E5 - Solution

A - 三角形判断

难度	考点
1	条件判断、函数的调用

题目分析

按照题目要求，依次判断输入的三条边是否能构成等边三角形、等腰三角形、普通三角形即可。

示例代码

```
#include<stdio.h>

int triangle_type(int a,int b,int c){
    if(a==b && a==c) return 3;
    if(a==b || a==c || b==c) return 2;
    if(a+b>c && a+c>b && b+c>a) return 1;
    return 0;
}

int main(){
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    int d=triangle_type(a,b,c);
    switch (d){
        case 0: printf("not a triangle!"); break;
        case 1: printf("regular triangle!"); break;
        case 2: printf("isosceles triangle!"); break;
        case 3: printf("equilateral triangle!"); break;
    }
    return 0;
}
```

B - 广义斐波那契数列

难度	考点
2	递归、递推

题目分析

将斐波那契数列扩展得来的广义递推，类比斐波那契数列的递归过程进行分析，可以得到如下理论：

考虑我们当前递归到某一层，这一层任务是求F_n，会有三种情况：

1. 当$n == 1$时，由于$F_1 = p$是题目中给出的条件，所以$F_n = F_1 = p$，直接让当前递归层返回p即可；
2. 当$n == 2$时，由于$F_2 = q$是题目中给出的条件，所以$F_n = F_2 = q$，直接让当前递归层返回q即可；
3. 当n不满足上面两个条件时，说明$n \geq 3$，那么根据题目给出的公式，可以得知我们需要用到F_{n-1}, F_{n-2}的值，而在递归这一层我们不知道这两个值是多少，所以我们只能再调用下一层递归的返回值，也就是再递归计算F_{n-1}, F_{n-2}的值。

   通过调用递归我们现在知道了$F_{n-1}. F_{n-2}$的值，这样就能按照题目中的公式$F_n = A \times F_{n-1} + B \times F_{n-2} + C$计算即可，并返回$F_n$。

这就是一层递归的具体内容，具体实现可以参考下方代码。

注意开`long long`。

示例代码

```
#include<stdio.h>
long long p,q,A,B,C;
long long F(int n){
    if(n==1)
        return p;
    if(n==2)
        return q;
    return A*F(n-1)+B*F(n-2)+C;
}
int main(){
    int n;
    scanf("%d%lld%lld%lld%lld%lld",&n,&p,&q,&A,&B,&C);
    printf("%lld\n",F(n));
    return 0;
}
```

C - 「弹幕自由市场」

难度	考点
2	暴力枚举、一维数组、函数

题目分析

不难看出，如果要赚最多的钱，我们买入的时候必须买尽可能多的卡片。

法1: 简单的**枚举**买入的日期和卖出的日期。例如，总共有 n 天，我们不妨设在第 i 天买入，在第 j 天卖出（$1 \leq i \leq j \leq n$），其中，**不进行买入和卖出操作**与**在同一天买入和卖出**是等价的。时间复杂度 $O(n^2)$，足以AC本题。

法2: 利用**后缀**的思想。我们可以发现，如果在某一天买入后，能卖出最大价值一定对应之后 a_i **取到最大值**的时候。我们只需要用数组预处理下每一天到最后一天之间的最大值即可（具体实现见示例程序2）。时间复杂度 $O(n)$，比法1时间上更优。

注意到结果值可能超过int范围，所以要开`long long`。

示例代码

```cpp
#include <stdio.h>

int a[1005];
long long m;
long long max(long long x, long long y) { return x > y ? x : y; }
long long calc(int in, int out)
{
    return m / a[in] * a[out] + m % a[in];
}
int main()
{
    int n;
    long long ans = 0;
    scanf("%d%lld", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    for (int i = 1; i <= n; i++)
    {
        for (int j = i; j <= n; j++)
        {
            ans = max(ans, calc(i, j));
        }
    }
    printf("%lld", ans);
    return 0;
}
```

```cpp
// 本程序对应解法2
#include <stdio.h>

int a[1005];
int suc[1005];
long long max(long long x, long long y) { return x > y ? x : y; }
int main()
{
    int n;
    long long m, ans = 0;
    scanf("%d%lld", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    for (int i = n; i >= 1; i--)
        suc[i] = max(a[i], suc[i + 1]);
    for (int i = 1; i <= n; i++)
        ans = max(ans, (m / a[i]) * suc[i] + m % a[i]);
    printf("%lld", ans);
    return 0;
}
```

D - fff

难度	考点
3	函数，位运算

题目分析

先编写$f(x)$函数，然后直接调用$f(f(f(x)))$即可。

至于函数的具体实现，利用`(x>>i)&1`即可获取x从低到高第$i(i=0,1,\ldots,63)$个二进制位,统计这样的二进制位有多少个1即可。

注意数据范围，$2^{63}-1$是`long long`类型的最大值。

示例代码

```
#include<stdio.h>
int f(long long x)
{
    int result = 0;
    for(int i=0;i<64;i++)
    {
        if((x>>i)&1)//如果第i位是1
            result++;
    }
    return result;
}
int main()
{
    long long x;
    while(scanf("%lld",&x) != EOF)
    {
        printf("%d\n",f(f(f(x))));
    }
    return 0;
}
```

E - 三角形面积

难度	考点
2	函数调用

题目分析

可以用$S=\frac{a*b*sin(C)}{2}$来计算面积

利用勾股定理可以求出三条边的长度，从而求出周长（使用函数求边长能够简化代码）。

在得到三条边长以后，可以先求某角的cos值，再求sin值。

示例代码

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

double cal_len(double x[],double y[],int i1,int i2)
{
```

```
    double ret;
    ret=(x[i1]-x[i2])*(x[i1]-x[i2])+(y[i1]-y[i2])*(y[i1]-y[i2]);
    ret=sqrt(ret);
    return ret;
}

int main()
{
    int t;
    double x[5],y[5],l[5],perimeter,coss,area;
    scanf("%d",&t);
    while(t--)
    {
        perimeter=0;
        for(int i=0;i<3;i++)
            scanf("%lf%lf",&x[i],&y[i]);
        for(int i=0;i<3;i++)
        {
            l[i]=cal_len(x,y,i,(i+1)%3);//计算边长
            perimeter+=l[i];//求周长
        }
        coss=( l[0]*l[0] + l[1]*l[1] - l[2]*l[2] )/(2.0*l[0]*l[1] );//求cos()
        area= l[0]*l[1]*sqrt(1-coss*coss)/2.0;//求面积
        printf("%.3f %.3f\n",perimeter,area);
    }
    return 0;
}
```

F - Read Steiner

难度	考点
4	递归

题目分析

从两个角度分析递归怎样写：

1. 递归的层数，提示里说了课堂上的例子求阶乘：$f(n) = n \times f(n-1)$，实际上就是从第 n 层一直到第 0 层返回值然后递归回来。
   而本题的层数则是从第 0 层到第 n 层返回值，所以层数作为一个参数写进去。
2. 递归的表达式，提示和样例解释传达的想法就是下一层递归回来的值相加然后取模，用式子表达即：
   $f(i) = f_a(i+1) + f_b(i+1) + f_c(i+1)$
   当然这是对应的操作数 op=7 的情况，正常来说操作数二进制位会决定加上几项。如果非要用一个数学公式来表示，可以用0的次幂这个有趣的东西，因为只有$0^0 = 1$而其他的次幂都还是0，而二进制位只有$0, 1, 2$位，可以通过与$2^0, 2^1, 2^2$做与运算来实现。于是公式表达为：
   $f(i) = (1 - 0^{op\ \&\ 4}) \times f_a(i+1) + (1 - 0^{op\ \&\ 2}) \times f_b(i+1) + (1 - 0^{op\ \&\ 1}) \times f_c(i+1)$

于是递归部分的伪代码可以这样写：

```
f(i) {
    sa, sb, sc = 0, 0, 0
    if (op & 4) sa = a(i+1)
```

```
    if (op & 2) sb = b(i+1)
    if (op & 1) sc = c(i+1)
    return sa + sb + sc
}
a(i) {
    if (i == n) return A
    f(i)
}
b(i) {
    if (i == n) return B
    f(i)
}
c(i) {
    if (i == n) return C
    f(i)
}
```

这里也可以注意到出现了交叉的调用，即 `f()` 调用了 `a(),b(),c()` 而 `a(),b(),c()` 也调用了 `f()`，所以实际编写时需要注意先声明函数原型。

示例代码

```
#include <stdio.h>
#define bit2 1 << 0
#define bit1 1 << 1
#define bit0 1 << 2
#define MOD 1000000007
int n, a, b, c;
int rec[20];

int fa(int);
int fb(int);
int fc(int);
int recurs(int);
int fa(int i)
{
    if (i == n)
        return a;
    return recurs(i);
}

int fb(int i)
{
    if (i == n)
        return b;
    return recurs(i);
}

int fc(int i)
{
    if (i == n)
        return c;
    return recurs(i);
}
```

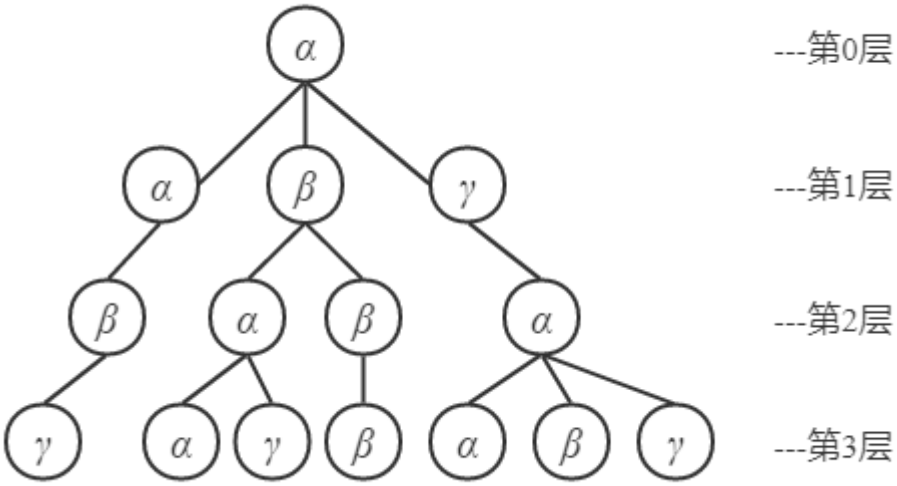
```
int recurs(int i)
{
    int sa = 0, sb = 0, sc = 0;
    if (rec[i] & bit0)
        sa = fa(i + 1) % MOD;
    if (rec[i] & bit1)
        sb = fb(i + 1) % MOD;
    if (rec[i] & bit2)
        sc = fc(i + 1) % MOD;
    return ((sa + sb) % MOD + sc) % MOD;
}
int main()
{
    scanf("%d%d%d%d", &n, &a, &b, &c);
    for (int i = 0; i < n; ++i)
        scanf("%d", &rec[i]);
    printf("%d\n", recurs(0));
    return 0;
}
```

关于对$10^9 + 7$取模的问题

因为这个数比较特殊：有`2*mod < INT_MAX < 3*mod`，也就是说两个数递归返回来相加都小于`mod`一定没问题不会溢出，但是三个数递归返回了相加就有可能溢出，所以要按照提示的方法前两个相加取模，再与第三个数相加再取模。

附录

从另外一个视角理解这个题的话，把伦太郎每次跳跃记录画图画出来，举个例子如图所示：



如果以后从数据结构的角度来看，就是一颗三叉树，实际上求的是所有叶节点也就是最下面一层不再有子节点的节点的值的和，顺便从树节点的角度理解一下本题的时间复杂度限制，考虑最坏的情况，每一次跳跃都会分出三个节点，最后形成一棵完全三叉树，节点数等比数列求和：$3^0 + 3^1 + \ldots + 3^{n-1} = \frac{3^n - 1}{2} = O(3^n)$，是非常恐怖的数量级，所以在OJ的1秒时限内，只能把n的范围压得很低。

G - 三点不共线

难度	考点
4	平面几何、递归

题目分析

这个题**无三线共点**真的节省了很大的工作量，因为我们可以判断，如果一条直线与n条直线不平行，那么就必会产生出**新的**n个交点

那么我们可以从$n = 5$时根据上述前提（三线不共点）推理一下

如果全部平行，交点数就为0

如果从中有一条叛变为不平行，那么交点数就为4

如果有两条叛变为不平行，那么当这两条直线平行的时候交点数为6，这两条直线不平行的时交点数为7

如果有三条叛变为不平行，那么当着三条直线平行的时候交点数为6，当两条直线平行另外一条不平行时交点数为8，当三条直线均不互相平行时交点数为9

如果有四条叛变为不平行，那么当四条直线平行的时候交点数仍为4，当三条平行一条不平行的时交点数为7，当两条平行两条不平行时交点数为9，当四条均不互相平行时交点数为10

所以可能的情况为$0, 4, 6, 7, 8, 9, 10$，共7种

我们可以从上述的过程分析出一个递推关系，因为每一次实际上我们是拿一条出来与其他的多条叛变直线的组合进行求交点数，而多条叛变的直线本身的组合数又可以再单拿出一条直线来求解更小的组合，所以这个题我们是可以通过递归的方式来解决的。

而递归的入口参数设置，我们可以分析出，由于每一种存在的平行线数目都会对应唯一一种与之对应的剩余直线组合，所以我们需要循环平行线的数目进行递归，而递归的入口需要的参数之一就是此时需要处理的直线数，这样我们每次循环取不同的平行线数目就保证递归的顺利进行了，并且递归的终止条件便为直线组合所含的直线数为0，但这个时候，我们仍然会存在一些问题，如果仅靠直线数向下传递，这时对应的交点数的重复情况我们无法筛除，所以需要将最终的交点数结果作为一个传递参数来不断更新，并引入一个数组记录当前的交点数是否出现过，这样便可达到去重的目的了。

所以，再结合上述的推理思路，我们可以推知：**对a条直线分情况讨论平行线的条数，已知在有i条平行线时有$(a-i)$条线与他们相交于$i \times (a-i)$个交点，再加上对于这$a-i$个直线的组合**

用公式写就是

$$ans(a) = i \times (a-i) + ans(a-i)$$

其中$ans(a)$表示的是直线组合与当前的平行线数目产生的交点数，并非一个独立的个体

因而从顶层写递归代码便为

```
递归（当前需要处理的直线数，上一层传递到当前状态的时候的已有交点数）{
    如果当前需要处理的直线数为0 {
        //进入结果判断
        如果当前计算出的交点数没有出现过，那么就设置为出现过了，并把结果数++
        否则当前对应的交点数无效（已经出现过了）
    }
    其他情况说明还没递归到终点 {
        //继续递归
        循环 从 当前处理的直线数全部都是平行的 直到仅有一条还处于原平行的状态 {
            递归（ 参数1: 当前需要处理的直线数-当前循环到的平行的直线数，
```

```
                    参数2: 当前平行的直线与不平行的直线产生的交点+上一层传到当前状态已有的交点数）
        }
    }
}
主函数调用   递归(n,0)
```

这个题按这个思路就可以AC啦~

示例代码

```
#include<stdio.h>
#include<stdbool.h>
int n,ans=0;
bool pd[10010];
void digui(int a,int b){
    if(a==0)
    {
        if(!pd[b])
            ans++;
        pd[b]=1;
    }
    else
        for(int i=a;i>=1;i--)
            digui(a-i,i*(a-i)+b);
}
int main(){
    scanf("%d",&n);
    digui(n,0);
    printf("%d",ans);
}
```

反思与改进

这个题这个做法并非一个正解

我们可以从上面的推理观察出，其实很多流程都是重复的，比如在四条叛变为不平行中两条平行两条不平行与三条叛变为不平行且这三条直线均不互相平行的两种情况实际上是完全重复的，还有很多其他情况就不一一列举了（在示例中结果数相同的就基本都是重复情况）

而这种重复情况并不是必然要出现的，是可以通过更优秀的算法（动态规划（就是dp））来避免重复工作，可以大大降低运算复杂度（因为递归过程中的函数调用本来就要比单一顺序块或循环块的执行速度要慢很多），通过大家的做题情况来看已经有很多同学选用这一方法，这也是非常不错的（但Sheep这里不能给大家放出来，因为猪脚们出dp的题目是要被罚鸡腿的）

另外有同学钻了数据量的空子，用数学计算的方法打表通过了这道题，但我希望这些同学下去要再练习一下这道题目的正规写法，对递归的掌握非常有帮助

说在最后的是，本题出在这里就是希望大家能练习一下递归的，所以才把数据量出的很低，但dp的做法要优秀很多很多，是可以支持很大的数据量的，这里递归纯粹是为了减少码量（奉上代码快逃）

H - 饿的幂次方

难度	考点
4	递归、位运算

题目分析

由题意可知，0 的 2 的幂次方表示为 0，而对于其他的正数 x，若 x 的二进制表示为 $000...0101$，则其表示为 2^2+2^0，显然两者存在一一对应关系。

当指数不为 0 时，需要将指数部分也表示为 2 的幂次方的形式，以此类推，直至指数为 0，所以该嵌套结构可以用递归来实现。

写递归函数重要的是找到递归出口，也就是什么时候停止递归。在本题中，显然当要表示的数为 0 时可以停止递归，直接输出 0 即可。

而对正数来说，遍历其二进制的每一位，如果为 1，则先输出底数 2 以及左括号，再对指数部分进行转化（调用递归函数），最后输出右括号。注意每一项之间还要输出加号。

示例代码

```
#include <stdio.h>

void printInt(int x)
{
    if (x == 0)
    {
        printf("0");
        return;
    }
    int flag = 0;
    for (int i = 31; i >= 0; i --)
        if ((x >> i) & 1)            //判断第i位是否为1
        {
            printf(flag ++ ? "+2(" : "2(");
            printInt(i);             //转化指数部分
            printf(")");
        }
}

int main()
{
    int n;
    scanf("%d", &n);
    printInt(n);
    return 0;
}
```

I - Monica的梦魇

难度	考点
5	函数嵌套

题目分析

题面基本给出了G函数的递归写法，只需要将其实现为程序代码可以。

求组合数使用了ppt上的递归写法，也可以写求阶乘最后取模的写法，不需要递归。

示例代码

```
#include <stdio.h>
#include <string.h>
#define ll long long
int p=13;
ll C(ll n,ll m)
{
    if(n<m)
        return 0;
    if(m==1)
        return n;
    if(n==m||m==0)
        return 1;
    return C(n-1,m)+C(n-1,m-1);
}
ll G(ll n,ll m)
{
    if(n<=p&&m<=p)
        return C(n,m)%p;
    else
        return C(n%p,m%p)*G(n/p,m/p)%p;
}
int main(){
    ll n,m;
    while(~scanf("%lld%lld",&n,&m))
        printf("%lld\n",G(n,m));
    return 0;
}
```

J - 黄金分割

难度	考点
7	高精度加法和除法

题目分析

如果看了提示，那么这个题思路应该就很清楚了，就是求出

$$\frac{F_n}{F_{n+1}}$$

其中F_i是斐波那契数列的第i项,n在1250左右。

但是F数列增长得是特别特别快的，到第100多项就超过了`long long`的范围，所以我们需要考虑用数组来模拟存储整数，并进行加法和乘法的操作。

接下来讲解一下代码每一部分的实现。

1. 定义大整数类型

```
#define maxn 10000+5
typedef struct {
    int number[maxn];//从0-lenth,是从低位到高位
} BIGNUM;
```

为了避免使用数组时后面产生大量的指针操作影响程序可读性，或者把所有模块都堆在main函数里影响整个程序的结构，这里用结构体定义了大整数类型变量`BIGNUM`，因为不涉及到负数的相关运算，所以只需要从低到高存储每一位就行了。结构体中只有一个`number[]`数组。

大家可能还没学结构体，这里进行一个简单的介绍：结构体（struct）是由一批数据组合而成的结构型数据。组成结构型数据的每个数据称为结构型数据的“成员”。其中成员可以是基本类型（`int`,`double`等）的数据、数组，也可以是已经定义过的结构体的数据，数组。其基本定义方式如下：

```
typedef struct{
    成员列表
} 类型的名字;
```

比如说我们要定义一个数据类型“日期”，那么我们可以用三个`int`型数据“年、月、日”表示如下：

```
typedef struct{
    int y,m,d;
} date;
```

这样就定义了一种新的数据类型为`date`，`date`型数据包含了三个`int`型数据，分别为`y,m,d`。

在使用时，我们可以直接

```
date a
```

声明一个`date`型的数据，其变量名为`a`。如果我们想要调用日期`a`的年份，那么我们可以写作：

```
a.y
```

同理，调用`a`的月份和日期时，也可以写作`a.m a.d`。

那么，如果我们想要输出a的年月日，只需要写作

```
printf("%d %d %d\n",a.y,a.m,a.d);
```

就行了。

2. 获取长度

```
int getLenth(BIGNUM a) {
    int i;
    for(i=(maxn-1); i>=0; --i) {
        if(a.number[i]>0 || i==0)
            break;
    }
    return i+1;
}
```

从高位到低位找到第一个非零的位数，那么这个位数就是整个数字的长度。需要注意的是，因为是从0开始的，所以需要给位数+1。也就是有效的位数是0 ~ getLenth(a)-1

3. 加法

```
BIGNUM BIGadd(BIGNUM a,BIGNUM b) { //都是正的加法
    int a_lenth=getLenth(a),b_lenth=getLenth(b);
    int maxLenth=a_lenth>b_lenth?a_lenth:b_lenth;
    BIGNUM ans;
    memset(ans.number,0,sizeof(a.number));//因为ans是局部变量，所以要赋初值。
    int flag=0;
    for(int i=0; i<maxLenth; ++i) {
        ans.number[i]=(flag+a.number[i]+b.number[i])%10;
        if(flag+a.number[i]+b.number[i]>=10)
            flag=1;
        else
            flag=0;
    }
    if(flag)
        ans.number[maxLenth]+=flag;
    return ans;
}
```

本函数传入两个`BIGNUM`大整数，返回一个`BIGNUM`大整数为二者之和。仅处理二者均为正数的情况。整体思路是模拟竖式操作，大家在纸上列个竖式计算，一步一步模拟一下就明白了。

4. 除法

```
void BIGdiv_print(BIGNUM a,BIGNUM b) {
    for(int i=1; i<=400; ++i) {
        a=BIGmul10(a);
        int t=0;
        while(BIGcmp(a,b)>=0) {
            a=BIGsub(a,b);
            ++t;
        }
        finalAns[i]=t;
    }
}
```

这里我们只关心$a < b$时的除法。为了计算小数点后的400位，我们就给a乘以一个10^{400}。当然不是一下子就乘上去，而是算一位乘一次。实际上也就是列除法竖式的过程。这里可能讲得不太清楚，举例说明一下：

例如我们需要计算

$$\frac{17}{49}$$

小数点后的前4位。

首先将17乘以10变成170，然后计算$170 \div 49 = 3 \cdots 23$，那么小数点后第一位就是3，然后此时$a$变为23

再将23乘以10变成230，计算$230 \div 49 = 4 \cdots 34$，那么小数点后第二位就是4，然后此时$a$变为34

再将34乘以10变成340，计算$340 \div 49 = 6 \cdots 46$，那么小数点后第三位就是6，然后此时$a$变为46

再将46乘以10变成460，计算$460 \div 49 = 9 \cdots 19$，那么小数点后第四位就是9。

于是，我们算出$17/49$小数点后前四位是0.3469

为了实现上述操作，我们定义了几个新的函数`BIGmul10`（乘以十），`BIGcmp`（比大小），`BIGsub`（减法），将在下面一一详细说明

5. 乘以十

```
BIGNUM BIGmul10(BIGNUM a) {
    int len=getLenth(a);
    for(int i=len; i>=1; --i)
        a.number[i]=a.number[i-1];
    a.number[0]=0;
    return a;
}
```

传入一个数，将它乘以十。整体往高移动一位，然后最低位补一个0。

6. 比大小

```
int BIGcmp(BIGNUM a,BIGNUM b) { //a>b:1 a<b:-1 a=b:0
    int a_lenth=getLenth(a),b_lenth=getLenth(b);
    if(a_lenth>b_lenth) return 1;
    else if(a_lenth<b_lenth) return -1;
    else if(a_lenth==b_lenth) {
        for(int i=a_lenth-1; i>=0; --i) {
            if(a.number[i]!=b.number[i])
                return a.number[i]>b.number[i]?1:-1;
        }
        return 0;
    }
}
```

传入两个数，比较它们的大小。如果$a > b$，返回1；如果$a < b$，返回-1；如果$a = b$，返回0。

先比长度，然后从高位往低位依次比较就行。

7. 减法

```
BIGNUM BIGsub(BIGNUM a,BIGNUM b) { //a>b
    int a_len=getLenth(a),b_len=getLenth(b);
    BIGNUM ans;
    memset(ans.number,0,sizeof(ans.number));//因为ans是局部变量，所以要赋初值。
    int flag=0;
    for(int i=0; i<a_len; ++i) {
        if(a.number[i]-flag>=b.number[i]) {
```

```
            ans.number[i]=a.number[i]-flag-b.number[i];
            flag=0;
        } else {
            ans.number[i]=a.number[i]-flag+(10-b.number[i]);
            flag=1;
        }
    }
    return ans;
}
```

本函数传入两个 `BIGNUM` 大整数，返回一个 `BIGNUM` 大整数为二者之差，仅处理$a > b$的情况。整体思路也是模拟竖式操作，大家在纸上列个竖式计算，一步一步模拟一下就明白了。

8. 输出

```
void BIGprint(BIGNUM a) {
    int len=getLenth(a);
    for(int i=len-1; i>=0; --i) {
        printf("%d",a.number[i]);
    }
    printf("\n");
}
```

在调试时，我们可能需要知道大整数的值，于是写了一个输出函数供大家参考，在实际运算中并不调用。

主函数：

```
int main() {
    BIGNUM a,b,temp;
    memset(a.number,0,sizeof(a.number));
    memset(b.number,0,sizeof(b.number));
    memset(temp.number,0,sizeof(temp.number));

    a.number[0]=1;
    b.number[0]=2;
    /*
    因为斐波那契数列中当前的值只和前一个、前前一个有关系
    为了避免过大的内存开销
    这里只使用3个BIGNUM变量进行循环计算。
    */
    for(int i=1; i<=1250; ++i) {
        temp=b;
        b=BIGadd(a,b);
        a=temp;
    }
    BIGdiv_print(a,b);
    int k;
    while(scanf("%d",&k)!=EOF){
        printf("%d\n",finalAns[k]);
    }
    return 0;
}
```

完整代码

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#define ll long long
#define maxn 10000+5
const double eps=1e-11;

typedef struct {
    int number[maxn];//从0-lenth,是从低位到高位
} BIGNUM;

int finalAns[1000];

int getLenth(BIGNUM a) {
    int i;
    for(i=(maxn-1); i>=0; --i) {
        if(a.number[i]>0 || i==0)
            break;
    }
    return i+1;
}

BIGNUM BIGadd(BIGNUM a,BIGNUM b) { //都是正的加法
    int a_lenth=getLenth(a),b_lenth=getLenth(b);
    int maxLenth=a_lenth>b_lenth?a_lenth:b_lenth;
    BIGNUM ans;
    memset(ans.number,0,sizeof(a.number));
    int flag=0;
    for(int i=0; i<maxLenth; ++i) {
        ans.number[i]=(flag+a.number[i]+b.number[i])%10;
        if(flag+a.number[i]+b.number[i]>=10)
            flag=1;
        else
            flag=0;
    }
    if(flag)
        ans.number[maxLenth]+=flag;
    return ans;
}

BIGNUM BIGsub(BIGNUM a,BIGNUM b) { //a>b
    int a_len=getLenth(a),b_len=getLenth(b);
    BIGNUM ans;
    memset(ans.number,0,sizeof(ans.number));
    int flag=0;
    for(int i=0; i<a_len; ++i) {
        if(a.number[i]-flag>=b.number[i]) {
            ans.number[i]=a.number[i]-flag-b.number[i];
            flag=0;
        } else {
            ans.number[i]=a.number[i]-flag+(10-b.number[i]);
```

```
            flag=1;
        }
    }
    return ans;
}

BIGNUM BIGmul10(BIGNUM a) {
    int len=getLenth(a);
    for(int i=len; i>=1; --i)
        a.number[i]=a.number[i-1];
    a.number[0]=0;
    return a;
}

int BIGcmp(BIGNUM a,BIGNUM b) { //a>b 1 a<b -1 a=b 0
    int a_lenth=getLenth(a),b_lenth=getLenth(b);
    if(a_lenth>b_lenth) return 1;
    else if(a_lenth<b_lenth) return -1;
    else if(a_lenth==b_lenth) {
        for(int i=a_lenth-1; i>=0; --i) {
            if(a.number[i]!=b.number[i])
                return a.number[i]>b.number[i]?1:-1;
        }
        return 0;
    }
}
void BIGprint(BIGNUM a) {
    int len=getLenth(a);
    for(int i=len-1; i>=0; --i) {
        printf("%d",a.number[i]);
    }
    printf("\n");
}


void BIGdiv_print(BIGNUM a,BIGNUM b) {
    for(int i=1; i<=400; ++i) {
        a=BIGmul10(a);
        int t=0;
        while(BIGcmp(a,b)>=0) {
            a=BIGsub(a,b);
            ++t;
        }
        finalAns[i]=t;
    }
}

int main() {
    BIGNUM a,b,temp;
    memset(a.number,0,sizeof(a.number));
    memset(b.number,0,sizeof(b.number));
    memset(temp.number,0,sizeof(temp.number));

    a.number[0]=1;
    b.number[0]=2;
    for(int i=1; i<=1250; ++i) {
```

```
            temp=b;
            b=BIGadd(a,b);
            a=temp;
        }
        BIGdiv_print(a,b);
        int k;
        while(scanf("%d",&k)!=EOF){
            printf("%d\n",finalAns[k]);
        }
        return 0;
}
```

K - 多项式的加法

难度	考点
4	数组

题目分析

对于每个多项式，可以开两个数组，第i个数组元素分别记录第i项的系数和次数。

注意由于数组占用的内存空间过大，我们需要将数组开成全局变量，并且由于多项式不一定次数都相等，所以我们的数组需要开成题目中数据范围的两倍大小。

然后是多项式的加法，我们可以同时开设两个循环变量，分别表示两个记录多项式的数组的循环下标，每次取两者当前项中次数较高的一项，输出；如果两项次数相等，那么将两项的系数相加之后再输出；如果某一个数组已经扫描完了，那么直接输出另外一个数组中的项即可。

注意：不要忘了看数据范围，这个题是要开`long long`的！

具体的细节可以看代码。

参考代码

```
#include <stdio.h>

#define N (200000 + 5)

long long a[N], b[N];
long long s[N], t[N];

int main()
{
    int n, m;
    int p = 1, q = 1;
    int i;

    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; i++)
        scanf("%lld%lld", &a[i], &s[i]);
    for (i = 1; i <= m; i++)
        scanf("%lld%lld", &b[i], &t[i]);
```

```
    while (p <= n || q <= m)
    {
        if ((p <= n && q <= m && s[p] > t[q]) || q > m)
        {
            // 只计算 p 的情况：p 对应的指数较大，或另一个数组已经扫描完了
            printf("%lld %lld ", a[p], s[p]);
            p++;
        }
        else if ((p <= n && q <= m && s[p] < t[q]) || p > n)
        {
            // 只计算 q 的情况：q 对应的指数较大，或另一个数组已经扫描完了
            printf("%lld %lld ", b[q], t[q]);
            q++;
        }
        else
        {
            // 合并系数的情况：能来到这个分支，一定是 pq 都没有扫描完，且 f 的第 p 项和 g 的第 y 项系
数相等
            if (a[p] + b[q] != 0)
                printf("%lld %lld ", a[p] + b[q], t[q]);
            p++, q++;
        }
    }

    return 0;
}
```