# 모듈러 멱승을 계산하는 일반화된 모델1)

김지은O 김동규
부산대학교 컴퓨터공학과
{jekimO, dkkim}@islab.ce.pusan.ac.kr

# Generalized Models for Computing Modular Exponentiation

Jieun KimO Dong Kyue Kim
Dept. of Computer Engineering
Pusan National University, Busan 609-735, Korea

**요 약**

모듈러 멱승은 주어진 값 X, E, N에 대하여 $X^E$ mod N으로 정의된다. 모듈러 멱승은 대부분의 공개키 암호시스템과 전자서명에 사용되므로, 이 연산을 빠르게 수행하는 문제는 암호학 분야에서 중요하게 연구되고 있다. 본 논문에서는 모듈러 멱승을 효율적으로 계산하기 위하여, 멱승 계산을 위한 일반화된 그래프 모델을 제시하였다. 이 모델은 기존의 방법들을 대부분 포용할 수 있으며, 특히 새로운 방법을 개발하는데 유용할 것이다. 이 모델의 장점을 정당화하기 위하여 기존 알고리즘 중 가장 성능이 좋은 VLNW(Variable Length Nonzero Window)방법과 실험을 통하여 비교하였으며, 확장성이 높음을 확인하였다.

## 1. 서 론

멱승은 양수 E에 대하여 $X^E$을 계산하는 것이고, 모듈러 멱승은 양수 E와 양수 N에 대하여 $X^E$ mod N을 계산하는 것이다. 모듈러 멱승을 빠르게 계산하는 것은 RSA와 같은 공개키 암호시스템이나 전자 서명 등의 암호 학에서 중요한 문제로 다루어 왔다. 모듈러 멱승을 빠르게 계산하는 문제는 지금까지 두 가지 방향으로 연구가 되어 왔는데, 한 가지는 모듈러 멱승에서 필요한 곱셈의 수를 줄이는 것이고, 나머지 한 가지는 곱셈을 한 번 하는데 걸리는 시간을 줄이는 것이다. 본 논문은 전자의 방향으로 이 문제에 접근한다.

현재까지 전자의 방향으로 많은 알고리즘이 연구되었다. 그 중 대표적인 알고리즘 몇 가지를 소개하면 다음과 같다. 먼저, 가장 간단한 알고리즘은 이진(binary) 방법이며, 이를 일반화한 것이 M-ary 방법이다. 가장 많이 사용되는 방법은 슬라이딩 윈도우 방법인데, 이 방법은 고정된 크기의 윈도우를 사용하는 CLNW (Constant Length NonZero Window)방법과 가변적인 크기의 윈도우를 사용하는 VLNW (Variable Length NonZero Window)방법이 있다. 그리고 덧셈 체인 방법, 인수 방법, 트리 방법, 및 레코딩 방법 등이 제시되었다.

위에서 열거한 것들 중 가장 성능이 좋은 것이 슬라이딩 윈도우 방법이다. 슬라이딩 윈도우 방법을 살펴보면 두 가지의 방향에서 모듈러 멱승 문제에 접근하고 있음을 알 수 있다. 첫 번째는 적절한 초기 블록 테이블을 만드는 방향이고, 두 번째는 초기 블록 테이블을 이용하여 윈도우를 만들어 가는 분할(partitioning) 문제이다. 이 방법들은 초기 블록 테이블을 결정할 때 실험을 통하여 결정하였으며, 윈도우를 분할하는 방법은 그리디(greedy)한 알고리즘으로 해결하고 있다.

본 논문에서는 모듈러 멱승 문제를 풀 수 있는 일반화된 계산 모델을 그래프를 이용하여 제시한다. 즉, 모듈러 멱승 문제에 입력되는 데이터들을 그래프의 정점(vertex)과 가중값(weight)을 가지는 간선(edge)으로 표현하였다. 제시된 모델은 앞에서 언급된 이진 방법이나 슬라이딩 윈도우와 같은 방법들을 수용할 수 있으며, 새로운 방법을 개발하기에 유용하다.

본 논문의 구성은 다음과 같다. 2장에서 기존 알고리즘을 분석하고, 3장에서 일반화된 모델을 제시한다. 그리고 4장에서 기존의 VLNW와 최단 경로 알고리즘을 적용한 그래프 모델의 실험적인 결과를 분석하고, 5장에서 결론을 맺는다.

## 2. 기존 알고리즘 분석

여기서는 모듈러 멱승을 계산하기 위하여 곱셈의 수를 줄이는 몇 가지 알고리즘을 기술한다. 먼저, 가장 간단한 알고리즘이 이진 방법이다. 이 방법은 이진수로 표현된 지수 E를 왼쪽부터 한 비트씩 읽어가면서 곱셈과

1) 본 연구는 한국과학재단 목적기초연구 (R01-2002-000-00589-0)지원으로 수행되었음

제곱을 수행한다. 이진 방법은 평균적으로 1.5(logE-1)의 곱셈을 수행하는 것으로 분석할 수 있다.

이진 방법을 일반화한 것이 M-ary 방법이다. M-ary 방법은 M개의 초기 블록을 미리 계산한다. 초기 블록의 길이는 $\log_2 M$이 된다. 이 방법은 이진수로 표현된 지수 E를 $\log_2 M$씩 읽어가면서, 초기 블록에 계산한 값들 중 방금 읽은 $\log_2 M$에 해당되는 값을 곱해주면서 진행된다. 즉, $\log_2 M$비트를 한번 읽을 때 한번의 곱셈이 필요하게 된다.

M-ary를 효율적으로 수정한 방법이 슬라이딩 윈도우(sliding window) 방법이다. 슬라이딩 윈도우 방법은 M-ary방법과 비교하면 다음과 같은 두 부분에서 차이를 보인다. 첫 번째는 초기 블록 테이블이다. 초기 블록 테이블에는 지수 승을 이진수로 표현 했을 때 '1'로 시작하여 '1'로 끝나는 블록들만 계산하여 저장한다. 실제로 저장되는 블록의 수는 M-ary에 비하여 반정도로 줄어든다. 두 번째는 $X^E \bmod N$을 계산하는 알고리즘 부분이다. 이 방법은 지수 E를 읽어 가면서 연속된 '0'으로만 으로 구성된 ZW(Zero Window)와 '1'이 포함된 NW(Nonzero Window)를 구분한다. ZW와 NW를 만들어 가는 과정을 하나의 분할(partitioning) 과정이라고 말할 수 있다. 슬라이딩 윈도우 방법은 지수 E를 어떤 방식으로 분할하는가에 따라 두가지 방식이 있다. 먼저, 첫 번째 방법은 CLNW(Constant Length Nonzero Window)라고 하는데, 이 방법은 ZW는 가변적인 길이를 가지도록 하고, NW의 길이는 일정하게 고정시키는 방법이다. 지수 E를 한쪽 방향으로 읽어가면서 다음과 같은 알고리즘을 적용한다.

- **ZW(Zero Window)** : 한 비트를 확인한다. '0'이면 ZW에 머물러 있고, '1'이면 NW를 시작한다.
- **NW(Nonzero Window)** : d 비트를 읽는 동안 NW에 머물러 있는다. 다음 한 비트를 읽어 '0'이면 ZW를 시작하고, '1'이면 NW를 시작한다.

예) d를 3으로 하고, E를 왼쪽에서 오른쪽 방향으로 한 비트씩 읽어간다.

E = 111 00 101 0 001

CLNW방법을 적용하게 되면 ZW가 생김으로 해서 가 M-ary방법 보다 줄어들게 된다. 곧 곱셈의 수가 줄어들게 되는 것이다.

두 번째 방법은 VLNW (Variable Length Nonzero Window)방법이다. 이 방법은 ZW와 NW의 길이를 모두 가변적이도록 하는 방법이다. 이 알고리즘을 기술하기 위해서는 몇 가지 인자(parameter)가 필요하다.

- **d** : NW의 최대 길이
- **q** : 현재의 NW를 끝내는 최소한의 영의 수
- **k**와 **r** : $d = 1 + kq + r$ ( $1 \leq r < q$)를 만족하는 정수

이 방법의 분할 알고리즘은 다음과 같다. 지수 E를 읽어 가면서 '0'이 나오면 ZW를 만든다. '1'이 나오면 NW를 시작하고 q씩 읽어간다. 읽은 것이 모두 영이면 ZW를 다시 시작한다. 그렇지 않으면 NW에 그대로 머물러 있게 된다. 윈도우의 길이가 d를 넘으면 다음 윈도우를 시작한다.

예) d를 5로 q를 3인 경우, 오른쪽에서 왼쪽 방향으로 읽어가면서 분할한 결과는 다음과 같다.

E = 101 11101 10011 0000 11 00 111 00 100111

위의 방법들을 이용하여 모듈러 멱승을 계산할 때 필요한 연산의 수를 셀 때는 다음 세 가지를 고려한다.

① 초기 블록들을 계산 하는 데에 필요한 연산 수
② NW의 개수에서 1을 뺀 수
③ 전체 길이에서 첫 번째 윈도우의 길이를 뺀 수

①과 ②은 필요한 곱셈의 수이고, ③은 제곱을 하는 수이다. ①, ②, ③을 모두 합하면 총 곱셈의 수가 된다.

위의 네 가지 방법을 비교해 보면 제곱하는 횟수는 비슷하다. 결국 곱셈의 수를 줄이는 것은 초기 블록을 계산하는 데 필요한 곱셈의 수와 분할 결과 생긴 NW의 개수에 영향을 받음을 알 수 있다.

이진 방법은 초기 블록을 계산하지 않으므로 NW의 수만 생각 하면 된다. 이진 방법이 만드는 윈도우는 '0'과 '1'뿐이므로 NW의 개수는 평균적으로 (E의길이)/2개 정도 될 것이다.

M-ary 방법은 M개의 초기 블록을 계산하여 주어야 하며, (E의길이)/$\log_2 M$개의 윈도우가 만들어진다. ZW도 포함되어 있겠지만 그 개수는 무시할 만하다. 하나의 윈도우는 여러 개의 '1'을 포함할 수 있으므로 이진 방법보다 윈도우의 수가 줄어들게 된다.

슬라이딩 윈도우 방법은 ZW와 NW로 나누기 때문에 초기 블록을 계산 할 때 NW를 지수로 가지는 블록만 계산하면 된다. 그래서 초기 블록을 계산하는 데에는 대략 M/2번의 연산이 필요하며, M-ary방법과 비교하면 연산의 수가 적다. 지수 E를 윈도우로 나누어 갈 때, ZW와 NW으로 구분하여 윈도우를 만들어 가기 때문에 NW의 수가 M-ary보다 더 줄어들게 된다.

## 3. 그래프 모델링

본 절에서는 입력되는 데이터에 대하여 그래프로 표현한 일반화된 모델을 제시한다. 그래프는 정점의 집합과 가중 값(weight)을 가진 간선의 집합으로 이루어져 있다. 즉, 주어진 지수 E의 각 비트들을 정점으로 정의하고 만들어 질 수 있는 윈도우를 간선으로 정의한다.

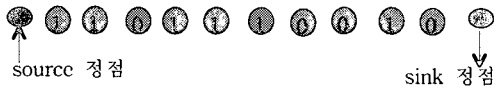
### 3.1 그래프 모델의 정의

그래프의 정점의 집합을 'V'로 나타내고 간선의 집합을 'E'로 나타낸다. 그리고 가중 값을 'W'로 나타낸다. 그리고 초기 블록의 지수 승의 최대 길이를 d로 한다. 각 정점은 비트 순으로 인덱스를 가진다. |정점 v의 인덱스 - 정점 u의 인덱스|는 정점 u와 정점 v사이의 거리를 나

타낸다.

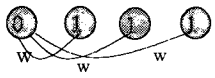
- **V** = { v | 지수 E를 이진수로 표현 했을 때 나타나는 모든 비트, source 정점, sink 정점 }
- **E** = { e=(v,u) | |정점 v의 인덱스 - 정점 u의 인덱스| ≤ d, for all v∈V, u∈V }
- **W** - {w(c) | c가 표현하는 윈도우를 계산하는데 필요한 곱셈의 수, for all c∈E }

정의에 따르면 E의 각 비트들은 하나의 정점으로 표현된다. 그리고 두개의 정점이 추가된다. E의 양끝에 source 정점과 sink 정점를 만들어 준다. source 정점는 이 그래프에서 만들어지는 경로의 출발 정점이고, sink 정점은 그 경로의 도착 정점이다. 정점들은 [그림 1]과같이 표현된다.




[그림 1] 지수 "1101110010"에 대한 정점 집합

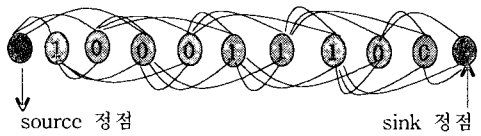
초기 블록으로 만들어지는 윈도우는 간선으로 표현된다. 가중 값은 간선이 표현하는 윈도우의 곱셈 수이다. 간선의 시작 정점은 윈도우의 시작 비트이고 간선의 도착 정점은 다음 윈도우의 시작 정점이 된다. 하나의 정점에서 만들 수 있는 간선은 [그림 2]와 같다.




[그림 2] d=3일 경우, 한 정점에서 정의되는 간선의 종류

[그림 2]의 첫 번째 정점에서 두 번째 정점을 잇는 간선은 '0'이라는 윈도우를 나타낸다. ZW이기 때문에 가중 값은 '0'이다. 첫 번째 정점 네 번째 정점을 잇는 간선은 '011'이라는 윈도우를 나타낸다.

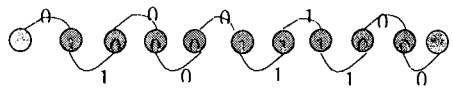
[그림 3]은 지수가 "100011100"이고, d=3인 경우, 정의된 그래프의 모든 정점과 간선들을 보여주고 있다. 간선들의 가중 값은 생략하였으며, 해당 윈도우를 계산하는데 필요한 곱셈의 수로 정의된다.



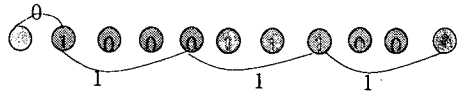

[그림 3 ] d=3인 경우, 정점과 간선들의 종류

### 3.2 그래프 모델링

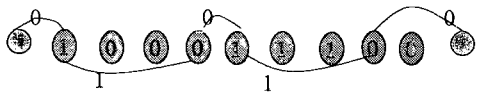
본 절에서는 제시하고 있는 그래프 모델이 기존의 방법들을 수용하는 일반화된 계산 모델임을 보인다. 즉 앞 절에서 설명한 이진 방법, M-ary방법, CLNW 방법, VLNW 방법에서 수행하고 있는 각각의 곱셈의 순서는 제시된 그래프 모델에서 source 정점에서 sink 정점으로 가는 하나의 경로(path)에 대응된다. 그리고 모델링된 그래프에는 각 방법에 따른 경로가 표현되어 있다.



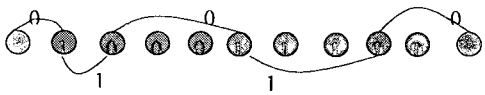
[그림 4] 이진 방법의 적용 예



[그림 5] M-arv 방법의 적용 예



[그림 6] CLNW 방법의 적용 예

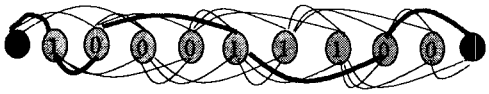


[그림 7 ] VLNW 방법의 적용 예

[그림 4]는 이진 방법이 적용된 예를 보이고 있다. 비트가 1일 때마다 한번의 곱셈을 나타내고 있고, 이 경로의 가중 값의 총합은 1의 개수와 동일하다. [그림 5]는 M-ary방법이 적용된 예를 보여준다. ZW와 NW의 구분 없이 d=3 비트씩 윈도우가 만들어지고 있으며, 이 것이 하나의 경로로 표현된다. 이때, 가중 값의 총합은 3이고 곱셈의 수는 3번이다. [그림 6]은 CLNW 방법의 적용예를 보이고 있다. 지수 E를 왼쪽에서 오른쪽으로 읽어가는 방식일 때, 1인 비트에서 길이가 3인 NW들이 만들어졌고, 가변적인 길이의 ZW들이 만들어 졌다. 이 경로의 가중 값의 총합은 2가 되며, 이것은 2번의 곱셈을 필요로 함을 의미한다. 마지막으로 [그림 7]은 VLNW 방법의 적용예를 보이고 있다. 왼쪽에서 오른쪽으로 읽어 가는 방식일 때, 가변적인 NW와 가변적인 ZW가 나타나며 이것은 하나의 경로로 표현할 수 있다. 경로의 가중 값의 총합은 2이다.

### 3.3 그래프의 최단경로 구하는 문제에 적용

모델링된 그래프에는 $d^L$ 개의 경로가 존재하는데, 하나의 경로를 하나의 분할을 의미한다. 간선에 있는 가중 값을 수행되는 곱셈의 수로 정의하고 있으므로, 그래프에서 source 정점에서 sink 정점까지의 경로중 최단 경로를 찾는 다는 것은 최소의 곱셈 수를 가지는 분할을 찾는 것과 동등하다. [그림 8]은 [그림 3]의 예에서 최단경로를 찾는 알고리즘을 적용한 예를 보여 주고 있다.



[그림 8] 그래프의 최단 경로로 적용한 예

본 논문에서 모델링된 그래프는 왼쪽에서 오른쪽으로, 또는 오른쪽에서 왼쪽으로 적용하더라도 반대 방향으로 가는 간선은 만들어지지 않기 때문에 DAG(directed acyclic graphs)의 구조를 가진다. DAG에서 최단 경로를 구하는 알고리즘은 선형시간이 소요되므로, 제시된 방법의 전체 시간복잡도는 O(d x 비트수)가 된다.

## 4. 실험 결과

본 논문에서는 그래프의 최단 경로 구하는 알고리즘을 적용한 것과 VLNW를 실험적으로 분석하였다. d의 값이 변화하면서, 제시된 그래프 모델에서 최단 경로 구하는 알고리즘을 적용한 것과 VLNW 방식을 비교하였다. VLNW 방식은 q값을 미리 설정하여야 하므로, 가장 좋은 성능을 보이는 q값의 결과와 비교하였다. 다음 표는 실험결과를 보여주고 있으며, 제곱의 수와 초기 블락의 수는 두 방법이 비슷하기 때문에 분할 후 생긴 NW의 수만 비교하였다. 실험 결과에서 보듯이 모든 d에 대하여 최단 경로를 적용한 방법이 VLNW 방법보다 미비하지만 성능이 향상되고 있다. 더 의미있는 것은 VLNW 방식은 사전에 q값을 미리 정해야하는 번거로움이 있지만, 제안한 방법에서는 q값에 무관하게 동작한다.

(a) 512 bits

| d | 4 | 5 | 6 | 7 |
|---|---|---|---|---|
| VLNW | 107 (q=3) | 88 (q=3) | 74 (q=4) | 64 (q=5) |
| Graph Model | 104 | 85 | 72 | 63 |

(b) 1024 bits

| d | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|
| VLNW | 207 (q=3) | 174 (q=3) | 147 (q=4) | 129 (q=5) | 115 (q=6) | 103 (q=7) | 93 (q=7) |
| Graph Model | 204 | 169 | 145 | 127 | 113 | 101 | 92 |

(c) 2048 bits

| d | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|
| VLNW | 410 (q=3) | 341 (q=3) | 293 (q=5) | 257 (q=5) | 227 (q=6) | 205 (q=7) | 187 (q=7) |
| Graph Model | 408 | 340 | 291 | 255 | 226 | 204 | 185 |

## 5. 결론

모듈러 멱승을 구하는 것은 보안 암호 학에서 끊임없이 연구 되어야하는 중요한 문제이다. 암호화하는 데이터의 길이가 계속 길어지고 있는 실정이므로, 모듈러 멱승을 빠르게 계산하는 방법의 개발이 시급하다. 모듈러 멱승 알고리즘을 지속적으로 발전시키기 위해서는 한번의 분할로 결정되어 버리는 기존의 방식을 개선할 수 있는 일반화된 모델이 필요하다. 본 논문에서는 이를 수용할 수 있는 일반화된 계산 모델을 제시하였다.

이 모델을 사용하면 VLNW 방법에서 q의 값을 설정해 줄 필요가 없어지므로, 데이터의 길이가 길어지더라도 적용하기 쉬워진다. 뿐만 아니라, 최단 경로 알고리즘을 사용하여 곱셈의 수가 가장 작은 경로를 찾아 주기 때문에 기존 알고리즘 보다 더 좋은 성능을 보임을 실험을 통하여 비교하였다.

## [참고문헌]


[1] D.E Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Addison-Wesley, U.S.A(1981)

[2] R.L Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public key cryptosystems, *Communications of the ACM* 21(2) 120-126 (1978)

[3] T. Elgamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory* 31(4)469-472(1985)

[4] National Institute for Standards and Technology, Digital Signature Standard(DSS), *Federal Register* 56 169(1991)

[5] J. Bos and M. Coster, Addition chain heuristics, In *Proc. Crypto'89*, Lecture Notes in Computer Science Vol. 435, pp.400 407, Springer-Verlag,(1990)

[6] P.Downey, B. Leong and R.Sethi, Computing sequences with addition chains, *SIAM Journal on Computing* 10(3) 638-646 (1981)

[7] C. K. Koc, "High-radix and bit recoding techniques for modular exponentiation", *International Journal of Computer Mathematics*, 40(3+4) pp.139-156 (1991)

[8] C.K. Koc, Analysis of sliding window techniques for exponentiation, *Computers & Mathematics with Applications* 30(10)17-247(1995)

[9] S.Hong, S. Oh, and H. Yoon, New modular multiplication algorithms for fast modular exponentiation, In *Proc. Eurocrypt'96* Lecture Notes in Computer Science Vol. 1070, pp.166-177, Springer-Verlag,(1996)

[10] P.L.Montgomery, Modular multiplication without trial division, *Mathematics of Computation* 44(170) 519 521(1985)

[11] H. Morita, A fast modular-multiplication algorithm based on a higher radix, In *Proc. Crypto'89*, Lecture Notes in Computer Science Vol. 435, pp.387-399, Springer-Verlag,(1990)

[12] C.D.Walter, Faster modular multiplication by operand scaling. In *Proc. Crypto'91*, Lecture Notes in Computer Science Vol.596,pp.313-323, Springer-Verlag,(1992)

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Introduction to Algorithms* pp.514-531(1990)