Гончаренко Валерий Евстафиевич
Кандидат технических наук,
доцент Ивановского филиала
РЭУ им. Г.В. Плеханова, г. Иваново.

Использование двоичной системы счисления и длинной арифметики в задаче оптимизации теста Люка – Лемера

Существует ряд задач, решение которых программным путём невозможно с использованием только стандартных типов данных и функций. В статье рассматривается одна из таких задач – это тест Люка – Лемера, который доказывает принадлежность чисел Мерсенна к простым или составным числам. Современные исследования проводятся со сверхбольшими числами Мерсенна, в записи которых содержатся миллионы десятичных цифр. Хранить и обрабатывать такие числа в памяти ЭВМ можно лишь по алгоритмам длинной арифметики, что требует больших затрат процессорного времени. Представление этих чисел в двоичной системе счисления позволило существенно снизить вычислительную сложность обработки этих чисел и реализовать оптимизированный алгоритм теста в программе на языке C++.

Исследование чисел – это область увлекательной науки – теории чисел. Многие великие математики с большим воодушевлением проводили свои изыскания в этой области, начиная с легендарного Евклида.

Определение простых чисел в бесконечном ряду натуральных – одна из задач теории чисел. Напомним, что простым является натуральное число, которое имеет только два целочисленных делителя – единицу и собственное значение. Эти делители априори известны для каждого числа; есть ли у числа хотя бы ещё один целочисленный делитель – это уже задача определения принадлежности числа к простым или составным числам. Многие без труда могут представить начало ряда простых чисел: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31.

Простые числа уже давно используются в качестве ключей в криптографии для шифрования и дешифрования данных. Шифрование данных как метод информационной безопасности становится одной из самых актуальных задач буквально во всех сферах деятельности современного общества. Чем больше цифр в записи простого числа, тем сложнее взломать шифр. Взломать шифр, конечно, можно, и это делают специальные программы, но им нужно для этого время. В большинстве случаев затраченное время на взлом шифра делает полученную информацию неактуальной. Так что простое число 9973 в качестве ключа будет лучше, чем 31. А может, будет ещё лучше, если использовать самые большие простые числа?

Как в альпинизме спортсмены стремятся покорить самую высокую и трудную вершину, так и в поиске простых чисел математики стремятся определить очередное самое-самое большое простое число. Такие числа невозможно назвать и проблематично записать, ведь в их записи используются миллионы десятичных цифр. Практическое применение таких чисел не определено, в том числе в криптографии. Поскольку ряд натуральных чисел бесконечен, то самого большого простого числа не существует, но существует самое большое простое число, известное человечеству. Такие числа принято называть сверхбольшими. Проверку принадлежности сверхбольшого числа к простым выполнить чрезвычайно трудно даже с помощью могучих вычислительных машин. Простой перебор потенциальных делителей числа или алгоритм решета Эратосфена уже не помогут.

Чтобы не тратить усилия на каждое очередное число на числовой оси при проверке его на принадлежность к простым числам, обращаются к числам, которые имеют гораздо бóльшую вероятность оказаться простыми числами. Одной из форм таких чисел являются числа Мерсенна вида

$$M_p = 2^p - 1,$$

где p, в свою очередь, является простым числом. Например, можно получить число Мерсенна $M_{11} = 2047$ или $M_{31} = 2147483647$, но факт их принадлежности к простым или составным числам ещё надо доказать. Для сверхбольших чисел Мерсенна это становится очень трудоёмкой задачей, но этим числам ещё «повезло». Для доказательства принадлежности чисел Мерсенна к простым числам существует относительно простой детерминированный тест Люка – Лемера. Выдающийся французский математик Франсуа Эдуард Анатоль Люка (1842 – 1891) разработал этот тест и в 1872 г. «с карандашом в руках» доказал принадлежность числа Мерсенна M_{127} к простым числам. В дальнейшем тест был усовершенствован американским математиком Лемером, после чего и стал именоваться тестом Люка – Лемера. Сегодня для этих целей необходимо использовать мощные вычислительные средства, совершенное программное обеспечение и информационные технологии.

В рамках проекта Great Internet Mersenne Prime Search (GIMPS) реализуется поиск очередного самого большого простого числа Мерсенна. В настоящее время таким числом является $M_p = 2^{57885161} - 1$, в его записи содержится 17 425 170 десятичных цифр, и оно является 48-м простым числом Мерсенна[1][1].

[1] Определил 48-е простое число Мерсенна математик Купер из Университета Центрального Миссури в США, что принесло ему премию в размере $3000. Открытие Купера является частью проекта Great Internet Mersenne Prime Search (GIMPS), в реализации которого используются технологии распределённых вычислений в сети Internet.

Расчёты по тесту Люка – Лемера заключаются в последовательных вычислениях критерия, начальное значение которого $S_0 = 4$. Каждое следующее значение критерия вычисляется по предыдущему значению и числу Мерсенна по рекуррентной формуле

$$S_m = (S_{m-1} * S_{m-1} - 2) \; mod \; M_p. \quad (1)$$

Всего выполняется $p - 2$ итераций, и если значение последнего критерия равно нулю, т. е. $S_{p-2} = 0$, то число Мерсенна является простым числом.

Всю последовательность вычислений в тесте можно наглядно продемонстрировать на примере числа $M_7 = 2^7 - 1 = 127$.

```
S0 = 4
S1 = (4 * 4 - 2) mod 127 = 14
S2 = (14 * 14 - 2) mod 127 = 67
S3 = (67 * 67 - 2) mod 127 = 42
S4 = (42 * 42 - 2) mod 127 = 111
S5 = (111 * 111 - 2) mod 127 = 0.
```

Значение последнего критерия равно нулю, значит, число 127 является простым числом. Легко проверить, что для следующего простого числа $p = 11$ тест Люка – Лемера не подтверждает простоту числа Мерсенна M_{11}.

Для больших значений степеней двойки p необходимо воспользоваться длинной арифметикой. Самой трудоёмкой по затратам процессорного времени является деление по модулю M_p, или, другими словами, определение целочисленного остатка при делении на M_p. Стандартными операциями *div* и *mod* уже не воспользуешься.

Решение главных проблем

Сверхбольшие числа невозможно представить и обрабатывать в ЭВМ с помощью стандартных типов данных и функций. Для таких случаев можно воспользоваться алгоритмами длинной арифметики, в которой числа представляются в виде линейного массива элементов их записи. Элементы записи, в свою очередь, могут быть представлены стандартными типами данных с возможностью использования для их обработки обычных арифметических операций и функций.

Для сверхбольших чисел Мерсенна определение целочисленного остатка представляет наибольшую трудоёмкость теста. В данной статье предлагаются методы оптимизации расчётов с использованием алгоритмов длинной арифметики и двоичной системы счисления (с/с). Несмотря на то, что запись чисел в двоичной с/с гораздо длиннее, выполнение арифметических операций над ними может оказаться намного проще. Справедливо отметить, что задолго до появления компьютеров Франсуа Эдуард Анатоль Люка отмечал преимущества двоичной с/с, предвосхитив «принципы фон Неймана», опубликованные в 1945 г.

Рассмотрим примеры для чисел обычных величин. Десятичное число 6583_{10} в двоичной записи будет 1100100110111_2, а его представление в виде линейного массива двоичных цифр будет $m_0 = 1$, $m_1 = 1$, $m_2 = 1$, $m_3 = 0$, $m_4 = 1$, $m_5 = 1$, $m_6 = 0$, $m_7 = 0$, $m_8 = 1$, $m_9 = 0$, $m_{10} = 0$, $m_{11} = 1$, $m_{12} = 1$, или $m[1,1,1,0,1,1,0,0,1,0,0,1,1]$, т. е. разряд

единиц крайний слева, а цифры старших разрядов смещаются правее.

Если числа представить в двоичной системе счисления, то определение целочисленного остатка при делении на число M_p можно определить без сложных вычислений. В двоичной системе счисления нет необходимости в вычислениях для представления величин вида 2^p или $2^p - 1$. Для представления числа 2^p достаточно в старший разряд с номером p записать единицу, а в остальные разряды – нули. Для представления чисел $2^p - 1$ достаточно во все разряды от нулевого до разряда с номером $p - 1$ записать единицы, например, для $p = 7$ получим $2^7 = 10000000_2$, $2^7 - 1 = 1111111_2$. Необходимо отметить, что нумерация разрядов и нумерация элементов линейного массива начинается с нулевого значения.

Рассмотрим определение целочисленного остатка на примере произвольных двоичных чисел, например делимого 1100100110111_2 и делителя $2^7 = 10000000_2$, который на единицу больше соответствующего числа Мерсенна. Целочисленный остаток от деления можно определить как часть записи делимого от нулевого разряда до разряда с номером $p - 1$ включительно, для приведённых чисел это будет 0110111_2, или без ведущих нулей 110111_2.

В общем случае определение целочисленного остатка можно представить последовательным вычитанием делителя из делимого до тех пор, пока делимое станет меньше делителя. Эта величина и будет целочисленным остатком от деления. В рассмотренном примере делитель заменили на величину, бóльшую на единицу соответствующего числа Мерсенна M_7, и эта единица многократно вычиталась из делимого. Количество этих «лишних» вычитаний легко определить – это вторая часть в записи делимого от старшего разряда до разряда с номером p. Для приведённого примера это 1100110_2, следовательно, при делении на $2^7 - 1$ эти «лишние» вычеты необходимо прибавить к ранее определённому целочисленному остатку 110111_2, в итоге получим 1101010_2. Полученное число может содержать в своей записи значащие цифры в разряде с номером p и выше. В этом случае необходимо применить ранее описанную процедуру, причём только один раз для величин в тесте Люка – Лемера, так как делимое не превышает квадрата делителя.

Для приведённого примера делимое $1100100110111_2 = 6583_{10}$, делитель $M_7 = 1111111_2 = 127_{10}$, а полученный остаток $11010110_2 = 106_{10}$. Легко проверить правильность полученного результата, вычислив $6583_{10}\ mod\ 127_{10}$.

Основные моменты программной реализации

Для программной реализации оптимизированного теста Люка – Лемера выбран язык программирования C++ благодаря целому ряду его преимуществ. Как эксперт ЕГЭ по информатике и ИКТ могу отметить постоянно возрастающее количество заданий С4, написанных школьниками на C или C++. В представленной программной реализации не используются сложные конструкции и технологии языка. У языка C++ многие составные операторы имеют такой же или близкий синтаксис, как в языках *Pascal, Basic, Java,* например, *for, while, if*. Следует обратить внимание, что арифметические операции целочис-

ленного деления и взятие остатка от целочисленного деления в C++ обозначаются соответственно как «/» и «%». Эти операции будут часто встречаться при нормализации записи числа (в каждом разряде остаётся только величина допустимой цифры). Унарная операция увеличения или уменьшения операнда на единицу обозначается соответственно как два плюса и два минуса (инкремент и декремент). В языке C++ часто используется более короткая и быстрее выполняемая запись выражения, например, *a += b;* вместо *a = a + b;*. Надеюсь, что эти краткие пояснения помогут легче понять исходный код программы на C++ и при желании перекодировать его правильно на свой любимый язык программирования. Желательно попытаться найти другие возможности его оптимизации.

В программной реализации алгоритма теста Люка – Лемера необходимо использовать динамическую память (*heap*), предоставляющую бóльшие возможности для размещения в ней огромных массивов, а также её освобождение и повторное использование.

Рассмотрим последовательно выражения исходного кода программы на C++.

Объявление и ввод с клавиатуры степени двойки числа Мерсенна

```
cout<<"Введите степень числа Мерсенна (простое число)  ";
cin>>PM;
```

Объявление указателей на начало динамических массивов для вычисления и записи целочисленного остатка в соответствии с формулой (1), где

`a` – для остатка с предыдущей итерации;

`axa` – для остатка, формируемого на текущей итерации;

`tmp` – вспомогательный указатель для обмена адресами.

```
unsigned int * tmp;
unsigned int * axa = new unsigned int [PM];
unsigned int * a   = new unsigned int [PM];
```

Объявление индексов элементов массивов. Так как значения индексов часто и много используются, желательно их разместить в наиболее быстрой регистровой памяти, однако оптимизирующие компиляторы C++ могут и игнорировать это указание.

```
register unsigned int i, j, k;
```

Обнуление ячеек памяти, отведённых для динамических массивов:

```
for(i = 0; i < PM; ++i)
{
      axa[i] = 0;
      a[i]   = 0;
}
```

В следующем фрагменте исходного кода объявляется переменная `test` – номер итерации в тесте, `reg_a` – количество эле-

ментов в динамическом массиве (разрядность числа) и начальное значение целочисленного остатка (1110_2).

```
unsigned int test;
unsigned int reg_a;
//первоначальное значение остатка "a" = 4*4 -2 = 14 (0111)
reg_a = 4;
a[0] = 0; a[1] = 1; a[2] = 1; a[3] = 1;
```

Объявляется величина – переполнение текущего разряда в записи двоичного числа типа беззнакового целого числа.

```
unsigned int pp;
```

Теперь можно приступить к итерациям теста, в котором номер последней итерации не должен превышать значения ($PM - 1$).

```
for(test = 2; test < (PM - 1); ++test)
{
        //тело цикла по test
}
```

В начале каждой итерации по `test` необходимо вычислить квадрат целочисленного остатка с предыдущей итерации. Определение целочисленного остатка по описанному выше алгоритму в программной реализации уже начинается на этом этапе теста. При реализации структуры умножения столбиком элементы суммирования сразу направляются в соответствующий разряд по признаку суммы индексов для множимого и множителя `k = i + j`. Таким образом, сразу выполняется сложение двух частей в записи двоичного числа для определения целочисленного остатка. Операции с нулевыми значениями игнорируются оператором `continue`.

```
for(i = 0; i < PM; ++i)
      {
            if (a[i] == 0) continue;
            for(j =0; j < PM; ++j)
            {
                  if (a[j] == 0) continue;
                  k = i + j;
                  if (k >= PM) k -= PM;
                  ++axa[k];
             }
       }
```

После возведения в квадрат в соответствии с формулой (1) необходимо вычесть 2_{10}, что соответствует значению элемента массива `axa[1] = 1`, но он может оказаться нулевым, тогда необходимо выполнить вычитание с заёмом из ближайшего старшего разряда, номер которого изначально неизвестен.

```
if (axa[1]) --axa[1];
      else
            {
                  i = 1;
                  while (!axa[i])
                        {
                              axa[i] = 1;
                              ++i;
                        }
                  --axa[i];
            }
```

Необходимо массив axa преобразовать в нормальную запись двоичного числа без переполнений в его разрядах. Величина pp и будет на каждой итерации цикла этим переполнением, переносимым в старший разряд.

```
pp = 0;
for(i=0; i < PM; i++)
{
       axa[i] += pp;
       pp = axa[i] / 2;
       axa[i] %= 2;
 }
```

На последней итерации цикла while значение переполнения pp может и не быть нулевым, в этом случае оно должно переноситься в разряд с номером бóльшим чем (PM – 1). Для получения целочисленного остатка необходимо это переполнение прибавить к первому элементу массива с последующей нормализацией записи двоичного числа.

```
if (pp)
      {

        i = 0;
        while (pp)
              {
                    axa[i] += pp;
                    pp = axa[i] / 2;
                    axa[i] %= 2;
                    ++i;
              }
  }

while (axa[reg_a - 1] == 0) {reg_a--;}
```

Проверка программы «с карандашом в руках»

Представляет интерес выполнение проверки программы «с карандашом в руках» на примере обычных значений чисел. Рассмотрим расчёты для числа Мерсенна $M_7 = 1111111_2 = 127_{10}$ для случая, когда целочисленный остаток с предыдущей итерации принимает наи-

большее значение 1111110_2, т. е. всего на 1 меньше числа Мерсенна.

Результат умножения столбиком выглядит следующим образом:

						×	1	1	1	1	1	1	0
							1	1	1	1	1	1	0
							0	0	0	0	0	0	0
						1	1	1	1	1	1	0	
					1	1	1	1	1	1	0		
				1	1	1	1	1	1	0			
			1	1	1	1	1	1	0				
		1	1	1	1	1	1	0					
	1	1	1	1	1	1	0						
1	2	3	4	5	6	5	4	3	2	1	0	0	

В соответствии с кодом программы накопление величин сложения от разряда с номером p и выше будет суммироваться в разрядах от нулевого и выше, поэтому получим

+	5	4	3	2	1	0	0
		1	2	3	4	5	6
	5	5	5	5	5	5	6

Затем происходит вычитание единицы из разряда с номером 1

−	5	5	5	5	5	5	6
						1	0
	5	5	5	5	5	4	6

Нормализация записи двоичного числа

4	4	4	4	4	3	3	
	5	5	5	5	5	4	6
	1	1	1	1	0	1	0

Последнее переполнение pp = 4 должно перейти в разряд, который в цикле не предусмотрен, эта величина опять направляется в нулевой разряд с последующей нормализацией записи двоичного числа

+	1	1	1	1	0	1	0
							4
	1	1	1	1	0	1	4

Окончательно получим

					1	2	
1	1	1	1	0	1	4	
1	1	1	1	1	1	0	

На этом этапе завершается окончательное формирование целочисленного остатка, и для случая максимально возможной величины остатка с предыдущей итерации потребовалось лишь дважды использовать процедуру сложения различных частей в записи двоичного числа. Окончательно получили целочисленный остаток $11111110_2 = 126_2$. Выполним проверку полученного результата в десятичной с/с:

$$(126*126 - 2)\ mod\ 127 =$$
$$= 15874\ mod\ 127 = 126.$$

Целью выполненной проверки является подтверждение достаточности одной или двух процедур сложения двух частей в записи двоичного числа и соответствия исходного кода программы вышеописанному алгоритму.

Для приведённого примера результат предсказуемый. В соответствии с формулой (1) при получении целочисленного остатка, на 1 меньшего числа Мерсенна, на последующих итерациях теста он будет воспроизводиться с таким же значением. Очевидно, что для всех простых чисел Мерсенна такой ситуации не возникает.

Последние шаги программной реализации

Вернёмся к исходному коду программы. Использование на текущей итерации теста массива a на этом этапе завершается, и значения его элементов необходимо обнулить, так как в дальнейшем при обмене адресами это место в динамической памяти будет повторно использоваться для формирования нового массива axa.

```
for(i = 0; i < reg_a; ++i) a[i] = 0;
reg_a = PM;
```

После завершения формирования целочисленного остатка на текущей итерации под именем a, на следующей итерации он должен использоваться под именем axa, поэтому необходимо поменять адресами начала динамических массивов a и axa.

```
tmp = a;
a = axa;
axa = tmp;
```

Для того чтобы пользователь мог отслеживать прохождение теста от первой до последней итерации цикла по test, на монитор можно выводить очередной его текущий номер в очередной новой строке.

```
            cout<<"test "<<test<<endl;
} // конец цикла по test
```

После завершения цикла по test в представленной реализации программы возможны два состояния окончательно сформированного целочисленного остатка, подтверждающего простоту числа Мерсенна. В первом случае все элементы массива равны нулю и соответствующее значение разрядности (длина массива) равно нулю. Возможна ситуация, когда в массиве остаётся запись значения числа Мерсенна (второй случай). После завершения цикла по test программа должна вывести сообщение о принадлежности числа Мерсенна к простым числам либо к составным. Для этой цели используется дополнительная переменная логического типа

```
bool b = true;
for(i = 0; i < reg_a; ++i)
          if (!a[i]) { b = !b; break; }

if ((reg_a == 0) || b)
      cout<<"Число Мерсенна - простое\n";
            else cout<<"Число Мерсенна - составное\n";

cout<<endl;
```

Следующие выражения освобождают динамическую память:

```
delete [] a;
delete [] axa;
```

На этом завершаются основные выражения программы, отражающие суть алгоритма теста. Рассмотренный алгоритм был реализован в программе на C++ «Тест Люка – Лемера для сверхбольших чисел Мерсенна в двоичной системе счисления» [2]. Ниже исходный код программы представлен в полном объёме.

```
//Тест должен давать положительный ответ для начальных значений ряда
//степеней чисел Мерсенна PM: 7, 13, 17, 19, 31, 61, 89, 107,127, 521,
```

```
// 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213
и т. д.

#include<iostream>
using namespace std;

int main()
{
      system("chcp 1251>NULL");
      unsigned int  PM;
      cout<<"Введите степень числа Мерсенна (простое число)  ";
      cin>>PM;
      try
  {
      unsigned int * tmp;
      unsigned int * axa = new unsigned int [PM];
      unsigned int * a   = new unsigned int [PM];
      register unsigned int i, j, k;
      for(i = 0; i < PM; ++i)
      {
            axa[i] = 0;
           a[i]   = 0;
      }

      unsigned int test;
      unsigned int reg_a;
      //первоначальное значение остатка "a" = 4*4 -2 = 14 (0111)
      reg_a = 4;
      a[0] = 0; a[1] = 1; a[2] = 1; a[3] = 1;
      unsigned int pp;

      for(test = 2; test < (PM - 1); ++test)
      {
                  for(i = 0; i < PM; ++i)
                  {
                        if (a[i] == 0) continue;

                        for(j =0; j < PM; ++j)
                        {
                              if (a[j] == 0) continue;
                              k = i + j;
                              if (k >= PM) k -= PM;
                              ++axa[k];
                        }
                  }

                  for(i = 0; i < reg_a; ++i) a[i] = 0;

                  reg_a = PM;
```

```
                if (axa[1]) --axa[1];
                else
                        {
                                i = 1;
                                while (!axa[i])
                                        {
                                                axa[i] = 1;
                                                ++i;
                                        }
                                --axa[i];
                        }
                pp = 0;
                for(i=0; i < PM; i++)
                {
                        axa[i] += pp;
                        pp = axa[i] / 2;
                        axa[i] %= 2;
                }
                if (pp)
                {
                        i = 0;
                        while (pp)
                        {
                                axa[i] += pp;
                                pp = axa[i] / 2;
                                axa[i] %= 2;
                                ++i;
                        }
                }

        while (axa[reg_a - 1] == 0) {reg_a--;}

        tmp = a;
        a = axa;
        axa = tmp;
        cout<<"test "<<test<<endl;
} // конец цикла по test
bool b = true;
for(i = 0; i < reg_a; ++i)
                if (!a[i]) { b = !b; break; }

if ((reg_a == 0) ||  b)
        cout<<"Число Мерсенна - простое\n";
                else cout<<"Число Мерсенна - составное\n";

cout<<endl;

delete [] a;
delete [] axa;
```

```
        } //tru
        catch (std::bad_alloc& ba)
         {
        std::cerr << "bad_alloc caught: " << ba.what() << '\n';
         }

        system("pause");
        return 0;
}
```

Подведение итогов

Отметим основные результаты оптимизации теста Люка – Лемера для сверхбольших чисел Мерсенна:

1. Представление сверхбольших чисел в формате двоичной с/с позволило разительно сократить трудоёмкость определения целочисленного остатка от деления на число Мерсенна. Только при представлении делимого в двоичной с/с целочисленный остаток определяется простой арифметической операцией сложения двух частей в его записи.

2. При представлении числа в виде линейного массива его длина для всех промежуточных вычислений не превышает значения степени в записи числа Мерсенна. На каждой итерации цикла теста выполняется возведение в квадрат значения критерия с предыдущей итерации, что практически вдвое увеличивает разрядность числа (длину массива), но это исключается, так как значения из предполагаемых старших разрядов сразу прибавляются в младшие разряды по схеме сложения двух частей в записи числа.

В конце каждой итерации теста адреса динамических массивов, в которых хранятся значения критериев с предыдущей итерации и формируемое значение на текущей итерации, меняются адресами, что исключает необходимость выделения новых участков динамической памяти и освобождения ранее использованных.

Следует отметить, что для рекордных значений p порядка 50 млн и выше трудоёмкость теста Люка – Лемера остаётся непосильной задачей для обычного компьютера и необходимость в использовании распределённых вычислений в сети *Internet* остаётся.

Литература

1. URL: www.mersenne.org

2. Свидетельство о государственной регистрации программы для ЭВМ № 2014660821. Тест Люка – Лемера для сверхбольших чисел Мерсенна в двоичной системе счисления/ В. Е. Гончаренко, А.А. Тихонов; дата поступления 04.07.14, зарегистрировано 16.10.14.

Юмор Юмор Юмор Юмор Юмор Юмор

Оказывается...

Математика – это то, чем занимались Гаусс, Чебышев, Ляпунов, Стеклов и занимаюсь я.

А.А. Марков