大数运算实现

目录

一、预定义的一些函数和宏（在.h 文件中）

1.1024 位的大整数 BN 2048 位 BND

它们的第 0 “位”都用来表示“位”数，每“位”是一个 32 位的二进制数 uint32_t

```
#define BITPERDIGIT 32//每个小块是uint32，32位顺序和人期望是一致的，当黑盒就好
#define BASE 0x100000000U//每个小块uint32，基数就是2^32,可以在模的时候用上
#define BNMAXDGT 32//最多32“位”个32位
#define ALLONE 0xffffffffU
#define ALLONEL 0xffffffffUL
typedef uint32_t BN[BNMAXDGT + 1];//BN是1024+32位的，多一位表示位数
#define BNMAXBIT BNMAXDGT<<5 //BN最大能处理1024位
#define BNDMAXBIT BNMAXBIT<<1 //BND最大能处理2048位
typedef uint32_t BND[1 + (BNMAXDGT << 1)];//BND能处理2048位，长度是2048+32
#define BNSIZE sizeof(BN)//BN 占用的字节数
```

2.常数 0，1，2，其中 0 是最常用的

```
BN ZERO_BN = { 0 };//0
BN ONE_BN = { 1,1 };//1
BN TWO_BN = { 1,2 };//2
```

3.状态标志 flag，一开始写的时候用上了，后面写着就没用上，以后修改扩展的时候可以用

```
#define FLAG_OK 0 //正常
#define FLAG_OF 1 //上溢了，最后一次[31]+[31]出现了进位，自动模除不用管
#define FLAG_UF 2//可能是减法下溢了
#define FLAG_DIVZERO 3//除0错误
#define FLAG_FILE_ERROR 4//读写文件错误
#define FLAG_NOINV 5//不互素没有逆元
#define FLAG_ERROR 6//出错了
```

4.常用函数

```
#define MIN(a,b) ((a)<(b)?(a):(b))  //比较大小
#define DIGITS_B(n_b) ((uint32_t)*(n_b)) //获得位数,[0]里面存的是 uint32 格式的位数
#define SETDIGITS_B(n_b, b)  (*(n_b) = (uint32_t)(b)) //设置位数，置[0]为想要的位数
#define MSDPTR_B(n_b)  ((n_b) + DIGITS_B (n_b)) //指向最高位,用指针加法
#define LSDPTR_B(n_b)    ((n_b)+1)  //指向最低位,[0]是位数，[1]才是数字
#define SETZERO_B(n_b)    (*(n_b) = 0)  //置为全0，位数置为0就可以了，0位，以后可以覆盖
inline void INCDIGITS_B(BN n_b) {*(n_b) = *(n_b)+1; }  //增加1位位数
inline void DECDIGITS_B(BN n_b)  //减少1位位数
```

```
{   if (DIGITS_B(n_b) > 0)     *(n_b) = *(n_b)-1; }
//消除前导0，只要有32为基数的位数，且那一位为0，就位数减去1位，获得实际“位”数
inline void RMLDZRS_B(BN n_b)
{   while ((DIGITS_B(n_b) > 0) && (*MSDPTR_B(n_b) == 0))
    {DECDIGITS_B(n_b);}
}

//设置一个uint32数为大数，只占1“位”
inline void SETONEBIT_B(BN num, uint32_t u)
{
    *LSDPTR_B(num) = u;
    SETDIGITS_B(num, 1);
    RMLDZRS_B(num);
}

//拷贝src_b到dest_b中，可以复制BND，与结构无关
inline void cpy_b(BN dest_b, BN src_b);

//比较大小两个大数大小，相等返回0，大于返回1，小于返回-1
inline int cmp_b(BN a, BN b)

//设置每一位为0xffffff来实现小减大取模，减法用到
inline void setmax_b(BN & a)
```

二、主要函数概览(在.h 文件中)

1.加法

```
int add_b(BN a, BN b, BN sum);//a+b=sum，会模掉1024位
void add(BN a, BN b, BN sum);//add的另一个形式，区别是sum可能比预料的多一位，在求模加的时候用得到,没有模1024位
int adduint_b(BN a, uint32_t b, BN sum);//a加一个uint32形式的b,a+b=sum
int adduint_b(uint32_t a, BN b, BN &sum);
```

2.减法

```
int sub_b(BN a, BN b, BN &result);//result=a-b
int subuint_b(BN a, uint32_t b, BN &result);//减一个uint32
void sub(BN a, BN b, BN result);//没有模1024位的减法，会用的到
```

3.乘法

```
int mul_b(BN a, BN b, BN &result);//result = a*b 留下1024位
```

```
int mul(BN a, BN b, BN result);//没有模2^32^32的乘法
```

4. 除法

```
int div_b(BN a, BN b, BN q, BN rem);//a=qb+rem，商是q，余数是rem，a可以是BND，应付模幂运算
int remdiv_b(BN a, BN b, BN & rem);//取余数除法，a/b的余数是rem
```

5. 取模

```
int modn_b(BN a, BN n, BN & rem);//rem=a mod n
int modadd_b(BN a, BN b, BN n, BN & result);//result=a+b(mod n)
void modsub_b(BN a, BN b, BN n, BN &result);//result = a + b(mod n)
void modmul(BN a, BN b, BN n, BN & result);//模乘，result=a*b (mod n )
```

6. 信安数学中的函数

```
int gcd_b(BN a, BN b, BN & result);//求公因子，result=(a,b)
int inv_b(BN a, BN n, BN & x);//如果(a,n)=1,则有ax = 1 (mod n); 否则异常没有逆元，返回0，显然逆元不可能为0
int modexp_b(BN b, BN n, BN m, BN & result);//模幂运算，用的模平方,result= b^n (mod m)
int fermat_b(BN a);//用费马小定理检测a是不是素数，选的是2/3/5/7，200位的a会出现误报，500位还没有发现误报
void crt_b(BN a, BN b, BN p, BN q, BN & result);//用中国剩余定理求模幂，a^b mod(p*q)
```

7. 位运算函数

```
int shl_b(BN a);//左移一位
int shr_b(BN a);//右移一位
uint32_t getbits_b(BN a);//获取a的二进制位数，a可以是BND形式,最大2048
```

8. 特殊操作的函数

```
void exclu();//求前几十个素数的乘积，存在文件中，便于后面找素数的时候利用乘积求gcd，多求gcd，少进行费马检测
int genBN(BN result, int bits);//利用sha-1结果产生bits个二进制位的大数result
void writerand(char * addr);//把随机数写到addr中，后面对它求哈希
int findprime(BN a, int bits);//寻找bits位素数a，会调用genBN产生一个大数，然后利用exclu()结果进行调整才去费马
void genpq(char * p_path, char * q_path);//产生私钥p和q存在p_path和q_path文件中
```

9. SHA-1 的函数和常数

```
uint32_t H0 = 0x67452301;
```

```
uint32_t H1 = 0xEFCDAB89;
uint32_t H2 = 0x98BADCFE;
uint32_t H3 = 0x10325476;
uint32_t H4 = 0xC3D2E1F0;

const uint32_t K0 = 0x5A827999;
const uint32_t K1 = 0x6ED9EBA1;
const uint32_t K2 = 0x8F1BBCDC;
const uint32_t K3 = 0xCA62C1D6;
void subround(uint32_t & A, uint32_t & B, uint32_t & C, uint32_t & D, uint32_t & E, uint32_t &W, uint32_t K, int mode);
long long msgsize(char*plainaddr);
inline uint32_t cirleft(uint32_t word, int bit);
int mysha1(char *inputfileaddr, char *output);//每次调用用的是全局的，相当于一个状态向量
int SHA1(char *inputfileaddr, char *output);//每次调用里面会重新初始化向量
int checkresult(char * plainpath, char * decrypath);//检查明文和解密以后是否相同，相同返回1
```

10. 字符串和文件处理

```
string bn2str(BN bignum);//bignum转化为字符串形式返回，返回的不显示前导0，和正常预期是一样的，可以吞下BND
int str2bn(BN & bignum, string strbn);//字符串形式的转化为大数，只能转化为BN
int readbn(BN &bignum, string filename);//从文件filename中读取大数到bignum中，字符串是16进制的不能是0x开头
int writebn(string filename, BN bignum);//把大数bignum写入到文件filename中，不带前导0和前缀0x
```

三、函数细节说明

1. 加法

1.1 int add_b(BN a, BN b, BN sum)

①结果可能是1025位的，这种情况下会留下1024位。所以结果先存在temp中，temp比BN多1位。然后用aptr和bptr分别指向长和短的那个数的最低位，maptr和mbptr指向最高位。
②使用循环，来一位位加，中间结果存在carry中，carry是64位的，carry的高32位中存的是进位，低32位是本位和，把本位和赋值给和。
③最后可能会多了1位，这种情况下需要处理，如果超过了1024位，取模

1.2 void add(BN a, BN b, BN sum)

相比于 add_b()，只是最后少了一步返回溢出标志，发生移除时不置为 32“位”取模。

1.3 int adduint_b(BN a, uint32_t b, BN sum)

调用add_b()来实现加1个uint32_t类型的数据。

2. 减法

2.1 int sub_b(BN a, BN b, BN result);//result=a-b

与加法类似，减法也采用循环来实现，先判断 a 和 b 的大小，如果 a>b 就按正常的减，和加法类似，中间结果存在 carry 中，可以判断是否有借位，每次 a 的位减 b 的位时，还要减去上一次的借位。否则执行 b-a，然后取 1024 位模。

2.2 void sub(BN a, BN b, BN result)

这个函数和之前那个 sub_b 区别在于，没有比较 a 和 b 大小，直接做 a-b。一般情况下已知 a>b，直接调 sub，也不需要模 1024。

3. 乘法

3.1 int mul_b(BN a, BN b, BN &result)

乘法计算 a 和 b 的乘积，结果存在 result 中，结果大于 1024 位就模掉 1024 位。计算的时候，也用 uint64_t carry 保存本位和和进位，结果先存在 BND 形式的变量中，因为结果很可能会超过 1024 位。

先完成(bn-1bn-2...b1)·a0，然后再进入循环，此时 a[1]已经做了，从 a[2]开始循环做乘法，加上上一次进位 carry。循环完之后去掉前导 0，判断是不是超过 1024 位，如果是，取 1024 位，然后复制到结果 result 中。

3.2 int mul(BN a, BN b, BN result)

与前一个乘法的区别在于，没有模 1024 位，结果可以是 2048 位，做模幂的时候会用到，那时候乘积很可能是大于 1024 位的，如计算 p*p。

4. 除法

4.1 int mydiv_b(BN a, BN b, BN q, BN rem)

除法采用的是笨方法，计算机组成原理里面最简单的除法。把 a 想象成二进制展开了，把 b 想象成二进制展开了，然后把 b 左移移到和 a 对齐，然后进行比较大小操作，一个个二进制位上商。注意的地方是，除法很可能会出现 a 是 BND，然后商也是 BND，所以为了能对齐，把 a 存在 r_t 中，把 b

存在 BND 格式的 b_t 中，方便移位对齐。

除法调用的减法是 sub()，因为不能模 1024 位。除法调用的加法是 adduint()，商每次加 1 以后左移 1 位或者不加左移 1 位。最后 r_t 中剩下的就是余数。这个除法效率不高，所以导致了程序整体不快。

4.2 int remdiv_b(BN a, BN b, BN & rem)

取余除法，值返回余数，不需要商的情况下使用，它调用的是除法 div_b。

4.3 int div_b(BN a, BN b, BN rem)

使用 Knuth 方法的除法估计每次上的商 q'。根据 Knuth 的定理，在满足前提的情况下 q'至多比真实的 Q 大 2。扩大 a 和 b 的倍数，但是 b 的位数不变时，依然满足这种前提。

For programming this procedure we must repeatedly determine, for two large numbers $R = (r_n r_{n-1} \ldots r_0)_B$ and $b = (b_{n-1} b_{n-2} \ldots b_0)_B$ with $\lfloor R/b \rfloor < B$, the quotient $Q := \lfloor R/b \rfloor$ ($r_n = 0$ is a possibility). Here we take from Knuth the given approximation $\hat{q}$ of Q, which is computed from the leading digits of R and b.

Let

$$\hat{q} := \min\left\{ \left\lfloor \frac{r_n B + r_{n-1}}{b_{n-1}} \right\rfloor, B - 1 \right\}. \tag{4.1}$$

If $b_{n-1} \geq \lfloor R/b \rfloor$, then for $\hat{q}$ (see [Knut], Section 4.3.1, Theorems A and B), we have $\hat{q} - 2 \leq Q \leq \hat{q}$. Under the favorable assumption that the leading digit of the divisor is sufficiently large in comparison to B then as an approximation to Q, $\hat{q}$ is at most too large by 2 and is never too small. B/2

By scaling the operands a and b this can always be achieved. We choose $d > 0$ such that $d b_{n-1} \geq \lfloor B/2 \rfloor$, set $\hat{a} := ad = (\hat{a}_{m+n} \hat{a}_{m+n-1} \ldots \hat{a}_0)_B$, and set $\hat{b} := bd = (\hat{b}_{n-1} \hat{b}_{n-2} \ldots \hat{b}_0)_B$. The choice of d is then made in such a way that the number of digits of $\hat{b}$ never increases in comparison to that of b. In the above notation it is taken into account that $\hat{a}$ possibly contains one more digit than a (if this is not the case, then we set $\hat{a}_{m+n} = 0$). In any case, it is practical to choose d as a power of 2, since then the scaling of the operands can be carried out with simple shift operations. Since both operands are multiplied by a common factor, the quotient is unchanged; we have $\lfloor \hat{a}/\hat{b} \rfloor = \lfloor a/b \rfloor$.

在《计算机程序设计艺术 第二卷》中第三章如下阐述：

设 $u = (u_n u_{n-1} \cdots u_1 u_0)_b$ 和 $v = (v_{n-1} \cdots v_1 v_0)_b$ 是 b 进制下的非负整数，其中 $u/v < b$。试求一个算法以确定 $q = \lfloor u/v \rfloor$。

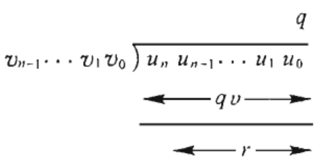

图 6　要求：一个快速地确定 q 的方式

是使得 $0 \leqslant r < v$ 的惟一整数。

这个问题的最明显的解决方法，是根据 u 和 v 的最高位数字，来对 q 做推测。这一方法是否可靠不够明显，但这种算法是值得研究的。因此我们置

$$\hat{q} = \min\left(\left\lfloor \frac{u_n b + u_{n-1}}{v_{n-1}} \right\rfloor, b-1\right) \tag{2}$$

这个公式说，$\hat{q}$ 是通过把 u 的两位前导数字除以 v 的前导数字得到的；而且若结果是 b 或更大，则以 $b-1$ 代替之。

现在我们研究一个值得注意的事实，即只要 v_{n-1} 适当大，这个值 $\hat{q}$ 总是非常近似于所求的答案 q。为了分析 $\hat{q}$ 怎样接近于 q，我们首先要说明 $\hat{q}$ 绝不会太小。

定理 A　*在上述记号下 $\hat{q} \geqslant q$。*

定理 B　*如果 $v_{n-1} \geqslant \lfloor b/2 \rfloor$，则 $\hat{q} - 2 \leqslant q \leqslant \hat{q}$。*

算法如下，但是细节处理上有不同：

Algorithm for division with remainder of $a = (a_{m+n-1}a_{m+n-2}\ldots a_0)_B \geq 0$ **by** $b = (b_{n-1}b_{n-2}\ldots b_0)_B > 0$

1. Determine the scaling factor d as given above.
2. Set $r := (r_{m+n}r_{n+m-1}r_{m+n-2}\ldots r_0)_B \leftarrow (0a_{m+n-1}a_{m+n-2}\ldots a_0)_B$.
3. Set $i \leftarrow m+n, j \leftarrow m$.
4. Set $\hat{q} \leftarrow \min\left\{\left\lfloor \frac{\hat{r}_i B + \hat{r}_{i-1}}{\hat{b}_{n-1}} \right\rfloor, B-1\right\}$ with the digits $\hat{r}_i$, $\hat{r}_i - 1$, and $\hat{b}_{n-1}$ obtained from scaling by d (see above). If $\hat{b}_{n-2}\hat{q} > \left(\hat{r}_i B + \hat{r}_{i-1} - \hat{q}\hat{b}_{n-1}\right) B + \hat{r}_{i-2}$, set $\hat{q} \leftarrow \hat{q} - 1$ and repeat this test.
5. If $r - b\hat{q} < 0$, set $\hat{q} \leftarrow \hat{q} - 1$.
6. Set $r := (r_i r_{i-1}\ldots r_{i-n})_B \leftarrow (r_i r_{i-1}\ldots r_{i-n})_B - b\hat{q}$ and $q_j \leftarrow \hat{q}$.
7. Set $i \leftarrow i-1$ and $j \leftarrow j-1$; if $i \geq n$, go to step 4.
8. Output $q = (q_m q_{m-1}\ldots q_0)_B$ and $r = (r_{n-1}r_{n-2}\ldots r_0)_B$.

实现细节：

①uint32_t 的数移动 32 位的结果是不动不是 0，如果要移位 32 位，哪怕是 uint32_t 的也要先换成 uint64_t 进行操作。

②超过 32 位的两个数相乘结果会溢出，超过 64 位。

③做检验时，括号中不会超过 64 位，但是括号中结果乘以 B 就不知道了，可以做检验，检验括号中的数是否比 B(2^{32})大，如果大于 B，不等式右边就比 B*B 大；不等式左边 b_{n-2}*q'小于 B*B，不用做不等式比较，跳过了溢出这种可能性；重复做的时候也是需要这样考虑。

4. Set $\hat{q} \leftarrow \min\left\{\left\lfloor \frac{\hat{r}_i B+\hat{r}_{i-1}}{\hat{b}_{n-1}} \right\rfloor, B-1\right\}$ with the digits $\hat{r}_i$, $\hat{r}_i - 1$, and $\hat{b}_{n-1}$ obtained from scaling by d (see above). If $\hat{b}_{n-2}\hat{q} > \left(\hat{r}_i B + \hat{r}_{i-1} - \hat{q}\hat{b}_{n-1}\right) B + \hat{r}_{i-2}$, set $\hat{q} \leftarrow \hat{q} - 1$ and repeat this test.

④第⑤步判断，可以先执行 r-q*b，如果最后有借位，说明商估计大了，需要把一个 b 加回去，q-1 是真实的商。

5. 取模

5.1 int modn_b(BN a, BN n, BN & rem) rem=a mod n

直接调用除法 div_b(a,n,q,rem)。

5.2 int modadd_b(BN a, BN b, BN n, BN & result);//result=a+b(mod n)

模加，使用 add()加完以后用 modn_b()取模。

5.3 void modsub_b(BN a, BN b, BN n, BN &result);//result = a - b(mod n)

模减，使用 sub()减完以后用 modn_b()取模。

5.4 void modmul(BN a, BN b, BN n, BN & result) //模乘，result=a*b (mod n)

模乘，使用 mul()乘完以后用 modn_b()取模。

6. 信安数学中常用函数

6.1 int gcd_b(BN a, BN b, BN & result);//求最大公因子，result=(a,b)

求公因子，使用欧几里得除法，辗转相除，利用(a,b)=(b,r)，每次执行除法 div_b(at,bt,q,ri)，当 ri 为 0 时，ri-1 就是最大公因子。

6.2 int inv_b(BN a, BN n, BN & x) //求逆

如果(a,n)=1,则有 ax = 1 (mod n)；否则异常没有逆元，返回 0，显然逆元不可能为 0。函数执行步骤如下：

①调用了 gcd_b(BN a, BN b, BN & result)函数，先求公因子判断是否存在逆元，然后才求逆。

②求逆时，对于 inv_b(a,n,x)，初始化 u=1,g=a,v1=0,v3=n.

③用 div_l(g,v3,q,t3)计算带余除法,g=q•v3+t3，令 t1=u-q•v1 (mod n)，u=v1，g=v3，v1=t1，v3=t3。

④如果 v3=0，则把 u 赋值给 x，作为结果，否则回到步骤②。

Extended Euclidean algorithm for calculating $\gcd(a, n)$ and the multiplicative inverse of $a \bmod n, 0 \leq a, 0 < n$

1. Set $u \leftarrow 1, g \leftarrow a, v_1 \leftarrow 0$, and $v_3 \leftarrow n$.
2. Calculate q, t_3 with $g = q \cdot v_3 + t_3$ and $t_3 < v_3$ by division with remainder and set $t_1 \leftarrow u - q \cdot v_1 \bmod n, u \leftarrow v_1, g \leftarrow v_3, v_1 \leftarrow t_1, v_3 \leftarrow t_3$.
3. If $v_3 = 0$, output g as $\gcd(a, n)$ and u as the inverse of $a \bmod n$ and terminate the algorithm; otherwise, return to step 2.

6.3 int modexp_b(BN b, BN n, BN m, BN & result);//模幂运算

①使用模平方算法，配合调用 modmul()模乘函数;
②对指数 n 进行二进制展开，初始化 a=1，b=b，m=m。
③对于 n 的二进制展开，从低到高一位位来判断，b=b*b (mod m)，如果这一位为 1，则 a=a*b(mod n)，否则 a=a;
④最后 n 的二进制位都算完了，a 就是结果。

6.4 int fermat_b(BN a); //用费马小定理检测 a 是不是素数

选的参数是 2/3/5/7，200 位的 a 会出现检测失效，500 位还没有发现检测失效。

先是求(a,2),(a,3),(a,5),(a,7)确认是互素的，然后再求 t^{a-1} mod a，调用 modexp()函数来求，其中 t 依次带入 2/3/5/7，如果模幂结果不是 1，就返回 0（a 是合数），否则返回 1（a 是素数）。

6.5 void crt_b(BN a, BN b, BN p, BN q, BN & result);//中国剩余定理

求模幂，a^b mod(p*q)
①先计算 b1 和 b2：a^b = b1 mod p　　a^b = b2 mod q
②得到方程组　　x=b1 mod p
　　　　　　　　x=b2 mod q
③求逆然后得到结果：

m1=p;m2=q　m=m1*m2;
M1=m2=q　M2=m1=p
M1*M1’=1 mod p　M2*M2’=1　mod q
result= b1*M1’*M1 + b2* M2’*M2　mod(m)

7.位运算函数

7.1 int shl_b(BN a);//左移一位

也是用一个 64 位的 carry 保存中间结果，carry = ((uint64_t)(*aptr) << 1) | (carry >> BITPERDIGIT),每次 carry 为本位左移 1 位同时最右边 1 位也就是第 1 位为上一次左移到的第 33 位。然后再将低 32 位赋给 a 的对应位，进行循环。左移和加法一样最后检查 carry 的第 33 位是不是 1，是的话，再次赋值，总位数加 1。

7.2 int shr_b(BN a) //右移一位

与左移类似，只是会出现有一位移到后面去了，赋值本位的时候需要把最左那位赋值为上次移动到后面的那位。使用中间变量 undercarry。

7.3 uint32_t getbits_b(BN a) //获取 a 的二进制位数

先去掉前导 0，然后剩下的位数就是以 2^{32} 为基数的位数，这个位数先乘 32，得到的总位数并不是最终结果，因为最高位可能没有满，也可能满了。拿最高位 high 和 bit32=80000000U 比较，如果 high 小于它，总位数减 1，high 左移 1 位，进行循环，最后得到的就是真实的位内也不含前导 0 的位数。

8. 特殊操作函数

8.1 void exclu();//求前几十个素数的乘积，存在文件中

这个函数是为了后面找大素数时求 gcd 用的，使用时需要手动进入修改产生对应大素数乘积个数，文件名也要修改，findprime 中对应位置也需要修改。

8.2 int genBN(BN result, int bits);//产生 bits 个二进制位的大数 result

①产生方法为，对于从低到高每一位（一位包含了 32 个二进制位），调用 void writerand(char * addr)函数，往 addr 中写一个随机数，然后调用之前写的 SHA-1 函数，计算 addr 出随机数文件的哈希值，然后取 32 个二进制位(8 个十六进制位，也就是 8 个哈希结果的字符)。
②最高位(最高的可能有 1-32 个二进制位)需要特殊处理，因为最高如果是要 32 位，产生的哈希值第一位如果是 A 以下，如 0-9，那最高就没有 32 位，得到的大数会小于 bits。这种情况下，特殊化处理，多次产生直到产生的是所要的为止。其它时候可以左右移位来满足最高位的需求。

8.3 void writerand(char * addr);//把 rand()产生的随机数写到 addr 中

使用的是 rand()函数，设置了 srand(time)。为了后面对它求哈希准备。

8.4 int findprime(BN a, int bits);//找 bits 位素数 a，30 素数 17 次循环(20-35 素数都还好)

①该函数调用了 genBN(BN result, int bits)函数产生随机数，调用了 int fermat_b(BN a)费马检测函数对大数进行素性检测，同时它利用了 exclu()函数预先产生的前几十个素数的乘积结果，利用了 gcd_b(BN a, BN b, BN & result)函数求公因子。同时需要用到 SETONEBIT_B(BN num, uint32_t u)函数设置大数 num 为一个 uint32_t 的数 u。用到了加法函数 add_b(BN a, BN b, BN sum)。
②过程是：
(1)首先读取前 20 个素数的乘积 fac，第 20 个素数存在 temp3 中；
(2)随机产生满足位数要求的大数 bignum；
(3)求公因子 gcd_b(fac,bignum,gcd)，如果 gcd=1，就拿这个大数去做费马检测步骤(7)。如果 gcd 比第 20 个素数还要大，或者很不幸这个大数是偶数 gcd=2 或者 gcd 比 temp3 大，就回到(2)；

(4)对大数进行微调，初始化微调变量 linshi=2，循环变量 i=0；
(5)大数 bignum 加上 linshi，得到调整后的 adjnum，对 adjnum 和 fac 求最大公因子 gcd。
(6)如果 gcd 为 1，就进行费马检测，步骤(7)，linshi 设为 2。如果 gcd 不是 1，且 gcd<temp3，则 linshi=linshi*gcd，循环变量 i++，回到(5)；否则重新生成一个，回到(2)。
(7)如果费马检测结果为 1，则判定产生的是素数，否则回到(2)。

8.5 `void genpq(char * p_path, char * q_path);//产生私钥p和q`

产生私钥 p 和 q 存在 p_path 和 q_path 文件中。会打印出 p 和 q 的位数。默认 p 是 524 位，q 是 500 位，乘起来大概是 1024 位。如果需要修改位数，进入手动修改。

9. 哈希函数 SHA-1 部分

```
//SHA-1的函数和常数
uint32_t H0 = 0x67452301;
uint32_t H1 = 0xEFCDAB89;
uint32_t H2 = 0x98BADCFE;
uint32_t H3 = 0x10325476;
uint32_t H4 = 0xC3D2E1F0;
const uint32_t K0 = 0x5A827999;
const uint32_t K1 = 0x6ED9EBA1;
const uint32_t K2 = 0x8F1BBCDC;
const uint32_t K3 = 0xCA62C1D6;
void subround(uint32_t & A, uint32_t & B, uint32_t & C, uint32_t & D, uint32_t & E,
uint32_t &W, uint32_t K, int mode);
long long msgsize(char*plainaddr);
inline uint32_t cirleft(uint32_t word, int bit);
int mysha1(char *inputfileaddr, char *output);//每次调用用的是全局的，相当于一个状态向量
int SHA1(char *inputfileaddr, char *output);//每次调用里面会重新初始化向量
int checkresult(char * plainpath, char * decrypath);//检查明文和解密以后是否相同，相同返回 1
```

其中需要注意的地方是，因为 H0、H1、H2、H3、H4 是全局变量，在后面 checkresult()的时候调用的是 SHA-1 而不是 mysha1()。SHA-1 开头重新对初始化向量进行了赋值，而 mysha1 是没有赋值的，相当于每次调用 mysha1()它的初始化状态是由前一次决定的,这样产生的随机数用来寻找素数比较快，而调用 SHA-1()找素数就非常慢，运气好几十秒，运气不好跑几分钟都没有出来一个。

10. 字符串处理部分

10.1 `string bn2str(BN bignum);//bignum转化为字符串形式返回`

返回的不显示前导 0，和正常预期是一样的，可以吞下 BND 然后吐出字符串。内部的缓冲区 char strbignum[520]，如果是 265 大小，则只能吞下 BN。需要检查的是，到底高位的位内有没有前导零，

有的话需要用字符串处理的方式去掉，中间“位”有零也必须显示，而高位为了方便，不显示 0，只显示有效数字。

10.2 `int str2bn(BN & bignum, string strbn);//字符串形式的转化为大数`

只能转化为 BN，转的时候，倒着转，先找到字符串的末位，然后每次取 8 个字符，拼成 32 位，直到到了最高位如果剩余的不够 8 个字符，再拼出最高位。

10.3 `int readbn(BN &bignum, string filename)//从文本中读取大数`

从文本中读取字符串以后调用 str2bn()。

10.4 `int writebn(string filename, BN bignum)//写入大数到文本`

调用 bn2str()把大数转为字符串，然后写入到文本。

四、参考文献

实现时，格式一开始参考了《密码学 C/C++语言实现》中的格式，经典的算法参照《信息安全数学基础》及《密码学 C/C++语言实现》进行了实现。

[1]陈恭亮.信息安全数学基础(第2版)[M].北京：清华大学出版社.2014：7-8，；20-37,76-78,80-82,97-99,198-201.
[2]迈克尔·威尔森巴赫.密码学C/C++语言实现(第2版)[M].北京：机械工业出版社.2016：7-9,12-17,21-23,43-46,78-79,112-113.
[3]Knuth.计算机程序设计艺术 第二卷(第3版)[M].北京：国防工业出版社.2009：245-247