Лабораторная работа №5

Подмогильный Иван Александрович - студент группы НПМмд-02-22

18.09.2022

Вероятностные алгоритмы проверки чисел на простоту

Прагматика выполнения

Умение использовать и программировать вероятностные алгоритмы проверки чисел на простоту

Цель выполнения лабораторной работы

Освоить на практике вычисление проверки числа на простоту с помощью вероятности

Задачи выполнения работы

1. Реализовать алгоритм теста Ферма
2. Реализовать вычисления символа Якоби
3. Реализовать алгоритм теста Соловэя-Штрассена
4. Реализовать алгоритм теста Миллера-Рабина

Результаты выполнения лабораторной работы.

Написал код для вычисления теста Ферма. Все функции реализованы шаблонами:

```cpp
#ifndef LAB05_PRIMENUMBERSHELPER_H
#define LAB05_PRIMENUMBERSHELPER_H

#include <random>
#include <stdexcept>

class PrimeNumbersHelper {
public:
    template <typename T>
    // true - probably prime number
    // false - composite number
    static bool FermatTest(const T& n){
        if (!check_number(n, 5)){
            throw std::invalid_argument( "number must be >= 5 and odd number" );
        }
        T a = randN(2, n - 2);
        T r = (T)pow(a, n - 1) % n;
        if (r == 1){
            return true;
        }
        else{
            return false;
        }
    }
```

Figure 1: Функция теста Ферма

Написал код для вычисления теста Соловэя-Штрассена.

```cpp
template<typename T>
// true - probably prime number
// false - composite number
static bool SolovayStrassenTest(const T& n){
    if (!check_number(n, 5)){
        throw std::invalid_argument( "number must be >= 5 and odd number" );
    }
    T a = randN(2, n - 2);
    T r = (T)pow(a, (n-1)/2) % n;
    if (r != 1 && r != (n-1)){
        return false;
    }
    T s = symbolJacobi(n, a);
    if (r % n == s){
        return false;
    }
    else{
        return true;
    }
}
```

Figure 2: Функция теста Соловэя-Штрассена

Реализовал вычисление теста Миллера-Рабина

```cpp
template<typename T>
// true - probably prime number
// false - composite number
static bool millerRabinTest(const T& n){
    if (!check_number(n, 5)){
        throw std::invalid_argument( "number must be >= 5 and odd number" );
    }
    T s = dividersCount(n-1, 2);
    T r = (n-1) / (T)pow(2, s);
    T a = randN(2, n-2);
    T y = (T)pow(a, r) % n;
    if (y != 1 && y != n-1){
        T j = 1;
        if (j < s - 1 && y != n - 1){
            y = y*y % n;
            if (y == 1){
                return false;
            }
            j += 1;
        }
        if (y != n-1){
            return false;
        }
    }
    return true;
}
```

Figure 3: тест Миллера-Рабина

Реализовал вспомогательную функцию вычисления случайного числа

```
template <typename T>
static decltype(auto) randN(const T& low, const T& high){
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distr(low, high);
    return distr( &: gen);
}
```

Figure 4: Вспомогательная функция вычисления случайного числа

Написал вспомогательную функцию вычисления символа Якоби

```cpp
template<typename T>
static decltype(auto) symbolJacobi(const T& n, const T& a){
    if (n < 3){
        throw std::invalid_argument( "number must be >= 3" );
    }
    T g = 1;
    T res, s;
    T nCopy = n;
    T aCopy = a;

    while(true){
        if (aCopy == 0){
            res = 0;
            return res;
        }
        else if (aCopy == 1){
            res = g;
            return res;
        }
        T k = dividersCount(aCopy, 2);
        T a1 = aCopy / (T)pow(2, k);
        if (k % 2 == 0){
            s = 1;
        }
        else{
            if (nCopy % 8 == 1 | nCopy % 8 == -1){
                s = 1;
            }
            else if (nCopy % 8 == 3 | nCopy % 8 == -3){
                s = -1;
            }
        }
        if (a1 ==1){
            res = g * s;
            return res;
        }
        if (nCopy % 4 == 3 && a1 % 4 == 3){
            s = -s;
        }
        aCopy = nCopy % a1;
        nCopy = a1;
        g = g * s;
    }

}
```

Написал вспомогательную функцию подсчета делителей divider в числе n

```cpp
template<typename T>
static decltype(auto) dividersCount(const T& n, const T& divider){
    int i = 0;
    T nCopy = n;
    while (nCopy % divider == 0){
        i += 1;
        nCopy /= divider;
    }
    return i;
}
```

Figure 6: Вспомогательная функция подсчета делителей

Написал вспомогательную функцию проверки числа (она используется в начале каждой функции)

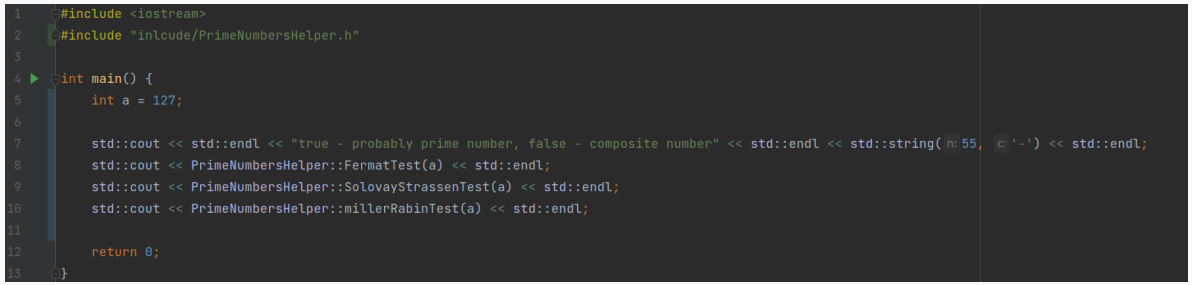
```
    template<typename T>
    static bool check_number(const T& n, const T& threshold){
        if (n < threshold){
            return false;
        }
        if (n % 2 == 0){
            return false;
        }
        return true;
    };
};
```

Figure 7: Вспомогательная функция проверки числа

Написал CMake файл

```
1  cmake_minimum_required(VERSION 3.20)
2  project(lab05)
3
4  set(CMAKE_CXX_STANDARD 14)
5
6  add_library(lab05 src/PrimeNumbersHelper.cpp inlcude/PrimeNumbersHelper.h)
7
8  add_executable(main main.cpp)
9  target_link_libraries(main lab05)
```

Написал main.cpp файл, в котором есть тесты реализованных функций.



```cpp
#include <iostream>
#include "inlcude/PrimeNumbersHelper.h"

int main() {
    int a = 127;

    std::cout << std::endl << "true - probably prime number, false - composite number" << std::endl << std::string( n: 55,  c: '-') << std::endl;
    std::cout << PrimeNumbersHelper::FermatTest(a) << std::endl;
    std::cout << PrimeNumbersHelper::SolovayStrassenTest(a) << std::endl;
    std::cout << PrimeNumbersHelper::millerRabinTest(a) << std::endl;

    return 0;
}
```

Figure 9: main.cpp файл

Результаты тестов.

```
/home/pi/education/pfur_masters/matOsnovyInfBez/labs/lab05/cmake-build-debug/main

true - probably prime number, false - composite number
-------------------------------------------------------
0
0
0

Process finished with exit code 0
```

Figure 10: Результаты тестов

Выводы

Освоил на практике вычисление наибольшего делителя разными способами