Оглавление

[Введение 4](#_Toc90929114)

[1.1 Тест Соловея-Штрассена. Теория. 4](#_Toc90929115)

[1.2 Алгоритм Соловея – Штрассена. 5](#_Toc90929116)

[2.1 Реализация Теста Соловея-Штрассена. 6](#_Toc90929117)

[2.2 Результаты тестирования 9](#_Toc90929118)

[Заключение 10](#_Toc90929119)

[Литература 11](#_Toc90929120)

[Приложение. Полный исходный код. 12](#_Toc90929121)

# Введение

Целью данной курсовой работы является программная реализация теста проверки простоты Соловея – Штрассена, оценка реализованного алгоритма.

Для реализации поставленной задачи было решено использовать язык программирования С++. Код написан в редакторе исходного кода “Visual Studio Code”. Версия gcc 8.3.0.

Тестом простоты (или проверкой простоты) называется алгоритм, который, приняв на входе число {\displaystyle N}N, позволяет либо не подтвердить предположение о составности числа, либо точно утверждать его простоту. Таким образом, тест простоты представляет собой только гипотезу о том, что если алгоритм не подтвердил предположение о составности числа {\displaystyle N}N, то это число может являться простым с определённой вероятностью.

Задача теста простоты относится к классу сложности P, то есть время работы алгоритмов её решения зависит от размера входных данных полиноминально.

В данной курсовой работе будут рассмотрен и реализован тест Соловея-Штрассена.

# 1.1 Тест Соловея-Штрассена. Теория.

Тест Соловея — Штрассена — вероятностный тест простоты, открытый в 1970-х годах Робертом Мартином Соловеем совместно с Фолькером Штрассеном. Тест всегда корректно определяет, что простое число является простым, но для составных чисел с некоторой вероятностью он может дать неверный ответ. Основное преимущество теста заключается в том, что он, в отличие от теста Ферма, распознает числа Кармайкла как составные.

Тест Соловея — Штрассена опирается на малую теорему Ферма, которая гласит: Если число p – простое число и α – целое число, не делящееся на p, то делится на p. Формальная запись (1):

(1)

И свойства символа Якоби: Если *n* — нечетное составное число, то количество целых чисел *a*, взаимнопростых с n удовлетворяющих сравнению (2):

{\displaystyle \textstyle a^{(n-1)/2}\equiv \left({\frac {a}{n}}\right){\pmod {n}}}, не превосходит {\displaystyle {\frac {n}{2}}} . (2)

Алгоритм Соловея —Штрассена  параметризуется количеством раундов *k*. В каждом раунде случайным образом выбирается число *a* < *n*. Если НОД(*a*,*n*) > 1, то выносится решение, что *n* составное. Иначе проверяется справедливость сравнения  {\displaystyle \textstyle a^{(n-1)/2}\equiv \left({a \over n}\right){\pmod {n}}}. Если оно не выполняется, то выносится решение, что *n* — составное. Если это сравнение выполняется, то *a* является свидетелем простоты числа *n*.

Далее выбирается другое случайное *a* и процедура повторяется. После нахождения *k* свидетелей простоты в *k* раундах выносится заключение, что *n* является простым числом с вероятностью .

# 1**.2 Алгоритм Соловея – Штрассена.**

1. Тест полученного число (n > 2), оно должно быть нечетным и натуральным.
2. Назначение количество итераций – параметр определяющий точность теста.
3. Начало итерации.
4. Нахождение случайного числа a, такого, что 2 < a < n.
5. Вычисление НОД(a, n), если он больше 1 – число составное, если нет – выполняется следующий пункт алгоритма.
6. Вычисление значения символа Якоби .
7. Проверка выполнения сравнения: . Если сравнение выполняется, число a становится свидетелем простоты числа n, в противном случае число n – составное.
8. Начало следующей итерации.
9. По окончании итераций, число n признается простым с вероятностью .

# **2.1 Реализация Теста Соловея-Штрассена**.

Программа принимает на вход 2 значения:

1. Количество итераций (itarations);
2. Число, которое необходимо проверить на простоту (number);

int iterations;

long long number;

std::cout << "Choose number of iterations: : ";

std::cin >> iterations;

std::cout << "Choose your number: ";

std::cin >> number;

Выходным значением будет одно из следующих сообщений:

1. Число простое с вероятностью ;
2. Число составное;

Функция «**SoloveyStrassen**» реализует проверку на простоту числа p (number).

bool SoloveyStrassen(long long p, int iterations)

{

if (p < 2)

return false;

if (p != 2 && p % 2 == 0)

return false;

for (int i = 0; i < iterations; i++)

{

// Создаем рандомное значение для a (a < n)

long long a = rand() % (p - 1) + 1;

// Вычисляем НОД

if (GKD(a, p) != 1)

return false;

// Считаем Значение символа Якоби

long long jacobian = (p + calculateJacobian(a, p)) % p;

// Считаем значение a^((p - 1) / 2) (mod p)

long long mod = Modul(a, (p - 1) / 2, p);

// Смотрим на эквивалентность

if (!jacobian || mod != jacobian)

return false;

}

return true;

}

Для проверки создается случайное число a (< p). Далее вычисляется значение НОД. Функция «**GCD**» по алгоритму Евклида рассчитывает значение НОД.

long long GKD(long long a, long long b) {

if (a % b == 0)

return b;

if (b % a == 0)

return a;

if (a > b)

return GKD(a%b, b);

return GKD(a, b%a);

}

Далее, если значение НОД равно единице, вычисляется значение символа Якоби. В противном случае, функция «**SoloveyStrassen**» возвращает значение – false и число p признается составным.

Символ Якоби рассчитывается в функции «**CalculateJacobian**»:

int calculateJacobian(long long a, long long n)

{

if (!a)

return 0;// (0/n) = 0

int ans = 1;

// Переход к положительным числам

if (a < 0)

{

a = -a;

if (n % 4 == 3)

ans = -ans; // (-1/n) = -1 if n = 3 (mod 4)

}

if (a == 1)

return ans;// (1/n) = 1

while (a)

{

if (a < 0)

{

a = -a;// (a/n) = (-a/n)\*(-1/n)

if (n % 4 == 3)

ans = -ans;// (-1/n) = -1 if n = 3 (mod 4)

}

// Избавление от четности

while (a % 2 == 0)

{

a = a / 2;

if (n % 8 == 3 || n % 8 == 5)

ans = -ans;

}

std::swap(a, n);

// Квадратичный закон взаимности

if (a % 4 == 3 && n % 4 == 3)

ans = -ans;

a = a % n;

if (a > n / 2)

a = a - n;

}

if (n == 1)

return ans;

return 0;

}

Полученное значение Символа Якоби сравнивается по модулю p с :

Правая часть сравнения вычисляется в функции «**Modul**»:

long long Modul(long long base, long long exponent,

long long mod)

{

long long x = 1;

long long y = base;

while (exponent > 0)

{

if (exponent % 2 == 1)

x = (x \* y) % mod;

y = (y \* y) % mod;

exponent = exponent / 2;

}

return x % mod;

}

Если значения функций «**Modul**» и «**CalculateJacobian**» совпадают, цикл переходит к следующей итерации, иначе число признается составным. По окончании цикла, число признается простым с вероятностью .

Временная сложность данной реализации: , что совпадает с теоритической сложностью.

# 2.2 Результаты тестирования

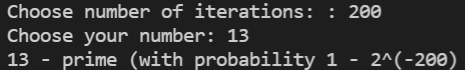


Рисунок 1 – Тест числа 13 (простое).

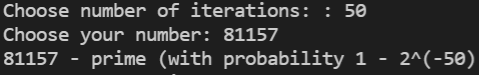


Рисунок 2 – Тест числа 81157 (простое).

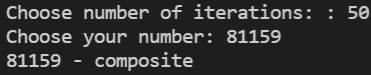


Рисунок 3 – Тест числа 81159 (составное).

# Заключение

В данной курсовой работе была продемонстрирована реализация теста проверки простоты Соловея-Штрассена. Данный алгоритм является вероятностным тестом простоты, однако уже при количестве итераций равной 10 и выше, вероятность, что число является простым будет 99,9%, при этом при числе итераций равном единице, вероятность равна 50%, при двойке 75% из чего можно сделать вывод, что вероятность и количество итераций растут непропорционально.

Временная сложность алгоритма, представленного в данной курсовой работе совпадает с теоритической.

# Литература

1. «Тест Соловея-Штрассена» [Электронный ресурс], URL: <https://ru.wikipedia.org/wiki/Тест_Соловея_—_Штрассена>
2. «Символ Якоби» [Электронный ресурс], URL: <https://ru.wikipedia.org/wiki/Символ_Якоби>
3. «Основная теорема арифметики. — Популярные лекции по математике» Калужин Л. А., 33 с. 1969.
4. Конспект лекций.

# Приложение. Полный исходный код.

#include <iostream>

// Функция подсчета НОД

long long GKD(long long a, long long b) {

if (a % b == 0)

return b;

if (b % a == 0)

return a;

if (a > b)

return GKD(a%b, b);

return GKD(a, b%a);

}

// Функция подсчета a^((p - 1) / 2) (mod p)

long long Modul(long long base, long long exponent,

long long mod)

{

long long x = 1;

long long y = base;

while (exponent > 0)

{

if (exponent % 2 == 1)

x = (x \* y) % mod;

y = (y \* y) % mod;

exponent = exponent / 2;

}

return x % mod;

}

// Считаем значение символя Якоби

int calculateJacobian(long long a, long long n)

{

if (!a)

return 0;// (0/n) = 0

int ans = 1;

// Переход к положительным числам

if (a < 0)

{

a = -a;

if (n % 4 == 3)

ans = -ans; // (-1/n) = -1 if n = 3 (mod 4)

}

if (a == 1)

return ans;// (1/n) = 1

while (a)

{

if (a < 0)

{

a = -a;// (a/n) = (-a/n)\*(-1/n)

if (n % 4 == 3)

ans = -ans;// (-1/n) = -1 if n = 3 (mod 4)

}

// Избавление от четности

while (a % 2 == 0)

{

a = a / 2;

if (n % 8 == 3 || n % 8 == 5)

ans = -ans;

}

std::swap(a, n);

// Квадратичный закон взаимности

if (a % 4 == 3 && n % 4 == 3)

ans = -ans;

a = a % n;

if (a > n / 2)

a = a - n;

}

if (n == 1)

return ans;

return 0;

}

// Функция проверки на простоту

bool SoloveyStrassen(long long p, int iterations)

{

if (p < 2)

return false;

if (p != 2 && p % 2 == 0)

return false;

for (int i = 0; i < iterations; i++)

{

// Создаем рандомное значение для a (a < n)

long long a = rand() % (p - 1) + 1;

// Вычисляем НОД

if (GKD(a, p) != 1)

return false;

// Считаем Значение символа Якоби

long long jacobian = (p + calculateJacobian(a, p)) % p;

// Считаем значение a^((p - 1) / 2) (mod p)

long long mod = Modul(a, (p - 1) / 2, p);

// Смотрим на эквивалентность

if (!jacobian || mod != jacobian)

return false;

}

return true;

}

int main()

{

int iterations;

long long number;

std::cout << "Choose number of iterations: : ";

std::cin >> iterations;

std::cout << "Choose your number: ";

std::cin >> number;

if (SoloveyStrassen(number, iterations))

std::cout << number << " - prime (with probability 1 - 2^(-" << iterations << ")" << std::endl;

else

std::cout << number << " - composite" << std::endl;

return 0;

}