한국학교수학회논문집 제 6 권, 제 2 호
Journal of the Korean School Mathematics Society
Volume 6, Number 2, 87-99, December 2003

학교수학에서 이산수학 교수 방안 연구

한 근 희[1)]

본 논문에서는 제 7차 교과과정에 의하여 새로운 선택과목으로 선정된 이산수학의 효율적인 교수방안을 논의한다. 이를 위하여 이산수학의 가장 핵심적인 내용인 알고리즘의 개발 필요성 및 목적 등을 강조하기 위하여 행렬 곱셈에 관한 전통적인 방법의 문제점을 분석하며 또한 효율적인 행렬 곱셈 알고리즘을 분석한다.

주요용어 : 이산수학, 알고리즘, 행렬 곱셈

I. 서 론

최근에 정립된 제7차 교과과정 (교육부) 에는 고등학생 들을 대상으로 이산수학 (Discrete Mathematics)이 10 단계의 수학에 도달 여부와 관계없이 학생들이 선택할 수 있는 심화선택 과목 중의 하나로 선택이 되었다. 7 차 교육과정에 따르면 이산수학 교과과정의 목표는 수학의 기본적인 지식과 기능을 활용하여 실생활의 이산적인 상황의 문제를 수학적으로 사고하는 능력을 기르고, 합리적으로 의사를 결정하며, 창의적으로 문제를 해결 할 수 있도록 하는 것에 있으며 세부 목표는 다음과 같이 정의되어 있다.

가) 일상적인 정보에서 수량적인 관계나 법칙을 계산기나 컴퓨터를 이용하여 이해하고, 이를 활용 할 수 있도록 한다.
나) 세기의 기본이 되는 방법과 집합이나 자연수를 나누는 방법을 이해하고, 이를 이용하여 실생활에서 여러 가지 경우의 수를 구할 수 있도록 한다.
다) 사물의 현상을 그래프와 행렬 등을 이용하여 조직 및 해석하고, 이를 활용할 수 있도록 한다.
라) 여러 가지 문제를 알고리즘 적으로 사고하고 처리하는 능력을 배양시킨다.
마) 다양한 의사 결정 과정과 상충적인 상황에서 합리적이고 논리적인 사고를 하여 문제를 해결할 수 있도록 한다.

7차 교과과정에서는 이러한 교육 목표를 달성하기 위하여 이산수학의 세부 내용을 다음 표 1.1과 같이 정립하고 있다. 표 1.1을 분석하면 이산수학의 교과 영역 중 선택과 배열은 고등학교에서 이미 다루었던 것이므로 핵심적인 영역은 그래프 및 알고리즘 영역이 라는 것을 알 수 있으며 또한 이들 영역들로 인하여 학생들에게 이산수학이 다소 생소한 분야의 과목이라는 느낌을 받도록 한다. 그러나 알고리즘 및 그래프 영역들은 그 동안 하나의 독립적

1) 공주대학교 응용수학과 (kehan@kongju.ac.kr)

인 과목으로 취급을 하지 않았을 뿐 사실은 전혀 생소한 분야는 아니다. 이는 알고리즘의 목적이 실생활에서 발생되는 다양한 문제들에 대한 효율적인 해결 방안을 제시하는 것으로서 우리가 일상적으로 추구하는 목적과 부합하기 때문이다.

알고리즘을 단순하게 정의한다면 문제를 해결 하는 방안을 제시하는 것이다. 다만 여기서 알고리즘의 대상이 되는 문제들은 학생들이 수학 교과 과정에서 접하던 미분방정식 등과 같은 정형화된 문제들보다는 실생활에서 발생되는 다양한 형태의 문제라는 것이 다른 점이다. 알고리즘을 더욱 정확하게 표현하면 알고리즘은 주어진 문제를 풀기 위한 유한한 계산 단계의 집합이며 다음과 같은 특성을 지녀야한다 (Johnsonbaugh, 1990).

표 1.1. 7차 교육과정의 이산수학 교과 내용

영 역	내 용	
선택과 배열	순열과 조합	• 순열 • 조합
	세기의 방법	• 배열의 존재성 • 포함배제의 원리 • 집합의 분할 • 수의 분할 • 여러 가지 분배의 수
그래프	그래프	• 그래프의 뜻 • 여러 가지 그래프
	수형도	• 여러 가지 수형도 • 생성 수형도
	여러 가지 회로	• 오일러 회로 • 해밀턴 회로
	그래프의 활용	• 행렬의 뜻 • 그래프와 행렬 • 색칠 문제
알고리즘	수와 알고리즘	• 수와 규칙성 • 수와 알고리즘
	점화 관계	• 두 항 사이의 관계식 • 세 항 사이의 관계식
의사 결정의 최적화	의사 결정 과정	• 2 × 2 게임 • 선거와 정당성
	최적화 알고리즘	• 계획 세우기 • 그래프와 최적화

- 정확성 (Precision) : 각 단계들은 명확하게 기술되어야 한다.
- 유일성 (Uniqueness) : 각 단계가 실행된 후의 결과는 입력과 바로 전 단계의 결과에 따라 유일하게 결정되어야 한다.
- 유한성 (Finiteness) : 알고리즘은 제한된 수의 단계가 수행된 후에 종료되어야 한다.

- 입력 (Input) : 알고리즘은 입력을 받는다.
- 출력 (Output) : 알고리즘은 출력을 생성한다.
- 일반성 (Generality) : 알고리즘은 모든 입력집합에 적용될 수 있어야 한다.

이러한 알고리즘의 특성은 결코 주입식 교육으로는 소기의 목적을 달성할 수 없으며 또한 기계적인 해결 방안이 존재하는 것도 아니기 때문에 학생들에게 창의적인 문제 해결 능력을 배양하는 데에도 알고리즘 학습은 매우 효과적이라고 판단된다. 또한 알고리즘 개발 과정에는 반드시 수학적 증명 과정이 가장 핵심적인 역할을 하므로 학생들에게 지금가지 배워온 수학적 사고 능력 및 기술들을 실제로 적용해 나가는 과정을 통하여 수학분야의 학습에 대한 동기를 유발시킬 것이라 판단된다.

상기한 알고리즘의 특성들의 자세한 분석을 위하여 3개의 정수들 중 가장 큰 정수를 찾는 다음 알고리즘을 분석해보자.

알고리즘 1.1. 3개의 정수들 중 가장 큰 수 찾기.
입력 : 3 개의 정수 a, b 및 c.
출력 : $x = \mathrm{Max}\{a, b, c\}$
단계 1: $x := a$
단계 2: If $b > x$, then $x := b$
단계 3: If $c > x$, then $x := c$

알고리즘 1.1 은 3개의 수 a, b, c 들을 차례대로 조사하여 이들 중 가장 큰 값을 x 에 저장하는 것으로서 본 알고리즘의 결과로서 x 는 3 개의 수들 중 가장 큰 수와 동일한 값을 가지게 된다. 알고리즘에서 $x := y$ 는 y 의 값을 x 에 복사하라는 것으로서 y 의 값은 변화하지 않는다. 기호 := 는 대입연산자(assignment operator) 라고 한다. 알고리즘의 정확성은 알고리즘이 궁극적으로는 컴퓨터 프로그램으로 구현되어야 하기 때문에 명확히 기술되어야 한다. 이는 알고리즘을 기술하는 정형화된 언어가 정의되지 않았기 때문이며 또한, 알고리즘 개발자와 프로그래머는 서로 다른 사람 일 수 있기 때문에 알고리즘의 정확성은 매우 중요하다. 알고리즘에 입력 값이 주어지면 알고리즘의 각 단계에서는 유일한 결과가 산출되어야 한다. 예를 들어, $a = 1$, $b = 5$, $c = 3$ 이 주어지면 알고리즘 1.1은 어떠한 사람이나 컴퓨터가 알고리즘을 수행하더라도 결과는 $x = 5$ 가 된다. 유일성이 필요한 이유는 현존하는 컴퓨터들의 특성 때문으로서 컴퓨터는 어떠한 가정을 할 수 없기 때문이다. 알고리즘은 주어진 문제를 해결하는 유한한 개수의 단계들을 수행한 후에 종료되어야 한다. 예를 들어, 알고리즘 1.1 은 3 단계를 수행한 후에 종료되며 주어진 3개의 숫자들 중 가장 큰 값을 출력한다. 이러한 유한성을 간단히 표현하면 어떠한 알고리즘도 주어진 시간 내에서 계산을 완료하여야 한다는 것이다. 알고리즘은 일반적이어야 한다. 예를 들어, 알고리즘 1.1은 어떤 3개의 숫자들 중에서도 가장 큰 값을 찾아낼 수 있다.

고등학교 교과과정에서 알고리즘을 강의하는데 따르는 어려움 중의 하나는 알고리즘이라는 단어 자체가 주는 생소함을 극복하는 것이라고 판단된다. 따라서 가능한 한 학생들이 이미 친숙하다고 느끼는 문제들 중에서 그 문제의 전통적인 해결방법이 포함하고 있는 계산 한계성을 분석하고 이러 한 한계성을 극복하기 위하여 더욱 효율적인 계산 방법을 제시하는 과정이 필요하다고 판단된다. 이러한 과정을 통하여 학생들에게 알고리즘을 배우고자 하는

동기를 유발하는 것이 새로운 분야의 강의 과정에서 반드시 필요하다고 판단된다. 행렬 곱셈은 학생들이 매우 능숙하게 수행할 수 있는 단순한 수학적 기술이다. 그러나 행렬들의 크기가 증가함에 따라 전통적인 방법의 행렬 곱셈 방 법(행렬 곱셈 알고리즘)으로는 계산이 불가능하거나 또는 무의미할 수 있다. 따라서 다음절에서는 전통적인 행렬 곱셈 문제를 통하여 효율적인 알고리즘의 필요성을 분석하고 이어서 더욱 효율적인 행렬 곱셈 알고리즘을 소개 및 분석한다.

Ⅱ. 효율적인 알고리즘의 필요성

A, B 를 다음과 같이 3×3 크기의 정방 행렬이라 하고 우리에게 익숙한 방법으로 2 개 행렬의 곱 $C = A \times B$ 를 계산하는 과정은 그림 2.1과 같다.

$$A = \begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix}, \quad B = \begin{bmatrix} b11 & b12 & b13 \\ b21 & b22 & b23 \\ b31 & b32 & b33 \end{bmatrix}, \quad C = A \times B = \begin{bmatrix} c11 = a11 \times b11 + a12 \times b21 + a13 \times b31 & - & - \\ - & - & - \\ - & - & - \end{bmatrix}$$

그림 2.1. $C = A \times B$.

이때 $c11$을 계산하는 과정에는 모두 3번의 곱셈 이 필요하며 C 에는 모두 $3^2 = 9$ 개의 요소가 있으므로 C 를 계산하기 위해서는 모두 $3 \times 3^2 = 3^3 = 9$ 번의 곱셈이 요구된다. 따라서 일반적으로 $n \times n$ 크기의 정방 행렬 2 개를 곱하는 데에는 모두 n^3번의 곱셈이 요구된다. 만일 $n = 100{,}000$ 이라 면 모두 10^{14}번의 곱셈이 요구된다. 1초에 1,000,000 번의 곱셈을 수행할 수 있는 컴퓨터로 이러한 계산을 한다면 계산을 완료하는데 모두 10^9 초가 소요되며 10^9 초는 대략 32년과 동일한 시간이다. 따라서 전통적인 방법으로 100000×100000 과 같이 크기가 매우 큰 행렬들을 곱하는 것은 계산이 완료되는 필요한 시간이 32 년이므로 이러한 계산은 현실적으로 불가능하거나 또는 무의미하다고 할 수 있다. 그렇다면 이러한 크기의 행렬 곱셈이 실제로 필요한가에 대한 의문이 있을 수 있으나 이러한 종류의 계산은 원유 탐사, 기후 변화 예측 및 공간에서 물체 위치 추적 등에서 매우 빈번히 요구되는 계산 과정이다. 특히 여기서 우리는 요구되는 덧셈 및 뺄셈의 양은 고려하지도 않았다. 따라서 행렬 곱셈을 통해서 보듯이 우리는 더욱 효율적인 행렬 곱셈 방법 즉 더욱 효율적인 행렬 곱셈 알고리즘이 반드시 필요한 것이다.

효율적인 알고리즘의 필요성은 표 2.1로부터 다시 한번 확인할 수 있다. 표 2.1은 요구되는 계산량이 서로 다른 알고리즘들이 입력 길이 n 에 대하여 계산을 종료하는데 필요한 시간들을 계산한 것이다. 여기서 계산은 1초당 1,000,000번의 계산을 수행할 수 있는 컴퓨터를 사용하였다고 가정하였다. 표 2.1을 통하여 입력 길이 n 에 대하여 n^3 또는 2^n 만큼의 계산량을 요구하는 알고리즘은 거의 모든 입력 길이에 대하여 실제적인 계산이 불가능하다는 것을 알 수 있다. 그렇다면 과연 어떠한 문제들에 대하여 2^n 만큼의 계산 량이 요구되는 알고리즘만이 존재하는가 하는 의문을 가질 수 있지만 이러한 문제들을 NP-complete 계열의 문제라고 하며 이러한 문제들을 모든 책 (Garey) 이 이미 오래 전에 발간되었다.

만일 우리가 $n \times n$ 행렬 2개의 곱셈을 n^2 만큼의 계산 량으로 수행할 수 있는 알고리즘을

개발하였다고 하자. 그렇다면 이전 분석에서 $n = 100{,}000$ 에 대하여 최소한 32 년이 요구되던 계산이 단지 3 시간 만에 해결할 수 있다는 것을 표 2.1로부터 확인할 수 있다.

알고리즘 개발의 궁극적인 목적은 최적 알고리즘 (optimal algorithm)을 개발하는데 있다. 최적 알고리즘이란 주어진 문제를 해결하는데 최소한의 계산 량을 요구하는 알고리즘을 의미하며 다음과 같은 방법으로 이를 증명할 수 있다. 주어진 문제 P 를 해결할 수 있는 알고리즘 A 가 있다고 하자. 먼저 수학적 분석을 통하여 어떠한 알고리즘도 문제 P 를 해결하는데 최소한 $F(n)$ 만큼의 계산 량이 요구된다는 것을 증명한다. 그 다음에는 수학적 분석을 통하여 알고리즘 A 는 최악의 경우에 $A(n)$ 만큼의 계산 량이 요구된다는 것을 증명한다. 끝으로 다음과 같이 $F(n)$ 과 $A(n)$ 값의 비교를 통하여 최적화 여부를 판단한다.

- $F(n) = A(n)$: 이 경우 어떠한 다른 알고리즘도 A 보다 적은 양의 계산만으로 문제 P 를 해결할 수 없으므로 알고리즘 A 는 최적 알고리즘이다.
- $F(n) \neq A(n)$: 이 경우에는 2 가지 해석이 가능하다. 알고리즘 A 보다 더욱 효율적인 알고리즘을 개발할 수 있거나 또는 $F(n)$ 값을 상향 조정하야 할 가능성이 있다.

표 2.1. 서로 다른 알고리즘이 요구하는 계산 량.

알고리즘이 요구하는계산량	입력 길이			
	$n = 50$	$n = 100$	$n = 1{,}000$	$n = 10^5$
$\log_2 n$	6×10^{-6}초	7×10^{-6}초	10^{-5}초	2×10^{-5}초
n	5×10^{-5}초	10^{-4} 초	10^{-3}초	0.1 초
$n \log_2 n$	3×10^{-4}초	7×10^{-4}초	10^{-2}초	2 초
n^2	3×10^{-3}초	0.01 초	1 초	3 시간
n^3	0.13 초	1 초	16.7분	32 년
2^n	36 년	4×10^{16}년	3×10^{287}년	3×10^{30089}년

이러한 개념들을 바탕으로 다음절에서는 행렬 곱셈에서 기존에 알려진 방법보다 더욱 효율적인 Strassen 알고리즘을 분석하도록 한다.

Ⅲ. 효율적인 행렬 곱셈 알고리즘

1. Strassen 행렬 곱셈 알고리즘

이전 절에서 (2×2) 행렬 2 개의 곱셈 과정에는 모두 $2^3 = 8$ 개의 곱셈이 요구된다는 것을 분석하였으며 그림 3.1은 이러한 곱셈 과정을 보여준다. Strassen 알고리즘의 핵심은 (2×2) 행렬 2개를 곱할 때 그림 3.1처럼 8개의 곱셈을 사용하는 것 이 아니라 모두 7개의 곱셈만을 사용한다는 점에 있다. $A \times B = C$ 에서 A, B 를 (2×2) 크기의 행렬이라 하자. 그렇다면 결과값 행렬인 C 또한 (2×2) 행렬이 된다 (그림 3.2 참조). Strassen 알고리즘은 먼저 그림 3.3처럼 $x1 \sim x7$ 까지의 계산을 수행한다.

$$\begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix} \times \begin{bmatrix} b11 & b12 \\ b21 & b22 \end{bmatrix} = \begin{bmatrix} a11\times b11 + a11 \times b21 & a11\times b12 + a12 \times b22 \\ a21\times b11 + a22 \times b21 & a21\times b12 + a22 \times b22 \end{bmatrix}$$

그림 3.1. $A \times B$

$$A = \begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix}, \quad B = \begin{bmatrix} b11 & b12 \\ b21 & b22 \end{bmatrix}, \quad C = \begin{bmatrix} c11 & c12 \\ c21 & c22 \end{bmatrix}$$

그림 3.2. (2×2) 행렬 A, B 및 C.

$$x1 = (a11 + a22) \times (b11 + b22)$$
$$x2 = (a21 + a22) \times b11$$
$$x3 = a11 \times (b12 - b22)$$
$$x4 = a22 \times (b21 - b11)$$
$$x5 = (a11 + a12) \times b22$$
$$x6 = (a21 - a11) \times (b11 + b12)$$
$$x7 = (a12 - a22) \times (b21 + b22)$$

그림 3.3. $x1 \sim x7$ 의 계산 과정.

그림 3.3에서 $x1 \sim x7$ 을 계산하는 과정에는 모두 7 개의 곱셈과 10 개의 덧셈 / 뺄셈이 사용되었으며 이들을 통한 실제적인 행렬 곱셈은 그림 3.4와 같이 수행된다.

$$c11 = x1 + x4 - x5 + x7 \ (= a11 \times b11 + a12 \times b21)$$
$$c12 = x3 + x5 \ (= a11 \times b12 + a12 \times b22)$$
$$c21 = x2 + x4 \ (= a21 \times b11 + a22 \times b21)$$
$$c22 = x1 + x3 - x2 + x6 \ (= a21 \times b12 + a11 \times b22)$$

그림 3.4. Strassen 알고리즘의 실제 행렬 곱셈 과정.

따라서 행렬 C 를 계산하는 과정에서 추가적으로 모두 8 개의 덧셈 / 뺄셈이 사용되었으며 결론적으로 Strassen 알고리즘에서 (2×2) 행렬 2 개를 곱하는 과정에서 모두 7 개의 곱셈과 18 개의 덧 셈 / 뺄셈이 요구된다. 지금까지는 Strassen 알고리즘에서 (2×2) 행렬들을 곱하는 과정을 분석하였다. Strassen 알고리즘에서 일반적인 크기의 행렬들의 곱셈은 (2×2) 행렬들을 곱하는 방법을 기반으로 다음과 같이 수행된다. n 을 2 의 멱승 형태의 숫자라고 하자 (즉, $n = 2^k$). ($n \times n$) 크기를 갖는 행렬들의 곱셈에서 Strassen 알고리즘은 A 및 B를 각각 4 개의 ($n/2$) × ($n/2$) 행렬들로 분할한다. 그림 3.5는 (4×4) 크기를 갖는 A 행렬 및 분할이 이루어진 A' 행렬을 보여준다. A' 행렬은 (2×2) 크기를 갖는 4 개의 부분 행렬들로 이루어진 (2×2) 행렬로 취급할 수 있다. 따라서 A 및 B 행렬 들은 각각 4 개의 ($n/2$) × ($n/2$) 크기의 부분 행렬들로 구성된 (2×2) 행렬들로 취급할 수 있으며 이들을 통하여 그림 3.3의 $x1 \sim x7$ 의 계산을 수행한다. 물론 $x1 \sim x7$ 의 계산 과정에 서 다시 Strassen 의 알고리즘이 반복적으로 적용 된다.

$$A = \begin{bmatrix} a11 & a12 & a13 & a14 \\ a21 & a22 & a23 & a24 \\ a31 & a32 & a33 & a34 \\ a41 & a42 & a43 & a44 \end{bmatrix}, \quad A' = \begin{bmatrix} a11 & a12 & a13 & a14 \\ a21 & a22 & a23 & a24 \\ a31 & a32 & a33 & a34 \\ a41 & a42 & a43 & a44 \end{bmatrix}$$

그림 3.5. (4×4) 행렬 A 의 분할 과정.

여기서 매우 중요한 사항은 Strassen 알고리즘에 서 요구하는 7 개의 계산, 즉, $x1 \sim x7$ 의 계산 과정을 보면 곱셈의 교환 법칙이 적용되지를 않았다는 것이다. 즉, $ab = ba$ 는 전혀 사용이 되지를 않았다는 것이다. 이는 Strassen 알고리즘에 매우 중요한 사항으로서 곱셈에서 항목들이 행렬이라면 교환 법칙이 성립되지를 않기 때문이다. 즉, 만일 A, B 가 행렬들이라면 $AB \neq BA$ 이다 (물론 $A = B$ 라면 $AB = BA$ 이지만). 따라서 예를 들어, $x4$를 계산할 때 반드시 $a22 * (b21 - b11)$ 의 순서로 계산되어야 한다. 이 것을 $(b21 - b11) * a_{22}$ 로 계산하면 결과 값은 틀린 것이 나올 수 있게 된다. Strassen 알고리즘의 정확한 계산 과정을 이해하기 위하여 실제적인 예제를 들어보기로 하자. A 및 B 행렬들을 그림 3.6에 표현된 행렬들이라 하고 Strassen 알고리즘에 따라서 행렬 곱셈 $C = A \times B$ 를 계산한다.

$$A = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 2 & 1 & 1 & 2 \\ 3 & 2 & 1 & 1 \\ 2 & 1 & 1 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 1 & 3 & 2 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 1 & 2 \\ 1 & 3 & 2 & 1 \end{bmatrix}$$

그림 3.6. (4×4) 행렬 A 및 B.

그림 3.6의 A, B 행렬들의 크기는 (2×2) 가 아니므로 먼저 그림 3.7처럼 분할이 이루어진다.

$$a11 = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}, \; a12 = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}, \; a21 = \begin{bmatrix} 3 & 2 \\ 2 & 1 \end{bmatrix}, \; a22 = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

$$b11 = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}, \; b12 = \begin{bmatrix} 3 & 2 \\ 1 & 1 \end{bmatrix}, \; b21 = \begin{bmatrix} 2 & 2 \\ 1 & 3 \end{bmatrix}, \; b22 = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

그림 3.7. (4×4) 행렬의 분할 과정.

이제 그림 3.7의 분할된 행렬들로부터 다음과 같이 $x1 \sim x7$ 의 계산과정이 수행된다.

$$x1 = \left(\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}\right) \times \left(\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}\right) = \begin{bmatrix} 2 & 3 \\ 3 & 3 \end{bmatrix} \times \begin{bmatrix} 3 & 3 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 15 & 12 \\ 18 & 15 \end{bmatrix}$$

$$x2 = \left(\begin{bmatrix} 3 & 2 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}\right) \times \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 3 \\ 3 & 3 \end{bmatrix} \times \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 11 & 7 \\ 9 & 6 \end{bmatrix}$$

$$x3 = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \times \left(\begin{bmatrix} 3 & 2 \\ 1 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}\right) = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 0 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 3 & 0 \end{bmatrix}$$

$$x4 = \begin{bmatrix}1&1\\1&2\end{bmatrix} \times \left(\begin{bmatrix}2&2\\1&3\end{bmatrix} - \begin{bmatrix}2&1\\1&1\end{bmatrix}\right) = \begin{bmatrix}1&1\\1&2\end{bmatrix} \times \begin{bmatrix}0&1\\0&2\end{bmatrix} = \begin{bmatrix}0&3\\0&5\end{bmatrix}$$

$$x5 = \left(\begin{bmatrix}1&2\\2&1\end{bmatrix} + \begin{bmatrix}1&2\\1&2\end{bmatrix}\right) \times \begin{bmatrix}1&2\\2&1\end{bmatrix} = \begin{bmatrix}2&4\\3&3\end{bmatrix} \times \begin{bmatrix}1&2\\2&1\end{bmatrix} = \begin{bmatrix}10&8\\9&9\end{bmatrix}$$

$$x6 = \left(\begin{bmatrix}3&2\\2&1\end{bmatrix} - \begin{bmatrix}1&2\\2&1\end{bmatrix}\right) \times \left(\begin{bmatrix}2&1\\1&1\end{bmatrix} + \begin{bmatrix}3&2\\1&1\end{bmatrix}\right) = \begin{bmatrix}2&0\\2&0\end{bmatrix} \times \begin{bmatrix}5&3\\2&2\end{bmatrix} = \begin{bmatrix}10&6\\0&0\end{bmatrix}$$

$$x7 = \left(\begin{bmatrix}1&2\\1&2\end{bmatrix} - \begin{bmatrix}1&1\\1&2\end{bmatrix}\right) \times \left(\begin{bmatrix}2&2\\1&3\end{bmatrix} + \begin{bmatrix}1&2\\2&1\end{bmatrix}\right) = \begin{bmatrix}0&1\\0&0\end{bmatrix} \times \begin{bmatrix}3&4\\3&4\end{bmatrix} = \begin{bmatrix}3&4\\0&0\end{bmatrix}$$

여기서 $x1 \sim x7$ 의 마지막 곱셈 과정은 각 요소들 의 크기가 (2×2)이므로 이전에 분석된 방법에 따라 수행된다. 즉, 예를 들어 $x1$ 의 최종 결과 값 은 (2×2) 행렬들의 곱셈이 필요하므로 다시 $x1 \sim x7$ 들의 값이 계산되면서 Strassen 의 (2×2) 행렬 곱셈 알고리즘을 따른다는 것이다.

이제 최종적으로 $C = A \times B$ 행렬은 다음과 같이 계산된다.

$$c11 = x1 + x4 - x5 + x7 = \begin{bmatrix}15&12\\18&15\end{bmatrix} + \begin{bmatrix}0&3\\0&5\end{bmatrix} - \begin{bmatrix}10&8\\9&9\end{bmatrix} + \begin{bmatrix}3&4\\0&0\end{bmatrix} = \begin{bmatrix}8&11\\9&11\end{bmatrix}$$

$$c12 = x3 + x5 = \begin{bmatrix}0&0\\3&0\end{bmatrix} + \begin{bmatrix}10&8\\9&9\end{bmatrix} = \begin{bmatrix}10&8\\12&9\end{bmatrix}$$

$$c21 = x2 + x4 = \begin{bmatrix}11&7\\9&6\end{bmatrix} + \begin{bmatrix}0&3\\0&5\end{bmatrix} = \begin{bmatrix}11&10\\9&11\end{bmatrix}$$

$$c22 = x1 + x3 - x2 + x6 = \begin{bmatrix}15&12\\18&15\end{bmatrix} + \begin{bmatrix}0&0\\3&0\end{bmatrix} - \begin{bmatrix}11&7\\9&6\end{bmatrix} + \begin{bmatrix}10&6\\0&0\end{bmatrix} = \begin{bmatrix}14&11\\12&9\end{bmatrix}$$

따라서, $c11 \sim c22$ 로부터 그림 3.8과 같이 $C = A \times B$ 의 최종 결과 값을 계산할 수 있다.

$$C = \begin{bmatrix}8&11&10&8\\9&11&12&9\\11&10&14&11\\9&11&12&9\end{bmatrix}$$

그림 3.8. $C = A \times B$ 의 최종 결과값.

$M(k)$ 를 Strassen 알고리즘을 이용하여 ($n \times n$) 행렬들을 곱할 때 발생하는 곱셈의 수라 하자 ($n = 2^k$). Strassen 알고리즘에서 곱셈은 모두 $x1 \sim x7$ 을 계산하는 과정에서만 필요하며 모두 7개의 곱셈이 요구된다. 그러나 이때 곱셈의 대상들이 다 시 부분행렬이 될 수 있음에 주의하여야 한다. 그러나 곱셈의 대상이 부분행렬이라 하여도 분할된 후 이들의 크기는 $2^{k-1} \times 2^{k-1}$ 이므로,

$M(0) = 1$

$M(k) = 7M(k-1)$ for $k > 0$.

라는 재귀 함수를 쉽게 구할 수 있다. 여기서 $k=0$ 이라면 $n=1$ 이므로 이것은 (1×1) 행렬, 즉, 2 개의 실수를 곱하는 것이므로 단 1 개의 곱셈이 요구된다. 따라서 $M(0) = 1$ 이다. 상기한 재귀 함수를 전개하면

$$M(k) = 7\,M(k-1) = 7(7\,M(k-2)) = 7^2M(k-2)$$
$$= 7^2(7\,M(k-3)) = 7^3M(k-3)\cdots\cdots = 7^pM(k-p)$$

이 된다. 따라서 $k=p$ 이면 $M(0) = 1$이므로 최종적으로 $M(k) = 7^k = 7^{\lg n} = n^{\lg 7} \approx n^{2.81}$ 이 된다. 즉, Strassen 알고리즘은 $(n\times n)$ 크기의 행렬들을 곱하는 과정에서 전통적인 행렬 곱셈 알고리즘이 요구하는 n^3 번의 곱셈보다 훨씬 적은 $n^{2.81}$ 번의 곱셈만이 요구되는 것이다. 그림 3.9의 그래프는 행렬의 크기 n 이 증가함에 따라 전통적 인 행렬곱셈 알고리즘이 요구하는 곱셈 량 n^3 과 Strassen 알고리즘이 요구하는 곱셈 량 $n^{2.81}$ 이 변화하는 모양을 그래프로 보여준다.

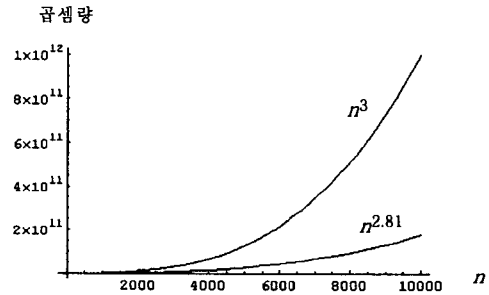

그림 3.9. n^3 과 $n^{2.81}$ 함수의 그래프.

2. Winograd 행렬 곱셈 알고리즘

이전 절에서 Strassen 의 행렬 곱셈 알고리즘이 전통적인 행렬 곱셈 알고리즘 보다 훨씬 효율적이라는 것을 분석하였다. 그러나 Strassen 의 알고리즘은 행렬들을 동일한 크기의 부분 행렬들로 나누어야 하기 때문에 정방행렬 ($n \times n$ 행렬) 이 아닌 일반적인 행렬 곱셈에서는 효율적이지 못할 수가 있다. 따라서 본 절에서는 Winograd 에 의하여 개발된 또 다른 행렬 곱셈 알고리즘을 소개한다. A 및 B 를 그림 3.10의 4 × 4 행렬이라 하자.

$$A = \begin{bmatrix} a11 & a12 & a13 & a14 \\ a21 & a22 & a23 & a24 \\ a31 & a32 & a33 & a34 \\ a41 & a42 & a43 & a44 \end{bmatrix}, \quad B = \begin{bmatrix} b11 & b12 & b13 & b14 \\ b21 & b22 & b23 & b24 \\ b31 & b32 & b33 & b34 \\ b41 & b42 & b43 & b44 \end{bmatrix}$$

그림 3.10. 4 × 4 행렬 A 및 B.

그림 4.1에서 A_i 를 행렬 A 의 i 번째 열(row) 라 하고 B_j 를 B 의 j 번째 행(column) 이라 하고 $C = A \times B$ 라 하자. 예를 들어 A_1 = (a11 a12 a13 a14) 이며 B_1 = (b11 b21 b31

$b41$) 이 된다. 그렇다면 행렬 곱셈은 이들 1차원 벡터들의 연속적인 곱으로 표현될 수 있으며 예를 들어,

$C11 = A_1 \times B_1 = (a11\ a12\ a13\ a14) \times (b11\ b21\ b31\ b41)$

$= (a11b11 + a12b21 + a13b31 + a14b41)$ -(1)

으로 계산된다. Winograd 알고리즘은 이러한 1차원 벡터곱을 다음과 같은 방법으로 계산한다.

$C11 = A_1 \times B_1 = (a11\ a12\ a13\ a14) \times (b11\ b21\ b31\ b41)$

$=(a11 + b21)(a12 + b11) + (a13 + b41)(a14 + b31) - a11a12 - a13a14 - b11b21 - b31b41$ -(2)

(1) 식의 계산과정에는 모두 4 개의 곱셈이 요구되지만 (2) 식의 계산에는 모두 6 번의 곱셈이 요구된다. 따라서 Winograd 알고리즘이 전통적인 방법보다 더 많은 곱셈 수를 요구하는 것처럼 보이지만 Winograd 알고리즘에서는 반복되는 계산이 발생하도록 식이 구성되어 있기 때문에 실제적으로는 전통적인 방법보다 훨씬 적은 수의 곱셈이 요구된다. (2) 식의 계산 과정에는 Strassen 알고리즘과는 달리 곱셈의 교환법칙이 적용된다. 즉, 전통적인 1차원 벡터 곱의 값인 (1) 식과 Winograd 알고리즘으로 계산한 결과 값인 (2) 식은 동일한 것이며 이는 Strassen 알고리즘과는 달리 곱셈의 모든 요소들이 스칼라 값이기 때문에 가능한 것이다. 다음은 상기한 방법에 따라 Winograd 알고리즘에서 $A \times B$ 를 계산하는 과정의 일부분을 보여주고 있다.

[1] $A_1 \times B_1 = (a11 + b21)(a12 + b11) + (a13 + b41)(a14 + b31) - a11a12 - a13a14 - b11b21 - b31b41.$

[2] $A_1 \times B_2 = (a11 + b22)(a12 + b12) + (a13 + b42)(a14 + b32) - a11a12 - a13a14 - b12b22 - b32b42.$

[3] $A_1 \times B_3 = (a11 + b23)(a12 + b13) + (a13 + b43)(a14 + b33) - a11a12 - a13a14 - b13b23 - b33b43.$

[4] $A_1 \times B_4 = (a11 + b24)(a12 + b14) + (a13 + b44)(a14 + b34) - a11a12 - a13a14 - b14b24 - b34b44.$

[5] $A_2 \times B_1 = (a21 + b21)(a22 + b11) + (a23 + b41)(a24 + b31) - a21a22 - a23a24 - b11b21 - b31b41.$

[6] $A_2 \times B_2 = (a21 + b22)(a22 + b12) + (a23 + b42)(a24 + b32) - a21a22 - a23a24 - b12b22 - b32b42.$

[7] $A_2 \times B_3 = (a21 + b23)(a22 + b13) + (a23 + b43)(a24 + b33) - a21a22 - a23a24 - b13b23 - b33b43.$

[8] $A_2 \times B_4 = (a21 + b24)(a22 + b14) + (a23 + b44)(a24 + b34) - a11a22 - a23a24 - b14b24 - b34b44.$

[9] $A_3 \times B_1 = (a31 + b21)(a32 + b11) + (a33 + b41)(a34 + b31) - a31a32 - a33a34 - b11b21 - b31b41.$

[10] $A_3 \times B_2 = (a31 + b22)(a32 + b12) + (a33 + b42)(a34 + b32) - a31a32 - a33a34 - b12b22 - b32b42.$

[11] $A_3 \times B_3 = (a31 + b23)(a32 + b13) + (a33 + b43)(a34 + b33) - a31a32 - a33a34 - b13b23 - b33b43.$

[12] $A_3 \times B_4 = (a31 + b24)(a32 + b14) + (a33 + b44)(a34 + b34) - a31a32 - a33a34 - b14b24 - b34b44.$

[13] $A_4 \times B_1 = (a41 + b21)(a42 + b11) + (a43 + b41)(a44 + b31) - a41a42 - a43a44 - b11b21 - b31b41.$

[14] $A_4 \times B_2 = (a41 + b22)(a42 + b12) + (a43 + b42)(a44 + b32) - a41a42 - a43a44 - b12b22 - b32b42.$

[15] $A_4 \times B_3 = (a41 + b23)(a42 + b13) + (a43 + b43)(a44 + b33) - a41a42 - a43a44 - b13b23 - b31b43.$

[16] $A_4 \times B_4 = (a41 + b24)(a42 + b14) + (a43 + b44)(a44 + b34) - a41a42 - a43a44 - b14b24 - b34b44.$

상기한 계산과정을 분석하면 다음과 같은 특징들을 분석할 수 있다.

- [1] 에서 계산된 $a11a12 - a13a14$ 는 [2], [3], [4] 에서도 다시 중복되게 계산된다.
- [5] 에서 계산된 $a21a22 - a23a24$ 는 [6], [7], [8] 에서도 다시 중복되게 계산된다.
- [9] 에서 계산된 $a31a32 - a33a44$ 는 [10], [11], [12] 에서도 다시 중복되게 계산된다.
- [13] 에서 계산된 $a41a42 - a43a44$ 는 [14], [15], [16] 에서도 다시 중복되게 계산된다.
- [1] 에서 계산된 $b11b21 - b31b41$ 는 [5], [9], [13] 에서도 다시 중복되게 계산된다.
- [2] 에서 계산된 $b12b22 - b32b42$ 는 [6], [10], [14] 에서도 다시 중복되게 계산된다.
- [3] 에서 계산된 $b13b23 - b33b43$ 는 [7], [11], [15] 에서도 다시 중복되게 계산된다.

• [4] 에서 계산된 $b14b24 - b34b44$ 는 [8], [12], [16] 에서도 다시 중복되게 계산된다.

따라서 Winograd 알고리즘에서는 상기한 분석처럼 중복되는 계산들은 한번씩만 계산한 후 이 값을 저장하였다가 필요할 때마다 다시 계산할 필요 없이 저장된 값을 재사용한다는 것이 알고리즘의 핵심이다.

일반적으로 Winograd 알고리즘에서 의하면 길이가 n 인 벡터 $A_x \cdot B_y$ 의 계산을 다음과 같이 수행한다.

(i) 만일 n 이 짝수라면, 즉 $n = 2p$ 인 경우.

$$A_xB_y = \sum_{i=1}^{P}(a_{2i-1} + b_{2i})(a_{2i} + b_{2i-1}) - \sum_{i=1}^{P} a_{2i-1}a_{2i} - \sum_{i=1}^{P} b_{2i-1}b_{2i}$$

(ii) 만일 n 이 홀수라면, p = □□$n/2$ □□

$$A_xB_y = \sum_{i=1}^{P}(a_{2i-1} + b_{2i})(a_{2i} + b_{2i-1}) - \sum_{i=1}^{P} a_{2i-1}a_{2i} - \sum_{i=1}^{P} b_{2i-1}b_{2i} + a_nb_n.$$

만일 n 이 홀수라면 A_xB_y 에서 벡터들의 마지막 항을 포함시키지 않고 단순히 a_nb_n 을 더한다는 것을 (ii) 를 통해서 알 수 있다. 또한 (i) 및 (ii) 를 보면 각각 3개의 $\sum$ 에서 각각 p 번의 곱셈이 발생함을 알 수 있다. 따라서, n 이 짝수이거나 또는 홀수일 때 최대한 3□□$n/2$ □□번의 곱셈이 요구된다. (물론, n 이 홀수일 때에는 마지막 a_nb_n 이 있지만 이는 고려하지 않는다). 따라서 $(m \times n)$ 및 $(n \times q)$ 인 2개 행렬들을 Winograd 알고리즘을 이용하면 모두 mq × 3□□$n/2$ □□번의 곱셈이 발생하며 전통적인 방법에서 요구되는 mnq 번의 곱셈보다 더 많은 수의 곱셈이 발생할 수도 있음을 알 수 있다. 그러나 예제를 통해서 보았듯이 Winograd 알고리즘에서는 중복되는 계산들이 있다. 따라서 이들 값을 먼저 처리한 후 이들 값들을 반복적으로 계산하지 않고 다시 사용하면 전통적인 방법보다 적은 수의 곱셈이 요구되며 실제적으로는 모두 $n/2(m + q) + mnq/2$ 개의 곱셈이 요구된다.

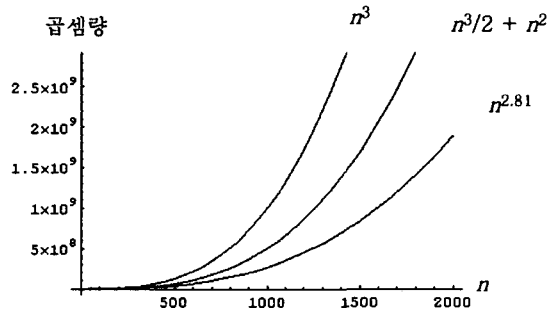

그림 3.11. 3 가지 행렬 곱셈의 효율 그래프

만일 $n \times n$ 크기의 행렬들을 Winograd 알고리즘으로 계산하였다고 가정하면 모두 $(n^3/2) + n^2$ 번의 곱셈이 요구되며 이것은 n^3 번의 곱셈을 요구하는 전통적인 행렬 곱셈 알고리즘에 비하여 훨씬 효율적이라는 것을 쉽게 알 수 있다. 그림 3.11는 $n \times n$ 크기의 행렬 곱셈

에 있어서 전통적인 방법 (n^3), Strassen 알고리즘 ($n^{2.81}$) 및 Winograd 알고리즘 ($n^3/2 + n^2$) 이 요구하는 곱셈 량을 그래프로 나타낸 것이다. 여기서 Winograd 알고리즘은 Strassen 알고리즘보다 많은 양의 곱셈을 요구하지만 Strassen 알고리즘은 정방행렬의 곱셈에 최적이지만 Winograd 알고리즘은 정방행렬이 아닌 일반적인 크기의 행렬 곱셈에도 쉽게 적용될 수 있다는 점을 감안하여야 한다.

Ⅳ. 결 론

7차 교육과정에서 새로이 소개된 이산수학 분야에는 알고리즘이라는 고등학생들에게는 다소 생소한 분야가 포함되어 있다. 알고리즘은 대개의 학생들이 처음 접하는 분야로서 알고리즘을 교수하는 과정에는 학생들로부터 학습에 대한 동기를 유발 할 수 있도록 학생들이 이미 풀이 과정에 익숙하다고 느끼고 있는 문제에 대한 풀이 과정에 실제적으로 포함하고 있는 문제점을 제시하고 이러한 문제점들을 해결하기 위하여 알고리즘에 대한 학습이 필요하다는 것을 보여주는 과정이 반드시 필요하다고 판단된다. 이러한 목적을 위하여 본 연구에서는 알고리즘의 정의 및 특성을 분석하고 행렬 곱셈의 문제를 통하여 전통적인 행렬 곱셈 알고리즘이 포함하고 있는 한계성을 지적하고 이를 극복 할 수 있는 더욱 효율적인 알고리즘을 소개 및 분석하였다.

미국 NCTM Standard 에서는 이산 수학 교육을 K-12 전 학년에 걸쳐 실시할 것을 권장하고 있다. 이는 고등학교는 물론 초등학교에서부터 이산수학 교과과정의 내용을 다루어 왔다는 것을 의미한다. 이산 수학의 가장 핵심적인 내용은 효율 적인 알고리즘 개발에 있으며 이는 컴퓨터 및 IT 산업의 발전에 필수적인 요소이기 때문이다. 알고리즘 개발에는 수학적 계산 능력보다 수학적인 사 고 능력이 더욱 요구되는 분야이므로 학생들의 자발적인 학습 참여가 중요하다. 따라서 알고리즘 교과 과정은 학생들의 학습 참여도를 높일 수 있는 방향으로 진행되어야 할 것이다.

참 고 문 헌

교육부 (1997). 제 7차 수학과 교육과정, 교육부 고시 제 1997-15 호.

Johnsonbaugh, R. (1990). Discrete Mathematics, Macmillan Publishing Co.

Garey, M. R. & Johnson,D.S. (1979). Computers and Intractability: A Guide to the Theory of NP-completeness, Freeman, San Francisco, California.

NCTM(1995). Curriculum and evaluation standards for school mathematics, Reston, VA : The National Council of Teachers Mathematics. INC.

Research on the Teaching Method for the Discrete Mathematics in School

Han, Keun-Hee[1)]

Abstract

As the development of computer science discrete mathematics has been developed accordingly. Discrete mathematics is one of the vital element for the development of the computer and IT technologies since it is the theoretical basis for these field of technologies. Currently, according the Seventh Curriculum Standards in Mathematics, high school students may participate in the class of discrete mathematics as one of their optional curriculum. However, discrete mathematics is a new to the most students in high school. Therefore, the teaching methods for the class of discrete mathematics must be carefully designed so that students acknowledge the importance of this new subject. For this purpose, we first show that why the algorithm is needed and then analyze the problem involved in the method of the traditional matrix multiplications. Finally, we suggest two matrix multiplication algorithms which are more efficient than the traditional method.

Key words : Discrete Mathematics, Algorithms, Matrix Multiplication

1) Department of Applied Mathematics, Kongju University, kehan@kongju.ac.kr