C5 - Solution

A 一般通过计算式

难度	考点
1	函数

题意分析

按照题意，我们需要引入 `math.h` 库并调用其中的库函数 `exp`，`cos`，`atan`，`log`，`cosh`来进行计算。可以得到如下示例代码。注意输入为不定行输入，输出保留三位小数。

示例代码

```
#include <stdio.h>
#include <math.h>
int main(void) {
    double x, y;
    while (scanf("%lf", &x) != EOF) {
        y = exp(cos(atan(x))) / log(cosh(x) + 1);
        printf("%.3f\n", y);
    }
    return 0;
}
```

B a^b Problem Ver.⑨

难度	考点
1	函数调用

问题分析

直接调用函数即可。

参考代码

```
#include <stdio.h>

long long qpow(long long a, unsigned long long b, long long p)
{
    long long ans = 1;
    a = a % p;
    while (b)
    {
        if (b & 1)
```

```
            ans = (ans * a) % p;
        b >>= 1;
        a = a * a % p;
    }
    return ans;
}
int main()
{
    long long a, p;
    unsigned long long b;
    scanf("%lld%llu%lld", &a, &b, &p);
    printf("%lld", qpow(a, b, p));
    return 0;
}
```

C 水獭函数

难度	考点
2	函数

题目分析

根据题意，我们可以先实现 `f(x)` ，再输出 `f(f(f(x)))` ；要特殊注意的就是在实现 `f(x)` 的过程中如果直接用 $x * x$ 可能会超出 int 范围。

同时这里补充一下 C 语言的取余运算，我们知道比如在同余的角度下，$4 \bmod 3 = 1$ 或者 $4 \bmod 3 = -2$ 都可以说是对的，那么 C 语言是如何规定取余操作的呢？可以尝试以下的代码片段：

```
#include <stdio.h>

int main()
{
    printf("4 %% 3 = %d\n",4 % 3);
    printf("-2 %% 3 = %d\n",(-2) % 3);
    printf("-5 %% 3 = %d\n",(-5) % 3);
    printf("(-2 + 3) %% 3 = %d\n",(-2+3) % 3);
    printf("(-5 + 3) %% 3 = %d\n",(-5+3) % 3);
    printf("(-2 %% 3 + 3) %% 3 = %d\n",(-2%3 + 3) % 3);
    printf("(-5 %% 3 + 3) %% 3 = %d\n",(-5%3 + 3) % 3);

    printf("\n");

    printf("4 %% -3 = %d\n",4 % -3);
    printf("-2 %% -3 = %d\n",(-2) % -3);
    printf("-5 %% -3 = %d\n",(-5) % -3);
    printf("(-2 + 3) %% -3 = %d\n",(-2+3) % -3);
    printf("(-5 + 3) %% -3 = %d\n",(-5+3) % -3);
    printf("(-2 %% -3 + 3) %% -3 = %d\n",(-2%3 + 3) % -3);
    printf("(-5 %% -3 + 3) %% -3 = %d\n",(-5%3 + 3) % -3);
    return 0;
}
```

在 C 语言中，余数的计算方法可以表示为 $a \bmod b = a - a/b * b$ 。而 C 语言中除法是趋零截尾的，也就是整数除法的结果相当于除法的结果向 0 的方向，将小数部分阶段，比如计算整数除法 $7/(-3)$ ，$\frac{7}{-3} = -2.33$ ，所以整数除法结果就是 -2 ，即直接截断了小数点后的部分。

所以余数和被除数的符号一致，也就是说余数并不总是和通常的取余运算结果相一致（通常数学上的取余操作 $a \bmod b$ 结果在 0 到 $b - 1$） ，所以在执行完 `ans = a % b` 后，为了使结果 ans 一定在 0 到 $b - 1$ 之间，我们用 `ans = (ans + b) % b`。这样，如果 ans 以前的值就是一个正数，那么 ans 的值不会受到影响；如果 ans 的值之前是一个负数，这么做就相当于把它转化成了同余意义下等价的正数，符合我们通常数学中的取余操作。

示例代码

```
#include <stdio.h>

int a,b;
int f(int x) {
    // 这段代码里的 mod b 有两个作用：防止结果超过 int 范围；保证余数的范围
    return (((a * x % b) * x + x) % b + b) % b;
}

int main()
{
    int x;
    scanf("%d%d",&a,&b);
    while(~scanf("%d",&x)){
        printf("%d\n",f(f(f(x))));
    }
    return 0;
}
```

当然，为了防止计算 `f(x)` 的计算过程中产生超出 int 的结果使用 long long 类型也是可以的；其次就是函数 `f(x)` 的实现可以写的更简单一些，大家可以思考一下

D 圆锥曲线和离心率

难度	考点
3	函数

题目分析

直接根据 Hint 及有关的数学公式完成函数即可，注意函数的返回值为 `double` 类型，以及本题所需的数组大小。由 Hint 可知，本题需要讨论 a, b 的大小关系，这些都需要在函数中予以体现。

示例代码

```
#include <stdio.h>
#include <math.h>
double eccentricity(int a, int b) {
    double e = 0;
    if (a > b) {
```

```
        e = sqrt(a * a - b * b) / a;
    } else if (a < b) {
        e = sqrt(b * b - a * a) / b;
    }
    return e;
}
int main() {
    int p[9];
    double e[6];
    for (int i = 1; i <= 8; i++) {
        scanf("%d", &p[i]);
    }
    e[1] = eccentricity(p[1], p[5]);
    e[2] = eccentricity(p[3], p[1]);
    e[3] = eccentricity(p[7], p[6]);
    e[4] = eccentricity(p[2], p[4]);
    e[5] = eccentricity(p[8], p[3]);
    for (int i = 1; i <= 5; i++) {
        printf("%.2f ", e[i]);
    }
    return 0;
}
```

E $Catalan$ 数

难度	考点
3	函数与递归

题目分析

题中已给出数列的递推公式，因此我们可以使用递归函数来进行运算。
根据题意我们可以得到以下递归函数

```
int Catalan(int n) {
    // 边界条件为 n == 0
    if (n == 0) {
        return 1;
    }else{
        // n != 0 的情况 按照题意进行乘法和累加 注意sum初始化为0
        int sum = 0;
        for (int i = 0; i <= n - 1; i++) {
            // 递归调用函数
            sum += (Catalan(i) * Catalan(n - 1 - i));
        }
        // 记得输出返回值
        return sum;
    }
}
```

示例代码

```
#include <stdio.h>
int Catalan(int n);
int main(void) {
    int T, a;
    scanf("%d", &T);
    for (int i = 0; i < T; i++) {
        scanf("%d", &a);
        printf("%d\n", Catalan(a));
    }
    return 0;
}
int Catalan(int n) {
    if (n == 0) {
        return 1;
    }else{
        int sum = 0;
        for (int i = 0; i <= n - 1; i++) {
            sum += (Catalan(i) * Catalan(n - 1 - i));
        }
        return sum;
    }
}
```

F 孪生素数猜想

难度	考点
4	函数，判断素数

题目分析

本题与教材例5-4类似。

我们只需要使 p 从 l 开始遍历到 $r-2$，判断 p 和 $q=p+2$ 是否为素数，输出结果即可。

判断素数的函数是下方示例代码中的 `isPrime` 函数，若参数 x 是素数则返回 1，否则返回 0。

注意：判断素数时 i 只需要枚举到 $\sqrt{x}$ 即可，若枚举到 x 会超时；此外特别要注意 1 不是素数。

示例代码中主函数中的循环是从大于 l 的第一个奇数(`p = l | 1`)开始枚举的，从 l 开始枚举也没有问题。

示例代码

```
#include <stdio.h>
int isPrime(int x)
{
    if(x == 1) return 0;
    if(x % 2 == 0) return 0;
    for(int i = 3; i * i <= x; i += 2)
```

```c
    {
        if(x % i == 0) return 0;
    }
    return 1;
}
int main()
{
    int l, r;
    int n = 0;
    scanf("%d%d", &l, &r);
    //for(int p = l; p + 2 <= r; p++)
    for(int p = l | 1; p + 2 <= r; p += 2)
    {
        if(isPrime(p) && isPrime(p + 2))
        {
            printf("%d %d\n", p, p + 2);
            n++;
        }
    }
    printf("%d", n);
    return 0;
}
```

G 戎璎花的汉诺塔

难度	考点
4	递归，汉诺塔

题目分析

经典递归问题之汉诺塔——课本上就有源代码

让我们再回顾其思路

假设有一个 n 层的汉诺塔在左侧的柱子上，要将其移动到右侧的柱子，需要怎么移动最快？

首先，至少，我们需要将最大的，最底下的第 n 圆盘移动到最右侧的柱子上。

由于圆盘只能放在更大的圆盘上，因此为了将第 n 号圆盘移动到最右侧的柱子上，我们需要：

> 第一步，将前 $n-1$ 个圆盘组成的汉诺塔移动到中间的柱子上

> 第二步，将第 n 个圆盘移动到最右侧柱子上

> 第三步，将前 $n-1$ 个圆盘组成的汉诺塔移动到右侧柱子上。

第二步可以直接输出，接下来需要处理的就是第一步和第三步，注意到第 n 层圆盘的存在完全不会影响前 $n-1$ 层圆盘的移动，那么第一步和第三步本质就是 $n-1$ 层的汉诺塔的问题。

那么这 $n-1$ 层的汉诺塔问题又可以再一次化归成 $n-2$ 层的汉诺塔，化归成 $n-3$ 层的汉诺塔......直到变成最基本的情况：只有一个圆盘，直接移动即可。

由此我们就得到了递归关系和初始状态，就能解决整个问题了。

另外，其实输出的字符串存在错别字（毕竟璎花只是个夭折的不识字的水子），希望大伙可以复制题目给出的输入/输出而不是手敲一遍，又慢又容易打错

示例代码

```c
#include <stdio.h>
void hanoi(int n, char from, char via, char to);
void move(int n, char from, char to);
int main(void) {
    int n;
    char from, via, to;
    scanf(" %c %c %c %d", &from, &via, &to, &n);
    hanoi(n, from, via, to);
    return 0;
}
void hanoi(int n, char from, char via, char to) {
    // 将n层的汉诺塔从from柱经过via柱移动到to柱上
    if (1 == n) {
        // 只有一层的初始状态，直接输出即可
        move(n, from, to);
        return;
    }
    hanoi(n - 1, from, to, via);
    // 第一步，将前n-1层组成的汉诺塔从from柱经过to柱移动到via柱上
    move(n, from, to);
    // 第二步，移动第n层圆盘至to柱上
    hanoi(n - 1, via, from, to);
    // 第三步，将前n-1层组成的汉诺塔从via柱经过from柱移动到to柱上
}
void move(int n, char from, char to) {
    printf("Eika moved Koishi %02d form the %c to the %c\n", n, from, to);
}
```

H 禁止抄袭!

难度	考点
5	递归输出

题目分析

此题与教材例5-17疫情期间的座位选择有相似的地方，但是除了过程有微小区别之外，最大的问题就是本题狠狠的限制了空间，如果使用书上的做法会 `MLE`，只能得 `0.4` 或 `0.6` 分。于是我们可以不用数组去模拟，而是直接递归输出的过程。

注意到对于**大多数情况**而言，输出的字符串可以看作由首尾两个 `1` 和中间 $n-2$ 个待输出字符组成的。而这 $n-2$ 个字符就是我们进行递归输出的主体，它被从中间划分后，可以看作三部分：左边一半待输出字符，中间的 `1`，右边一半待输出字符。再对左右两边的待输出字符进行同样处理，直到中间不再能放下 `1` 了，即这一段待输出字符的数量小于等于 2 时，输出对应数量的 `0`。

这题还需要考虑两个特殊情况，即 $n=1$ 和 $n=2$ 的情况，此时输出的字符串不再可以看作由首尾两个 `1` 和中间 $n-2$ 个待输出字符组成的，需要单独处理。

示例代码

```c
#include <stdio.h>
void f(int n)
{
    if (n == 1) // n == 1 和 n == 2 的时候是终止情况，不能继续向下递归
        printf("0");
    else if (n == 2)
        printf("00");
    else
    {
        f((n - 1) / 2); // 左半部分，与右半部分合起来共有n-1个字符
        printf("1");    // 中间的1
        f(n / 2);
    }
}
int main(void)
{
    int n;
    scanf("%d", &n);
    if (n == 1) // 两个特殊情况
        printf("1");
    else if (n == 2)
        printf("10");
    else
    {
        printf("1"); // 输出首位的1
        f(n - 2);    // 输出中间部分，共n-2个字符
        printf("1"); // 输出末位的1
    }
    return 0;
}
```

拓展

这题虽然和教材上的例题很相像，但这只是单纯的撞车。

这题真正的灵感来源是[【毕导】男同胞福音！如何解决尿尿时最尴尬的难题？建议偷偷收藏]，是一个很典型的使用递归思想解决问题的案例。

这题甚至一开始的题面都和视频里的一样，但是为了联系现状，提醒同学们不要抄袭 ~~（原题面没过审）~~ 就临时改了。

I dyc写代码

难度	考点
6	进制，函数

题目分析

先证明一个结论：$f(x)$ 的值等于 x 在三进制下每一位之和与位数的和。
使用数学归纳法证明，当 $x \le 3$ 时结论显然成立，假设对于所有的 $x < k$ 都成立，下考虑 $x = k$ 的情形，若 x 是 3 的倍数，此时由于 $f(x) = f(\frac{x}{3}) + 1$，由于 x 在三进制下相当于比 $\frac{x}{3}$ 添加了一个0，结论成立。若不是 3 的倍数，则相当于在最后一位加 1，结论依旧成立。
回到原题，我们要找到 l 与 r 之间最大的 x 使得 $f(x)$ 最大，当 l 和 r 相差的足够小时，我们显然可以直接计算，下面讨论 $r - l > 3$ 的情况，分以下两种情况：

- 当 l 和 r 三进制下位数相同时，假设分别为 $(a_0a_1\cdots a_k)_3$ 和 $(b_0b_1\cdots b_k)_3$，假设从高位开始第一位不同的为第 m 位，可知 $0 \le m < k$，那么我们可以知道若 $b_i = 2(i \le m \le k)$ 最大的一定是 $(a_0a_1\cdots a_{m-1}333\cdots 3)_3$，否则一定是 $(a_0a_1\cdots a_{m-1}(b_m-1)33\cdots 3)_3$。
- 当 l 和 r 三进制下位数不同时，假设分别为 $(a_0a_1\cdots a_{k_1})_3$ 和 $(b_0b_1\cdots b_{k_2})_3$，分两段来考虑，考查跟 r 三进制下位数相同的最小的数 y，l 到 y 之间最大的答案显然为 $2(k_2 - 1)$，y 到 r 之间同上面一种情况的讨论。

示例代码

```
#include<stdio.h>
#define max(a,b) (((a) > (b)) ? (a) : (b))
int t,a[100],b[100];
long long l,r;
long long f(long long x)//计算f(x)的函数
{
    if(x == 0)
    return 1;
    if(x % 3 == 0)
    return f(x / 3) + 1;
    else
    return f(x - 1) + 1;
}
int main()
{
    scanf("%d",&t);
    while(t--)
    {
        scanf("%lld %lld",&l,&r);
        if(r - l <= 3)//当差值小于3的时候直接计算
        {
            long long ans = 0;
            for(long long i = l;i <= r;i++)
            ans = max(ans,f(i));
            printf("%lld\n",ans);
        }
        else
        {
            int l1 = 0,l2 = 0;
            long long ans = 0;
            for(int i = 0;i < 50;i++)
            a[i] = 0,b[i] = 0;
            while(l)//分别计算l和r三进制下的表达式
            a[l1] = l % 3,l = l / 3,l1++;
            while(r)
            b[l2] = r % 3,r = r / 3,l2++;
```

```
            if(l1 != l2)//如果位数不同
            {
                ans = 3 * (l2 - 1);//一种情形是l2-1个2
                a[l2 - 1] = 1;//另一种情形让l成为最小的l2位的数
                for(int i = 0;i < l2 - 1;i++)
                a[i] = 0;
            }
            int k,flag = 0;
            for(int i = 50;i >= 0;i--)
            {
                if(a[i] != b[i])//找到l和r不同的第一位
                {
                    k = i;
                    break;
                }
            }
            for(int i = k - 1;i >= 0;i--)//判断r后面是否都是2
            {
                if(b[i] != 2)
                flag = 1;
            }
            if(flag)//如果都不是2，相应赋值
            {
                b[k] = b[k] - 1;
                for(int i = k - 1;i >= 0;i--)
                b[i] = 2;
            }
            long long x = 0;
            for(int i = 50;i >= 0;i--)
            x = x * 3 + b[i];
            ans = max(ans,f(x));//计算最后的答案
            printf("%lld\n",ans);
        }
    }
    return 0;
}
```

J 哪吒的递归函数

难度	考点
6~7	二进制，求贡献

题目分析

本题暴力递归会超时。那如何计算呢?

设 x 的二进制表示为 $n_k n_{k-1} \cdots n_1 n_{0(2)}$ ，其中 $n_i \in \{0,1\}$ ，且一定有 $n_k = 1$ 。

当 $k = 0$ 时， $x = n_{0(2)}$ ，即 x 为 0 或 1 。 $f(0) = 1, f(1) = 2$ 。

当 $k>0$ 时，$\left\lfloor \dfrac{x}{2} \right\rfloor$ 的二进制表示为 $n_k n_{k-1} \cdots n_{1(2)}$ ，相当于去掉最低位或者右移；$x-2^{\lfloor \log_2 x \rfloor}$ 的二进制表示为 $n_{k'} \cdots n_1 n_{0(2)}$ ，其中 k' 是满足 $k'<k$ 且 $n_{k'}=1$ 的最大的自然数，相当于去掉最高位的 1 （和后面连续的 0 ）。有
$f(x)=f(n_k n_{k-1} \cdots n_1 n_{0(2)})=f(n_k n_{k-1} \cdots n_{1(2)})+f(n_{k'} \cdots n_1 n_{0(2)})$ 。

可见递归过程并没有对 x 的二进制表示的每一位进行修改，只是去掉了其中若干项而已。

也就是说，一定有 $f(x)=a_k f(n_k)+a_{k-1} f(n_{k-1})+\cdots+a_0 f(n_0)=\sum_{i=0}^{k} a_i f(n_i)$ ，其中 a_i 是每一位的贡献。

例如 $10=n_3 n_2 n_1 n_{0(2)}=1010_{(2)}$

$$
\begin{aligned}
f(10)=f(1010_{(2)})&=f(101_{(2)})+f(10_{(2)})\\
&=f(10_{(2)})+f(1_{(2)})+f(1_{(2)})+f(0_{(2)})\\
&=f(1_{(2)})+f(0_{(2)})+f(1_{(2)})+f(1_{(2)})+f(0_{(2)})
\end{aligned}
$$

写作 n_i 的形式就是：

$$
\begin{aligned}
f(10)=f(n_3 n_2 n_1 n_{0(2)})&=f(n_3 n_2 n_{1(2)})+f(n_1 n_{0(2)})\\
&=f(n_3 n_{2(2)})+f(n_{1(2)})+f(n_{1(2)})+f(n_{0(2)})\\
&=f(n_{3(2)})+f(n_{2(2)})+f(n_{1(2)})+f(n_{1(2)})+f(n_{0(2)})\\
&=f(n_3)+f(n_2)+2f(n_1)+f(n_0)
\end{aligned}
$$

即 $a_3=1, a_2=1, a_1=2, a_0=1$ ，代入 $f(0)=1, f(1)=2$ 得 $f(10)=8$ 。

问题就转化为了如何求每一位的贡献 a_i 。

每个基本情况 $f(n_i)$ ，都是通过对 $x=n_k n_{k-1} \cdots n_1 n_{0(2)}$ 进行若干次右移操作和若干次去掉最高位的 1 （和后面连续的 0 ）操作得到的。也就是说 n_i 的贡献 a_i 就是通过这两种操作将 $x=n_k n_{k-1} \cdots n_1 n_{0(2)}$ 变为 n_i 的方法数。

如 $x=10=1010_{(2)}$ ，对于 n_2 ，只有“右移两次，去掉最高位”这一种方法能将 x 变为 n_2 ，因此 $a_2=1$ ；对于 n_1 ，有“去掉最高位，右移”和“右移，去掉最高位”两种方法使得 x 变为 n_1 ，因此 $a_1=2$ 。

对于 n_i 而言，想要递归到基本情况，显然右移操作一共需要进行 i 次；去掉最高位的操作需要进行几次呢？

$n_i=1$ 时，设满足 $i \le j \le k, n_j=1$ 的 j 的个数为 cnt_i ，即 n_i 和更高位中 1 的数量，显然需要进行 cnt_i-1 次“去掉最高位”的操作，且两种操作无顺序要求，因此方法数为 $\binom{cnt_i-1+i}{i}=\dfrac{(cnt_i-1+i)!}{i! \cdot (cnt_i-1)!}$ （组合数）。

$n_i=0$ 时，设满足 $i \le j \le k, n_j=1$ 的 j 的个数为 cnt_i ，即更高位中 1 的数量，显然需要进行 cnt_i 次“去掉最高位”的操作，但是最后一次操作一定是“去掉最高位”而不能是“右移”，（如 $x=10$ ，n_2 的情况），因此只有 cnt_i-1 次“去掉最高位”的操作顺序是可以变的，因此方法数也为 $\binom{cnt_i-1+i}{i}=\dfrac{(cnt_i-1+i)!}{i! \cdot (cnt_i-1)!}$ 。

即 $a_i=\binom{cnt_i-1+i}{i}=\dfrac{(cnt_i-1+i)!}{i! \cdot (cnt_i-1)!}$ 。

所以最终的答案为
$f(x)=a_k f(n_k)+a_{k-1} f(n_{k-1})+\cdots+a_0 f(n_0)=\sum_{i=0}^{k} a_i f(n_i)=\sum_{i=0}^{k} \binom{cnt_i-1+i}{i} f(n_i)$ 。

a_i 可通过递推求得，已知 a_{i+1} ，求 a_i 时，若 $n_i = 0$ ， $cnt_i = cnt_{i+1}$ ，则 $a_i = \dfrac{(i+1) \cdot a_{i+1}}{cnt_i + i}$ ；若 $n_i = 1$ ， $cnt_i = cnt_{i+1} + 1$ ，则 $a_i = \dfrac{(i+1) \cdot a_{i+1}}{cnt_{i+1}}$ ；特别地，当 $i = k$ 时， $cnt_i = 1$ ， $a_i = 1$ 。

从高到低遍历 x 的每一位，按公式计算 a_i ，乘上 $f(0) = 1$ 或 $f(1) = 2$ ，再求和即可得到答案。

示例代码

```
#include <stdio.h>

int main()
{
    unsigned long long x;
    while(~scanf("%llu", &x))
    {
        if(x == 0)
        {
            puts("1");
            continue;
        }
        unsigned long long ans = 0;
        int i, cnt = 0;
        for(i = 0; x >> i; ++i); //求出x二进制表示有几位，此时i=k+1
        unsigned long long a = 1;
        for(--i; i >= 0; --i)
        {
            if(x >> i & 1) //ni=1
            {
                if(cnt != 0) a = a * (i + 1) / cnt; //i!=k时
                cnt++; //更新cnt
            }
            else //ni=0
                a = a * (i + 1) / (cnt + i);
            ans += x >> i & 1 ? 2 * a : a;
        }
        printf("%llu\n", ans);
    }
    return 0;
}
```

补充：WA代码

0.1 分代码 - 暴力递归求解

```
#include<stdio.h>
int g(unsigned long long x)
{
    int cnt = 0;
    while(x)
    {
        ++cnt;
        x >>= 1;
    }
```

```
    return cnt;
}
unsigned long long f(unsigned long long x)
{
    if(x == 0) return 1ull;
    return f(x >> 1) + f(x - (1ull << g(x) - 1));
}
int main()
{
    unsigned long long x;
    while(~scanf("%llu", &x))
        printf("%llu\n", f(x));
    return 0;
}
```

0.4 分代码 - 记忆化存储递归求解

设 $x = n_k n_{k-1} \cdots n_1 n_{0(2)}$，
$f(x) = f(n_k n_{k-1} \cdots n_1 n_{0(2)}) = f(n_k n_{k-1} \cdots n_{1(2)}) + f(n_{k'} \cdots n_1 n_{0(2)})$，其中 k' 是满足 $k' < k$ 且 $n_{k'} = 1$ 的最大的自然数。

用 $A[i][j]$ 存储 $f(n_j n_{j-1} \cdots n_{i(2)})$，递归求解。

```
#include <stdio.h>
unsigned long long f(unsigned long long x, int i, int j, unsigned long long A[64]
[64])
{
    if(A[i][j]) return A[i][j]; //如果A[i][j]不为0，说明计算过了，直接返回
    if(i == j) return A[i][j] = x >> i & 1 ? 2 : 1; //基本情况
    int k;
    for(k = j - 1; k > i && !(x >> k & 1); --k); //求出k'
    return A[i][j] = f(x, i + 1, j, A) + f(x, i, k, A); //递归求解，同时用A[i][j]存
储答案
}
int main()
{
    unsigned long long x;
    while(~scanf("%llu", &x))
    {
        if(x == 0) //特殊情况
        {
            puts("1");
            continue;
        }
        unsigned long long A[64][64] = {0};
        int k;
        for(k = 0; x >> k; ++k);
        printf("%llu\n", f(x, 0, k - 1, A));
    }
    return 0;
}
```

Author: 哪吒

- End -