**МИНОБРНАУКИ РОССИИ**
**ФГБОУ ВО «СГУ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

# АЛГОРИТМЫ АЛГЕБРЫ И ТЕОРИИ ЧИСЕЛ

ЛАБОРАТОРНАЯ РАБОТА №14

студента 4 курса 431 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Никитина Арсения Владимировича

Проверил
доцент ____________ А. С. Гераськин

Саратов 2022

# СОДЕРЖАНИЕ

1 Задание лабораторной работы . . . . . . . . . . 3
2 Теоретическая часть . . . . . . . . . . 4
2.1 Расширенный алгоритм Евклида для полиномов над полем (Extended Euclidean Algorithm for Polynomials over a Field) . . . . . . . . . . 4
2.2 Обобщенный алгоритм Евклида для полиномов над це- целыми числами (Generalized Euclidean Algorithm for Polynomials over the Integers) . . . . . . . . . . 4
3 Практическая часть . . . . . . . . . . 6
3.1 Пример работы алгоритма . . . . . . . . . . 6
3.2 Код программы, реализующей рассмотренный алгоритм . . . . . . . . . . 6

# 1 Задание лабораторной работы

Реализовать алгоритмы вычисления НОД в кольцах полиномов:

1. Расширенный алгоритм Евклида для полиномов над полем.
2. Обобщённый алгоритм Евклида для полиномов над целыми числами.

## 2 Теоретическая часть

### 2.1 Расширенный алгоритм Евклида для полиномов над полем (Extended Euclidean Algorithm for Polynomials over a Field)

*Вход:* $p_1(x),\ p_2(x) \in J[x],\ p_2(x) \neq 0,\ m = deg[p_1(x)] \geqslant deg[p_2(x)] = n$, где $J$ — поле.

*Выход:* $p_(x),\ f(x),\ g(x) \in J[x]$, такие, что $deg[f(x)] < deg[p_1(x)] - deg[p_h(x)],\ deg[g(x)] < deg[p_2(x)] - deg[p_h(x)]$ и $p_h(x) = gcd[p_1(x), p_2(x)] = p_1(x)g(x) + p_2(x)f(x)$.

1. Инициализировать переменные:
   *а)* $[p_0(x), p_1(x)] := [p_1(x), p_2(x)]$;
   *б)* $[g_0(x), g_1(x)] = [1, 0]$;
   *в)* $[f_0(x), f_1(x)] = [0, 1]$;
2. Пока $p_1(x) \neq 0$:
   *а)* $q(x) := \mathtt{PDF}[p_0(x), p_1(x)]$;
   *б)* $[p_0(x), p_1(x)] := [p_1(x), p_0(x) - p_1(x)q(x)]$;
   *в)* $[g_0(x), g_1(x)] := [g_1(x), g_0(x) - g_1(x)q(x)]$;
   *г)* $[f_0(x), f_1(x)] := [f_1(x), f_0(x) - f_1(x)q(x)]$;
3. Вернуть $[p_h(x), g(x), f(x)] := [p_0(x), g_0(x), f_0(x)]$.

### 2.2 Обобщенный алгоритм Евклида для полиномов над це- целыми числами (Generalized Euclidean Algorithm for Polynomials over the Integers)

*Контент полинома (cont)* — наибольший общий делитель всех коэффициентов полинома.

*Примитивная часть полинома (primitive part (pp))* — исходный полином, разделенный на его контент.

Полином также называют примитивным, если его контент равен единице, так как примитивная часть полинома в данном случае будет являться исходным полиномом.

*Вход:* $p_1(x), p_2(x)$ — ненулевые полиномы в $\mathbb{Z}[x]$; $deg[p_1(x)] = n_1, deg[p_2(x)] = n_2,\ n_1 \geqslant n_2$.

*Выход:* $gcd[p_1(x), p_2(x)]$, НОД полиномов $p_1(x)$ и $p_2(x)$.

1. Вычислить содержания gcd. Для этого перемнной $c$ присвоить значение $gcd(cont[p_1(x)], cont[p_2(x)])$.
2. Вычислить примитивные части полиномов:

   а) $p_1^{'}(x) = p_1(x)/cont[p_1(x)]$;
   б) $p_2^{'}(x) = p_2(x)/cont[p_2(x)]$;
3. Построить PRS. Для этого выполнить последовательное деление полиномов в кольце целых чисел. Пока остаток в PDF не равен нулю:
   а) $p_1(x), p_2(x) = p2(x), \texttt{pdf}(p_1(x), p2(x))$.
4. Если $deg[p_h(x)] = 0$ (предпоследний остаток от деления многочленов), то вернуть в качестве ответа $c$, иначе вернуть в качестве ответа $c \cdot pp[p_h(x)]$.

# 3 Практическая часть

## 3.1 Пример работы алгоритма

```
Выполнить нахождение НОД полиномов с помощью расширенного алгоритма Евклида над полем или выполнить обобщенн
ый алгоритм Евклида для полиномов над целыми числами - \enter
Выход из программы - 2
Введите значение:
Выполнить нахождение НОД полиномов с помощью расширенного алгоритма Евклида над полем - 1, Обобщенный алгори
тм Евклидадля полиномов над целыми числами - 2
2
Введите коэффициенты полинома, начиная с коэффициента при наибольшей степени:
1 -1 -2 2 1 -1
Введите от 1 до 6 коэффициентов
Введите коэффициенты полинома, начиная с коэффициента при наибольшей степени:
5 -4 -6 4 1
GCD( (x^5 + -1*x^4 + -2*x^3 + 2*x^2 + x^1 + -1), (5*x^4 + -4*x^3 + -6*x^2 + 4*x^1 + 1) ) = (x^3 + -1*x^2 + -
1*x^1 + 1)

Выполнить нахождение НОД полиномов с помощью расширенного алгоритма Евклида над полем или выполнить обобщенн
ый алгоритм Евклида для полиномов над целыми числами - \enter
Выход из программы - 2
Введите значение:
Выполнить нахождение НОД полиномов с помощью расширенного алгоритма Евклида над полем - 1, Обобщенный алгори
тм Евклидадля полиномов над целыми числами - 2
1
Введите поле, в котором требуется поделить многочлены: 11
Введите коэффициенты полинома, начиная с коэффициента при наибольшей степени:
3 4 5 6 3 2 1 42 5
Полином 1 имеет вид:
3*x^8 + 4*x^7 + 5*x^6 + 6*x^5 + 3*x^4 + 2*x^3 + x^2 + 9*x^1 + 5
Введите от 1 до 9 коэффициентов
Введите коэффициенты полинома, начиная с коэффициента при наибольшей степени:
3 4 5 4 3 2
Полином 2 имеет вид
3*x^5 + 4*x^4 + 5*x^3 + 4*x^2 + 3*x^1 + 2
(8*x^1 + 4) = (3*x^8 + 4*x^7 + 5*x^6 + 6*x^5 + 3*x^4 + 2*x^3 + x^2 + 9*x^1 + 5) * (6*x^3 + x^2 + 7*x^1 + 4)
+ (3*x^5 + 4*x^4 + 5*x^3 + 4*x^2 + 3*x^1 + 2) * (5*x^6 + 10*x^5 + 4*x^4 + 3*x^3 + x^2 + 8*x^1 + 3)

Выполнить нахождение НОД полиномов с помощью расширенного алгоритма Евклида над полем или выполнить обобщенн
ый алгоритм Евклида для полиномов над целыми числами - \enter
```

Рисунок 1

## 3.2 Код программы, реализующей рассмотренный алгоритм

```
import sympy


def polinomial_view(coefs, flag=False):
    n = len(coefs) - 1
    a = ''
    mul = '*'
    if coefs:

        last = coefs[-1]
        coefs = coefs[:-1:]
        if coefs:
            for i, coef in enumerate(coefs):
```

```
                if coef:
                    print(f'{str(coef) + mul if coef != 1 else a}x^{n - i} +', end=' ')
        print(last, end='')
    else:
        print(coefs, end='')
    if not flag:
        print()
    return


def get_field():
    n = int(input('Введите поле, в котором требуется поделить многочлены: '))
    if not sympy.isprime(n):
        print('Вы ввели не простое число')
        return get_field()
    else:
        return n


def get_coefs(j=None):
    print('Введите коэффициенты полинома, начиная с коэффициента при' +
    ' наибольшей степени:')
    koef_modula = lambda x : int(x) % j
    koef_integer = lambda x : int(x)
    coefs = map(koef_modula if j is not None else koef_integer, input().split())
    return list(coefs)


def gcdExtended(a, b):
    if a == 0 :
        return b, 0, 1

    gcd, x1, y1 = gcdExtended(b % a, a)
    x = y1 - (b // a) * x1
    y = x1
    return gcd, x, y


def divide_in_modula(a, b, j):
```

```python
    _, x, _ = gcdExtended(b, j)
    b_inversed = ((x % j + j) % j)
    return (a * b_inversed) % j


def pdf(m_coefs_aboba, n_coefs, p, flag=False):

    m_coefs = m_coefs_aboba.copy()
    n = len(n_coefs) - 1

    right = len(m_coefs) - len(n_coefs)
    q_coefs = [0] * (right + 1)

    for k in range(right, -1, -1):

        if not flag:
            q_coefs[k] = divide_in_modula(m_coefs[n + k], n_coefs[n], p)
        else:
            q_coefs[k] = m_coefs[n + k] // n_coefs[n]

        for j in range(n + k - 1, k - 1, -1):
            m_coefs[j] = (m_coefs[j] - q_coefs[k] * n_coefs[j - k])
            if not flag:
                m_coefs[j] %= p

    return poly_reduction(q_coefs), poly_reduction((m_coefs[0:n]))


def poly_subtraction(p1, p2, modula):

    if len(p2) > len(p1):

        p2 = list(map(lambda x: -x, p2))
        for i, coef in enumerate(p1):
            p2[i] += coef
        return poly_reduction([i % modula for i in p2])

    else:
        for i, coef in enumerate(p2):
            p1[i] -= coef
        return poly_reduction([i % modula for i in p1])
```

```
def poly_reduction(f):
    i = 0
    f = f[::-1]
    while i < len(f) and f[i] == 0:
        i += 1
    return (f[i:])[::-1]


def poly_multiplication(p1, p2, modula):

    len_p1, len_p2 = len(p1), len(p2)

    result = [0] * (len_p1 + len_p2 - 1)

    for i in range(len_p1):
        for j in range(len_p2):
            result[i + j] += p1[i] * p2[j]
    return poly_reduction([i % modula for i in result])


def poly_gcd_extedned(p1, p2, j):

    p_0, p_1 = p1, p2
    g_0, g_1 = [1], [0]
    f_0, f_1 = [0], [1]

    while p_1:
        q, _ = pdf(p_0, p_1, j)

        p_0, p_1 = p_1, poly_subtraction(p_0, poly_multiplication(p_1, q, j),
         ↪  j)
        g_0, g_1 = g_1, poly_subtraction(g_0, poly_multiplication(g_1, \
            q, j), j)
        f_0, f_1 = f_1, poly_subtraction(f_0,\
            poly_multiplication(f_1, q, j), j)

    return p_0[::-1], g_0[::-1], f_0[::-1]
```

```python
def get_content(p):
    for i in range(abs(max(p, key=abs)), 0, -1):
        flag = True
        for j in p:
            if j % i:
                flag = False
                break
        if flag:
            return i if max(p, key=abs) > 0 else -i


def geap(p_1, p_2):
    p1, p2 = p_1.copy(), p_2.copy()

    p1_content, p2_content = get_content(p1), get_content(p2)

    c = sympy.gcd(p1_content, p2_content)

    remainder = None

    while p2:

        p1_content, p2_content = get_content(p1), get_content(p2)
        p1 = [coef // p1_content for coef in p1]
        p2 = [coef // p2_content for coef in p2]

        lc = p2[-1] ** (len(p1) - len(p2) + 1)
        p1 = [i * lc for i in p1]

        p1, p2 = p2, pdf(p1, p2, None, True)[1]
        if p2:
            remainder = p2

    if len(remainder) == 1:
        return c
    else:
        last_content = get_content(remainder)
        return [c * i // last_content for i in remainder]


def left_bracket():
```

```python
    print('(', end='')
    return


def main():

    while True:

        print('\nВыполнить нахождение НОД полиномов с помощью расширенного' +
        ' алгоритма Евклида над полем или выполнить обобщенный алгоритм ' +
        'Евклида для полиномов над целыми числами - \enter')

        print('Выход из программы - 2')

        try:
            value = int(input('Введите значение: '))

        except ValueError:
            value = 1

        if value == 1:

            print('Выполнить нахождение НОД полиномов с помощью расширенного ' +
            'алгоритма Евклида над полем - 1, Обобщенный алгоритм Евклида' +
            'для полиномов над целыми числами - 2')

            division_option = int(input())

            j = None

            if division_option == 1:

                j = get_field()
                p1 = get_coefs(j)
                print('Полином 1 имеет вид:')
                polinomial_view(p1)

                print(f'Введите от 1 до {len(p1)} коэффициентов')
                p2 = get_coefs(j)
                print('Полином 2 имеет вид')
```

```
        polinomial_view(p2)

        left, a, b = poly_gcd_extedned(p1[::-1], p2[::-1], j)


        left_bracket()
        polinomial_view(left, flag=True)
        print(') = ', end='')

        left_bracket()
        polinomial_view(p1, flag=True)
        print(') * ', end='')

        left_bracket()
        polinomial_view(a, flag=True)
        print(') + ', end='')

        left_bracket()
        polinomial_view(p2, flag=True)
        print(') * ', end='')

        left_bracket()
        polinomial_view(b, flag=True)
        print(')')

    else:

        m_coefs = get_coefs()
        print(f'Введите от 1 до {len(m_coefs)} коэффициентов')
        n_coefs = get_coefs()

        print('GCD( (', end='')
        polinomial_view(m_coefs, True)
        print('), ', end='')

        left_bracket()
        polinomial_view(n_coefs, True)
        print(') ) = ', end='')

        g = geap(m_coefs[::-1], n_coefs[::-1])

```

```
                if isinstance(g, int):
                    print(g)
                else:
                    left_bracket()
                    polinomial_view(g, True)
                    print(')')

        elif value == 2:
            print('Работа программы завершена')
            return


if __name__ == "__main__":
    main()

# Пример:
# 1, -1, -2, 2, 1, -1
# 5, -4, -6, 4, 1

# Должно получиться:
# p1 = 1 -1 -2 2 1 -1
# p2 = 5 -4 -6 4 1
# p3 = -24 24 24 -24
# p4 = 0 0 0
```