E2-2021级航类-第二次练习赛题解

A 简单浮点数(水题速来)

水题，没啥细节，有问题的同学要抓紧了

参考代码

```
#include <stdio.h>
int main()
{
    int n, i;
    float sum = 0;
    float x;
```

```
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
    {
        scanf("%f", &x);
        sum += x;
    }

    printf("%.2f", sum * 0.95);
    return 0;
}
```

B 简单位运算

水题,但是需要注意数据范围

参考代码

```
#include <stdio.h>
int main(){
    long long a, b, c;
    scanf("%lld%lld%lld", &a, &b, &c);
    printf("%lld\n", a ^ b ^ c);
    return 0;
}
```

C 简单的字符串统计

题目主要帮助大家练习用EOF判断字符数

参考代码

```
#include <stdio.h>
int main()
{
    int n = 0;
    while (~getchar()) //这里的~是位运算中的“非”
    {                  //等价于 getchar()!=EOF
        n++;           //因为EOF的值实际上是-1，-1的二进制表示为全1
    }
    printf("%d", n);
    return 0;
}
```

D 2的次幂

难度	考点
2	位运算

问题分析

本题是考察位运算和数据范围，当然不用位运算而使用循环也能通过，一下这段代码可以简单的判断一个数是不是2的次幂：

```
if((n&(n-1))==0)
    puts("n是2的次幂");
```

至于找到不超过n的最大的2的次幂，很多同学会有如下做法：

```
unsigned int n, i = 1;
while (i < n)
    i = i * 2;

if (i == n)
    printf("%u\n", i);
else
    printf("%u\n", i / 2);
```

这样的写法会出现一些问题，就如我公告里提示的，当输入的 $n = 2147483649(2^{31}+1)$ 时，$i = 2^{31}$ 时会再次进入循环，但是此时再乘以2就会导致溢出，溢出的结果是 0 ，从而陷入了死循环，导致 TLE

解决的办法有两种，一直使用long long类型的数据去累乘，或者限制乘法的次数

参考代码1

```
#include <stdio.h>
int main()
{
    unsigned int a;
    long long i = 1;
    scanf("%u", &a);
    if ((a & (a - 1)) == 0)
        printf("Yes\n%u\n", a);
    else
    {
        while (i <= a)
            i *= 2;
        printf("No\n%lld\n", i / 2);
    }
    return 0;
```

```
}
```

参考代码2

也可以计算a的二进制中1的个数来判断

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    unsigned int a;
    scanf("%u", &a);

    int cnt = 0, maxi = 0;
    for (int i = 0; i < 32; i++)
        if (a & (1u << i))
            cnt++, maxi = i;

    printf(cnt == 1 ? "Yes\n" : "No\n");
    printf("%u\n", 1u << maxi);

    return 0;
}
```

E 找内鬼

难度	考点
2	异或运算

题目分析

解决本题的关键点在于理解异或运算。异或运算“$\bigoplus$”有如下一些性质（以下 $\bigoplus$ 为异或运算）：

- $0 \bigoplus a = a$
- $a \bigoplus a = 0$
- 异或具有交换律
- 异或具有结合律

对于偶数次出现的两个数 a，有 $a \bigoplus a = 0$。假设有一组数列 $\{a, b, c, a, b, c\}$，运用异或运算的交换律和结合律可得：

$$
\begin{aligned}
& a \bigoplus b \bigoplus c \bigoplus a \bigoplus b \bigoplus c \\
= & (a \bigoplus a) \bigoplus (b \bigoplus b) \bigoplus (c \bigoplus c) \\
= & 0 \bigoplus 0 \bigoplus 0
\end{aligned}
$$

$$= 0$$

对于仅有一个数字 d 出现奇数次，其他数均出现偶数次的数列 $\{a, b, c, d, a, b, c\}$ 运用异或运算的交换律和结合律可得：

$$\begin{aligned} & a \bigoplus b \bigoplus c \bigoplus d \bigoplus a \bigoplus b \bigoplus c \\ = & (a \bigoplus a) \bigoplus (b \bigoplus b) \bigoplus (c \bigoplus c) \bigoplus d \\ = & 0 \bigoplus 0 \bigoplus 0 \bigoplus d \\ = & d \end{aligned}$$

由以上两个例子考察本题，若输入的数全部出现了偶数次，则输入的数据个数将会是**偶数**；若输入的 n 个数中有1个数出现了奇数次，其他数字都出现了偶数次，将输入数据全部进行异或运算，则最后的运算结果就会是唯一出现奇数次的数字。代码如下：

参考代码

```
#include<stdio.h>

int main(){
    int input;
    int m, i, temp = 0;
    scanf("%d", &m);
    for(i = 0; i < m; i++){
        scanf("%d", &input);
        temp = temp ^ input;
    }
    if(m % 2)//等效于if(m % 2 == 1),即当 m 为奇数时为真
        printf("%d", temp);
    else
        printf("False Alarm.\n");
    return 0;
}
```

本题并不需要数组保存输入数据，因为每个数据只需要处理一次便可以丢弃，因此在输入数据的阶段一边输入一边处理即可。

F LJF赶校车

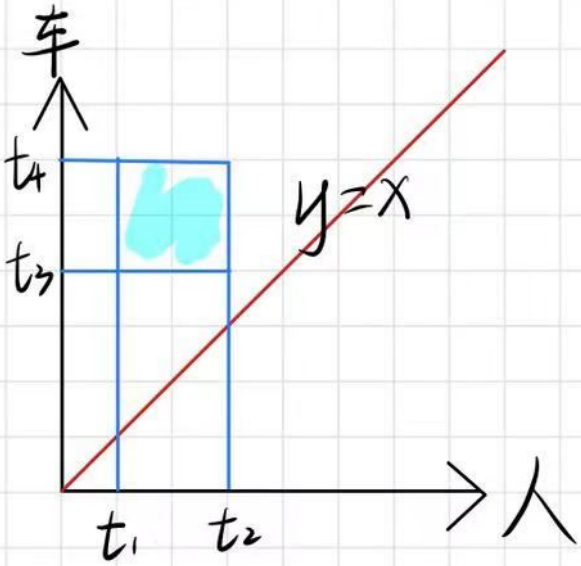
难度	考点
3	数学计算，几何概型

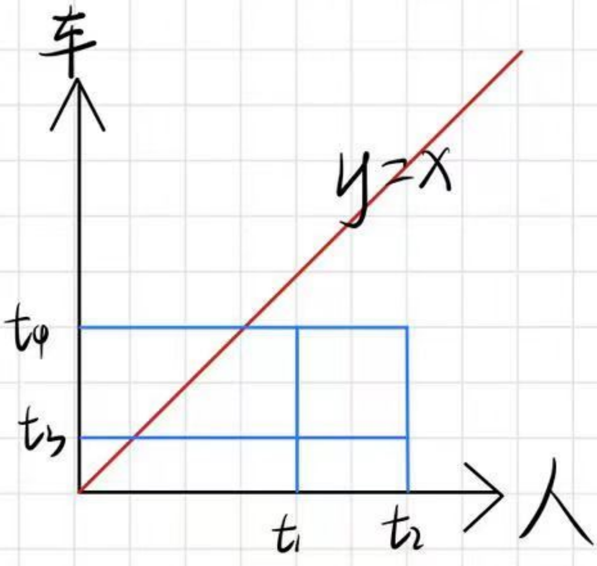
问题分析

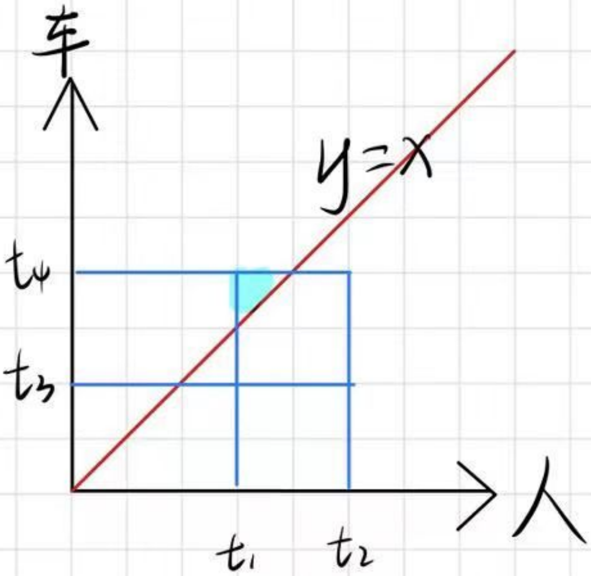
分别给出人、车的到达最早以及最晚时间，利用平面直角坐标系，给出直线$y=x$，计算出满足$y>x$部分面积与矩形总面积之比。

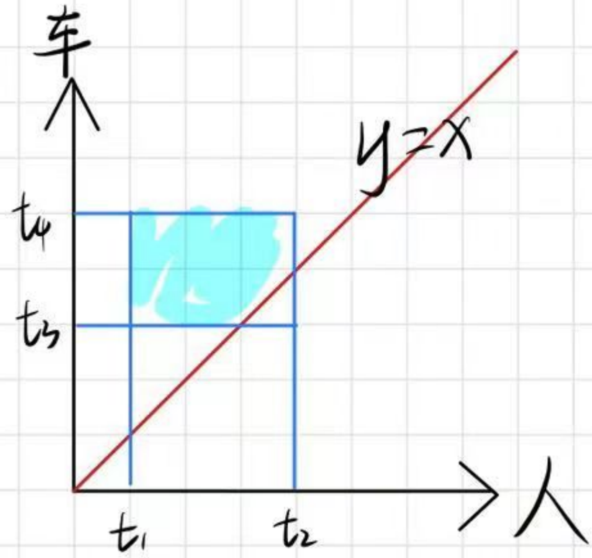
为了方便计算，可将4个时间均化为分钟，记作t_1,t_2,t_3,t_4。

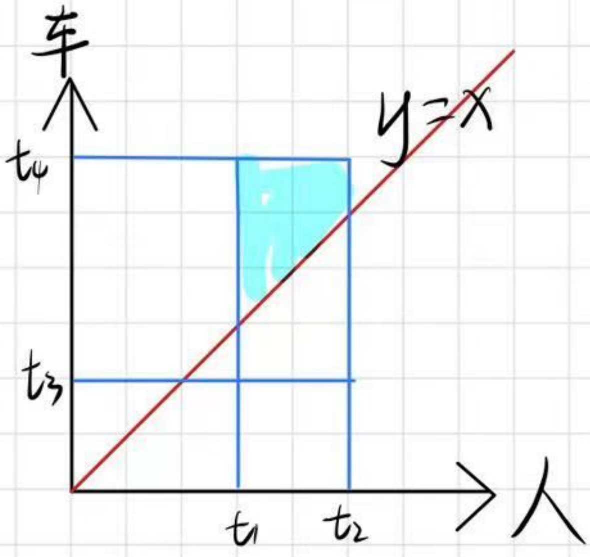
具体可分6大类进行讨论，具体如下图：

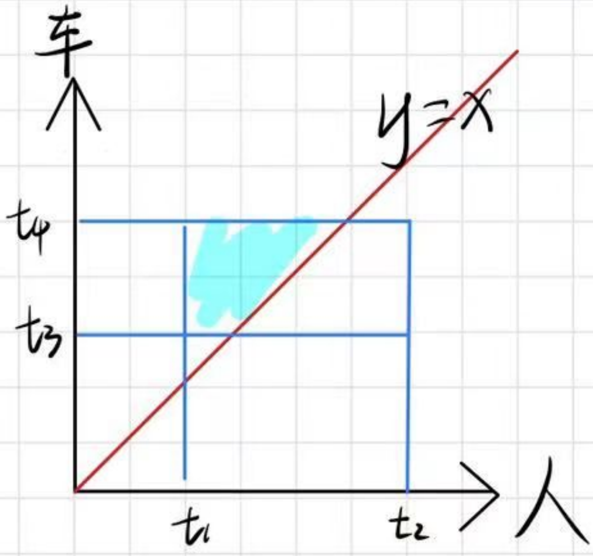

车
人
t_1
t_2
t_3
t_4
$y=x$


车
人
t_1
t_2
t_3
t_4
$y=x$


车
人
t_1
t_2
t_3
t_4
$y=x$


车
人
t_1
t_2
t_3
t_4
$y=x$


车
人
t_1
t_2
t_3
t_4
$y=x$


车
人
t_1
t_2
t_3
t_4
$y=x$

参考代码

```
#include <stdio.h>
int main()
{
    int h[5], m[5], t[5];
    for (int i = 1; i <= 4; i++)
    {//将时间化为分钟
        scanf("%d:%d", &h[i], &m[i]);
        t[i] = h[i] * 60 + m[i];
    }
    int Area = (t[2] - t[1]) * (t[4] - t[3]); //计算矩形面积
    int S=0;//阴影处面积

    //接下来分情况讨论
    if (t[3] >= t[2]) //图1-1
        puts("GOOD!");
    else if (t[4] <= t[1]) //图1-2
        puts("0HN0!");
    else if (t[3] <= t[1] && t[4] <= t[2]) //图2-1
        S = (t[4] - t[1]) * (t[4] - t[1]) / 2;
    else if (t[3] > t[1] && t[4] > t[2]) //图2-2
        S = Area - (t[2] - t[3]) * (t[2] - t[3]) / 2;
    else if (t[3] <= t[1] && t[4] > t[2])//图3-1
        S = ((t[4] - t[2]) + (t[4] - t[1])) * (t[2] - t[1]) / 2;
    else if (t[3] > t[1] && t[4] <= t[2])//图3-2
        S = ((t[4] - t[1]) + (t[3] - t[1])) * (t[4] - t[3]) / 2;
    if(S)
        printf("%.8f", 1.0 * S / Area);
}
```

G 今天也要牵绊变身

问题分析

根据unsigned int类型的长度为32位，得知只需5次即可完成任务。

当然，左移32位的话，根据C语言循环左移的特性，会得到原数。对于任意一个unsigned int类型的数x，(x<<32)仍旧是x本身，于是异或的结果x^(x<<32)就是0。如果执行了这一步（多执行了第6步），程序就会失败。因此应当恰好执行5次。

参考代码1

```
#include<stdio.h>

int main()
{
    unsigned int k1;
    while(scanf("%u",&k1)!=EOF)
    {
        unsigned int k2=k1^(k1<<1);
        unsigned int k4=k2^(k2<<2);
        unsigned int k8=k4^(k4<<4);
        unsigned int k16=k8^(k8<<8);
        unsigned int k32=k16^(k16<<16);
        printf("%u\n",k32);
    }
}
```

参考代码2

其实这题也能从最低位开始逐位反解

```
// AUTHOR:cbd
#include <stdio.h>
int main()
{
    unsigned int a;
    while(scanf("%u",&a)!=EOF)
    {
        unsigned b=0,c=0;
        for(int i=0;i<32;i++)
        {
            if(i)c|=((b>>(i-1))&1)<<i;
            b|=(((a^c)>>i)&1)<<i;
        }
        printf("%u\n",b);
    }

    return 0;
}
```

H 5421码

问题分析

这道题的思路较为清晰，首先0-9的5421BCD码与其二进制码之间的对应关系，0-4的5421码与其二进制码完全相同，而5-9的bcd码与其二进制码正好差3。在确定两者的映射关系后，对十进制的每一位转换成相应的bcd码，迭代进行左移运算和位或运算。

在求解二进制码不同位的个数时可灵活利用异或运算，两数异或完二进制码中1的个数即不同位的个数。在进行数据处理前需要保存a的副本，不能直接对修改后的a进行操作。

这道题的数据范围也值得关注，由于a的最大值为16位10进制，对应二进制64位，因此需要使用unsigned long long数据类型，否则会出现溢出现象。

参考代码

```
#include <stdio.h>
int main()
{
    unsigned long long a, b = 0ull;
    scanf("%llu", &a);
    unsigned long long tem = a; //提前保存好a的副本
    int count = 0;
    while (a > 0)
    {
        unsigned long long num = a % 10ull; //取出a的十进制每一位
        if (num < 5)
        {
            b |= (num << (4ull * count)); //对于0-4,bcd码与二进制码相同，直接左移位或即可
        }
        else
        {
            b |= ((num + 3ull) << (4ull * count)); //对于5-9,需要在二进制码基础上加3
        }
        a /= 10ull;
        count++;
    }
    unsigned long long k = tem ^ b; //异或运算求解不同位的个数
    int c = 0;
    for (int i = 0; i < 64; i++)
    {
        if ((1ull << i) & k)
        { //若该位为1，说明该位二进制码不同
            c++;
        }
    }
    printf("%llu %d", b, c);
    return 0;
}
```

I 浮点数协议

难度	考点
3	浮点数协议、位运算

问题分析

完成这题需要的一个关键步骤已经在HINT中给出：`float`类型的数据是无法参加位运算的，因此我们将读入的单精度浮点数的每一个二进制位直接复制进一个`int`类型的变量中，这样我们就可以通过这个`int`型变量来操作读入的单精度浮点数的二进制位了。

之后我们需要使用位运算提取出这个`int`型变量的$e_7e_6...e_0$和$f_{22}f_{21}...f_0$，计算出E和F。注意计算E的时候不需要自己写循环计算十进制值，可以将$e_7e_6...e_0$存储进`int`变量然后直接输出；另外需要注意$f_{22}f_{21}...f_0$全为1的时候E应该为1。

参考代码

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

#define bi 32

int main(int argc, const char * argv[])
{
    float a;
    while (~scanf("%f", &a))
    {
        int e_mask = 0x7f800000, da; // e_mask: 0 11111111 00000000000000000000000
        memcpy(&da, &a, 4);
        int bits[bi];
        for (int i = 0; i < bi; i++)  // 提取出每一位
        {
            bits[i] = 1 & (da >> (bi - 1 - i));
        }
        printf((e_mask & da) == 0 ? "0." : "1."); // 输出F
        for (int i = 9; i < bi; i++)
        {
            printf("%d", bits[i]);
        }
        printf(" %d\n", (e_mask & da) >> 23 == 0 ? 1 : (e_mask & da) >> 23); // 输出E
    }

    return 0;
}
```

J n重循环移位

难度	考点
4	位运算

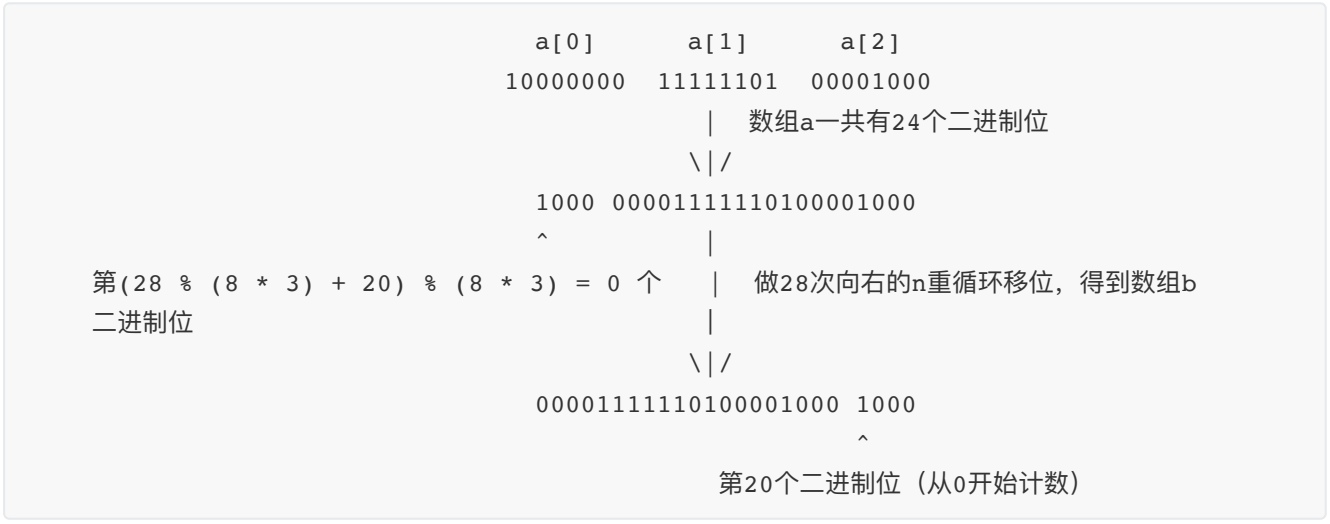
问题分析

对于一个有`n`个`int`型数据的数组`a`，假设我们想对它做`t`次“n重循环移位”操作，首先我们需要想到：

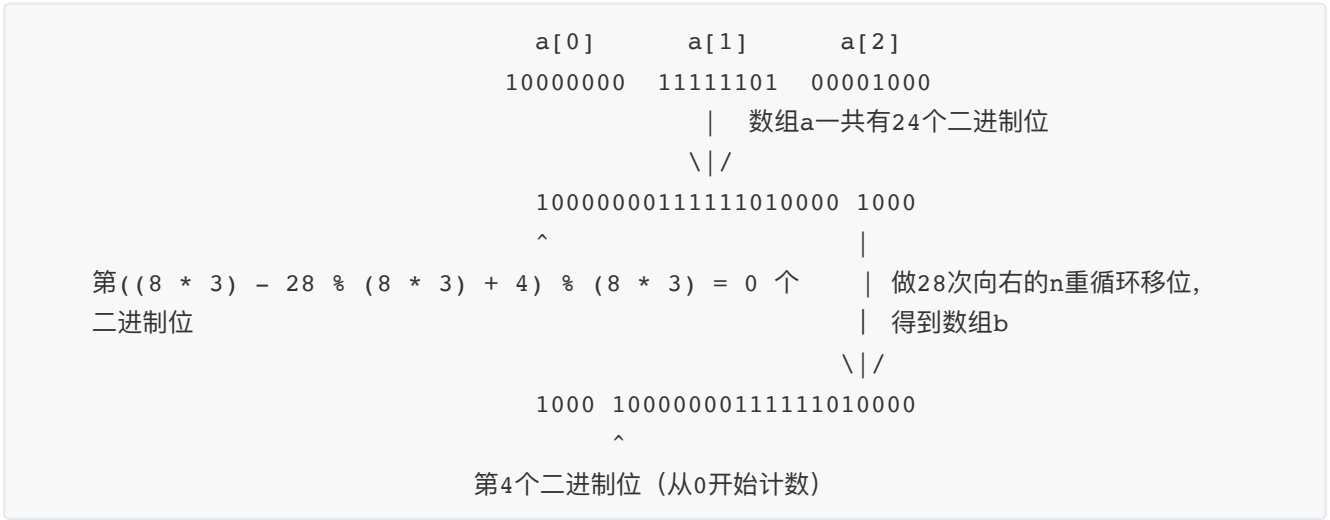
一个`int`型数据有32个二进制位，那么数组`a`中就总共有`32 * n`个二进制位，因此我们做`t`次“n重循环移位”操作和做`t % (32 * n)`次“n重循环移位”操作得到的结果是相同的。

其次，假设我们将做完`t`次“n重循环移位”操作的数组`a`称为数组`b`，我们可以想到：

- 对于向左的“n重循环移位”：数组`a`中的第`(t % (32 * n) + i) % (32 * n)`个二进制位和数组`b`中的第`i`个二进制位是一样的；



- 对于向右的“n重循环移位”：数组`a`中的第`((32 * n) - t % (32 * n) + i) % (32 * n)`个二进制位和数组`b`中的第`i`个二进制位是一样的。



因此，我们可以在数组`a`中找到`b[i]`的32个二进制位的位置，将它们找出来之后拼成一个`int`型数据，就得到了结束数组中的元素`b[i]`，具体写法参考示例代码。

参考代码

```
#include <stdio.h>

#define bi 32 // int型数据有32个二进制位

int main()
{
    int n, a[100001], t;
    char op;
    while (~scanf("%d", &n)) // 读入n
    {
        for (int i = 0; i < n; i++) // 读入数组的所有数据
        {
            scanf("%d", &a[i]);
        }
        getchar(); // 吃掉换行符
        scanf("%c %d", &op, &t); // 读入op和t
        int ds = n * bi, cnt = 0, num = 0, start, i;
        /* ds为数组a中的二进制位数
         * cnt记录读到的二进制位数，满32位就输出num
         * num存储b[i]
         * start为b的第0个二进制位在数组a中的位置
         * i记录目前处理到数组a中的第几个二进制位
         */
        start = op == 'l' ? t % ds : ds - t % ds; // start为b的第0个二进制位在数组a中的位置
        i = start; // 从第start个二进制位开始处理
        do {
            num = num << 1 | ((a[i / bi] & 1 << (bi - i % bi - 1)) == 0 ? 0 : 1);
            /* 数组a的第i个二进制位的所在位置为：
             * (a[i / 32] & 1 << (32 - i % 32 - 1)
             */
            cnt++;
            i = (i + 1) % ds;
            if (cnt == bi) // 找满32位就输出
            {
                printf("%d ", num);
                cnt = 0; // 重置cnt和num
                num = 0;
            }
        } while (i % ds != start);
        printf("\n");
    }

    return 0;
}
```