Algorytmiczne Aspekty Kryptografii
Potęgowanie modularne

Krzysztof Sikora

27 maja 2002

Spis treści

1 Wstęp

W systemie kryptograficznym RSA wykorzystywana jest operacja podnoszenia jednej liczby do potęgi modulo druga liczba, zwane także potęgowaniem modularnym. Stosuje się ją zarówno przy szyfrowaniu jak i deszyfrowaniu wiadomości. Ma ona także zastosowania w procedurach sprawdzających czy dana liczba jest liczbą pierwszą. Dokładniej, musimy obliczyć wyrażenie:

$$M^e \bmod n.$$

Zauważmy, że aby je uzyskać nie musimy najpierw obliczać potęgi, a dopiero potem brać resztę. Tymczasowe wyniki mogą być redukowane modulo n przy każdym kroku. Gdybyśmy tak nie postępowali pamięć potrzebna do obliczania liczby M^e byłaby ogromna. Załóżmy, że M i e mają po 256 bitów każda, wtedy do zapamiętania wyniku potęgowania koniecznych byłoby

$$\log_2(M^e) = e \cdot \log_2 M = 2^{256} \cdot 256 = 2^{264} \approx 10^{80}$$

bitów, jak widać jest to nieosiągalna wielkość dla współczesnych komputerów. Zapis takiego złego rozwiązania przedstawia Algorytm 1.

Algorytm 1

1: **if** $e = 0$ **then**
2: $\quad C := 1$
3: **else**
4: $\quad C := M$
5: **end if**
6: **for** $i := 2$ **to** e **do**
7: $\quad C := C \cdot M$
8: **end for**
9: $C := C\ (mod\ n)$

Czas działania tego algorytmu jest oczywiście wykładniczy ze względu na długość zapisu binarnego liczb e lub M, chciażby dlatego, że wykonuje on $O(e)$ obrotów pętli. Nie polepszy go nawet zastosowanie potęgowania binarnego. Wtedy mielibyśmy tylko $O(\log e)$ iteracji pętli z Kroku 6, jednak nie zmienilaby się przecież długosc C. W każdym kroku wzrastałaby podwójnie (w pierwotnym sposobie o długość M) co powodowuje że w obydwu przypadkach algorytm staje się bezużyteczny.

2 Metoda "naiwna"

Algorytm 2 przedstawia pseudokod metody naiwnej.

Algorytm 2 naive

1: **if** $e = 0$ **then**
2: $\quad C := 1$
3: **else**
4: $\quad C := M$
5: **end if**
6: **for** $i := 2$ **to** e **do**
7: $\quad C := C \cdot M\ (mod\ n)$
8: **end for**

Widać, że wykonuje $e - 1$ modularnych mnożeń. Jego złożność jest zatem wykładnicza ze względu na długość zapisu binarnego e.

3 Metoda binarna

Inna nazwa: Podnoszenie do kwadratu i mnożenie. Stanowi ona efektywne rozwiązanie problemu. Połączenie pomysłów z dwóch wcześniejszych punktów: szybkie potęgowanie oraz branie reszty w każdym obrocie pętli. Podnoszenie do kwadratu odbywa się w każdym kroku, a wykonanie mnożenia jest zależe od zapisu binarnego wykładnika e. Niech k będzie liczbą bitów e i niech

$$(e_{k-1}e_{k-2}\dots e_0)_2 = \sum_{i=0}^{k-1} e_i 2^i$$

będzie binarną reprezentacją e.

Algorytm 3 binary

```
1:  if e_{k-1} = 0 then
2:     C := 1
3:  else
4:     C := M
5:  end if
6:  for i := k - 2 downto 0 do
7:     C := C^2 (mod n)
8:     if e_i = 1 then
9:        C := C · M (mod n)
10:    end if
11: end for
```

Dla k-bitowej liczby e, metoda binarna wymaga:

- (Krok 7) $k-1$ potęgowań
- (Krok 9) $H(e)-1$ mnożeń, gdzie $H(e)$ jest wagą Hamminga (liczba jedynek w zapisie binarnym) e

Załóżmy, że $e > 0$ i $e_{k-1} = 1$, mamy $0 \leq H(e) - 1 \leq k-1$. Całkowita liczba mnożeń jest równa:

- Maksimum: $(k-1) + (k-1) = 2(k-1)$
- Minimum: $(k-1) + 0 = k-1$
- Średnia: $(k-1) + \frac{1}{2}(k-1) = \frac{3}{2}(k-1)$

Algorytm wykonuje $O(k)$ obrotów pętli. Każdy z kroków zajmuje $O(k^2)$ (zakładając, że M i n są k-bitowe). W sumie $O(k^3)$, k jest oczywiście $O(\log e)$.

Przykład Niech $e = 250 = (11111010)_2$, długość zapisu binarnego $k = 8$. Początkowo $C := M$ gdyż $e_{k-1} = e_7 = 1$. Iteracja pętli przebiega następująco:

i	**Krok 7**	c_i	**Krok 9**
6	$(M)^2 = M^2$	1	$M^2 \cdot M = M^3$
5	$(M^3)^2 = M^6$	1	$M^6 \cdot M = M^7$
4	$(M^7)^2 = M^{14}$	1	$M^{14} \cdot M = M^{15}$
3	$(M^{15})^2 = M^{30}$	1	$M^{30} \cdot M = M^{31}$
2	$(M^{31})^2 = M^{62}$	0	M^{62}
1	$(M^{62})^2 = M^{124}$	1	$M^{124} \cdot M = M^{125}$
0	$(M^{125})^2 = M^{250}$	0	M^{250}

Liczba mnożeń równa się $7 + 5 = 12$.

4 Metoda m-arna

Jest generalizacja metody binarnej. Polega na łaczeniu $\log_2 m$ bitów e, zamiast braniu tylko jednego. Niech $e = (e_{k-1}, e_{k-2}, \dots, e_0)_2$. Reprezentacja binarna jest dzielona na s r-bitowych bloków. Jeśli r nie dzieli k, to dostawiane są zera na początek. Niech

$$F_i = (e_{ir+r-1} e_{ir+r-2} \dots e_{ir})_2 = \sum_{j=0}^{r-1} e_{ir+j} 2^j.$$

Zauważmy, że $0 \leq F_i < m$ i $e = \sum_{i=0}^{s-1} F_i 2^{ir}$. Metoda m-arna najpierw oblicza wartości $M^w \ (mod\, n)$ dla $w = 2, 3, \dots, m-1$. Potem podobnie jak w metodzie binarnej. Pseduokod metody m-arnej jest przedstawiony w Algorytmie 4.

Algorytm 4 m-ary

1: Oblicz i zapamiętaj M^w $(mod\ n)$ dla $w = 2, 3, \dots, m-1$.
2: Stwórz słowa F_i dla $i = 0, 1, \dots, s-1$.
3: $C := M^{F_{s-1}}$ $(mod\ n)$
4: **for** $i := s-2$ **downto** 0 **do**
5: $\quad C := C^{2^r}$ $(mod\ n)$
6: $\quad$ **if** $F_i \neq 0$ **then**
7: $\quad\quad C := C \cdot M^{F_i}$ $(mod\ n)$
8: $\quad$ **end if**
9: **end for**

4.1 Metoda ósemkowa

Grupowanie bitów po 3. Niech $e = 250 = (011\ 111\ 010)_2$ wtedy $s = k/r = 9/3 = 3$. W Kroku 2 obliczamy:

bity	w	M^w
000	0	1
001	1	M
010	2	$M \cdot M = M^2$
011	3	$M^2 \cdot M = M^3$
100	4	$M^3 \cdot M = M^4$
101	5	$M^4 \cdot M = M^5$
110	6	$M^5 \cdot M = M^6$
111	7	$M^6 \cdot M = M^7$

Początkowo $C := M^{F_2} = M^3$ $(mod\ n)$. Iteracja pętli przebiega następująco:

i	**Krok 5**	F_i	**Krok 7**
1	$(M^3)^8 = M^{24}$	111	$M^{24} \cdot M^7 = M^{31}$
0	$(M^{31})^8 = M^{248}$	010	$M^{248} \cdot M^2 = M^{250}$

Liczba mnożeń równa się $6 + 6 + 2 = 14$. Jednakże nie musimy obliczać wszystkich M^w w Kroku 1. W przykładzie potrzebujemy tylko 3, 7 i 2, a te liczby możemy obliczyć przy pomocy tylko 4 mnożeń.

bity	w	M^w
000	0	1
001	1	M
010	2	$M \cdot M = M^2$
011	3	$M^2 \cdot M = M^3$
100	4	$M^3 \cdot M = M^4$
111	7	$M^4 \cdot M^3 = M^7$

Teraz liczba wszystkich mnożeń równa się $4 + 6 + 2 = 12$.

Podsumowując metodę m-arną policzmy średnią liczbę mnożeń i potęgowań modularnych. Załóżmy, że $2^r = m$ i r dzieli k.

- Krok 1: $m - 2 = 2^r - 2$
- Krok 5: $(\frac{k}{r} - 1) \cdot r = k - r$
- Krok 7: $(\frac{k}{r} - 1)(1 - \frac{1}{m}) = (\frac{k}{r} - 1)(1 - 2^{-r})$

W sumie

$$2^r - 2 + k - r + (\frac{k}{r} - 1)(1 - 2^{-r})$$

średnio. Podstawiając $r = 1$ i $m = 2$ wyjdzie $\frac{3}{2}(k - 1)$ tak jak uzyskaliśmy w metodzie binarnej. Należy zaznaczyć, że istnieje optymalne $r = r^*$ dla każdego k, takie że średnia liczba mnożeń w metodzie m-arnej jest minimalna.

k	**binarna**	m**-arna**	r^*	**oszczędność %**
8	11	10	2	9.1
16	23	21	2	8.6
32	47	43	2,3	8.5
64	95	85	3	10.5
128	191	167	3,4	12.6
256	383	325	4	15.1
512	767	635	5	17.2
1024	1535	1246	5	18.8
2048	3071	2439	6	20.6

Asymptotyczna wartość oszczęności w metodzie m-arnej wynosi 33 %.

$$\lim_{k \to \infty} \frac{2^r - 2 + k - r + (\frac{k}{r} - 1)(1 - 2^{-r})}{\frac{3}{2}(k - 1)} = \frac{2}{3}(1 + \frac{1 - 2^{-r}}{r}) \approx \frac{2}{3}$$

5 Przerobione metody m-arne

Jak już zauważyliśmy wcześniej, liczba działań w preprocessingu (Krok 1) może być zredukowana jeśli podział wykładnika e w postaci binarnej nie zawiera wszystkich możliwych podziałów, a tym samym wartości w. Jeśli także przerobimy algorytm żeby dzielił wykładnik na serię słów zerowych i niezerowych to zmniejszymy liczbę mnożeń z Kroku 7 w metodzie m-arnej.

5.1 Redukcja mnożeń w preprocessingu

Nadal dzielmy binarną reprezentację wykładnika na d-bitowe słowa. Wyobraźmy sobie następujący wykładnik $e = (1011\ 0011\ 0111\ 1000)_2$, $k = 16$ i $d = 4$. Potrzebujemy obliczyć M^w $(mod\ n)$ dla $w = 3, 7, 8, 11$. Te wartości mogą być uzyskane następująco:

$$
\begin{aligned}
M^2 &= M \cdot M \\
M^3 &= M^2 \cdot M \\
M^4 &= M^2 \cdot M^2 \\
M^7 &= M^4 \cdot M^3 \\
M^8 &= M^4 \cdot M^4 \\
M^{11} &= M^8 \cdot M^3
\end{aligned}
$$

i potrzeba 6 mnożeń. Metoda m-arna lekceważy to i wykonuje 14 działań w Kroku 1. Liczba działań jaka może dostać zaoszczędzona jest ograniczona przez $m - 2 = 2^d - 2$, na przykład dla $e = (0001\,0001\,0001\,0001)_2$ gdzie maksymalna wartość słowa wynosi 1. Tu nic nie musimy obliczać w Kroku 1, wystarczy tylko M. Zdarza się to jednak całkiem rzadko. W ogólności chcemy obliczyć $M^w \ (mod\ n)$ dla wszyskich $w = w_0, w_1, \ldots, w_{p-1}$. Jeśli do obliczenia w_i dla $i = 0, 1, \ldots, p-1$ są potrzebne wszystkie wartości ze zbioru $\{2, 3, \ldots, 2^d - 1\}$ to nie ma oszczędności. Jednakże gdy wystarczy podzbiór (najlepiej mały) to oszczędność może być osiągnięta jeśli umiemy obliczyć w_i dla $i = 0, 1, \ldots, p-1$ używając mniej niż $2^d - 2$ mnożeń. Algorytm obliczania wartości p wykładników nazywany jest *vectorial addition chain*, a gdy $p = 1$ *addition chain*. Niestety problem uzyskania łańcucha o minimalnej długości jest NP-zupełny. Szerszy opis tych zagadnień znajduje się w dwóch ostatnich rozdziałach.

5.2 Techniki ruchomego okna

Metoda m-arna dokonuje dekompozycji bitów wykładnika na d-bitowe słowa. Prawdopodobieństwo, że słowo długości d jest zerem wynosi 2^{-d}, zakładając, że prawdopodobieństwo wystąpienia 0 i 1 jest równe. W algorytmie m-arnym wykonujemy Krok 7 gdy słowo równa się własnie zero co jest sprawdzane w Kroku 6. W związku z tym gdy d rośnie, to prawdopodobieństwo że wykonamy operacje z Krok 7 także wzrasta. Z drugiej strony gdy d maleje całkowita liczba mnożeń rośnie. Algorytmy opisywane w tym rozdziale idą na kompromis między zerowymi i niezerowymi słowami zmiennej długości. Ta strategia polega na zwiększeniu liczby zerowych słów przy jednocześnie relatywnie dużej wartości d.

Algorytm ruchomego okna najpierw rozbija e na zerowe i niezerowe słowa (okna) F_i o długości $L(F_i)$. Liczba okien p może być równa $\frac{k}{d}$. W ogólności nie jest wymagane żeby długości okien były równe. Weźmy d jako długość najdłuższego okna. Potem gdy F_i jest niezerowym oknem, to najmniej znaczący bit F_i jest równe 1. Dzieje się tak dlatego, że partycjonowanie e rozpoczynamy od najmniej znaczącego bitu i nie ma sensu zaczynać niezerowego okna od zerowego bitu. W konsekwencji liczba mnożeń podczas preprocessingu zmaleje okolo 2 razy, gdyż potrzebne są tylko nieparzyste w.

Algorytm 5 sliding window

1: Oblicz i zapamiętaj $M^w \ (mod\ n)$ dla $w = 3, 5, 7 \ldots, 2^d - 1$.
2: Stwórz zerowe i niezerowe okna F_i o długości $L(F_i)$ dla $i = 0, 1, 2 \ldots, p-1$.
3: $C := M^{F_{p-1}} \ (mod\ n)$
4: **for** $i := p-2$ **downto** 0 **do**
5: $\quad C := C^{2^{L(F_i)}} \ (mod\ n)$
6: $\quad$ **if** $F_i \neq 0$ **then**
7: $\quad\quad C := C \cdot M^{F_i} \ (mod\ n)$
8: $\quad$ **end if**
9: **end for**

Algorytm 5 przedstawia ogólny schemat rozwiązania problemu ruchomego okna. Teraz przedstawię dwie strategie wykonania Kroku 2.

5.3 Niezerowe okna stałej długości

CLNW (Constant Length Nonzero Windows).

Algorytm scanuje bity wykładnika od najmniej znaczącego. Dzieli e na słowa zero o dowolnej długości i niezerowe o długości d.

Przykład Niech $d = 3$ wtedy podział na okna wygląda następująco:

$$e = 3665 = (111\ 00\ 101\ 0\ 001)_2$$

W Kroku 1 algorytm oblicza $M^w\ (mod\ n)$ dla $w = 3, 5, 7$.

bity	w	M^w
001	1	M
010	2	$M \cdot M = M^2$
011	3	$M^2 \cdot M = M^3$
101	5	$M^2 \cdot M^3 = M^5$
111	7	$M^5 \cdot M^2 = M^7$

W Kroku 3 następuje przypisanie $C := M^{F_4} = M^7\ (mod\ n)$, iteracja pętli przebiega następująco:

i	$L(F_i)$	**Krok 5**	F_i	**Krok 7**
3	2	$(M^7)^4 = M^{28}$	00	M^{28}
2	3	$(M^{28})^8 = M^{224}$	101	$M^{224} \cdot M^5 = M^{229}$
1	1	$(M^{229})^2 = M^{458}$	0	M^{458}
0	3	$(M^{458})^8 = M^{3664}$	001	$M^{3664} \cdot M = M^{3665}$

Całkowita liczba mnożeń modularnych wynosi $4 + 9 + 2 = 15$. Tabela przedstawia porównanie metody m-arnej z CLNW. Przy każdej z metod wartość d^* to optymalne d w zależności od k.

k	m-**arna**	d^*	**CLNW**	d^*	**oszczędność %**
128	167	4	156	4	7.14
256	325	4	308	5	5.52
512	635	5	607	5	4.56
1024	1246	5	1195	6	4.17
2048	2439	6	2360	7	3.28

5.4 Niezerowe okna zmiennej długości

VLNW (Variable Length Nonzero Windows).
Algorytm scanuje bity wykładnika od najmniej znaczącego. Dzieli e na słowa zero o długości przynajmniej q i niezerowe o długości co najwyżej d.

Przykład Niech $d = 5$ i $q = 2$ wtedy podział na okna wygląda następująco:

$$e = (101\ 0\ 11101\ 00\ 101\ 10111\ 000000\ 1\ 00\ 111\ 000\ 1011)_2$$

Przykład Niech $d = 10$ i $q = 4$ wtedy podział na okna wygląda następująco:

$$e = (1011011\ 0000\ 11\ 0000\ 11110111\ 00\ 1111110101\ 0000\ 11011)_2$$

Tabela przedstawia porównanie metody m-arnej z VLNW. Przy każdej z metod wartość d^* to optymalne d w zależności od k, a q^* to optymalne q w metodzie VLNW.

k	m-**arna** (T/k)	d^*	**VLNW** (T_2/k)	d^*	**oszczędność % dla** q^*
128	1.31	4	1.20	4	7.82
256	1.27	4	1.18	4	6.77
512	1.24	5	1.16	5	6.37
1024	1.22	5	1.15	6	5.83
2048	1.19	6	1.13	6	4.95

6 Metoda faktoryzacyjna

Metoda ta bazuje na faktoryzacji wykładnika $e = rs$, gdzie r jest najmniejszym pierwszym dzielnikiem e i $s > 1$. Obliczamy M^e licząc najpierw M^r, a potem podnosząc to do potęgi s-tej.

$$C_1 = M^r$$
$$C_2 = C_1^s = M^{rs} = M^e$$

Jeśli e jest liczbą pierwszą to liczymy M^{e-1} i potem mnożymy przez M. Algorytm jest rekurencyjny i liczy się tak długo jak wykładnik jest większy od 2.

Przykład Niech $e = 55 = 5 \cdot 11$.

- Policz: $M \to M^2 \to M^4 \to M^5$
- Podstaw: $a := M^5$
- Policz: $a \to a^2$
- Podstaw: $b := a^2$
- Policz: $b \to b^2 \to b^4 \to b^5$
- Policz: $b^5 \to b^5 a = M^{55}$

Potrzebnych było 8 mnożeń podczas gdy w metodzie binarnej 9.

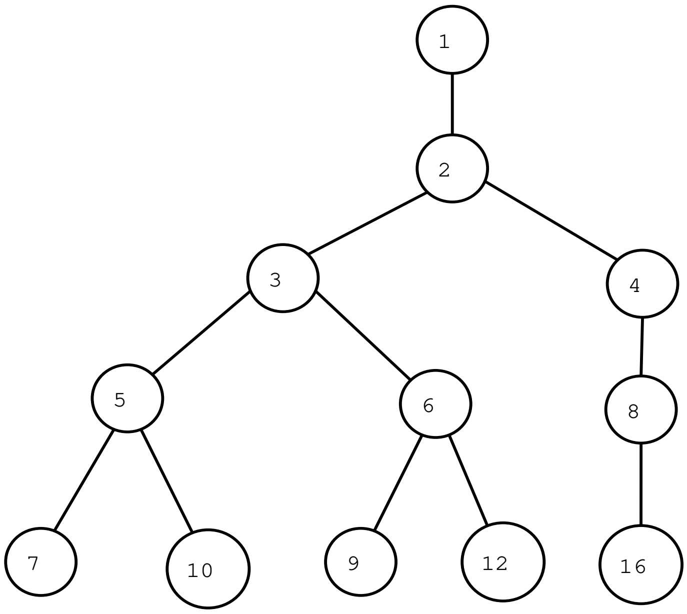
Niestety ta metoda wymaga znajdowania czynnika, co może być bardzo trudne dla dużych liczb. Jednakże może być stosowana w kryptosystemie RSA gdy wykładnik jest mały. Może także być użytaczna gdy wykładnik jest dobierany starannie, na przykład w sposób pozwalający szybko faktoryzować.

7 Metoda drzewa potęgowego

Algorytm opiera się na heurystyce konstrukcji drzewa. Wierzchołki drzewa są etykietowane dodatnimi liczbami całkowitymi. Korzeń otrzymuje 1. Załóżmy, że drzewo jest stworzone do poziomu k-tego. Weźmy lewy liść drzewa i nazwijmy go v, $k+1$ poziom tworzymy doczepiając do v wierzchołki

$$v + a_1, v + a_2, v + a_3, \ldots, v + a_k$$

gdzie $a_1, a_2, a_3, \ldots, a_k$ jest ścieżką od korzenia do v. Tak postępujemy dla wszystkich liści k-tego poziomu, od lewej do prawej. Duplikaty wierzchołków usuwamy.



Gdy chcemy policzyć M^e tworzymy drzewo dla e. Ciąg potrzebny do obliczenia znajduje się na ścieżce od korzenia od e. Dla niektórych wartości e metoda drzewa potęgowego wymaga mniejszej liczby mnożeń. Na przykład dla M^{23} potrzeba 6 mnożeń:

$$M \to M^2 \to M^3 \to M^5 \to M^{10} \to M^{13} \to M^{23}$$

podczas gdy przy metodzie binarnej $4 + 3 = 7$

$$M \to M^2 \to M^4 \to M^5 \to M^{10} \to M^{11} \to M^{22} \to M^{23},$$

a metodzie faktoryzacyjnej także $1 + 5 + 1 = 7$

$$M \to M^2 \to M^4 \to M^8 \to M^{16} \to M^{20} \to M^{22} \to M^{23}.$$

Wadą tej metody jest stosowalność dla małych wykładników, ponieważ drzewo musi być umieszczone w pamięci.

8 Addition Chain

W skrócie będziemy nazywać łańcuchem. Wyobraźmy sobie ciąg liczb całkowitych

$$a_0, a_1, a_2, \dots, a_r$$

gdzie $a_0 = 1$ i $a_r = e$. Ciąg jest konstruowany w następujący sposób: dla każdego k istnieją $i, j < k$, że $a_k = a_i + a_j$. Długością ciągu jest r. Tak skonstruwany ciąg dla zadanego e jest rozwiązaniem problemu potęgowania modularnego. Zaczynamy od M^1, a potem obliczamy M^{a_k} korzystając z wcześniej policzonych M^{a_i} i M^{a_j}.

m. binarna:	1	2	3	6	12	13	26	27	54	55	
m. ósemkowa:	1	2	3	4	5	6	7	12	24	48	55
m. faktoryzacyjna:	1	2	4	5	10	20	40	50	55		
m. drzewa potęgowego:	1	2	3	5	10	11	22	44	55		

Jak już wspomniałem wcześniej znajdowanie najkrótszego łańcucha jest NP-zupełne. Jednakże od 1937 kiedy to Scholz postawił ten problem znaleziono kilka własności:

- Górna granica na długości minimalnego łańcucha jest równa $\lfloor \log_2 e \rfloor + H(e) - 1$, gdzie $H(e)$ jest wagą Hamminga (liczba jedynek w zapisie binarnym) e. Ten wynik uzyskiwany jest metoda binarną.
- Dolna granica podana przez Schönhage wynosi $\log_2 e + \log_2 H(e) - 2.13$.

Wysztkie podane wcześniej algorytmy są heurystykami dla problemu minimalnego łańcucha. Heurystykami dlatego, że nie gwarantują minimalności.

9 Vectorial Addition Chain

W skrócie będziemy nazywać wektorowym łańcuchem. Jest to podobny problem do opisywanego w poprzednim rozdziale. Różni się tym, że zamiast liczb mamy wektory długości (wymiaru) i. A oto własności:

- inicjalne wekotry to wektory jednostkowe $[1, 0, \ldots, 0], [0, 1, 0 \ldots, 0], \ldots, [0, \ldots, 0, 1]$
- każdy wektor jest sumą dwóch poprzednich
- ostatni wektor jest równy zadanemu wektorowi

Przykład Chcemy uzyskać wektor $[7, 15, 23]$, otrzymujemy wtedy wektorowy łańcuch:

$$[0,1,1]\ [1,1,1]\ [0,1,2]\ [1,2,3]\ [1,3,5]\ [2,4,6]\ [3,7,11]\ [4,8,12]\ [7,15,23]$$

długości 9. Krótki łańcuch wektorowy może być użyty do efektywnego obliczenia M^{w_i} dla pewnych liczba całkowitych. Ten problem pojawia się w fazie preprocessingu w metodzie m-arnej.

Addition sequence (sekwencja) jest to łańcuch gdzie chcemy aby i podanych liczb pojawiło się gdzieś w łańcuchu. Olivos udowodnił, że obliczenie wektorowego łańcucha to to samo co obliczenie sekwencji. Wynik z przykładu $7, 15, 23$ można uzyskać za pomocą łańcucha:

$$1\ 2\ 3\ 4\ 7\ 8\ 15\ 23$$

o długości 7. Zostało udowodnione, że sekwencja o długości r i i żądanych liczbach może być przekonwertowana na wektorowy łańcuch o długości $r + i - 1$ i wymiarze i.