

# Presentation

2024-Fall. Computer Science Study

Data Structure

Algorithm

Operating System

Computer Network

Database

Computer Architecture

Design Pattern

Security

# Havruta



이 스터디의 목적은 컴퓨터 과학 관련 개념의 큰 그림을 그리는 것을 목적으로 한다.


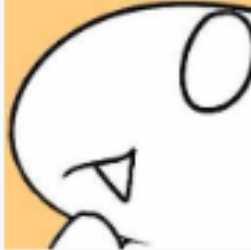


컴퓨터 과학(Computer Science)은 자료구조, 알고리즘, 운영체제, 데이터베이스, 네트워크를 포함하며, 넓게는 컴퓨터 구조, 디자인 패턴, 보안을 포함한다. 이러한 개념을 복습하고, 머리 속에서 다시 한번 정리하여 생각하는 방법을 기르려한다.

각자 자기가 공부해온 부분에 대해 설명한다.  
딱딱한 분위기가 아닌, 서로 틀릴 수 있음에도 자유롭게 의견을 이야기 할 수 있는 스터디를 진행한다.

# Introduction

---

## Member

| 스터디장  | 팀원  | 팀원   | 팀원  |
|---|---|--|---|
| <u><a href="#">양경식</a></u>  | <u><a href="#">김소명</a></u>  | <u><a href="#">김혜란</a></u>   | <u><a href="#">문강민</a></u>  |
|  |  |  |  |

**Schedule** : 수요일 21시 (NET 동아리실) - 80분 ; 실시간-대면

**Handout** : 제작 후 수업 전 공유

([https://github.com/NET-Organization/24-2-Computer\\_Science\\_Study](https://github.com/NET-Organization/24-2-Computer_Science_Study))

# Curriculum

---

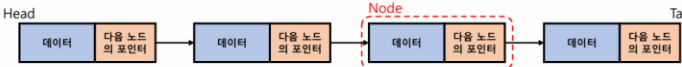
|                 | Theory                    | Solving               |
|-----------------|---------------------------|-----------------------|
| Week 1 (09/04)  | Orientation               |                       |
| Week 2 (09/11)  | Data Structure            |                       |
| Week 3 (09/25)  | Algorithm 1               | Data Structure        |
| Week 4 (10/02)  | Algorithm 2               | Algorithm 1           |
| Week 5 (10/16)  | Computer Architecture     | Algorithm 2           |
| Week 6 (11/06)  | Operating System          | Computer Architecture |
| Week 7 (11/13)  | Computer Network          | Operating System      |
| Week 8 (11/20)  | Database                  | Computer Network      |
| Week 9 (11/27)  | Design Pattern & Security | Database              |
| Week 10 (12/04) |                           | Real Problem          |

Homework : 매 수업 마지막에 다음 주제에 관해 미리 발표할 부분을 나누기 + 복습 문제 3개 (T/F 2 + 서술형 1)

- 노드라는 구조체 안에 데이터와 다음 노드의 address를 가지고 있는 자료구조
- 물리적인 메모리상에는 비연속적이거나 다음 노드를 가리키는 address가 있기때문에 논리적인 연속성을 가지고 있다고 할 수 있음

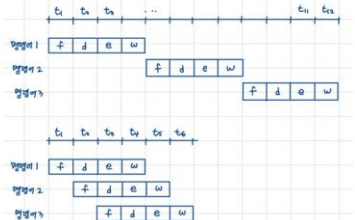
- 장점
- 데이터가 추가되는 시점에 메모리가 할당되기 때문에 메모리를 효율적으로 사용 가능
- 어느 원소를 추가, 삭제 하더라도 node에서 다음주소를 가리키는 부분만 다른 주소 값으로 변경하면 되기 때문에 shift할 필요 없어 시간 복잡도가  $O(1)$

- 단점
- 순차 접근(Sequential Access)만 가능 -> 특정 index의 데이터를 조회하기 위해  $O(n)$ 의 시간 걸림
- 다음 노드의 address를 가지고 있어야 하기 때문에 데이터 하나당 차지하는 메모리가 커짐



## Pipelining

- CPU의 성능을 향상시키기 위해 명령어 처리를 여러 단계로 나누어 동시에 실행하는 명령어 병렬 처리 기법



- Fetch : 주기억장치로부터 명령어를 읽어옵니다. 이때 프로그램 카운터의 값이 명령어의 메모리 주소를 가리키며, 이 값을 다음 명령어 주소로 업데이트
- Decode : 읽어들인 명령어를 해석하여 명령어와 필요한 데이터를 구분합니다. 이때, 명령어의 종류와 수행할 작업을 파악
- Execute : 해석된 명령어에 따라 필요한 연산을 수행합니다. 예를 들어 산술, 논리, 비트 연산 등 다양한 작업을 실행할 수 있습니다. 이 작업은 ALU에서 수행
- Write back : 실행 결과를 레지스터에 저장합니다. 이 과정에서 연산 결과 레지스터를 업데이트

### 애트리뷰트 / 필드 / 열

릴레이션에서 이름을 가진 열

### 도메인

한 애트리뷰트에 나타날 수 있는 값들의 집합

### 레코드/튜플/행

릴레이션 각 행

### 차수

릴레이션에 들어있는 애트리뷰트 수

### 카디날리티

릴레이션 튜플수

### 인덱스

DBMS에서 실제 데이터 레코드를 더 빠르게 찾아가기 위해서, 데이터 레코드를 참조하는 인덱스를 만들거

| EMPLOYEE |         | 릴레이션 이름 |     | 열 (애트리뷰트) |  |
|----------|---------|---------|-----|-----------|--|
|          |         | 기본 키    |     |           |  |
| EMPNO    | EMPNAME | TITLE   | DNO | SALARY    |  |
| 2106     | 김창섭     | 대리      | 2   | 2,000,000 |  |
| 3426     | 박영권     | 과장      | 3   | 2,500,000 |  |
| 3011     | 이수민     | 부장      | 1   | 3,000,000 |  |
| 1003     | 조민희     | 대리      | 1   | 2,000,000 |  |

행 (로우) →

← 카운터 (행의 수)

← (열들의 수)

| Index      |         | EMPLOYEE                       |
|------------|---------|--------------------------------|
| Empnoindex | Pointer | EMPNO EMPNAME TITLE DNO SALARY |
| 1003       | →       | 2106 김창섭 대리 2 2,000,000        |
| 2106       | →       | 3426 박영권 과장 3 2,500,000        |
| 3011       | →       | 3011 이수민 부장 1 3,000,000        |
| 3426       | →       | 1003 조민희 대리 1 2,000,000        |

| <inj까지 연산 최댓값> |   |    |    |    |    |
|----------------|---|----|----|----|----|
| j \ i          | 1 | 2  | 3  | 4  | 5  |
| 1              |   | 24 | 48 | 60 | 76 |
| 2              |   |    | 36 | 48 | 72 |
| 3              |   |    |    | 24 | 56 |
| 4              |   |    |    |    | 24 |
| 5              |   |    |    |    |    |

| <최댓값의 분할위치> |   |   |   |   |   |
|-------------|---|---|---|---|---|
| j \ i       | 1 | 2 | 3 | 4 | 5 |
| 1           |   | 1 | 2 | 1 | 4 |
| 2           |   |   | 2 | 2 | 4 |
| 3           |   |   |   | 3 | 4 |
| 4           |   |   |   |   | 4 |
| 5           |   |   |   |   |   |

### Process

#### Process address space

메모리 주소 공간은 운영체제에서 프로세스가 사용할 수 있는 메모리 공간을 의미합니다. 각 프로세스는 고유한 주소 공간을 가지며, 다른 프로세스의 메모리 공간에 접근할 수 없습니다.



**Stack** : 함수의 호출과 관계되는 지역 변수와 매개변수가 저장되는 영역으로, 함수의 호출과 함께 할당되고 함수의 호출이 종료되면 사라진다. 재귀함수를 너무 깊게 호출되어 stack의 영역을 초과하면 stack overflow 에러가 일어날 수 있다.

**Heap** : 동적 메모리 할당을 위해 사용되는 공간으로 런타임 동안 메모리를 할당/해제할 수 있다. 주로 참조형 데이터 (클래스, 배열, 문자열)가 할당된다.

**Data** : 전역 변수나 static 변수 등 프로그램이 사용할 수 있는 데이터를 저장하는 영역으로 프로그램의 시작과 함께 할당되며, 프로그램이 종료되면 소멸한다. 프로그램에서 전역/static 변수를 사용한다면 컴파일 후 data영역을 참조하게 된다.

**Text** : 프로그램이 실행될 수 있도록 cpu가 해석 가능한 기계어 코드가 저장되어 있는 공간이다.

### SQL injection

: 사용자의 입력을 그대로 SQL 질의에 넣어 발생하는 보안 취약점  
→ 데이터베이스를 조작하거나 민감한 정보 탈취

[SQL]  
**SELECT \* FROM users WHERE username = "user\_input" AND password = "password\_input";**

[Input]  
user\_input = ' or '1' = '1 # SQL에서 '는 주석이다. 뒷 문장이 다 주석처리가 되어 항상 참이 되도록 함.  
password\_input = {anything}

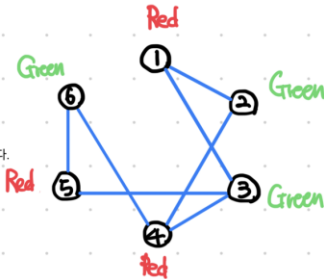
- **Classic Injection** : 위의 방식 대로 인증 우회
- **Blind SQL Injection** : 데이터를 직접 보지 못하는 상태에서 참/거짓에 따른 행동 차이 분석
- **Union-based Injection** : UNION 키워드를 추가해 데이터 노출 (UNION : 조건절을 추가하여 데이터베이스 정보도 출력하도록 함. **UNION SELECT \***)
- **Error-based Injection** : 오류 메시지를 통해 데이터베이스 정보를 출력

### N-coloring

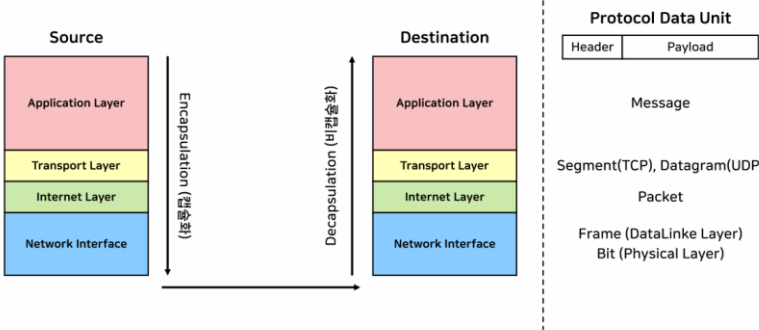
N-Coloring (지구본 색칠하기)

- 인접한 노드는 다른 색을 칠한다.

1. 매번 색칠할 때마다 제약 조건을 확인한다.
  - 제약 조건 : 인접한 노드에 같은 색이 없다.
  - 제약 조건에 위배되면 (Back Tracking) : 해당 경로 탐색 중단
  - 아니면 계속 색칠하기
2. 색을 모두 칠했으면 Solutions에 추가



## Data Transport



1 선착순 10만 명의 사용자에게 reward를 제공하는 시스템을 개발 중이다. reward를 제공하는 transaction은 중간에 취소될 수 있고, 그 경우 그 사용자는 선착순에서 밀려나도록 한다.

사용자가 밀려난 경우 후속(ex. 100,001번째) 사용자에게 reward를 제공한다. 이 시스템의 데이터베이스와 서버를 설계해보시오.

2 현재 단일 서버 환경에서 사용자에게 서비스를 제공하고 있다. 다양한 사용자에게 제공하기 위해서는 어떠한 방식을 취할 수 있는가?

3 Stack 2개를 활용하여 Queue를 구현해보시오. (실제 코딩할 필요는 없고, Queue의 특징을 어떻게 Stack으로 구현할 수 있는지 서술)

“ Think big, and you will see a bigger world. ”