

Лабораторная работа №4

Подмогильный Иван Александрович - студент группы НПМмд-02-22

18.09.2022

Вычисление наибольшего общего делителя

Умение вычислять наибольшего делителя разными способами

Цель выполнения лабораторной работы

Освоить на практике вычисление наибольшего делителя разными способами

1. Реализовать вычисление НОД алгоритмом Евклида
2. Реализовать вычисление НОД бинарным алгоритмом Евклида
3. Реализовать вычисление НОД расширенным алгоритмом Евклида
4. Реализовать вычисление НОД расширенным бинарным алгоритмом Евклида

Результаты выполнения лабораторной работы. Написал код для вычисление НОД алгоритмом Евклида

```
#ifndef LAB04_GCDHELDER_H
#define LAB04_GCDHELDER_H

#include <stdexcept>

class GCDHelder {
public:
    // bear in mind that user might use int32_t, int64_t, uint, char, etc...
    template<typename T>
    static void eucClassic(const T& arg1, const T& arg2, T& gcd){
        check_args(arg1, arg2);

        T a, b;
        find_init_values(arg1, arg2, a, b);

        auto r0 = a, r1 = b;
        gcd = b;
        while (true){
            gcd = r0 % r1;
            if (gcd == 0){
                break;
            }
            r0 = r1;
            r1 = gcd;
        }
    }
}
```

Figure 1: вычисление НОД алгоритмом Евклида

Реализовал вычисление НОД бинарным алгоритмом Евклида

```
template<typename T>
static void eucBinary(const T& arg1, const T& arg2, T& gcd){
    check_args(arg1, arg2);
    T a, b;
    find_init_values(arg1, arg2, a, b);
    T g = 1;
    while ( a % 2 == 0 && b % 2 == 0 ){
        a /= 2;
        b /= 2;
        g *= 2;
    }
    T u = a, v = b;
    while (u != 0){
        while (u % 2 == 0){
            u /= 2;
        }
        while (v % 2 == 0){
            v /= 2;
        }

        if (u >= v){
            u -= v;
        }
        else{
            v -= u;
        }
    }
    gcd = g * v;
}
```

Реализовать вычисление НОД расширенным алгоритмом Евклида

```
template<typename T>
static std::pair<T, T> eucExpanded(const T& arg1, const T& arg2, T& gcd){
    check_args(arg1, arg2);
    T a, b;
    find_init_values(arg1, arg2, a, b);
    auto r0 = a, r1 = b;
    T x0 = 1, x1 = 0, y0 = 0, y1 = 1;
    gcd = b;

    while (true){
        T q;
        gcd = r0 % r1;
        if (gcd == 0){
            std::pair<T, T> res{x1, y1};
            return res;
        }
        r0 = r1;
        r1 = gcd;
        q = r0 / r1;
        T tmpX = x1, tmpY = y1;
        x1 = x0 - q * x1;
        y1 = y0 - q * y1;
        x0 = tmpX;
        y0 = tmpY;
    }
}
```

Figure 3: НОД расширенным алгоритмом Евклида

Реализовать вычисление НОД расширенным бинарным алгоритмом Евклида

```
template<typename T>
static std::pair<T, T> eucBinaryExpanded(const T& arg1, const T& arg2, T& gcd){
    check_args(arg1, arg2);
    T a, b;
    find_init_values(arg1, arg2, a, b);
    T g = 1;
    while ( a % 2 == 0 && b % 2 == 0 ){
        a /= 2;
        b /= 2;
        g *= 2;
    }

    T u = a, v = b;
    T A = 1, B = 0, C = 0, D = 1;
    while (u != 0){
        while (u % 2 == 0){
            u /= 2;
            if (A % 2 == 0 && B % 2 == 0){
                A /= 2;
                B /= 2;
            }
            else{
                A = (A + b) / 2;
                B = (B - a) / 2;
            }
        }
        while (v % 2 == 0){
            v /= 2;
            if (C % 2 == 0 && D % 2 == 0){
                C /= 2;
                D /= 2;
            }
            else{
                C = (C + b) / 2;
                D = (D - a) / 2;
            }
        }
    }
```

```
        if (u >= v){
            u -= v;
            A -= C;
            B -= D;
        }
        else{
            v -= u;
            C -= A;
            D -= B;
        }
    }

    gcd = g * v;
    T x = C;
    T y = D;
    std::pair<T, T> xy{x, y};
    return xy;
}
```

Figure 5: НОД расширенным бинарным алгоритмом Евклида 2

Написал вспомогательные функции которые определены приватными.

```
private:
    template<typename T>
    static void check_args(const T& arg1, const T& arg2){
        if (arg1 < 0 | arg2 < 0){
            auto error = "a is: " + std::to_string(arg1) + ", and b is: " + std::to_string(arg2) +
                + ", but only values >= 0 are accepted";
            throw std::invalid_argument( error );
        }
    }
}

template<typename T>
static void find_init_values(const T& arg1, const T& arg2, T& a, T& b){
    if (arg1 >= arg2){
        a = arg1;
        b = arg2;
    }
    else{
        a = arg2;
        b = arg1;
    }
}

};

#endif //LAB04_GCDHELDER_H
```

Figure 6: вспомогательные функции

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

```
1  #include <iostream>
2  #include "include/GCDHelder.h"
3
4  int main() {
5      // TODO: Check the validity of the encrypted messages
6      int a, b, gcd;
7      a = 105;
8      b = 154;
9
10     GCDHelder::eucClassic(a, b, &gcd);
11     std::cout << gcd << std::endl;
12     GCDHelder::eucBinary(a, b, &gcd);
13     std::cout << gcd << std::endl;
14     GCDHelder::eucExpanded(a, b, &gcd);
15     std::cout << gcd << std::endl;
16     GCDHelder::eucBinaryExpanded(a, b, &gcd);
17     std::cout << gcd << std::endl;
18
19
20     return 0;
21 }
```

Figure 7: main.cpp файл

```
/home/pi/education/pfur_masters/mat0snovyInfBez/labs/lab04/cmake-build-debug/main
0
7
0
7

Process finished with exit code 0
```

Figure 8: Результаты тестов

Освоил на практике вычисление наибольшего делителя разными способами