C3-2021级航类-第三次上机题解

A Ct

示例代码

```
#include <stdio.h>

int main()
{
    int n1, o1, n2, o2;
    scanf("%d %d %d %d", &n1, &o1, &n2, &o2);
    if (n1 >= 35 && o1 >= 35 && n2 >= 35 && o2 >= 35)
        printf("True, but the TA would like you to copy this paragraph instead of
typing it by hand.");
    else
        printf("False");
    printf("\n%d", (n1 + o1 + n2 + o2) / 4);
    return 0;
}
```

B 求Fibo数列的第n项

问题分析

利用取模运算的性质和递推公式即可解决

参考代码1

用数组辅助

```
#include <stdio.h>
long long fib[100010] = {0};
int main()
{
    fib[0] = 1;
    fib[1] = 1;
    int i, n;
    scanf("%d", &n);
    if (n >= 2)
    {
        for (i = 2; i <= n; i++)
        {
            fib[i] = ((fib[i - 1] % 100000000711) + (fib[i - 2] % 100000000711)) %
100000000711;
        }
    }
    printf("%lld\n", fib[n]);
    return 0;
}
```

参考代码2

循环迭代法

```
// AUTHOR: 肖文磊老师
#include <stdio.h>

int main()
{
    int n;
    int a1, a2, an;
    scanf("%d", &n);
    a1 = 1;
    a2 = 1;

    if (n == 0 || n == 1)
    {
        an = 1;
    }
    else
    {
        while (--n)
        {
            an = (a1 + a2) % 1000000007;
            a1 = a2;
            a2 = an;
        }
    }
    printf("%d\n", an);
    return 0;
}
```

C 三角形形状判断

问题分析

首先筛选出最大的边，将其换到a边。然后利用三角形b^2+c^2与a^2的关系，判断三角形的最大角的形状，接着再用三条边的长度判断等腰性。

参考代码

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

int main()
{
```

```c
    int a = 0, b = 0, c = 0, tmp = 0;
    scanf("%d %d %d", &a, &b, &c);
    if (a < b)
    {
        tmp = a;
        a = b;
        b = tmp;
    }
    if (a < c)
    {
        tmp = a;
        a = c;
        c = tmp;
    }

    if (a >= b + c)
    {
        printf("no triangle\n");
    }
    else
    {
        if (a * a > b * b + c * c)
        {
            printf("obtuse triangle\n");
        }
        else if (a * a == b * b + c * c)
        {
            printf("right triangle\n");
        }
        else
        {
            printf("acute triangle\n");
        }

        if (a == b && b == c)
        {
            printf("equilateral triangle");
        }
        else if (a == b || b == c || a == c)
        {
            printf("isosceles triangle");
        }
    }

    return 0;
}
```

D LJF坐地铁

问题分析

三重循环，分别遍历5元钞票、2元钞票和1元钞票的可能值，然后判断方案是否可行再输出即可

参考代码

```
#include <stdio.h>
int n;
int main()
{
    scanf("%d", &n);
    for (int i = 0; i * 5 <= n; i++) //5元钞票张数
    {
        for (int j = 0; j * 2 + i * 5 <= n; j++) //2元钞票张数
        {
            for (int k = 0; k + j * 2 + i * 5 <= n; k++) //1元钞票张数
                if (k + j * 2 + i * 5 == n)
                    printf("%d %d %d\n", i, j, k);
        }
    }
}
```

E 胆密欧&傅里叶

题目分析

$$S_m(x) = \frac{\pi}{2} + \frac{2}{\pi}\sum_{n=1}^{m}\frac{(-1)^n - 1}{n^2}cosnx$$

从1到m循环模拟这个公式的求解过程即可。

参考代码

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define ll long long
int main()
{
    int m;
    double x,ans,pi=acos(-1);
    scanf("%d%lf",&m,&x);
    for(int i=1;i<=m;i++)
```

```
    {
        if(i%2) //只有i为奇数时，该项不为零
            ans+=-2.0/(i*i)*cos(i*x); //求和公式中的第i项值
    }
    ans=pi/2+2*ans/pi; //外层公式
    printf("%.4lf\n",ans); //四位小数输出
    // system("pause");
    return 0;
}
```

F 普通的GCD和LCM

题目分析

特判一下a,b,c有2个或3个0的情况，这个时候输出：

```
0h that 1s t00 bad!
0
```

其余的情况:

最大公约数(gcd):

$$gcd_1 = gcd(a,b)$$

$$gcd(a,b,c) = gcd(gcd_1,c)$$

最小公倍数(lcm)：

$$lcm_1 = lcm(a,b) = \frac{a \times b}{gcd(a,b)} = \frac{a \times b}{gcd_1}$$

$$gcd_2 = gcd(lcm_1,c)$$

$$lcm(a,b,c) = \frac{lcm(a,b) \times c}{gcd(lcm(a,b),c)} = \frac{lcm_1 \times c}{gcd_2}$$

求a,b的最大公约数用辗转相除法：

```
int r, gcd;
for(r = a%b; r != 0; r = a%b) {
    a = b;
    b = r;
}
gcd = b;
```

示例代码

```
#include <stdio.h>
int main()
{
    long long a, b, c;
    scanf("%lld%lld%lld", &a, &b, &c);

    if (a * b == 0 && b * c == 0 && a * c == 0)
    {
        puts("0h that 1s t00 bad!");
        puts("0");
        return 0;
    }

    long long GCD1, GCD2, GCD, LCM1, LCM;
    long long a1, b1, r;
    a1 = a;
    b1 = b;
    for (r = a1 % b1; r != 0; r = a1 % b1)
    {
        a1 = b1;
        b1 = r;
    }
    GCD1 = b1;

    a1 = GCD1;
    b1 = c;
    for (r = a1 % b1; r != 0; r = a1 % b1)
    {
        a1 = b1;
        b1 = r;
    }
    GCD = b1;

    LCM1 = a * b / GCD1;

    a1 = LCM1;
    b1 = c;
    for (r = a1 % b1; r != 0; r = a1 % b1)
    {
        a1 = b1;
        b1 = r;
    }
    GCD2 = b1;

    LCM = LCM1 * c / GCD2;

    printf("%lld\n%lld", GCD, LCM);
```

```
    return 0;
}
```

G X同学的日期统计

题目分析

我们使用一个数组cnt来存储数据，其中cnt[i]表示在星期i的特殊日期的总数是多少（**其中cnt[0]表示在星期日的特殊日期总数**），在输出的时候注意判断cnt[i]是不是0。

对于一个具体的日期（格式为yyyymmdd），我们使用Zeller公式计算出这个日期是星期几。由于Zeller公式在课上就讲过，题面中也对变量进行了解释，所以这里只提醒几个可能的易错点。

先列出Zeller公式如下：

$$W = (\lfloor \frac{C}{4} \rfloor - 2C + Y + \lfloor \frac{Y}{4} \rfloor + \lfloor \frac{26(M+1)}{10} \rfloor + D - 1) \bmod 7$$

1. 公式M和Y的计算：**某年的1、2月要看成上一年的13、14月**，此时**年份需要减一**。由于会出现2000年1月1日（相当于1999年13月1日），公式中C和Y的计算应该放到M和D之后。
2. W的最后结果：必须保证W是非负数

示例代码

```
#include <stdio.h>

int main()
{
    int cnt[10], i; // cnt[i]表示在星期i的特殊日期总数

    //不要忘了对局部变量初始化
    for (i = 0; i < 10; i++)
    {
        cnt[i] = 0;
    }

    int n; // n表示日期总数
    scanf("%d", &n);

    while (n--)
    {
        int day;
        scanf("%d", &day); // day代表yyyymmdd

        //以下是Zeller公式的部分
        int c, y, m, d, w;
```

```
        y = day / 10000;
        m = (day % 10000) / 100;
        d = day % 100;

        //计算公式中的M
        if (m < 3)
        {
            y--;
            m += 12;
        }

        //切记在计算完M之后再计算C和Y
        c = y / 100;
        y = y % 100;
        w = (y + y / 4 + c / 4 - 2 * c + (26 * (m + 1)) / 10 + d - 1) % 7; //代公式
        if (w < 0)
            w += 7; //保证w非负
        cnt[w]++;   //记录w是星期几
    }

    for (int i = 0; i < 7; i++)
    {
        if (cnt[i] != 0)
        {
            printf("%d %d\n", i, cnt[i]); // cnt[i]==0的话就不输出了
        }
    }

    return 0;
}
```

H zhn の 数论（一）

问题分析

1. 我们首先定义如下数组：prime[i]，含义是i这个数是否为素数，如果是素数，那么数组值为1，否则为0。
2. 之后我们预处理prime[]这个数组，在输入n之后，我们把小于等于n的素数全都预处理出来，这里要注意时间复复杂度，预期是n*\sqrt{n}的时间复杂度，n^2的大概一定会被我卡掉（
3. 本题要的输出要求**乘积最小**，由高中数学知识可以知道，两个正数，和一定，差的越大，乘积越小。（如果不会的话自己写个函数算一下就知道了）所以我们在输出的时候，我们从小到大枚举i，找到第一组i与n-i的都是素数的时候就可以输出，跳出

参考代码

```
#include <stdio.h>
#include <math.h>
int prime[100007];
int n;
int main()
{
    scanf("%d", &n);
    // prime[i]=1,代表这个数是素数
    prime[1] = 0;
    prime[2] = 1;
    for (int i = 3; i <= n; i++)
    {
        int flag = 0;
        for (int j = 2; j <= sqrt(i); j++)
        {
            if (i % j == 0)
            {
                flag = 1;
                break;
            }
        }
        if (!flag)
            prime[i] = 1;
    }
    for (int i = 2; i <= n; i++)
    {
        if (prime[i] && prime[n - i])
        {
            printf("%d %d", i, n - i);
            break;
        }
    }
    return 0;
}
```

I 图图玩狼人杀

题目分析

由于题目限定了只有两个坏人，因此我们可以通过**两重循环**枚举出所有狼人的情况，题目难点或是易错点在于**“已知图图是好人，且图图说的是真话”**，我们可以先把这一已知条件拿掉，分步完成这道题，代码如下：

```
/* 这是去掉已知条件“图图是好人，且图图说的是真话”后的解答方法 */
#include <stdio.h>
```

```c
int main() {
    /*
        say[i]表示编号为 i 的玩家说的话；
        realIdentity[i]表示编号为 i 的玩家的真实身份，用 1 表示好人，-1 表示狼人；
        numOfTruth 表示所有玩家中真话有多少，其值为 n - m；
        cnt 表示在一次枚举中所有玩家的真话数，其值为 numOfTruth 时代表此次枚举成功，找到两名狼人；
        found 表示是否有解，初始假设无解，值为 0；
    */
    int n, m, idTutu, i, j, k, say[20], realIdentity[20], numOfTruth, cnt, found = 0;
    scanf("%d%d%d", &n, &m, &idTutu);
    numOfTruth = n - m;

    for (i = 1; i <= n; i++)
        realIdentity[i] = 1;    // 初始认为每位玩家身份都是好人

    for (i = 1; i <= n; i++)
        scanf("%d", &say[i]);   // 读入每位玩家说的话

    for (i = 1; i <= n; i++) {
        realIdentity[i] = -1;           // 开始枚举，假设编号为 i 的玩家是狼人
        for (j = i + 1; j <= n; j++) {
            realIdentity[j] = -1;       // 假设编号为 j 的玩家是狼人
            cnt = 0;                    // 置 cnt 值为 0
            for (k = 1; k <= n; k++) {  // 检验此次枚举有多少玩家说真话
                if (say[k] > 0) {
                    cnt += realIdentity[say[k]] == 1;
                } else {
                    cnt += realIdentity[-say[k]] == -1;
                }
            }
            if (cnt == numOfTruth) {    // 检验此次枚举是否符合条件
                if (!found)
                    printf("Thank God I know!\n");
                found = 1;
                printf("%d %d\n", i, j);
            }
            realIdentity[j] = 1;        // 本次检验完毕后，重新置 realIdentity[j] 为 1
        }
        realIdentity[i] = 1;            // 本轮检验完毕后，重新置 realIdentity[i] 为 1
    }

    if (found == 0)
        printf("Still can't find.\n");

    return 0;
}
```

以上代码并不难理解，现将"图图是好人，且图图说的是真话"条件加上，对于"图图是好人"这个条件，较容易处理，在枚举狼人时若编号为图图的编号则 $continue$ 即可；对于"图图说的是真话"这个条件，要分情况讨论，如果图图指认了一个好人，那么同理在枚举狼人时该编号 $cointinue$，如果图图指认一个坏人，那么只需要一重循环，找到另一名狼人即可，同时每次枚举时 cnt 置 1，在检验时不必再检验图图说的话是真是假。故完整代码如下：

示例代码

```
#include <stdio.h>

int main() {
    int n, m, idTutu, i, j, k, say[20], realIdentity[20], numOfTruth, cnt, found = 0;
    scanf("%d%d%d", &n, &m, &idTutu);
    numOfTruth = n - m;

    for (i = 1; i <= n; i++)
        realIdentity[i] = 1;

    for (i = 1; i <= n; i++)
        scanf("%d", &say[i]);

    if (say[idTutu] < 0) {      // 图图指认了一个狼人的情况
        realIdentity[-say[idTutu]] = -1;    // 先标记该编号玩家为狼人
        for (i = 1; i <= n; i++) {
            if (i == idTutu || i == -say[idTutu])   // 不检验 idTutu 和 -say[idTutu] 两个
编号
                continue;
            realIdentity[i] = -1;
            cnt = 1;                        // 真话数初始为1
            for (k = 1; k <= n; k++) {
                if (k == idTutu)            // 不必检验idTutu
                    continue;
                if (say[k] > 0) {
                    cnt += realIdentity[say[k]] == 1;
                } else {
                    cnt += realIdentity[-say[k]] == -1;
                }
            }
            if (cnt == numOfTruth) {
                if (!found)
                    printf("Thank God I know!\n");
                found = 1;
                if (i < -say[idTutu])       // 按从小到大输出
                    printf("%d %d\n", i, -say[idTutu]);
                else
                    printf("%d %d\n", -say[idTutu], i);
            }
            realIdentity[i] = 1;
        }
```

```
    } else {    // 图图指认了一个好人的情况
        for (i = 1; i <= n; i++) {
            if (i == idTutu || i == say[idTutu])    // 不检验 idTutu 和 say[idTutu] 两个编
号
                continue;
            realIdentity[i] = -1;
            for (j = i + 1; j <= n; j++) {
                if (j == idTutu || j == say[idTutu])
                    continue;
                realIdentity[j] = -1;
                cnt = 1;
                for (k = 1; k <= n; k++) {
                    if (k == idTutu)
                        continue;
                    if (say[k] > 0) {
                        cnt += realIdentity[say[k]] == 1;
                    } else {
                        cnt += realIdentity[-say[k]] == -1;
                    }
                }
                if (cnt == numOfTruth) {
                    if (!found)
                        printf("Thank God I know!\n");
                    found = 1;
                    printf("%d %d\n", i, j);
                }
                realIdentity[j] = 1;
            }
            realIdentity[i] = 1;
        }
    }

    if (!found)
        printf("Still can't find.\n");

    return 0;
}
```

J LJF算复数

题目分析

本题考察了格式较为复杂的输入输出，switch-case语句，多重情况的判断与讨论，浮点数大小比较，浮点数是否为整数的判断。基本不涉及数据范围与精度问题。

示例代码

```
#include <stdio.h>

#include <math.h>

int main()
{

    double a, b, c, d, e, f;

    char sym;

    while (scanf("(%lf%lfi)%c(%lf%lfi)", &a, &b, &sym, &c, &d) != EOF)
    {

        //吃掉每行结尾多余的\n等字符，防止读取下一行开头时卡住
        char ctemp = getchar();
        while (ctemp != '\n' && ctemp != EOF)
        {
            ctemp = getchar();
        }
        //基本switch-case语句
        switch (sym)
        {
        case '+':
            e = a + c;
            f = b + d;
            break;
        case '-':
            e = a - c;
            f = b - d;
            break;
        case '*':
            e = a * c - b * d;
            f = b * c + a * d;
            break;
        case '/':
            e = (a * c + b * d) / (c * c + d * d);
            f = (b * c - a * d) / (c * c + d * d);
            break;
        }
        //以下涉及浮点数大小比较，浮点数是否为整数的判断
        //如果实部为零
```

```
        if (fabs(e) < 1e-5)
        {
            //如果实部虚部都为零
            if (fabs(f) < 1e-5)
            {
                printf("0");
            }
            //虚部不为零 只输出虚部，没有前导"+"号
            //虚部为1 则只输出i
            else if (fabs(f - 1) < 1e-5)
            {
                printf("i");
            }
            //虚部为-1 则只输出-i
            else if (fabs(f + 1) < 1e-5)
            {
                printf("-i");
            }
            //虚部为整数，输出整数，没有前导"+"号
            else if (fabs(round(f) - f) < 1e-5)
            {
                printf("%.0fi", f);
            }
            //虚部有小数部分，保留两位小数，没有前导"+"号
            else
            {
                printf("%.2fi", f);
            }
        }
        //如果实部不为零
        else
        {
            //正常输出实部
            if (fabs(round(e) - e) < 1e-5)
            {
                printf("%.0f", e);
            }
            else
            {
                printf("%.2f", e);
            }
            //输出虚部 有前导"+"-"号
            //虚部为零 不输出
            if (fabs(f) < 1e-5)
            {
            }
            //虚部为1 则只输出+i
            else if (fabs(f - 1) < 1e-5)
            {
```

```
                printf("+i");
            }
            //虚部为-1 则只输出-i
            else if (fabs(f + 1) < 1e-5)
            {
                printf("-i");
            }
            //以下对虚部为整数小数进行讨论，注意有前导+号
            else if (fabs(round(f) - f) < 1e-5)
            {
                printf("%+.0fi", f);
            }
            else
            {
                printf("%+.2fi", f);
            }
        }
        printf("\n");
    }
    return 0;
}
```

对输入部分的一些讨论

本题 `scanf` 部分最简便的实现方式是以下这种：（由于已经提示了评测数据每行结尾一定包含 \n ）

```
while(scanf("(%lf%lfi)%c(%lf%lfi)\n",&a,&b,&sym,&c,&d)!=EOF){
}
```

但是以上语句可能出现，输入第二行后回车才返回上一行答案的情况，问题出在`scanf`函数结束输入的方式：**1 回车2遇到非法字符3指定宽度结束 例如%5s**。由于上面的方案每行输入匹配了`\n`，没有多余`\n`供`scanf`结束输入，程序就会等待直到下一行输入数据出现。**但这种方法最后输出数据是完整的，并不影响程序运行。**

更好的方式是这样：

```
while (scanf("(%lf%lfi)%c(%lf%lfi)", &a, &b, &sym, &c, &d) != EOF)
{
    char ctemp = getchar();
    while (ctemp != '\n' && ctemp != EOF)
    {
        ctemp = getchar();
    }
}
```

这样的方式可以让调试时更加优雅。

当然还有其他技术方案，例如用`gets()`读入一整行字符串，使用`sscanf()`函数解析，但由于大家目前还没有学习字符串知识，这里不做详细展开。

K cxccxc与谢尔宾斯基三角形

题目分析

通过题面可以看出，可以用递归直接模拟谢尔宾斯基三角形的构造过程，但是这种方法过于复杂。

下面给出一种较为简洁的做法，需要~~亿点点~~数学洞察力

我们考虑杨辉三角，即$n = 1, 2, 3, \ldots$的二项式系数从上到下排列，如下所示

```
          1
         1 1
        1 2 1
       1 3 3 1
      1 4 6 4 1
     ......
```

观察到如果对杨辉三角的每个数对2取余，会得到下面的模2杨辉三角

```
          1
         1 1
        1 0 1
       1 1 1 1
      1 0 0 0 1
     1 1 0 0 1 1
    1 0 1 0 1 0 1
   1 1 1 1 1 1 1 1
  ......
```

我们惊喜地发现，这与谢尔宾斯基三角形极其相似！说明二者间有着千丝万缕的联系！我们可以借助杨辉三角画出谢尔宾斯基三角形。

通过观察样例可以看出，对于n阶谢尔宾斯基三角形字符画，我们只需抽出模2杨辉三角前2^{n+1}行中的偶数层。$n = 2$的示例如下。

```
          1 1
        1 1 1 1
      1 1 0 0 1 1
    1 1 1 1 1 1 1 1
```

再将"1"的位置填上'/'，'_'或'\'，具体是哪个字符可通过观察样例找出规律。最后将"0"的位置填上空格，即可画出符合题目要求的谢尔宾斯基三角形。

```
          /\
         /__\
        /\  /\
       /__\/__\
```

参考代码

```
#include<stdio.h>
int n;
int a[2050],len=1,b[2050],x;
void iter(){
    for(int i=1;i<=len;i++) b[i]=a[i];
    for(int i=1;i<=len+1;i++) a[i]=(b[i]+b[i-1])&1;
    len++;
}
int main(){
    scanf("%d",&n);
    a[1]=1;
    iter();
    for(int i=1;i<=(1<<n);i++){
        for(int j=1;j<=(1<<n)-i;j++) printf(" ");
        for(int j=1;j<=len;j++){
            if(!a[j]){
                printf(" ");
                continue;
            }
            if(x==0) printf("/"),x=1;
            else if(j%4!=0&&a[j+1]) printf("_");
            else printf("\\"),x=0;
        }
        printf("\n");
        iter();
        iter();
    }
    return 0;
}
```

正确性证明

至于为什么模2杨辉三角和谢尔宾斯基三角形有着一样的图案，下面给出证明。该证明照搬参考了这个视频

我们从杨辉三角出发。由于是二项式系数，杨辉三角中的每一个数都可以用一个组合数来表示。为表示方便，我们以杨辉三角的顶角为原点，顶角指向左底角和右底角的方向分别为x轴和y轴，构建斜坐标系，则位于坐标(x,y)的数为C_{x+y}^{x}（或C_{x+y}^{y}，二者相等）

由于

$$C_{x+y}^{x}=\frac{(x+y)!}{x!y!}$$

另外，对$\forall N\in\mathbb{N}^*$进行质因数分解可得

$$N=2^{p_2(N)}3^{p_3(N)}5^{p_5(N)}\cdots$$

因此，

$$C_{x+y}^{x} \equiv 1\ (mod\ 2)$$

$$\Leftrightarrow p_2(\frac{(x+y)!}{x!y!}) = 0$$

$$\Leftrightarrow p_2((x+y)!) = p_2(x!) + p_2(y!)$$

到此仍然看不出x与y的关系，我们需要进一步简化

因为对于$\forall M \in \mathbb{N}^*$,

$$p_2(M!) = p_2(1) + p_2(2) + \cdots + p_2(M)$$

若将$\forall a \in \mathbb{N}^*$看成二进制数，则$p_2(a)$就是$a$中1的最低位数。例如$p_2((12)_{10}) = p_2((1100)_2) = 2$。

所以1到M中每个数在二进制下1的最低位数之和即为$p_2(M!)$

对于这一问题，我们不妨换个角度思考。与其从1算到M，不如按位考虑，对于从低到高的每一位，统计不超过M的二进制数中这一位为0且更低位没有1的数的个数。

如$M = 6$时，二进制下的1-6为

```
001
010
011
100
101
110
```

我们可以按原方法计算出$p_2(6!) = 0 + 1 + 0 + 2 + 0 + 1 = 4$，更便捷的方法是按位计算，由$3 + 1 = 4$得出$p_2(6!)$。

显然，第$i(i = 0, 1, 2, 3, \dots)$位符合条件的数就是所有不超过M的二进制数中最后$i + 1$位全为0的数，这样的数共有$M >> (i + 1)$个，>>表示将二进制数右移。
由此，我们得到

$$p_2(M!) = \sum_{i=1}^{\infty}(M >> i)$$

不幸的是，这样的变换并没有让问题得到简化。但是，接下来我们观察发现，对于二进制数M，如果其第p位为1，那么这个1将分别在M右移的过程中对级数做出$2^{p-1}, 2^{p-2}, \dots, 1$的贡献，求和后算出总贡献为$2^p - 1$。如果$M$中有c(M)个1，分别在第$i_1, i_2, \dots, i_{c(M)}$位，则有

$$\sum_{i=1}^{\infty}(M >> i) = \sum_{k=1}^{c(M)}(2^{i_k} - 1)$$

而

$$M = \sum_{k=1}^{c(M)} 2^{i_k}$$

因此

$$\sum_{i=1}^{\infty}(M >> i) = M - c(M)$$

综上，

$$p_2((x+y)!) = p_2(x!) + p_2(y!) \Leftrightarrow x + y - c(x+y) = x - c(x) + y - c(y)$$

$$\Leftrightarrow c(x+y) = c(x) + c(y)$$

当二进制下的x与y任意一位都不同时为1，即$x+y$不会发生进位时，$c(x+y) = c(x) + c(y)$始终成立。如果有某一位同时为1，则原来分别在x和y里的两个1变成了进位后的一个1，此时有$c(x+y) < c(x) + c(y)$

至此，我们得到了模2杨辉三角某点为1的充要条件是**该点x，y坐标在二进制下每一位均不能同时为1**。

这些点构成的集合为

$$S = \{\forall i \in \mathbb{N}, \alpha_i, \beta_i \in \{0,1\}, \alpha_i + \beta_i < 2 | (\sum_{i=0}^{\infty}\alpha_i 2^i, \sum_{i=0}^{\infty}\beta_i 2^i)\}$$

为便于后续分析，我们将$2^n (n \to \infty)$行杨辉三角压缩进边长为1的等边三角形，保持原点和x，y轴方向不变，则上面点集描述方式变为

$$S' = \{\forall i \in \mathbb{N}, \alpha_i, \beta_i \in \{0,1\}, \alpha_i + \beta_i < 2 | \lim_{n\to\infty}(\sum_{i=0}^{n-1}\frac{\alpha_i 2^i}{2^n}, \sum_{i=0}^{n-1}\frac{\beta_i 2^i}{2^n})\}$$

$$\Leftrightarrow \{\forall i \in \mathbb{N}, \alpha_i, \beta_i \in \{0,1\}, \alpha_i + \beta_i < 2 | \lim_{n\to\infty}(\sum_{i=0}^{n-1}\frac{\alpha_i}{2^{n-i}}, \sum_{i=0}^{n-1}\frac{\beta_i}{2^{n-i}})\}$$

$$\Leftrightarrow \{\forall i \in \mathbb{N}, \alpha_i, \beta_i \in \{0,1\}, \alpha_i + \beta_i < 2 | \lim_{n\to\infty}(\sum_{i=0}^{n-1}\frac{\alpha_{n-i}}{2^{n-i}}, \sum_{i=0}^{n-1}\frac{\beta_{n-i}}{2^{n-i}})\}$$

$$\Leftrightarrow \{\forall i \in \mathbb{N}, \alpha_i, \beta_i \in \{0,1\}, \alpha_i + \beta_i < 2 | \lim_{n\to\infty}(\sum_{i=1}^{n}\frac{\alpha_i}{2^i}, \sum_{i=1}^{n}\frac{\beta_i}{2^i})\}$$

$$\Leftrightarrow \{\forall i \in \mathbb{N}, \alpha_i, \beta_i \in \{0,1\}, \alpha_i + \beta_i < 2 | (\sum_{n=1}^{\infty}\frac{\alpha_n}{2^n}, \sum_{n=1}^{\infty}\frac{\beta_n}{2^n})\}$$

最后，我们看看这样的点集表示什么样的图案

对于$\forall P(x_P, y_P) \in S'$,

$$x_P = \frac{\alpha_1}{2} + \frac{\alpha_2}{4} + \frac{\alpha_3}{8} + \frac{\alpha_4}{16} + \cdots$$

$$y_P = \frac{\beta_1}{2} + \frac{\beta_2}{4} + \frac{\beta_3}{8} + \frac{\beta_4}{16} + \cdots$$

所以

$$\vec{OP} = \frac{1}{2}\{(\alpha_1, \beta_1) + \frac{1}{2}\{(\alpha_2, \beta_2) + \frac{1}{2}\{(\alpha_3, \beta_3) + \frac{1}{2}\{(\alpha_4, \beta_4) + \cdots$$

当我们从$i = 1$开始分配α_i, β_i时，P点位置的确定等价于如下过程：

(1)由于α_1,β_1不能同时为1，第一步从原点出发只能选择沿x走$\frac{1}{2}$长度$(\alpha_1=1,\beta_1=0)$或沿y轴方向走$\frac{1}{2}$长度$(\alpha_1=0,\beta_1=1)$，或者原地不动$(\alpha_1=0,\beta_1=0)$

(2)将(1)可能到达的3个终点作为新的起点，分配α_2,β_2，同样的三选一，只不过这次的步长缩短为$\frac{1}{4}$

(3)将(2)可能到达的9个终点作为新的起点，分配α_3,β_3，再做三选一，这次的步长缩短为$\frac{1}{8}$

.....

我们发现，这样的“移动”有着自相似性。从第k+1步开始的移动等价于将起点分放在第k步后的3^k个终点，将步长缩小至原来的$\frac{1}{2}$，再像第k步一样选择并移动。

因此，若用当前等边三角形区域的三条中位线将其分割成四个全等的小三角形，则P点只会走到最上方的小三角形的三个顶点。

所以，从全局来看，初始边长为1的三角形四等分后最中间的边长为$\frac{1}{2}$小三角形永远不会有S'中的点。将视野放在其它三个小三角形，对其再次四等分后也是如此。因此，任何一步的等边三角形“视野”中，最中间的小三角形永远为空。

而对于“视野”中其它位置的点，将其x,y坐标转换为二进制后必然对应一种$(\alpha_i,\beta_i),i=1,2,3..$.的分配方案，因此总有该点属于$S'$。

我们知道，不断将等边三角形用中位线四等分，并移除最中间的小三角形正是谢尔宾斯基三角形的构造方法。

因此，$2^n(n\to\infty)$行模2杨辉三角与谢尔宾斯基三角形有着完全一致的图案。证毕。

递归解法

本题还可以用递归来求解（还没学，可以不了解），参考代码如下

```
#include <stdio.h>
#include <math.h>
char mem[3000][6000];
void draw(int x,int y,int n)
{
    if(n==1)
    {
        mem[x][y]=mem[x-1][y+1]='/';
        mem[x][y+1]=mem[x][y+2]='_';
        mem[x][y+3]=mem[x-1][y+2]='\\';
        return ;
    }
    int w=1<<n;
    int h=1<<(n-1);
    draw(x,y,n-1);
    draw(x,y+w,n-1);
    draw(x-h,y+w/2,n-1);
}
int main()
{
    int n;
    scanf("%d",&n);
```

```
    draw((1<<n)-1,0,n);
    for(int i=0;i<(1<<n);i++)
    {
        for(int j=0;j<(1<<(n+1));j++)
        if(mem[i][j])putchar(mem[i][j]);
        else putchar(' ');
        puts("");
    }
    return 0;
}
```