<자료구조 및 실습>

2. 자료구조와 알고리즘

한국외국어대학교
컴퓨터.전자시스템공학전공
2016년 1학기
고 석 훈

학습 목표

- 자료구조와 알고리즘의 중요성을 이해한다.
- 자료구조, 추상 자료형의 개념을 이해한다.
- 알고리즘의 개념과 성능 분석 방법을 이해한다.

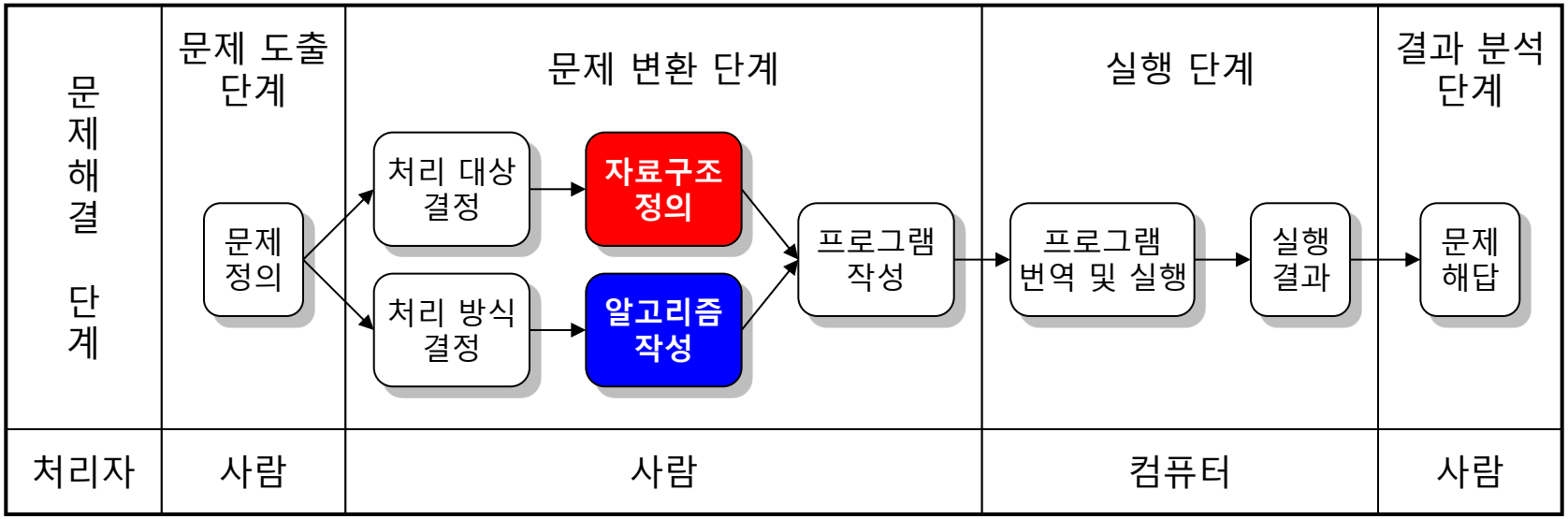
문제해결 방법

- 문제를 해결하는 방법
  - 독창적인 새로운 방법을 고안
  - 기존에 연구된 방법을 조합, 응용, 수정하여 재사용

- 훌륭한 프로그래머?
  - 대표적인 기존의 문제 해결 방법에 대한 이해를 바탕으로
  - 주어진 문제에 가장 효과적인 방법을 선택, 적용하는 능력 필요

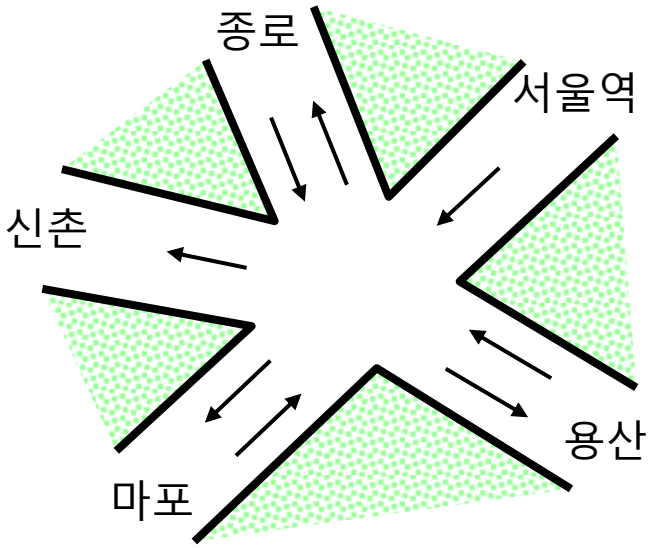
문제해결 과정

- 프로그램 = 자료구조 + 알고리즘
  - 효율적으로 문제를 처리하기 위해서는 문제를 분석하여 그에 대한 최적의 **자료구조(data structure)**를 정의하고, 최적의 **알고리즘(algorithm)**을 설계하는 능력이 중요



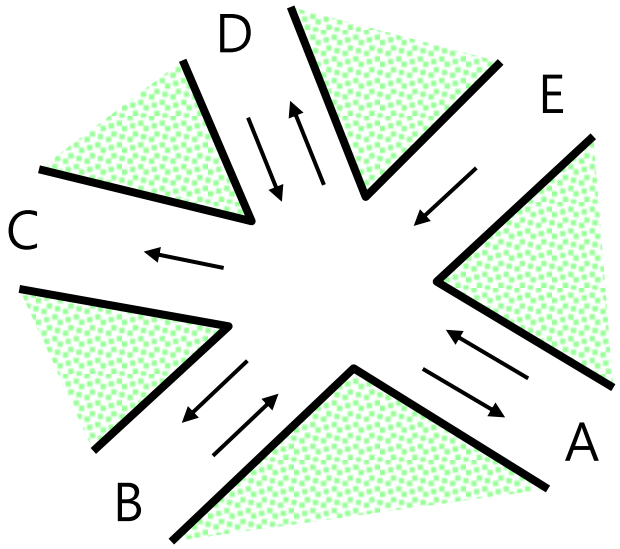
교차로 문제 [1/8]

- 교차로 문제
  - 어떤 교차로에서 그룹의 수를 최소로 하며,
    충돌 없이 동시에 진행 가능한 방향의 그룹을 구하는 문제



교차로 문제 [2/8]

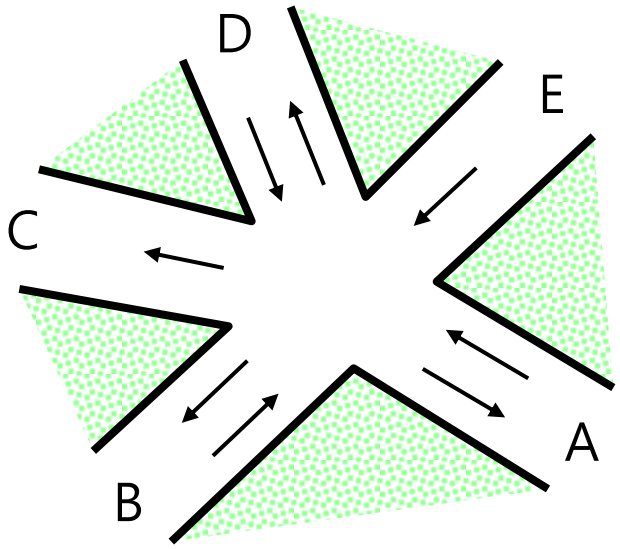
- 표(table)를 이용한 교차로 모델링 방법1
  - 가로열: 출발지, 세로열: 도착지, 진행 가능한 조합을 구분
  - 동시에 진행 가능한 방향을 구별하지 못함



From / To	A	B	C	D	E
A		1		1	1
B	1			1	1
C	1	1		1	1
D	1	1			1
E					

교차로 문제 [3/8]

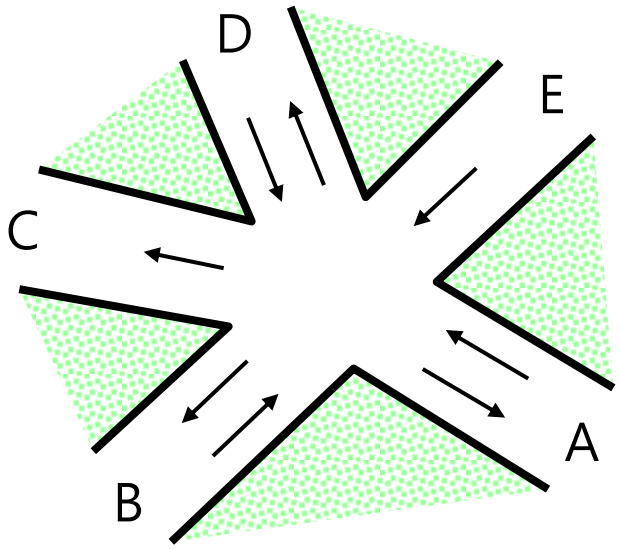
- 표(table)를 이용한 교차로 모델링 방법2
  - 가로열, 세로열: 출발, 도착지 기호를 붙여 진행방향 표시
  - 가로, 세로열의 진행방향이 서로 충돌하는 경우 구분

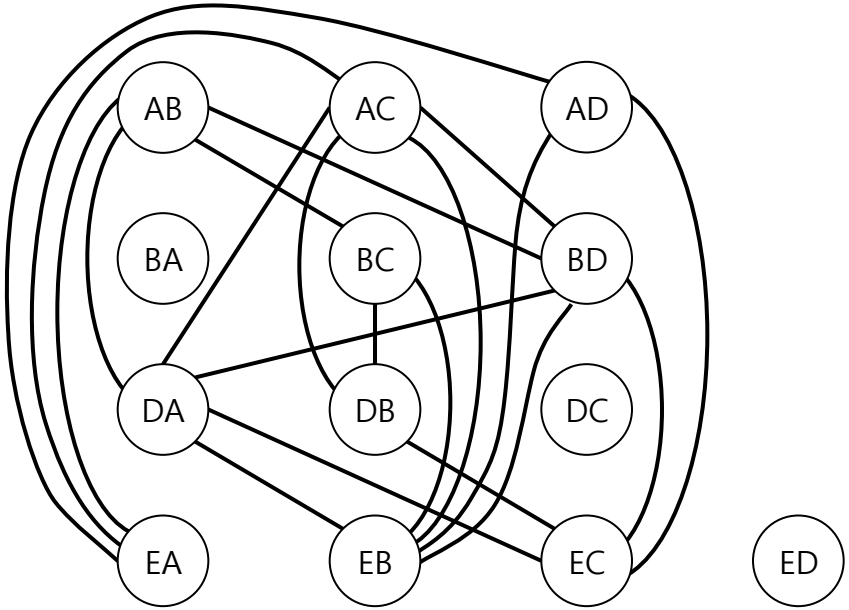


	AB	AC	AD	BA	BC	BD	DA	DB	DC	EA	EB	EC	ED
AB					1	1	1			1			
AC						1	1	1		1	1		
AD										1	1	1	
BA													
BC	1							1			1		
BD	1	1					1				1	1	
DA	1	1				1					1	1	
DB		1			1							1	
DC													
EA	1	1	1										
EB		1	1		1	1	1						
EC			1			1	1	1					
ED													

교차로 문제 [4/8]

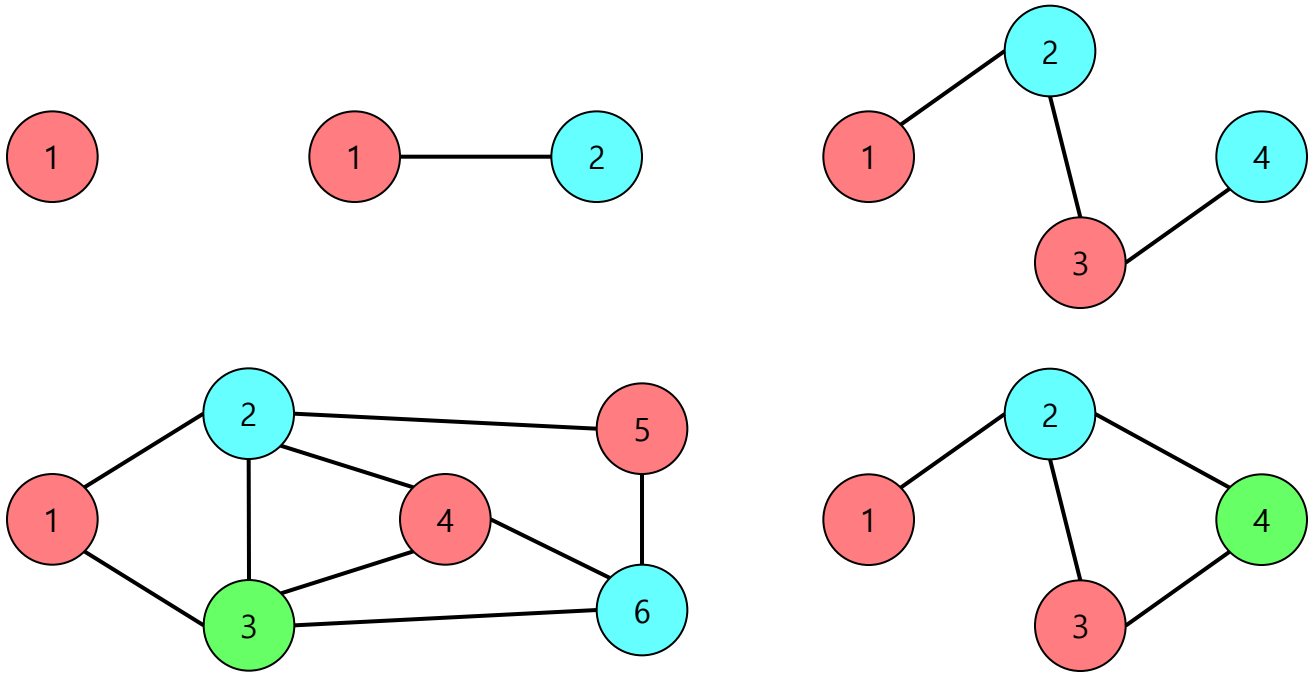
- 그래프(graph)를 이용한 교차로 모델링
  - 정점(vertex): 출발, 도착지 기호를 붙여 진행방향 표시
  - 간선(edge): 서로 충돌이 발생하는 진행방향을 연결





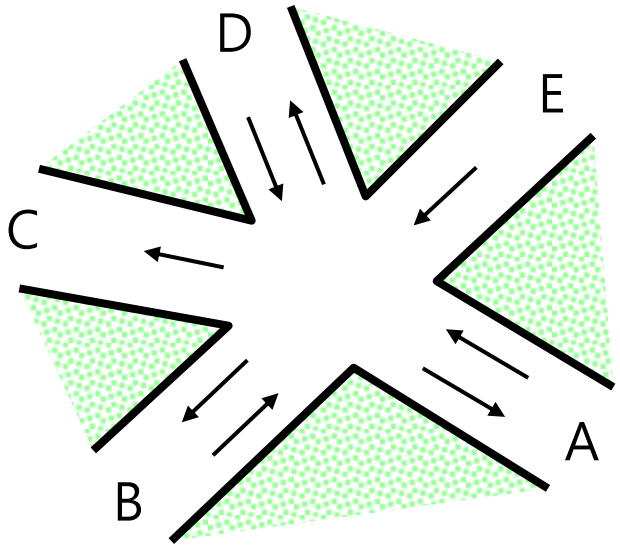
교차로 문제 [5/8]

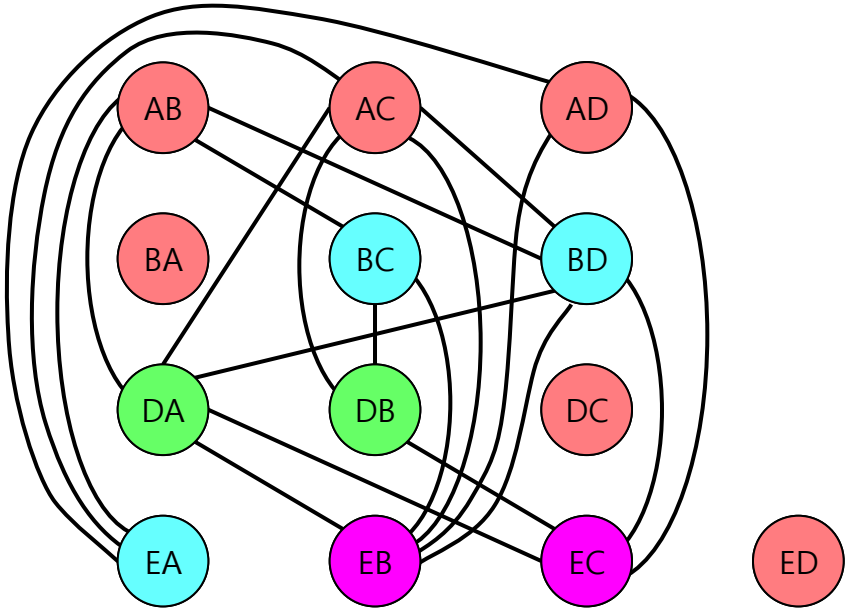
- 그래프 컬러링 문제(graph coloring problem)
  - 그래프에서 최소의 색을 사용하여 edge가 연결된 vertex에 서로 다른 색을 지정하는 문제



교차로 문제 [6/8]

- Greedy 알고리즘
  - Step 1: 색이 없는 하나의 노드를 골라 새로운 색을 지정한다.
  - Step 2: 색이 없는 노드에 대해, 새로운 색으로 지정된 노드와 연결된 에지가 없는 경우 새로운 색을 지정한다.





교차로 문제 [7/8]

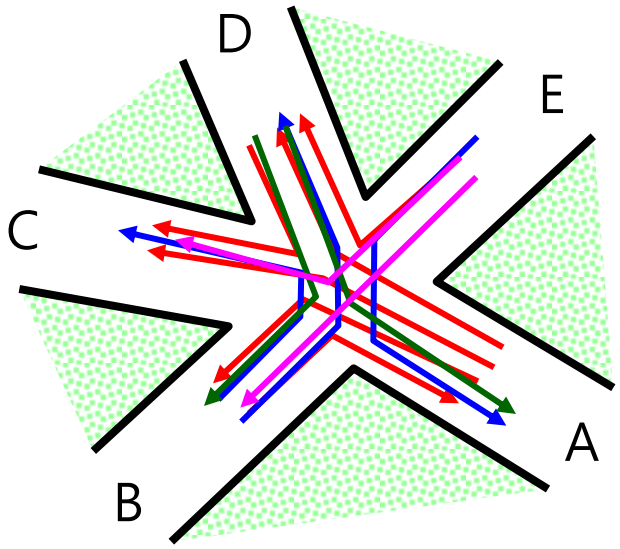
- 유사 코드 (pseudo-code)

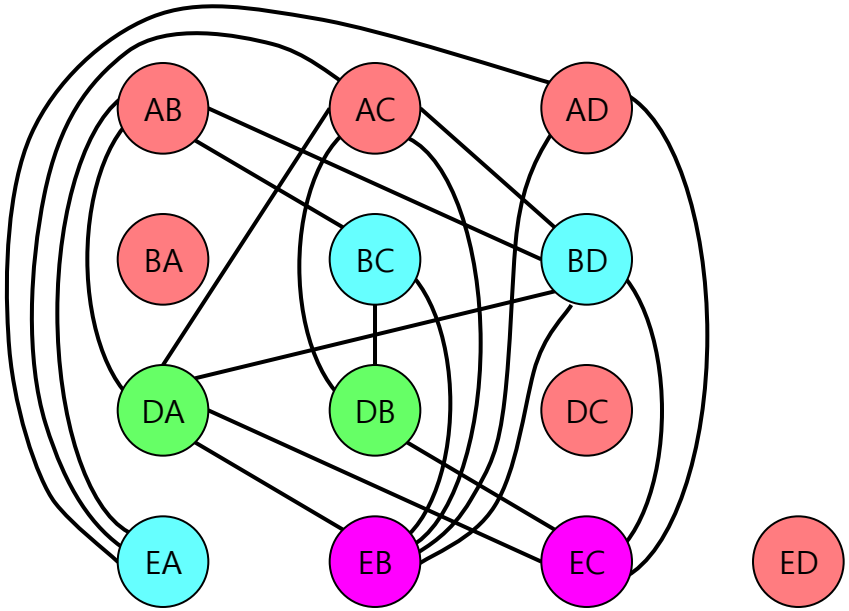
```
Procedure Greedy(graph G)

newColor ← set new color
v ← first uncolored vertex in G
while (v ≠ null) do begin
    found ← false
    w ← first vertex in newColor
    while (w ≠ null) do begin
        if (there exist edge(v, w) in G)
            found ← true
        w ← next vertex in newColor
    end
    if (found = false) then begin
        mark v colored
        add v to newColor
    end
    newColor ← set new color
    v ← next uncolored vertex in G
end
```

교차로 문제 [8/8]

- 결과 확인

{ AB, AC, AD, BA, DC, ED }, { BC, BD, EA }, { DA, DB }, { EB, EC }

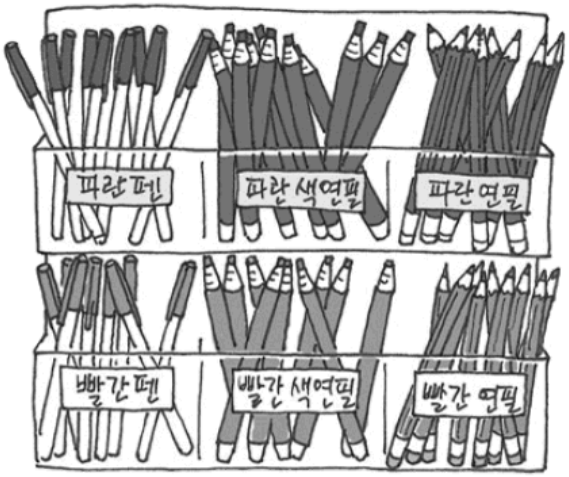




자료구조(Data Structure)

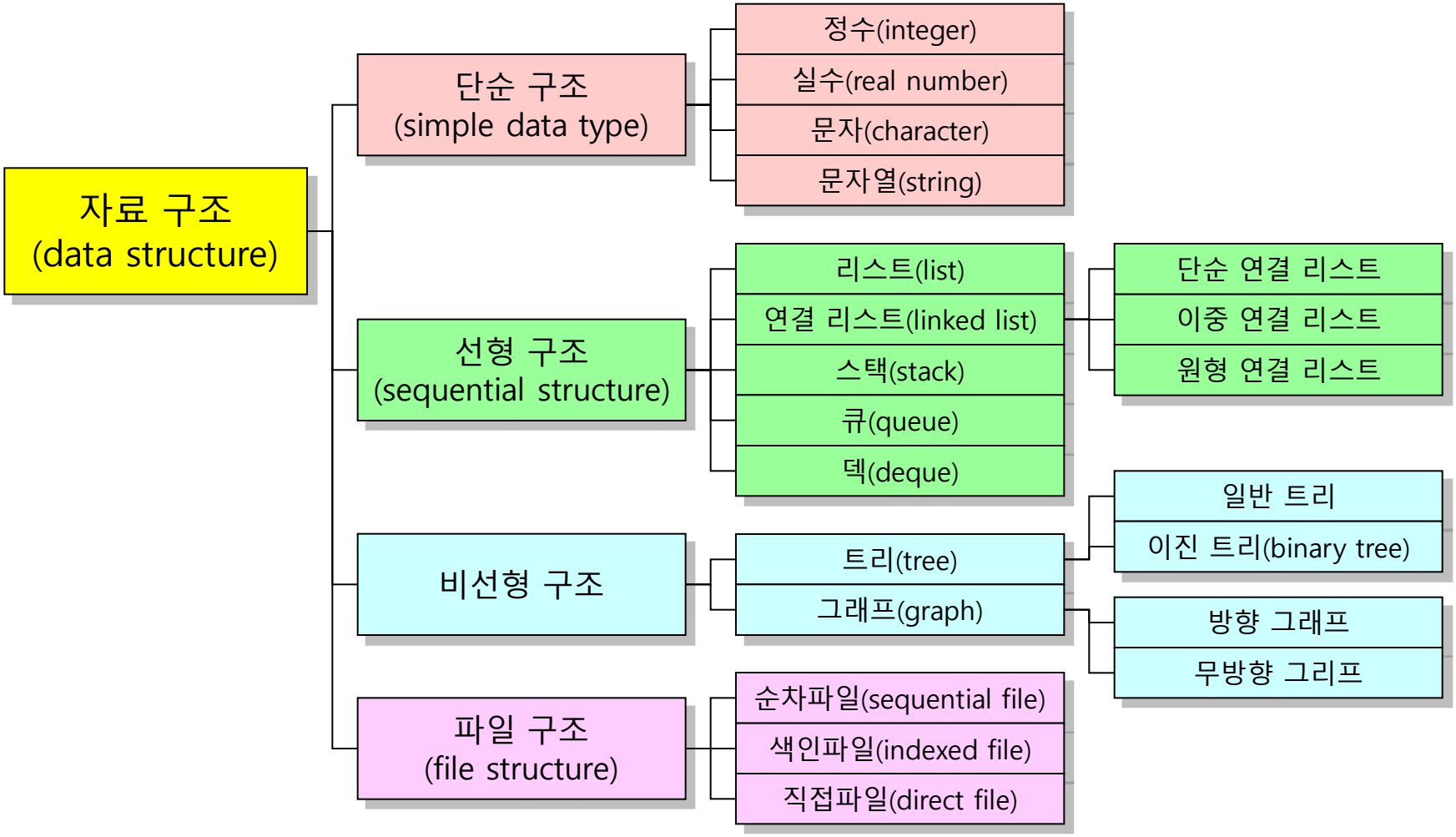
- 자료구조(data structure)
  - 컴퓨터에서 **자료(data)**를 효율적으로 처리할 수 있도록 표현하고 저장하는 **논리적인 구조(structure)**와 프로그램적인 처리 방법

나쁜 자료 구조



좋은 자료 구조

자료구조의 분류


자료 구조
(data structure)
단순 구조
(simple data type)
정수(integer)
실수(real number)
문자(character)
문자열(string)
선형 구조
(sequential structure)
리스트(list)
연결 리스트(linked list)
스택(stack)
큐(queue)
덱(deque)
단순 연결 리스트
이중 연결 리스트
원형 연결 리스트
비선형 구조
트리(tree)
그래프(graph)
일반 트리
이진 트리(binary tree)
방향 그래프
무방향 그리프
파일 구조
(file structure)
순차파일(sequential file)
색인파일(indexed file)
직접파일(direct file)

자료형(Data Type)

- 자료형(data type)
  - 자료형이 가질 수 있는 **값의 집합(data)**과
    그 값들에 적용 가능한 **연산들의 집합(operations)**으로 정의 할 수 있다.

- 예) int 자료형
  - 자료형 이름:
  - 값들의 집합:
  - 연산의 집합:

추상화(Abstraction) [1/2]

- 추상화(abstraction)
  - 객체의 본질을 이것이 작동되는 방법이나 구성되는 방법과 같은 세부적인 사항과 분리시키는 것을 말한다.

- 추상화의 특징
  - 어떻게(how)가 아니라 무엇(what)에 초점을 둔다.
  - 추상화를 통해 문제의 복잡도를 낮추면,
    복잡한 문제를 쉽게 해결할 수 있다.
  - 추상화 개념은 크고 복잡한 소프트웨어
    프로젝트를 관리하는데 필수적으로 사용된다.

추상화(Abstraction) [2/2]

- 제어 추상화(control abstraction)
  - 행위(action)의 논리적인 속성을 그 구현으로부터 분리

함수의 명세	구현
▪ 무엇을 하는 함수인가? ▪ 파라메터는 무엇인가?	▪ 함수의 실제 구성 ▪ 결과를 도출하는 알고리즘

- 자료 추상화(data abstraction)
  - 자료형의 논리적 속성을 상세한 구현으로부터 분리

논리적 속성	구현
▪ 가능한 값은 무엇인가? ▪ 필요한 연산은 무엇인가?	▪ C로 어떻게 구현되었는가? ▪ 자료구조를 어떻게 이용하는가?

추상 자료형(ADT, Abstract Data Type)

- 추상 자료형(ADT, abstract data type)
  - 자료 추상화(data abstraction)에 의해 상세한 구현으로부터 분리되어 논리적 속성으로 기술되는 자료형

- 예) Date 추상 자료형
  - 자료형 이름: Date
  - 값들의 집합: (year, month, day)
    year = 0~65535, month = 1~12, day = 1~31
  - 연산의 집합:

```
SetDate      // 날짜 지정
PrintDate    // 날짜 출력
GetDays      // 두 날짜 사이의 날수 계산
AddDate      // 두 날짜 더하기
SubDate      // 두 날짜 빼기
```

알고리즘(Algorithm)

- 알고리즘(algorithm)
  - 문제를 해결하기 위해 명령어의 단계적 절차 (sequence of instruction)를 기술해 놓은 명세서
- 예) 코끼리를 냉장고에 넣는 방법?

알고리즘의 조건

- 알고리즘의 5가지 조건
  - 입력(input)
  - 출력(output)
  - 명백성(definiteness)
  - 유효성(effectiveness)
  - 유한성(finiteness)

- 알고리즘과 프로그램의 차이
  - 유한성 : 유한하지 않은 프로그램 데몬(deamon)

알고리즘 기술 방법 [1/4]

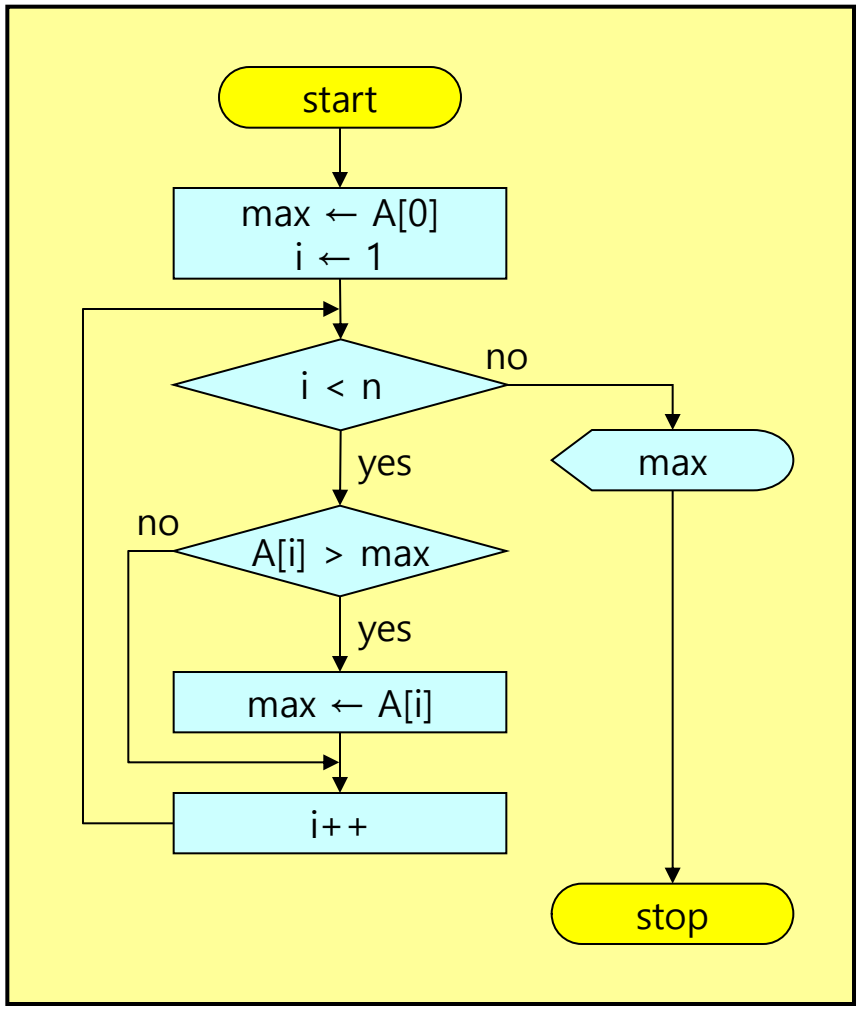
- 자연어로 표기된 알고리즘
  - 영어나 한국어 등 자연어로 알고리즘 기술
  - 인간이 읽기가 쉽다. 누구나 읽을 수 있다.
  - 의미 전달이 모호해질 우려가 있다.

- 예제) 배열에서 최대값 찾기

ArrayMax(A, n)
1. 배열 A의 첫 번째 요소를 변수 max에 복사
2. 배열 A의 다음 요소를 max와 비교하여 더 크면 max로 복사
3. 배열 A의 모든 요소를 비교했으면 max를 반환, 아니면 2번 반복

알고리즘 기술 방법 [2/4]

- 흐름도로 표기된 알고리즘
  - 직관적이고 이해하기 쉽다.
  - 규모가 커지면 복잡해 진다.



알고리즘 기술 방법 [3/4]

- C로 표현된 알고리즘
  - 알고리즘을 가장 정확하고, 상세하게 기술할 수 있다.
  - 반면, 구현에 관련된 많은 구체적인 사항들이 알고리즘의 핵심 내용의 이해를 방해할 수 있다.

```
#define MAX_ELEMENTS 100
int A[MAX_ELEMENTS];

int find_array_max(int n)
{
    int i, max;

    max = A[0];
    for (i = 1; i < n; i++) {
        if (A[i] > max) {
            max = A[i];
        }
    }
    return max;
}
```

알고리즘 기술 방법 [4/4]

- 유사코드(pseudo-code)로 표현된 알고리즘
  - 알고리즘 기술에 가장 많이 사용하는 고수준 기술 방법으로 자연어보다는 구조적이며, 프로그래밍 언어보다는 덜 구체적이다.
  - 프로그램을 구현할 때에 필요한 복잡한 문제들을 감출 수 있다. 즉, 알고리즘의 핵심적인 내용에만 집중할 수 있다.

```
ArrayMax(A,n)
max ← A[0];
for i ← 1 to n-1 do
    if max < A[i] then
        max ← A[i];
return max;
```

알고리즘의 성능 분석

- 공간 복잡도 (Space Complexity)
  - 알고리즘을 프로그램으로 실행하여 완료하기까지의 필요한 총 저장 공간의 양
  - 공간 복잡도 = 고정 공간 + 가변 공간

- 시간 복잡도 (Time Complexity)
  - 알고리즘을 프로그램으로 실행하여 완료하기까지의 총 소요시간
  - 시간 복잡도 = 컴파일 시간 + 실행 시간
    - 컴파일 시간: 프로그램마다 거의 고정적인 시간 소요
    - 실행 시간: 컴퓨터의 성능에 따라 달라질 수 있으므로 실제 실행시간 보다는 명령문의 실행 빈도수에 따라 계산

시간 복잡도 측정 방법 [1/3]

- C로 구현하여 측정
  - 알고리즘을 C로 구현하여 컴퓨터에서 실행하여 수행시간을 측정
  - C 코드의 상세한 부분을 모두 구현해야 한다.
- 수행시간 측정 방법
  - clock_t clock(void)로 프로세스 타임을 읽는다.
  - CLOCKS_PER_SEC 단위

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void main()
{
  clock_t start, finish;
  double  duration;

  start = clock();
  // 수행시간을 측정하고 하는 코드...
  finish = clock();
  duration = (double)(finish
           - start)/CLOCKS_PER_SEC;
  printf("%f초입니다.\n", duration);
}
```

시간 복잡도 측정 방법 [2/3]

- Pseudo 코드로 측정
  - 알고리즘을 pseudo 코드로 기술한 후, 각 입력에 대해 수행되는 기본 연산의 개수를 계산 (추상적으로 실행)
  - 연산의 개수는 고정된 값이 아니라, 입력 개수 n에 대한 연산의 수행 회수를 측정하여 **시간 복잡도 함수 T(n)**으로 표현
  - 알고리즘은 입력 데이터에 따라 연산 회수가 달라질 수 있는데, 시간 복잡도는 최악의 경우를 기준으로 한다.
    - 최선의 경우(best case): 의미가 없다.
    - 평균의 경우(average case): 계산하기 매우 어렵다.
    - 최악의 경우(worst case): 계산하기 쉽고, 중요한 의미를 갖는다.

시간 복잡도 측정 방법 [3/3]

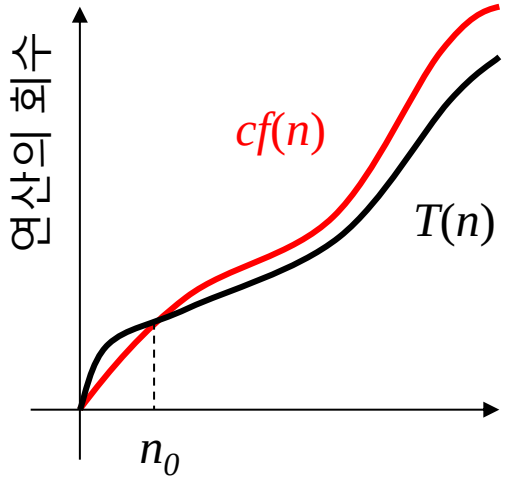
- 피보나치 수열의 시간 복잡도
  - 앞의 두수의 합이 다음 수가 되는 수열
    1 1 2 3 5 8 13 . . .

```
fibonacci(n)
01  if (n < 0) then
02      stop;
03  if (n ≤ 1) then
04      return n;
05  f1 ← 0;
06  f2 ← 1;
07  for (i ← 2; i ≤ n; i ← i + 1) do {
08      fn ← f1 + f2;
09      f1 ← f2;
10      f2 ← fn;
11  }
12  return fn;
```

행	n < 0	n ≤ 1	n > 1
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

빅-오(Big-Oh) 표기법

- 빅-오(Big-Oh)의 정의



> 모든 $n \geq n_0$에서 $T(n) \leq cf(n)$을 만족하는
> 상수 c와 n_0가 존재하면 $T(n)$은 $O(f(n))$이다.

  - 빅-오는 함수의 상한을 표시한다.
  - 빅-오(Big-Oh)를 구하는 방법
    - 실행시간 함수의 값에 가장 큰 영향을 주는 n에 대한 항에 대해
    - 계수는 생략하고 O(Big-Oh)의 오른쪽 괄호 안에 표시
  - 예) 피보나치 수열의 시간 복잡도
    - T(n) = 4n + 2
    -
    -

빅-오(Big-Oh) 종류 비교

$O(1)$: 상수형

$O(\log n)$: 로그형(logarithmic)

$O(n)$: 선형(linear)

$O(n\log n)$: 로그선형

$O(n^2)$: 2차형(quadratic)

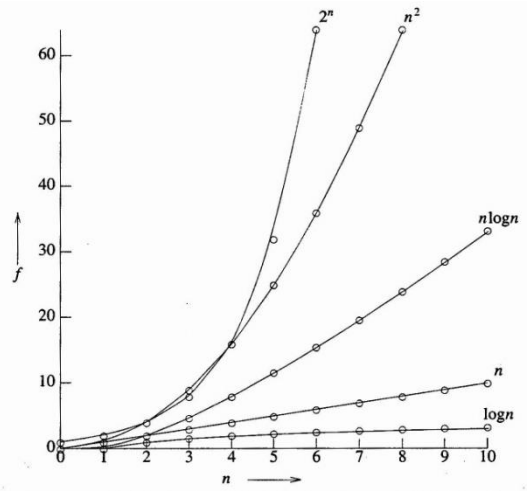
$O(n^3)$: 3차형

$O(n^k)$: k차형(polynomial) ***efficient***

$O(2^n)$: 지수형(exponential) ***inefficient***

$O(n!)$: 팩토리얼형

시간복잡도	n				
	1	2	4	8	16
1	1	1	1	1	1
$\log n$	0	1	2	3	4
n	1	2	4	8	16
$n \log n$	0	2	8	24	64
n^2	1	4	16	64	256
n^3	1	8	64	512	4096
2^n	2	4	16	256	65536
$n!$	1	2	24	40326	20922789888000



빅-오(Big-Oh) 예제 1

```
algorithmA(n)
01   a ← 0;
02   for (i ← 1; i ≤ n; i ← i + 1) do {
03       if (i%2 == 0)
04           a ← a + i;
05   }
06   return a;
```

행	연산
1	
2	
3	
4	
5	
6	

$$T(n) =$$

빅-오(Big-Oh) 예제 2

```
algorithmB(n)
01  x ← 0;
02  y ← 0;
03  for (i ← 1; i ≤ n; i++) do {
04      for (j ← 1; j ≤ i; j++) do {
05          x ← x + j;
06          y ← y + x;
07      }
08  }
09  return x;
```

행	연산
1	
2	
3	
4	
5	
6	
7	
8	
9	

$T(n) =$

빅-오(Big-Oh) 예제 3

$T(n) = 2n - 16$ =

$T(n) = 55n^2 + 10n + 5$ =

$T(n) = 6$ =

$T(n) = 3 \log n + 4 \log \log n$ =

$T(n) = 2n^3 + 4n^2 \log n$ =

$T(n) = 2^n + n^{17} - 7n^2$ =

- 빅-오의 정의에 의하면 $7n - 3 = O(n^3)$이라 할 수 있다. 맞는가?

요약

- 자료구조와 알고리즘의 중요성을 이해한다.
  - 프로그램 = 자료구조 + 알고리즘
  - 훌륭한 프로그래머는 문제에 가장 효과적인 방법을 선택, 적용한다.
- 자료구조, 추상 자료형의 개념을 이해한다.
  - **자료구조(data structure)**: 자료(data)를 효율적으로 처리할 수 있도록 표현하고 저장하는 논리적인 구조(structure)와 프로그램적인 처리 방법
  - **추상 자료형(ADT, abstract data type)**: 자료형의 논리적 속성을 상세한 구현으로부터 분리하여 논리적 속성으로 기술되는 자료형
- 알고리즘의 개념과 성능 분석 방법을 이해한다.
  - **알고리즘(algorithm)**: 문제를 해결하기 위해 명령어의 단계적 절차(sequence of instruction)를 기술해 놓은 명세서
  - 알고리즘의 시간 복잡도의 상한을 표현하는 **빅-오(Big-Oh) 표기법**