РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

Тема: Вычисление наибольшего общего делителя

дисциплина: Математические основы защиты информации и информационной безопасности

Студент: Койфман Кирилл Дмитриевич

Группа: НФИмд-01-25

Введение

Цель работы

Получение практических навыков реализации алгоритмов, вычисляющих наибольший общий делитель (НОД).

Задачи

1. Реализовать алгоритм Евклида, бинарный алгоритм Евклида, расширенный алгоритм Евклида, расширенный бинарный алгоритм Евклида.

Теория

Пусть числа a и b целые и $b\neq0$. Разделить a на b с остатком - значит предоставить a в виде $a=qb+\gamma$, где $q, \gamma\in Z$ и $0\leq\gamma\leq|b|$. Число q называется неполным частным, число γ - неполным остатком от деления a на b.

Целое число $d\neq0$ называется *наибольшим общим делителем* целых чисел $a_1, a_2, ..., a_k$ (обозначается $d=НОД(a_1, a_2, ..., a_k)$), если выполняются следующие условия:

1. каждое из чисел $a_1, a_2, ..., a_k$ делится на d;
2. если $d\neq0$ - другой общий делитель чисел $a_1, a_2, ..., a_k$, то d делится на d_1. Например, $НОД(12345, 24690) = 12345$, $НОД(12345, 54321) = 3$, $НОД(12345, 12541) = 1$.

Ненулевые целые числа a и b называются *ассоциированными* (обозначается a~b), если a делится на b и b делится на a.

Для любых целых чисел $a_1, a_2, ..., a_k$ существует наибольший общий делитель d и его можно представить в виде *линейной комбинации* этих чисел: $$ d=c_1a_1+c_2a_2+...+c_ka_k,c_i\in Z (Z - множество целых чисел). $$ Например, НОД чисел 91, 105, 154 равен 7. В качестве линейного представления можно взять $$ 7=7*91+(-6)*105+0*154, $$ либо $$ 7=4*91+1*105-3*154. $$

Целые числа $a_1, a_2, ..., a_k$ называются *взаимно простыми в совокупности*, если $НОД(a_1, a_2, ..., a_k)=1$. Целые числа a и b называются *взаимно простыми*, если $НОД(a, b)=1$.

Целые числа $a_1, a_2, ..., a_k$, называются *попарно взаимно простыми*, если $НОД(a_i, a_j)=1$ для всех $1\leq i\neq j\leq k$.

Ход работы

Для решения поставленной задачи реализуем описанные в тексте лабораторной работы алгоритмы для вычисления наибольшего общего делителя (НОД) на языке программирования C++ (Листинг-1 - Листинг-4), а также проведём тест данных алгоритмов, чтобы проверить корректность их работы (Листинг-5):

```
int algorithmEuclid(int a, int b)
{
    if (b > a)
        std::swap(a, b);

    //d=НОД(a, b)
    int d = 0;

    int gamma_prev = a;
    int gamma_curr = b;
    int gamma_next = 0;
    while (true)
    {
        int remainder = gamma_prev % gamma_curr;
        gamma_next = remainder;

        if (gamma_next == 0)
        {
```

```
            d = gamma_curr;
            break;
        }
        else
        {
            gamma_prev = gamma_curr;
            gamma_curr = gamma_next;
        }
    }

    return d;
}
```

Листинг-1(реализация алгоритма Евклида)

```
int binaryAlgorithmEuclid(int a, int b)
{
    if (b > a)
        std::swap(a, b);

    //d=НОД(a, b)
    int d = 0;

    int g = 1;
    //until one in pair (a or b) becomes odd
    while (a % 2 == 0 && b % 2 == 0)
    {
        a = a / 2;
        b = b / 2;
        g = 2 * g;
    }

    int u = a;
    int v = b;
    while (u != 0)
    {
        if (u % 2 == 0)
            u = u / 2;

        if (v % 2 == 0)
            v = v / 2;

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

```
    return d;
}
```

Листинг-2(реализация бинарного алгоритма Евклида)

```
struct EuclidAlgoVars
{
    int d, x, y;
};

EuclidAlgoVars extendedAlgorithmEuclid(int a, int b)
{
    bool swapXY = false;
    if (b > a)
    {
        swapXY = true;
        std::swap(a, b);
    }

    //a * x + b * y = d
    int d = 0;
    int x = 0;
    int y = 0;

    //gamma_0{i-1}
    int gamma_prev = a;
    //gamma_1{i}
    int gamma_curr = b;
    //gamma_{i+1}
    int gamma_next = 0;

    //x_0{i-1}
    int x_prev = 1;
    //x_1{i}
    int x_curr = 0;
    //x_{i+1}
    int x_next = 0;

    //y_0{i-1}
    int y_prev = 0;
    //y_1{i}
    int y_curr = 1;
    //y_{i+1}
    int y_next = 0;

    while (true)
    {
        int remainder = gamma_prev / gamma_curr;
        //q_i = remainder
        int q_curr = remainder;
```

```
        //gamma_{i+1} = gamma_{i-1} - q_i * gamma_i
        gamma_next = gamma_prev - q_curr * gamma_curr;

        if (gamma_next == 0)
        {
            d = gamma_curr;
            x = x_curr;
            y = y_curr;
            break;
        }
        else
        {
            x_next = x_prev - q_curr * x_curr;
            y_next = y_prev - q_curr * y_curr;

            gamma_prev = gamma_curr;
            gamma_curr = gamma_next;

            x_prev = x_curr;
            x_curr = x_next;

            y_prev = y_curr;
            y_curr = y_next;
        }
    }

    if (swapXY)
        std::swap(x, y);

    return EuclidAlgoVars{ d,x,y };
}
```

Листинг-3(реализация расширенного алгоритма Евклида)

```
EuclidAlgoVars binaryExtendedAlgorithmEuclid(int a, int b)
{
    bool swapXY = false;
    if (b > a)
    {
        swapXY = true;
        std::swap(a, b);
    }
    //a * x + b * y = d
    int d = 0;
    int x = 0;
    int y = 0;

    int g = 1;
    //until one in pair (a or b) becomes odd
    while (a % 2 == 0 && b % 2 == 0)
```

```
    {
        a = a / 2;
        b = b / 2;
        g = 2 * g;
    }

    int u = a;
    int v = b;
    int A = 1;
    int B = 0;
    int C = 0;
    int D = 1;

    while (u != 0)
    {
        if (u % 2 == 0)
        {
            u = u / 2;
            if (A % 2 == 0 && B % 2 == 0)
            {
                A = A / 2;
                B = B / 2;
            }
            else
            {
                A = (A + b) / 2;
                B = (B - a) / 2;
            }
        }

        if (v % 2 == 0)
        {
            v = v / 2;
            if (C % 2 == 0 && D % 2 == 0)
            {
                C = C / 2;
                D = D / 2;
            }
            else
            {
                C = (C + b) / 2;
                D = (D - a) / 2;
            }
        }

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

```
            C = C - A;
            D = D - B;
        }

    }
    d = g * v;
    x = C;
    y = D;

    if (swapXY)
        std::swap(x, y);

    return EuclidAlgoVars{ d,x,y };
}
```

Листинг-4(реализация расширенного бинарного алгоритма Евклида)

```
-------------------------Testing GCD-algorithms-------------------------
TEST-1: a = 14, b = 21
Great Common Divisor(GCD) for pair{a=14, b=21} with [1]Euclid Algorithm: 7
Great Common Divisor(GCD) for pair{a=14, b=21} with [2]Binary Euclid
Algorithm: 7
Great Common Divisor(GCD) for pair{a=14, b=21} with [3]Extended Euclid
Algorithm: 7 with condition that a * x + b * y = d, d = 7, x = -1, y = 1:
14 * -1 + 21 * 1 = 7(true)
Great Common Divisor(GCD) for pair{a=14, b=21} with [4]Binary Extended
Euclid Algorithm: 7 with condition that a * x + b * y = d, d = 7, x = -10,
y = 7:
14 * -10 + 21 * 7 = 7(true)

TEST-2: a = 48, b = 36
Great Common Divisor(GCD) for pair{a=48, b=36} with [1]Euclid Algorithm:
12
Great Common Divisor(GCD) for pair{a=48, b=36} with [2]Binary Euclid
Algorithm: 12
Great Common Divisor(GCD) for pair{a=48, b=36} with [3]Extended Euclid
Algorithm: 12 with condition that a * x + b * y = d, d = 12, x = 1, y =
-1:
48 * 1 + 36 * -1 = 12(true)
Great Common Divisor(GCD) for pair{a=48, b=36} with [4]Binary Extended
Euclid Algorithm: 12 with condition that a * x + b * y = d, d = 12, x =
-5, y = 7:
48 * -5 + 36 * 7 = 12(true)

TEST-3: a = 17, b = 51
Great Common Divisor(GCD) for pair{a=17, b=51} with [1]Euclid Algorithm:
17
Great Common Divisor(GCD) for pair{a=17, b=51} with [2]Binary Euclid
Algorithm: 17
Great Common Divisor(GCD) for pair{a=17, b=51} with [3]Extended Euclid
```

```
Algorithm: 17 with condition that a * x + b * y = d, d = 17, x = 1, y = 0:
17 * 1 + 51 * 0 = 17(true)
Great Common Divisor(GCD) for pair{a=17, b=51} with [4]Binary Extended
Euclid Algorithm: 17 with condition that a * x + b * y = d, d = 17, x = 1,
y = 0:
17 * 1 + 51 * 0 = 17(true)

TEST-4: a = 75, b = 250
Great Common Divisor(GCD) for pair{a=75, b=250} with [1]Euclid Algorithm:
25
Great Common Divisor(GCD) for pair{a=75, b=250} with [2]Binary Euclid
Algorithm: 25
Great Common Divisor(GCD) for pair{a=75, b=250} with [3]Extended Euclid
Algorithm: 25 with condition that a * x + b * y = d, d = 25, x = -3, y =
1:
75 * -3 + 250 * 1 = 25(true)
Great Common Divisor(GCD) for pair{a=75, b=250} with [4]Binary Extended
Euclid Algorithm: 25 with condition that a * x + b * y = d, d = 25, x =
-93, y = 28:
75 * -93 + 250 * 28 = 25(true)

TEST-5: a = 72, b = 120
Great Common Divisor(GCD) for pair{a=72, b=120} with [1]Euclid Algorithm:
24
Great Common Divisor(GCD) for pair{a=72, b=120} with [2]Binary Euclid
Algorithm: 24
Great Common Divisor(GCD) for pair{a=72, b=120} with [3]Extended Euclid
Algorithm: 24 with condition that a * x + b * y = d, d = 24, x = 2, y =
-1:
72 * 2 + 120 * -1 = 24(true)
Great Common Divisor(GCD) for pair{a=72, b=120} with [4]Binary Extended
Euclid Algorithm: 24 with condition that a * x + b * y = d, d = 24, x =
-3, y = 2:
72 * -3 + 120 * 2 = 24(true)

TEST-6: a = 81, b = 65
Great Common Divisor(GCD) for pair{a=81, b=65} with [1]Euclid Algorithm: 1
Great Common Divisor(GCD) for pair{a=81, b=65} with [2]Binary Euclid
Algorithm: 1
Great Common Divisor(GCD) for pair{a=81, b=65} with [3]Extended Euclid
Algorithm: 1 with condition that a * x + b * y = d, d = 1, x = -4, y = 5:
81 * -4 + 65 * 5 = 1(true)
Great Common Divisor(GCD) for pair{a=81, b=65} with [4]Binary Extended
Euclid Algorithm: 1 with condition that a * x + b * y = d, d = 1, x =
-134, y = 167:
81 * -134 + 65 * 167 = 1(true)
```

Листинг-5(результаты работы алгоритмов, вычисляющих НОД)

Исходя из полученных результатов (Листинг-5), можно судить о том, что реализованные алгоритмы успешно вычисляют НОД для пар целочисленных значений.

Заключение

В ходе проделанной лабораторной работы мной были получены навыки по реализации алгоритмов, вычисляющих НОД.