

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

1	Oznaczenia	3
2	Mnożenie modularne dużych liczb	4
3	Poszukiwanie relacji i faktoryzacja w bazie	5
3.1	Szybkie potęgowanie modularne	5
3.2	Faktoryzacja w bazie	7
3.3	Budowa relacji	9
4	Eliminacja Gaussa w pierścieniu \mathbb{Z}_{p-1}	11
4.1	Algorytm Euklidesa	11
4.2	Rozszerzony algorytm Euklidesa	12
5	Eliminacja Gaussa	14

1 Oznaczenia

Tablica 1: Oznaczenia funkcji i operatorów. *Źródło: opracowanie własne*

Symbol	Opis
>>	przesunięcie bitowe
&&	koniunkcja bitowa
%	reszta z dzielenia
to_bin()	zwraca postać binarną argumentu
length()	zwraca długość argumentu
randomInt()	zwraca losową liczbę całkowitą
swap(a,b)	zmiennej a przypisuje wartość zmiennej b i odwrotnie

2 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

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

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  $(half_a + half_b + half_c + half_d) \% p$ 
```

3 Poszukiwanie relacji i faktoryzacja w bazie

3.1 Szybkie potęgowanie modularne

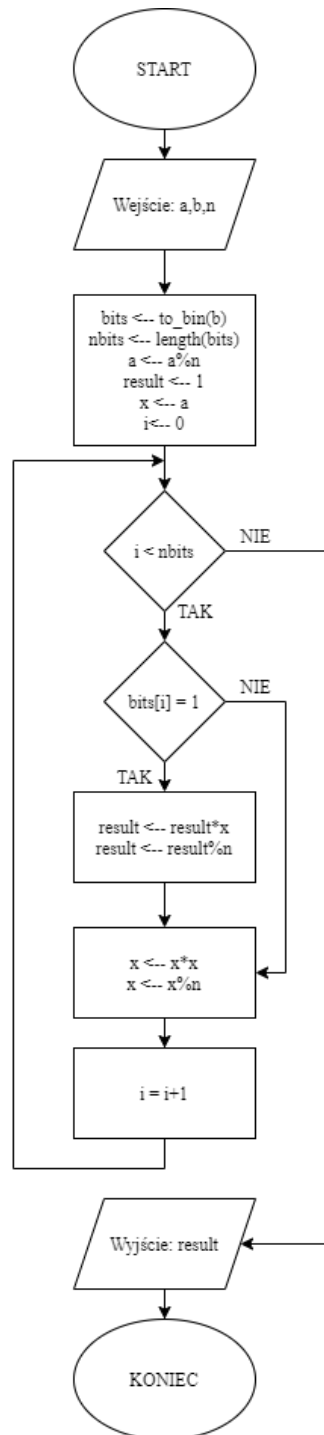
Metoda indeksu wymaga obliczenia wartości typu $a^b \bmod n$. Szybkie potęgowanie modularne jest prostym algorytmem pozwalającym zredukować liczbę mnożeń i dzieleń modulo n do $O(\log b)$.

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

Input : podstawa potęgi a , wykładnik potęgi b , moduł n

Output: $a^b \bmod n$

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



Rysunek 1: Schemat blokowy algorytmu 3

3.2 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**

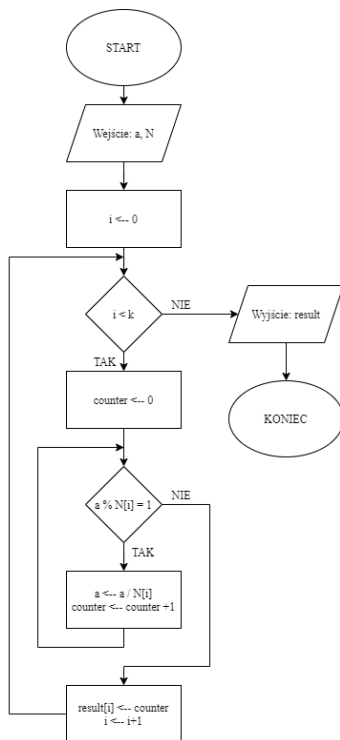
Input: a - faktoryzowana liczba, \mathcal{N} - baza rozkładu

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

```

1 for  $i \leftarrow 0$  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 return  $result$ 

```



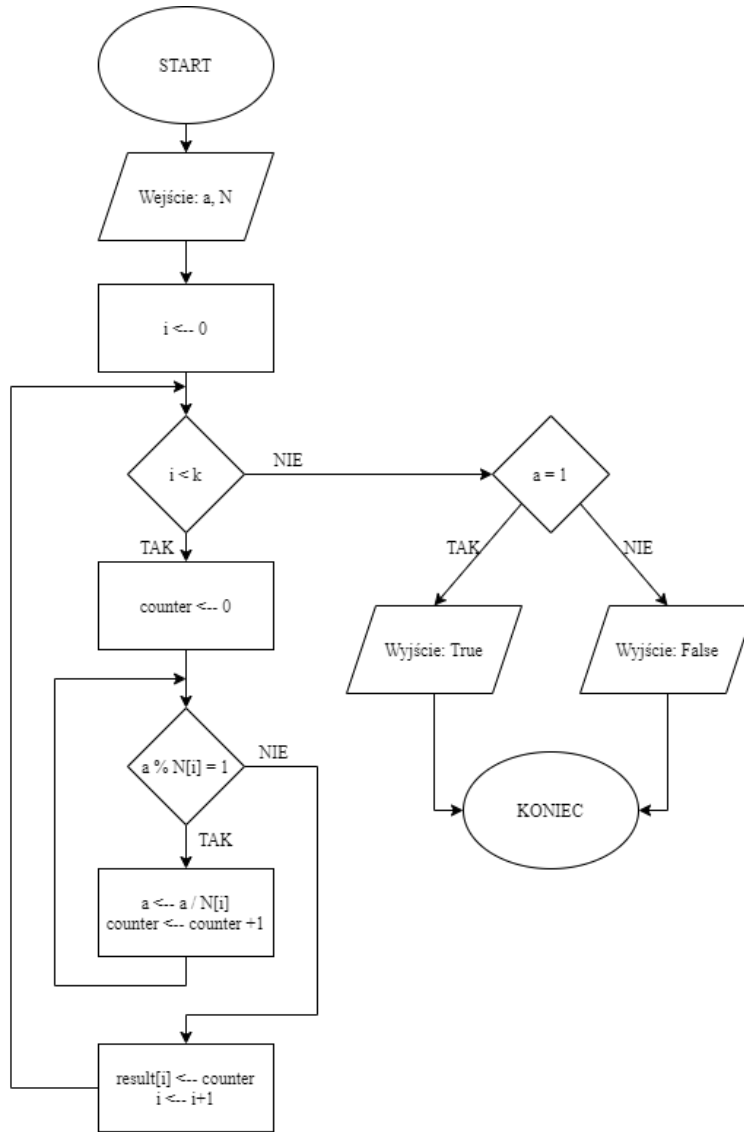
Rysunek 2: Schemat blokowy algorytmu 4

Algorithm 5: Sprawdzenie czy liczba faktoryzuje się w wybranej bazie,
IsFactored

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
```



Rysunek 3: Schemat blokowy algorytmu 5

3.3 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 wykorzy-

staniem procesora graficznego.

Algorithm 6: Budowa relacji, `relationBuild`

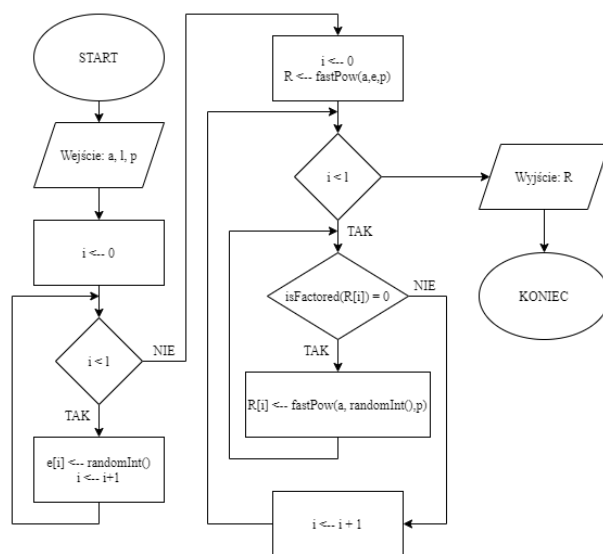
Input: a - generator, l - wielkość zbioru relacji, p - modułnik

Output: R - zbiór relacji

```

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

```



Rysunek 4: Schemat blokowy algorytmu 6

4 Eliminacja Gaussa w pierścieniu \mathbb{Z}_{p-1}

4.1 Algorytm Euklidesa

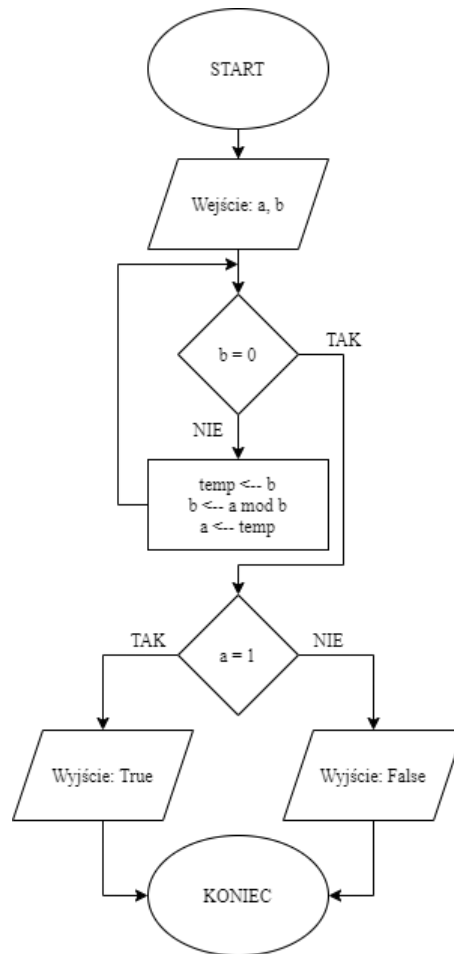
Poniższy schemat wykorzystuje algorytm Euklidesa do sprawdzenia, czy podane na wejściu dwie liczby są względnie pierwsze.

Algorithm 7: Algorytm Euklidesa, `isInversible`

Input: a, b - liczby naturalne

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

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



Rysunek 5: Schemat blokowy algorytmu 7

4.2 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

$$at \equiv 1 \bmod p,$$

czyli t jest elementem odwrotnym a w pierścieniu Z_p .

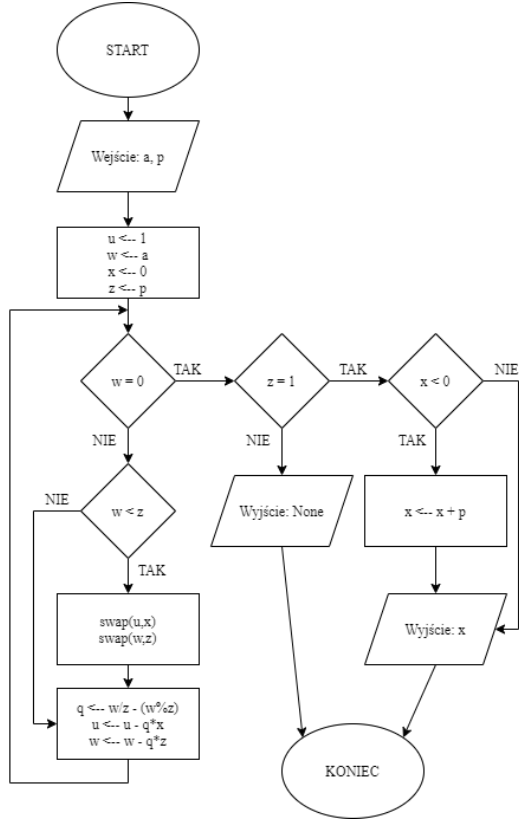
Wykorzystując poniższy algorytm możemy znaleźć odwrotność w dowolnym pierścieniu liczbowym.

Algorithm 8: Rozszerzony Algorytm Euklidesa, *inverse*

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   return None
10 if  $x < 0$  then
11    $x \leftarrow x + p$ 
12 return  $x$ 
```



Rysunek 6: Schemat blokowy algorytmu 8

5 Eliminacja Gaussa

Niech

$$\mathbb{A}_{l,k} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,k} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,k} \\ \vdots & \vdots & & \vdots \\ a_{l,0} & a_{l,1} & \cdots & a_{l,k} \end{bmatrix}.$$

Jako $\mathbb{A}_i = [a_{i,0}, a_{i,1}, \dots, a_{i,k}]$ rozumiany jest i -wiersz macierzy. Wtedy, następujący zapis $\mathbb{C} = \mathbb{A}_n - \mathbb{A}_m$, gdzie $\mathbb{C} = [c_0, c_1, \dots, c_k]$ oznacza jednoczesne wykonanie działania $c_j = \mathbb{A}_{n,j} - \mathbb{A}_{m,j}$ dla każdego elementu wiersza z wykorzystaniem procesora graficznego. Natomiast $\mathbb{C} = \text{mult}(\mathbb{A}_m, b)$ rozumiany jest jako jednoczesne wykonanie $\text{mult}(a_{m,j}, b)$. Zapis $\mathbb{A}_{l+1,k} \leftarrow \frac{\mathbb{A}_{l,k}}{\text{vector}}$ oznacza dopisanie

wiersza *vector* jako ostatni wiersz macierzy.

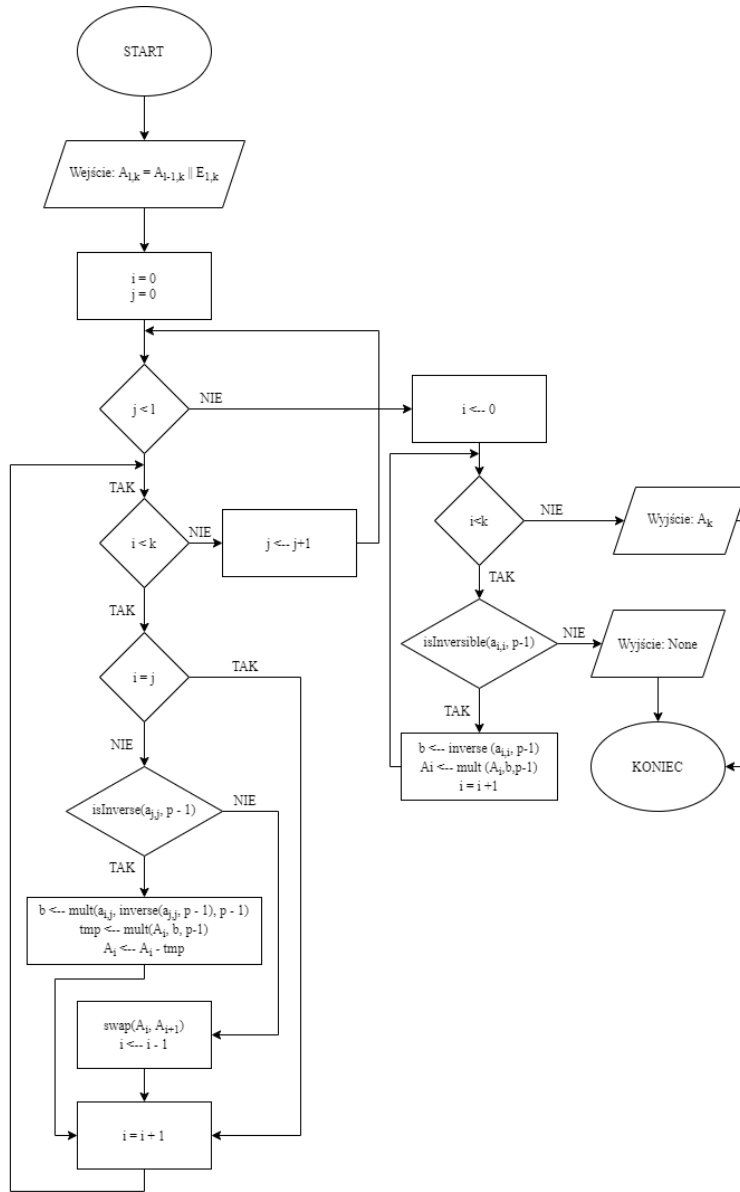
Algorithm 9: Algorytm eliminacji Gaussa, Gauss

Input: $\mathbb{A}_{l,k} = \mathbb{A}_{l-1,k} \parallel \mathbb{E}_{1,k}$
Output: $\mathbb{X}_{1,k}$

```

1 for  $j \leftarrow 0$  to  $l$  do
2   for  $i \leftarrow 0$  to  $k$  do
3     if  $i \neq j$  then
4        $inverseIdx \leftarrow inverseIndex(i)$ 
5       while  $inverseIdx == -1$  and not  $isInversible(a_{j,j})$  do
6          $\mathbb{A}_{l+1,k} \leftarrow \frac{\mathbb{A}_{l,k}}{relationBuild(a,1,p)}$ 
7          $l \leftarrow l + 1$ 
8          $inverseIdx \leftarrow inverseIndex(i)$ 
9          $swap(\mathbb{A}_i, \mathbb{A}_{inverseIdx})$ 
10      end
11       $b \leftarrow mult(a_{i,j}, inverse(a_{j,j}, p-1), p-1)$ 
12       $tmp \leftarrow mult(\mathbb{A}_i, b, p-1)$ 
13       $\mathbb{A}_i \leftarrow \mathbb{A}_i - tmp$ 
14    end
15  end
16 end
17 for  $i \leftarrow 0$  to  $k$  do
18    $b \leftarrow inverse(a_{i,i}, p-1)$ 
19    $\mathbb{A}_i \leftarrow mult(\mathbb{A}_i, b, p-1)$ 
20 end
21 return  $\mathbb{A}_k$ 

```



Rysunek 7: Schemat blokowy algorytmu 9