**[摘要]** 大数运算不仅仅运用在密码学中，还运用在一些物理学研究、生物学，化学等科目中。大数运算，意味着参加的值和计算结果通常是以上百位数，上千位数以及更大长度之间的整数运算。例如大家所熟知圆周率π的值，在一般的数值计算中用到圆周率的不须要多大的精度，但在计算一些星球或是星系上的体积面积时便显的误差很大了，这就要求π值计算的精度达到几百万位甚至更高，才能缩小误差。人工计算是远远不行了，而且本身误差也无法估计。只有在计算机中用大数运算求π值了。又如，考古学家计算石头内的碳元素衰变来考证地球形成的时间，更是将计算的结果精确到了百年以内。所以说大数的运算是涉及领域多，应用范广，与我们生活息息关。在此，我采用一个在C语言下实现计算大数运算的一个程序为例，讲解包括了大数的加法，减法，乘法和除法及求幂运算的算法及代码。  
  
**[关键词]** 大数计算 网络安全 密码学  
     随着计算机网络技术的发展和因特网的广泛普及，网络安全事故逐年增加，黑客的攻击已经和病毒并列成为对信息安全影响最严重的两大危害。其很大程度上是被黑客破解了用户的计算机名及登陆密码及资料的加密较差，而使得黑客来对网民的资料如同自己般的随意更改和破坏。而安全的密码和账号成为了网民的安全之本，怎么才能提高安全问题成为的人们和社会关注的问题。而加密大部又是以大素数的计算为基础的，如非对称密码体制RSA的安全性依赖于对大数进行因数分解的耗时性。一个二进制数n的因数分解所需的机器周期大约是exp{[ln(n)ln(ln(n))]1/2}。若机器周期为1μs，b为二进制数的位数，分解 n=2b 所需时间如下表所示：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 位数 | 100 | 200 | 300 | 500 | 750 | 1000 |
| 时间 | 30秒 | 3天 | 9年 | 1兆年 | 2\*109年 | 6\*1015年 |

     实际应用中，p、q的选择都在10200 数字以上，这样每个明文块的二进制位数可达664比特，即83个字节。而DES只有8个字符。一般认为，对于当前的计算机水平，选择1024位长的密钥就可认为是无法攻破的了。  
  
  
**1. 数字存储的实现**  
     大数计算的因数和结果精度一般是少则数十位，多则几万位。在C语言中定义的类型中精度最多只有二十多位，因而我们采取用链表存贮的方式来存放大数。在计算中会用到从高位开始计算，和从低位开始计算数值的两种情况。所以我们将链表定义为双向链表，其中为一个单元来存贮数据，一个指针指向前方的数据，另一个指向后的数据。其结构为：

|  |
| --- |
| struct Node     // 定义一个双向链表用来存贮结果 { char data;      // 数据\*定义为字符的原因：要显示负号 Node \*next;     // 尾指针 Node \*ahead;    // 首指针 }; |

**2. 各种算法**  
2.1 加法运算的实现  
     加法计算还是比较容易的，我们也是先从低位算起，因为只须要对应的位相加，再加上前一位的进位，再去判断是否本位是否有进位， 有则把本位数字改为减去它的权，也就是10，再置进位为1。如果没有进位，则给进位赋值0。

     在程序时应注意：  
     1、两个加数中那一个数的位数长，以位数长的作为循环变量；  
     2、结束循环时，不仅仅是最后一位加完就停止，还应加入如果有进位，也要再循环一次。如最后一位是9，进位是1，则相加时进位，要加上进位这一位值。具体看代码，输入输出时和乘法的一样。

|  |
| --- |
| /\*-------------------------------------------------------------------------- \*函数名称： 大数加法 \*函数过程：1 比较两个数那一个长                   \* 2 以长的作为循环次数                  \* 3 对应项相加 进位存贮直到下高位相加用                  \* 4 直到循环结束                  \* 5 !!!!!!没设计负数相加 \*入口参数：numa，numb，result字符串 \*出口参数：无 \*编辑环境：winSP2 + VC2003 + C++ \*--------------------------------------------------------------------------\*/ void addition(char \*numa, char \*numb,char \*result) // 计算两大数之和 {    char \*pna = findend(numa); // 指向numa的一个指针。point numa pna 指向乘数的最低位，    char \*pnb = findend(numb); //指向numb的一个指针 //pnb 指向被乘数的最低位，    int along=(int)strlen(numa); //标记数字a的长度；    int blong=(int)strlen(numb); //标记数字b的长度；    int times = 0; // 标致要计算多少次。    int carry=0,temp\_result; //存贮进位 和临时结果的     Node \*head, // 用于存贮头指针      \*pstart, // 用于存贮计算时的首指针      \*pnew; //作于申请新结点     head = pstart =new Node; //初始化首结点和头结点。    pstart -> data = 0;    pstart -> next = NULL;    pstart -> ahead = NULL;      if (abigerb(numa ,numb)>=1)    times = (int)strlen(numa); //比较两个字符串长度，以大的作为循环次数    else      {        times = (int)strlen(numb);        pna = findend(numb); //交换指针        pnb = findend(numa);      along=(int)strlen(numb); //标记数字a的长度；      blong=(int)strlen(numa); //标记数字b的长度；      }      while ((times-- && (times>=0))|| carry != 0)      {         if(!pstart->next) //如果当前为空结点，则申请新结点        {          pnew = new Node;          pnew -> data = 0;          pnew -> next = NULL;          pnew -> ahead = pstart;          pstart -> next = pnew;        }        else temp\_result =(pstart->data +(\*pna-48)+(\*pnb-48)+carry) ; //自身的值+新值+进位 作为当前的新值         pstart -> data = temp\_result%10; //存贮个位        carry = temp\_result/10; //存贮进位        pstart = pstart -> next; //结点移动      blong--;        if(blong>0)pnb--; //指针移向被加数高位        else \*pnb=48; //之后相减就变为了0不作任何运算；        pna--; //加数指针移动，      }    pstart =head; //寻找链表的结尾点    while(pstart->next != 0)    {    pstart->data += 48; //!!<<<因为我们的输出是字符。所以再此加上48>>>> 逆顺输出    pstart = pstart->next ;    }     int tip = 0; //转为字符串用      pstart = pstart->ahead ; //找有效字      //cout<<"\n结果是 : ";      while(pstart != 0) //输出正序的结果；      {      result[tip++] = pstart->data;       //cout<data;      pstart = pstart->ahead ;      }      result[tip] = '\0';     pstart =head; //释放空间    while(pstart->next != 0)    {    pnew = pstart->next ;delete pstart;    pstart =pnew;     }    return ; } |

2.2 减法运算的实现  
     减法稍微有一点复杂， 因为会处理负数，而我们所用的是字符串的形式来保存数字也是为这一点。否则用整型或其它类型时，则表示处理、保存时会相当复杂。算法也是从低位开始减。先要判断减数和被减数那一个位数长，减数位数长是正常减；被减数位数长，则被减数减减数，最后还要加上负号；两数位数长度相等时，最好比那一个数字大，否则负号处理会很繁琐；处理每一项时要，如果前一位相减有借位，就先减去上一位的借位，无则不减，再去判断是否能够减开被减数，如果减不开，就要借位后再去减，同时置借位为1，否则置借位为0。

     这有一我写的比较两个数字符串大小的函数：

|  |
| --- |
| //返回值说明:0是alongblong ; 2是along=blong int abigerb(char \*numa, char \*numb) //比较两个数最高位那一个大 {    int along=(int)strlen(numa); //标记数字a的长度；    int blong=(int)strlen(numb); //标记数字b的长度；    char \*pna = numa; //指向数A的最高位，    char \*pnb = numb; //指向数B的最高位，    if (along>blong) return 1;    if (along==blong)    {      while(\*pna) //比较两个数那一个大      {        if(\*pna>\*pnb)return 1;        else if(\*pna<\*pnb)return 0 ;        else if(\*pna==\*pnb){pna++;pnb++;} //1111与1112       }      return 2; //这表明找到最后了还是全相等;    }    return 0 ; } |

     在程序时应注意：  
     1、如果被减数大于减数时，指针和相应循环长度控制变量也要做相应的修改。  
     2、结果可能会出现前面是一堆0的情况，要处理好，如当减数为11112，而被减数为11111时，会出现00001，处理过程见代码。

|  |
| --- |
| /\*-------------------------------------------------------------------------- \*函数名称： 大数减法 \*函数过程：1 比较两个数那一个长 \* 2 以长的作为循环次数 \* 3 如果两个数长度相等，从最高位开始比直到发现那一个数更大，使大项减去小项 \* 4 对应项相减 进位存贮直到下高位相加用 \* 5 每一位对应项相减时，处理三种可能的情况，a=b,ab; \* 6 a=b时，则计算,11-12,111-112,要考虑借位 \* 7 直到循环结束 \*入口参数：numa，numb，result字符串 \*出口参数：无 \*--------------------------------------------------------------------------\*/  void subtract(char \*numa, char \*numb,char \*result)//计算减  {  char \*pna = findend(numa);//指向numa的一个指针。point numa pna 指向减数的最低位，  char \*pnb = findend(numb);//指向numb的一个指针 //pnb 指向被减数的最低位，  int along=(int)strlen(numa);//标记数字a的长度；  int blong=(int)strlen(numb);//标记数字b的长度；  int times = 0; // 标记要计算多少次。  int carry=0; //存贮借位  int clear0=0; //消除结果最前面无用的'0' 13-5 = 08 的效果！！  int isnegative=0; //用来加上被减数大于减数时补上最后一个负号  Node \*head, // 用于存贮头指针  \*pstart, // 用于存贮计算时的首指针  \*pnew; //作于申请新结点    head = pstart =new Node;//初始化首结点和头结点。  pstart -> data = 0;  pstart -> next = NULL;  pstart -> ahead = NULL;  if (abigerb(numa ,numb))  times = strlen(numa);//比较两个字符串长度，以大的作为循环次数  else //交换位置以降低难度  {  times = strlen(numb);//让数（字符串）长的减去数（字符串）短的  pna = findend(numb);//交换指针  pnb = findend(numa);  along=(int)strlen(numb);//标记数字a的长度；  blong=(int)strlen(numa);//标记数字b的长度；  isnegative=1;//标记最后要加上负号  }  while ((times-- && (times>=0))|| carry != 0)//carry != 0 说没有借位时  {  if(!pstart->next)//如果当前为空结点，则申请新结点  {  pnew = new Node;  pnew -> data = 0;  pnew -> next = NULL;  pnew -> ahead = pstart;  pstart -> next = pnew;  }  if(times<0)//如果计算完之后，借位等于1，，说明必然为负值;  {  pstart -> data = -3 ;//让它等于负号 '-'//-3来源于负号与0相差3。。 break;  }  else  {  if ( \*pna == \*pnb )//减数等于被减数时。结果等于直截相减的结果；并置借位为0  {  if(carry==0)pstart -> data = (\*pna-48)-(\*pnb-48); //111-11的情况  else  {  pstart->data = (\*pna-48)-(\*pnb-48)+10 -carry;//1121-1112  carry=1;  }  }  if( \*pna > \*pnb )//减数大于被减数时。结果等于直截相减的结果；并置借位为0  {  pstart -> data = (\*pna-48)-(\*pnb-48)-carry; //存贮个位  carry=0;  }  else if( \*pna < \*pnb )//说明被减数大于减数，让结果加10，相当于借位 (carry)为1  {  if(times>0) pstart->data = (\*pna-48)-(\*pnb-48)+10 -carry;//13-5的情况作为新值  else pstart->data = (\*pnb-48)-(\*pna-48) -carry; //3-5 作为当前的新值  carry=1;  }  }  pstart = pstart -> next; //结点移动  blong--;  if(blong>0)pnb--;//指针移向被减数高位  else \*pnb=48;//之后相减就变为了0不作任何运算，其实可以优化的。但代码会长！而且还需要重新开结点。所以放弃；  pna--;//被数指针移动，  }  if(isnegative==1)////加上负号处理。增加一长度并置为负号  {  pnew = new Node;  pnew -> data = 0;  pnew -> next = NULL;  pnew -> ahead = pstart;  pstart -> next = pnew;  pstart->data=-3;//因为寻找链表的结尾点要统一加48。又因为‘-’是45。所以等于‘-3’  }  pstart =head;//寻找链表的结尾点  while(pstart->next != 0)  {  pstart->data += 48;//!!<<<因为我们的输出是字符。所以再此加上48>>>> 逆顺输出  pstart = pstart->next ;  }  int tip = 0;//转为字符串用  clear0=0;// 消除结果最前面无用的'0' 13-5 = 08 的效果 ..>>修改字符串的首指针  pstart = pstart->ahead ;//找有效字  while(pstart != 0)//输出正序的结果；  {  if (clear0==0 && ((int)pstart->data)==48&&pstart->ahead!=0)// 消除结果最前面无用的'0'  ;//不输出任何东西  else  result[tip++] = pstart->data;    if(((int)pstart->data)!=48&&((int)pstart->data)!=45)clear0=1;//'－'号  pstart = pstart->ahead ;  }  result[tip] = '\0';  pstart =head; //释放空间  while(pstart->next != 0)  {  pnew = pstart->next ;delete pstart;  pstart =pnew;  }  return ;  } |

2.3 乘法运算的实现  
     首先说一下乘法计算的算法，从低位向高位乘，在竖式计算中，我们是将乘数第一位与被乘数的每一位相乘，记录结果，之后，用第二位相乘，记录结果并且左移一位，以此类推，直到计算完最后一位，再将各项结果相加。得出最后结果。当然我们可以直接用这种方法，但要用多个链表来保存计算出的分结果， 之后结果再相加得到最后结果，但是这样会浪费很多空间，我们可以再优化一下，就是只用一人链表来表示结果，先把第一位乘数与被乘数的结果保存在链表中，之后把存储结果的头部后移一位、也就是从链表的第二加起，当第二位乘数与被乘数结果加到第二之后的各个项内。以此类推，直到结束。这样就可以用一个链表来存储相乘后的结果。

     在程序时应注意：  
     1、传入的乘数和被乘数是以字符串形式放入的话，要让指针指向最后一位，我自己写了个函数来完成这件事。链表传入也要找到最后一向来计算；  
     2、因为传入和保存的都是字符，所以计算时要将字符转化为数字，算完再转化为字符存储；  
     3、另外进位时要处理，当前的值加上进位的值再看本位数字是否又有进位；  
     4、输出时要逆序输出，因为链表首指针是存的结果的最低位。我的函数为了对应输入输出结果的一致性，都有采用了字符串输出，所以在输出时又将链表转为了字符串。  
     具体代码如下：

|  |
| --- |
| /\*-------------------------------------------------------------------------- \*函数名称： 大数乘法 \*函数过程：1 输入两个大数作为字符串 \* 2 作一个双向链表 \* 3 两个指针分别指向数字字符串的最低位 \* 4 以第一个数的最低的一个位乘以第二个数的所有项存于链表中 \* 5 链表首指针移 \* 6 重复4，5依次从最低位乘到最高位 \* 7 乘完后因为最低位是链表首，最后一位是链表尾。所以在逆顺输出链表。 \* 4 直到循环结束 \*入口参数：numa，numb，result字符串 \*出口参数：无 \*--------------------------------------------------------------------------\*/  void multiply(char \*numa, char \*numb ,char \*result)//用来储结果的)//计算乘积  {  char \*pna = findend(numa);//指向numa的一个指针。point numa pna 指向乘数的最低位，  char \*pnb = findend(numb);//指向numb的一个指针 //pnb 指向被乘数的最低位，  int along=(int)strlen(numa);//标记数字a的长度；  int blong=(int)strlen(numb);//标记数字b的长度；  int carry=0,temp\_result;//存贮进位 和临时结果的  Node \*head, // 用于存贮头指针  \*pstart, // 用于存贮计算时的首指针  \*pnew, //作于申请新结点  \*pgo; //作为每计算完一行时，回到下一行起始节点用,移位标致来用  head = pstart =new Node;//初始化首结点和头结点。  pstart -> data = 0;  pstart -> next = NULL;  pstart -> ahead = NULL;  while (along--)  {  pgo = pstart;//保存进位点  blong = (int)strlen(numb);//初始化长度  pnb = findend(numb); //初始化指针  while ((blong-- && (blong>=0))|| carry != 0)  {  if(!pstart->next)//如果当前为空结点，则申请新结点  {  pnew = new Node;  pnew -> data = 0;  pnew -> next = NULL;  pnew -> ahead = pstart;  pstart -> next = pnew;  }  if(blong<0)temp\_result = carry ;//处理只有进位的情况  else temp\_result =(pstart->data+(\*pna-48)\*(\*pnb-48)+carry);//自身值+新值+进位作为新值  pstart -> data = temp\_result%10; //存贮个位  carry = temp\_result/10; //存贮进位  pstart = pstart -> next; //结点移动  pnb--; //指针移向被乘数高位  }  pstart = pgo->next; //前进一个位置；  pna--; //指针移向乘数高位  }  pstart =head;//寻找链表的结尾点  while(pstart->next != 0)  {  pstart->data += 48;//!!<<<因为我们的输出是字符。所以再此加上48>>>> 逆顺输出  pstart = pstart->next ;  }  int tip = 0;//转为字符串用  pstart = pstart->ahead ;//找有效字  while(pstart != 0)//输出正序的结果；  {  result[tip++] = pstart->data;  pstart = pstart->ahead ;  }  result[tip] = '\0';  pstart =head; //释放空间  while(pstart->next != 0)  {  pnew = pstart->next ;delete pstart;  pstart =pnew;  }  return ;  } |

2.4 除法运算的实现  
     首先说一下我们所要的结果，当除数除不开被子除数时，不用除到小数，当除数小于被除数时，除数作为余数既可，不用再向下除了。  
     除法算法是最容易想到的啦！我们学数字电路，或模拟电路时里面有用门实现的除法器，做法是用除数减被除数，再用一个加法器去计算存储结果，算法简单明了，结果我实现之后，计算速度大大出乎我的遇料，比算乘方时还慢，经仔细想想电路中计算的长度不过8位，16位，32位，而且都是硬件实现，我们要算的是几千位，几万位。特别是两数位长度差很大时计算速度相当慢，如10000-3时要进行多少次链表运算啊。  
     否定了这条算法，经不断思考，发现另一种方法可行，且效率仅次于减法。就是从高位向低位减，减时以被除数长度为单位，从高位取出大于被除数的字符串，和被除数相减，减的次数为结果，余数从剩下的除数高位再取出几位做补位，直到大于被除数。再减，循环减到被减数大于余数加上补位，那么这个新的余数作为结果返回。

     在程序时应注意：  
     1、除法算法计算时是用的最高位开始向低位减，所以要注意指针的位置。  
     2、余数和被减数相减时，一定要注意：一旦能够减开被除数时，一定要每从除数那里取一个字符时，结果也要对应补一个0。如111222/111。

|  |
| --- |
| /\*-------------------------------------------------------------------------- \*函数名称： 大数除法2--  \*函数想法：1 用指针指向除数的最高位，保存临时字符串； tempstr[a++] = pna \* 2 如果临时字符串大于被除数，则相减。结果等于余数 \* if(tempstr>numb)tempstr = tempstr - numb \* 3 如果小于被除数且指针到头，余数 等于 临时字符串 \* if(tempstr \*  \*入口参数：numa，numb，result,remainder字符串 \*出口参数：无 \*--------------------------------------------------------------------------\*/  void divide2( char \*numa, char \*numb,char \*result,char \*remainder)//计算除法2  {  char one[]="1";//临时字符串....  char one2[]="1";//  char zero[]="0";//  char numb2[6048];//  char tempstr[6018]="";//临时字符串  int ia=0,ia2=0;//tempstr的指示器  bool moveon=false;//翻转牌  char \*pna = numa;//指向numa的一个指针。point numa pna 指向减数的最低位，  char \*pnb = findend(numb);//指向numb的一个指针 //pnb 指向被减数的最低位，  Node \*head, // 用于存贮头指针  \*pstart, // 用于存贮计算时的首指针  \*pnew; //作于申请新结点    head = pstart =new Node;//初始化首结点和头结点。  pstart -> data = 0;  pstart -> next = NULL;  pstart -> ahead = NULL;  moveon = false;  while(\*pna)  {  if(!pstart->next)//如果当前为空结点，则申请新结点  {  pnew = new Node;  pnew -> data = 0;  pnew -> next = NULL;  pnew -> ahead = pstart;  pstart -> next = pnew;  }    ia=(int)strlen(tempstr);//取的长度  tempstr[ia++] = \*(pna++);    tempstr[ia] ='\0'; //转换为字符串  if(tempstr[0]=='0')//处理高位也是0的那种 如00  {  ia2=0;  while(tempstr[ia2]=='0')++ia2;  while(ia2>=1)//清除无用的0  {  ia=ia2-1;  tempstr[ia]=tempstr[ia2];  --ia2;  }  tempstr[++ia2]='\0';  }  while(abigerb(tempstr,numb)>0)//如果tempstr大于等于numb  {  if(tempstr[0]=='0')//处理高位也是0的那种 如00----此乃冗余代码，留做记念用  {  ia2=0;  while(tempstr[ia2]=='0')++ia2;  if(ia==ia2 )  {  moveon = true;  break;  }  }  strcpy(numb2,numb);  subtract(tempstr, numb,tempstr);//A-B  strcpy(numb,numb2);  if(tempstr[0]=='-')//若判断的不准，变为了负数就再加上B。。  {  strcpy(numb2,numb);  addition(tempstr, numb,tempstr);//A-B  strcpy(numb,numb2);  ia2=0; //修正之后的长度。因为加法中未做负数运算  ia=0; //为了消除最后的那一个负号，整体向前移动。  while(tempstr[ia2]!='\0')++ia2;  while(ia2>=1)//清除无用的0  {  tempstr[ia]=tempstr[ia+1];  ++ia;  --ia2;  }  tempstr[ia]='\0';  moveon = true;  break;  }  pstart->data++ ; //结果自加  moveon = true;  }  if(moveon) pstart = pstart -> next; //结点移动  }  strcpy(remainder,tempstr);//存贮余数  int tip = 0;//转为字符串用  pstart =head;//寻找链表的结尾点  while(pstart->next != 0)  {  pstart->data += 48;//!!<<<因为我们的输出是字符。所以再此加上48>>>> 逆顺输出  result[tip++] = pstart->data;  pstart = pstart->next ;  }  result[tip] = '\0';//存贮结果  pstart =head; //释放空间  while(pstart->next != 0)  {  pnew = pstart->next ;delete pstart;  pstart =pnew;  }  return ;  } |

2.5 幂运算的实现  
     幂的实现是最为简单的了，国为有了前面的算法做铺垫，就是调用乘法函数，来循环去自乘，幂指数相应减1，直到幂指数变为0时结束。其详细代码如下：

|  |
| --- |
| /\*-------------------------------------------------------------------------- \*函数名称： 大数幂 \*函数想法：1 B自减1直到，，作为循环控制变量.保存结果；  \* 2 结果初值1 每次乘以A \* 3 重复1、2步骤到结束.  \*入口参数：numa，numb，result字符串 \*出口参数：无 \*--------------------------------------------------------------------------\*/ void power (char \*numa, char \*numb,char \*result) //计算幂 { char one[]="1"; //临时字符串.... char one2[]="1";  char zero[]="0";  char numb2[6048];  char remainder[6048];  strcpy(result,one); //给结果初值为1 strcpy(numb2,numb);  subtract(numb,one ,remainder); //B自减1 strcpy(numb,numb2);  strcpy(numb2,numb); multiply(result,numa,result ); //A自己乘自己 strcpy(numb,numb2);   while(strcmp(remainder,zero)>0) //A大于相等B时 就是numb不等于0时 { strcpy(numb2,numb); multiply(result,numa,result ); //A自己乘自己 strcpy(numb,numb2);  strcpy(one,one2); subtract(remainder,one ,remainder); //B减一 }  return ; } |

**3. 程序运行介面**  
     程序的介面制作就不多加描述了，以下是我的大数程序的运行介面。

**4. 总结与评价**  
     1) 本文主旨是对算法加以了实现，但未做相应的优化。因为本文的目地是对大数运算的过程产生一个具体设计方法与实现思路。而还是不是工程上能所直接应用的。当然在要作中是也许用的设计语言并不是C语言，可能是JAVA、 C++、 Delphi 、PHP等等。但不管是什么样的设计语言，设计思路都是相同的。  
     2) 关于代码和算法优化问题，本文中代码并不是简短，高效。现在对此说明个人认为在工程要改正的几个方面。首先每个链表节点的元素所存的数据小的可怜，只存储了一位数据，但这是最容易想到的，模仿十进制那样进位。但这并没考虑机器的计算的情况，在我们常用的32位计算中，CPU中加减乘除的一次运算是32位的值，也就是说2的32次方的一个值，这就是说1加上1的CPU工作量和小于2的32的两位数相加是用的相同的周期。如下图所示:

     为避免运算结果大于2的32次，因为大于的话又会造成一次CPU周期运算，但这样当然也可以做。但建议用下面方法，考虑乘法的原因，两个2的16次方相乘就是2的32次方的情况，所以我们可以定义每个链表节点值为2的16次方，也就是 18446744073709551616的一个整数值，C++语言中\_int64类型的范围就是这么大。再把它当作以前处理十进制的权10去处理，效率会翻2的16方倍了。结点数巨减，主要是开内存方面，有了极大的减少。  
     3) 另外本程序中是用数组去传递参数的，这非常不好，而且会限制你结果的长度。只有一点好处，易懂！个人认为工程中，一定要用链表指针来传递参数， 这样就解决了可算到无限长的问题。只是释放链表时要在最后用完再释放了。对于本文用的方法就显得明了多了。  
     4) 总体来说本文重在说明方法，效率较差。工作时定要注重效率问题。在C++系列下的程序尽量多用库函数，会减少大量的代码。适当的去加入些汇编以提高效率。  
  
**5. 软件下载演示**  
[点击下载大数运算软件](http://adultedu.ustc.edu.cn/img/pap/wh/wh.rar" \t "_blank)  
  
**6. 参考文献**  
1. 李艺《网络安全讲义》PPT文档  
2. 谭浩强《C程序设计》第二版,清华大学出版社  
3. Adam Drozdek《数据结构与算法》第三版,清华大学出版社  
4. 何德全《网络安全》清华大学出版社  
5. http://www.tup.com.cn  
http://adultedu.ustc.edu.cn/img/pap/wh/