P1045 [NOIP2003 普及组] 麦森数

题目描述

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）

输入格式

文件中只包含一个整数P（1000<P<3100000）

输出格式

第一行：十进制高精度数2^P-1的位数。

第2-11行：十进制高精度数2^P-1的最后500位数字。（每行输出50位，共输出10行，不足500位时高位补0）

不必验证2^P-1与P是否为素数。

输入样例

1279

输出样例

386
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
00000000000000104079321946643990819252403273640855
38615262247266704805319112350403608059673360298012
23944173232418484242161395428100779138356624832346
49081399066056773207629241295093892203457731833496
61583550472959420547689811211693677147548478866962
50138443826029173234888531116082853841658502825560
46662248318909188018470682222031405210266984354887
32958028878050869736186900714720710555703168729087

解析

本题存在三个难点：

1、如何计算结果的长度。

因为本题的位数极多，因此我们的计算并不是完整的计算，所以得到的结果长度也不是真正的长度，我们需要更加合适的算法。观察下面几个例子：

10的位数是2位
100的位数是3位
1000的位数是4位
12345的位数是5位

我们可以推出n的位数 = $floor(\log_{10}^{n})+1$。

因此，我们可以推出$2^{p}-1$的位数为：$floor(\log_{10}2^{p}-1)+1$。

这里我们考虑到2p的末尾只能是2,4,6,8，不可能是其他的数字，因此，我们能够得出2p-1与2p具有相同的位数。所以，我们可以将上面个公式优化成$floor(\log_{10}2^{p})+1$。

利用对数乘法公式$\log_{10}m^{n} = n\log_{10}m$,则有：

$floor(\log_{10}2^{p})+1 = floor(p*\log_{10}2)+1$。

可以表达成如下代码：int(p * log10(2)) + 1

2、最后的五百位。由于本题的数位超大，所以我们无法将其完整的计算出来，必须进行优化

思考这个问题，如果想求A * B的最后两位结果，与什么相关？观察下列计算式：

```
1123  * 47856 = 53742288
 123  *  456 = 56088
  23  *   56 = 1288
```

很明显，想求A*B的后两位，那么我们只需要计算A的最后两位乘以B的最后两位，其余部分可以省略掉。

因此，本题中计算2的p次幂对500取余，我们可以在计算过程中进行优化，即只计算500位以内的乘法。那么如何只计算500位呢？其实很简单，在高精度乘法中，我们只输出最后的五百位即可。

3、高精度的快速幂

快速幂参考《快速幂算法》，高精度涉及到了高精度的乘法和减法。最后别忘了高精度的乘法最多保留500位。

编码

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

using namespace std;
```

```
//输出的字符总数，用于换行检测
int LineCount = 0;


string Sub(string s1, string s2) {
    int lena = s1.length();
    int lenb = s2.length();
    int a[505], b[505], c[505];
    memset(a, 0, sizeof(a));
    memset(b, 0, sizeof(b));
    memset(c, 0, sizeof(c));

    //将字符串写入到数组A中
    for (int i = 0; i < lena; i++) {
        //倒序写入
        a[i] = s1[lena - i - 1] - '0';
    }

    //将字符串写入到数组B中
    for (int i = 0; i < lenb; i++) {
        //倒序写入
        b[i] = s2[lenb - i - 1] - '0';
    }

    //模拟竖式减法
    for (int i = 0; i < lena; i++) {
        if (a[i] < b[i]) {
            //有借位
            a[i + 1]--;
            a[i] += 10;
        }
        c[i] = a[i] - b[i];
    }

    //删除前导零
    for (int i = lena - 1; i >= 0; i--) {
        //因为我们是从索引 0 开始，所以最高位是保存在 len-1
        if (0 == c[i] && lena > 1) {
            //注意要有 lena>1 这个条件。考虑特殊情况，
            //加法结果为 00，我们实际要输出 0。
            lena--;
        } else {
            //第一个不是零的最高位，结束删除
            break;
        }
    }

    string res = "";
```

```
    //逆序打印输出
    for (int i = lena - 1; i >= 0; i--) {
        res += c[i] + '0';
    }
    return res;
}

//高精度乘法
string Mult(string numa, string numb) {
    int a[1005], a1[1005], a2[1005];
    memset(a1, 0, sizeof(a1));
    memset(a2, 0, sizeof(a2));
    memset(a, 0, sizeof(a));
    //限定500位以内的计算，避免超时
    int lena = numa.size();
    int lenb = numb.size();
    for (int i = 0; i < lena; i++) {
        a1[i] = numa[lena - 1 - i] - '0';
    }

    for (int i = 0; i < lenb; i++) {
        a2[i] = numb[lenb - 1 - i] - '0';
    }
    for (int i = 0; i < lena; i++) {
        for (int j = 0; j < lenb; j++) {
            a[i + j] += a1[i] * a2[j];
        }
    }
    int len = lena + lenb;
    for (int i = 0; i < len; i++) {
        a[i + 1] += a[i] / 10;
        a[i] %= 10;
    }
    while (a[len] == 0 && len > 0) {
        len--;
    }
    string res = "";
    for (int i = len; i >= 0; i--) {
        res += a[i] + '0';
    }
    //截取最后500位，可以理解为取余
    if (res.length() > 500) {
        res = res.substr(res.length() - 500, 500);
    }
    return res;
}

//快速幂计算
```

```
string FastPower(string base, long long power) {
    string result = "1";
    while (power > 0) {
        if (power % 2 == 1) {
            result = Mult(result, base);
        }
        power = power / 2;
        base = Mult(base, base);
    }
    return result;
}


//换行检测
void LineCheck() {
    LineCount++;
    if (LineCount >= 50) {
        LineCount = 0;
        cout << endl;
    }
}

//打印倒数500位
void PrintRes(string res) {
    int size = res.size();
    //50个字符一换行

   //超过500位时正常打印
   if (size >= 500) {
        for (int i = size - 500; i < size; i++) {
            cout << res[i];
            LineCheck();
        }
    }
        //长度不足500位时，前面补0
    else {
        for (int i = 0; i < 500 - size; i++) {
            cout << 0;
            LineCheck();
        }
        for (int i = 0; i < size; i++) {
            cout << res[i];
            LineCheck();
        }
    }
}


int main() {
```

```
    int p;
    cin >> p;
    //调用快速幂
    string res = FastPower("2", p);
    //调用高精度减法
    res = Sub(res, "1");
    //打印长度
    printf("%d\n", int(p * log10(2)) + 1);
    //输出最终结果
    PrintRes(res);
    return 0;
}
```