Metoda sita liczbowego

Michał Bartoszuk

I Ogólne informacje o algorytmie

Metoda sita liczbowego jest algorytmem służącym do rozkładu liczb całkowitych na czynniki pierwsze. W praktyce nie jest on używany, ale służy do ułatwienia zrozumienia bardziej efektywnych i zaawansowanych algorytmów faktoryzacji, takich jak:

- Ogólne sito ciała liczbowego (GNFS)
- Szczególne sito ciała liczbowego (SNFS)
- Sito kwadratowe
- Metoda faktoryzacji Dixona

II Sposób działania metody sita liczbowego

II.1 Wstępne zagadnienia i oznaczenia matematyczne

II.1.1 Liczba gładka

Liczba naturalna n jest nazywana B-gładką, jeśli wszystkie jej dzielnik pierwsze są nie większe niż B.

Przykłady:

- $8398 = 19 \cdot 17 \cdot 13 \cdot 2$ zatem liczba 8398 jest 19-gładka, 23-gładka, 29-gładka itd.
- $210 = 7 \cdot 5 \cdot 3 \cdot 2$ zatem liczba 210 jest 7-gładka, 11-gładka, 13-gładka itd.

II.1.2 Baza czynników pierwszych

Baza czynników pierwszych jest skończonym zbiorem wszystkich liczb pierwszych nie większych od ustalonej liczby k.
Oznaczenie: P_k
Przykład: P_{17} ={2, 3, 5, 7, 11, 13, 17}

II.1.3 Iloczyn częściowy

Niech P będzie n-elementowym zbiorem liczb, wtedy iloczyn częściowy tego zbioru jest równy:

$$\prod_{p_i \in P} p_i = p_1 \cdot p_2 \cdot ... \cdot p_n$$

II.1.4 Kongruencja

Kongruencja jest relacją utożsamiającą ze sobą liczby o tej samej reszcie z dzielenia przez n:

$$a \equiv b \pmod{n}$$

Przykłady:

- $56 \equiv 31 \pmod{5}$ bo $55 = 5 \cdot 11 + 1$ i $31 = 5 \cdot 6 + 1$
- $187 \equiv 251 \pmod{16}$ bo $187 = 16 \cdot 11 + 11$ i $251 = 16 \cdot 15 + 11$

Wybrane własności kongruencji:

- Jeśli $a \equiv b$ (mod n) to n | (a - b)
- Kongruencje o tym samym module można dodawać, odejmować, mnożyć stronami oraz dzielić stronami w przypadku gdy liczba, przez którą się dzieli jest względnie pierwsza z modułem.
- Jeśli $a \equiv b$ (mod n) to $a^k \equiv b^k$ (mod n)

II.2 Metoda sita liczbowego

Problem: Faktoryzacja liczby naturalnej złożonej n.
Rozwiązanie metodą sita liczbowego:

1. Wybieramy granicę B i określamy bazę czynników pierwszych P_B. Jeśli któryś z elementów P_B dzieli n, wtedy już znaleźliśmy część (lub wszystkie) czynniki pierwsze n, a algorytm należy kontynuować dla n podzielonego przez te czynniki.

2. Wyszukujemy liczby naturalne z, takie że zarówno z, jak i $z + n$ są B-gładkie czyli wszystkie ich czynniki pierwsze są w zbiorze P_B. Takie liczby możemy zapisać w postaci:

$$z = \prod_{p_i \in P_B} p_i^{a_i}$$

$$z + n = \prod_{p_i \in P_B} p_i^{b_i}$$

dla odpowiednich, nieujemnych a_i, b_i.

3. Każda taka para liczb z i $z + n$ spełnia kongruencję modulo n:

$$\prod_{p_i \in P_B} p_i^{a_i} \equiv \prod_{p_i \in P_B} p_i^{b_i} \text{ (mod n)}$$

bo

$$z \equiv z + n \text{ (mod n)}$$

4. Generujemy wystarczającą liczbę powyższych kongruencji (z reguły trochę większą niż liczba elementów w P_B), aby stworzyć układ, a następnie mnożymy (lub dzielimy) je stronami w taki sposób, by wykładniki wszystkich czynników pierwszych były parzyste. Dzięki tym operacją można otrzymać kongruencję kwadratów w postaci:

$$x^2 \equiv y^2 \text{ (mod n)}$$

5. Powyższą kongruencję można przekształcić w następujący sposób:
   $x^2 \equiv y^2$ (mod n)
   $x^2 - y^2 \equiv 0$ (mod n)
   $(x - y)(x + y) \equiv 0$ (mod n)
   Z własności kongruencji wiadomo, iż (x-y) i (x+y) są podzielne przez n, czyli zawierają czynniki pierwsze n. Aby otrzymać te czynniki należy wyliczyć NWD(x+y, n) oraz NWD(x-y, n), gdyż $n = NWD(x - y, n) \cdot NWD(x + y, n)$. W przypadku gdy otrzymaliśmy dzielnik w postaci liczby złożonej, należy wykonać algorytm dla tego dzielnika (lub użyć innej metody faktoryzacji).

6. W przypadku otrzymania faktoryzacji trywialnej z obliczania powyższych NWD np. $n = n \cdot 1$, powtarzamy tworzenie kongruencji kwadratów, przemnażając stronami w podpunkcie 4. inne kongruencje. W przypadku nietrywialnej faktoryzacji, liczba n została rozłożona na czynniki pierwsze.

II.2.1 Przykład I

Dla $n = 187$

1. Niech B = 7 (losowo, odpowiednio duże), zatem $P_B = \{2, 3, 5, 7\}$. Żaden z elementów P_B nie dzieli 187.
2. Następnie wyszukujemy pary z i $z + n$ spełniające odpowiednie warunki np. (2, 189), (5, 192), (9, 196), (56, 243) i generujemy kongruencje:

   1. (2, 189): $2^1 \cdot 3^0 \cdot 5^0 \cdot 7^0 \equiv 2^0 \cdot 3^3 \cdot 5^0 \cdot 7^1 \pmod{187}$
   2. (5, 192): $2^0 \cdot 3^0 \cdot 5^1 \cdot 7^0 \equiv 2^6 \cdot 3^1 \cdot 5^0 \cdot 7^0 \pmod{187}$
   3. (9, 196): $2^0 \cdot 3^2 \cdot 5^0 \cdot 7^0 \equiv 2^2 \cdot 3^0 \cdot 5^0 \cdot 7^2 \pmod{187}$
   4. (56, 243): $2^3 \cdot 3^0 \cdot 5^0 \cdot 7^1 \equiv 2^0 \cdot 3^5 \cdot 5^0 \cdot 7^0 \pmod{187}$

3. Sposobów na przemnożenie stronami i otrzymanie wykładników parzystych jest kilka, poniżej zostało przemnożone równanie 1. z 4.:

$$2^4 \cdot 3^0 \cdot 5^0 \cdot 7^1 \equiv 2^0 \cdot 3^8 \cdot 5^0 \cdot 7^1 \pmod{187} \ / :7$$

$$2^4 \equiv 3^8 \pmod{187}$$

4. Otrzymaliśmy kongruencję kwadratów, gdyż powyższą relację można zapisać jako:

$$4^2 \equiv 81^2 \pmod{187}$$

5. Możemy dokonać teraz faktoryzacji liczby 187, gdyż $187 = NWD(81 - 4, 187) \cdot NWD(81 + 4, 187) = 11 \cdot 17$.

II.2.2 Przykład II

Dla $n = 11305$

1. Niech B = 13 (losowo, odpowiednio duże), $P_B = \{2, 3, 5, 7, 11, 13\}$. Liczby 5 i 7 dzielą 11305 zatem znaleźliśmy 2 czynniki pierwsze, a od teraz będziemy rozpatrywać $n = 11305/35 = 323$.
2. Następnie wyszukujemy pary z i $z + n$ spełniające odpowiednie warunki np. (2, 325), (7, 330), (20, 343), (27, 350), (40, 363), (52, 375), i generujemy kongruencje:

   1. (2, 325): $2^1 \cdot 3^0 \cdot 5^0 \cdot 7^0 \cdot 11^0 \cdot 13^0 \equiv 2^0 \cdot 3^0 \cdot 5^2 \cdot 7^0 \cdot 11^0 \cdot 13^1 \pmod{323}$

   2. (7, 330): $2^0 \cdot 3^0 \cdot 5^0 \cdot 7^1 \cdot 11^0 \cdot 13^0 \equiv 2^1 \cdot 3^1 \cdot 5^1 \cdot 7^0 \cdot 11^1 \cdot 13^0 \pmod{323}$

   3. (20, 343): $2^2 \cdot 3^0 \cdot 5^1 \cdot 7^0 \cdot 11^0 \cdot 13^0 \equiv 2^0 \cdot 3^0 \cdot 5^0 \cdot 7^3 \cdot 11^0 \cdot 13^0 \pmod{323}$

   4. (27, 350): $2^0 \cdot 3^3 \cdot 5^0 \cdot 7^0 \cdot 11^0 \cdot 13^0 \equiv 2^1 \cdot 3^0 \cdot 5^2 \cdot 7^1 \cdot 11^0 \cdot 13^0 \pmod{323}$

   5. (40, 363): $2^3 \cdot 3^0 \cdot 5^1 \cdot 7^0 \cdot 11^0 \cdot 13^0 \equiv 2^0 \cdot 3^1 \cdot 5^0 \cdot 7^0 \cdot 11^2 \cdot 13^0 \pmod{323}$

   6. (52, 375): $2^2 \cdot 3^0 \cdot 5^0 \cdot 7^0 \cdot 11^0 \cdot 13^1 \equiv 2^0 \cdot 3^1 \cdot 5^3 \cdot 7^0 \cdot 11^0 \cdot 13^0 \pmod{323}$

3. Sposobów na przemnożenie stronami i otrzymanie wykładników parzystych jest kilka, poniżej zostało przemnożone równanie 1. z 5. oraz 6.:

$$2^6 \cdot 3^0 \cdot 5^1 \cdot 7^0 \cdot 11^0 \cdot 13^1 \equiv 2^0 \cdot 3^2 \cdot 5^5 \cdot 7^0 \cdot 11^2 \cdot 13^1 \pmod{323} \ / :(5*13)$$

$$2^6 \equiv 3^2 \cdot 5^4 \cdot 11^2 \pmod{323}$$

4. Otrzymaliśmy kongruencję kwadratów, gdyż powyższą relację można zapisać jako:

$$8^2 \equiv 825^2 \pmod{323}$$

5. Możemy dokonać teraz faktoryzacji liczby 323, gdyż $323 = NWD(825-8, 323) \cdot NWD(825+8, 323) = 19 \cdot 17$ zatem $11305 = 5 \cdot 7 \cdot 17 \cdot 19$.

II.3 Implementacja w Pythonie

```
import math

def check_prime(n): # Test pierwszości
    if n < 2:
        return False

    for i in range(2, int(math.sqrt(n))+1):
        if n % i == 0:
            return False
    return True

def pow_idx(n, p): # Wykładnik danego czynnika pierwszego
    p_idx = 0
    while n % p == 0:
        p_idx += 1
        n /= p
    return int(n), p_idx

def preliminary_check(n, base): # Sprawdzenie czy czynniki n są w bazie czynników pierwszych
    p_indexes = [0]*len(base)

    for i, p in enumerate(base):
        n, p_indexes[i] = pow_idx(n, p)

    return n, p_indexes

def factor_base(B): # Generowanie bazy czynników pierwszych
    fbase = []

    for i in range(2, B+1):
        if check_prime(i):
            fbase.append(i)

    return fbase

# Sprawdzanie czy liczba jest B-gładka (Tak -> zwraca wykładniki czynników pierw.)
def b_smooth(z, base):
    z, p_indexes = preliminary_check(z, base)

    if z != 1:
        return False
    return p_indexes

def gcd(a, b): # Największy wspólny dzielnik (algorytm Euklidesa)
    while b:
        a, b = b, a%b
    return a

def create_relations(n, B): # Wyszukiwanie par z i z+n
    relations = []  # Przechowuje wykładniki czynników pierwszych
    fbase = factor_base(B)
```

```
    for i in range(2, int(n)):
        z_pow = b_smooth(i, fbase)
        z_n_pow = b_smooth(i+n, fbase)

        if z_pow and z_n_pow: # z i z+n są b-gładkie
            relations.append([z_pow, z_n_pow])

        if len(relations) >= 2*len(fbase):
            break

    return relations

def sum_list(list_a, list_b): # Dodawanie elementów 2 list
    return [a + b for a,b in zip(list_a, list_b)]

def even(pow_idxes1, pow_idxes2): # Sprawdza czy wykładniki po odejmowaniu są parzyste
    L = sum_list(pow_idxes1[0], pow_idxes2[0])
    P = sum_list(pow_idxes1[1], pow_idxes2[1])
    for l, p in zip(L, P):
        if (l-p) % 2 != 0:
            return False

    return [L, P]

def get_factors(L, P, n, base):
    # Dzięki funkcji even() wiemy, że wykładniki nieparzyste możemy odjąć i otrzymać parzyste
    evenL = []
    evenP = []

    for l, p in zip(L, P):
        if p%2 == 1:  # Skoro wykładnik po prawej jest nieparzysty to po lewej też musi^
            if l >= p:
                evenL.append(l-p)
                evenP.append(0)
            else:
                evenP.append(p-l)
                evenL.append(0)
        else:
            evenL.append(l)
            evenP.append(p)

    l_number = 1
    p_number = 1

    for i, prime in enumerate(base):
        # Dzielenie przez 2 aby otrzymac x, a nie x^2 w kongruencji kwad.
        l_number *= pow(prime, evenL[i]/2)
        p_number *= pow(prime, evenP[i]/2)

    factor1 = gcd(abs(l_number - p_number), n)
    factor2 = gcd(l_number + p_number, n)

    if factor1 == 1 or factor2 == 1: # Faktoryzacja trywialna
        return False

    new_factors = []

    if not check_prime(factor1): # Sprawdanie czy czynnik jest l. złożoną
        new_factors += rational_sieve(int(factor1), base[-1], base)
    else:
        new_factors.append(factor1)
```

```
    if not check_prime(factor2):
        new_factors += rational_sieve(int(factor2), base[-1], base)
    else:
        new_factors.append(factor2)

    return new_factors


def combine_relations(n, relations, base): # Tworzenie kongruencji kwadratowej

    for i, r1 in enumerate(relations):
        for r2 in relations[i+1:]:
            quadratic = even(r1, r2)

            if quadratic:
                factors = get_factors(quadratic[0], quadratic[1], n, base)
                if factors:
                    return factors

    return False

def rational_sieve(n, B, primeBase=[]):

    if primeBase == []:
        primeBase = factor_base(B)

    n, factors_idxs = preliminary_check(n, primeBase)
    factors=[[primeBase[i],factors_idxs[i]] for i in range(len(factors_idxs)) if factors_idxs[i]]

    while n > 1:
        if check_prime(n):
            factors.append([n, 1])
            break

        relations = create_relations(n, B)
        new_factors = combine_relations(n, relations, primeBase)

        if not new_factors:
            print("Faktoryzacja nie powiodła się, liczba postaci p^m lub źle dobrany parameter B")
            return False

        else:
            for f in new_factors:
                if isinstance(f, list):
                    # W przypadku gdy NWD zwróciło l. złożoną,
                    # to zwrócona została lista (rational_sieve() dla dzielnika)
                    factors.append(f)
                    n /= pow(f[0], f[1])
                else:
                    n, f_idx = pow_idx(n, f)
                    factors.append([int(f), f_idx])

    return factors

if __name__ == "__main__":
    # W przypadku zmiany testów sprawdzić sys.maxsize
    tests = [(23423454, 580), (45234523423, 3600),
    (5523452342346, 30000), (7523452342312, 30000), (8523452343241, 30000)]
    # Poprawność testów sprawdzona z Wolframalpha
```

```
base30000 = factor_base(30000)

for t in tests:
    if t[1] == 30000:
        factors = rational_sieve(t[0], t[1], base30000)
    else:
        factors = rational_sieve(t[0], t[1])

    if factors:
        result = f"{t[0]} = "
        for f in factors:
            result += f"{f[0]}^{f[1]} * "

        print(result.rstrip(" *"))
```

III Złożoność obliczeniowa i ograniczenia

Złożoność obliczeniowa zależy nie tylko od liczby faktoryzowanej n, ale również od wyboru granicy B. Optymalnym wyborem B jest:

$$B = exp((\sqrt{2}/2 + o(1)) \cdot (log_2 n)^{1/2} \cdot (log_2 log_2 n)^{1/2})$$

Dla tak wyznaczonego B, czas wykonywania algorytmu wynosi:

$$exp((\sqrt{2} + o(1)) \cdot (log_2 n)^{1/2} \cdot (log_2 log_2 n)^{1/2})$$

Metodą sita liczbowego (jaki i GNFS) nie można znaleźć czynników pierwszych liczby postaci p^m, gdzie p jest liczbą pierwszą większą od granicy B, a m jest całkowite. Można ten problem w dość prosty sposób rozwiązać, sprawdzając czy n jest takiej postaci. Najszybszym sposobem na sprawdzenie tego jest użycie testów pierwszeństwa klasy AKS.

Większym problemem jest znalezienie wystarczającej liczby wartości z, takich że z i $z + n$ jest B-gładka. Im większa liczba, tym trudniej znaleźć liczby z będące B-gładkie dla danego B. W przypadku gdy n jest rzędu 10^{100} lub większa, znalezienie liczb z, które pozwolą na otrzymanie kongruencji kwadratowej jest bardzo mało prawdopodobne lub niemożliwe.

IV Bibliografia

- A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, The factorization of the ninth Fermat Number
- A. K. Lenstra, H. W. Lenstra, Jr. (eds.) The Development of the Number Field Sieve
- Steve Byrnes, The Number Field Sieve