

Projekt Zespołowy

Etap projektu – projektowanie  
rozwiązania na zadaną  
architekturę

Autorzy:  
Biernacka Kamila  
Kania Dominik  
Leśniak Mateusz  
Maziarz Wojciech

kwiecień 2021

## Streszczenie

Poniższe sprawozdanie jest wynikiem pracy na drugim etapie projektu zespołowego z implementacji metody indeksu w architekturach GPU. Przedstawiono w nim przygotowane przez zespół projekty i rysunki koncepcyjne wymaganych do zaimplementowania algorytmów.

## Spis treści

<b>1</b>	<b>Mnożenie modularne dużych liczb</b>	<b>3</b>
<b>2</b>	<b>Poszukiwanie relacji i faktoryzacja w bazie</b>	<b>4</b>
2.1	Szybkie potęgowanie modularne . . . . .	4
2.2	Faktoryzacja w bazie . . . . .	4
2.3	Budowa relacji . . . . .	5
<b>3</b>	<b>Eliminacja Gaussa w pierścieniu <math>\mathbb{Z}_{p-1}</math></b>	<b>6</b>
3.1	Algorytm Euklidesa . . . . .	6
3.2	Rozszerzony algorytm Euklidesa . . . . .	6

# 1 Mnożenie modularne dużych liczb

W celu wykonania mnożenia dużych liczb  $a, b \in \mathbb{F}_p$  wykorzystany zostanie algorytm 2. Pierwszym krokiem jest przedstawienie liczb  $a, b$  w postaci  $a = x_1 \cdot 2^{32} + y_1$  oraz  $b = x_2 \cdot 2^{32} + y_2$ .

Wtedy

$$r_p(ab) = r_p((x_1 \cdot 2^{32} + y_1)(x_2 \cdot 2^{32} + y_2)) = r_p(x_1 x_2 \cdot 2^{64}) +_p r_p(x_1 y_2 \cdot 2^{32}) +_p r_p(x_2 y_1 \cdot 2^{32}) +_p r_p(y_1 y_2).$$

Do wyznaczenia pośrednich wartości  $r_p$  wykorzystywany jest algorytm 1. Algorytm mnożenia pośredniego działa analogicznie do algorytmu 2. Różnicą jest przedstawienie czynników jako  $x \cdot 2^{16} + y$ .

---

## Algorithm 1: Mnożenie pośrednie, `halfMult`

---

**Input:**  $a, b$  - dwie liczby całkowite,  $p$  - modułnik

**Output:** *result* - wynik mnożenia

```

1  $x_1 \leftarrow a \gg 16$ 
2  $y_1 \leftarrow a \&\& 0xffff$ 
3  $x_2 \leftarrow b \gg 16$ 
4  $y_2 \leftarrow b \&\& 0xffff$ 
5  $half_a \leftarrow (((x_1 x_2) \% p) \cdot r_p(2^{32})) \% p$ 
6  $half_b \leftarrow (((x_1 y_2) \% p) \cdot r_p(2^{16})) \% p$ 
7  $half_c \leftarrow (((x_2 y_1) \% p) \cdot r_p(2^{16})) \% p$ 
8  $half_d \leftarrow (y_1 y_2) \% p$ 
9 return  $(half_a + half_b + half_c + half_d) \% p$ 
```

---



---

## Algorithm 2: Pełne mnożenie modularne dwóch liczb, `mult`

---

**Input:**  $a, b$  - dwie liczby całkowite,  $p$  - modułnik

**Output:** *result* - wynik mnożenia

```

1  $x_1 \leftarrow a \gg 32$ 
2  $y_1 \leftarrow a \&\& 0xffffffff$ 
3  $x_2 \leftarrow b \gg 32$ 
4  $y_2 \leftarrow b \&\& 0xffffffff$ 
5  $half_a \leftarrow (halfMult(x_1, x_2) \cdot r_p(2^{64})) \% p$ 
6  $half_b \leftarrow (halfMult(x_1, y_2) \cdot r_p(2^{32})) \% p$ 
7  $half_c \leftarrow (halfMult(x_2, y_1) \cdot r_p(2^{32})) \% p$ 
8  $half_d \leftarrow halfMult(y_1, y_2)$ 
9 return  $(half_a + half_b + half_c + half_d) \% p$ 
```

---

## 2 Poszukiwanie relacji i faktoryzacja w bazie

### 2.1 Szybkie potęgowanie modularne

Metoda indeksu wymaga obliczenia wartości typu  $a^b \bmod n$ . Szybkie potęgowanie modularne jest prostym algorytmem pozwalającym zredukować liczbę mnożeń i dzielení modulo z  $b$  do  $O(\log b)$ .

---

**Algorithm 3:** Szybkie potęgowanie modularne, **fastPow**

---

**Input :** podstawa potęgi  $a$ , wykładnik potęgi  $b$ , modułnik  $n$

**Output:**  $a^b \bmod n$

```
1 bits ← to_bin(b)
2 nbits ← length(bits)
3 a ← a%n
4 result ← 1
5 x ← a
6 for i ← 0 to nbits do
7   if bits[i] == 1 then
8     result ← result * x
9     result ← result%n
10  x ← x * x
11  x ← x%n
12 return result
```

---

### 2.2 Faktoryzacja w bazie

Dana jest baza  $\mathcal{N} = \{2, 3, \dots, p_k\}$ , gdzie  $p_i \in \mathcal{P}$  i  $p_k$  jest największą liczbą pierwszą mniejszą  $B$ . W celu faktoryzacji wykorzystany zostanie algorytm 4.

---

**Algorithm 4:** Faktoryzacja w bazie, **factor**

---

**Input:**  $a$  - faktoryzowana liczba,  $\mathcal{N}$  - baza rozkładu

**Output:**  $result = [e_1, e_2, \dots, e_k]$  - czynniki

```
1 for i ← 0 to k do
2   counter ← 0
3   while a % N[i] do
4     a ← a / N[i]
5     counter ← counter + 1
6   end
7   result[i] ← counter
8 end
9 return result
```

---

---

**Algorithm 5:** Sprawdzenie czy liczba faktoryzuje się w wybranej bazie, `IsFactor`

---

**Input:**  $a$  - faktoryzowana liczba,  $\mathcal{N}$  - baza rozkładu  
**Output:**  $result = [e_1, e_2, \dots, e_k]$  - czynniki

```

1 for  $i \leftarrow 0$  to  $k$  do
2    $counter \leftarrow 0$ 
3   while  $a \% N[i]$  do
4      $a \leftarrow \frac{a}{N[i]}$ 
5      $counter \leftarrow counter + 1$ 
6   end
7    $result[i] \leftarrow counter$ 
8 end
9 if  $a == 1$  then
10  return True
11 end
12 else
13  return False
14 end

```

---

## 2.3 Budowa relacji

W celu zbudowania relacji  $\mathcal{R}$  generowanych jest  $l$  liczb losowych  $e_i$ . Następnie, przy wykorzystaniu algorytmu 3, wyznaczone są wartości  $a^{e_i}$ . Niech  $e$  będzie tablicą  $l$ -elementową, jako  $fastPow(a, e, p)$  rozumiane jest jednoczesne wykonanie  $fastPow(a, e_i, p)$  dla każdego elementu tablicy z wykorzystaniem karty graficznej.

---

**Algorithm 6:** Budowa relacji, `relationBuild`

---

**Input:**  $a$  - generator,  $l$  - wielkość zbioru relacji,  $p$  - modułnik  
**Output:**  $R$  - zbiór relacji

```

1 for  $i \leftarrow 0$  to  $l$  do
2    $e[i] \leftarrow randomInt()$ 
3 end
4  $R \leftarrow fastPow(a, e, p)$ 
5 for  $i \leftarrow 0$  to  $l$  do
6   while not  $isFactored(R[i])$  do
7      $R[i] \leftarrow fastPow(a, randomInt(), p)$ 
8   end
9 end
10 return  $R$ 

```

---

### 3 Eliminacja Gaussa w pierścieniu $\mathbb{Z}_{p-1}$

#### 3.1 Algorytm Euklidesa

Poniższy schemat wykorzystuje algorytm Euklidesa do sprawdzenia, czy podane na wejściu dwie liczby są względnie pierwsze.

---

**Algorithm 7:** Algorytm Euklidesa

---

**Input:**  $a, b$  - liczby naturalne

**Output:** True, jeśli  $\gcd(a, b) == 1$ , False w przeciwnym przypadku.

```
1 while  $b \neq 0$  do
2   |  $temp := b$ 
3   |  $b := a \bmod b$ 
4   |  $a := temp$ 
5 if  $a == 1$  then
6   |  $output := True$ 
7 else
8   |  $output := False$ 
```

**Result:** output

---

#### 3.2 Rozszerzony algorytm Euklidesa

Tożsamość Bezout mówi, że liczby  $a$  i  $p$  są względnie pierwsze i wtedy i tylko wtedy, gdy istnieją takie liczby  $s$  i  $t$ , że

$$ps + at = 1,$$

Wówczas, po zredukowaniu tej równości modulo  $p$ , otrzymuje się

$$at \equiv 1 \pmod{p},$$

czyli  $t$  jest elementem odwrotnym  $a$  w pierścieniu  $\mathbb{Z}_p$ .

Poniższy algorytm odnajduje element odwrotny do elementu  $a$  w pierścieniu  $\mathbb{Z}_p$ .

---

**Algorithm 8:** Rozszerzony Algorytm Euklidesa

---

**Input:**  $a, p$  - liczby naturalne

**Output:**  $x = a^{-1} \bmod p$  lub informacja, że taka liczba nie istnieje

```
1  $u \leftarrow 1, w \leftarrow a, x \leftarrow 0, z \leftarrow p$ 
2 while  $w \neq 0$  do
3   if  $w < z$  then
4      $\text{swap}(u, x)$ 
5      $\text{swap}(w, z)$ 
6    $q \leftarrow w / z - (w \% z)$ 
7    $u \leftarrow u - q \cdot x$   $w \leftarrow w - q \cdot z$ 
8 if  $z \neq 1$  then
9   Result None
10 if  $x < 0$  then
11    $x \leftarrow x + p$ 
Result:  $x$ 
```

---