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

АЛГОРИТМЫ АЛГЕБРЫ И ТЕОРИИ ЧИСЕЛ

ЛАБОРАТОРНАЯ РАБОТА №17

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

Разложение полиномов на свободные от квадратов множители:

— Алгоритм разложения на свободные от квадратов множители

2 Теоретическая часть

Разложение полиномов на свободные от квадратов мно- множители (Polynomial Squarefree Factorization)

Вход: $p_(x)$ — примитивный полином положительной степени от одной переменной над областью J характеристики нуль с однозначным разложением на множители.

Выход: Полиномы $s_i(x)$ и число e, такие, что $p(x) = \prod_{i=1}^{e}[s_i(x)]^i$ — разложение полинома $p(x)$ на свободные от квадратов множители.

1. Инициализировать $r(x) := \mathtt{gcd}[p(x), p^{'}(x)],\ t := p(x)/r(x),\ j := 1$
2. Если $deg[r(x)] = 0$, то ответ — $e = j,\ s_j(x) = t(x)$.
3. Вычислить s_j: $v(x) := \mathtt{gcd}[r(x), t(x)],\ s_j := t(x)/v(x)$.
4. $r(x)$ присвоить значение $r(x)/v(x)$, $t(x)$ присвоить значение $v(x)$, $j := j + 1$. Перейти к шагу 2.

3 Практическая часть

3.1 Пример работы алгоритма

```
Выполнить разложение полинома на свободные от квадратовмножители  - \enter
Выход из программы - 2
Введите значение:
Введите коэффициенты полинома, начиная с коэффициента при наибольшей степени:
1 -1 1 1 -1 1
Полином имеет вид:
x^5 + -1*x^4 + x^3 + x^2 + -1*x^1 + 1
x^5 + -1*x^4 + x^3 + x^2 + -1*x^1 + 1 = (x^1 + 1)^1 * (x^2 + -1*x^1 + 1)^2 * 1

Выполнить разложение полинома на свободные от квадратовмножители  - \enter
Выход из программы - 2
Введите значение:
Введите коэффициенты полинома, начиная с коэффициента при наибольшей степени:
1 -1 -2 2 1 -1
Полином имеет вид:
x^5 + -1*x^4 + -2*x^3 + 2*x^2 + x^1 + -1
x^5 + -1*x^4 + -2*x^3 + 2*x^2 + x^1 + -1 = (x^1 + 1)^2 * (x^1 + -1)^3 * 1

Выполнить разложение полинома на свободные от квадратовмножители  - \enter
Выход из программы - 2
Введите значение: 2
Работа программы завершена
```

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='')
```

```
17      else:
18          print(coefs, end='')
19      if not flag:
20          print()
21      return
22
23
24  def get_field():
25      n = int(input('Введите поле, в котором требуется поделить многочлены: '))
26      if not sympy.isprime(n):
27          print('Вы ввели не простое число')
28          return get_field()
29      else:
30          return n
31
32
33  def get_coefs(j=None):
34      print('Введите коэффициенты полинома, начиная с коэффициента при' +
35      ' наибольшей степени:')
36      koef_modula = lambda x : int(x) % j
37      koef_integer = lambda x : int(x)
38      coefs = map(koef_modula if j is not None else koef_integer,
        ↪  input().split())
39      return list(coefs)
40
41
42  def pdf(m_coefs_aboba, n_coefs):
43
44      m_coefs = m_coefs_aboba.copy()
45      n = len(n_coefs) - 1
46
47      right = len(m_coefs) - len(n_coefs)
48      q_coefs = [0] * (right + 1)
49
50      for k in range(right, -1, -1):
51          q_coefs[k] = m_coefs[n + k] // n_coefs[n]
52
53          for j in range(n + k - 1, k - 1, -1):
54              m_coefs[j] = (m_coefs[j] - q_coefs[k] * n_coefs[j - k])
55
56      return poly_reduction(q_coefs), poly_reduction((m_coefs[0:n]))
```

```
57
58
59  def poly_subtraction(p1, p2, modula):
60
61      if len(p2) > len(p1):
62
63          p2 = list(map(lambda x: -x, p2))
64          for i, coef in enumerate(p1):
65              p2[i] += coef
66          return poly_reduction([i % modula for i in p2])
67
68      else:
69          for i, coef in enumerate(p2):
70              p1[i] -= coef
71          return poly_reduction([i % modula for i in p1])
72
73
74  def poly_reduction(f):
75      i = 0
76      f = f[::-1]
77      while i < len(f) and f[i] == 0:
78          i += 1
79      return (f[i:])[::-1]
80
81
82  def poly_multiplication(p1, p2, modula):
83
84      len_p1, len_p2 = len(p1), len(p2)
85
86      result = [0] * (len_p1 + len_p2 - 1)
87
88      for i in range(len_p1):
89          for j in range(len_p2):
90              result[i + j] += p1[i] * p2[j]
91      return poly_reduction([i % modula for i in result])
92
93
94  def get_content(p):
95
96      for i in range(abs(max(p, key=abs)), 0, -1):
97          flag = True
```

```
        for j in p:
            if j % i:
                flag = False
                break
        if flag:
            if p[-1] < 0:
                return -i
            else:
                 return 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)[1]
        if p2:
            remainder = p2

    if remainder is None:
        return p_2

    if len(remainder) == 1:
        return c
    else:
```

```
        last_content = get_content(remainder)
        return [c * i // last_content for i in remainder]


def get_derivative(p):
    return [coef * (i + 1) for i, coef in enumerate(p[1::])]



def psqff(p):

    p_derivative = get_derivative(p)
    r = geap(p, p_derivative)
    t = pdf(p, r)[0]
    j = 1
    ls = []

    while len(r) != 1:

        v = geap(r, t)
        s = pdf(t, v)[0]
        ls.append(s)
        r = pdf(r, v)[0]
        t = v
        j += 1

    return j, ls + [t]


def left_bracket():
    print('(', end='')
    return


def main():

    while True:

        print('\nВыполнить разложение полинома на свободные от квадратов' +
        'множители  - \enter')

```

```
180         print('Выход из программы - 2')
181 
182         try:
183             value = int(input('Введите значение: '))
184 
185         except ValueError:
186             value = 1
187 
188         if value == 1:
189 
190             p = get_coefs()
191             print('Полином имеет вид: ')
192             polinomial_view(p)
193 
194             power, brackets = psqff(p[::-1])
195             if brackets[0] == [1]:
196                 brackets = brackets[1::]
197                 start_value = 2
198             else:
199                  start_value = 1
200 
201             polinomial_view(p, True)
202             print(' = ', end='')
203 
204             for bracket in brackets:
205                 left_bracket()
206                 polinomial_view(bracket[::-1], True)
207                 print(f')^{start_value} * ', end = '')
208                 start_value += 1
209             print('1')
210         else:
211             print('Работа программы завершена')
212             return
213 
214 
215 if __name__ == "__main__":
216     main()
217 
218 # 1 -1 1 1 -1 1
```