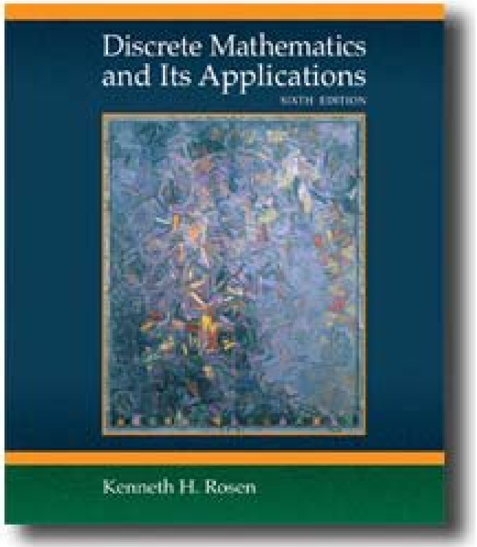
컴퓨터 수학1

3장. 알고리즘, 정수, 행렬



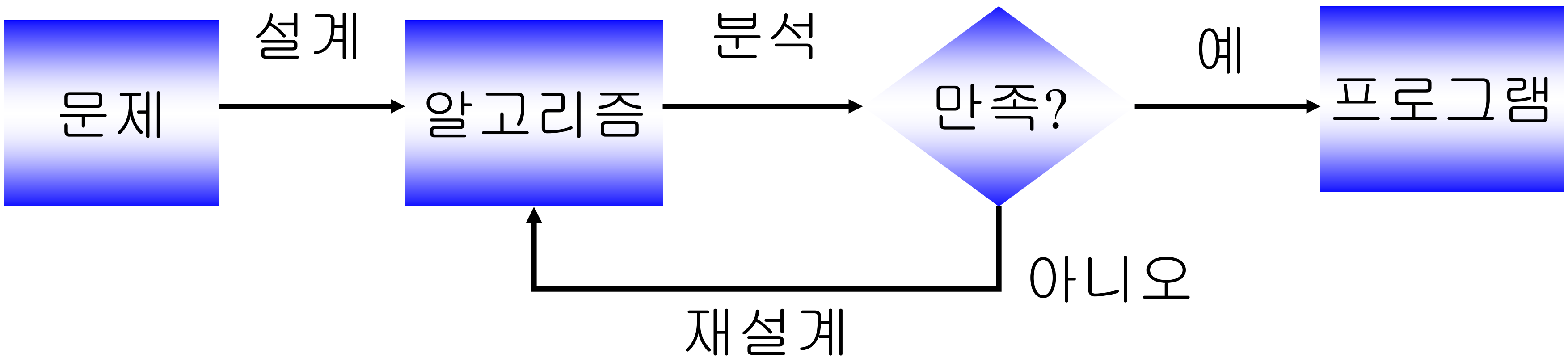
Contents

- 알고리즘
- 함수의 증가
- 알고리즘의 복잡도
- 정수와 나눗셈
- 소수와 최대공약수
- 정수와 알고리즘
- 정수론의 응용
- 행렬

3.1 알고리즘(Algorithm)

- 프로그램(Program)
  - 컴퓨터를 이용하여 실세계의 문제를 모델링하고 이를 해결하기 위한 일련의 절차
  - 문제 해결을 위해 실행되어야 하는 명령들의 순서

- 프로그램의 설계 과정



알고리즘(Algorithm)

- 알고리즘
  - 문제에 대한 해답을 찾기 위한 계산 절차(프로시져)
    - 단계별로 주의 깊게 설계
  - 입력을 받아서 출력으로 변화시키기 위한 일련의 계산 절차

- 프로그램과의 관계
  - 프로그램은 문제를 해결하기 위한 작업 단위(모듈)로 구성
  - 알고리즘: 특정 작업을 수행하는 모듈 설계

- 알고리즘의 기대 효과
  - 프로그램의 안정성 보장
  - 효율적인 프로그램의 작성 (적은 비용)

알고리즘(Algorithm)

- 효율적인 알고리즘의 중요성
  - 문제: 전화번호부에서 홍길동’의 전화번호를 찾는다.
  - 알고리즘
    - 순차검색 (Sequential Search)
      - 첫 페이지부터 끝까지 하나씩 검사하여, 홍길동이 나올 때까지 찾는다.
    - 수정된 이진 검색 (Modified Binary Search)
      - 전화번호부는 가다다 순으로 작성
      - ‘ㅎ’ 부분으로 찾아가 ‘홍’을 찾는다
      - ‘ㄱ’, ‘ㄷ’등 도 같은 방식으로 찾는다.

배열의 크기	순차검색	이진검색
n	n	$\log n + 1$
128	128	8
1,024	1,024	11
1,048,576	1,048,576	21
4,294,967,296	4,294,967,296	33

알고리즘(Algorithm)

- 알고리즘의 예: 유한한 정수 수열(집합)에서 최대값 찾기
  - 알고리즘:
    1. 수열에서 첫 정수를 임시 최대값으로 설정.
    2. 수열의 다음 정수를 임시 최대값과 비교, 임시 최대값보다 크면 그 값을 임시 최대값으로 설정.
    3. 수열에 남은 정수가 있다면 2번을 반복.
    4. 수열 안에 있는 모든 정수와의 비교가 끝나면 임시 최대값을 반환.

- 알고리즘의 표시
  - <u>자연어</u>: 한글 또는 영어
  - <u>프로그래밍언어</u>: C, Pascal, C++, Java, ML 등
  - <u>Pseudo-code(의사코드)</u>
    - 직접 실행할 수 있는 프로그래밍 언어는 아님
    - 실제 프로그램에 가깝게 계산과정을 표현할 수 있는 언어
  - 알고리즘은 일반적으로 Pseudo-Code 사용

알고리즘(Algorithm)

- (예) 수열의 최대값을 찾는 알고리즘
  - Pascal 언어 기반

procedure *max*(a_1, a_2, …, a_n : integers)

  max :=a_1

  for i:=2 **to** n

    if *max*<a_i **then** *max*:=a_i

  {*max* is the largest element}

  - 배열 인덱스의 범위에 제한 없음
    - 프로그래밍 언어는 특정 값(예: 0 또는 1)부터 시작
    - Pseudo-Code(의사코드)는 임의의 값 사용 가능
  - 프로시저의 파라미터에 2차원 배열 크기의 가변성 허용
    - 예: void pname(A[ ][ ]) { … }

알고리즘(Algorithm)

- 의사 코드(Pseudo-code)
  - 지역배열에 변수 인덱스 허용
    - 예: keytype S[low..high];
  - 수학적 표현식 허용
    - low <= x && x <= high   low <= x <= high
    - temp = x; x = y; y = temp   exchange x and y
  - 임의 Type 사용 가능
    - index : 첨자로 사용되는 정수 변수
    - number : 정수(int) 또는 실수(float) 모두 사용가능
    - bool : "true" 나 "false" 값을 가질 수 있는 변수
    - 논리 연산자 and, or, not 사용
  - 제어 구조
    - repeat (n times) { … }
  - 프로시저와 함수
    - 프로시저 : void pname(…) {…}
    - 함 수 : returntype fname (…) {… return x;}

알고리즘(Algorithm)

- 문제의 표기
  - 문제 : 답을 찾고자 던지는 질문
  - 파라미터 : 문제에서 특정값이 주어지지 않은 변수
  - 문제의 사례 (입력) : 파라미터에 특정 값을 지정한 것
  - 사례에 대한 해답 (출력) : 주어진 사례에 관한 질문에 대한 답

- 알고리즘의 특징
  - 명확성(definiteness) : 알고리즘의 각 단계는 명확히 정의되야 한다.
  - 정확성(correctness) : 알고리즘의 각 입력값에 대해 정확한 출력값을 만들어야 한다.
  - 유한성(finiteness) : 알고리즘은 집합에 있는 어떤 입력에 대해서도 유한한 수의 단계를 거친 후 원하는 출력이 나와야 한다.
  - 효율성(effectiveness) : 알고리즘의 각 단계를 유한한 양의 시간 안에 수행될 수 있어야 한다.
  - 일반성(generality) : 프로시저는 단지 특정한 집합의 입력값에 대해서만 아니라 요구하는 모든 형태의 문제에 적용될 수 있어야 한다.

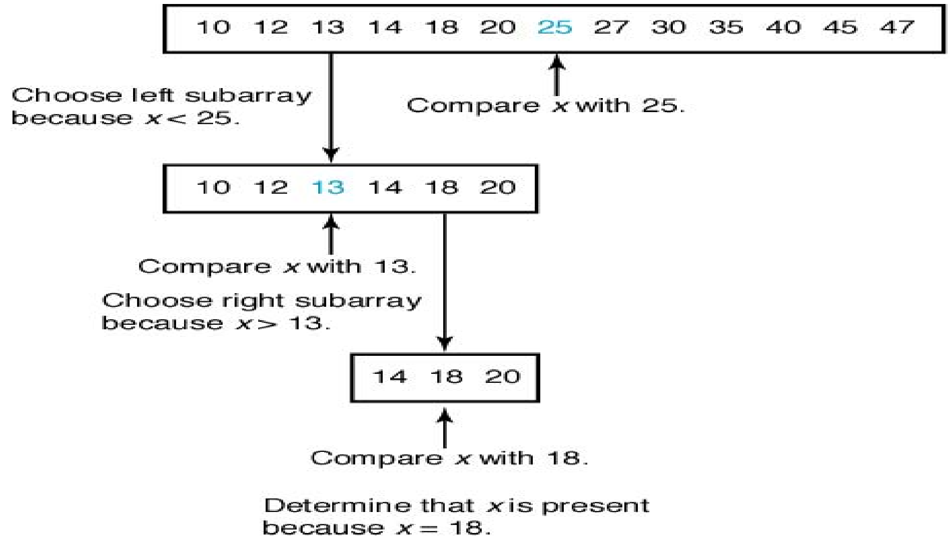
알고리즘(Algorithm)

- 탐색 알고리즘(Searching Algorithm)
  - 리스트(배열, 집합)에서 특정 원소를 찾는 문제
  - 선형 탐색(sequential search, linear search)
    - 리스트를 구성하는 원소들을 순서대로 비교하여 특정 원소를 찾는 방법
  - (선형 탐색 알고리즘): $\{a_1, a_2, \cdots, a_n\}$에서 x를 찾음

procedure *linearSearch*(x:integer, $a_1, a_2, \cdots, a_n$:distinct integers)
i := 1
while($i \leq n$ and $x \neq a_i$)
  i := i + 1
if $i \leq n$ then *location* := i
else *location* := 0
{*location* is the subscript of the term that equals x, x가 발견안되면 location=0}

알고리즘(Algorithm)

- 탐색 알고리즘(Searching Algorithm)
  - 이진 탐색(binary Search)
    - 리스트의 원소들이 정렬되어 있는 경우에 사용
    - 리스트의 중앙에 위치한 원소와 찾고자 하는 원소를 비교
      - 찾고자 하는 원소가 중앙의 원소보다 크면 리스트의 오른쪽 영역을 탐색
      - 찾고자 하는 원소가 중앙의 원소보다 작다면 리스트의 왼쪽 영역을 탐색
    - 리스트의 오른쪽 또는 왼쪽 영역을 탐색할 경우도 마찬가지 방법을 적용
      - 리스트의 탐색 영역이 절반으로 축소



알고리즘(Algorithm)

- 탐색 알고리즘(Searching Algorithm)
  - (이진 탐색 알고리즘): 순차적으로 정렬된 $\{a_1, a_2, \cdots, a_n\}$에서 x를 찾음

procedure *binarySearch*(x:integer, $a_1, a_2, \cdots, a_n$: increasing integers)
i := 1 {i is left endPoint of search interval}
j := n {j is right endPoint of search interval}
while(i < j)
begin
    m := $\lfloor(i+j)/2\rfloor$
    if $x > a_m$ then i := m + 1
    else j := m
end
If $x = a_i$ then *location* := i
else *location* := 0
{*location* is the subscript of the term that equals x, x가 발견안되면 location=0}

알고리즘(Algorithm)

- 정렬 알고리즘(Sorting Algorithm)
  - 리스트의 원소들을 특정 순서로 정리하는 문제
  - Example :
    - 원래 리스트 : 7, 2, 1, 4, 5, 9
    - 오름차순 정렬 : 1, 2, 4, 5, 7, 9
    - 내림차순 정렬 : 9, 7, 5, 4, 2, 1

    - 원래 리스트 : d, h, c, a, f
    - 오름차순 정렬 : a, c, d, f, h
    - 내림차순 정렬 : h, f, d, c, a
  - 정렬 알고리즘
    - Selection Sort, Insertion Sort, Exchange Sort, Bubble Sort, Merge Sort, Quick Sort, Heap Sort, etc

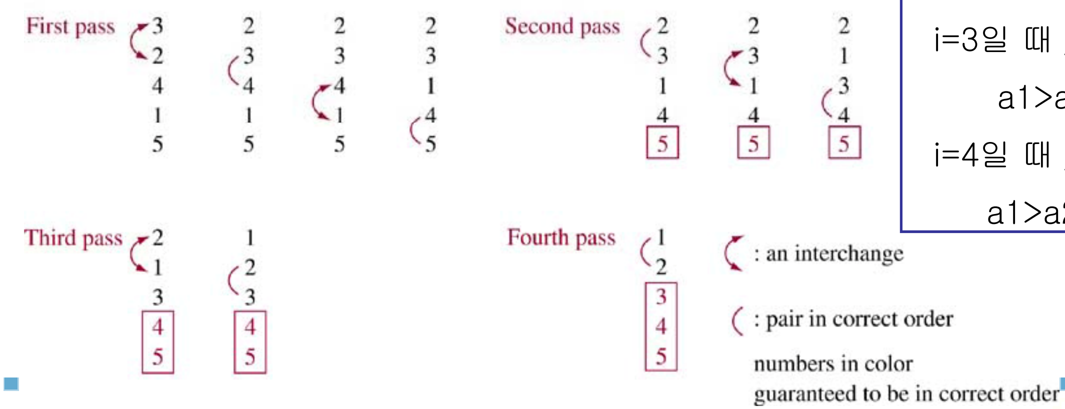
알고리즘(Algorithm)

- 정렬 알고리즘(Sorting Algorithm)
  - 버블 정렬(Bubble Sort)
    - 계속해서 인접 원소와 비교하여 순서에 맞지 않으면 서로 교환
    - 기포가 떠 오르듯, 작은 원소가 큰 원소와 위치가 바뀌어 위로 떠오르고 큰 원소는 바닥으로 가라앉음.
  - 의사 코드

procedure *bubbleSort*(a_1, a_2, ⋯, a_n : number)
for(i :=1 to n-1)
    for(j :=1 to n-i)
        if a_j > a_{j+1} then interchange a_j and a_{j+1}
{a_1, a_2, ⋯, a_n is in increasing order}

n=5일 때
i=1~4
i=1일 때 j=1~4
a1>a2?, a2>a3?,a3>a4?,a4>a5?
i=2일 때 j=1~3
a1>a2?, a2>a3?,a3>a4?
i=3일 때 j=1~2
a1>a2?, a2>a3?
i=4일 때 j=1
a1>a2?

(예)



알고리즘(Algorithm)

- 정렬 알고리즘(Sorting Algorithm)
  - 삽입 정렬(Insertion Sort)
    - 리스트의 원소 하나 하나를, 완전 정렬되었을 때 들어가야 할 위치에 삽입
  - (예): 3, 2, 4, 1, 5 의 정렬

3 2 4 1 5

2 3 4 1 5

2 3 4 1 5

1 2 3 4 5

1 2 3 4 5

알고리즘(Algorithm)

- 삽입 정렬의 의사 코드
  - procedure *insertion sort*($a_1, a_2, \cdots, a_n$: real numbers with n≥2)

    for j:=2 to n

    begin

     i:=1

     while $a_j > a_i$

      i:=i+1

     m:= a_j

     for k:=0 to j-i-1

      $a_{j-k} := a_{j-k-1}$

     a_i := m

    end {$a_1, a_2, \cdots, a_n$ are sorted}

<3,2,4,1,5 를 정렬 시킬 경우>

j=4일 때

i=1

a4>a1 가 아니므로 while loop을 벗어남(i는 그대로 1)

m=a4

for k=0 to 4−1−1=2

a4는 a3 값으로 대치

a3는 a2 값으로 대치

a2는 a1 값으로 대치

a1은 m(즉, a4값)으로 대치

알고리즘(Algorithm)

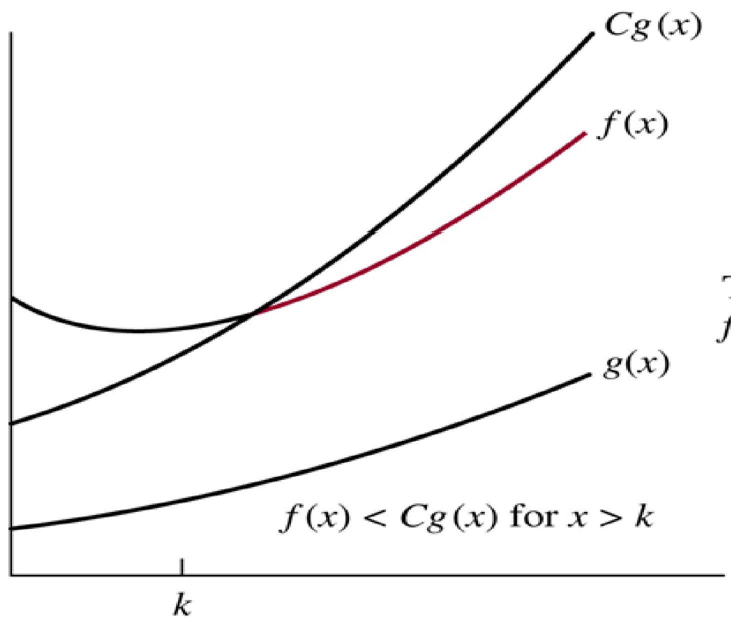
- 욕심쟁이 알고리즘 (Greedy Algorithm)
  - 결정을 해야 할 때마다 그 순간에 가장 좋다고 생각되는 해답을 선택함으로써 최종적인 해답에 도달
  - 그 순간의 선택은 당시에는 최적(locally optimal)
  - 그러나 최종적으로 얻어진 얻어진 해답이 반드시 최적해(optimal solution)가 되는 것은 아님
    - 최적해가 아니라는 것을 보이기 위해서는 반례(counterexample)을 보여야 함
  - 최적의 해(Solution)은 아닐지라도 납득할 만한 해를 요구할 때 가능하다.
    - 서울에서 부산까지 가려면?
    - 잔돈을 거슬러 주는 경우의 문제?

알고리즘(Algorithm)

- (예) 동전의 개수가 최소가 되도록 거스름 돈을 주는 문제
  - 욕심쟁이 알고리즘
    - 거스름돈을 x라 하자.
    - 먼저, 가치가 가장 높은 동전부터 x가 초과되지 않도록 계속 내준다.
    - 이 과정을 가치가 높은 동전부터 내림순으로 총액이 정확히 x가 될 때까지 계속한다.
  - 문제의 예 :
    - 67센트의 거스름돈을 25,10,5,1센트 동전들로 지불하는 경우
    - 욕심쟁이 알고리즘에 의하면 25,25,10,5,1,1 센트 동전 순서로 선택하게 된다
    - 거스름돈 문제에서는 욕심쟁이 알고리즘이 최소의 동전수로 구성하는 해를 제공
      - 즉, 최적의 해
    - But, 특정동전들만 사용해야 한다면 최적의 해가 안 될 수도 있다
      - (예) 30센트의 거스름돈을 구성하는 경우 25,10,1센트 동전들만 사용해야 한다면 욕심쟁이 알고리즘은 25,1,1,1,1,1로 구성하게 되나, 3개의 10센트 동전으로 구성하는 것이 최적해이다.

3.2 함수의 증가

- (Def.1) big-O 표기
  - x > k 일 때 |f(x)| ≤ C|g(x)| 인 C,k가 존재하면 f(x)가 O(g(x))라 함.
    - This is read as "f(x) is big-oh of g(x)"
    - f,g는 정수 또는 실수의 집합으로부터 실수의 집합으로의 함수
    - k, C 는 상수이며 증인(witness)라 불림

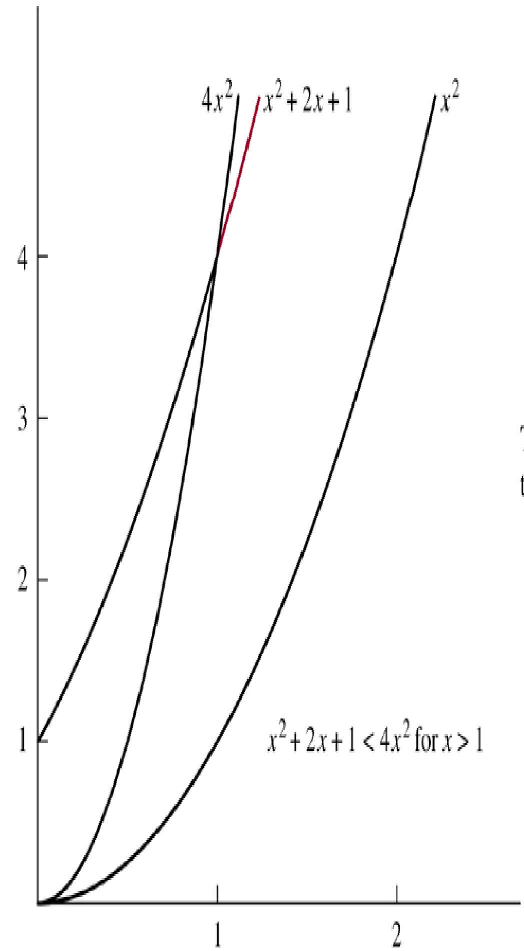


The part of the graph of $f(x)$ that satisfies $f(x) < Cg(x)$ is shown in color.

함수의 증가

- (예) $f(x)=x^2+2x+1$은 $O(x^2)$임을 보여라
  - $0 \le x^2+2x+1 \le x^2+2x^2+x^2 = 4x^2$ ($x>1$ 때)
    - $\therefore C=4$, $x>1$ 일 때 $f(x)= O(x^2)$
  - $0 \le x^2+2x+1 \le x^2+x^2+x^2 = 3x^2$ ($x>2$ 때)
    - $\therefore C=4$, $x>2$ 일 때 $f(x)= O(x^2)$

- (예) $7x^2$이 $O(x^3)$임을 보여라
  - $7x^2 < x^3$ ($x>7$ 일 때)
    - $\therefore C=1$, $x>7$ 일 때 $f(x)= O(x^3)$



The part of the graph of $f(x)=x^2+2x+1$ that satisfies $f(x) < 4x^2$ is shown in color.

함수의 증가

- (정리 1) 다항식의 big-O: n차 또는 이보다 낮은 차수의 다항식은 $O(x^n)$
  - $f(x) = a_n x^n + a_{n-1} x^{n-1} + \quad + a_1 x + a_0$ (단, 계수는 모두 실수) 이면 $f(x)$는 $O(x^n)$

- (예) 첫 n개의 양의 정수의 합의 big-O ?
  - $1+2+ \quad +n$ 에서 각 정수는 n을 초과하지 않으므로 $1+2+ \quad +n \leq n+n+ \quad +n = n^2$
  - $\therefore$ $1+2+ \quad +n$ 은 C=1, k=1을 증인으로 갖는 $O(n^2)$

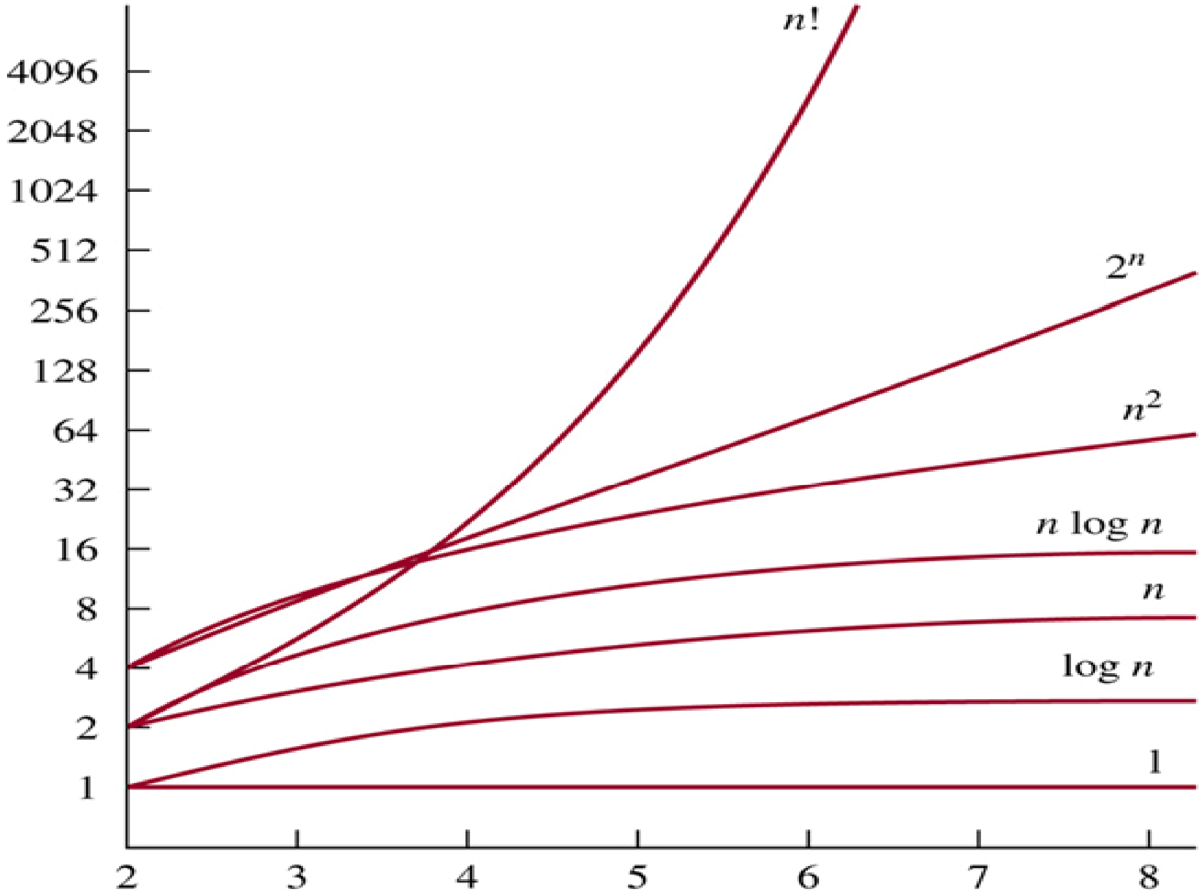
함수의 증가

- (예) 계승함수의 big-O ?
  - $n!=1*2*3* \quad *n$
  - 곱셈의 각항은 n을 초과하지 않으므로 $n!=1*2*3* \quad *n \le n*n* \quad *n=n^n$
  - ∴ n!은 C=1, k=1을 증인으로 갖는 $O(n^n)$
- (예) log n! 의 big-O ?
  - $\log n! \le \log n^n = n \log n$
  - ∴ log n!은 C=1과 k=1을 증인으로 가지는 $O(n \log n)$

- (예) log n의 big-O (n이 양의 정수일 때 $n<2^n$을 이용)?
  - $n<2^n$ 의 양쪽에 $\log_2$ 를 취하면 $\log_2 n < n$
  - ∴ $\log_2 n$은 C=1과 k=1을 증인으로 가지는 $O(n)$
  - $n<2^n$ 의 양쪽에 임의의 log를 취하면 $\log_b n=(\log_2 n)/(\log_2 b)<n/(\log_2 b)$
  - ∴ $\log_b n$은 C=1과 k=1을 증인으로 가지는 $O(n)$

함수의 증가

- Big-O 추정에 자주 사용되는 함수들의 그래프



함수의 증가

- 함수결합의 증가(growth of combinations of functions)
  - $x>k_1$일 때 $|f_1(x)| \le C_1|g_1(x)|$ 이고 $x>k_2$일 때 $|f_2(x)| \le C_2|g_2(x)|$
  - $|a+b| \le |a|+|b|$를 이용하여 $|(f_1+f_2)(x)|=|f_1(x)+f_2(x)| \le |f_1(x)|+|f_2(x)|$
  - $|f_1(x)|+|f_2(x)| \le C_1|g_1(x)|+C_2|g_2(x)| \le C_1|g(x)|+C_2|g(x)| = C|g(x)|$
    - 단, $k=\max(k_1,k_2)$, $g(x)=\max(\ |g_1(x)|,\ |g_2(x)|\ )$, $C=C1+C2$

- (정리 2) $f_1(x)$가 $O(g_1(x))$이고 $f_2(x)$가 $O(g_2(x))$이면 $(f_1+f_2)(x)$는 $O(\max(|g_1(x)|,\ |g_2(x)|)$이다

- (따름정리 1) $f_1(x)$와 $f_2(x)$가 같은 $O(g(x))$면 $(f_1+f_2)(x)$는 $O(g(x)$

- (정리 3) $f_1(x)$가 $O(g_1(x))$이고 $f_2(x)$가 $O(g_2(x))$이면 $(f_1f_2)(x)$는 $O(g_1(x)g_2(x))$

함수의 증가

- (예) $f(x)=(x+1)\log(x^2+1)+3x^2$의 big-O ?
  - $(x+1)$은 $O(x)$
  - $x^2+1 \le 2x^2$ ($x \ge 1$ 때)
  - $\log_2(x^2+1) \le \log_2(2x^2) = \log_2 2 + \log_2(x^2) = \log_2 2 + 2\log_2 x \le 3\log_2 x$ ($x>2$)
  - $\therefore \log_2(x^2+1) = O(\log_2 x)$
  - 정리3으로부터 $(x+1)\log(x^2+1)$은 $O(x\log x)$
  - $3x^2$은 $O(x^2)$
  - 정리2로부터 $f(x)$는 $O(\max(x\log x, x^2))$
  - $x \ge 1$일 때 $x\log x \le x^2$이므로 $f(x)$는 $O(x^2)$

함수의 증가

- big-Omega 표기: 하한의 표기
- big-Theta big-Theta 표기: 하한과 상한의 동시 표기

- (Def. 2) big-Omega 표기
  - $x > k$ 일 때 $|f(x)| \geq C|g(x)|$ 인 C,k가 존재하면 f(x)가 $\Omega(g(x))$라 함
    - f,g는 정수 또는 실수의 집합으로부터 실수의 집합으로의 함수
    - f(x) is big-Omega of g(x)

- (예) $f(x)=8x^3+5x^2+7$의 big-Omega ?
  - 모든 양의 실수 x에 대하여 $f(x)=8x^3+5x^2+7 \geq 8x^3$
  - $\therefore$ f(x)는 $\Omega(x^3)$

함수의 증가

- (Def. 3) big-Theta 표기
  - f(x)가 $O(g(x))$이고 $\Omega(g(x))$면 f(x)는 $\Theta(g(x))$
    - f is big-Theta of g(x) 또는 f(x) is of order g(x) 라고 함.

- (예) $3x^2+8x\log x$ 가 $\Theta(x^2)$임을 증명
  - $x \geq 1$일 때 $x\log x \leq x^2$이므로 $0 \leq 8x\log x \leq 8x^2$
  - $x \geq 1$일 때 $8x\log x+3x^2 \leq 11x^2$ 이므로 $8x\log x+3x^2$ 는 $O(x^2)$
  - x^2은 $O(3x^2+8x\log x)$: 즉 x^2이 $3x^2+8x\log x$의 하한
  - $\therefore$ $3x^2+8x\log x$ 가 $\Theta(x^2)$

- (정리 4) 다항식의 Theta-O: n차 또는 이보다 낮은 차수의 다항식은 $\Theta(x^n)$
  - $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ (단, 계수는 모두 실수, $a_n \neq 0$) 이면 f(x)는 $\Theta(x^n)$

3.3 알고리즘의 복잡도

- Algorithm Complexity Analysis
  - 시간복잡도 분석(Time Complexity Analysis)
    - 입력 크기에 따라 단위 연산이 몇 번 수행되는가를 분석
    - 문제 해결을 위해 얼마나 시간이 오래 걸리는가?
  - 공간복잡도 분석(Space Complexity Analysis)
    - 입력 크기에 따라 얼마나 많은 메모리를 필요로 하는가?
- 표현 척도
  - 단위 연산(Basic Operation)
    - 비교(Comparison) – if
    - 지정(Assignment) – a = a+1;   값을 부여하는 연산
  - 입력 크기
    - 배열의 크기
    - 리스트의 길이
    - 행렬에서 행과 열의 크기
    - 트리에서 마디와 이음선의 수

알고리즘의 복잡도

- 시간복잡도 분석의 방법
  - Every-case analysis
    - 입력크기에만 종속
    - 항상 수행해야 되는 것이므로, 입력 값과는 무관하게 결과 값은 항상 일정
  - Worst-case analysis
    - 입력크기와 입력 값 모두에 종속
    - 단위연산이 수행되는 횟수가 최대인 경우 선택
  - Average-case analysis
    - 입력크기와 입력 값 모두에 종속
    - 모든 입력에 대해서 단위연산이 수행되는 기대치(평균)
    - 각 입력에 대해서 확률 할당 가능
    - 일반적으로 최악의 경우보다 계산이 복잡
  - Best-case analysis
    - 입력크기와 입력 값 모두에 종속
    - 단위연산이 수행되는 횟수가 최소인 경우 선택

알고리즘의 복잡도

- Algorithm Correctness Analysis
  - 최악, 평균, 최선의 경우 분석 방법 중에서 어떤 분석이 가장 정확한가?
  - 최악, 평균, 최선의 경우 분석 방법 중에서 어떤 분석을 사용할 것인가?

  - 알고리즘이 의도한 대로 수행되는지를 증명하는 절차

- 정확한 알고리즘
  - 어떠한 입력에 대해서도 답을 출력하면서 멈추는 알고리즘

- 정확하지 않은 알고리즘
  - 어떤 입력에 대해서 멈추지 않거나, 또는 틀린 답을 출력하면서 멈추는 알고리즘

알고리즘의 복잡도

- (시간복잡도 분석 예) 크기가 n인 배열(집합) S의 수를 모두 더하기
  - Algorithm Pseudo-code

```
number sum (int n, const number S[ ]) {
    index i;
    number result;

    result = 0;
    for (i = 1; i <= n; i++)
        result = result + S[i];
    return result;
}
```

알고리즘의 복잡도

- (시간복잡도 분석 예) 크기가 n인 배열(집합) S의 수를 모두 더하기

  - 단위연산 1: 덧셈, 입력크기: 배열의 크기 n
  - Every-case analysis
    - 배열 값에 상관없이 n번의 for-loop반복: loop당 1회의 덧셈 연산 수행

    ∴n에 대한 덧셈 수행의 총 횟수 T(n) = n

  - 단위 연산 2: 지정연산, 입력크기: 배열의 크기 n
  - Every-case analysis
    - result 초기화를 위한 지정 연산: 1회
    - 배열 값에 상관없이 n번의 for-loop반복: loop당 2회의 지정 연산 수행(i 와 result의 값 변경)

    ∴ 지정 연산 수행의 총 횟수 T(n) = n + n + 1

알고리즘의 복잡도

- (시간복잡도 분석 예) Exchange Sort
  - Pseudo-code

```
void exchangesort (int n, keytype S[ ]) {
    index i, j;
    for (i = 1; i <= n-1; i++)
        for (j = i+1; j <= n; j++)
            if (S[j] < S[i])
                exchange S[i] and S[j];}
```

i=1

j=1	3	2	4	1	5
j=2	2	3	4	1	5
j=3	2	3	4	1	5
j=4	1	3	4	2	5
j=5	1	3	4	2	5

i=2

1	3	4	2	5
1	3	4	2	5
1	2	4	3	5
1	2	4	3	5

i=3

1	2	4	3	5
1	2	3	4	5
1	2	3	4	5

i=4

1	2	3	4	5
1	2	3	4	5

알고리즘의 복잡도

- (시간복잡도 분석 예) Exchange Sort
  - 단위 연산: 비교 연산 – *S[i]* and *S[j]*의 비교
  - 입력 크기: 정렬할 배열의 크기 *n*
  - Every case analysis :
    - i = 1; j-loop *n*-1번 수행
    - i = 2; j-loop *n*-2번 수행
    - i = 3; j-loop *n*-3번 수행

      …
    - i = n-1; j-loop 1번 수행

$$T(n) = (n-1) + (n-2) + \cdots + 1 = \frac{(n-1)n}{2}$$

알고리즘의 복잡도

- (시간복잡도 분석 예) Exchange Sort
  - 단위 연산: exchange $S[i]$와 $S[j]$의 교환
  - 입력 크기: 정렬할 배열의 크기 n
  - Worst case analysis:
    - 입력 배열이 거꾸로 정렬되어 있는 경우, 모든 비교문에서 연산 수행

$$T(n) = \frac{(n-1)n}{2}$$

  - Best case analysis:
    - 입력 배열이 순서대로 정렬되어 있는 경우, 연산 수행하지 않음

$$T(n) = 0$$

알고리즘의 복잡도

- (시간복잡도 분석 예) Linear search(선형 탐색)
  - (알고리즘) 입력 x와 매치되는 원소를 찾기 위하여 list의 첫번째 원소부터 차례로 비교

    procedure *linearSearch*(x:integer, a_1, a_2, ⋯, a_n:distinct integers)
    i := 1
    while($i \le n$ and $x \ne a_i$)
        i := i + 1
    if $i \le n$ then *location* := i
    else *location* := 0

  <비교횟수를 시간복잡도의 척도로 사용>
  - x가 i번째 요소와 같을 때
    - while loop안에 있는 2개의 비교를 i번 수행하게 됨. while loop을 벗어난 후에는 if 문의 비교를 한번 하게 됨 ∴2i+1번의 비교 동작 필요
  - x가 list에 없을 때
    - n개의 요소가 있는 list라면 while loop안에 있는 2개의 비교를 n번 수행하게 됨. i가 n+1이 되었을 때 while loop안의 $i \le n$ 비교를 한번 수행하고 loop를 벗어남. Loop을 벗어난 후 if 문의 비교를 한 번 하게 됨. ∴2n+2번의 비교 동작을 수행

알고리즘의 복잡도

- Notation
  - 알고리즘의 복잡도를 표시하기 위해 사용하는 표기법
  - 근사치를 이용한 복잡도 표시
  - Order(차수)라는 용어를 사용하기도 함
  - Big-O: O, asymptotic upper bound(접근적 상한)
  - Big-Omega: Ω, asymptotic lower bound(접근적 하한)
  - Big-theta: Θ, asymptotic tight bound ($O \cap \Omega$) (접근적 동일)

- 어떤 알고리즘의 시간 복잡도가 O(f(n)) 이라면
  - 입력의 크기 n에 대해서 이 알고리즘의 수행시간은 아무리 늦어도 f(n)은 된다.
    - 이 알고리즘은 수행시간이 f(n)보다 더 걸리지 않는다는 것을 의미한다.
    - 아무리 늦어도 f(n)이다.

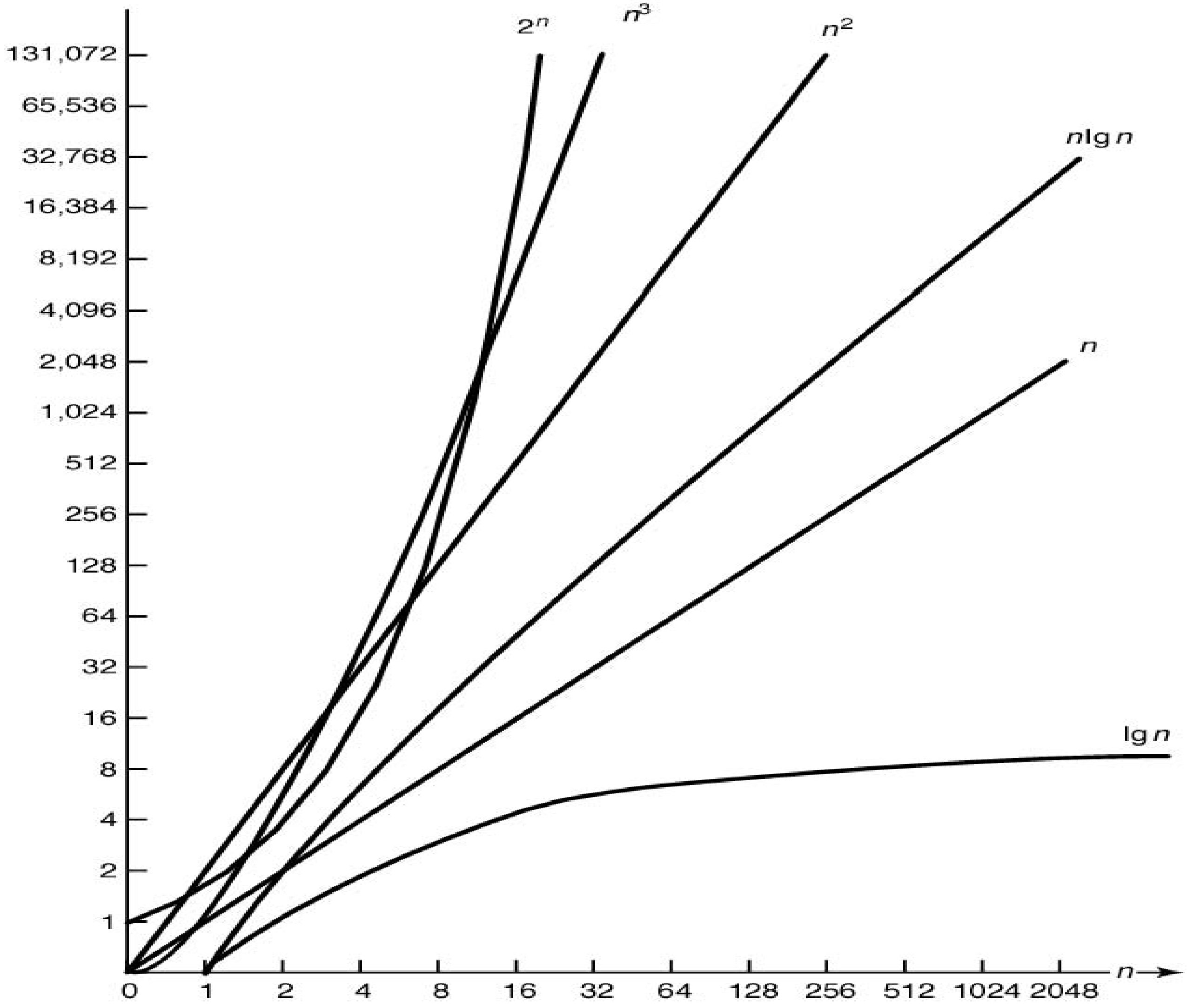
알고리즘의 복잡도

- 흔히 나타나는 알고리즘 복잡도 종류

TABLE 1 Commonly Used Terminology for the Complexity of Algorithms.

Complexity	*Terminology*
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	$n \log n$ complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

알고리즘의 복잡도



131,072
65,536
32,768
16,384
8,192
4,096
2,048
1,024
512
256
128
64
32
16
8
4
2
1
0
1
2
4
8
16
32
64
128
256
512
1024
2048
n
2^n
n^3
n^2
n lg n
n
lg n

알고리즘의 복잡도

TABLE 2 The Computer Time Used by Algorithms.

Problem Size	*Bit Operations Used*					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	3×10^{-9} s	10^{-8} s	3×10^{-8} s	10^{-7} s	10^{-6} s	3×10^{-3} s
10^2	7×10^{-9} s	10^{-7} s	7×10^{-7} s	10^{-5} s	4×10^{13} yr	*
10^3	1.0×10^{-8} s	10^{-6} s	1×10^{-5} s	10^{-3} s	*	*
10^4	1.3×10^{-8} s	10^{-5} s	1×10^{-4} s	10^{-1} s	*	*
10^5	1.7×10^{-8} s	10^{-4} s	2×10^{-3} s	10 s	*	*
10^6	2×10^{-8} s	10^{-3} s	2×10^{-2} s	17 min	*	*

알고리즘의 복잡도

- 다항식에서 알고리즘의 복잡도는 고차 항이 궁극적으로 지배

n	$0.1 \times n^2$	$0.1 \times n^2+n+100$
10	10	120
20	40	160
50	250	400
100	1,000	1,200
1,000	100,000	101,100

- $g(n) = 5n^2+100n+20 \in \Theta(n^2)$, order of n^2

3.4 정수와 나눗셈

- (Def.1) 나눗셈(Division)
  - a, b가 정수이고 a≠0일 때, b=ac인 정수 c가 존재하면 a가 b를 나눈다고 말한다. 이 때

    a는 b의 인수, b는 a의 배수 라고 한다.
  - 표기: $a \mid b$ (a가 b를 나눈다), $a \nmid b$ (a가 b를 나누지 않는다)

- 예제
  - n과 d가 양의 정수 일 때 d에 의해 나누어지고, n을 넘지 않은 정수의 수는?
    - d로 나누어지는 양의 정수는 dk 형태의 모든 정수이며, k는 양의 정수.
    - n을 넘지않고 d로 나누어지는 양의 정수의 개수는 $0< dk \leq n$인 정수 k의 개수
    - 부등식을 d로 나누면 $0 < k \leq (n/d)$
    - ∴ k의 수 : $\lfloor n/d \rfloor$

정수와 나눗셈

- 나눗셈(Division)
  - (정리 1) a,b,c가 정수라 하자. 그러면
    - 1. a | b이고, a | c이면 a | (b+c) 이다.
    - 2. a | b이면, 모든 상수 c에 대해서 a | bc 이다.
    - 3. a | b이고, b | c이면 a | c 이다.
  - 따름 정리 1
    - a, b, c가 a | b, a | c를 만족하면, 정수 m과 n에 대해서 a | (mb+nc)이다.
  - (정리 2) let, a는 정수, d는 양의 정수. a=dq+r을 만족하는 q와 r이 유일하게 존재함(단, 0≤r<d)
    - q는 몫, r은 나머지

- 증명
  - 정리 1.1
    - a | b, a | c 라면 b=as, c=at를 만족하는 정수 s, t가 존재
    - 이를 더하면 b+c = as+at = a(s+t)
    - ∴ a | (b+c)

정수와 나눗셈

- (예) 101을 11로 나눌 때 몫과 나머지 ?
  - 101=11*9+2
  - 몫: 101 div 11 = 9,   나머지: 101 mod 11 = 2
- (예) -11을 3으로 나눌 때 몫과 나머지 ?
  - -11 = 3*(-4) +1
  - 몫: -11 div 3 = -3,   나머지: -11 mod 3 = 1
  - 나머지는 음이 될 수 없다

정수와 나눗셈

- 모듈로 연산(Modular Arithmetic)
  - (Def.2)
    - a=dq+r (정리 2 참조)이면 q=a div r, r=a mod d로 표기 (q는 몫, r은 나머지)
  - (Def.3) 나머지가 동일한 두 수는 "모듈로 합동" 이라 함
    - m이 (a-b)를 나누면, a는 b에 대해 모듈로 m합동(a is congruent to b modulo m)이라 함(단, a와 b는 정수, m은 양의 정수).
    - 표기법: $a \equiv b \pmod{m}$, $a \not\equiv b \pmod{m}$ (합동이 아닐 때)
  - (정리 3) a와 b가 정수, m이 양의 정수라 하자. <u>$a \equiv b \pmod{m}$의 필요 충분 조건은 a mod m = b mod m</u>
    - 즉, a와 b를 m으로 나눈 나머지가 같아야 한다.
  - (정리 4) m을 양의 정수라 하자. 정수 a와 b가 $a \equiv b \pmod{m}$이기 위한 필요충분 조건은 a=b+km인 k가 존재.
    - $a \equiv b \pmod{m}$이면 m | (a-b), 이는 a-b=km인 정수 k가 존재함을 의미 -> a=b+km
    - 역으로 a=b+km이 정수 k가 존재하면 km=a-b, 즉, m|(a-b)이므로 $a \equiv b \pmod{m}$
  - (정리 5) m을 양의 정수라 하자. $a \equiv b \pmod{m}$, $c \equiv d \pmod{m}$ 이면 $a+c \equiv b+d \pmod{m}$, $ac \equiv bd \pmod{m}$
    - c와 d가 같을 때는 $ac \equiv bc \pmod{m}$ 즉, 합동의 양쪽에 같은 수를 곱해도 됨

정수와 나눗셈

- 모듈로 연산(Modular Arithmetic)
  - 따름정리 2
    - (a + b) mod m = { (a mod m) + (b mod m) } mod m
    - ab mod m = { (a mod m) (b mod m) } mod m
  - (정의 3의 예) 17이 5와 모듈로 6 합동인가 (17 ≡ 5 mod 6) ?
    - 6이 17-5를 나누므로 합동임
  - (정의 3의 예) 24가 14와 모듈로 6 합동인가 (24 ≡ 14 mod 6) ?
    - 6이 24-14을 나누지 못하므로 합동이 아님
  - (정리 5의 예)
    - 7 ≡ 2 mod 5 이고, 11 ≡ 1 mod 5이므로
      - 18=7+11 ≡ 2 + 1 = 3 mod m
      - 77 = 7 * 11 ≡ 2 * 1 = 2 mod 5
  - 합동의 응용
    - Hashing function, Randomize, Cryptology

정수와 나눗셈

- 소수(Prime)
  - 오직 1과 자기 자신에 의해서만 나누어지는 1보다 큰 양의 정수
  - (Def. 1) 1보다 큰 양의 정수 p의 양의 인수가 단지 1과 p이면 p는 소수라 함. 소수가 아닌 1보다 큰 양의 정수는 합성수(Composite)라 함.
    - 정수 n이 합성수 => $a \mid n$과 $1 < a < n$을 만족하는 정수 a가 있다는 의미
    - (예) 정수 7은 양의 인수가 1과 7만 존재하므로 소수. 정수 9는 3으로 나누어지므로 소수가 아님.
  - (정리 1) 소인수분해
    - 1보다 큰 모든 양의 정수는 소수이거나 둘 이상의 소수의 곱으로 표현 가능
    - (예) 100의 소인수분해: 100=2*2*5*5, 999의 소인수분해: $999=3*3*3*37=3^3*37$
  - (정리 2) 소인수분해시 소수의 값의 상한
    - n이 합성수이면 n은, $\sqrt{n}$보다 작거나 같은 소인수를 갖는다.
    - 증명 : n이 합성수라면 $1<a<n$인 인수 a가 있다. 따라서 $n = ab$
    - 이 때 a와 b는 1보다 큰 양의 정수
    - 만약 $a>\sqrt{n}$, $b>\sqrt{n}$이면 $ab>\sqrt{n}\sqrt{n} = n$이므로 $a \leq \sqrt{n}$, $b \leq \sqrt{n}$

정수와 나눗셈

- (정리 2의 예) 101 이 소수임을 보이시오
  - $\sqrt{101}$ 을 초과하지 않는 소수는 2,3,5,7 이다. 그런데 101 은 2,3,5,6로 나누어지지 않으므로 101 은 소수이다
- (정리 2의 예) 7007 을 소인수분해 하시오
  - 7007을 2,3,5,7 의 연속된 소수로 나눈다
    - 2,3,5,는 7007을 나누지 못함
    - 7은 7007을 나눔: 7007/7=1001
    - 1001을 7부터 시작하여 연속된 소수로 나눈다
    - 7은 1001을 나눔: 1001/7=143
    - 143을 7부터 시작하여 연속된 소수로 나눈다
    - 7은 143을 나누지 못함
    - 11은 143을 나눔: 143/11=13
    - 13은 소수이므로 완료
    - 7007=7*7*11*13

- (정리 3) 무한히 많은 소수가 존재한다.

정수와 나눗셈

- (Def.) 최대공약수
  - a와 b를 0이 아닌 정수라 하자.
    - 공약수 : d | a, d | b를 만족하는 정수 d
    - 최대공약수 : 공약수 중 최대값, gcd(a,b)로 표기
  - (Def. 3) 두 정수 a,b의 최대공약수가 1이면 이를 a와 b는 서로소 또는 상대 소수 (relatively prime)
- (Def.) 최소공배수
  - 양의 정수 a와 b에 대해서 a와 b에 의해 모두 나누어 질 수 있는 가장 작은 양의 정수
    - 즉, a | d, b | d를 만족하는 양의 정수 d중 최소값, lcm(a,b)로 표기
- (Def.) $ab = \gcd(a,b) \times \mathrm{lcm}(a,b)$ (a,b는 양의 정수)

정수와 나눗셈

- (예) 24와 36의 최대공약수 ?
  - 공약수는 1,2,3,4,6,12 ∴최대공약수 gcd(24,36)=12
- (예) 17과 22의 최대공약수 ?
  - 약수는 1뿐이다. ∴최대공약수 gcd(17,22)=1
  - 17과 22는 서로소 (Def.3 참조)
- (예) 10,17,21은 서로소인가 ?
  - gcd(10,17)=1, gcd(10,21)=1, gcd(17,21)=1 ∴**서로소**
- (예) 10,19,14은 서로소인가 ?
  - gcd(10,24)=2 이므로 서로소가 아님

정수와 나눗셈

- 소인수 분해를 이용하여 최대공약수와 최소공배수 찾기
  - 최대공약수 찾기
    - 0이 아닌 정수 a와 b의 소인수분해가 각각 다음과 같다고 하자.

$$a = p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}, \quad b = p_1^{b_1} p_2^{b_2} \dots p_n^{b_n}$$

    - 이 때 두 정수의 최대공약수는 다음과 같다

$$gcd(a,b) = p_1^{min(a_1,b_1)} p_2^{min(a_2,b_2)} \dots p_n^{min(a_n,b_n)}$$

  - 최소공배수 찾기
    - 0이 아닌 정수 a와 b의 소인수분해가 각각 다음과 같다고 하자.

$$a = p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}, \quad b = p_1^{b_1} p_2^{b_2} \dots p_n^{b_n}$$

    - 이 때 두 정수의 최소공배수는 다음과 같다

$$lcm(a,b) = p_1^{max(a_1,b_1)} p_2^{max(a_2,b_2)} \dots p_n^{max(a_n,b_n)}$$

정수와 나눗셈

- (예) 120과 500의 최대공약수 ?
  - $120 = 2^3 * 3 * 5, \quad 500 = 2^2 * 5^3$
  - $\therefore \gcd(120,500)=2^{\min(3,2)} * 3^{\min(1,0)} * 5^{\min(1,3)} = 2^2 3^0 5^1 = 20$
- (예) $2^3 3^5 7^2$ 와 $2^4 3^3$의 최소공배수 ?
  - $\mathrm{lcm}(2^3 3^5 7^2 , 2^4 3^3) = 2^{\max(3,4)} * 3^{\max(5,3)} * 7^{\max(2,0)} = 2^4 3^5 7^2$

3.6 정수와 알고리즘

- 정수의 표현
  - 10진법
    - 일상적인 수를 사용하기 위해 0~9까지 9개의 숫자를 사용
    - $965 = 9 \cdot 10^2 + 6 \cdot 10^1 + 5 \cdot 10^0$
- 정수의 다른 표현 방법
  - 정수 n의 밑수 b 전개 : b진법
    - b를 1보다 큰 양의 정수이고 n이 양의 정수라 할 때 n은 다음과 같은 표현 가능
    - $n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b^1 + a_0 b^0$
  - 2진법 : 0,1 사용, 밑수 : 2
  - 8진법 : 0~7 사용, 밑수 : 8
  - 16진법 : 0~9, A~F 사용, 밑수 : 16

TABLE 1 Hexadecimal, Octal, and Binary Representation of the Integers 0 through 15.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Octal	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
Binary	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

정수와 알고리즘

- 진법 변환
  - 10진법의 2,8,16진법으로의 변환
  - 2,8,16진법의 10진법으로의 변환

  - (예) $(12345)_{10}$의 밑수 8 (또는 8진수) 전개 ?
    - 8로 계속나눌때 나머지들이 밑수 8 전개의 숫자들이 된다
    - 12345 = 8 * 1543 + 1
    - 1543 = 8 * 192 + 7
    - 192 = 8 * 24 + 0
    - 24 = 8 * 3 + 0
    - 3 = 8 * 0 + 3
    - $(12345)_{10} = (30071)_8$
  - (예) $(177130)_{10}$의 16진 전개 ? : $(2B3EA)_{16}$
  - (예) $(241)_{10}$ 의 2진 전개 ? : $(1111\ 0001)_2$

정수와 알고리즘

- (예) (11 1110 1011 1100)2의 16진 전개 ?
  - 이진 숫자를 4개씩 묶어 변환
  - 맨 앞쪽 블록이 4비트가 안되면 앞에 0을 채워 4비트로 만듦
  - $0011_2=3_{16}$, $1110_2=E_{16}$, $1011_2=B_{16}$, $1100_2=C_{16}$ → $(3EBC)_{16}$
- (예) $(A8D)_{16}$을 이진으로 전개 ?
  - 16진수의 각 자리를 4개의 이진블럭으로 대치
  - 답: $(1010\ 1000\ 1101)_2$

정수와 알고리즘

- 유클리드 알고리즘 (Euclid's Algorithm)
  - 최대공약수를 찾는 효율적 알고리즘
  - 3.5절에서 설명한 두 정수의 소인수 분해를 사용하여 최대공약수를 구하는 것은 시간이 많이 걸려 비효율적임
  - gcd(91,287)
    - 287 = 91 * 3 + 14
    - 287과 91의 어떤 약수던 14의 약수임
    - 91과 14의 어떤 약수던 287의 약수임
    - ∴ 91과 287의 gcd를 찾는 문제는 91과 14의 gcd를 찾는 문제로 축소됨
      - 즉, gcd(91,287) = gcd(91,14)
    - 91 = 14 * 6 + 7
    - 91과 14의 공약수는 7을 나눔
    - 14와 7의 공약수는 91을 나눔
    - ∴ gcd(91,14)=gcd(14,7)
    - 14 = 7 * 2
    - 7이 14를 나누므로 gcd(14,7)=7
    - gcd(14,7)=gcd(91,14)=gcd(91,287)=7

정수와 알고리즘

- 유클리드 알고리즘의 보조정리
  - a = b * q + r 이면 gcd(a,b)=gcd(b,r)   (a,b,q,r은 정수)

- 유클리드 알고리즘: 위의 보조정리를 이용하여 나눗셈을 나머지가 0일 때 까지 계속 진행한 후 마지막 0이 아닌 나머지가 gcd임

- (예) 유클리드 알고리즘 이용하여 414와 662의 최대공약수 찾기
  - 662 (a) = 414 (b) * 1 (q) + 248 (r)
  - 414 = 248 * 1 + 166
  - 248 = 166 * 1 + 82
  - 166 = 82 * 2 + 2
  - 82 = 2 * 41

3.7 정수론의 응용

- (정리 1) gcd(a,b)는 a와 b의 선형결합으로 표현 가능하다
  - 즉, a와 b가 양의 정수이면, gcd(a,b)=sa+tb인 s와 t가 존재
  - (예) gcd(252,198)=18을 252와 198의 선형결합으로 표시하시오
    - 우선 유클리드 알고리즘으로 gcb 를 얻을 때까지 전개

      252 = 1 * 198 + 54 -> 54 = 252 – 1 * 198

      198 = 3 * 54 + 36 -> 36 = 198 – 3 * 54

      54 = 1 * 36 + 18 -> 18 = 54 – 1 * 36

      36 = 2 * 18
    - 18을 유클리드 알고리즘 전개한 순서를 역으로 거슬러올라가며 전개

      18 = 54 – 1 * 36 = 54 – 1 * (198 – 3 * 54) = 4 * 54 – 1 * 198 = 4 * (252 – 1 * 198) – 1 * 198 = 4 * 252 – 5 * 198
    - 즉 18 = 4 * 252 – 5 * 198

정수론의 응용

- (보조정리 1) gcb(a,b)=1이고 albc면 alc
- (보조정리 2) p가 소수이고 a_i가 정수일때 $pla_1a_2 \quad a_n$이면 어떤 i 에 대해 pla_i이다 (보조정리 1의 일반화한 형태)
- 양의 정수의 **소인수분해는 유일하다**
  - (증명) contradiction (모순)dp 의한 증명방법 사용
  - n을 $p_1p_2 \quad p_s$ 와 $q_1q_2 \quad q_t$의 두가지로 소인수 분해할 수 있다고 하자
  - 즉, $n = p_1p_2 \quad p_s = q_1q_2 \quad q_t$
  - 여기서 공통되는 소수를 제거하면 $p_{i1}p_{i2} \quad p_{iu} = q_{j1}q_{j2} \quad q_{jv}$
  - p_{i1}은 $p_{i1}p_{i2} \quad p_{iu}$ 을 나누므로 이 것과 동일한 $q_{j1}q_{j2} \quad q_{jv}$ 도 나누게 된다
  - 그러면 보조정리2에 의해 p_{i1} 에 의해 나누어지는 어떤 q_{jk} 가 존재
  - 그런데 p_{j1} 과 q_{jk} 는 서로소이므로(공통되는 소수는 제외한 상태이므로) 나누지 못한다 -→ 가정에 모순

정수론의 응용

- (정리 2) 모듈로 m합동의 양쪽을 어떤 정수로 나누면 합동이 안 될 수 있다. 그러나 이 나누는 정수가 m과 서로소일때는 합동이 된다.
  - (예) $14 \equiv 8 \pmod{6}$의 양쪽을 2로 나누면 $7 \not\equiv 4 \pmod{6}$
- (정리 3) 모듈로 합동의 역(inverse): $a\bar{a} \equiv 1 \pmod{m}$
  - a와 m이 서로소이면 a modulo m의 역이 존재하며, 유일하다
  - (증명) 정리 1이 의해 gcd(a,m)= sa+tm으로 표현가능
    - gcd(a,m)=1이므로 sa+tm =1, 즉, $sa+tm \equiv 1 \pmod{m}$
    - $tm \equiv 1 \pmod{m}$ 이므로 $sa \equiv 1 \pmod{m}$ → s가 a mod m의 역
    - a와 m을 선형전개(합이 1인) 했을 때 a의 계수가 a mod m의 역이 된다
  - (예) 3 modulo 7의 역은 ?
    - 3과 7이 서로소이므로 역이 존재. 유클리드 알고리즘 적용
      - 7=2*3 + 1 → -2 * 3 + 1* 7 = 1
      - 3과 7의 선형결합(합이 1인)으로 표현시 3의 계수가 역이 되므로 답은 -2
      - -2 mod 7이 합동인 모든 정수 5, -9, 12 등도 3 mod 7의 역임

정수론의 응용

- (정리 4) 중국 나머지 정리 (Chinese remainder theorem)
  - $x \equiv a_1 \pmod{m_1}$, $x \equiv a_2 \pmod{m_2}$ $x \equiv a_n \pmod{m_n}$은 해 $x = a_1M_1y_1+a_2m_2y_2+ \quad +a_nM_ny_n$을 가지며 ($M_k=m/m_k$, y_k는 M_k의 역, $m=m_1m_2 \quad m_n$), 해 중 m 보다 작은 해는 x mod m으로 유일하게 구해진다. (즉, $0 \le x < m$이며 모든 다른 해는 modulo m 합동이다)
  - (예) $x \equiv 2 \pmod 3$, $x \equiv 3 \pmod 5$, $x \equiv 2 \pmod 7$일 때 x는 ?
    - $m=3*5*7=105$, $M_1=m/3 =35$, $M_2 = m/5=21$, $M_3=m/7=15$
    - y_1은 M_1 mod m_1의 역이므로 35 mod 3의 역
      - 35 mod 3 $\equiv$ 2 mod 3 의 역을 구하면 됨. 3=2*1 + 1, 1=3+2*(-1) 역은 -1임, 2, 5 등도 역수
    - y_2는 M_2 mod m_2의 역이므로 21 mod 5의 역
      - 21 mod 5 $\equiv$ 1 mod 5의 역을 구하면 됨. 정리 3의 역의 정의에 의해 1 mod x = 1*1 mod x이므로 역은 1
    - y3는 M3 mod m3의 역이므로 15 mod 7의 역
      - 15 mod 7 $\equiv$ 1 mod 7의 역을 구하면 됨. 1 mod 7의 역은 1
    - $x= 2*35*2+3*21*1+2*15*1=233 \equiv 23 \pmod{105}$

정수론의 응용

- 고대 중국의 수학자는 소수를 결정하기 위한 효과적인 방법을 연구하여, 어떤 n이라는 숫자가 $2^{n-1} \equiv 1 (\mathrm{mod}\ n)$을 만족하면 이 n은 소수라고 생각하였음
  - But, n이 소수면 $2^{n-1} \equiv 1 (\mathrm{mod}\ n)$ 이 되지만 그 역은 항상 성립 안 함
    - (counterexample) $2^{n-1} \equiv 1 (\mathrm{mod}\ n)$인 합성 정수 n이 존재하며 이러한 n을 의사소수(pseudoprime)라 함 (즉 소수는 아니나, $2^{n-1} \equiv 1 (\mathrm{mod}\ n)$을 만족한다는 뜻)
    - (예) 341은 합성정수 (341=11*31)이나 $2^{340} \equiv 1 \mathrm{mod}\ 341$ 을 만족시킴
- (정리 5) Fermat의 작은 정리
  - p가 소수, a가 p에 의해 나누어질 수 없는 정수면 $a^{p-1} \equiv 1 \ (\mathrm{mod}\ p)$
  - 즉, p가 소수면 $a^{p-1} \equiv 1 \ (\mathrm{mod}\ p)$ 인 p 존재
- (Def. 1) $b^{n-1} \equiv 1 \ (\mathrm{mod}\ n)$을 만족하는 합성정수 n을 b를 밑수로 하는 의사소수라 함
  - 의사소수의 개수는 소수에 비해 작다
    - 10^{10}이하에서 소수는 455,052,512개 있지만, 의사소수는 14,884개만 존재

정수론의 응용

- RSA 암호
  - 공개키 암호방식의 일종
  - 1976년 MIT의 세 연구자 Ronald Rivest, Adi Shamir, Leonard Adleman이 고안
  - 암호화 (encryption)
    - 원 메시지 M
    - M^e mod n 하여 전송
      - 공개키는 n(두 소수 p와 q의 곱) 과 e
  - 복호화 (decryption)
    - 수신된 메시지를 d승한 후 mod n을 취함
      - 비밀키는 d: n과 소수인 수 e를 선택한 후, e mod (p-1)(q-1)의 역 (즉, ed mod (p-1)(q-1) ≡ 1)
  - 공개키 시스템의 안전성
    - 나의 공개키 e와 n은 공개해놓고 있으므로, p와 q를 알면 누구나 나의 비밀키 d를 계산해낼 수 있다. 그러나 공개키로 부터 p,q를 알아내기가 어렵다 (n을 소인수 분해해야 하므로 시간이 오래 걸림. p, q가 큰 숫자 일수록 더욱 안전)
      - 400자리 정수 인수분해: 몇십억년 소요(2005년)

정수론의 응용

- 복호화 과정이 성립하는 이유
  - ed mod (p-1)(q-1) ≡ 1이면 ed = 1+k(p-1)(q-1)인 정수 k 존재
  - $C^d \equiv (M^e)^d = M^{ed} = M^{1+k(p-1)(q-1)} \pmod{n}$
  - gcd(M,p)=gcd(M,q)=1이 드문 예외를 제외하고 성립한다고 가정하면 Fermat의 작은 정리에 의해 $M^{p-1} \equiv 1 \pmod{p}$ 이고 $M^{q-1} \equiv 1 \pmod{q}$
    - $C^d \equiv M \times (M^{p-1})^{k(q-1)} \equiv M * 1 \equiv M \pmod{p}$
    - $C^d \equiv M \times (M^{q-1})^{k(p-1)} \equiv M * 1 \equiv M \pmod{q}$
    - $\therefore C^d \equiv M \pmod{pq}$
      - (예) C^d=17이면 , 17 ≡ 2 mod 3, 17 ≡2 mod 5 → 17 ≡ 2 mod 15

정수론의 응용

- (예) "STOP"을 p=43, q=59로 암호화
  - $n = p*q = 43*59 = 2537$
  - e=12
    - gcd (13, 2537)=1
  - "STOP"의 이진표현: 1819 1415
  - $C = M^{13} \bmod 2537$
    - $1819^{13} \bmod 2537 = 2081$, $1415^{13} \bmod 2537 = 2182$
    - 전송하는 메시지 C 는 2081 2182 임
- (예) 0981 0461을 수신하였을 때 복호화
  - d= 937: 13 mod 42*58의 역
  - $P = C^d \bmod n = C^{937} \bmod 2537$
    - $0982^{937} \bmod 2537 = 0704$
    - $0461^{937} \bmod 2537 = 1115$
  - 해독된 문자: 0704 1115 = "HELP"

정수론의 응용

- ed mod (p-1)(q-1) ≡ 1이면 ed = 1+k(p-1)(q-1)인 정수 k 존재
- $C^d \equiv (M^e)^d = M^{ed} = M^{1+k(p-1)(q-1)} \pmod{n}$
- gcd(M,p)=gcd(M,q)=1이 드문 예외를 제외하고 성립한다고 가정하면 Fermat의 작은 정리에 의해 $M^{p-1} \equiv 1 \pmod{p}$ 이고 $M^{q-1} \equiv 1 \pmod{q}$
  - $C^d \equiv M\ (M^{p-1})^{k(q-1)} \equiv M * 1 \equiv M \pmod{p}$
  - $C^d \equiv M\ (M^{q-1})^{k(p-1)} \equiv M * 1 \equiv M \pmod{q}$
- gcd(p,q)이므로 중국 나머지 정리에 의해
  - a1=M, M1=pq/p=q, M1y1 ≡ 1 (mod pq)
  - a2=M, M2=pq/q=p, M2y2 ≡ 1 (mod pq)
  - $C^d = Mq\overline{q} + M\ p\overline{p} \pmod{pq}$
  - 모듈로 합동의 따름정리 2에 의해
    - $Mq\bar{q} \pmod{pq} \equiv (M \pmod{pq} * q\bar{q} \pmod{pq}) \bmod pq = (M \pmod{pq} * 1) \bmod pq = (M \pmod{pq}) \bmod pq = M \bmod pq$

3.8 행렬(Matrices)

- 행렬(Matrix)의 정의
  - 여러 수 또는 문자를 사각형의 형태로 배열한 것
  - 행렬의 성분, 원소 : 배열한 숫자나 문자
    - 행(row) : 가로로 배열된 원소
    - 열(column) : 세로로 배열된 원소
    - m×n 행렬 : m개의 행과 n개의 열을 갖는 행렬
  - 정방 행렬(Square)
    - 같은 수의 행과 열을 갖는 행렬
  - 두 행렬의 행과 열의 숫자가 같고, 각 위치의 원소가 같으면 같은 행렬

행렬(Matrices)

- 행렬의 용어 정리

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

A의 i번째 행은 $1 \times \mathrm{n}$ 행렬 $[a_{i1} \quad a_{i2} \quad \cdots \quad a_{in}]$이다.

A의 j번째 열은 $\mathrm{n} \times 1$ 행렬 $\begin{bmatrix} a_{1j} \\ a_{2j} \\ \cdots \\ a_{nj} \end{bmatrix}$이다.

A의 (i, j)번째 원소는 원소 a_{ij}, 즉 i번째행과 j번째열에있는수

일반적인 행렬의 약기 표현 $\mathrm{A} = [a_{ij}]$

행렬(Matrices)

- 행렬의 연산
  - 덧셈(뺄셈)
    - $A=[a_{ij}]$, $B=[b_{ij}]$이며 크기가 $m \times n$이라고 하자.
    - 두 행렬의 덧셈 $A+B = [a_{ij}+b_{ij}]$
    - 주의 : 크기가 다른 행렬은 더할 수 없다.

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 2 & -3 \\ 3 & 4 & 0 \end{bmatrix} + \begin{bmatrix} 3 & 4 & -1 \\ 1 & -3 & 0 \\ -1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 4 & -2 \\ 3 & -1 & -3 \\ 2 & 5 & 2 \end{bmatrix}$$

행렬(Matrices)

- 곱셈
  - A는 m×k행렬이고, B는 k×n행렬이라고 하자.
  - 이때 두 행렬의 곱 AB = $[c_{ij}]$라면
  - $c_{ij} = a_{i1}b_{ij} + a_{i2}b_{2j} + \cdots + a_{ik}b_{kj}$
  - 행렬의 곱에서는 앞 행렬의 열의 수와 뒤 행렬의 행의 수가 같아야만 함.
  - 따라서 행렬 곱셈의 교환 법칙, 결합 법칙은 성립하지 않음

$$\begin{bmatrix} 1 & 0 & 4 \\ 2 & 1 & 1 \\ 3 & 1 & 0 \\ 0 & 2 & 2 \end{bmatrix} \times \begin{bmatrix} 2 & 4 \\ 1 & 1 \\ 3 & 0 \end{bmatrix} = \begin{bmatrix} 14 & 4 \\ 8 & 9 \\ 7 & 13 \\ 8 & 2 \end{bmatrix}$$

행렬(Matrices)

- (예)

$$A=\begin{bmatrix}1 & 1\\ 2 & 1\end{bmatrix},\quad \mathrm{B}=\begin{bmatrix}2 & 1\\ 1 & 1\end{bmatrix}\text{일 때}$$

$$\mathrm{AB}=\mathrm{BA}\text{인가 ?}$$

$$\mathrm{AB}=\begin{bmatrix}3 & 2\\ 5 & 3\end{bmatrix},\quad \mathrm{BA}=\begin{bmatrix}4 & 3\\ 3 & 2\end{bmatrix}$$

$$\therefore\ \mathrm{AB}\neq \mathrm{BA}$$

행렬(Matrices)

- 두 $n \times n$ 행렬의 곱에 사용되는 덧셈과 곱셈의 횟수는 ?
  - A와 B의 곱의 결과 행렬에는 n^2의 원소가 존재
  - 각 원소를 구하기 위해서는 n번의 곱셈과 n-1번의 덧셈 필요
  - $n^2 \times \{n$ 곱셈 + $(n-1)$번 덧셈$\}$ = n^3 곱셈 + $n^2(n-1)$ 덧셈

행렬(Matrices)

- n차 항등 행렬(Identity matrix of order n)
  - I_n 으로 표시, 행렬의 크기 : n×n
  - 행렬의 원소 : i=j이면 원소의 값이 1, 나머지는 0, 즉, 대각선만 1
  - 어떤 행렬에 항등 행렬을 곱해도 원래 행렬값을 갖는다.

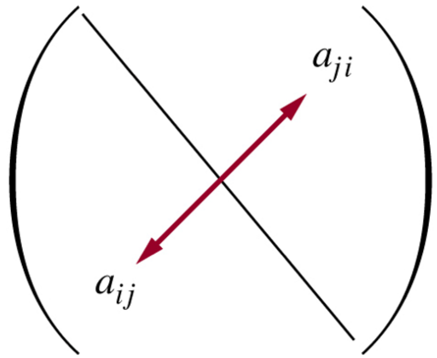
$$I_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

- 전치 행렬(Transpose)
  - 크기가 m×n인 행렬 A = $[a_{ij}]$가 있다고 하자.
  - 이 때 A의 행과 열을 서로 교환하여 얻어지는 크기 n×m 행렬 $[a_{ji}]$를 전치 행렬이라고 부르며 A^t로 표기한다.
    - (예)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \text{의 전치행렬은} \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \text{임}$$

행렬(Matrices)

- 대칭 행렬(Symmetric)
  - $A = A^t$인 행렬을 대칭 행렬이라고 함
    - 즉, $a_{ij}=a_{ji}$



- (예)

$$\text{행렬}\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}\text{은 대칭행렬임}$$

행렬(Matrices)

- 0-1 행렬(Zero-One Matrices)
  - 원소가 0 또는 1로 구성된 행렬
  - 부울 연산을 이용한 부울 산술에 사용

부울연산$(Boolean\ \ operation)$

$$b_1 \wedge b_2 = \begin{cases} 1 & if\ \ b_1 = b_2 = 1 \\ 0 & otherwise \end{cases}$$

$$b_1 \vee b_2 = \begin{cases} 1 & if\ \ b_1 = 1\ \ or\ \ b_2 = 1 \\ 0 & otherwise \end{cases}$$

- 0-1행렬의 부울 연산
  - A = $[a_{ij}]$, B=$[b_{ij}]$가 각각 m×n 0-1행렬이라고 하자.
  - A와 B의 join
    - A와 B의 결합의 (i, j)번째 원소는 $a_{ij} \vee b_{ij}$를 값으로 하는 0-1행렬
    - A ∨ B로 표기
  - A와 B의 meet
    - A와 B의 만남의 (i, j)번째 원소는 $a_{ij} \wedge b_{ij}$를 값으로 하는 0-1행렬
    - A ∧ B로 표기
  - A와 B의 부울 곱 : A⊙B
    - $c_{ij} = (a_{i1} \wedge b_{ij}) \vee (a_{i2} \wedge b_{2j}) \vee \cdots \vee (a_{ik} \wedge b_{kj})$

행렬(Matrices)

- (예) 0-1행렬의 결합과 만남

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathrm{B} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \text{일 때}$$

$$A \vee B = \begin{bmatrix} 1 \vee 0 & 0 \vee 1 & 1 \vee 0 \\ 0 \vee 1 & 1 \vee 1 & 0 \vee 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$A \wedge B = \begin{bmatrix} 1 \wedge 0 & 0 \wedge 1 & 1 \wedge 0 \\ 0 \wedge 1 & 1 \wedge 1 & 0 \wedge 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

행렬(Matrices)

- (예) A와 B의 부울 곱

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$A \Theta B = \begin{bmatrix} (1 \wedge 1) \vee (0 \wedge 0) & (1 \wedge 1) \vee (0 \wedge 1) & (1 \wedge 0) \vee (0 \wedge 1) \\ (0 \wedge 1) \vee (1 \wedge 0) & (0 \wedge 1) \vee (1 \wedge 1) & (0 \wedge 0) \vee (1 \wedge 1) \\ (1 \wedge 1) \vee (0 \wedge 0) & (1 \wedge 1) \vee (0 \wedge 1) & (1 \wedge 0) \vee (0 \wedge 1) \end{bmatrix}$$

$$= \begin{bmatrix} 1 \vee 0 & 1 \vee 0 & 0 \vee 0 \\ 0 \vee 0 & 0 \vee 1 & 0 \vee 1 \\ 1 \vee 0 & 1 \vee 0 & 0 \vee 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$