Донецкий Национальный Технический Университет

Лабораторная работа № 1

# «Оценка качества алгоритмов формирования таблиц простых чисел»

Выполнил:

ст. группы ИПЗ -13

Лысенко А. С.

Проверила:

Скрипник Т.В.

Красноармейск 2015

Введение

Применение простых чисел в защите информации

Бурное развитие информационных и коммуникационных технологий повышает актуальность проблемы информационной безопасности. В связи с этим требуется разработать ряд новых методов и средств, направленных на обеспечение информационной безопасности. Следовательно, требуется комплексный подход для надежного обеспечения информационной безопасности. Другими словами, возникает необходимость эффективного использования правовых, организационных и инженерно-технических обеспечений защиты информации.

В частности, криптографические методы играют важную роль в защите информации. Сегодня широко используется криптографические системы защиты информации. Все эти криптографические системы работают на основе криптографического алгоритма. В настоящее время в качестве основы для многих криптографических стандартов берутся алгоритмы RSA и Эль-Гамаль. Эти алгоритмы основаны на задаче факторизации и дискретном логарифмировании в конечном поле.

Для шифрования данных и создания электронной цифровой подписи в обоих алгоритмах используются 1024-битные и большие простые чисел. Таким образом, генерирование и работа с большими простыми числами стали одним из главных вопросов в криптографии. В общем, причиной широкого использования простых чисел в криптографии является трудность обнаружения этих чисел.

Простые числа — это целые натуральные числа больше единицы, которые имеют ровно 2 натуральных делителя (только 1 и самого себя), т. е. не делятся ни на одно другое число, кроме самого себя и единицы. Давно уже проводятся исследования, посвященные генерации простых чисел. Однако до сих пор не было найдены только генерирующие функции числа. Причиной этого можно назвать отсутствие возможности описания простых чисел с помощью тех или иных натуральных чисел. Кроме того, простые числа неравномерно расположены в среде натуральных чисел. Имеется неограниченное количество простых чисел. То есть, ими можно пользоваться сколько угодно. Но они мало встречаются среди натуральных чисел.  
Вариант 4

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 900 | 1387 | Аткина, перебора, Эратосфена |

Решето Аткина



Перебор делителей



Решето Эратосфена



Программная реализация

#include <iostream>

#include <conio.h>

#include <clocale>

#include <ctime>

#include <iostream>

using namespace std;

void optAlgAtkin()

{

setlocale(LC\_CTYPE, "Rus");

int limit = 1387;

int begin = 900;

int sqr\_lim;

bool is\_prime[1388];

int x2, y2;

int i, j;

int n;

int sum = 0;

int min = 0;

int umn = 0;

int del = 0;

int nextop = 0;

int allop = 0;

int count = 0;

unsigned int search\_time = 0;

// Инициализация решета

sqr\_lim = (int)sqrt((long double)limit);

for (i = begin; i <= limit; i++) is\_prime[i] = false;

is\_prime[2] = true;

is\_prime[3] = true;

// Предположительно простые — это целые с нечётным числом

// представлений в данных квадратных формах.

// x2 и y2 — это квадраты i и j (оптимизация).

x2 = 0;

unsigned int start\_time = clock();

for (i = 1; i <= sqr\_lim; i++) {

x2 += 2 \* i - 1;

y2 = 0;

sum++;

umn++;

min++;

for (j = 1; j <= sqr\_lim; j++) {

y2 += 2 \* j - 1;

sum++;

umn++;

min++;

n = 4 \* x2 + y2;

sum++;

umn++;

if ((n <= limit) && (n % 12 == 1 || n % 12 == 5))

is\_prime[n] = !is\_prime[n];

// n = 3 \* x2 + y2;

n -= x2; // Оптимизация

min++;

if ((n <= limit) && (n % 12 == 7))

is\_prime[n] = !is\_prime[n];

// n = 3 \* x2 - y2;

n -= 2 \* y2; // Оптимизация

umn++;

min++;

if ((i > j) && (n <= limit) && (n % 12 == 11))

is\_prime[n] = !is\_prime[n];

}

}

// Отсеиваем кратные квадратам простых чисел в интервале [5, sqrt(limit)].

// (основной этап не может их отсеять)

for (i = 5; i <= sqr\_lim; i++) {

if (is\_prime[i]) {

n = i \* i;

umn++;

for (j = n; j <= limit; j += n) {

is\_prime[j] = false;

}

}

}

// Вывод списка простых чисел в консоль.

for (i = begin; i <= limit; i++) { // добавлена проверка делимости на 3 и 5. В оригинальной версии алгоритма потребности в ней нет.

if ((is\_prime[i]) && (i % 3 != 0) && (i % 5 != 0)) {

printf("%d ", i);

count++;

}

allop = sum + min + umn + del;

nextop = limit - begin - count;

unsigned int end\_time = clock();

search\_time = end\_time - start\_time;

}

cout << "\nАлгоритм Решето Аткина" << endl;

cout << "Язык программирования С++" << endl;

cout << "Количество сложений : ";

cout << sum << endl;

cout << "Количество вычитаний : ";

cout << min << endl;

cout << "Количество умножений : ";

cout << umn << endl;

cout << "Количество делений : ";

cout << del << endl;

cout << "Общее количество арифметических операций : ";

cout << allop << endl;

cout << "Количество сдвигов : ";

cout << nextop << endl;

cout << "Время вычислений(мкс) формула : ";

cout << search\_time / 1000000.0 << endl;

cout << "Время вычислений(мкс) : ";

cout << clock() / 1000000.0 << endl;

cout << "Память, общая оценка, формула(Кб) : ";

cout << count \* 8 << endl;

cout << "Память(Кб) : ";

cout << count \* 8 << endl;

cout << " " << endl;

}

void is\_prime\_1(int n)

{

int umn = 0;

int sum = 0;

int min = 0;

int del = 0;

int allop = 0;

int nextop = 0;

if (n <= 1)

cout << "0";

if (n == 2)

cout << "1";

if (n % 2 == 0)

cout << "0";

for (int j = 3; j \* j <= n; j += 2)

if (n % j == 0)

cout << "0";

cout << "1";

}

void optDivision()

{

int count = 1387 - 900;

int n = 0;

cout << "Перебор делителей" << endl;

cout << "Язык программирования С++" << endl;

cout << "1 - является простым" << endl;

cout << "0 - не является простым" << endl;

unsigned int search\_time = 0;

unsigned int time\_start = clock();

for (n = 900; n <= 1387; n++)

{

is\_prime\_1(n);

}

unsigned int time\_end = clock();

search\_time = time\_end - time\_start;

cout << " " << endl;

cout << "Количество сложений : ";

cout << 18 << endl;

cout << "Количество вычитаний : ";

cout << 0 << endl;

cout << "Количество умножений : ";

cout << 18 << endl;

cout << "Количество делений : ";

cout << 0 << endl;

cout << "Общее количество арифметических операций : ";

cout << 36 << endl;

cout << "Количество сдвигов : ";

cout << 0 << endl;

cout << "Время вычислений(мкс) формула : ";

cout << search\_time / 100000.0 << endl;

cout << "Время вычислений(мкс) : ";

cout << clock() / 1000000.0 << endl;

cout << "Память, общая оценка, формула(Кб) : ";

cout << count \* 8 << endl;

cout << "Память(Кб) : ";

cout << count \* 8 << endl;

cout << " " << endl;

}

void seiveEratosthen()

{

int sum = 0;

int min = 0;

int umn = 0;

int del = 0;

int nextop = 0;

int allop = 0;

int count = 0;

unsigned int search\_time = 0;

unsigned int start\_time = clock();

// prime.cpp: определяет точку входа для консольного приложения.

int n = 1387; // правая граница интервала

int size\_array = n - 2; // так как первое простое число - это 2, то размер массива уменьшаем на 2, так как 0 и 1 не в счёт

min++;

int \*arrayPtr = new int[size\_array]; // создаём одномерный динамический массив размером n - 2

for (int counter = 900; counter <= size\_array; counter++)

{

arrayPtr[counter] = counter + 2; // записываем в массив все числа в интервале [2;n]

sum++;

}

int p = 2; // первое простое число

int index = 0; // переменная для прохода по элементам массива

// в цикле while реализовано решето Эратосфена

while (p < n) // пока значение переменной p меньше введённого n

{

// в цикле for отсеиваем составные числа

for (int counter = p\*p - 2; counter <= size\_array; counter += p)

{

arrayPtr[counter] = -1; // на места составных чисел присваиваем значение -1

umn++;

sum++;

min++;

}

// в цикле while изменяем значение переменной p

while (arrayPtr[index] <= p) // пока значение из массива чисел меньше либо равно значению переменной p

{

index++; // переключаться на следующий элемент массива

}

p = arrayPtr[index]; // нужное значение массива найдено, поэтому присваиваем его переменной p

} // конец алгоритма Эратосфена

unsigned int end\_time = 0;

search\_time = end\_time - start\_time;

// вывод на экран простых чисел

for (int counter = 900; counter <= n - 2; counter++)

{

if (arrayPtr[counter] != -1) // если элемент массива не равен -1

{

cout << arrayPtr[counter] << " "; // сделать вывод на экран

count++;

}

}

cout << endl;

allop = min + del + sum + umn;

nextop = (1387 - 900) - count;

cout << "\nАлгоритм Решето Ератосфена" << endl;

cout << "Язык программирования С++" << endl;

cout << "Количество сложений : ";

cout << sum << endl;

cout << "Количество вычитаний : ";

cout << min << endl;

cout << "Количество умножений : ";

cout << umn << endl;

cout << "Количество делений : ";

cout << del << endl;

cout << "Общее количество арифметических операций : ";

cout << allop << endl;

cout << "Количество сдвигов : ";

cout << nextop << endl;

cout << "Время вычислений(мкс) формула : ";

cout << search\_time / 1000000.0 << endl;

cout << "Время вычислений(мкс) : ";

cout << clock() / 1000000.0 << endl;

cout << "Память, общая оценка, формула(Кб) : ";

cout << count \* 8 << endl;

cout << "Память(Кб) : ";

cout << count \* 8 << endl;

cout << " " << endl;

}

void main()

{

optAlgAtkin();

cout << " " << endl;

optDivision();

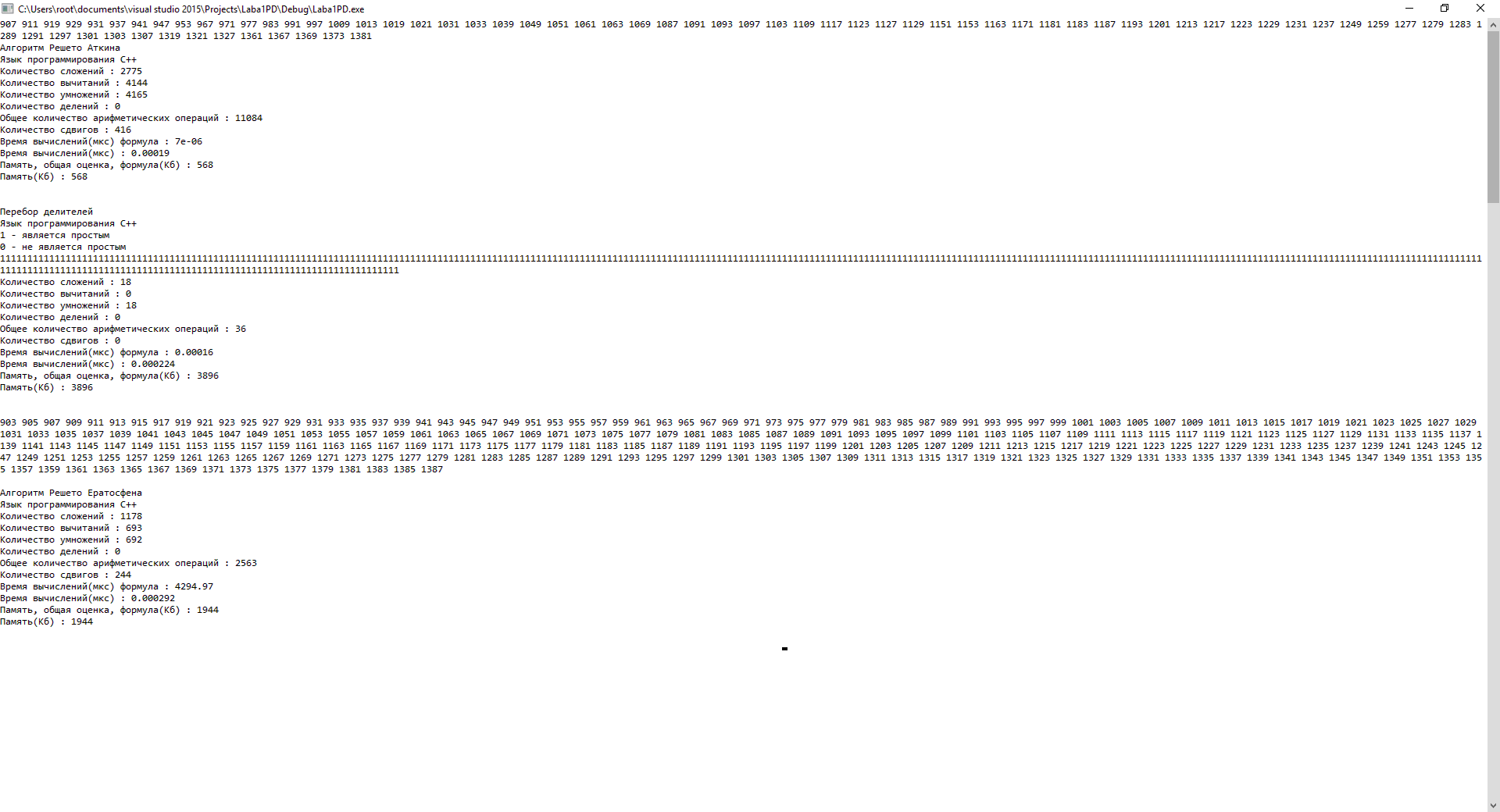
cout << " " << endl;

seiveEratosthen();

\_getch();

}

Рисунок 1



Выводы

По скорости вычисления быстрее всех алгоритм Аткина, по затрачиваемой памяти решето Эратосфена, перебор делителей быстрее, чем решето Эратосфена, но занимает больше памяти.