МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Одномерный алгоритм Кронекера
Многомерный алгоритм Кронекера

ЛАБОРАТОРНАЯ РАБОТА

студента 4 курса 431 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Серебрякова Алексея Владимировича

Научный руководитель
доцент, к. п. н. ____________ А. С. Гераськин
подпись, дата

Саратов 2022

Описание метода

Рассмотрим $f(x) \in \mathbb{Z}[x]$ – многочлен степени n. Пусть $f(x)$ приводим над $\mathbb{Z}$. Тогда $f(x) = g(x)h(x)$ и один из двух многочленов $g(x)$ и $h(x)$ имеет степень не выше $n/2$. Пусть без ограничения общности $\deg g(x) \leqslant n/2$. Тогда $\forall a \in \mathbb{Z}\ h(a) \in \mathbb{Z}$, следовательно $f(a) \vdots g(a)$. Рассмотрим $m = n/2 + 1$ различных целых чисел $a_i, i = \overline{1, m}$ таких, что $f(a_i) \neq 0$. Поскольку числа $f(a_i)$ имеют конечное количество целых делителей, можно перебрать всевозможные наборы значений для $g(a_i)$. По каждому такому набору построим интерполяционный многочлен $g^*(x)$ степени $m - 1 = n/2$. Если теперь $f(x) \vdots g^*(x)$, к многочленам $g^*(x)$ и $\dfrac{f(x)}{g^*(x)}$ можно применить тот же метод, и так до тех пор, пока все множители не станут неприводимыми. В противном случае, если $\forall g^*(x)\ f(x) \not\vdots\ g^*(x)$, многочлен $f(x)$ уже является неприводимым.

Одномерный алгоритм Кронекера [править | править код]

Запись алгоритма [править | править код]

Дано: $f(x) \in \mathbb{Z}[x]$

Надо: $g(x) \in \mathbb{Z}[x]$

Где: n — степень полинома f, m — степень полинома g, i — целочисленное.

Цикл от $i = 0$ до $[n/2]$

- ***Если*** $(f(i) = 0)$ то
  - $g = x - i$
  - $m = 1$
  - Ответ найден.
- ***Конец если***

Конец цикла

Если (ответ не найден) то

- U – множество делителей $f(0)$ (целочисленных)
- ***Цикл*** от $i = 1$ до $[n/2]$
  - M – множество делителей $f(i)$ (целочисленных)
  - $U :=$ декартово произведение U и M
  - ***Цикл*** для каждого $u \in U$
    - Построить многочлен g степени i, такой, что $g(j) = u(j)$ для $j = 0...i$
    - ***Если*** (f делится на g) то
      - $m = i$
      - Решение успешно найдено, ответ g
    - ***Конец если***
  - ***Конец цикла***
- ***Конец цикла***

Конец если

Конец.

Многомерный алгоритм Кронекера [править | править код]

Запись условий задачи [править | править код]

Пусть D — область целостности с однозначным разложением на множители, $f(x_1, \dots, x_n) \in D[x_1, \dots, x_n]$. Требуется разложить f на неприводимые множители.

Запись алгоритма [править | править код]

Дано: $f \in \mathbb{Z}[x_1, \dots, x_n]$

Надо: G — разложение

Переменные: многочлен $\bar{f} \in \mathbb{Z}[y]$, разложение $\bar{G}$ многочлена $\bar{f}$, множество M элементов типа $\mathbb{Z}$.

Идея реализации: Редуцировать задачу к одномерному случаю, путём введения новой неизвестной и заменой всех переменных достаточно высокими степенями этой неизвестной. Факторизовать получившийся многочлен. Выполнить обратную подстановку, пробным делением убедиться, получено ли желаемое разложение.

Начало

Выбрать целое d большее, чем степени отдельных переменных в f заменить все переменные степенями новой неизвестной y:

$$\bar{f}(y) := S_d(f) = f(y, y^d, \dots, y^{d^{n-1}})$$

Разложить $\bar{f}(y)$ на неприводимые множители, то есть

$$\bar{f}(y) = \bar{g}_1(y) \dots \bar{g}_s(y), \quad \bar{g}_i(y) \in \mathbb{Z}[y], \quad 1 \le i \le s$$

G.число_множителей := 1

$m := 1$

$M := \{1, \dots, s\}$

Цикл пока $m \le [s/2]$

  Цикл для каждого подмножества $i_1, \dots, i_m \subset M$ ***пока*** $m \le [s/2]$

    $g_{i_1,\dots,i_m}(x_1, \dots, x_n) := S_d^{-1}(\bar{g}_{i_1}(y)\bar{g}_{i_2}(y) \dots \bar{g}_{i_m}(y))$

    Если f делится на g ***то***

      G.множитель[G.число_множителей]:=g

      G.число_множителей:=G.число_множителей + 1

      $f := f/g$

      $s := s - m$

      M.удалить$\{i_1, \dots, i_m\}$

    Конец если

  Конец цикла

  $m := m + 1$

Конец цикла

G.множитель[G.число_множителей]:=f

Конец

```
● [alse0722@alse0722-ms7a34 coding_teory]$ /bin/python /home/alse0722/Desktop/univer/coding_teory/fin/n16.py
  5*x1**2*x2 + x1*x2 + 5*x1 + 1  and  2
  [5*x1 + 1, x1*x2 + 1, 1]
○ [alse0722@alse0722-ms7a34 coding_teory]$
```

```
● [alse0722@alse0722-ms7a34 coding_teory]$ /bin/python /home/alse0722/Desktop/univer/coding_teory/fin/n16.py
  x1**3 + x1*x2 - x1 + x2**2 + 2*x2 + 7  and  2
  [x1**3 + x1*x2 - x1 + x2**2 + 2*x2 + 7]
○ [alse0722@alse0722-ms7a34 coding_teory]$
```

```
import itertools
from sympy import *
x = symbols('x')

f = x**5 - x**4 - 2*x**3 - 8*x**2 + 6*x - 1
n = 5


def shift(lst, steps):
    if steps < 0:
        steps = abs(steps)
        for i in range(steps):
            lst.append(lst.pop(0))
    else:
        for i in range(steps):
            lst.insert(0, lst.pop())


def primes():
    def is_odd_prime(n):
        if n % 3 == 0:
            return False
```

```
        i, w = 5, 2
        while i * i <= n:
            if n % i == 0:
                return False
            i += w
            w = 6 - w
        return True
    n, w = 5, 2
    yield from (2, 3, n)
    while True:
        n += w
        if n < 25 or is_odd_prime(n):
            yield n
        w = 6 - w


def prime_facts(n):
    for p in primes():
        if n < p * p:
            break
        t = n
        while t % p == 0:
            t //= p
            yield p


def facts(n):
    dd, tt = [1], []
    for p in primes():
        if n < p * p:
            break
        t, e = n, 1
        while t % p == 0:
            tt += [d * p ** e for d in dd]
            t //= p
            e += 1
        if e > 1:
            dd += tt
            del tt[:]
    if n != dd[-1]:
        dd += [n // d for d in dd]
    return dd


# n = 600851475143
# print(facts(n))


def Kroneker(f, n):
    for i in range(0, int(n/2)+1):
        if f.subs(x, i) == 0:
            g = f - i
            m = 1
            return (g, m)

    U = facts(f.subs(x, 0))
    for i in range(1, int(n/2)+1):
        M = facts(-1 * f.subs(x, i))
        for z in M:
            if M.count(z*(-1)) == 0:
                M.append(z*(-1))
        #print(U, M)
        if i == 1:
```

```
        U1 = []
        for k in U:
            for j in M:
                U1.append([k] + [j])
        U = U1
    else:
        U1 = []
        for k in U:
            for j in M:
                U1.append(k + [j])
        U = U1

    for u in U1:
        #print(i, u, len(U1))
        shift(u, -1)
        g = '%d ' % (u[0])
        for j in range(1, i+1):
            g += '+ %d * x**%d' % (u[j], j)

        g = simplify(g)
        # print(g)
        q, r = div(f, g, domain='QQ')
        #print(f, '|||', g,'|||', q,'|||', r)
        if r == 0:
            m = i
            return(g)


# Функция выдающее множество всех подмножеств множества S
def subsets(S):
    sets = []
    len_S = len(S)
    for i in range(1 << len_S):
        subset = [S[bit] for bit in range(len_S) if i & (1 << bit)]
        sets.append(subset)
    return sets


# Обычный счетчик для перебора всех векторов компоненты которых
# не превосходят некоторого d


def inc(b, d):
    if len(b) != 0:
        b[-1] += 1
        if b[-1] == d:
            # print(inc(b[:-1],d))
            b = inc(b[:-1], d) + [0]
    return b

# Функция возвращает список b такой что
# k = b[0]*d^0 + b[1]*d^1 + ... + b[len_sum]*d^len_sum


def sum_d_b(d, k, len_sum):
    b = [0]*len_sum
    x = 0
    for i in range(0, len_sum):
        x += b[i]*(d**i)
    while x != k:
        x = 0
        b = inc(b, d)
        for i in range(0, len_sum):
```

```
            x += b[i]*(d**i)
    return b

# Выполняем обратное преобразование,то есть переход от одной
# переменной ко многим она принимает одномерный полином от y
# (этот полином входит в раздожение полинома от одной переменной
# полученного в ходе замены),
# число d и кортеж l переменных в многомерном полиноме


def Sd(g_i, d, l):
    deg = degree(g_i, gen=y)
    ans = 0
    for i in range(0, deg+1):
        var = 1
        b_s = sum_d_b(d, i, len(l))
        for j in range(0, len(l)):
            var *= l[j]**b_s[j]
        ans += g_i.coeff(y, i) * var
    return ans

# Получаем полином от нескольких переменных и его старшую степень


def M_Kroneker(f, n):
    f_ = f
    d = n+1
    i = 0
    # производим замены переменных xi на y^(d^i)
    for z in l:
        f_ = f_.subs(z, y**(d**i))
        i += 1
    # получаем разложение полученного полинома
    # от одной переменной на множители
    Gtmp = factor_list(f_)[1]
    G_ = []
    for a in Gtmp:
        for i in range(0, a[1]):
            G_.append(a[0])

    G_count = 1
    m = 1
    M = list(range(0, len(G_)))
    s = int(len(G_)/2)

    G = []
    while m <= s:
        # для каждого подмножества M размера m
        for i_set in subsets(M):
            if len(i_set) == m:
                g = 1
                sd_arg = []
                # вибираемсоответствующие элементы разложения
                for i in i_set:
                    sd_arg.append(G_[i])
                # получим многочлен g как плоизведение обратных преобразований
                # проведенных над полиномами разложения
                for g_i in sd_arg:
                    g *= Sd(g_i, d, l)
                q, r = div(f, g, domain='QQ')
                # если f делится на g
                if r == 0:
                    # то g - делитель
```

```
                G.append(g)
                G_count += 1
                # и f приравниваем частному деления
                f = q
                s = s-m
                for e in i_set:
                    # из M удалим рассмотренное подмножество
                    M.remove(e)
        m += 1
    G.append(f)
    return G


l = symbols('x1 x2')
x1, x2, y = symbols('x1 x2 y')

#f2 = x1**3 + x2**2 + x1*x2 + 2*x2 - x1 +7
#f2 = 2*x1**2 - 5*x1*x2 + 2*x2**2

f2 = 5*x1**2*x2 + x1*x2 + 5*x1 + 1
n2 = 2

print(f2, " and ",n2)

print(M_Kroneker(f2, n2))



U = [[1, 4], [1, 5], [1, 6], [2, 4], [2, 5], [2, 6], [3, 4], [3, 5], [3, 6]]
M = [4, 5, 6]
```