高精度专题

高精度专题
- BigNumber模板
- 详细版
- 二进制算法求GCD

BigNumber模板

```
struct BigNumber{ //高精度专用
    long long num[N];
    BigNumber(){memset(num,0,sizeof(num));}
    void print(){
        for(int i=num[0];i;i--){
            if(i==num[0]) printf("%lld",num[i]); //把最后几位数字输出
            else printf("%08lld",num[i]); //从小到大按8位进行输出
        }
        printf("\n");
    }
    BigNumber operator + (const BigNumber &a)const{ //高精度加法
        BigNumber ans;
        ans.num[0]=max(a.num[0],num[0]); //位数取两者较大
        long long tmp=0;
        for(int i=1;i<=ans.num[0];i++){ //每一位相加
            ans.num[i]=num[i]+a.num[i]+tmp;
            tmp=ans.num[i]/mod; //超过的部分加到下一个1e9
            ans.num[i]%=mod;
        }
        while(tmp){
            ans.num[++ans.num[0]]=tmp%mod; //进位
            tmp/=mod;
        }
        return ans;
    }

    BigNumber operator -(const BigNumber &a)const{//高精度减法
        BigNumber ans;
        ans.num[0]=num[0];
        long long tmp=0;
        for(int i=1;i<=ans.num[0];i++){
            if(num[i]-tmp-a.num[i]<0) ans.num[i]=num[i]-tmp-a.num[i]+mod,tmp=1;
            else ans.num[i]=num[i]-tmp-a.num[i],tmp=0;
        }
        while(ans.num[ans.num[0]]==0) ans.num[0]--;
        return ans;
    }

    BigNumber operator *(const BigNumber &a)const{
        BigNumber ans,tmp;
        long long tmp1=0,tmp2;
```

```
        for(int i=1;i<=a.num[0];i++){
            tmp2=a.num[i]; //保存某八位的乘数
            tmp.num[0]=i-1; //
            tmp1=0; //保存进位
            tmp.num[tmp.num[0]]=0;
            for(int j=1;j<=num[0];j++){
                tmp.num[++tmp.num[0]]=num[j]*tmp2+tmp1; //更新当前位
                tmp1=tmp.num[tmp.num[0]]/mod;  //保存进位
                tmp.num[tmp.num[0]]%=mod;
            }
            while(tmp1){
                tmp.num[++tmp.num[0]]=tmp1%mod; //进位
                tmp1/=mod;
            }
            ans=ans+tmp; //更新答案
        }
        return ans;
    }

    BigNumber operator ^ (const int &a)const{ //高精度乘方2^i
        BigNumber ans,x;
        int y=a;
        ans.num[0]=ans.num[1]=1;
        for(int i=0;i<=num[0];i++) x.num[i]=num[i];
        while(y){ //快速幂
            if(y&1) ans=ans*x;
            x=x*x;
            y>>=1;
        }
        return ans;
    }
};
```

详细版

```
#include <bits/stdc++.h>

using namespace std;

const int MAXN = 5005;

struct bign
{
    int len, s[MAXN];
    bign ()  //初始化
    {
        memset(s, 0, sizeof(s));
        len = 1;
    }
    bign (int num) { *this = num; }
    bign (const char *num) { *this = num; }  //让this指针指向当前字符串
    bign operator = (const int num)
    {
        char s[MAXN];
        sprintf(s, "%d", num);  //sprintf函数将整型映到字符串中
```

```
        *this = s;
        return *this;  //再将字符串转到下面字符串转化的函数中
    }
    bign operator = (const char *num)
    {
        for(int i = 0; num[i] == '0'; num++) ;  //去前导0
        len = strlen(num);
        for(int i = 0; i < len; i++) s[i] = num[len-i-1] - '0'; //反着存
        return *this;
    }
    bign operator + (const bign &b) const //对应位相加，最为简单
    {
        bign c;
        c.len = 0;
        for(int i = 0, g = 0; g || i < max(len, b.len); i++)
        {
            int x = g;
            if(i < len) x += s[i];
            if(i < b.len) x += b.s[i];
            c.s[c.len++] = x % 10;  //关于加法进位
            g = x / 10;
        }
        return c;
    }
    bign operator += (const bign &b)  //如上文所说，此类运算符皆如此重载
    {
        *this = *this + b;
        return *this;
    }
    void clean()  //由于接下来的运算不能确定结果的长度，先大而估之然后再查
    {
        while(len > 1 && !s[len-1]) len--;  //首位部分‘0’故删除该部分长度
    }
    bign operator * (const bign &b) //乘法重载在于列竖式，再将竖式中的数转为抽象，即可看出运算法则。
    {
        bign c;
        c.len = len + b.len;
        for(int i = 0; i < len; i++)
        {
            for(int j = 0; j < b.len; j++)
            {
                c.s[i+j] += s[i] * b.s[j];//不妨列个竖式看一看
            }
        }
        for(int i = 0; i < c.len; i++) //关于进位，与加法意同
        {
            c.s[i+1] += c.s[i]/10;
            c.s[i] %= 10;
        }
        c.clean();  //我们估的位数是a+b的长度和，但可能比它小（1*1 = 1）
        return c;
    }
    bign operator *= (const bign &b)
    {
        *this = *this * b;
        return *this;
    }
```

```
bign operator - (const bign &b)  //对应位相减，加法的进位改为借1
{  //不考虑负数
    bign c;
    c.len = 0;
    for(int i = 0, g = 0; i < len; i++)
    {
        int x = s[i] - g;
        if(i < b.len) x -= b.s[i];  //可能长度不等
        if(x >= 0) g = 0;  //是否向上移位借1
        else
        {
            g = 1;
            x += 10;
        }
        c.s[c.len++] = x;
    }
    c.clean();
    return c;
}
bign operator -= (const bign &b)
{
    *this = *this - b;
    return *this;
}
bign operator / (const bign &b)  //运用除是减的本质，不停地减，直到小于被减数
{
    bign c, f = 0; //可能会在使用减法时出现高精度运算
    for(int i = len-1; i >= 0; i--)  //正常顺序，从最高位开始
    {
        f = f*10;  //上面位的剩余到下一位*10
        f.s[0] = s[i];  //加上当前位
        while(f >= b)
        {
            f -= b;
            c.s[i]++;
        }
    }
    c.len = len;  //估最长位
    c.clean();
    return c;
}
bign operator /= (const bign &b)
{
    *this  = *this / b;
    return *this;
}
bign operator % (const bign &b)  //取模就是除完剩下的
{
    bign r = *this / b;
    r = *this - r*b;
    r.clean();
    return r;
}
bign operator %= (const bign &b)
{
    *this = *this % b;
    return *this;
}
```

```
    bool operator < (const bign &b) //字符串比较原理
    {
        if(len != b.len) return len < b.len;
        for(int i = len-1; i != -1; i--)
        {
            if(s[i] != b.s[i]) return s[i] < b.s[i];
        }
        return false;
    }
    bool operator > (const bign &b)  //同理
    {
        if(len != b.len) return len > b.len;
        for(int i = len-1; i != -1; i--)
        {
            if(s[i] != b.s[i]) return s[i] > b.s[i];
        }
        return false;
    }
    bool operator == (const bign &b)
    {
        return !(*this > b) && !(*this < b);
    }
    bool operator != (const bign &b)
    {
        return !(*this == b);
    }
    bool operator <= (const bign &b)
    {
        return *this < b || *this == b;
    }
    bool operator >= (const bign &b)
    {
        return *this > b || *this == b;
    }
    string str() const  //将结果转化为字符串（用于输出）
    {
        string res = "";
        for(int i = 0; i < len; i++) res = char(s[i]+'0')+res;
        return res;
    }
};

istream& operator >> (istream &in, bign &x) //重载输入流
{
    string s;
    in >> s;
    x = s.c_str();  //string转化为char[]
    return in;
}

ostream& operator << (ostream &out, const bign &x)  //重载输出流
{
    out << x.str();
    return out;
}

int main()
{
```

```
    bign a;//除了声明外其他如整型般使用
    //……
    return 0;
}
```

二进制算法求GCD

主要用于大整数求GCD，不过真的不用Python吗?

```
void Gcd(int a[],int b[],int t){ //针对大整数求GCD
    if(comp(a,b)==0){T=t;return;}//相等，返回提取2共T次
    if(comp(a,b)<0){Gcd(b,a,t);return;}
    int ta,tb;
    if(a[1]%2==0){Div(a,2);ta=1;}
    else ta=0;
    if(b[1]%2==0){Div(b,2);tb=1;}
    else tb=0;
    if(ta&&tb) Gcd(a,b,t+1); //均为偶数
    else if(!ta&&!tb){Minus(a,b);Gcd(a,b,t);}//均为奇数
    else Gcd(a,b,t);
}
```

```
from sys import *
setrecursionlimit(100000)
def gcd(x,y):
    if y==0:
        return x
    else:
        return gcd(y,x%y)

a=int(input())
b=int(input())

print(gcd(a,b))
```