《程序设计进阶与实践》实验报告

实验名称：梅森质数判定

姓名：吴越 学号：PB21000004 日期：2022.3.21

实验环境：CPU :Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz;内存：4.00 GB;操作系统：Windows 10;软件平台：VS code

1.问题分析与求解思路：

问题：对于1000以内的素数p，判定2^p-1是否为素数

这是一个复杂的大问题，分析这个问题时，可以将这些大问题分解成4个子问题，再分别处理。
拆解可知解决这个问题主要分4步：（伪代码）
（1）产生1000以内的素数p。
（2）对每个素数p，计算出2^p-1的值。
（3）对每个2^p-1做素性测试。
（4）将是素数的p与2^p-1打印输出。
其中难点集中在(2)，(3)两个模块。

子问题1：产生1000以内的素数p较为简单，只需要定义一个判断素性的函数，再对1000以内的数进行遍历， 将判断为素数的数用一个prime数组存储即可。

子问题2：要计算出2^p-1的值，首先做一个估计。最大的素数为997，$2^{997}-1$的位数达到了$\lg(2^{997}-1)$约为300位。因此我们应当使用数组来储存大数。然后我们考虑到如果采用每次乘2的方法，则要乘996次，效率显然太低。为了提高速度，我采用了快速幂(请参考2次幂运算函数)的算法来求取2^p-1，为此还需要定义大数乘法等函数。最后将结果储存在Mesen数组中即可。

子问题3：难点在于素数判定，对于大数而言，使用普通的遍历筛选法显然效率过低。因此需要采用其他算法。在这里我采用了$Lucas-Lehmer$检验法(请参考$Lucas$素性测试函数)用于素性检测。为此要产生一连串的Lucas数列。但当项数n很大时，该数列的通项远大于Mesen素数本身，用数组储存会有越界问题。由于我们只需要它的余数，因此在每次运算时都要对其做取模运算防止其越界，为此还需要定义大数取模运算(请参考二分试除法算法)等函数。最后若$Lucas-Lehmer$余数为0，则可判定为素数，反之亦然。

子问题4：输出模块也较为简单，在经过素性测试后，将满足条件的p与2^p-1打印输出。由于是采用数组储存大数，因此采用与输出数组各项元素的方法即可。

2.核心代码说明：

(1)2次幂运算函数

```
void TwoPower(ll * a, ll p)             //计算2的p次方
{
    ll twoPow[LENGTH] = { 2 };              //用于储存2^i
    ll result[LENGTH] = { 1 };              //结果初始化为1
    while (p > 0)
    {
        if (p & 1)                          //判断位运算i次后最后一位是否是1
            MulLarge(result, twoPow);       //若为1，则说明已经得到2^i,累乘入结果
        p >>= 1;                            //每次右移一位
        MulLarge(twoPow, twoPow);           //计算2^i
    }
    memcpy(a, result, LENGTH * sizeof(ll)); //将结果copy到a上，完成计算
}
```

算法分析：我们考虑p的二进制表示，特殊的如10010.....则$2^p = 2^{(2^0)} \times 2^{(2^3)} \times ......$。而$2^{(2^3)} = ((2^2)^2)^2$ 即2经过3次平方。再由特殊情况化归到一般情况：因此我们可以通过位运算判断1在第i位，然后2经过i次平方得到2^i。再将所有2^i累乘即可得到结果。可以计算，这样的算法时间复杂度为$O(log\ n)$。比乘p次2的算法效率更高。

(2)$Lucas$素性测试函数

```
void LucasTest(ll p, ll * m, ll * n)
{
    ll lucas[LENGTH] = { 4 };                   //储存Lucas数列
    if (p == 2)
        memset(m, 0, LENGTH * sizeof(ll));      //p=2时是素数，赋0
    else {
        for (int i = 0; i < p - 2; i++) {
            MulLarge(lucas, lucas);             //平方操作
            lucas[0] -= 2;                      //-2得到下一个数
            BinaryDivide(lucas, n);             //每次都进行取模运算，防止越界
        }
        memcpy(m, lucas, LENGTH * sizeof(ll));//将结果copy到main中的Lucas数上
    }
}
```

算法分析:
首先简要介绍一下$Lucas-Lehmer$检验法:定义Lucas数列$\{L_n\}:L_0=4,L_n=L_{n-1}^2-2,\ n>0$。那么可以在数学上 证明 梅森素数$M_p=2^p-1$为素数当且仅当 $L_{p-2}\equiv 0(modM_p)$成立。因此，我们通过计算出Lucas数列的第$p-2$项，再对M_p做取模操作。若余数为0，则可以判断 M_p是素数，反之亦然。

然而我们要注意到，当p很大时，L_{p-2}远大于梅森素数M_p本身，有数组越界的风险，而且考虑到我们只需要结果的余数，L_{p-2}运算中的其他各位属于无效信息，大大降低了我们程序的运行效率。因此，我们应当对每次得到的L_p对M_p做取模处理，这样操作既能限制L_p的大小防止越界，又可以显著降低无效信息的含量，进而让我们程序的运行效率更高。

(3)二分试除法算法

```
for (i = 0; i <= times; i++) {//开始二分试除法
    if (NumberCmp(a, b + i, Len(a), Len(b) - i) > 0) {//若a比b小，不用试除
        int left = 0, right = DIGIT - 1;
        int mid = (left + right) / 2;
        while (left <= right)                       //类似二分查找
        {
```

```
            MulSmall(b + i, mid, i);
            if (NumberCmp(a, b + i, Len(a), Len(b) - i) == 0) {
                memcpy(b, changedb, LENGTH * sizeof(ll));//将b的值初始化
                break;                                  //整除，直接跳出
            }
            else if (NumberCmp(a, b + i, Len(a), Len(b) - i) > 0) {
                left = mid + 1;
                mid = (left + right) / 2;               //缩小区间范围
            }
            else {
                right = mid - 1;
                mid = (left + right) / 2;               //缩小区间范围
            }
            memcpy(b, changedb, LENGTH * sizeof(ll));//将b的值初始化
        }
        MulSmall(b + i, mid, i);                        //得到的mid为最大除数
        Substract(a, b + i, Len(a));                    //进行减法操作
    }
    memcpy(b, changedb, LENGTH * sizeof(ll));
}
```

算法分析：这里我们主要讨论取模运算函数中的算法：二分试除法。设两个大数分别为a,b。在一般的大数除法函数中，通常是使用平移减法，即将b扩大至与a位数相同，再循环做减法运算，将b缩小直至得到结果。但是在本题中，由于采用了压位操作，设一位中储存的数据为D进制，那么可以看出，每次进行的减法运算最大有可能达到$D-1$次(如$\overline{(D-1)}-1$))当D较大时，运算次数过多，效率低。

因此我们在这里采用二分试除法：将b扩大至与a位数相同，利用数的单调性，采用类似于二分查找算法的方法，逐步缩小区间，直至找到a能减去最多多少个b，再进行一次减法运算即可。由于二分查找算法时间复杂度为$O(log\ D)$，当D的值较大时远小于$O(D)$，因此我们在这里采用二分试除法，可以有效提高取模运算的效率。

3.测试，运行与分析：

多次运行程序后，输出结果都为：

```
1.      p=2
3
2.      p=3
7
3.      p=5
31
4.      p=7
127
5.      p=13
8191
6.      p=17
131071
7.      p=19
524287
8.      p=31
2147483647
9.      p=61
2305843009213693951
10.     p=89
618970019642690137449562111
11.     p=107
162259276829213363391578010288127
12.     p=127
170141183460469231731687303715884105727
13.     p=521
6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151
14.     p=607
531137992816767098689588206552468627329593117727031923199444138200403559860852242739162502265229285668889329486246501015346579337652707239409519978766587351943831270835393219031728127
```

一共输出了14个梅森素数，在网络上查表(参见梅森素数)对照，比较得程序运行结果正确。
在运行过程中，可以发现程序在输出127后有较长时间停顿，这是因为127后的Mesen数为512，中间所隔数较多。此外在运行至607后也出现较长时间停顿，这是因为607为1000内最后一个Mesen数，且之后的数数值较大，较之前的数需要花费更多时间，属于正常现象。
综上所述程序运行正常，可以判定程序完成了实验任务。

4.备注

1.关于压位数的选取与数组的长度：

一般情况下，我们使用数组的一位来储存一个十进制数。但是数组一位可以储存一个字节的数据，若只储存一个十进制数显然是对空间的巨大浪费。因此我们一位存储一个D进制的数，这样就可以充分利用数组空间(也即压位)。但是压位数不能选取过大，否则一会造成越位，二会导致二分试除法时间过长。数组长度也要适中，否则有的空间没用到遍历时产生浪费。因此选用合适的压位数的选取与数组的长度才能使效率最高。因此本实验还有改进空间。

2.关于素数判定的其他素性测试：

在本实验中，我们采取了$Lucas-Lehmer$检验法，但是这一判别法只针对梅森素数有效，局限性较大。对于一般的素数判定，我们还可以采用$Miller-Rabin$素数测试算法(主要利用费马小定理与二次探测)，这适用于更一般的数的素性判定。

总结：

通过本次梅森素数判定实验，我掌握了：
(1) C语言实现高精度计算的算法。
(2) 多种素性判定的算法与定理。
(3) 提高了编程与debug能力，加强了对C语言的理解认识。
(4) 有了初步进行单元测试的能力。
(5) 提高了分解，抽象，概括问题的能力，提升了计算思维。
当然实验还有许多能改进的地方，比如压位数，数组长度的设置，且程序运行耗时较长，还有一些算法可以优化提高程序运行效率，今后还有很大的学习进步空间。

另附：

源代码文件：Mesen.c

参考资料：

1. https://zhuanlan.zhihu.com/p/95902286
2. https://bbs.emath.ac.cn/thread-15737-1-1.html
3. https://baike.baidu.com/item/%E6%A2%85%E6%A3%AE%E7%B4%A0%E6%95%B0/816141#
4. https://blog.csdn.net/forever_dreams/article/details/82314237