E3 - Solution

A - 转为2进制

难度	考点
1	进制、位运算

题目分析

理解了二进制原码的表示方法后，本题就很简单了。

对于负数，可以通过判断来输出符号位。然后转而处理其相反数，因为使用原码表示数时，x和-x除符号位不同外，每一位都相同。

对于正数，可以使用“位运算”或“对2取余+除2”的方式获取N的每一位，然后输出。

示例代码

```
#include<stdio.h>
int main()
{
    int n;
    while(scanf("%d",&n)!=EOF){
        if(n>=0){//处理符号位
            printf("0");
        }
        else{
            printf("1");
            n=n*(-1);
        }
        //对每一位进行处理
        for(int i=30;i>=0;i--){
            if(n&(1<<i))
                printf("1");
            else
                printf("0");
        }
        printf("\n");
    }
    return 0;
}
```

B - 六边形战士

难度	知识点
1	循环，数学运算，特判

题目分析

本题的关键点是发现$S_{\Delta}=\frac{1}{2}ab\sin c$,然后把$n$维雷达图分成$n$个小三角形依次计算就行了

具体的，令`j=(i+1<=k?i+1:1)`，有：

$$\text{实力值}=\frac{100\sum_{i=1}^{k}\frac{1}{2}a_ia_j\sin\frac{2\pi}{k}}{\frac{1}{2}k\cdot10^2\sin\frac{2\pi}{k}}$$

$$=\frac{\sum_{i=1}^{k}a_ia_j}{k}$$

据此编写代码即可。

示例代码

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#define ll long long
const double eps=1e-11;
const double PI=3.141592654;

int main(){
    int N;
    scanf("%d",&N);
    while(N--){
        int k;
        scanf("%d",&k);
        int score[100];
        for(int i=1;i<=k;++i)
            scanf("%d",&score[i]);
        double S=0.0;
        for(int i=1;i<=k;++i){
            S+=score[i]*score[(i+1)<=k?(i+1):1];
        }
        printf("%.4lf\n",(S*1.0)/(k*1.0));
    }
    return 0;
}
```

代码中用到了“三目运算符”，具体来说，其结构是这样的：

```
逻辑语句?当逻辑语句为真时的取值:当逻辑语句为假时的取值
```

例如`x=(b==0?-1:a/b)`

等价于：

```
if(b==0)
    x=-1;
else
    x=a/b;
```

C - 三线共点===bug?

难度	考点
2	数学

题目分析

本题测试数据的数据量很大，如果暴力联立方程求有可能会TLE（笔者测试了一下，求解也还是可以过的，但不推荐），本题的正解是通过$A_1, B_1, C_1, A_2, B_2, C_2, A_3, B_3, C_3$构造出的行列式，等于0的时候即会退化到三线共点。

即：

$$\begin{vmatrix} A_1 & B_1 & C_1 \\ A_2 & B_2 & C_2 \\ A_3 & B_3 & C_3 \end{vmatrix} = 0$$

需要注意的是，仍需排除三线平行且不完全重合的情况，且需要注意$A_i, B_i, (1 \leq i \leq 3)$为0的情况

如果实在难以理解，可以把上述过程当成矩阵的一个降维，降维的条件可以等价于可以用其中两条直线推出另一条直线的方程，而三线共点可以满足两者方程便可推出第三者，且三线平行不重合时也可以满足两者方程推出第三者(三线重合计算入共点)，于是情况的分类便很清晰了

示例代码

```
#include<stdio.h>
int main(){
    int n,a[10];
    scanf("%d",&n);
    while(n--){
        for(int i=0;i<=8;i++)
            scanf("%d",&a[i]);
        if(a[0]*a[4]==a[1]*a[3]&&a[0]*a[7]==a[1]*a[6]&&(
           a[2]*a[3]!=a[5]*a[0]||a[2]*a[6]!=a[8]*a[0]||a[3]*a[8]!=a[6]*a[5]||
           a[2]*a[4]!=a[5]*a[1]||a[2]*a[7]!=a[8]*a[1]||a[8]*a[4]!=a[5]*a[7]))
            printf("A nice test data ~\n");
        else if(a[0]*a[4]*a[8]+a[3]*a[7]*a[2]+a[1]*a[5]*a[6]-
                a[2]*a[4]*a[6]-a[1]*a[3]*a[8]-a[0]*a[5]*a[7]==0)
            printf("Could be a bug!\n");
        else
            printf("A nice test data ~\n");
    }
    return 0;
}
```

D - Monica的密钥

难度	考点
4	二进制分析

题目分析

认真阅读$Hint$，基本给出了快速幂的思路和写法，只需要将求幂的累乘一并写在$while$循环里即可。

注意开$long\ long$。

示例代码

```
#include<stdio.h>
const int P = 998244353;
#define ll long long
int main(){
    ll a,b;
    scanf("%lld%lld",&a,&b);
    a%=P;

    ll k1=1,s=a;
    while(b){
        if(b&1)k1=k1*s%P;
        s=s*s%P;
        b>>=1;
    }//求k1

    ll k2=1;s=a;
    while(k1){
        if(k1&1)k2=k2*s%P;
        s=s*s%P;
        k1>>=1;
    }//求k2
    printf("%lld",k2);
    return 0;
}
```

E - 进行一个班的分

难度	考点
3	模拟

题目分析

用一个数组 a 来保存当前待分的班级中学生的成绩，用变量 n 记录当前班级中人数。

只要 $n \geq k$ 就继续分班。当前班级可分成的小班数为 $\frac{n}{k}$，对每一个小班找到其最高分和最低分，最高分**依次**存入 a 数组的开头（此时数组中前面的元素一定已经被分过班，可以直接覆盖），最低分则直接输出即可。

当不能分班时，还需要再进行一次找最低分并输出。

示例代码

```
#include <stdio.h>

int a[100005];

int main()
{
    int n, k, cnt, max, min;
    scanf("%d %d", &n, &k);
    for (int i = 0; i < n; i ++) //先将最开始的成绩信息存入a数组
        scanf("%d", &a[i]);
    while (n >= k) //只要n>=k就一直分班
    {
        cnt = n / k; //cnt为小班数，同时也是下一次分班的总人数
        for (int i = 0; i < cnt - 1; i ++)
        {
            max = 0, min = 100; //用max和min维护最大最小值
            for (int j = 0; j < k; j ++)
            {
                if (a[i*k + j] > max) max = a[i*k + j];
                if (a[i*k + j] < min) min = a[i*k + j];
            }
            a[i] = max; //最大值依次存入数组
            printf("%d\n", min); //最小值输出
        }
        max = 0, min = 100;
        for (int i = (cnt-1) * k; i < n; i ++) //对于最后一个小班，由于人数不同，需要单独处理
        {
            if (a[i] > max) max = a[i];
            if (a[i] < min) min = a[i];
        }
        a[cnt-1] = max;
        printf("%d\n", min);
        n = cnt; //下一轮分班的总人数为本次分出的小班数（每个班选一人组成临时班）
    }
    min = 100;
    for (int i = 0; i < n; i ++)
        if (a[i] < min) min = a[i];
    printf("%d\n", min);
    return 0;
}
```

F - 进制转换

难度	考点
4	进制转换

题目分析

如题，保证了输入的数在十进制下小于 2^{63}，也就是说可以用 long long 存下。

因此我们先将输入的数转为十进制，用 long long 储存，再转换为另一个进制就好了。

注意答案也是一个字符串（也就是会有字母），所以要将答案存在字符数组中，最后逆序输出。

示例代码

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

int n,m;
char ch[100],s[100];

signed main(){
    scanf("%s%d%d",ch,&n,&m);
    int len1=strlen(ch),len2=0;
    long long a=0;
    for(int i=0;i<len1;i++){            // 先将n进制下的a转换为十进制
        a=a*n;
        if(ch[i]>='0' && ch[i]<='9') a+=ch[i]-'0';
        else a+=ch[i]-'A'+10;
    }
    while(a){
        int p=a%m;
        if(p<=9) s[len2++]=p+'0';      // 将答案存在另一个字符数组中
        else s[len2++]=p-10+'A';
        a/=m;
    }
    for(int i=len2-1;i>=0;i--)
        printf("%c",s[i]);
    printf("\n");
    return 0;
}
```

G - 相反数

难度	考点
4	位运算，进制转换

题目分析

数据中$n \leq 2^{20} = 1048576$说明这个题不能再用$long\ long$解决，需要用数组模拟了。我们用字符串的形式把十六进制数读进来，并通过`strlen()`函数求长度。

首先抛开十六进制不看，二进制取相反数其实就是对二进制的数取反之后再加一，对于普通的int等整型、超长整形等类型的整数x来说，写成表达式就是`~x+1`。

而现在我们就是在用数组来模拟这个取反的过程，从头到尾循环一遍，是0变1，是1变0即可，可以用对每一位异或1的方式来解决。最后再加上1，记得进位就好。

再看十六进制转二进制，题目中提到的那个简便方法就是：对于十六进制的每一位，在二进制中都对应着四位，并且就是这一位上的数的二进制表示。

拿样例举个例子：

$$\begin{aligned} & (\quad F \quad 6 \quad 3 \quad F \quad)_{16} \\ = \ & (\quad 1111 \quad 0110 \quad 0011 \quad 1111 \quad)_2 \end{aligned}$$

于是这个进制转换就非常方便了。

示例代码

```
#include<stdio.h>
#include<string.h>
#define MAXN (1<<22)
int n,len,m=0,number[MAXN];
char str[MAXN];
int main(){
    int i,x=1;
    scanf("%d%s",&n,str);
    len=strlen(str);
    for(i=0;i<len;i++){
        if(str[i]>='0'&&str[i]<='9')
            number[i]=str[i]-'0';
        else
            number[i]=str[i]-'A'+10;
        number[i]^=15;
    }
    for(i=len-1;i>=0&&x;i--){
        number[i]+=x;
        x=number[i]>>4;
    }
    for(i=0;i<len;i++)
        printf("%d%d%d%d",(number[i]&8)>>3,(number[i]&4)>>2,(number[i]&2)>>1,
(number[i]&1));
    return 0;
}
```

H - 强迫症

难度	考点
5	位运算、排列组合

题目分析

初看题目似乎无从下手，枚举数列里的每个数不是个好方法。不妨先从简单情形入手，如果要考虑的数列中的数a_i都**只有一个二进制位**（即$k=1,m=0$或1），情况如何？

不难发现：

1. 若n个数（只能取 **0** 或 **1**，下同）做**与**运算结果为 **0**，则**至少有一个**数为 **0**，这n个数共2^n-1种取法（所有2^n种取法减去所有数全1的那一种取法）
2. 若n个数做**与**运算结果为 **1**，则**所有**数均为 **1**，这n个数仅有1种取法
3. 若n个数做**或**运算结果为 **0**，则**所有**数均为 **0**，这n个数共1种取法
4. 若n个数做**或**运算结果为 **1**，则**至少有一个**数为 **1**，这n个数共2^n-1种取法

如果扩展到这n个数都不止一个二进制位呢？由于每一位之间互不影响，我们可以简单对**每一位的情况单独做考虑**，然后做排列组合。 **每一位有多少种取法，仅仅取决于m的对应二进制位。** 也就是说，假设第i位有b_i种取法，$i=1,2,\ldots,k$,那么所有的取法一共$b_1\times b_2\times\ldots\times b_n$种。

更具体的，对于**与**运算，考虑m各个二进制位，如果为 **1** ，则这一位上，n个数只有1种取法；如果为 **0** ，则这一位上，n个数有2^n-1种取法；于是总的取法数就是$(2^n-1)^{cnt0}$种，其中$cnt0$代表m的二进制中 **0** 的个数。

对于**或**运算，考虑m各个二进制位，如果为 **0** ，则这一位上n个数只有1种取法；如果为 **1** ，则这一位上，n个数有2^n-1种取法；于是总的取法数就是$(2^n-1)^{cnt1}$种，其中$cnt1$代表m的二进制中 **0** 的个数。

于是我们只需要计算下方这两个式子，注意取模操作即可。

$$answer_1=(2^n-1)^{cnt0}$$
$$answer_2=(2^n-1)^{cnt1}$$

示例代码

```
#include <stdio.h>
long long n, k, m;
long long num = 1, cnt0 = 0, cnt1 = 0, ans1 = 1, ans2 = 1;
//注意开long long（为方便起见变量全改成long long类型），以及初始化
long long MOD = 1e9 + 7;//模数
int main()
{
    scanf("%lld%lld%lld", &n, &k, &m);

    //统计0和1个数
    for (int i = 0; i < k; i++)
    {
        if (m & (1LL << i))
            cnt1++;
        else
            cnt0++;
    }

    //依次计算: num = (2^n-1)
    //ans1 = num^cnt0
    //ans2 = num^cnt1
    for (int i = 1; i <= n; i++)
    {
        num *= 2;
        num %= MOD;
    }
    num -= 1;

    for (int i = 1; i <= cnt0; i++)
    {
        ans1 *= num;
        ans1 %= MOD;
    }
    for (int i = 1; i <= cnt1; i++)
    {
        ans2 *= num;
        ans2 %= MOD;
```

```
    }
    printf("%lld %lld", ans1, ans2);
    return 0;
}
```

I - 跳棋

难度	考点
6	一维数组、思维

题目分析

提示1

一次合法操作可视为将连续的两个1向左/右移动了一个格子

提示2

试着去掉所有连续的两个1，观察合法的初始和目标序列的特征。

题解

为表示方便，用①表示连续的两个1，例如1110表示为①10。不难发现，我们只能对①进行操作。

然后我们先分析一下对①的操作。首先是①在移动遇到了0，那么两者可以直接交换位置，即 ①0(110) ↔ 0①(011)；然后是①在移动遇到了1，那么两者可以视为交换位置，即 ①1(111) ↔ 1①(111)；最后是①在移动遇到了①，两者亦可以视为交换位置，即 ①①(1111) ↔ ①①(1111)。

于是我们可以得出结论，**①可以出现在原序列的任意位置！**

进而我们可以发现，**由原始序列生成的序列可以视为 所有的①对0和单独的1进行的"插空"。**

仍以①10（1110）为例，它可以产生的序列为将①插入‘1’前、‘1’和‘0’间、‘0’后，即①10（1110）、1①0（1110）、10①（1011）。

于是，我们仅需要把原始序列中所有的①去除，判断剩下的序列是否相同即可。

示例代码

```
#include <stdio.h>
#define maxn 1000010

int a[maxn], b[maxn];
int A[maxn], B[maxn];
int main()
{
    int t;
    scanf("%d", &t);
    while (t--)
    {
        int n;
        int pa = 0, pb = 0;
        scanf("%d", &n);
        for (int i = 1; i <= n; i++)
            scanf("%1d", &a[i]);
```

```
        for (int i = 1; i <= n; i++)
            scanf("%1d", &b[i]);
        // 处理原序列，提取0和单独的1
        for (int i = 1; i <= n; i++)
        {
            if (i + 1 <= n && a[i] == 1 && a[i + 1] == 1)
                i++;
            else
                A[++pa] = a[i];
        }
        for (int i = 1; i <= n; i++)
        {
            if (i + 1 <= n && b[i] == 1 && b[i + 1] == 1)
                i++;
            else
                B[++pb] = b[i];
        }
        // 判断处理后的序列是否相同
        int flag = 1;
        if (pa != pb)
        {
            flag = 0;
        }
        else
        {
            for (int i = 1; i <= pa; i++)
            {
                if (A[i] != B[i])
                {
                    flag = 0;
                    break;
                }
            }
        }

        if (flag)
            printf("YES\n");
        else
            printf("NO\n");
    }
    return 0;
}
```

J - Set Studio

难度	考点
7	位运算的综合运用

题目分析

题目的本质其实就实现三个方面：

- 如何用位运算把小写字母代表的元素存进集合

- 用二进制存集合后，通过位运算实现集合的基本运算
- 做第一步的逆操作，给定一个整数解析出对应的集合元素

我们看第一步：注意到小写字母$a\sim z$减去a的$ascii$码得到$0\sim 25$，如果对应上二进制的$0\sim 25$位，就想通了，即集合中有某个元素，就将其在小写字母中的位次对应的二进制位置为1，比如小写字母b，就将第一位置为1（注意本题最低位是从第0位开始算的也是便于移位运算），即与$1<<1$做或运算，对于一般的字符ch，也就是$1<<(ch-\text{'a'})$。

然后是第二步，也就是按部就班地把基本运算地数学语言转换成位运算再使用程序语言表达（接下来使用sa,sb表达存储集合的整数，s代表存储集合运算结果的整数）：

1. $A\cap B$，从集合论地定义，即有这样的元素 $x\in A,x\in B$，即对应的二进制位都为1，做与运算后，还是为1。所以集合交就是两数做与运算$sa\&sb$
2. $A\cup B$，做或运算$sa|sb$后，所有在sa,sb中分别置为1的二进制位都会被置为1，此时s的所有为1的二进制位代表A,B所有元素。
3. $\overline{A}$，此时做非运算$\sim sa$，二进制位本来为1的置为0，本来为0的置为1，也就对应了原来在集合的元素与补集中的元素。
4. $A-B$，其实有了前三条后面的运算都只是前三条的组合，又由集合差集的定义：$A-B=A\cap\overline{B}$，写成位运算：$sa\&(\sim sb)$即可。
5. $\overline{A\cap B}$和 $\overline{A\cup B}$放在一起说，可以简单地用前三种运算组合，即 $\sim(sa\&sb)$ 和 $\sim(sa|sb)$，也可以根据德摩根律：$\overline{A\cap B}=\overline{A}\cup\overline{B},\overline{A\cup B}=\overline{A}\cap\overline{B}$，写成$(\sim sa)|(\sim sb)$ 和 $(\sim sa)\&(\sim sb)$

最后一步，事实上不管操作数是0还是1，都得经过这最后一步，操作数是0则直接处理读入的整数，操作为1则处理位运算之后的整数。这里会用到移位运算，比如当前s的最低位如果是1，则第0位是1，集合有元素a，如果是0则没有，右移一位$s>>=1$，此时的s的最低位如果是1，则第1位是1，集合有元素$b\cdots\cdots$依此类推循环直到s为0。

当然需要特别注意的一点是，在做取反的位运算后，如果用`int`存储，则第$26\sim 31$位也会置为1，但我们只需要$0\sim 25$位，也就是此时s结束循环的条件不再是$s==0$，我们的处理思路是把这些1又变成0：

- 一种方法是挨个置0，即用异或运算不断异或高位，比如`s^(1<<26),...,s^(1<<31)`
- 另一种方法则是左移6位再右移6位，这个需要理解整型二进制表示与溢出的原理以及整型截断的知识，这里通过一个例子简单了解：比如32位二进制数：$(1111\ 1110\ 0000\ \ldots\ 0000)_2$，左移6位后得到的二进制数数学意义上是 $(11\ 1111\ 1000\ 0000\ \ldots\ 0000)_2$ 但是计算机只能存下从低到高的32位，也就是 $11\ 1111$ 这一部分会被舍去，留下的部分是$(1000\ 0000\ \ldots\ 0000)_2$，而此时右移6位时就有讲究了，必须用`unsigned`类型才能保证前面置0的效果即$(0000\ 0010\ \ldots\ 0000)_2$，如果是`int`类型而最高位为1时，因为涉及到符号位的问题，右移后，是会补全1的，即得到$(1111\ 1110\ 0000\ \ldots\ 0000)_2$。

示例代码

```
#include <stdio.h>
int main() {
    int op, na, nb, set_op;
    int bit, kase = 1;
    unsigned int s, sa, sb, uni;
    char elem;
    while (~scanf("%d", &op))
    {
        if (op == -1) break;
        printf("Case %d:\n", kase++);
        if (op == 0) scanf("%u", &s);
        else if (op == 1)
        {
            scanf("%d%d%d\n", &na, &nb, &set_op);
```

```
            sa = 0, sb = 0;
            for (int i = 0; i < na; ++i)
            {
                scanf("%c", &elem);
                sa |= (1 << (elem - 'a'));
            }
            getchar();
            for (int i = 0; i < nb; ++i)
            {
                scanf("%c", &elem);
                sb |= (1 << (elem - 'a'));
            }
            if (sa == sb) {
                puts("Set A is equal to set B!");
                continue;
            } else {
                uni = sa | sb;
                if (uni == sa) puts("Set B is the subset of set A!");
                else if (uni == sb) puts("Set A is the subset of set B!");
                else puts("No subset relationship!");
            }
            switch (set_op)
            {
                case 0:
                    s = sa & sb;
                    printf("The intersection of set A and set B is:");
                    break;
                case 1:
                    s = sa | sb;
                    printf("The union of set A and set B is:");
                    break;
                case 2:
                    s = ~sa;
                    printf("The complement of set A is:");
                    break;
                case 3:
                    s = sa & (~sb);
                    printf("The difference of set A and set B is:");
                    break;
                case 4:
                    s = (~sa) | (~sb);
                    printf("The complement of the intersection of set A and set B is:");
                    break;
                case 5:
                    s = (~sa) & (~sb);
                    printf("The complement of the union of set A and set B is:");
                    break;
                default:
                    break;
            }
        }
        if (s == 0) {
            puts("Empty set");
            continue;
        }
        bit = 0;
```

```
        putchar('{');
        s = ((s << 6) >> 6);
        while (s > 0 && bit < 26) {
            if (s & 1) {
                printf("%c", 'a' + bit);
                if (s / 2 > 0 && bit + 1 < 26) putchar(',');
            }
            ++bit;
            s >>= 1;
        }
        printf("}\n");

    }
    return 0;
}
```