E3 - Solution

A 摩卡与补码 2

难度	考点
1	位运算、循环

题目分析

首先，假如这道题不是多组输入，而是只输入一行 32 个 01 位，让我们还原这个整数。我们可以参考 hint，利用 hint 中的第二段代码可以实现一次读入一位，利用 hint 中的第一段代码可以实现根据一串补码还原出对应的整数，如下所示：

```
for(int i = 31; i >= 0 ;i--) {
    int bit ;
    scanf("%1d", &bit);
    ans |= bit << i;
}

printf("%d\n", ans);
```

现在，摆在我们眼前的唯一一个难题就是处理不定组输入。以前我们也做过不定组输入的题，而且还没少做，就是用 `EOF` 来控制循环的结束。实际上，对于 `scanf`，如果它返回了 `EOF`，就说明读到了输入结尾，没有东西可读了。

但是每次我们不能一下把一行整个整数读进来，第一因为它太大了，足足有 32 位；第二因为我们把它一下读进来后不好分离出 01 位了。我们仍然可以使用 hint 中的方法，一次**试着**读入一位，如果这次返回了 `EOF`，就说明输入没了，可以跳出循环结束程序了；如果没返回 `EOF`，就说明我们读入了新的一个补码的第一位（准确地讲是第 31 位），我们已经读入了一位，所以循环里只要再读入 31 次就可以了，但是别忘了提前读的这一位也要放在整数上。

```
while(scanf("%1d", &bit) != EOF) {
    int ans = bit << 31;

    // 读剩下的 31 位
    for(int i = 30; i >= 0 ;i--)
        do something ...
}
```

示例代码

```
#include <stdio.h>

int main() {
    int bit;
    // 想一想，写 scanf("%d", &bit) 会发生什么
    while(scanf("%1d", &bit) != EOF) {
        int ans = bit << 31;
```

```
        for(int i = 30; i >= 0 ;i--) {
            scanf("%1d", &bit);
            ans |= bit << i;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

B 摩卡与冰雹序列

难度	考点
1	模拟、数组

题目分析

分析题意，根据输入的 n，执行循环操作，直到 n 变到 1 为止。此时输出变换的次数（包括首尾）和变化序列。

我们有两种思路先输出次数再输出序列：一种思路是边循环边维护一个数组，最后输出这个数组；另一种是执行两遍循环，第一次专门用来算次数，第二次专门用来输出序列。

可以看到，两种方法一种用了更多的空间从而减小了时间，一种用了更多的时间从而减小了空间。在以后的学习中我们也经常会遇到这种时空的权衡。

下面只给出利用数组的代码。

示例代码

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

int num[300];

int main() {
    int n;
    scanf("%d", &n);

    int cnt = 1;    // 记录数组中元素的个数
    num[1] = n;

    while(n != 1) {
        if(n % 2 == 1)      // 是奇数
            n = n * 3 + 1;
        else
            n = n / 2;

        cnt++;
        num[cnt] = n;
    }

    printf("%d\n", cnt);
```

```
    for(int i = 1; i <= cnt ;i++)
        printf("%d ", num[i]);
    return 0;
}
```

C 奇怪的位运算 2024

难度	考点
2	位运算

题目分析

a_i	b_i	$a_i \odot b_i$
0	0	0
0	1	0
1	0	1
1	1	0

位运算对于每一位都是独立的，这个奇怪的位运算 $\odot$ 也一样。$\odot$ 可以看作是按位取反和按位求与的复合运算，即 $a_i \odot b_i = a_i \wedge (\sim b_i)$，其中 $\wedge$ 表示按位求与，$\sim$ 代表按位取反。于是我们可以先对 b 取反，再将此结果与 a 求与，即可直接得到 $a \odot b$ 的值。（或者你也可以看成 $a_i \odot b_i = a_i \wedge (a_i \oplus b_i)$，其中 $\oplus$ 代表按位异或）

当然，如果你执意要用数组的话，注意本题的数据范围，`1 << i` 是可能爆 int 的，需要写成 `1ll << i`，还要注意运算符优先级的问题。

（我说的不开数组指的是观察出 a, b 需要通过什么位运算的组合来直接得到答案，不是要你脱离数组之后还在一位一位加啊！）

示例代码

```
#include <stdio.h>
int main(void)
{
    long long a, b;
    while (scanf("%lld%lld", &a, &b) != EOF)
    {
        printf("%lld\n", a & ~b);
    }
    return 0;
}
```

示例代码（数组版，不推荐）

```
#include <stdio.h>
int main(void)
{
    int a1[66], b1[66];
    long long a, b;
    while (scanf("%lld%lld", &a, &b) != EOF)
    {
        for (int i = 0; i < 64; i++)
        {
            a1[i] = a >> i & 1;
            b1[i] = b >> i & 1;
        }
        long long c = 0;
        for (int i = 0; i < 64; i++)
        {
            if (a1[i] == 1 && b1[i] == 0)
            {
                c |= 1ll << i;
            }
        }
        printf("%lld\n", c);
    }
    return 0;
}
```

示例代码（逐位考虑，不推荐）

```
#include <stdio.h>
int main(void)
{
    long long a, b;
    while (scanf("%lld%lld", &a, &b) != EOF)
    {
        long long c = 0;
        for (int i = 0; i < 64; i++)
        {
            if ((a >> i & 1) && !(b >> i & 1)) // 注意运算符优先级问题，& 优先级低于
&&
            {
                c |= 1ll << i;
            }
        }
        printf("%lld\n", c);
    }
    return 0;
}
```

D 宁愿选择榴莲不放手

难度	考点
2	数组

题目分析

数组的下标除了用来表示数组中元素的顺序以外，还可以用来表示某种状态，此时数组元素的值可以用来计数。

因为榴莲数很大，但榴莲房数只有 10000 种可能，因此我们可以用下标表示房数，用对应的数组元素值表示相应房数的榴莲个数。

当输入完成后，我们只需要按顺序输出即可完成此题。

注：本题实际上对榴莲房数完成了排序，这种排序方法称为计数排序。感兴趣的同学可以自行搜索研究。

示例代码

```
#include<stdio.h>
int n,x,A[10086];
int main(){
    scanf("%d",&n);
    while ( n-- ){
        scanf("%d",&x);
        A[x]++;//房数为 x 的榴莲数量 +1
    }
    for ( int i=1 ; i<=10000 ; i++ )
        if ( A[i] )// A[i] 不为 0 才输出
            printf("%d : %d\n",i,A[i]);
    return 0;
}
```

E Orch1d 的军训

难度	考点
3	二进制转换

题目分析

通过观察可以发现，我们可以将同学们**面向教官**和**背对教官**分别定义为 **0** 和 **1** 两种状态，这样整个过程可以转化为用二进制计数：初始状态为 **0**；报数为 1 的同学为二进制最低位，以此类推；每次口令相当于让最低位进行**加一操作**，“如果报数为 i 的同学再次面向教官，则报数为 i + 1 的同学立刻向后转”相当于**进位操作**。

因此只需要将给出的十进制数转化为二进制后，求出所有位中 **1 的个数**，以及 **1 所在的位数**即可（注意按照从低位到高位的顺序输出）。

示例代码

```
#include<stdio.h>

int main()
{
    long long n;//记得开long long
    scanf("%lld",&n);
    int ans=0,num[100],sum=1;
    while(n)
    {
        if(n&1) num[++ans]=sum;
        n>>=1;sum++;
    }
    printf("%d\n",ans);
    for(int i=1;i<=ans;i++) printf("%d ",num[i]);//位数从1开始，不是0
    return 0;
}
```

F 与或异或

难度	考点
4	位运算、与或的性质，贪心

题目分析

由 $HINT$ 可知，与运算是数越多结果越小，而或运算是数越多结果越大。因此，若与运算中数的个数大于 1 ，其结果一定不大于将交界处的数分给或运算后的结果。根据贪心原理，与运算仅计算第一个数。因而，本题只需枚举或和异或的分界点，并用遍历求得最终价值即可。

拓展：原题如何求解？（即 $3 \leq n \leq 2 \times 10^6$）

$HINT$：如果你的代码所有测试点均小于 $10ms$ 或总评测时间小于 $50ms$ ，代表你通过原题，否则将 TLE。

示例代码

std：

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

long long s1[10000];

int main()
{
    long long n,ans,i,j,sum1,sum2;
    scanf("%lld",&n);
    for(i=1;i<=n;i++)
```

```
    {
        scanf("%lld",&s1[i]);
    }
    ans=0;
    for(j=2;j<n;j++)
    {
        sum1=sum2=0;
        for(i=2;i<=j;i++)
        {
            sum1|=s1[i];
        }
        for(;i<=n;i++)
        {
            sum2^=s1[i];
        }
        if(ans<=sum1+sum2)
        {
            ans=sum1+sum2;
        }
    }
    printf("%lld",ans+s1[1]);
    return 0;
}
```

G 让我把记忆煎成饼

难度	考点
5	状态压缩

题目分析

由于配料不超过 20 种，因此我们可以通过一个整型变量来表示每种配料是否添加。将这个变量的二进制表示写出来，从低位到高位，如果第 i 位为 1， 则表示添加了这种配料；为 0，则表示没有添加。

这样一来，这个变量的取值范围就是 0 到 2^n-1 之间，因此就可以用循环枚举。

这种方法的时间复杂度是 $O(n\times 2^n)$，可以通过本题。

注：本题还应当考虑最后的总价格是否在 `int` 范围内。事实上，当所有输入数据都取得最大值时，本题的答案为 609222656，已经比较接近 `int` 类型能表示的最大值了。本题并不好估算结果的范围，但输入数据较短，因此同学们可以自行尝试从小到大的数据，判断是否发生溢出。

示例代码

```
#include<stdio.h>
int n,A[25],ans;
int main(){
    scanf("%d",&n);
    for ( int i=0 ; i<n ; i++ )
        scanf("%d",&A[i]);
    for ( int status=(1<<n)-1 ; status>=0 ; ans+=6,status-- )//每个status都表示一种煎饼。这里需要注意运算符的优先级，如果写成 status=1<<n-1 的话，会先运算 n-1 ，再进行左移，从而使得运行结果不正确
        for ( int j=0,cnt=0 ; j<n ; j++ )
            if ( (status>>j)&1 )//添加了这种配料
                ans+=A[j]*(++cnt);
    printf("%d",ans);
    return 0;
}
```

H 粗心的 Orch1d

难度	考点
5	位运算、枚举

题目分析

根据 hint 进行读入和获取长度。

已知对于一个十进制数进行了两次进制转换，每次都会写错一位数，因此我们可以考虑更改每一位的值，求出所有种可能的情况，再在里面找出正确答案。
可以考虑从写错的二进制数入手，每次更改二进制的一位数的值，如果是 0 则置 1，如果是 1 则置 0，再将得到的结果转化为题目中的 k 进制数，通过对比如果只有一位不同，即为正确答案。

示例代码

```
#include<stdio.h>
#include<string.h>
int k,B[33];
unsigned ans,wa=0;
char a[33],b[33];
int main()
{
    scanf("%d%s%s",&k,a,b);
    int lena = strlen(a), lenb = strlen(b);
    for(int i=0;i<lena;++i)
        wa<<=1,wa+=a[i]-'0';
    for(int i=0;i<lenb;++i){
        if(b[i]>='0'&&b[i]<='9') B[i]=b[i]-'0';
        else B[i]=b[i]-'A'+10;
    }
    for(int i=0;i<31;++i){
```

```
            ans=wa^(1u<<i);
            int tot=0,c[33]={0},num=0;
            while(x){
                c[tot++]=x%k;
                x/=k;
            }
            for(int i=0;i<32;++i){
                if(i<strlen(b)) num+=(c[i]!=B[strlen(b)-1-i]);
                else num+=(c[i]!=0);
            }
            if(num==1){
                if(!ans){
                    printf("0");
                    break;
                }
                int anss[33]={0},tot=0;
                while(ans){
                    anss[tot++]=ans%k;
                    ans/=k;
                }
                for(int i=tot-1;i>=0;--i){
                    if(anss[i]>=0&&anss[i]<=9) printf("%d",anss[i]);
                    else printf("%c",anss[i]+'A'-10);
                }
                break;
            }
        }
        return 0;
    }
```

I 武魂融合技

难度	考点
6	位运算、贪心

题目分析

如果这道题采用暴力枚举的做法，即使用两个`for`循环枚举出每两个数之间的 & 结果，会出现时间超限的问题，因此考虑一种贪心的做法：
由于要找出与运算的最大值，所以我们首先将所有数转化为二进制，再从最高位开始考虑，只有**1所在的位数越高**，答案的值才可能越大；除此之外，由与运算的定义可知：只有 $1\&1=1$，其他情况的结果都为0，因此如果想要答案中这一位的值为1，必须存在**两个及以上**的数这一位为1。
综上，我们只需要从最高位到最低位遍历，如果第 i 位满足有**两个及以上**的数为1，则答案中这一位置为 1，在找第 i-1 位时只需在第 i 位中为 1 的数中查找即可；如果第 i 位不满足，则答案中这一位置 0，在找第 i-1 位时只需在比第 i 位高且满足条件的数中找即可。

示例代码

```
#include<stdio.h>
int main()
{
    int n,a[100005];
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d",&a[i]);
    }
    int jud[100005]={0},ans=0;//jud为1表示不用考虑该数
    for(int i=30;i>=0;i--)//从最高位到最低位
    {
        int sum=0;
        for(int j=1;j<=n;j++)
        {
            if(jud[j]) continue;
            if(a[j]>>i&1) sum++;
        }
        if(sum>=2)//满足有两个以上数这一位为1
        {
            ans|=(1<<i);//答案中置1
            for(int j=1;j<=n;j++)
            {
                if(!jud[j]) jud[j]=!(a[j]>>i&1);//如果第j个数该位为0，则下次循环不考虑
该数
            }
        }
    }
    printf("%d",ans);
    return 0;
}
```

J ddz 与位运算等式

难度	考点
6	按位计算、计数

题目分析

本题给定 a, b, c，求满足 $(d|a) \oplus (d\&b) = c$ 的 d 的个数，注意到这个等式的运算符只有位运算，说明可以按位计算，不同位之间互不影响。

我们对于 a, b, c 的每一位枚举所有可能的情况：

a	b	c	→	d
0	0	0		0
0	0	1		1

a	b	c	→	d
0	1	0		0或1
0	1	1		不存在
1	0	0		不存在
1	0	1		0或1
1	1	0		1
1	1	1		0

根据上表可以得到，如果 a 和 b 的某一位同时为 1，无论 c 的这一位是多少 d 在这一位都只有一种可能的取值，对于其他情况，如果 a 和 c 在这一位不相同，则无论 d 的这一位为何值都不可能，否则 d 在这一位可以是 0 或 1。由此对 a, b, c 的每一位求 d 在这一位的可能的取值的个数，将他们相乘即可。

示例代码

```
#include<stdio.h>

int main(){
    int n;
    long long a, b, c;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%lld %lld %lld", &a, &b, &c);
        long long result = 1;
        for (int j = 0; j < 60; ++j) {
            long long A = (a & (1ll << j));
            long long B = (b & (1ll << j));
            long long C = (c & (1ll << j));
            if (A ^ B) {
                if (A ^ C) {
                    result = 0;
                } else {
                    result *= 2;
                }
            }
        }
        printf("%lld\n", result);
    }
    return 0;
}
```

原题参考

https://codeforces.com/problemset/problem/2020/C

K max(a,b)

难度	考点
2	位运算

题目分析

这道题禁止了比较操作和部分函数，所以我们需要从一些更本质的地方入手。

$a > b$ 意味着 $a - b > 0$，也就是说只要知道了$a - b$ 的符号，就可以知道 a 和 b 的大小关系。

```
long long c = ((a - b) >> 63) & 1;
// a - b 可能超过 int 的表示范围
```

而如何通过 c 为 0 还是 1 判断输出的较大值是 a 还是 b 呢？显然我们可以利用简单的加减乘除运算，通过给 a 和 b 赋上不同的系数，以此在不使用 `if` 等结构的限制下，实现一个输出 a 还是 b 的“选择分支结构”。

```
printf("%lld\n", c * b + (1 - c) * a);
```

值得一提的是，如果想全程使用位运算的话，也是可以做到的。具体代码如下，感兴趣的同学可以自行分析理解。

```
c = ((b & ((a - b) >> 63)) | (a & ((~(a - b)) >> 63)));
```

示例代码

```
#include <stdio.h>

long long a, b, c;
int main() {
    scanf("%lld%lld", &a, &b);
    c = ((a - b) >> 63) & 1;
    printf("%lld\n", c * b + (1 - c) * a);
    scanf("%lld%lld", &a, &b);
    c = ((a - b) >> 63) & 1;
    printf("%lld\n", c * b + (1 - c) * a);
    scanf("%lld%lld", &a, &b);
    c = ((a - b) >> 63) & 1;
    printf("%lld\n", c * b + (1 - c) * a);
    return 0;
}
```