# 信息学算法竞赛——高精度运算

## 【本课要点】

1、 高精度数的存储与输入输出
2、 高精度加法
3、 高精度减法
4、 高精度乘法
5、 高精度除法
6、 例题

## 【本课内容】

整数基本类型的范围有限，最大的 long long 类型是 8 字节的，取值范围是 $-2^{63}$~$2^{63}-1$，对于更大的数值就无能为力了。

高精度计算的基本思想是用一个数组来保存一个大数，在此基础上进行数值计算。

### 一、高精度数的存储与输入输出

#### 1、存储

将一个大整数按每一个数位分别存放在一个一维数组中。个位数存在 a[1]，十位数存在 a[2]，依次类推；数位长度存放在一个变量中，也可以将数位长度存放在 a[0]。

可以构建一个如下的结构体：

```
struct Bign{
    int len, a[MAXN];
    Bign(){
        memset(a, 0, sizeof(a));
        len = 1;
    }
}
```

#### 2、输入

由于一个大整数输入时并不确定其位数，所以采取字符串读入的方式最方便。

```
void reand(Bign &x){
    string s;
    cin>>s;
    x.len =s.size();
    for(int i=0;i<x.len;i++)
        x.a[x.len-i]=s[i]-'0';
}
```

#### 3、输出

直接将数组中的每一位从高到低打印即可。

```
void write(Bign&x){
    for(int i=x.len;i>=1;i--)
        cout<<x.a[i];
}
```

### 二、高精度加法

方法：根据竖式计算的方法，用 C++模拟出来。对于进位的处理可以边计算边处理进位，也可以在计算完后处理进位。

```
void add(const Bign&x,const Bign&y,Bign &z){
    z.len=max(x.len,y.len);
    for(int i=1;i<=z.len;i++)//先将每一位总和计算
        z.a[i]=x.a[i]+y.a[i];
    for(int i=1;i<=z.len;i++){//处理进位
        z.a[i+1]+=z.a[i]/10;
        z.a[i]%=10;
    }
    if(z.a[z.len+1]>0)z.len++;//最高位处理
}
```

三、高精度减法

两个大整数减法，需要先判断两个数的大小，用大的数减去小的数，如果原本是小数减大数，则需要单独处理负号。

下面代码默认数 x 大于数 y

```
void jian(const Bign&x,const Bign&y,Bign&z){//默认 x 大于 y，结果为 z
    z.len=x.len;
    for(int i=1;i<=z.len;i++){
        z.a[i]+=x.a[i]-y.a[i];
        if(z.a[i]<0){//借位
            z.a[i]+=10;
            z.a[i+1]-=1;
        }
    }
    while(z.a[z.len]==0&&z.len>1)z.len--;//处理高位前导 0
}
```

四、高精度乘法

1、高精度乘以单精度

```
void mul1(const Bign&x,int y,Bign &ans){
    ans.len=x.len;
    for(int i=1;i<=ans.len;i++)
        ans.a[i]=x.a[i]*y;
    for(int i=1;i<=ans.len;i++){
        ans.a[i+1]+=ans.a[i]/10;
        ans.a[i]%=10;
    }
    while(ans.a[ans.len+1]>0){
        ans.len++;
        ans.a[ans.len+1]+=ans.a[ans.len]/10;
        ans.a[ans.len]%=10;
    }
}
```

2、高精度乘以高精度

```
void mul(const Bign&x,const Bign&y,Bign &z){
    z.len=x.len+y.len-1;
    for(int i=1;i<=x.len;i++)
        for(int j=1;j<=y.len;j++)
            z.a[i+j-1]+=x.a[i]*y.a[j];
    for(int i=1;i<=z.len;i++){
        z.a[i+1]+=z.a[i]/10;
        z.a[i]%=10;
    }
    while(z.a[z.len+1]>0){
        z.len++;
        z.a[z.len+1]+=z.a[z.len]/10;
        z.a[z.len]%=10;
    }
}
```

五、高精度除法

高精度除法也分为高精度除以单精度，高精度除以高精度。只要求掌握高精度除以单精度即可。

```
void div(const Bign&x,int y,Bign &z){//高精度除以单精度
       int r=0;
       z.len=x.len;
       if(x.len==1&&x.a[1]==0){
           z.a[1]=0;return;
       }
       for(int i=x.len;i>=1;i--){
           z.a[i]=((ll)r*10+x.a[i])/y;
           t=((ll)r*10+x.a[i])%y;
       }
       while(z.a[z.len]==0&&z.len>1)z.len--;
}
```

**六、例题**

**例一：P1500. 高精度加法**

**【模板题参考代码】**

```
#include<bits/stdc++.h>
using namespace std;
struct Bign{
    int len,a[1001];
    Bign(){
        memset(a,0,sizeof(a));len=1;
    }
};
Bign x,y,z;
```

```
string s;
void read(Bign&x){
    cin>>s;
    x.len =s.size();
    for(int i=0;i<x.len;i++)
        x.a[x.len-i]=s[i]-'0';
}
void write(Bign&x){
    for(int i=x.len;i>=1;i--)
        cout<<x.a[i];
}
void add(const Bign&x,const Bign&y,Bign &z){
    z.len=max(x.len,y.len);
    for(int i=1;i<=z.len;i++)//jia
        z.a[i]=x.a[i]+y.a[i];
    for(int i=1;i<=z.len;i++){//jinwei
        z.a[i+1]+=z.a[i]/10;
        z.a[i]%=10;
    }
    if(z.a[z.len+1]>0)z.len++;//zui gaowei
}
int main(){
    read(x);
    read(y);
    add(x,y,z);
    write(z);
    return 0;
}
```

例二、**P1502 阶乘之和**

【问题描述】用高精度计算出 S=1!+2!+3!+…+n!（n \le 50n≤50）。

其中“!”表示阶乘，例如： 5!=5×4×3×2×1。

【数据范围】对于 100% 的数据， 1≤n≤100。

【分析】

先求 i!，然后再累加。

需要用到大整数乘普通整数、高精度加法；

根据 n<=50 估算高精度数的位数：

粗略估算：

i 位*j 位最少(i+j-1)位，最多(i+j)位，1~9 各 1 位，10~99 各 2 位，总共不超过 9+80*2=170 位；

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
int n;
struct Bign{
```

```
    int len,a[200];
    Bign(){
        memset(a,0,sizeof(a));len=1;
    }
}x,ans;
void add(const Bign&x,const Bign&y,Bign &z){
    z.len=max(x.len,y.len);
    for(int i=1;i<=z.len;i++)//先将每一位总和计算
        z.a[i]=x.a[i]+y.a[i];
    for(int i=1;i<=z.len;i++){//处理进位
        z.a[i+1]+=z.a[i]/10;
        z.a[i]%=10;
    }
    if(z.a[z.len+1]>0)z.len++;//最高位处理
}
void mul(const Bign&x,int y,Bign &ans){
    ans.len=x.len;
    for(int i=1;i<=ans.len;i++)
    ans.a[i]=x.a[i]*y;
    for(int i=1;i<=ans.len;i++){
        ans.a[i+1]+=ans.a[i]/10;
        ans.a[i]%=10;
    }
    while(ans.a[ans.len+1]>0){
    ans.len++;
        ans.a[ans.len+1]+=ans.a[ans.len]/10;
        ans.a[ans.len]%=10;
    }
}
int main(){
    cin>>n;
    x.len=1;x.a[1]=1;
    ans.len=1;ans.a[1]=1;
    for(int i=2;i<=n;i++){
        mul(x,i,x);
        add(ans,x,ans);
    }
    for(int i=ans.len;i>=1;i--)
        cout<<ans.a[i];
    return 0;
}
```

例 3：**P1503 麦森数**

**问题描述**

形如 $2^P-1$ 的素数称为麦森数，这时 P 一定也是个素数。但反过来不一定，即如果 P 是个素数，$2^P-1$ 不一定也是素数。到 1998 年底，人们已找到了 37 个麦森数。最大的一个是 P=3021377，它有 909526 位。麦森数有许多重要应用，它与完全数密切相关。

**任务**：从文件中输入 P（1000<P<3100000），计算 $2^P-1$ 的位数和最后 500 位数字（用十进制高精度数表示）

【分析】

第一问计算位数。

需要计算的是 $2^P-1$，由于 $2^P$ 的个位一定不为 0，所有 $2^P-1$ 的位数与 $2^P$ 的位数一致。

设 ans=2P，$\log_{10}(ans)=\log_{10}(2^P)=P*\log_{10}(2)$

所以位数= $P*\log_{10}(2)+1$

第二问只要求保留最后 500 位，可以设计一个 500 位的大整数，采取求快速幂的方法计算最后答案。

主要涉及高精度乘法

【快速幂方法】

首先，如果要计算 $a^p$, 将 p 表示为二进制有：$2^{m[1]}+2^{m[2]}+2^{m[3]}+..+2^{m[k]}$, 则 $a^p=a^{2^{m[1]}}\cdot a^{2^{m[2]}}\cdot\ldots\cdot a^{2^{m[k]}}$；如 $5^{21}=5^{(16+4+1)}=5^{16}\cdot 5^4\cdot 5^1$。其中用二进制表示 $21=(10101)_2$，这样可以不停的倍增 $5^1,5^2,5^4$，$5^8,5^{16}$，在 21 的二进制对应位数存在 1 时，将数字累乘至答案即可。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
struct Bign{
    int len,a[1011];
    Bign(){
        memset(a,0,sizeof(a));len=1;
    }
    Bign operator =(const Bign & t){
        len=t.len;
        memcpy(a,t.a,sizeof(t.a));
    }
};
void write(const Bign&x){
    int k=0;
    for(int i=500;i>=1;i--){
        cout<<x.a[i];
        k++;
        if(k%50==0)cout<<endl;
    }
}
Bign mul(const Bign &x,const Bign& y ){
    Bign z;
```

```
    z.len=x.len+y.len-1;
    for(int i=1;i<=x.len;i++)
        for(int j=1;j<=y.len;j++)
            z.a[i+j-1]+=x.a[i]*y.a[j];
    for(int i=1;i<=z.len;i++){
        z.a[i+1]+=z.a[i]/10;
        z.a[i]%=10;
    }
    while(z.a[z.len+1]>0){
        z.len++;
        z.a[z.len+1]+=z.a[z.len]/10;
        z.a[z.len]%=10;
    }
    z.len=min(500,z.len);return z;
}
int p;Bign x,y;
int main(){
    cin>>p;
    cout<<(int)(p*log10(2))+1<<endl;
    x.a[1]=2;y.a[1]=1;
    while(p){
        if(p&1) y=mul(x,y);
        x=mul(x,x);
        p>>=1;
    }
    y.a[1]=y.a[1]-1;
    write(y);
    return 0;
}
```

**【知识拓展】操作符重载**

在大整数运算中，大量的运算需要根据需求设计各类函数完成计算。我们可以利用 C++ 的特性将涉及的运算符进行重载，这样在程序中可以更直观的使用各类计算。

C++ 允许在同一作用域中的某个函数和运算符指定多个定义，分别称为函数重载和运算符重载。

重载声明是指一个与之前已经在该作用域内声明过的函数或方法具有相同名称的声明，但是它们的参数列表和定义（实现）不相同。

可以简单理解，就是相同功能的函数，当有不同参数时，可以根据参数设计独立不同的函数。

比如下面两个函数，函数名完全一样，但是参数表不同，第二个函数就是对第一个函数的重载，

```
int   max(int a,int b){
        return a>b?a:b;
}
```

```
double    max(double a,double b){
          return a>b?a:b;
}
```

您可以重定义或重载大部分 C++ 内置的运算符。这样，您就能使用自定义类型的运算符。

重载的运算符是带有特殊名称的函数，函数名是由关键字 operator 和其后要重载的运算符符号构成的。与其他函数一样，重载运算符有一个返回类型和一个参数列表。

如重载小于符号：

```
bool operator <(const Bign&t)const{
    if(len!=t.len)return len<t.len;
    for(int i=len;i>=1;i--)
        if(a[i]!=t.a[i])
            return a[i]<t.a[i];
}
```

**例 4： 高精度常见操作重载写法**

```
#include<bits/stdc++.h>
using namespace std;
#define B 10000
//B 表示进制数
#define ll long long
struct BIG{//无符号
    int len,a[10001];
    BIG(){
        len=1;memset(a,0,sizeof(0));
    }
    BIG(int n){
        *this=n;
    }
    BIG operator=(int n){
        len=0;
        if(!n)len=1;
        while(n)
            a[++len]=n%B,n/=B;
    }
    BIG operator=(string s){
        len=1;int l=s.size(),k=1;
        for(int i=l-1;i>=0;i--){
            if(k>=B){
                k=1;len++;
            }
            a[len]=(s[i]-'0')*k+a[len];
            k*=10;
        }
    }
```

```
BIG operator =(const BIG & t){
    len=t.len;memcpy(a,t.a,sizeof(t.a));
}
bool operator <(const BIG&t)const{
    if(len!=t.len)return len<t.len;
    for(int i=len;i>=1;i--)
        if(a[i]!=t.a[i])
               return a[i]<t.a[i];
}
bool operator >(const BIG&t)const{
    return t < *this;
}
bool operator ==(const BIG&t)const{
    return !(*this<t&&*this>t);
}
bool operator <=(const BIG&t)const{
    return !(*this>t);
}
bool operator >=(const BIG&t)const{
    return !(*this<t);
}
bool operator !=(const BIG&t)const{
    return !(*this==t);
}

BIG operator+(  BIG& t) const{
    BIG ret;
    ret.len=max(len,t.len);
    for(int i=1;i<=ret.len;i++){
        ret.a[i]+=a[i]+t.a[i];
        ret.a[i+1]+=ret.a[i]/B;
        ret.a[i]%=B;
    }
    if(ret.a[ret.len+1]>0)ret.len++;
    return ret;
}
BIG operator-(BIG&t) const{//一定大减小
    BIG ret=*this;
    for(int i=1;i<=ret.len;i++){
        if(ret.a[i]<t.a[i]){
            ret.a[i]+=B;
            ret.a[i+1]-=1;
        }
        ret.a[i]=ret.a[i]-t.a[i];
```

```
        }
        while(ret.len>1&&ret.a[ret.len]==0)ret.len--;
        return ret;
    }
    BIG operator*(BIG&t) const{
        BIG ret;
        ret.len=len+t.len-1;
        for(int i=1;i<=len;i++)//竖式运算
            for(int j=1;j<=t.len;j++)
                ret.a[i+j-1]+=a[i]*t.a[j];
        for(int i=1;i<=ret.len;i++)//处理进位
            if(ret.a[i]>B){
                ret.a[i+1]+=ret.a[i]/B;
                ret.a[i]%=B;
            }
        int &l=ret.len;
        while(ret.a[l+1])l++,ret.a[l+1]+=ret.a[l]/B,ret.a[l]%=B;//
处理高位进位
        return ret;
    }

    BIG operator/(long long  t)const {
        BIG ret;
        int x=0;
        if(len==1&&a[1]==0)return 0;
        for(int i=len;i>=1;i--){
            ret.a[i]=((ll)x*B+a[i])/t;
            x=((ll)x*B+a[i])%t;
        }
        ret.len=len;
        while(ret.a[ret.len]==0&&ret.len>1)ret.len--;
        return ret;
    }
};
ostream & operator << (ostream &ou,const BIG& t){
    ou<<t.a[t.len];
    for(int i=t.len-1;i>=1;i--){
        int k=B/10,x=t.a[i];
        while(k){
            ou<<x/k;
            x%=k; k/=10;
        }
    }
    return ou;
```

```
}
istream&operator>>(istream&in,BIG & t){
    string s;in>>s;t=s;return in;
}
int main(){
    BIG a;long long  b;
    cin>>a;cin>>b;
    cout<<a/b;
    return 0;
}
```