[i])[j] ^= CiP; } } return
res; }
unsigned int *(*cev())[DiM] { unsigned int *(*res)[DiM] = (unsigned int *(*)
[DiM]) malloc(sizeof(*res)); for (int j = 0; j < DiM; j++) { (*res)[j] =
(unsigned int *) malloc(sizeof(*(*res)[j])); scanf("%3u", (*res)[j]); } return
res; }
unsigned int *(*(*nil())[DiM])[DiM] { unsigned int *(*(*res)[DiM])[DiM] =
(unsigned int *(*(*)[DiM])[DiM]) malloc(sizeof(*res)); for (int i = 0; i < DiM;
i++) { (*res)[i] = (unsigned int *(*)[DiM]) malloc(sizeof(*(*res)[i])); for (int
j = 0; j < DiM; j++) { (*(*res)[i])[j] = (unsigned int *) malloc(sizeof(*(*(*res)
[i])[j])); } } for (int i = 0; i < DiM; i++) { for (int j = 0; j < DiM; j++) {
scanf("%3u", (*(*res)[i])[j]); } } return res; }
void pri(unsigned int *(*vec)[DiM]) { for (int j = 0; j < DiM; j++) { printf("%u
", *(*vec)[j] ^ CiP); } printf("\n"); return; }
# define VeC(vec) unsigned int *(*vec)[DiM]
# define LiN(lin) unsigned int *(*(*lin)[DiM])[DiM]

int main(void) {
    VeC(y) = cev();                 // 定义空间向量 y 为此次加密读入一个空间向量的结果。
    LiN(g) = nil();                 // 定义线性映射 g 为此次加密读入一个线性映射的结果。
    VeC(x) = cev();                 // 定义空间向量 x 为此次加密读入一个空间向量的结果。
    LiN(h) = nil();                 // 定义线性映射 h 为此次加密读入一个线性映射的结果。
    LiN(f) = nil();                 // 定义线性映射 f 为此次加密读入一个线性映射的结果。
    LiN(p) = h;                     // 将线性映射 p 初始化为线性映射 f 。
    for (int i = 1; i < 256; i++) {
        p = plu(p, h);              // 将线性映射 p 修改为线性映射 p 加上线性映射 h 的结
果。
    }
    LiN(q) = com(f, plu(g, p));     // 定义线性映射 q 为线性映射 f 复合线性映射 g+p 的
结果。
    pri(eva(q, x));                 // 输出线性映射 q 位于空间向量 x 的解密结果。
    pri(eva(plu(q, f), y));         // 输出线性映射 q+f 位于空间向量 y 的解密结果。
    return 0;
}
```

R 雨果奖

难度	考点
5~6	6

题目解析

本题主要是考大家的代码能力的，没什么好说的，直接看示例代码吧。

示例代码里面有一个

```
fprintf(stderr,.....)
```

其实这个就实现了在题目里面所谓的「输出过程」功能。具体来说，我们平时的`printf`函数输出的东西都是在`stdout`标准输出流中的。而上面的`fprintf(stderr,...)`函数将内容输出到了`stderr`标准错误流中。我们的OJ评测时只取标准输出流的结果，而标准错误流会被忽视，但是控制台上则会同时显示标准输出流和标准错误流。

示例代码

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#define MAXN 5+100
#define MAXM 5+10000
const double eps = 1e-7;

int ticket[MAXM][MAXN] = {0};
int out[MAXN] = {0}, vote[MAXN] = {0};
int n, m;

int NoAwardTest(int x);
int RunOffTest(int a, int b);
void printPrize(int x);


int main() {
    fprintf(stderr, "开始运行\n");
    scanf("%d%d", &n, &m);
    memset(ticket, 255, sizeof(ticket));
    for (int i = 1; i <= m; ++i) {
        int cur = 1;
        while (1) {
            scanf("%d", &ticket[i][cur]);
            if (ticket[i][cur] == -1) {
                break;
            }
            ++cur;
        }
    }
```

```
    fprintf(stderr, "读入结束\n");
    int cnt = n + 1, final = 0;
    int nowCur[MAXM] = {0}, err_changed[MAXM] = {0};
    for (int i = 1; i <= m; ++i) {
        nowCur[i] = 1;//目前在看第i张票的第nowCur[i]名
    }
    int err_cntLun = 0;
    for (int i = 1; i <= m; ++i) {
        ++vote[ticket[i][nowCur[i]]];
    }
    fprintf(stderr, "    初始计票:\n");
    for (int i = 1; i <= n + 1; ++i) {
        fprintf(stderr, "    作品%d得票%d\n", i, vote[i]);
    }
    while (cnt > 2) { //Instant Round Vote
        ++err_cntLun;
        fprintf(stderr, "    Instant Round Vote 第%d轮:\n", err_cntLun);
        if (err_cntLun > 2 * n) {
            fprintf(stderr, "    ERROR!DEAD LOOP!\n");
            return 0;
        }

        for (int i = 1; i <= n + 1; ++i) {
            if (vote[i] * 2 > m) { //得票超过50%
                final = i;
                fprintf(stderr, "    Instant Round Vote胜利者:%d,得票%d\n", final,
vote[final]);
                int ans = NoAwardTest(final);
                printPrize(ans);
                return 0;
            }
        }

        int minVote = MAXM;
        fprintf(stderr, "    minVote=%d\n", minVote);
        for (int i = 1; i <= n + 1; ++i) {
            if (vote[i] < minVote && out[i] == 0) {
                minVote = vote[i];
            }
        }
        fprintf(stderr, "    minVote=%d\n", minVote);
        for (int i = 1; i <= n + 1; ++i) {
            if (vote[i] == minVote && out[i] == 0) {//淘汰
                fprintf(stderr, "    minVote=%d,Instant Round Vote 淘汰:%d\n",
minVote, i);
                out[i] = 1;
                vote[i] = 0;
                --cnt;
                for (int j = 1; j <= m; ++j) {
                    if (ticket[j][nowCur[j]] == i) {//标记这张选票需要被分票
                        err_changed[j] = 1;
                        ++nowCur[j];
                    }
                }
            }
        }
```

```
        memset(vote, 0, sizeof(vote));
        for (int i = 1; i <= m; ++i) {
            int nowVote = ticket[i][nowCur[i]];
            if (nowVote > 0 && out[nowVote] == 0) {
                ++vote[nowVote];
                if (err_changed[i]) {
                    fprintf(stderr, "    重新分配：第 %d 张票第 %d 名是 %d\n", i,
nowCur[i], nowVote);
                }
            } else {
                if (err_changed[i]) {
                    fprintf(stderr, "    分配失败：第 %d 张票第 %d 名是 %d\n", i,
nowCur[i], nowVote);
                }
            }
        }

        memset(err_changed, 0, sizeof(err_changed));
        fprintf(stderr, "    重配计票:\n");
        for (int i = 1; i <= n + 1; ++i) {
            fprintf(stderr, "    作品%d得票%d\n", i, vote[i]);
        }

        if (cnt == 1) {
            for (int i = 1; i <= n + 1; ++i) {
                if (out[i] == 0) {
                    int ans = NoAwardTest(i);
                    printPrize(ans);
                    return 0;
                }
            }
        }
    }
    int a = -1, b = -1;
    for (int i = 1; i <= n + 1; ++i) {
        if (out[i] == 0) {
            if (a == -1) {
                a = i;
            } else  {
                b = i;
            }
        }
    }
    int ans;
    if (vote[a] * 2 > m) {
        ans = NoAwardTest(a);
        printPrize(ans);
        return 0;
    } else if (vote[b] * 2 > m) {
        ans = NoAwardTest(b);
        printPrize(ans);
        return 0;
    } else {
        ans = RunOffTest(a, b);
    }
```

```
    if (ans == 0) {//两次平局
        int ans1 = NoAwardTest(a);
        int ans2 = NoAwardTest(b);
        if (ans1 == n + 1 && ans2 == n + 1) {
            printf("No Award!\n");
        } else if (ans1 == n + 1) {
            printf("%d\n", b);
        } else if (ans2 == n + 1) {
            printf("%d\n", a);
        } else {
            printf("%d %d\n", a, b);
        }
        return 0;
    }

    ans = NoAwardTest(ans);
    printPrize(ans);
    return 0;
}

int RunOffTest(int a, int b) {
    int flagequ = 0;
    if (vote[b] > vote[a]) {
        int temp = b;
        b = a;
        a = temp;
    } else if (vote[b] == vote[a]) {
        flagequ = 1;
    }
    fprintf(stderr, "    RunOfftest(%d,%d)\n", a, b);
    int x = 0, y = 0;
    for (int i = 1; i <= m; ++i) {
        int cura = MAXN, curb = MAXN;
        for (int j = 1; ticket[i][j] != -1; ++j) {
            if (ticket[i][j] == a) {
                cura = j;
            }
            if (ticket[i][j] == b) {
                curb = j;
            }
        }

        if (cura < curb) {
            ++x;
        } else if (curb < cura) {
            ++y;
        } else {
            ;
        }
    }
    fprintf(stderr, "    x=%d,y=%d\n", x, y);
    if (x == y && flagequ == 1) {
        return 0;
    } else {
        return (x >= y) ? a : b;
    }
```

```
}

int NoAwardTest(int x) {
    fprintf(stderr, "    NoAwardTest(%d)\n", x);
    if (x == n + 1) {
        return x;
    }
    vote[n + 1] = -1;
    return RunOffTest(x, n + 1);
}

void printPrize(int x) {
    if (x == n + 1) {
        printf("No Award!\n");
    } else {
        printf("%d\n", x);
    }
    return;
}
```

S 坐标去重输出23

难度	考点
2~3	qsort

题目分析

这道题第一步读取输入，将输入存入数组中。第二步对数组升序排序。第三步输出数组，输出的同时注意重复项不要输出。

在数组排序中，大家可能会遇到以下两个问题：

- 使用冒泡排序，结果`TLE`

  当前大家已经学过冒泡排序和 `qsort` 。冒泡排序的时间复杂度为 $O(n^2)$，而 `qsort` 的时间复杂度为 $O(nlog(n))$ ，显然后者的排序速度更快。因此大家在常规的排序中优先使用 `qsort` 。

- 使用了 `qsort` ，但`cmp` 函数是错误的。

  部分 `WA` 的同学使用下面这种 `cmp` 函数

  ```
  int cmp(const void* p1, const void* p2)
  {
      return *(int *)p1-*(int *)p2;
  }
  ```

  这个函数看似是可行的，但在某些情况下，这种实现可能会导致错误的结果。

  当两个整数的差值超出了整型变量的范围，就会发生溢出，这可能会导致比较结果不正确。此外，在某些平台上，指针的大小可能会超过`int`类型的大小，这也会导致错误。

  因此在编写`cmp` 函数时，一定不要两个数直接相减，而是采用下面这种比较的方式

```
int cmp(const void* p1, const void* p2) {  //升序排列
    int* a = (int*)p1, * b = (int*)p2;
    if (*a < *b)return -1;
    else if (*a > *b) return 1;
    else return 0;
}
```

示例代码

```
#include <stdio.h>
#include <stdlib.h>
int cmp(const void* p1, const void* p2) {  //升序排列
    int* a = (int*)p1, * b = (int*)p2;
    if (*a < *b)return -1;
    else if (*a > *b) return 1;
    else return 0;
}
int a[3000000];
int main()
{
    int n;
    scanf("%d",&n);
    for (int i = 0; i < n; ++i) {
        scanf("%d",&a[i]);

    }
    qsort(a,n,sizeof (int),cmp);
    int old=a[0];
    printf("%d ",a[0]);
    for (int i = 1; i < n; ++i) {
        if (old==a[i]) continue; //如果当前数和上一个数相同，就不输出
        printf("%d ",a[i]);
        old=a[i];
    }
    return 0;
}
```

T 后缀排序

难度	考点
4	qsort，指针数组，字符串

题意分析

字符串以 '\0' 作为结尾， $str[i]$ 的地址就代表了 str 去掉前 i 个字符后的剩下的后缀。可以建立存储 `char *` 类型的指针数组 p， $p[i]$ 指向 $str[i]$ 的地址，然后对数组 p 用 `qsort` 排序即可。注意cmp函数的写法。

也可以用 `int` 类型的数组代替数组 p，比较时通过下标访问地址，具体实现见示例代码。

示例代码-1：指针数组

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
char s[10005];
char *p[10005];
int cmp(const void *e1, const void *e2)
{
    return strcmp(*(char**)e1, *(char**)e2);
}
int main()
{
    gets(s);
    int len = strlen(s);
    for(int i = 0; i < len; ++i)
        p[i] = s + i;
    qsort(p, len, sizeof(char *), cmp);
    int t;
    scanf("%d", &t);
    puts(p[--t]); //由于p是从0开始计数，第t小的应该是p[t-1]
    return 0;
}
```

示例代码-2：下标数组

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
char s[10006];
int x[10006];
int cmp(const void *aa, const void *bb)
{
    int a = *(int*)aa, b = *(int*)bb;
    return strcmp(s + a, s + b);
}
int main()
{
    int n, len;
    scanf("%s", s);
    scanf("%d", &n);
    len = strlen(s);
    for (int i = 0; i < len; i++)
        x[i] = i;
    qsort(x, len, sizeof(int), cmp);
    printf("%s", s + x[n - 1]);
    return 0;
}
```

Author: 哪吒

U OJ排行榜

难度	考点
4	qsort，指针数组，字符串

题目分析

排序规则很简单，难点在于每个参赛者信息有字符串、浮点数和整数三种类型，很难用二维数组存储然后对二维数组排序。那我们可以用一个数组记录每个参赛者的下标，相当于人为给他们进行了编号，排序时不排序参赛者本身，而是对他们的编号排序，最后按照排序过后的编号顺序输出。

示例代码

```
#include <stdio.h>
#include <stdlib.h>
#define eps 1e-6

char name[100005][16];
double score[100005];
int time[100005];
int id[100005];

int cmp(const void *p, const void *q)
{
    int a = *(int*)p;
    int b = *(int*)q;
    if(score[a] - score[b] > eps) return -1;
    else if(score[a] - score[b] < -eps) return 1;
    else if(time[a] > time[b]) return 1;
    else if(time[a] < time[b]) return -1;
    else return 0;
}

int main()
{
    int n = 0;
    while(~scanf("%s%lf%d", name[n], &score[n], &time[n]))
    {
        id[n] = n;
        ++n;
    }
    qsort(id, n, sizeof(int), cmp);
    for(int i = 0; i < n; ++i)
        printf("%10s %8.2f %10d\n", name[id[i]], score[id[i]], time[id[i]]);
    return 0;
}
```

Author: 哪吒

V 区间合并 2.0

难度	考点
5	`qsort` 排序二维数组

题目分析

本题与E6H-“哪吒的区间合并（1）”的主要区别有两处，第一是数据范围较大，无法利用数组下标记录区间；第二是本题考虑了区间端点的开闭性，需要一些细节上的变化和处理。基本思路是将区间按照左端点递增进行排序，然后遍历数组根据右端点情况进行合并。具体实现思路如下：

利用二维数组或结构体记录区间，二维数组的每行共四个元素，分别记录左端点大小、右端点大小、左端点开闭性、右端点开闭性，其中开闭性可以用 1 表示闭， 0 表示开。

全部读入后，先对所有区间进行排序，排序规则：按区间左端点大小升序排序，当左端点相同时，闭区间在前，开区间在后。

然后遍历每一个区间，用一个临时一维数组或结构体 t 记录当前合并的结果，初始化 t 为当前遍历到的区间，然后继续遍历开始合并区间。只有当前区间左端点小于 t 右端点，或当前区间左端点等于 t 右端点且不全为开区间时，才能合并。

在合并时，如果当前区间右端点大于 t ，更新 t 的右端点及其开闭性为当前区间的右端点及其开闭性；如果当前区间右端点等于 t ，更新 t 的右端点的开闭性，当前区间右端点开闭性和 t 中有一者为闭则更新 t 右端点开闭性为闭，否则为开；如果当前区间右端点小于 t ，则不应改变 t 。

无法再进行合并时，输出 t 记录的区间合并结果，然后继续遍历区间，直至结束。

示例代码 1 - 二维数组

```c
#include <stdio.h>
#include <stdlib.h>
int cmp(const void *e1, const void *e2) //按区间左端点大小升序排序，当左端点相同时，闭区间在前，开区间在后。
{
    int *p = (int *)e1;
    int *q = (int *)e2;
    if(p[0] > q[0]) return 1;
    else if(p[0] < q[0]) return -1;
    else if(p[2] < q[2]) return 1;
    else if(p[2] > q[2]) return -1;
    else return 0;
}
/*
二维数组a记录输入区间
每行：
0-左端点大小
1-右端点大小
2-左端点开闭性，1表示闭，0表示开
3-右端点开闭性，1表示闭，0表示开
*/
int a[500005][4];
int main()
{
    int n = 0; //记录输入的区间数量
```

```
    char l, r;
    while(~scanf("%c%d,%d%c ", &l, &a[n][0], &a[n][1], &r))
    {
        a[n][2] = l == '[' ? 1 : 0;
        a[n][3] = r == ']' ? 1 : 0;
        ++n;
    }
    qsort(a, n, sizeof(a[0]), cmp); //按区间左端点大小升序排序，当左端点相同时，闭区间在前，开区间在后。
    for(int i = 0; i < n;)
    {
        int t[4] = {a[i][0], a[i][1], a[i][2], a[i][3]}; //记录本次合并结果，初始化t为本次合并的最初区间
        //只有区间a[i]左端点小于t右端点，或区间a[i]左端点等于t右端点且不全为开区间，才能合并
        for(++i; i < n && (a[i][0] < t[1] || (a[i][0] == t[1] && (a[i][2] | t[3]))); ++i) //至少一个闭时a[i][2]|t[3]为1，若均为开则为0
        {
            if(a[i][1] > t[1]) //a[i]右端点大于t，更新t右端点及其开闭性为a[i]右端点及其开闭性
                t[1] = a[i][1], t[3] = a[i][3];
            else if(a[i][1] == t[1]) //a[i]右端点等于t，更新t右端点的开闭性（a[i]和t中有一者为闭则更新t右端点开闭性为闭，否则为开）
                t[3] |= a[i][3];
            //若a[i][1]<t[1]，则不需要改变t
        }
        printf("%c%d,%d%c\n", t[2] ? '[' : '(', t[0], t[1], t[3] ? ']' : ')'); //输出本次合并结果
    }
    return 0;
}
```

示例代码 2 - 结构体数组

```
#include <stdio.h>
#include <stdlib.h>
typedef struct _Interval
{
    int left, right; //存储左右端点大小
    int loc, roc; //表示左/右端点开闭性，0-开，1-闭
} Interval, *pInterval;
int cmp(const void *e1, const void *e2) //按区间左端点大小升序排序，当左端点相同时，闭区间在前，开区间在后。
{
    pInterval p = (pInterval)e1, q = (pInterval)e2;
    if(p->left > q->left) return 1;
    if(p->left < q->left) return -1;
    if(p->loc < q->loc) return 1;
    if(p->loc > q->loc) return -1;
    return 0;
}
Interval a[500005];
int main()
{
    int n = 0; //记录输入的区间数量
    char l, r;
```

```
    while(~scanf(" %c%d,%d%c", &l, &a[n].left, &a[n].right, &r))
    {
        a[n].loc = l == '[' ? 1 : 0;
        a[n].roc = r == ']' ? 1 : 0;
        ++n;
    }
    qsort(a, n, sizeof(Interval), cmp); //按区间左端点大小升序排序，当左端点相同时，闭区间在前，开区间在后。
    for(int i = 0; i < n;)
    {
        Interval t = a[i]; //记录本次合并结果，初始化t为本次合并的最初区间
        //只有区间a[i]左端点小于t右端点，或区间a[i]左端点等于t右端点且不全为开区间时，才能合并
        for(++i; i < n && (a[i].left < t.right || (a[i].left == t.right && (a[i].loc | t.roc))); ++i) //至少一个闭时a[i].loc|t.roc为1，若均为开则为0
        {
            if(a[i].right > t.right) //a[i]右端点大于t，更新t右端点及其开闭性为a[i]右端点及其开闭性
                t.right = a[i].right, t.roc = a[i].roc;
            else if(a[i].right == t.right) //a[i]右端点等于t，更新t右端点的开闭性（a[i]和t中有一者为闭则更新t右端点开闭性为闭，否则为开）
                t.roc |= a[i].roc;
            //若a[i]右端点小于t，则不需要改变t
        }
        printf("%c%d,%d%c\n", t.loc ? '[' : '(', t.left, t.right, t.roc ? ']' : ')'); //输出本次合并结果
    }
    return 0;
}
```

补充

示例代码中用了一些三目运算符和位运算简化了代码，同学们也可以全用 if 和 $else$ 代替。

感兴趣的同学可以去理解一下为什么二维数组排序的 cmp 函数要这么写。

Author: 哪吒

W 超市促销

难度	考点
4	递归

题目分析

这道题的一种解决方法是递归。

通过题意，我们可以得出长度为 L 的活动区域的隔开的顺序数
$m[L]=m[L-a]+m[L-b]+m[L-c]$ 。

递归的基本条件是当 $L=0$ 时,隔开的顺序数量为 1 ，即 $m[0]=1$ ，当 $L<0$ ，,隔开的顺序数量为 0 。

要注意递归过程中要存储已经计算过的顺序数，避免重复计算造成超时。

示例代码 1 - 递归方法

```
#include <stdio.h>
long long m[401];
long long count(int a, int b, int c, int L) {
    // 如果活动区域的总长小于0，说明没有合法的隔开顺序，返回0
    if (L < 0) return 0;
    // 如果活动区域的总长等于0，说明刚好隔开到最后一个区域，返回1
    if (L == 0) return 1;
    // 如果m[L]已经计算过，直接返回，避免重复计算。
    if (m[L] != -1) return m[L];
    // 递归本体
    m[L] = count(a, b, c, L - a) + count(a, b, c, L - b) + count(a, b, c, L - c);
    return m[L];
}
int main()
{
    int a,b,c,l;
    while (scanf("%d%d%d%d", &a, &b, &c, &l) != EOF)
    {
        for (int i = 0; i <= l; ++i)
            m[i]=-1; //初始化计算过的长度的顺序个数为-1，表示均未计算过
        printf("%lld\n", count(a, b, c, l));
    }
    return 0;
}
```

示例代码 2 - 循环递推方法

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a, b, c, L;
    while(~scanf("%d%d%d%d", &a, &b, &c, &L))
    {
        long long f[410] = {1};
        for(int i = 5; i <= L; i += 5)
        {
            if(i >= a) f[i] += f[i - a];
            if(i >= b) f[i] += f[i - b];
            if(i >= c) f[i] += f[i - c];
        }
        printf("%lld\n", f[L]);
    }
    return 0;
}
```

X 哪吒找金币

难度	考点
4	递归搜索

题意分析

利用递归函数对地图进行搜索。递归函数 $f(i,j)$ 计算当前地图中 a_{ij} 所处的一堆金币的总数。对于每个点，若该点没有金币，直接返回0；否则，**将该点置零**，并对该点周围四个方向相邻位置递归调用函数 f ，返回五个值（该点本身和四个递归函数的返回值）的结果。

主函数中遍历整个地图，每次调用 f 即得到一堆金币的总数，取最大值即可。

这里下标从 1 开始，利用全局变量初始值为0，保证超过地图的时候会到达基本情况，递归终止，不会越界。

示例代码

```
#include <stdio.h>
int a[105][105], ans;
int f(int i, int j)
{
    if(!a[i][j]) return 0; //a[i][j]=0
    int t = a[i][j]; //临时变量存储a[i][j]
    a[i][j] = 0;
    return t + f(i + 1, j) + f(i - 1, j) + f(i, j - 1) + f(i, j + 1);
}
int main()
{
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; ++i)
        for(int j = 1; j <= n; ++j)
            scanf("%d", &a[i][j]);
    for(int i = 1; i <= n; ++i)
        for(int j = 1; j <= n; ++j)
            if(a[i][j])
            {
                int t = f(i, j); //注意f(i,j)调用一次后，由于将遍历过的点置零，再次调用会返回0，因此要用临时变量存储结果。
                if(t > ans) ans = t;
            }
    printf("%d", ans);
    return 0;
}
```

Author: 哪吒

Y 幂次展开

难度	考点
5	递归输出、二进制

题目分析

由题意可知，0 的 2 的幂次方表示为 0，而对于其他的正数 x，若 x 的二进制表示为 $000...0101$，则其表示为 2^2+2^0，显然两者存在一一对应关系。

当指数不为 0 时，需要将指数部分也表示为 2 的幂次方的形式，以此类推，直至指数为 0，所以该嵌套结构可以用递归来实现。

写递归函数重要的是找到递归出口，也就是什么时候停止递归。在本题中，显然当要表示的数为 0 时可以停止递归，直接输出 0 即可。

而对正数来说，遍历其二进制的每一位，如果为 1，则先输出底数 2 以及左括号，再对指数部分进行转化（调用递归函数），最后输出右括号。注意每一项之间还要输出加号。

示例代码

```c
#include <stdio.h>
void f(int x) {
    if(x == 0) {
        putchar('0');
        return ;
    }
    int first = 1;
    for(int i = 30; i >= 0; i --)
        if((x >> i) & 1) {
            if(first) first = 0; // 第一个式子不输出 +
            else putchar('+');
            printf("2(");
            f(i);
            printf(")");
        }
}
int main() {
    int n;
    scanf("%d", &n);
    f(n);
    return 0;
}
```

Z 朝田诗乃的桌游

难度	考点
5	深度优先搜索

题目分析

有些有基础的同学可能会认为这个问题是一个0/1背包，这是不对的，因为0/1背包问题中物品的放入顺序不影响结果，但是很明显此题会影响结果。

而如果使用一般的DFS解决此题，那么很容易看出搜索树上存在大量重复路径：搜索时会反复到达同一个“状态”，而到达“状态”后能得到的最大价值早已确定，不管搜索几次都不会改变。

（这里定义“状态”为所在格数 k ，四种牌的剩余数量 v_1, v_2, v_3, v_4 ，状态相同当且仅当五个值都相同）

既然如此，结合数据范围，不难想到使用一个数组 $f[k][v_1][v_2][v_3][v_4]$ 来存储已经搜索过的结果（该状态后续能提供的最大价值），再次访问该状态后直接返回该结果

于是一个清晰的整体思路就有了：从头开始递归，每到一个状态先判断该状态是否已被搜索过，若已被搜索过则直接返回；没被搜索过则继续进行搜索，并在返回前给 f 数组附上相应的值。

由于此题保证使用所有牌后正好到达 n ，所以 f 可以去掉 v_4 这一维度，改为通过 k、v_1、v_2、v_3、n 的关系推断 v_4 的值。

示例代码

```
#include<stdio.h>
int n,m,a[351],num[15],f[351][41][41][41];
int Max(int a,int b){
    if(a>=b) return a;
    else return b;
}
int dfs(int k,int v1,int v2,int v3,int v4){
    if(f[k][v1][v2][v3]) return f[k][v1][v2][v3];
    if(k==n) return a[n];
    if(v1>=1) f[k][v1][v2][v3]=Max(f[k][v1][v2][v3],dfs(k+1,v1-1,v2,v3,v4));
    if(v2>=1) f[k][v1][v2][v3]=Max(f[k][v1][v2][v3],dfs(k+2,v1,v2-1,v3,v4));
    if(v3>=1) f[k][v1][v2][v3]=Max(f[k][v1][v2][v3],dfs(k+3,v1,v2,v3-1,v4));
    if(v4>=1) f[k][v1][v2][v3]=Max(f[k][v1][v2][v3],dfs(k+4,v1,v2,v3,v4-1));
    f[k][v1][v2][v3]+=a[k];
    return f[k][v1][v2][v3];
}
int main()
{
    scanf("%d %d",&n,&m);
    for(int i=1; i<=n; i++) scanf("%d",&a[i]);
    for(int i=1; i<=m; i++){
        int k;
        scanf("%d",&k);
        num[k]++;
    }
    dfs(1,num[1],num[2],num[3],num[4]);
    printf("%d\n",f[1][num[1]][num[2]][num[3]]);
    return 0;
}
```

BA 顺序统计量 2023

难度	考点
5	分治 递归

题目分析

其实实现本题所有的思路在题目描述中已经尽数给出了，我们要做的就是把两段伪代码转化成 C 语言代码而已，这里不再赘述了。

我们已经提到过，题目中的实现只是**期望时间**为线性，对于一些特殊的输入最坏情况下算法的复杂度为平方级。我们在交换过程中每次都选取最后一个元素作为基准，所以可以使用一些构造样例的手段让我们的复杂度达到最坏。其实正确的做法是每次在要分割的数组中随机地选取一个元素作为用来分割的元素，也就是把它变成一个随机算法。

其次，题目中保证了所有元素互不重复，可以思考一下当可能有很多元素重复时会发生什么，我们的算法效率会比较低下，因为 `SELECT` 每次可能只移动一位。更好的做法是分割时把区间分成三段，小于一段，等于一段和大于一段；比较的时候也用区间判断。但是因为上述的两种更好的做法实现起来太过复杂，在这里我们只把它们放在题解中，仅作为一些扩展。

最后要注意的一点是交换时不要使用异或三次的做法，即 `i ^= j;j ^= i;i ^= j;` 这样在两个数的地址相同时就会出问题（具体可以参考 debug 大赛 G 题）。

示例代码

```c
#include <stdio.h>
int A[5000005];

void swap(int i,int j)
{
    int tem = A[i];
    A[i] = A[j];
    A[j] = tem;
}

int partition(int *A, int p, int r)
{
    int x = A[r];
    int i = p - 1;
    for (int j = p; j <= r - 1; j++)
        if (A[j] <= x)
        {
            i = i + 1;
            swap(i , j);
        }
    swap(i + 1, r);
    return i + 1;
}

int select(int *A, int p, int r, int i)
{
    if (p == r)
        return A[p];
```

```
    int q = partition(A, p, r);
    int k = q - p + 1;
    if (i == k)
        return A[q];
    else if (i < k)
        return select(A, p, q - 1, i);
    else
        return select(A, q + 1, r, i - k);
}

int main()
{
    int n, k;
    scanf("%d%d", &n, &k);
    for (int i = 1; i <= n; i++)
        scanf("%d", &A[i]);
    printf("%d\n", select(A, 1, n, k));
    return 0;
}
```

BB 宝藏套装

难度	考点
2~3	位运算

题目分析

位运算的精髓是什么?

1. 一个精髓是要懂得 **按位** 运算，也就是说不需要看这个数整体，而是直接对每一位进行处理。
2. 另一个精髓是要掌握一些运算规律，如与、或、异或都满足 **结合律** 和 **交换律**，

- 与运算 和 或运算 还满足 $a \,\&\, a = a,\ a \mid a = a$
- 异或运算还满足 $a \wedge a = 0$, 若 $a \wedge b = c$ 则有 $a \wedge c = b$ 和 $b \wedge c = a$

那么再来分析一下这道题，不难发现两个数 A 和 B 的任何一位（记作 A_i 和 B_i）的 “总价值” 都不如二者中最大值的二倍，可以列出下表：

A_i	B_i	$\& + \wedge$	$2 * \max\{A_i,\ B_i\}$
0	0	00	00
0	1	01	10
1	0	01	10
1	1	10	10

当且仅当 $A_i = B_i$ 时，二者相等。那么也就是说，除非 A 与 B 每一位都是 0，否则二者的 “总价值” 不如二者中最大值的二倍。这就实现了从每一位到整体的推广，于是这道题就迎刃而解了，只需要找出其中最大的一个数并乘 2 输出，甚至都不需要真的在程序中进行位运算。

示例代码

```
#include <stdio.h>

int main()
{
    int n,m,a,ans=0;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
    {
        scanf("%d",&a);
        if(a>ans) ans=a;
    }
    printf("%d",2*ans);
    return 0;
}
```

BC 一箭多雕

难度	考点
3~4	杂题，数学

题目分析

首先，我们易知，对于穿过相邻两层的直线，我们尽可能使其不要穿过格点，那么最大值一定是 $m+n-1$

~~(通过观察和YY，易知答案一定为 $m+n-1$)~~

通过数学归纳法

首先，我们知道 $m=1,n=1$的时候答案为 $m+n-1=1$

那么我们假设 $m=m_0,n=n_0$时答案成立，那么对于 $m'=m_0+1$ 或者 $n'=n_0+1$ 时，相当于在原方阵中加入一行或一列，易知，直线会多穿过一个格子

故可得，最终答案为 $m+n-1$

那么对于op2，其实就是在op1的基础上推导了，~~(此处略)~~

示例代码

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define ll long long
#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))
#define rep(i, a, b) for(int i = a; i <= b; i++)
#define per(i, a, b) for(int i = a; i >= b; i--)
```

```
#define mod 998244353

ll test, opt, u, v, w;

int main() {
    scanf("%lld", &test);
    while(test--) {
        scanf("%lld", &opt);
        if(opt == 1) {
            scanf("%lld %lld", &u, &v);
            printf("%lld\n", u + v - 1);
        } else {
            scanf("%lld %lld %lld", &u, &v, &w);
            ll x = min(u, v);
            ll y = w + 1 - x;
            x %= mod, y %= mod;
            u %= mod, v %= mod;
            printf("%lld\n", (((x * y) % mod - (u * v) % mod) % mod + mod) %
mod);
        }
    }
    // system("pause");
    return 0;
}
```

BD RA：操控

难度	考点
4	构造，倒推

题目分析

易知，从右至左，如果 `Z` 出现在了 `X or Y` 左边，那么一定不能构造出结果

另外，由于字符串的字典序通过第一个字符的比较可以直接确定，故思路就非常清晰了：

从右至左，对于 `Z` 我们输入两个一样的字符（如 `A`）

对于 `X` 我们输出 `B and A`；对于 `Y` 我们输出 `A and B` 即可

示例代码

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define ll long long
#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))
#define rep(i, a, b) for(int i = a; i <= b; i++)
#define per(i, a, b) for(int i = a; i >= b; i--)
```

```
#define maxn 1000005

char s[maxn], x[maxn], y[maxn];
int len, jud;

int main() {
    scanf("%s", s);
    len = strlen(s);
    for(int i = len - 1; i >= 0; i--) {
        if(s[i] == 'X') {
            x[i] = 'B', y[i] = 'A';
            jud = 1;
        } else if(s[i] == 'Y') {
            x[i] = 'A', y[i] = 'B';
            jud = 1;
        } else {
            x[i] = 'A', y[i] = 'A';
            if(jud) {
                printf("-1\n");
                return 0;
            }
        }
    }
    for(int i = 0; i < len; i++) printf("%c", x[i]);
    printf("\n");
    for(int i = 0; i < len; i++) printf("%c", y[i]);
    return 0;
}
```

BE 收集石樱

难度	考点
4	贪心

问题分析

为了求得收集石樱的最大值，可把石樱的来源分为两类：原有的和新生长的。

对于新生长的石樱，简单分析题意可知，由于节点在每个单位时间内都会长出一个石樱，为了收集到更多的新生石樱，应当不重复地尽量走完每一个没有收集过的节点，如果走回头路而导致最终没有将所有节点都遍历一遍，显然会造成浪费。因为往回走只会收集到 1 个新生石樱，而往前走收集到的新生石樱数量显然大于等于 1。因此有一个基本的贪心策略：**在没有一次走完 n 个节点的时候，不要走回头路。**对 m, n 的关系分类讨论如下：

1. 当 $m <= n$ 时，此时无法遍历所有的节点或刚好遍历完一次，则选择 m 个节点收集，那么所收集的新生石樱的数量为 $\frac{(m-1)m}{2}$，而所收集的原有石樱数量为连续 m 个节点的总和最大值。对于长为 m 的固定区间最大值求解：考虑遍历每一个节点为区间起始点，再遍历该区间内所有石樱数量，相加后更新最大值，时间复杂度为 $O(mn)$，注意到 $1 \leq n \leq 10^5$，无法通过本题；注意到每个区间都可以由上个区间平移一位得到，记 sum 为上一个区间和，考虑用如下方法计算每次区间的最大值：

```
    for (i = m; i < n; i++)
    {
        sum += a[i] - a[i - m];
        max = MAX(sum, max);
    }
```

2. 当 $m > n$ 时，此时可以收集完所有原有石樱。对于新生石樱的数量，由于遍历完一次后会有回头路，因此不能如 1 中情况计算。由于路径不同，不易求解收集到的新生石樱数量。考虑从节点 1 走到节点 n，**未收集**到的新生石樱数量分别为 $n, n-1, \ldots, 1$，即 $\frac{n(n+1)}{2}$。可以发现，如果每次走过 n 个节点，该值不会随起点和终点的位置而变化；进一步采用我们的贪心策略，该值在任何时候都不会发生变化。由于 m 个单位时间内新生石樱数量总和为 mn，可得收集的数量为 $mn - \frac{n(n+1)}{2}$。

注意开 `long long`。

由于两种情况代码有重复内容，重构后答案如下：

参考代码

```
#include <stdio.h>
#define MAX(a,b) (((a)<(b))?(b):(a))
#define LL long long

LL a[100005];

int main()
{
    LL n, m, T, i, sum, max;
    scanf("%lld", &T);
    while (T--)
    {
        sum = max = 0;
        scanf("%lld%lld", &n, &m);
        for (i = 0; i < n; i++)
        {
            scanf("%lld", &a[i]);
            if (i < m)
                sum += a[i];
            else
            {
                max = MAX(sum, max);
                sum += a[i] - a[i - m];
            }
        }
        max = MAX(sum, max);
        if (m > n)
            printf("%lld\n", sum + n * m - n * (n + 1) / 2);
        else
            printf("%lld\n", max + (m - 1)* m / 2);
    }
    return 0;
}
```

BF 保护大结界

难度	考点
4~5	快排

问题分析

考虑下述函数：

$$\mathrm{f}(x)=|x-a|+|x-b| \quad (a\le b)$$

其函数值有如下情况：

$$\mathrm{f}(x)=\begin{cases} a+b-2x\,, & x<a \\ b-a\,, & a\le x<b \\ 2x-a-b\,, & x\ge b \end{cases}$$

易知当 $x<a$ 或 $x\ge b$ 时，$\mathrm{f}(x)\ge b-a$，即当 $a\le x\le b$ 时取得最小值。（提示：可以在数轴上画图来快速得到结果）

事实上，由上面可以得到，对于函数：

$$\mathrm{F}(x)=\sum_{i=1}^{n}|x-a_i| \quad (a_i\le a_{i+1})$$

当 $n=2k-1(k\in\mathbb{N}^*)$ 时，$\mathrm{F}(x)$ 在 $x=a_{\frac{n+1}{2}}$ 处取得最小值；

当 $n=2k(k\in\mathbb{N}^*)$ 时，$\mathrm{F}(x)$ 在 $a_{\frac{n}{2}}\le x\le a_{\frac{n}{2}+1}$ 时取得最小值。

由于证明类似，此处略去。

由上述证明可知，当 x 取 $\{a_n\}$ 的中位数时，$\mathrm{F}(x)$ 取得最小值。

注意到 x,y,z 相互独立，因此可以分别排序求解。

参考代码

```c
#include <stdio.h>
#include <stdlib.h>

int mycomp(const void *p1, const void *p2)
{
    const int *a1 = (const int *)p1;
    const int *a2 = (const int *)p2;
    if (*a1 < *a2)
        return -1;
    else if (*a1 == *a2)
        return 0;
    else
        return 1;
}

int main()
{
```

```
    int n, d[3][100005];
    long long sum = 0;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < 3; j++)
            scanf("%d", &d[j][i]);
    for (int i = 0; i < 3; i++)
    {
        qsort(d[i], n, sizeof(int), mycomp);
        for (int j = 0; j < n / 2; j++)
            sum += d[i][n - j - 1] - d[i][j];
    }
    printf("%lld", sum);
    return 0;
}
```

BG 破损的三角铁

难度	考点
4~5	暴力，组合

简要题意

求从 n 根铁棒中选 4 根组成正三角形的方案数。

问题分析

用 4 根铁棒组成正三角形，则必有 2 根长度相等，且等于另外 2 根长度之和。

注意到铁棒长度 $a_i \leq 5000$，时间复杂度 $O(n^2)$ 可以通过。直接枚举上述两种铁棒的长度即可。

记 num_i 为长度为 i 的铁棒的个数。

1. 选取长度相等的 2 根木棒，方法数为 $C_{num_i}^2$。
2. 从剩下的木棒中取出 2 根长度之和为 i 的铁棒，令其中一根长度为 j，则另一根长度为 $i-j$。若 $j=i-j$，则方法数为 $C_{num_i}^2$；若 $j \neq i-j$，则方法数为 $C_{num_j}^1 \times C_{num_{i-j}}^1$。

由乘法原理将方案数相乘，再将所有情况相加即可得到答案。

注意随时取模。

参考代码

```
#include <stdio.h>
#define LL long long

const LL N = 1e9 + 7;

int main()
{
    int n, i, j, x;
    LL a[5005] = {0}, ans = 0, ans1;
```

```
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &x);
        a[x]++; // 记录每种长度铁棒的数量
    }
    for (i = 2; i <= 5000; i++)
    {
        if (a[i] >= 2)
        {
            ans1 = a[i] * (a[i] - 1) / 2 % N; // 取两根长度相等的铁棒的方法数
            for (j = 1; j <= i / 2; j++)
            {
                if (2 * j != i)
                    ans += ans1 * a[j] % N * a[i - j] % N;
                else if (2 * j == i)
                    ans += ans1 * a[j] % N * (a[j] - 1) / 2 % N;
                ans %= N;
            }
        }
    }
    printf("%lld", ans);
    return 0;
}
```

BH 贤者之石

难度	考点
5	前缀和

问题分析

由题意，我们需要找到一个连续子串，使得其中 `R` 和 `B` 的数量相同。

考虑枚举每一个字符，再枚举以该字符为头的字符串，时间复杂度为 $O(n^2)$。注意到数据范围 $|\mathtt{s}| \leq 2 \times 10^5$，无法通过本题。

记 $\mathtt{s}_{ij}$ 为 s 从第 i 个字符到第 j 个字符的连续子串。为了简化状态的表示，我们用 1 和 -1 来表示 `R` 和 `B`，并记第 i 个字符的数值为 a_i。当子串 $\mathtt{s}_{ij}$ 满足题意时，有：

$$\sum_{k=i}^{j} a_k = 0$$

记 S_n 为 $\{a_n\}$ 的前 n 项和，则有：

$$S_j - S_i = 0$$

因此我们可以计算出 $\{S_n\}$，求该数列中相同项的项数之差最大值。考虑用 pos_k 表示 S_n 的值为 k 出现的最早位置。由于数组下标为非负数，我们将 S_n 整体加上 $|\mathtt{s}|_{max}$，即 2×10^5。因为一次遍历即可同时更新 S_k 和 pos_k 的值，因此不需要为 $\{S_n\}$ 申请一个数组，用 sum 表示即可。

参考代码

```
#include <stdio.h>
#include <string.h>
#define N 200000
#define MAX(a, b) ((a) < (b) ? (b) : (a))

int sum = N, i, pos[2 * N + 5], ans;
char s[N + 5];

int main()
{
    scanf("%s", s);
    for (i = 1; s[i - 1]; i++)
    {
        sum += s[i - 1] == 'R' ? 1 : -1;
        if (sum != N && !pos[sum]) // 为什么?
            pos[sum] = i;
        else
            ans = MAX(ans, i - pos[sum]);
    }
    printf("%d", ans);
    return 0;
}
```

当 $pos_k = 0$ 时，表示数值 k 还未出现过，因此可以更新。但有一个例外，不能更新 pos_N 的值，因为 pos_N 显然一定为 0，而 $sum = N$ 表示此时的 $\mathtt{s}_{1i}$ 是满足题意的子串。

BI violet的跳跃子串

难度	考点
5	动态规划，递推

题目分析

此题的所定义的“跳跃子串”其实有他的学名——子序列，而本题所要求的其实就是一道经典例题：求子序列的个数。

首先我们建立一个长度为 6 的数组 a 和字符串数组 $sv[7] = \ "violet"$，$a[i]$ 代表字符串 s 中，以字母 $sv[i]$ 开头的字符串的子序列个数。例如，假设 $a[3] = 5$ 代表 s 串中以 $"let"$ 的子序列个数为 5。

由此，我们可以从后向前遍历字符串，不断更新 a 数组的值，最终遍历结束后，即求得正确答案。

更新方法如下：

- 若当前字符为 t，即所求子序列的最后一个字符，则 $a[5]++$
- 否则 $a[i] = a[i] + a[i+1]$

示例代码 - 1

```
#include <stdio.h>
#include <string.h>
char s[1000005];
int main()
{
    scanf("%s", s);
    int i, j, flag = 0, len = strlen(s);
    int a[6] = {0};
    char sv[] = "violet";
    for (i = len - 1; i >= 0; i--)
    {
        for (j = 0; j < 6; j++)
        {
            if (s[i] == sv[j])
            {
                if (j == 5)
                    a[5]++;
                else
                    a[j] += a[j + 1];
                break;
            }
        }
    }
    printf("%d", a[0]);
    return 0;
}
```

示例代码 - 2

```
#include <stdio.h>
char s[1000001], *name = "violet";
int main()
{
    gets(s);
    int dp[7] = {1};
    for(int i = 0; s[i]; ++i)
        for(int j = 6; j; --j)
            if(s[i] == name[j-1]) dp[j] = (dp[j] + dp[j - 1]);
    printf("%d", dp[6]);
    return 0;
}
```

BJ 阻止czx

难度	考点
4~5	队列

题目分析

每次找 `c`，`z`，`x` 时，如果暴力去顺序查找会超时。

本题的关键点在于，每次删除 `c`，`z`，`x` 时，删掉同一字符 C 的位置一定是递增的，即第 k 次删除 `c`，`z`，`x` 时删掉字符 C 的位置一定大于第 $k-1$ 次删除 `c`，`z`，`x` 时删掉字符 C 的位置，这里 C 可以是 `c`，`z`，`x` 中的任意一个。

因此，我们可以先遍历一遍字符串，将字符串中所有的 `c`，`z`，`x` 的位置预处理出来。

每次删除 `c`，`z`，`x` 时，删掉 C 时从上一次找到 C 的位置开始向后遍历即可。

这样每个字符最多只会被遍历一次，时间复杂度为 $O(n)$ 。

示例代码

```
#include <stdio.h>
char s[400001];
int x1[400001], x2[400001], x3[400001];
int cnt1, cnt2, cnt3;
int i, j, k;
int main()
{
    gets(s);
    for(int t = 0; s[t]; ++t)
    {
        if(s[t] == 'c' || s[t] == 'C') x1[cnt1++] = t;
        if(s[t] == 'z' || s[t] == 'Z') x2[cnt2++] = t;
        if(s[t] == 'x' || s[t] == 'X') x3[cnt3++] = t;
    }
    while(i < cnt1) //找'c'
    {
        while(j < cnt2 && x2[j] < x1[i]) j++; //找'z'
        if(j == cnt2) break; //没找到
        while(k < cnt3 && x3[k] < x2[j]) k++; //找'x'
        if(k == cnt3) break; //没找到
        s[x1[i++]] = s[x2[j++]] = s[x3[k++]] = -1; //找到了，标记为-1
    }
    for(int t = 0; s[t]; ++t)
        if(~s[t]) putchar(s[t]);
    return 0;
}
```

BK 复数与幂2023

难度	考点
5	快速幂

题目分析

对于一个复数 z，对于 z 的方幂，我们可以将指数分解为一系列数的和，然后将这些方幂相乘即可得到结果。

根据Hint可知，我们可以将指数转化为二进制形式，然后取出二进制数的每一位，计算这些对应的方幂（$z^1, z^2, z^4, z^8, \ldots, z^{2^n}$），最后将这些幂相乘即可。

示例代码

```
#include<stdio.h>
const long long M = 998244353;
long long multiR(long long a, long long b, long long c, long long d) //计算实部
{
    return (((a * c) - (b * d)) % M + M) % M;
}
long long multiI(long long a, long long b, long long c, long long d) //计算虚部
{
    return (((a * d) + (b * c)) % M + M) % M;
}
long long pr[70], pi[70];
int main()
{
    long long a, b;
    long long n;
    scanf("%lld%lld", &a, &b);
    scanf("%lld", &n);
    pr[0] = ((a % M) + M) % M, pi[0] = ((b % M) + M) % M;
    for(int i = 1; i <= 63; i++)
    {
        pr[i] = multiR(pr[i - 1], pi[i - 1], pr[i - 1], pi[i - 1]);
        pi[i] = multiI(pr[i - 1], pi[i - 1], pr[i - 1], pi[i - 1]);
    }
    long long ansa = 1ll, ansb = 0ll;
    for(int i = 0; i <= 62; i++)
    {
        if(((n >> i) & 1) == 1)
        {
            long long newa, newb; //不能让实部和虚部相互干扰
            newa = multiR(ansa, ansb, pr[i], pi[i]);
            newb = multiI(ansa, ansb, pr[i], pi[i]);
            ansa = newa, ansb = newb;
        }
    }
    printf("%lld %lld\n", ansa, ansb);
    return 0;
}
```

BL 和除以差

难度	考点
6	数论

题目分析

引理

若数列 $\{a_1, a_2, \ldots, a_n\}$ 对任意的 $1 \le i < j \le n$ 都有 $(a_i - a_j) | (a_i + a_j)$ （即 $(a_i - a_j)$ 整除 $(a_i + a_j)$ ），记 $b = \operatorname{lcm}\{a_1, a_2, \ldots, a_n\}$ 为 $a_1, a_2, \ldots, a_n$ 的最小公倍数，则 $\{b, b + a_1, b + a_2, \ldots, b + a_n\}$ 这 $n + 1$ 个数也满足上述差整除和的性质。

因此我们将这个题转换为了求多个数的最小公倍数。求解多个数的最小公倍数，可以采用逐个求解的方法。先求解第一个数和第二个数的最小公倍数，然后将这个数与第三个数一起求解这两个的最小公倍数，再与第四个、第五个共同求解，以此类推。

对于本题，我们可以比较容易的发现 $n = 4$ 有一组 $\{2, 3, 4, 6\}$ 的解，它的真子集也可满足 $n = 2, 3$ 的情况，因此从这一情况递推即可。

当然对于本题，你无需将你的生成过程提交进代码里，你只需要将你的答案存进数组里，根据相应的情况输出即可。

求解多个数的最小公倍数的代码

```
#include<stdio.h>
#define ll long long
ll a[100005];
ll gcd(ll a, ll b) {
    ll temp;
    if (a < b) {
        temp = a;
        a = b;
        b = temp;
    }
    while (b != 0) {
        temp = a % b;
        a = b;
        b = temp;
    }
    return a;
}
ll lcm(ll a, ll b) {
    ll c;
    c = a / gcd(a, b) *b ;//想一想为什么要先除后乘?
    return c;
}
ll arraylcm(ll s[], int start, int end) {
    if (end - start == 1) {
        return lcm(s[start], s[end]);
    } else if (end - start > 1) {
        return lcm(s[start], arraylcm(s, start + 1, end));
```

```
    }
    return 0;
}
int main() {
    int n;
    scanf("%d", &n);
    //n=3;
    int count = 0;
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
        if (a[i] == 0)count++;
    }
    if (count == 0 || count == 1) {
        printf("%lld\n", arraylcm(a, 1, n));
    } else {
        printf("0\n");
    }
    return 0;
}
```

可AC的代码（满足 $n=8$ 与 $n=9$ ）

```
#include <stdio.h>
long long a[10] = {
    14646139200,
    14646146340,
    14646146544,
    14646146680,
    14646146697,
    14646146748,
    14646146760,
    14646146816,
    14646146850
};
int main()
{
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        printf("%lld ",a[i]);
    }
}
```

More to say

如果你有更好的做法，欢迎和助教团队进行交流。

BM 锯齿数独

难度	考点
6	dfs

题目分析

这题的主要思路就是，数独有 9 行，9 列，9 宫共 27 个基本单元，对于每一个基本单元（行、列、宫）数字 1 到 9 只能出现一次。因此，对于每种单元，我们都开一个 9×9 的二维数组。其中，第一维 x 代表这种单元的编号（0 到 8），第二维 y 代表该单元里的数字 $y+1$ 是否出现过。例如 `row[3][5]` 代表第 3 行，数字 6 是否出现过，如果为 0 表示第 3 行里没出现过数字 6，如果为 1 表示第 3 行里出现过数字 6。

当我们对第 i 行 j 列进行 dfs 时，首先判断这个格子是待填写的格子还是题面给出数字的格子。如果这个格子待填写，从数字 1 到 9 进行遍历，如果这个格子所在的行、列、宫中都没有出现数字 n，就表示这里是可以填写数字 n 的。此时我们将这个格子里填上 n，并且将这个格子会影响的行、列、宫的表示数字 n 存在的变量置 1，然后向下 dfs。当回溯回来的时候，我们需要将这个格子重新置 0，并且把这个格子会影响的行、列、宫的表示数字 n 存在的变量重新置 0。如此递归下去，直到整个数独盘面被 dfs 到头了就找到解了。

示例代码 - 1

```
#include <stdio.h>
#include <stdlib.h>
int palaShape[9][9] = {}; // 存储每个格子处在几号宫内
int sudoku[9][9];
int row[9][9], cloumn[9][9], palace[9][9];
void dfs(int i, int j)
{
    if (i == 9) // 整个数独盘面被递归完了
    {
        for (int i = 0; i < 9; i++)
        {
            for (int j = 0; j < 9; j++)
            {
                printf("%d ", sudoku[i][j]);
            }
            putchar('\n');
        }
        exit(0); // 这里的作用是提前结束整个程序，因为只有唯一解，因此提前结束有助于优化时间
    }
    if (sudoku[i][j] != 0) // 如果是题面已给数字就直接向下递归
    {
        if (j < 8)
            dfs(i, j + 1);
        else
            dfs(i + 1, 0);
    }
    else
    {
        int num;
        for (num = 1; num <= 9; num++)
        {
            if (!row[i][num - 1] && !cloumn[j][num - 1] && !palace[palaShape[i]
[j]][num - 1])
            {
                sudoku[i][j] = num;
                row[i][num - 1] = 1;
                cloumn[j][num - 1] = 1;
```

```
                palace[palaShape[i][j]][num - 1] = 1;
                if (j < 8) // 向下递归
                    dfs(i, j + 1);
                else
                    dfs(i + 1, 0);
                sudoku[i][j] = 0;
                row[i][num - 1] = 0;
                cloumn[j][num - 1] = 0;
                palace[palaShape[i][j]][num - 1] = 0;
            }
        }
    }
}
int main(void)
{
    int i, j, num;
    for (i = 0; i < 9; i++)
        for (j = 0; j < 9; j++)
            scanf("%d", &palaShape[i][j]);
    for (i = 0; i < 9; i++)
    {
        for (j = 0; j < 9; j++)
        {
            scanf("%d", &num);
            sudoku[i][j] = num;
            if (num != 0) // 题面给出了初始数字，将这个格子会影响的行、列、宫进行初始化
            {
                row[i][num - 1] = 1;
                cloumn[j][num - 1] = 1;
                palace[palaShape[i][j]][num - 1] = 1;
            }
        }
    }
    dfs(0, 0); // 从头开始递归
    return 0;
}
```

示例代码 - 2

```
#include <stdio.h>
int a[9][9], b[9][9];
int row[9][9], column[9][9], region[9][9];
int dfs(int i, int j)
{
    if(i == 9) return 1;
    if(a[i][j]) return dfs(i + (j + 1) / 9, (j + 1) % 9);
    for(int k = 1; k <= 9; ++k)
    {
        if(!row[i][k - 1] && !column[j][k - 1] && !region[b[i][j]][k - 1])
        {
            a[i][j] = k;
            row[i][k - 1] = column[j][k - 1] = region[b[i][j]][k - 1] = 1;
            if(dfs(i + (j + 1) / 9, (j + 1) % 9)) return 1;
            a[i][j] = 0;
            row[i][k - 1] = column[j][k - 1] = region[b[i][j]][k - 1] = 0;
```

```
        }
    }
    return 0;
}
int main()
{
    int n = 0;
    for(int i = 0; i < 9; ++i)
        for(int j = 0; j < 9; ++j)
            scanf("%d", &b[i][j]);
    for(int i = 0; i < 9; ++i)
        for(int j = 0; j < 9; ++j)
        {
            scanf("%d", &a[i][j]);
            if(a[i][j])
                row[i][a[i][j] - 1] = column[j][a[i][j] - 1] = region[b[i][j]]
[a[i][j] - 1] = 1;
        }
    dfs(0, 0);
    for(int i = 0; i < 9; ++i)
        for(int j = 0; j < 9; ++j)
            printf("%d%c", a[i][j], "\n "[j < 8]);
    return 0;
}
```

- End -