유튜브 강의와 함께하는

단기간에 **핵심**만!

기초 자료구조특강

유튜브 강의 1강 - 10강

- 아답터(민 기술사)

BOOKK

단기간에 **핵심**만!

기초 자료구조특강

유튜브 강의 1강 - 10강

단기간에 **핵심**만!

기초 자료구조 특강

아답터
IT의 답을 터득하다

1강. 자료구조를 배워야 하는 이유와 개념

개발자, 데이터 사이언티스트, IT 수험생
이면 반드시 공부해야 하는 자료구조

대학생, 직장인, 수험생을 위한

필요 내용만 간략히 배우는 자료구조

자료구조를 배워야 하는 이유

아답터
IT의 답을 터득하다

자료구조란? 컴퓨터에서 처리할 데이터를 효율적으로 관리하고 구조화시키기 위한 방법

- 데이터를 체계적으로 저장하고 효율적으로 활용하기 위해
- 특정한 상황에서의 문제를 해결할 때, 상황에 적합한 자료구조를 활용
- 자료구조의 개념은 시간이 지나도 변하지 않음

개발자

프로그램의
실행 시간과
메모리 관리

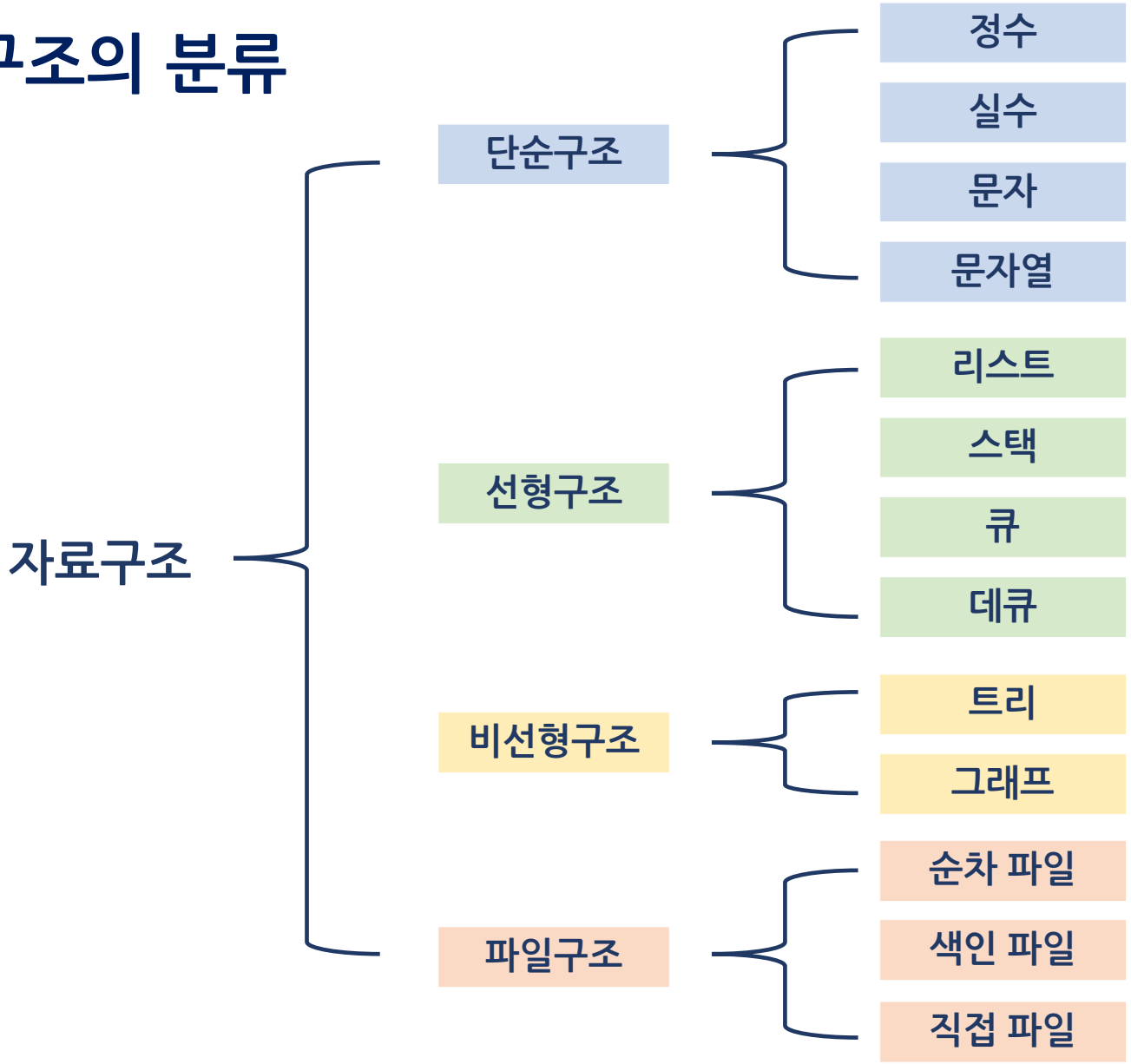
데이터 사이언티스트

대규모 데이터에 대한
효율적인 전처리, 학습, 분석

IT 수험생

컴퓨터 일반,
알고리즘,
프로그래밍론의
기초적 학문

자료구조의 분류



자료구조의 이용

정렬

기억장치에 저장된 데이터를 일정한 순서로 나열하는 것

탐색

기억장치에서 데이터를 찾는 것

파일 편성

데이터를 기억 장치에 저장할 때의 파일 구조

인덱스

특정 데이터를 빠르게 찾기 위한 것

단기간에 **핵심**만!

기초 자료구조 특강

아답터
IT의 답을 터득하다

2강. 알고리즘 성능분석

- 자료구조와 알고리즘의 관계
- 빅오(Big-O) 표기법

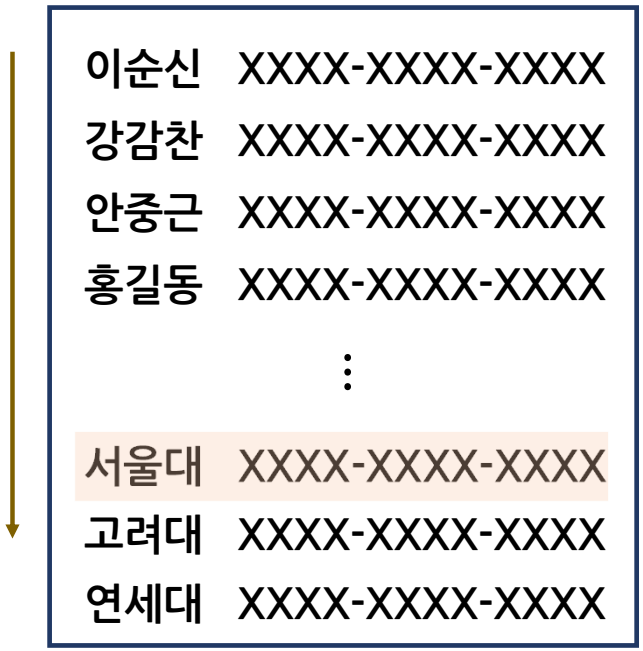
대학생, 직장인, 수험생을 위한

필요 내용만 간략히 배우는 자료구조

자료구조와 알고리즘의 관계

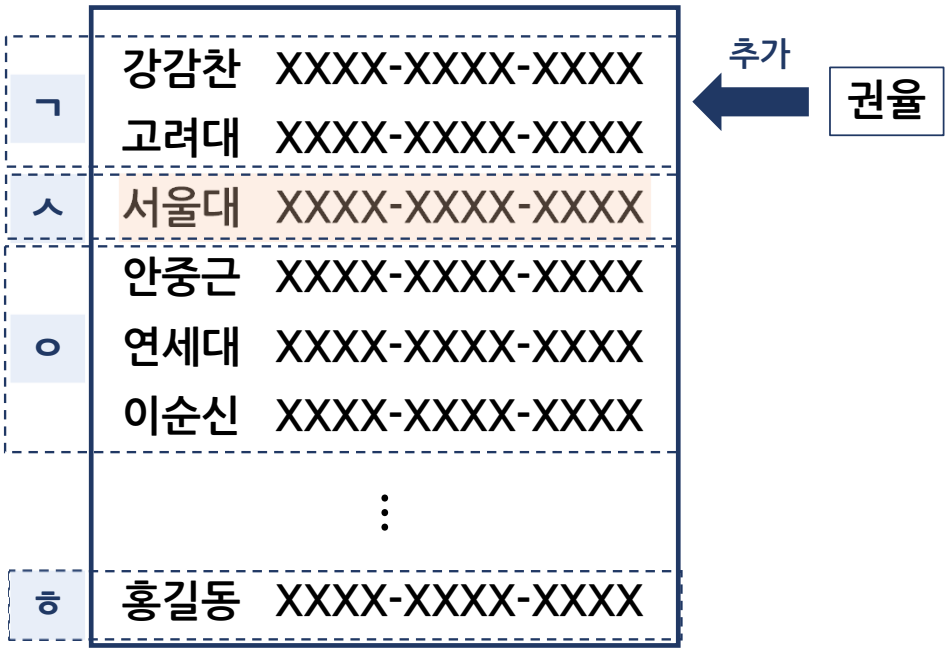
친구 추가 기능

친구를 추가하는 순서대로 저장



구현은 간단하지만
탐색에 시간이 많이 소요

카테고리별로 저장



탐색은 빠르지만
신규 데이터 추가 시 비효율적

자료구조와 알고리즘의 관계

아답터
IT의 답을 터득하다

친구 추가 기능

자료구조에 따라 알고리즘이 달라지며, 자료구조와 알고리즘은 서로 의존적인 관계

미리 카테고리 별 여유 공간을 확보

ㄱ	강감찬	XXXX-XXXX-XXXX
	고려대	XXXX-XXXX-XXXX
	권율	XXXX-XXXX-XXXX
ㅅ	서울대	XXXX-XXXX-XXXX
ㅇ	안중근	XXXX-XXXX-XXXX
	연세대	XXXX-XXXX-XXXX
	이순신	XXXX-XXXX-XXXX
ㅎ	홍길동	XXXX-XXXX-XXXX

비효율적인 메모리 관리

시간복잡도와 공간복잡도

좋은 알고리즘을 평가하는 요소는?

· 속도

시간복잡도 (Time Complexity)

· 메모리 사용량

공간복잡도 (Space Complexity)

일반적으로 메모리 사용량 보다 실행속도에 관심

어떻게?

일반적으로 데이터 수 n에 대하여 덧셈, 뺄셈, 곱셈 등의 횟수를 시간복잡도로 표현

- ·빅오(Big-O) 최악의 경우에 대한 성능을 판단
- ·빅오메가(Big-Ω) 최선의 경우에 대한 성능을 판단
- ·빅세타(Big-θ) 평균의 경우에 대한 성능을 판단

빅오(Big-O) 표기법

선형리스트(배열)에서 숫자 21을 찾아보자

순차탐색인 경우

최악의 경우 n번을 탐색해야 됨

$O(n)$

이진탐색인 경우

첫번째 탐색 범위

1/2

두번째 탐색 범위

1/2

세번째 탐색 범위

n이 1이 될 때까지 2로 k회 나눔

$\rightarrow n \times (\frac{1}{2})^k = 1$

$\rightarrow k = \log_2 n$

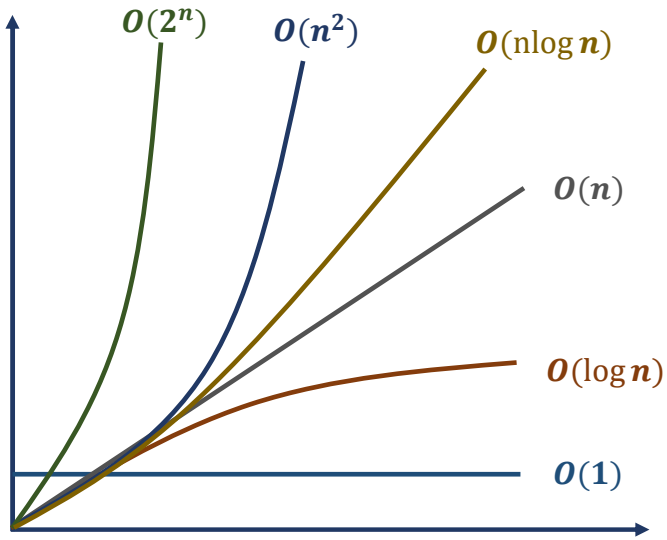
$\therefore O(\log_2 n)$

빅오(Big-O) 표기법

두 함수 f(n)과 g(n)이 주어졌을 때, 모든 n ≥ K에 대하여 f(n) ≤ Cg(n)을 만족하는 C와 K가 존재하면, f(n)의 빅오는 O(g(n))이다.

빅오 표기법 특징

- **상수항 무시** 5n → O(n)
  - 알고리즘의 효율성은 데이터의 개수가 충분히 크다는 가정 하에 사소한 상수항은 무시
- **영향력 없는 항 무시** n + 3 → O(n)
  - 데이터 개수가 충분히 크다면 가장 영향력이 큰 항 외에 다른 항들은 무시 가능



Faster — Slower

$$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(2^n)$$

일반적인 알고리즘 성능 | 부적절

- $O(1)$: 데이터 수의 관계없이 연산 횟수가 고정 — 스택의 Push, Pop
- $O(\log n)$: 데이터 수 증가율에 비해 연산횟수 증가율이 낮음 — 이진탐색
- $O(n)$: 데이터 수와 연산횟수가 비례 — 일반적인 For문
- $O(n\log n)$: 데이터수가 많아질수록 연산횟수가 조금 더 증가

  병합정렬, 힙정렬, 퀵정렬

빅오(Big-O) 표기법

아답터
IT의 답을 터득하다

아래 식들을 빅오로 표현해 보자

(1) $5n + 3$

(2) $3n^3 + 6n^2 + 1$

(3) $2^n + 3n^2$

(4) $2^n + 4n^3$

(5) $n + \log n$

(6) $n + n\log n$

(1) $O(n)$ (2) $O(n^3)$ (3) $O(2^n)$ (4) $O(2^n)$ (5) $O(n)$ (6) $O(n\log n)$

단기간에 **핵심**만!

기초 자료구조 특강

아답터
IT의 답을 터득하다

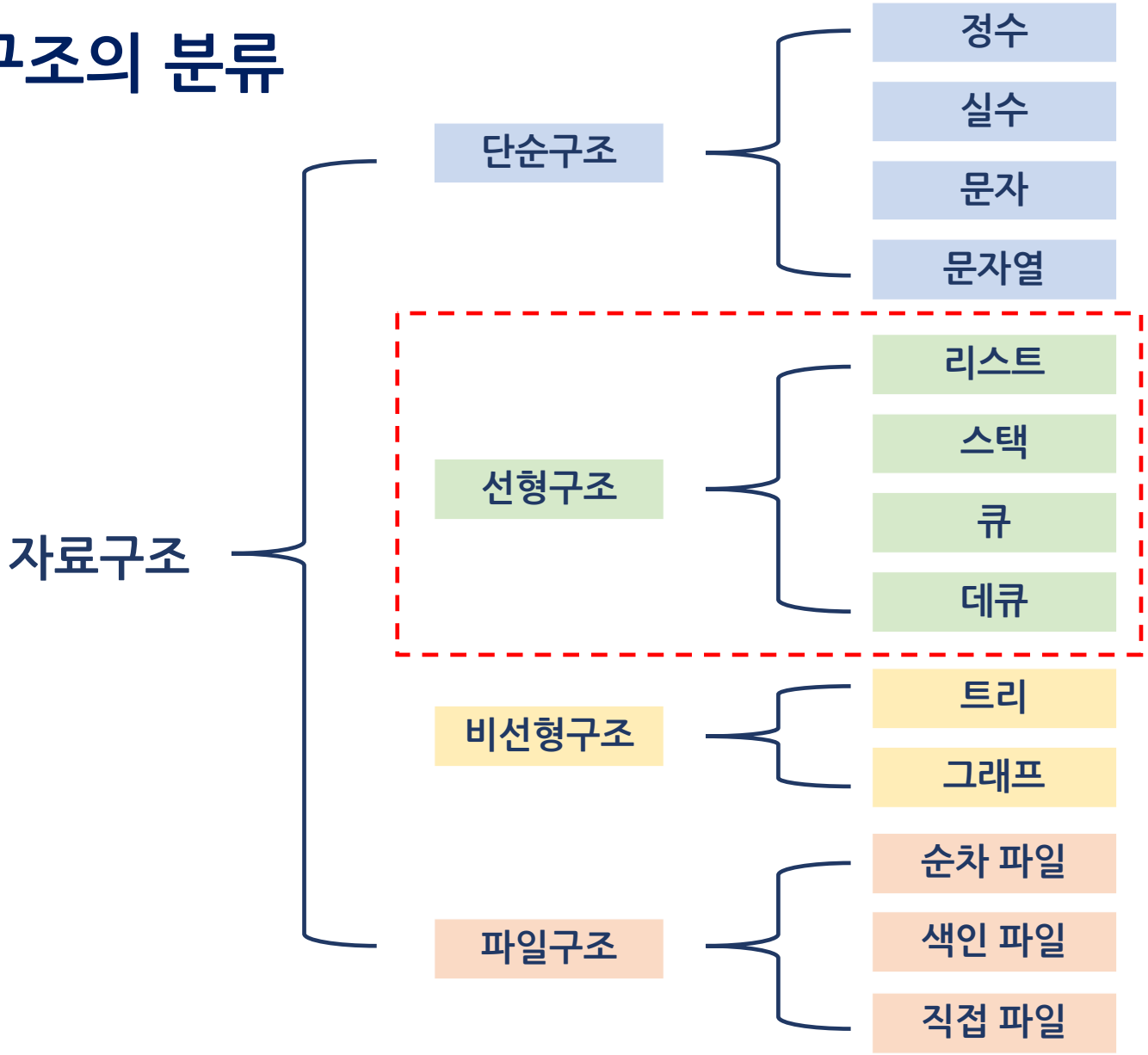
3강. 선형 자료구조

- 리스트(List)
- 스택(Stack)
- 큐(Queue)와 데큐(DeQue)

대학생, 직장인, 수험생을 위한

필요 내용만 간략히 배우는 자료구조

자료구조의 분류

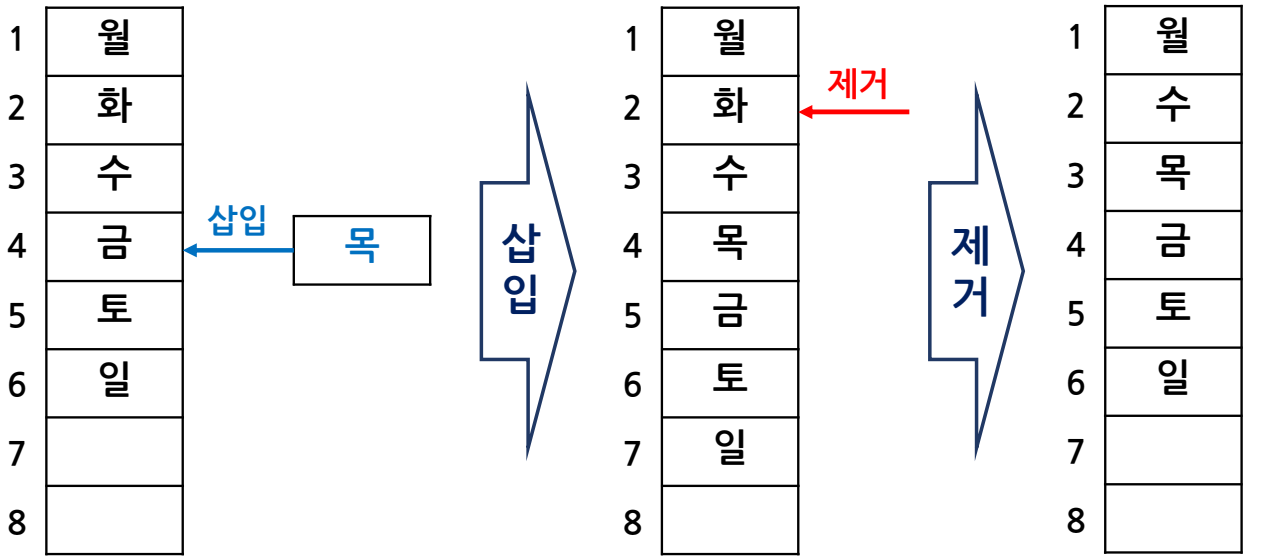


리스트(List)

선형 리스트(Linear List)

연결 리스트(Linked List)

연속되는 기억장치에 저장되는 자료구조 (=배열)



특징

- 가장 간단한 자료구조
- 빠른 접근속도
- 효율적인 메모리 공간
- 삽입, 삭제 시 데이터 이동으로 인한 번거로운 작업 필요

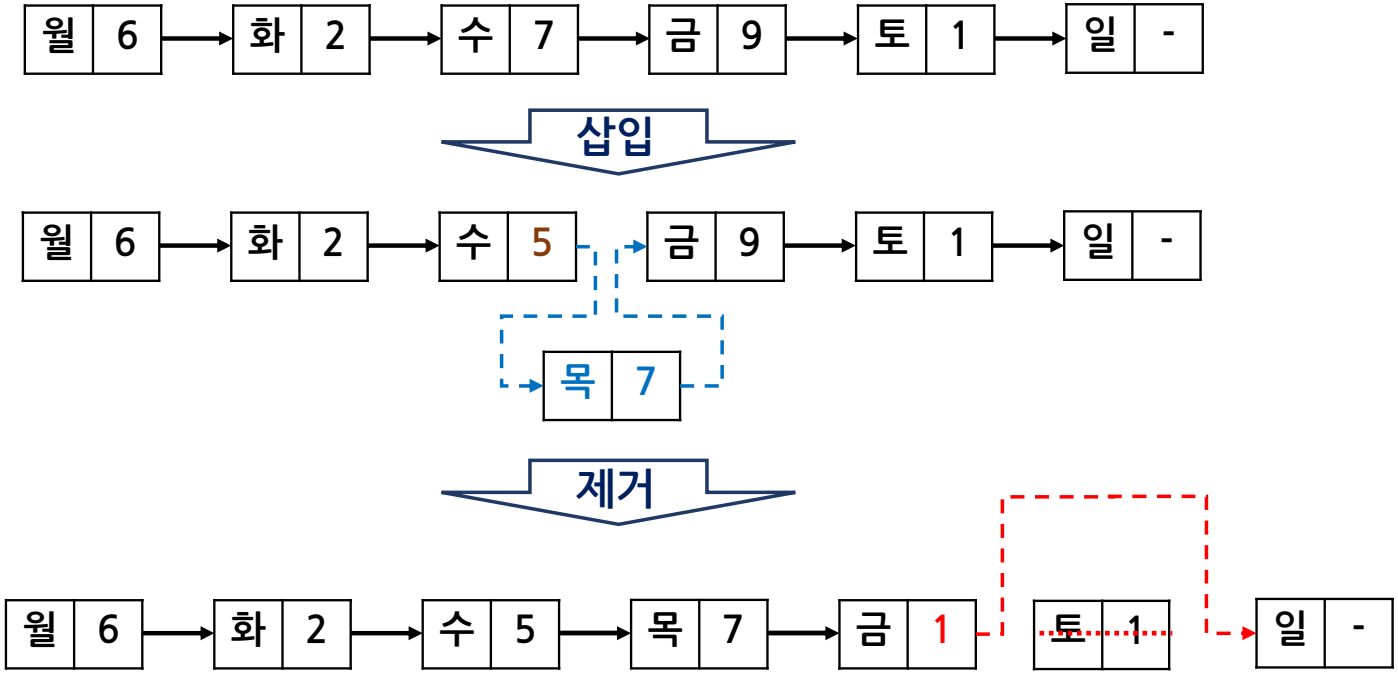
리스트(List)

선형 리스트(Linear List)

연결 리스트(Linked List)

자료를 임의의 공간에 기억시키고, 순서에 따라 포인터로 연결시킨 자료구조

* 포인터 : 기억장치의 주소를 가르키는 변수



리스트(List)

선형 리스트(Linear List)

연결 리스트(Linked List)

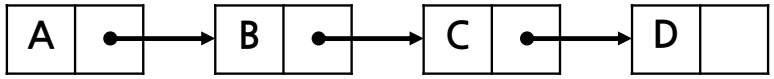
자료를 임의의 공간에 기억시키고, 순서에 따라 포인터로 연결시킨 자료구조

특징

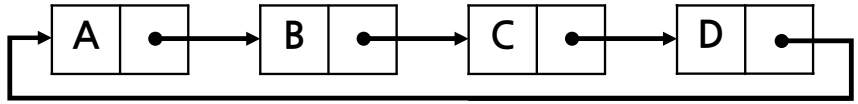
- 노드의 삽입, 제거가 용이
- 기억공간이 연속적일 필요 없음
- 포인터 부분이 필요하여 선형리스트 대비 낮은 메모리 이용 효율
- 연결을 통해 순차적으로 접근해야 하므로 느린 접근 속도

종류

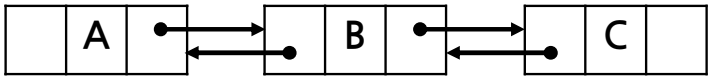
- 단순 연결 리스트(Singly Linked List)



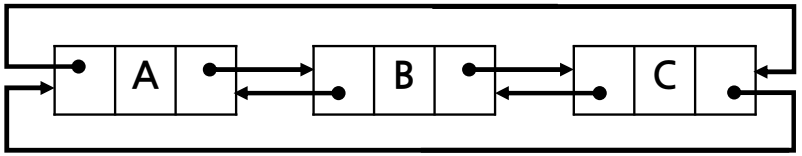
- 단순 원형 연결 리스트(Singly Circular Linked List)



- 이중 연결 리스트(Doubly Linked List)

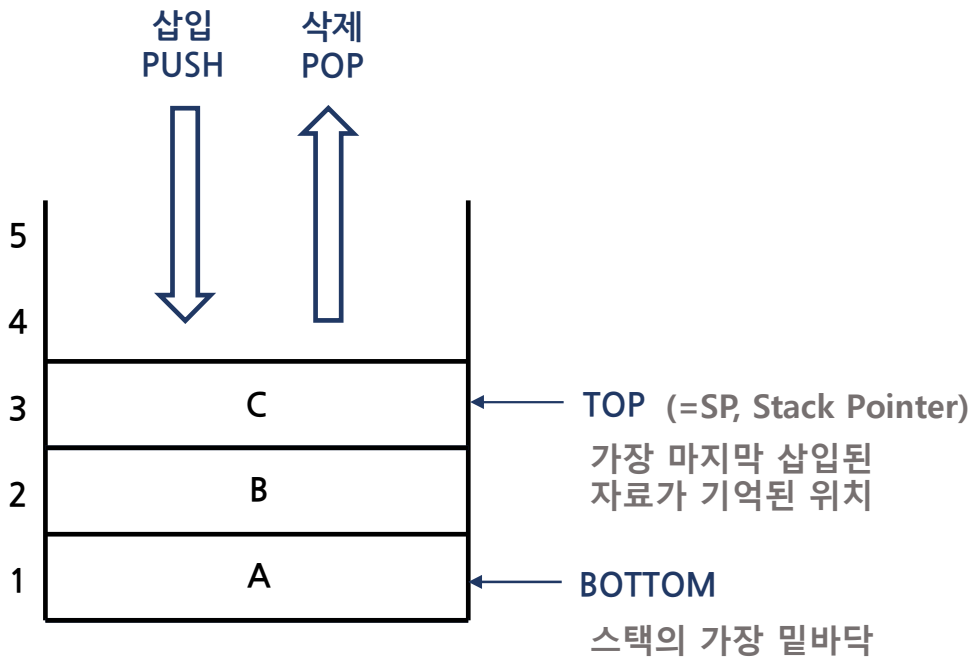


- 이중 원형 연결 리스트(Doubly Circular Linked List)



스택(Stack)

리스트의 한쪽 끝으로만 자료의 삽입, 삭제가 이루어지는 자료구조



특징

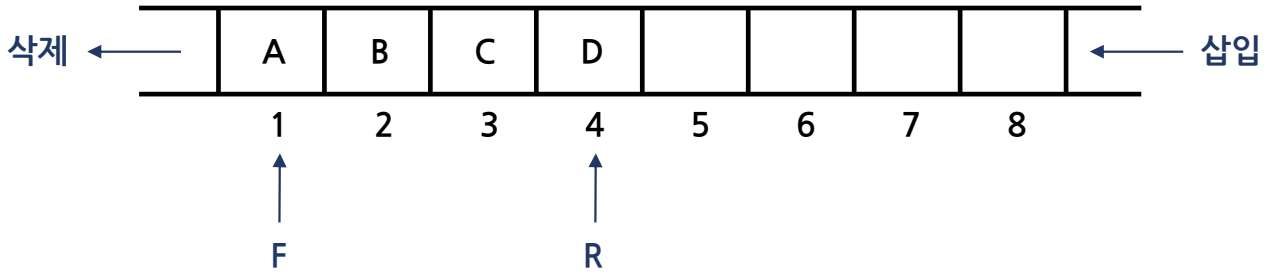
- 가장 나중에 입력된 데이터가 먼저 삭제되는 후입선출(LIFO:Last In First Out) 구조
- 스택에 모든 메모리가 채워져 있는 상태에서 자료를 삽입하려고 하면 Overflow가 발생
- 스택에 모든 메모리가 비워져 있는 상태에서 자료를 제거하려고 하면 Underflow가 발생

활용 사례

- 웹 브라우저 방문기록 뒤로가기
- 프로그램의 실행 취소 (Undo)
- 인터럽트가 발생하여 복귀주소 저장 시
- 후위 표기법으로 수식을 표현
- 함수 호출 순서 제어
- 깊이 우선 탐색(DFS) 구현

큐(Queue)

한쪽에서 데이터의 삽입이 이루어지고 다른 한쪽에서 데이터의 삭제가 이루어지는 자료구조



특징

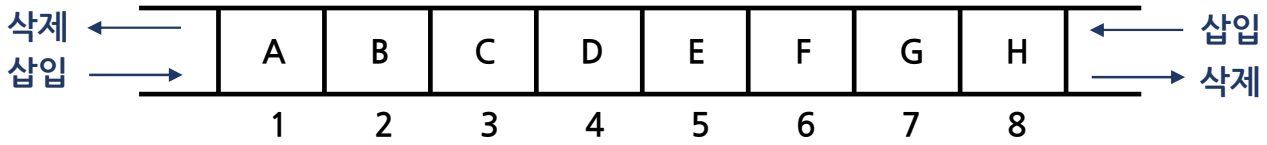
- 가장 먼저 입력된 데이터가 먼저 삭제되는 선입선출(FIFO:First In First Out) 구조
- 프런트(F, Front) 포인터 : 가장 먼저 입력된 데이터의 기억공간을 가리키며, 삭제 작업을 할 때 사용
- 리어(R, Rear) 포인터 : 가장 마지막에 입력된 데이터의 기억공간을 가리키며, 삽입 작업을 할 때 사용

활용 사례

- 우선순위가 같은 작업 예약(프린터 인쇄)
- 은행 창구 업무의 서비스 순서 대기 처리
- 콜센터 고객 대기시간
- 운영 체제의 작업 스케줄링
- 너비 우선 탐색(BFS) 구현

데큐(DeQue)

삽입과 삭제가 양쪽 끝에서 모두 발생할 수 있는 자료구조



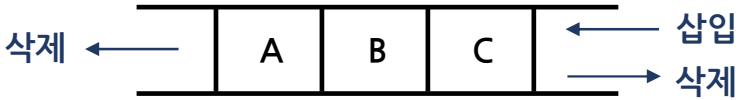
특징

- Double Ended Queue의 약자
- Stack과 Queue의 장점만을 활용하여 구성

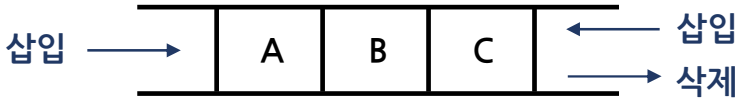
활용 사례

- 우선순위를 조절하는 스케줄링
- 문서 편집의 Undo/Redo

입력제한데큐 (Scroll)



출력제한데큐 (Shelf)



단기간에 **핵심**만!

기초 자료구조 특강

아답터
IT의 답을 터득하다

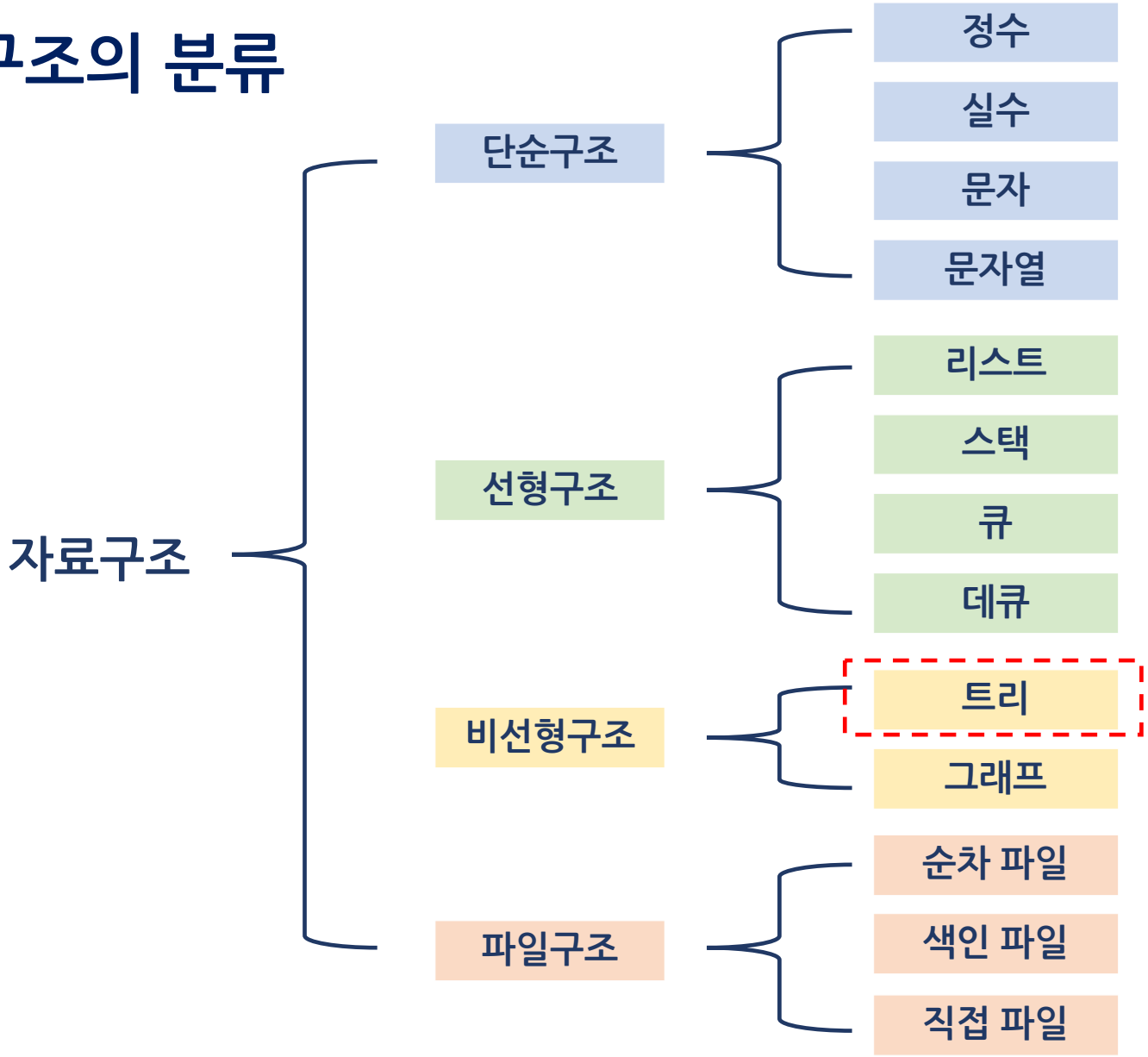
4강. 비선형 자료구조 - 트리

- 트리의 개념
- 이진 트리의 순회
- 이진 수식 트리

대학생, 직장인, 수험생을 위한

필요 내용만 간략히 배우는 자료구조

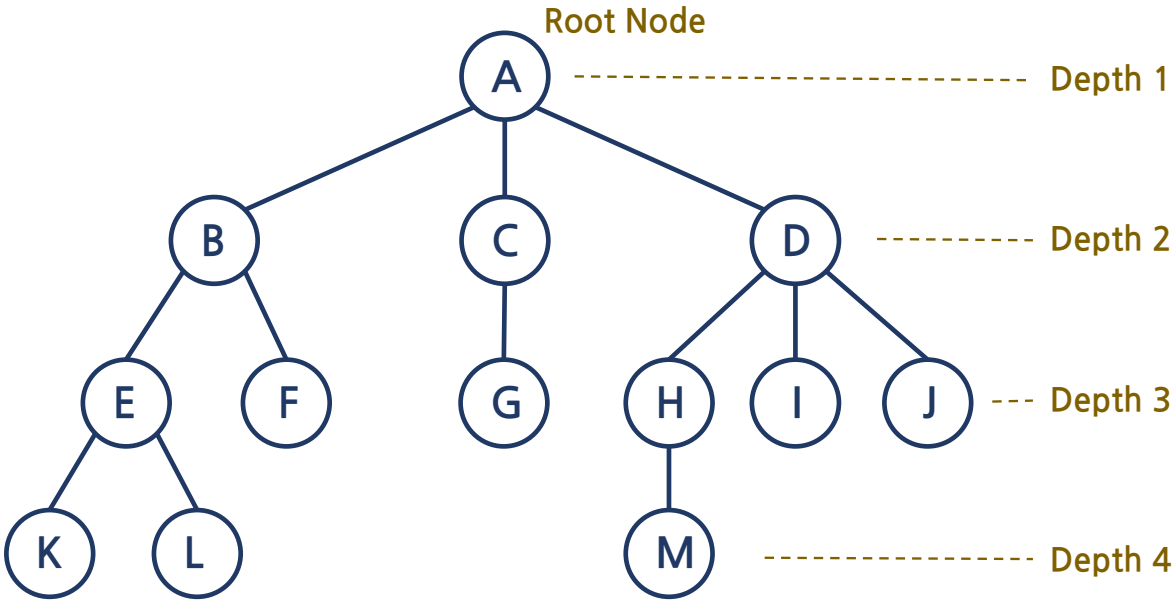
자료구조의 분류



트리(Tree)

노드들이 나무 가지처럼 연결된 계층적 자료구조

가족 계보, 연산 수식, 회사 조직 구조도, Heap 표현하기 적합

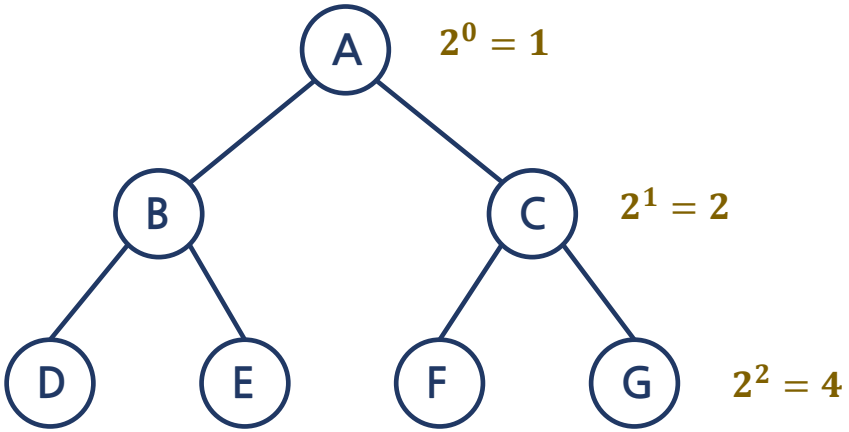


- 노드(Node)
  : 자료 항목을 가지는 트리를 구성하는 기본요소
  A, B, C, D, E, F, G, H, I, J, K, L, M
- 근 노드(Root Node)
  : 트리의 가장 맨 상단에 있는 노드
  A
- 차수(Degree)
  : 각 노드에서 뻗어나온 가지 수
  A=3, B=2, C=1, D=3
- 잎사귀 노드(Leaf Node)
  : 자식이 하나도 없는 노드 (차수가 0인 노드)
  K, L, F, G, M, I, J
- 트리의 깊이(Depth)
  : 트리에서의 최대 깊이
  Depth : 4
- 트리의 차수(Degree)
  : 노드들의 차수 중 최대 차수
  Degree : 3

이진 트리(Binary Tree)

자식이 둘 이하(차수가 2 이하)인 노드로만 구성된 트리

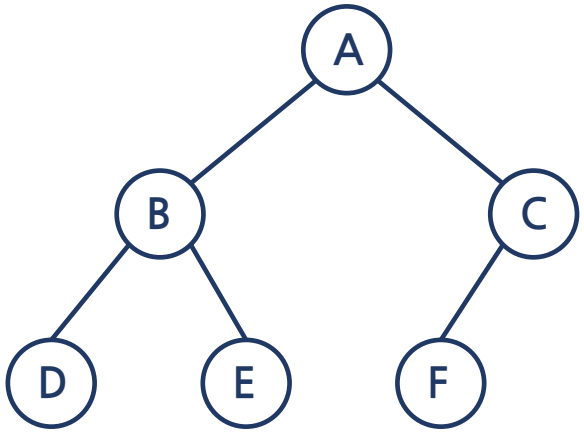
전 이진 트리(Full binary Tree)



모든 노드가 0개 또는 2개의
자식노드를 갖는 트리

- 깊이 i인 노드의 최대 노드의 수 $= 2^{i-1}$
- 깊이 k인 트리의 최대 노드의 수 $= 2^k - 1$

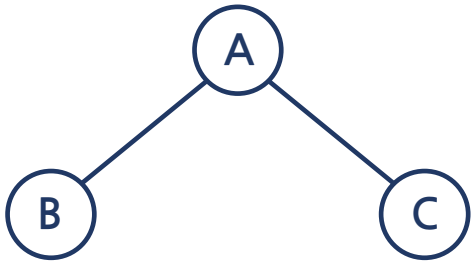
완전 이진 트리(Complete binary Tree)



마지막 깊이를 제외한 모든 깊이의 노드가
완전히 채워져 있는 트리
(모든 노드는 왼쪽부터 채워짐)

이진 트리의 순회

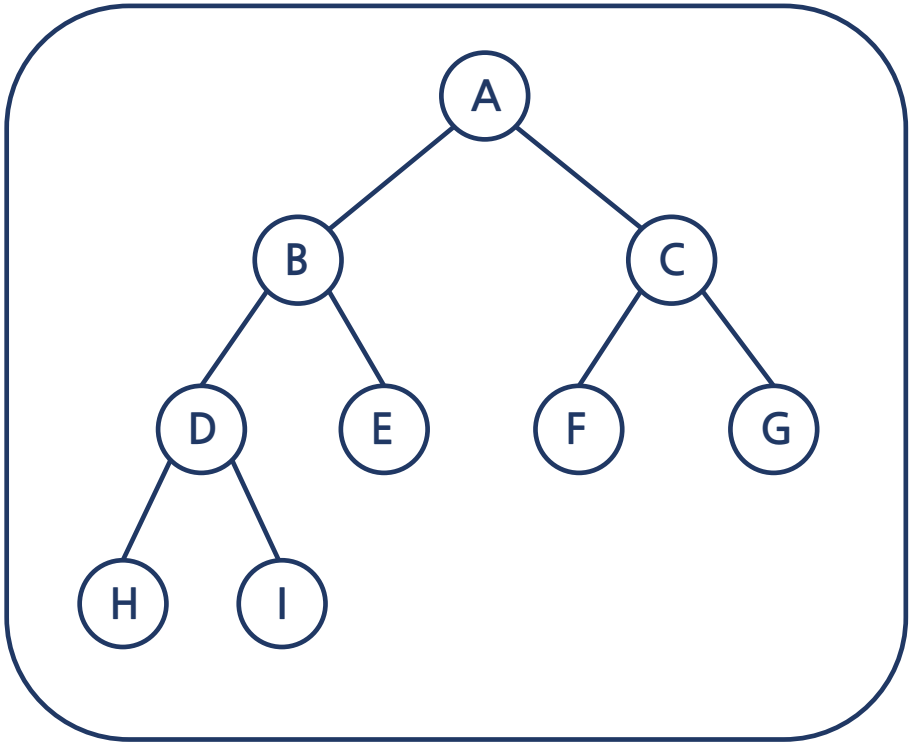
트리를 구성하는 노드들을 모두 방문하는 방법



전위 순회 (Pre-order)	Root → Left → Right	A → B→ C
중위 순회 (In-order)	Left → Root → Right	B → A→ C
후위 순회 (Post-order)	Left → Right → Root	B → C→ A

이진 트리의 순회

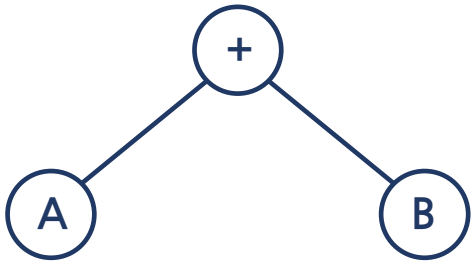
다음 트리를 Pre-order, In-order, Post-order 순으로 운행할 때 각 방문 순서는?



(1) Pre-order : ABDHIECFG (2) In-order : HDIBEAFCG (3) Post-order : HIDEBFGCA

이진 수식 트리

산술식을 계산하기 위해 이진트리에 기억시키는 방법



전위 표기법 (Pre-Fix)	Root → Left → Right	+AB
중위 표기법 (In-Fix)	Left → Root → Right	A+B
후위 표기법 (Post-Fix)	Left → Right → Root	AB+

이진 수식 트리

A / B * (C + D) + E

컴퓨터에서는 중위 표기법을 사용 시, 어떤 수식을 먼저 연산해야 할지 확인절차 필요!

스택으로 연산이 가능한 Post-Fix 나 Pre-Fix 방식을 활용

In-fix 표기를 Pre-Fix로 변환

1) 연산 우선순위에 따라 괄호로 묶음

(((A / B) * (C + D)) + E)

2) 연산자를 괄호의 앞으로 이동

+ (* (/ (A B) + (C D)) E)

3) 괄호를 제거

+* /AB+CDE

In-fix 표기를 Post-Fix로 변환

1) 연산 우선순위에 따라 괄호로 묶음

(((A / B) * (C + D)) + E)

2) 연산자를 괄호의 뒤로 이동

(((A B) / (C D) +) * E) +

3) 괄호를 제거

AB/CD+*E+

이진 수식 트리

아답터
IT의 답을 터득하다

Pre-fix 표기를 In-Fix로 변환

+ / A - B C * D + E F

1) 피연산자 두 개와 왼쪽 연산자를 괄호를 묶음

(+ (/ A (- B C)) (* D (+ E F)))

2) 연산자를 피연산자 사이로 이동

(((A / (B - C)) + (D * (E + F)))

3) 필요없는 괄호를 제거

A / (B - C) + (D * (E + F))

Post-fix 표기를 In-Fix로 변환

A B C - / D E F + * +

1) 피연산자 두 개와 오른쪽 연산자를 괄호를 묶음

((A (B C -) /) (D (E F +) *) +)

2) 연산자를 피연산자 사이로 이동

(((A / (B - C)) + (D * (E + F)))

3) 필요없는 괄호를 제거

A / (B - C) + (D * (E + F))

단기간에 **핵심**만!

기초 자료구조 특강

아답터
IT의 답을 터득하다

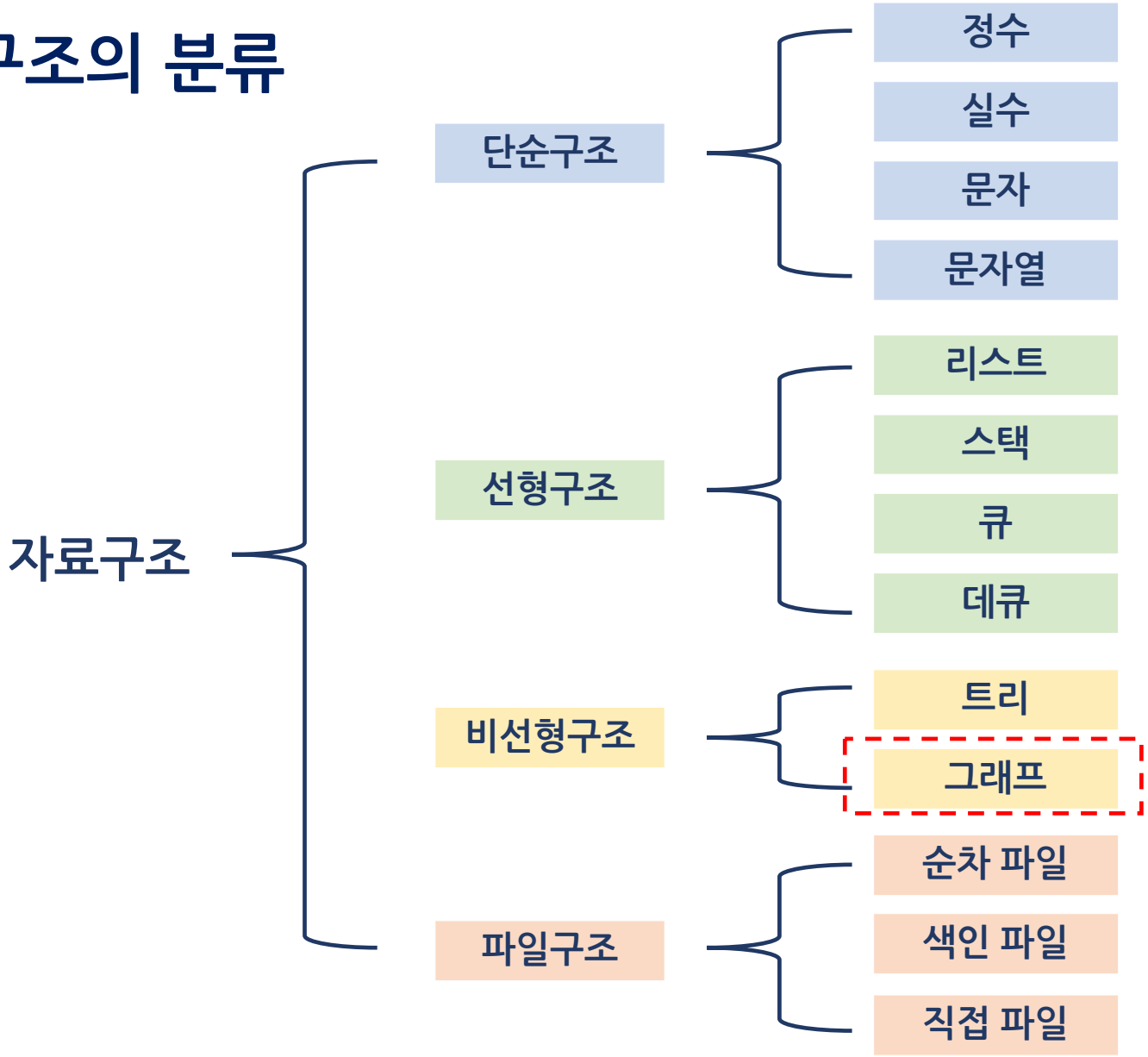
5강. 비선형 자료구조 - 그래프

- 그래프의 개념
- 그래프의 구현 방법
- 그래프 알고리즘

대학생, 직장인, 수험생을 위한

필요 내용만 간략히 배우는 자료구조

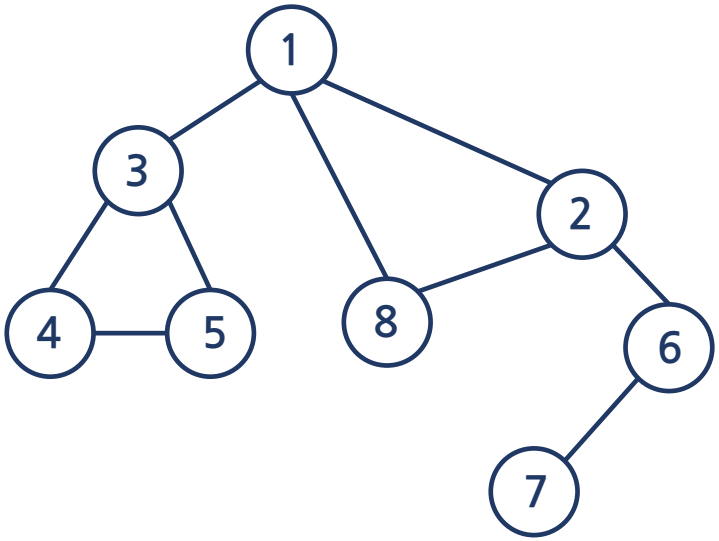
자료구조의 분류



그래프(Graph)

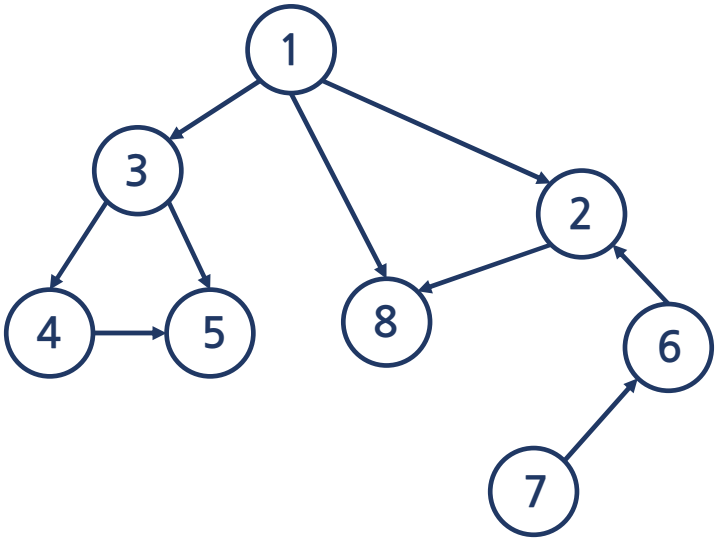
정점(Vertex) 혹은 노드(Node)와 이들을 연결하는 간선(Edge)의 집합으로 이루어진 자료구조

통신망(Network), 교통 시스템, 소셜 네트워크 분석, 인공지능 및 머신러닝



[무방향(Undirected) 그래프]

간선을 통해 양방향 이동 가능



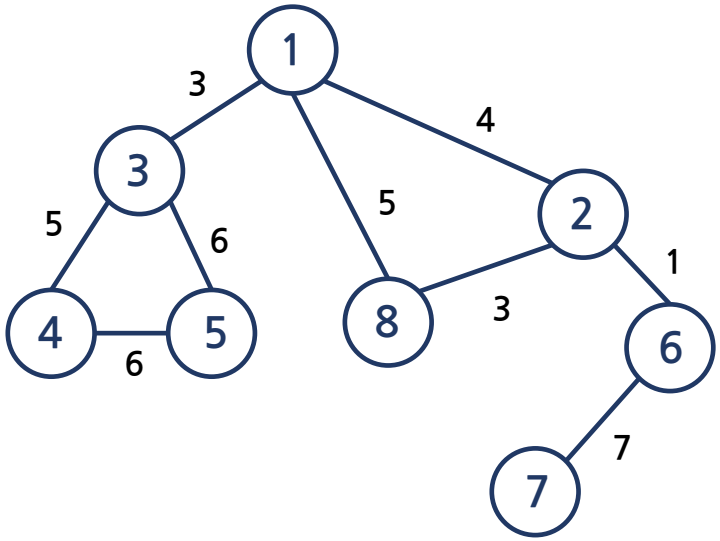
[방향(Directed) 그래프]

일방향으로만 이동이 가능

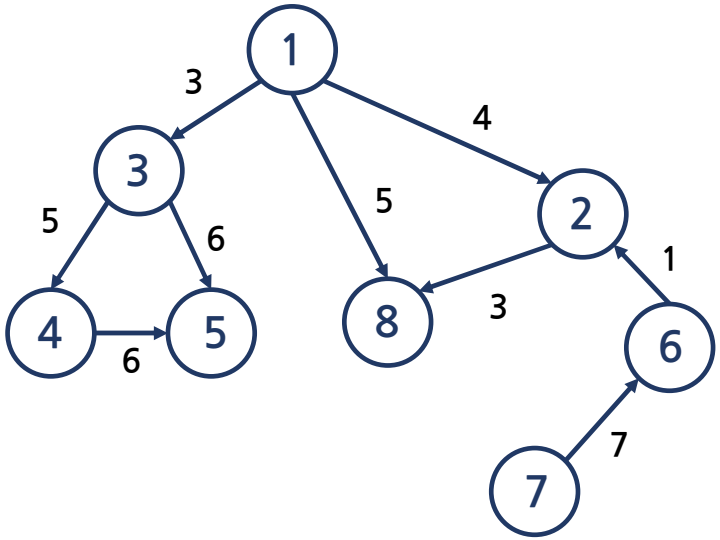
그래프(Graph)

정점(Vertex) 혹은 노드(Node)와 이들을 연결하는 간선(Edge)의 집합으로 이루어진 자료구조

통신망(Network), 교통 시스템, 소셜 네트워크 분석, 인공지능 및 머신러닝



[가중치 무방향(Weighted Undirected) 그래프]



[가중치 방향(Weighted Directed) 그래프]

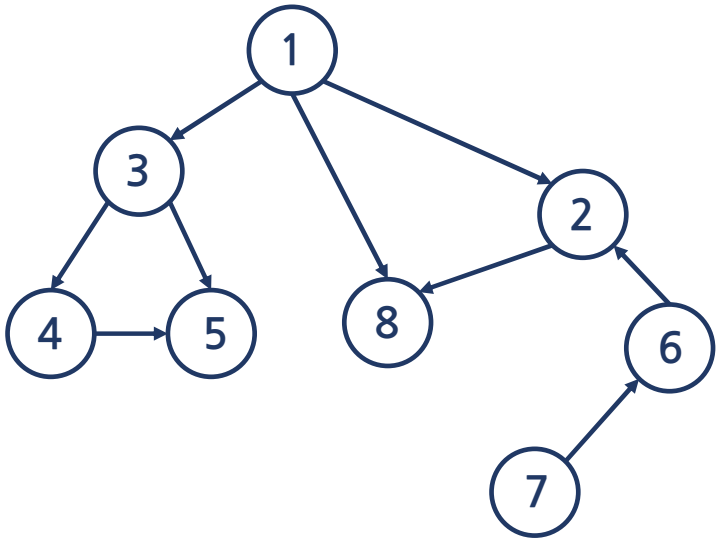
간선에 비용이나 가중치가 할당된 그래프

도로 건설 비용, 통신망의 사용료 등

그래프의 구현 방법

인접 리스트(Adjacency List)

인접한 정점을 인접 리스트에 저장하는 방법



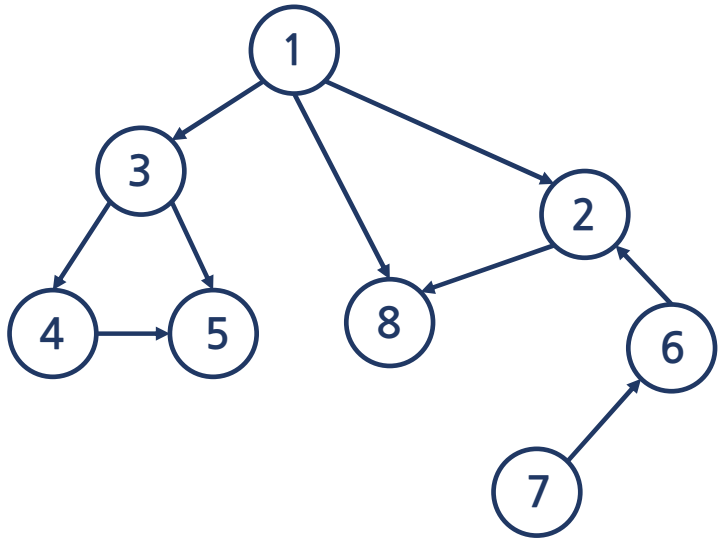
1	2, 3, 8
2	8
3	4, 5
4	5
5	
6	2
7	6
8	

인접 행렬(Adjacency Matrix)

그래프의 구현 방법

아답터
IT의 답을 터득하다

인접 리스트(Adjacency List)



i \ j	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	1
2	0	0	0	0	0	0	0	1
3	0	0	0	1	1	0	0	0
4	0	0	0	0	1	0	0	0
5	0	0	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0
7	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0

i 에서 j로 방향 간선이 존재하면 1, 없으면 0을 표현

인접 행렬(Adjacency Matrix)

2차원 배열을 이용하여 노드 간의 연결 관계를 나타내는 방법

그래프 알고리즘 종류

아답터
IT의 답을 터득하다

그래프 탐색 알고리즘

특정 정점(노드)를 찾는 알고리즘

너비 우선 탐색(BFS)

깊이 우선 탐색(DFS)

최단 경로 알고리즘

두 정점 간의
최소 가중치 합을 찾는 알고리즘

다익스트라(Dijkstra)

벨만-포드(Bellman-Ford)

플로이드 와샬(Floyd Warshall)

최소 신장 트리 알고리즘

MST, Minimum Spanning Tree

모든 정점을 최소한의 간선으로
연결하는 그래프를 찾는 알고리즘

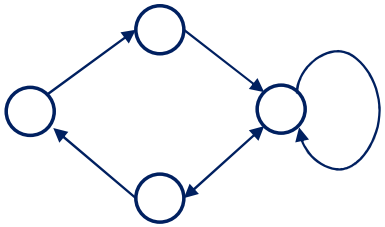
크루스칼(Kruskal)

프림(Prim)

그래프와 트리의 관계

아답터
IT의 답을 터득하다

그래프(Graph)



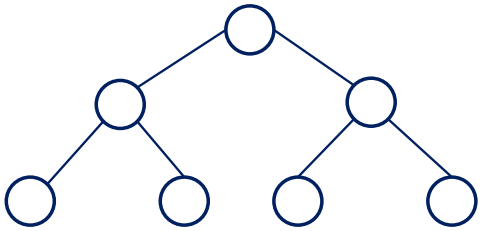
노드와 노드를 연결하는 간선으로 구성

사이클, 자체 순환 가능

부모 - 자식 개념이 없음

네트워크 모델

트리(Tree)



방향성이 있는 비순환 그래프
(트리 ⊂ 그래프)

사이클, 자체순환 불가능

부모 - 자식 관계가 존재

계층 모델

단기간에 **핵심**만!

기초 자료구조 특강

아답터
IT의 답을 터득하다

6강. 정렬(Sorting) - 1

- 내부 정렬과 외부 정렬
- 버블 정렬, 선택 정렬, 삽입 정렬, 쉘 정렬

대학생, 직장인, 수험생을 위한

필요 내용만 간략히 배우는 자료구조

자료구조의 이용

정렬

기억장치에 저장된 데이터를 일정한 순서로 나열하는 것

탐색

기억장치에서 데이터를 찾는 것

파일 편성

데이터를 기억 장치에 저장할 때의 파일 구조

인덱스

특정 데이터를 빠르게 찾기 위한 것

정렬(Sorting)

각 데이터를 특정 항목으로 오름차순 혹은 내림차순으로 재배열 하는 작업

- **내부정렬** 데이터량이 적을 때 주기억장치에서 정렬하는 방법
  - (1) 삽입법 : 삽입 정렬, 쉘 정렬
  - (2) 교환법 : 버블 정렬, 선택 정렬, 퀵 정렬
  - (3) 선택법 : 힙정렬
  - (4) 병합법 : 머지정렬(합병정렬)
  - (5) 분배법 : 기수정렬

- **외부정렬** 대용량의 데이터를 보조기억장치에서 기억시켜서 정렬하는 방법

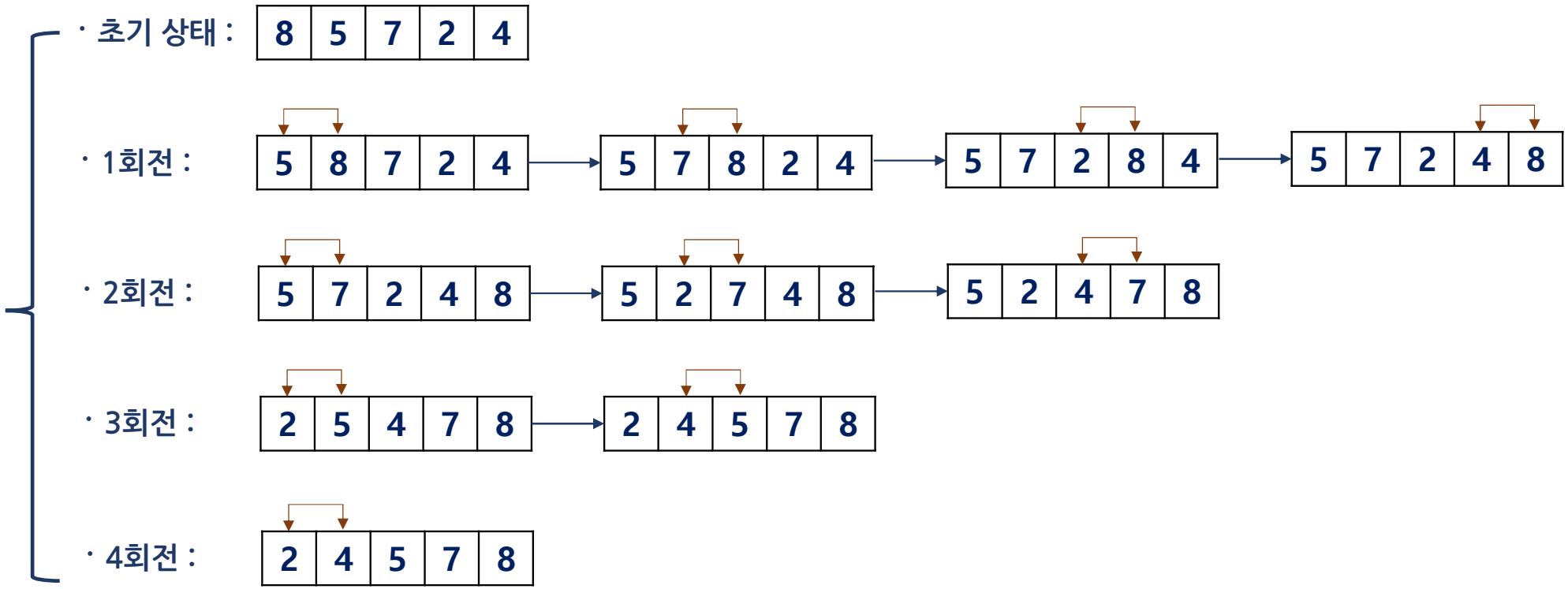
  밸런스 병합 정렬, 캐스케이드 병합 정렬, 폴리파즈 병합 정렬, 오실레이팅 병합 정렬

버블 정렬(Bubble Sort)

두 개의 데이터를 비교하여 크기에 따라 위치를 서로 교환하는 정렬 방식

평균과 최악 모두 시간복잡도는 $O(n^2)$

8,5,7,2,4를 버블 정렬로 정렬하기

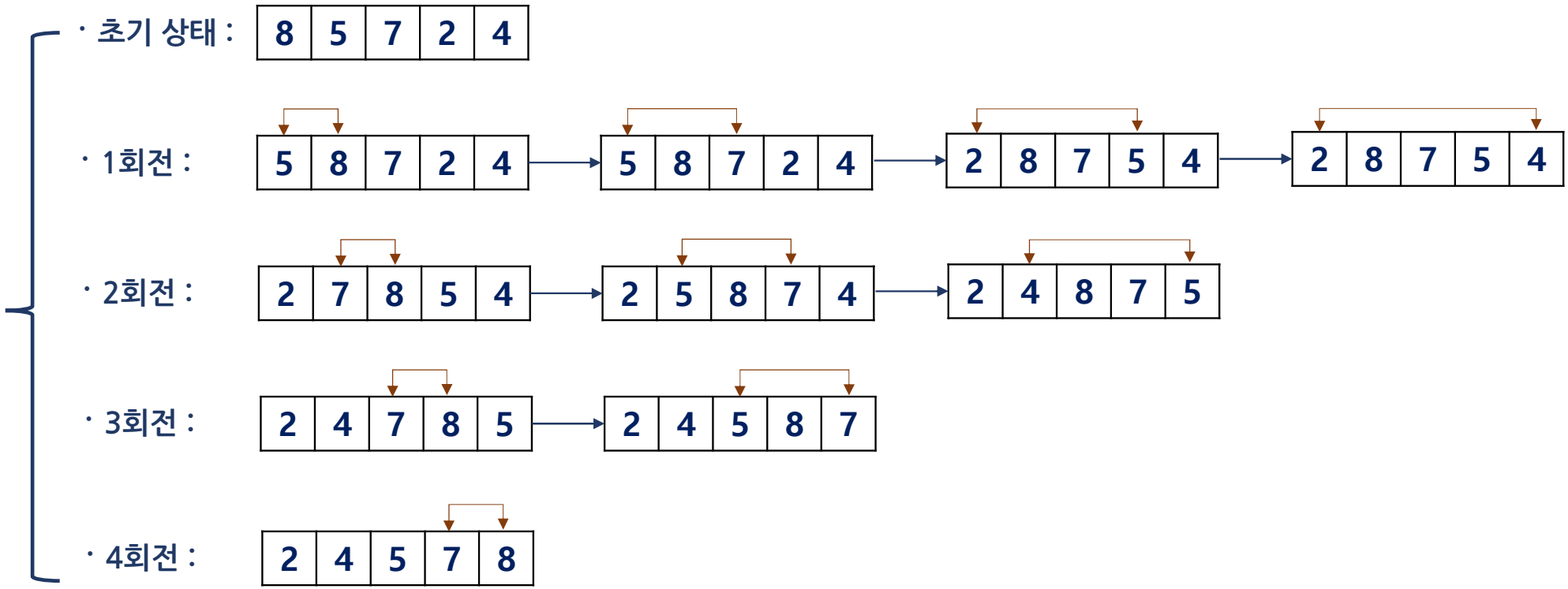


선택 정렬(Selection Sort)

N개의 데이터 중 최솟값을 찾아 첫 번째 위치에 놓고, 나머지 N-1개로 반복하여 정렬하는 방식

평균과 최악 모두 시간복잡도는 $O(n^2)$

8,5,7,2,4를 선택 정렬로 정렬하기



삽입 정렬(Insertion Sort)

순서화된 데이터에 새로운 데이터를 순서에 맞게 삽입하여 정렬하는 방식

평균과 최악 모두 시간복잡도는 $O(n^2)$

8,5,7,2,4를 삽입 정렬로 정렬하기

두 번째 값을 첫 번째 값과 비교하여
5를 첫 번째 자리에 삽입하고 8을 한 칸 뒤로 이동

세 번째 값을 첫 번째, 두 번째 값과 비교하여
7을 8자리에 삽입하고 8을 한 칸 뒤로 이동

네 번째 값을 첫 번째, 두 번째, 세 번째 값과 비교하여
2를 5자리에 삽입하고 나머지를 한 칸 뒤로 이동

다섯 번째 값을 처음부터 비교하여
4를 5자리에 삽입하고 나머지를 한 칸 뒤로 이동

쉘 정렬(Shell Sort)

삽입 정렬이 어느 정도 정렬된 배열에서 빠른 것에 착안한 삽입 정렬의 확장 기법

평균 시간복잡도는 $O(n^{1.5})$, 최악 시간복잡도는 $O(n^2)$

삽입 정렬의 문제점 : 삽입해야 할 위치가 상당히 멀다면 많은 이동이 필요

(1) 여러 개의 부분 리스트를 만들고, 각 부분 리스트를 삽입 정렬을 통하여 정렬

(2) 모든 부분 리스트의 정렬이 완료되면 더 적은 부분 리스트로 만들어 반복

단기간에 **핵심**만!

기초 자료구조 특강

아답터
IT의 답을 터득하다

7강. 정렬(Sorting) - 2

· 퀵 정렬, 힙 정렬, 병합 정렬, 기수 정렬
· 우선순위 큐

대학생, 직장인, 수험생을 위한

필요 내용만 간략히 배우는 자료구조

정렬(Sorting)

각 데이터를 특정 항목으로 오름차순 혹은 내림차순으로 재배열 하는 작업

- **내부정렬** 데이터량이 적을 때 주기억장치에서 정렬하는 방법
  - (1) 삽입법 : 삽입 정렬, 쉘 정렬
  - (2) 교환법 : 버블 정렬, 선택 정렬, 퀵 정렬
  - (3) 선택법 : 힙정렬
  - (4) 병합법 : 머지정렬(합병정렬)
  - (5) 분배법 : 기수정렬

- **외부정렬** 대용량의 데이터를 보조기억장치에서 기억시켜서 정렬하는 방법

  밸런스 병합 정렬, 캐스케이드 병합 정렬, 폴리파즈 병합 정렬, 오실레이팅 병합 정렬

퀵 정렬(Quick Sort)

데이터들을 피벗(키)을 부분적으로 나누어 가며 정렬하는 방법

평균 시간복잡도는 $O(n \log n)$, 최악 시간복잡도는 $O(n^2)$

특징

- 피벗(Pivot)을 기준으로 작은 값은 왼쪽, 큰 값은 오른쪽으로 분할
- 프로그램을 Recall해야 하므로 스택이 필요
- 분할(Divide)과 정복(Conquer)을 활용

알고리즘

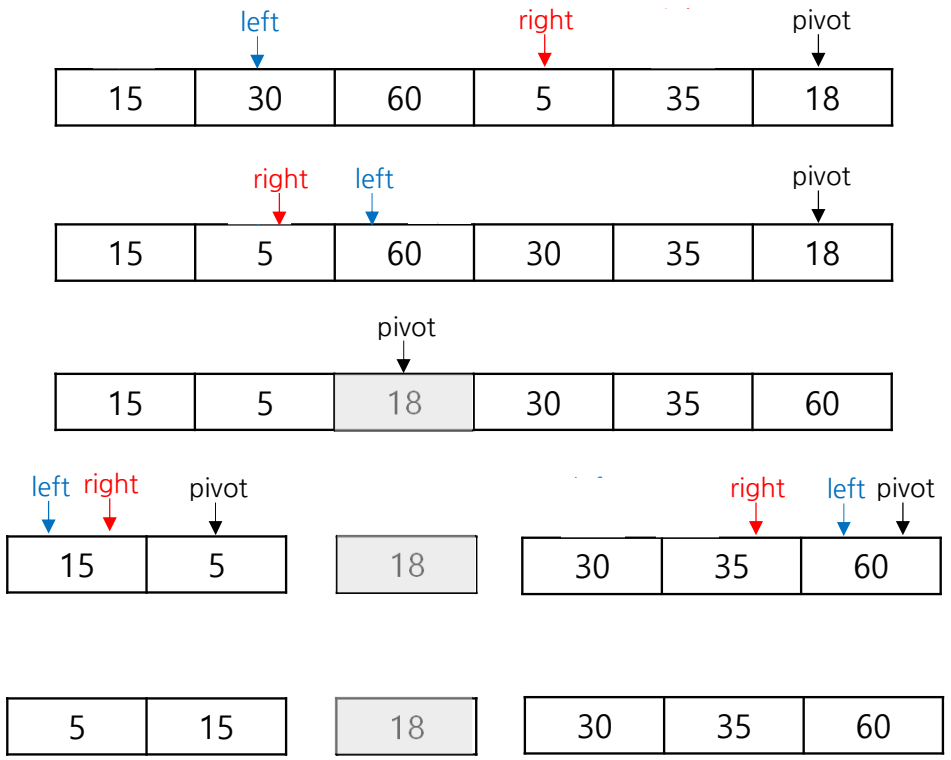
(1) Left 값 > pivot 값 만족 시 까지 Left 증가

(2) Right 값 < pivot 값 만족 시 까지 Right 감소

(3) Left >= Right 이면, Left와 Pivot 값 변경

(4) Left < Right 이면, Left와 Right 값 변경

(5) Pivot 기준으로 분할 및 반복



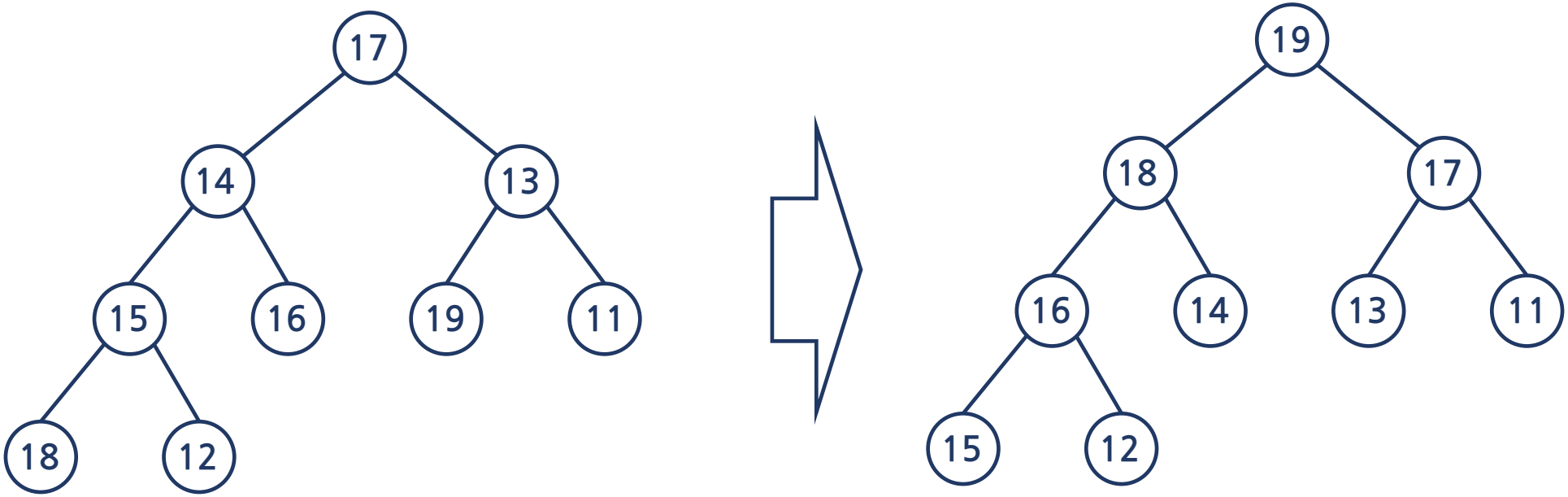
힙 정렬(Heap Sort)

전이진 트리(Complete Binary Tree)를 이용한 정렬 방식

평균과 최악 모두 시간복잡도는 $O(n \log n)$

17,14,13,15,16,19,11,18,12를 Heap트리로 구성

노드의 역순으로 자식 노드와 부모 노드를 비교하여 큰 값을 위로 올림

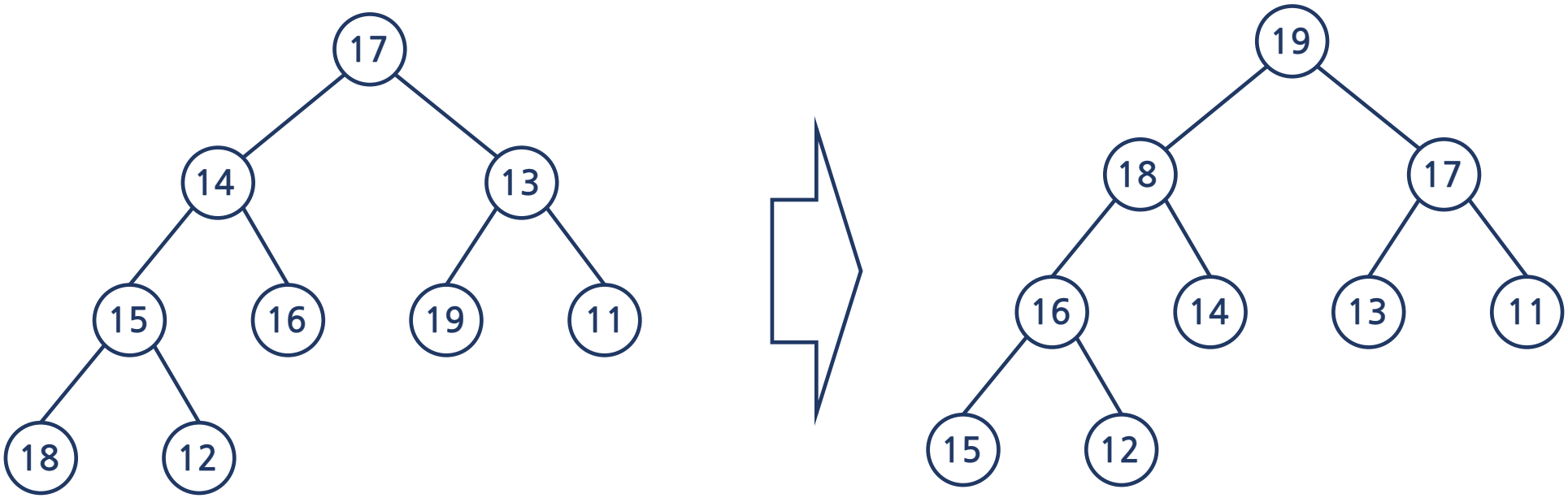


힙 정렬(Heap Sort)

아답터
IT의 답을 터득하다

- 힙 기반 데이터 저장 시 정렬 시간 복잡도 $O(\log n)$
- 힙 기반 데이터 삭제 시 정렬 시간 복잡도 $O(\log n)$

- 리스트 기반 데이터 저장 시 정렬 시간 복잡도 $O(n)$
- 리스트 기반 데이터 삭제 시 정렬 시간 복잡도 $O(1)$



힙 정렬(Heap Sort)

아답터
IT의 답을 터득하다

- 힙 기반 데이터 저장 시 정렬 시간 복잡도 $O(\log n)$
- 힙 기반 데이터 삭제 시 정렬 시간 복잡도 $O(\log n)$

우선순위 큐 (Priority Queue)

우선순위를 가진 항목들을 저장하는 큐

스택	가장 나중에 들어온 데이터가 먼저 삭제
큐	가장 먼저 들어온 데이터가 먼저 삭제
우선순위 큐	가장 우선순위가 높은 데이터가 먼저 삭제 · 네트워크 트래픽 제어 · 운영체제의 작업 스케줄링

병합 정렬(Merge Sort)

이미 정렬되어 있는 2개의 리스트를 하나의 리스트로 합병하는 방식

평균과 최악 모두 시간복잡도는 $O(n \log n)$

71,2,38,6,7,61,11,25,53,42를 병합 정렬로 정렬하기

- 1회전 : (71, 2) (38, 6) (7, 61) (11, 25) (52, 42)
  → (2, 71) (6, 38) (7, 61) (11, 25) (42, 52)
- 2회전 : ((2, 71) (6, 38)) ((7, 61) (11, 25)) (42, 52)
  → (2, 6, 38, 71) (7, 11, 25, 61) (42, 52)
- 3회전 : ((2, 6, 38, 71) (7, 11, 25, 61)) (42, 52)
  → (2, 6, 7, 11, 25, 38, 61, 71) (42, 52)
- 4회전 : ((2, 6, 7, 11, 25, 38, 61, 71) (42, 52))
  → 2, 6, 7, 11, 25, 38, 42, 52, 61, 71

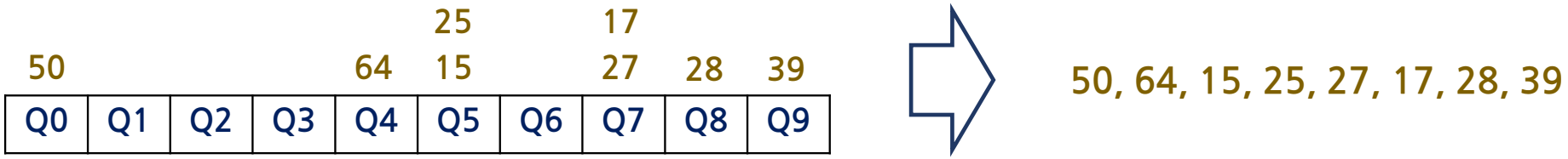
기수 정렬(Radix Sort = Bucket Sort)

큐(Queue)를 사용하여 자릿수(Digit)별로 정렬하는 방식

평균과 최악 모두 시간복잡도는 $O(dn)$

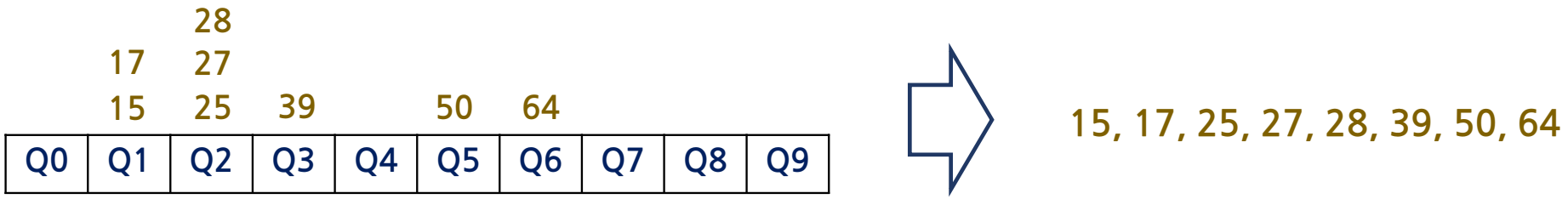
15, 27, 64, 25, 50, 17, 39, 28를 기수 정렬로 정렬하기

(1) 1의 자리에 해당하는 Bucket(큐 자료구조)에 데이터를 분배 후, 꺼내어 정렬



50, 64, 15, 25, 27, 17, 28, 39

(2) 10의 자리에 해당하는 Bucket(큐 자료구조)에 데이터를 분배 후, 꺼내어 정렬



15, 17, 25, 27, 28, 39, 50, 64

단기간에 **핵심**만!

기초 자료구조 특강

아답터
IT의 답을 터득하다

8강. 탐색(Search)

- 순차 탐색, 이진 탐색
- 피보나치 탐색, 보간 탐색, 블록 탐색, 이진 트리 탐색

대학생, 직장인, 수험생을 위한

필요 내용만 간략히 배우는 자료구조

자료구조의 이용

정렬

기억장치에 저장된 데이터를 일정한 순서로 나열하는 것

탐색

기억장치에서 데이터를 찾는 것

파일 편성

데이터를 기억 장치에 저장할 때의 파일 구조

인덱스

특정 데이터를 빠르게 찾기 위한 것

탐색(Search)

기억공간에 저장된 특정 데이터를 찾아내는 작업

- **순차 탐색** 순서화 되어 있지 않은 데이터를 첫 번째 부터 차례대로 탐색하는 방식

- **제어 탐색** 반드시 순서화 되어있는 데이터를 탐색하는 방식

  - (1) 이진 탐색(Binary Search)
  - (2) 피보나치 탐색(Fibonacci Search)
  - (3) 보간 탐색(Interpolation Search)
  - (4) 블록 탐색(Block Seach)
  - (5) 이진 트리 탐색(Binary Tree Search)

순차탐색과 이진탐색

선형리스트(배열)에서 숫자 21을 찾아보자

순차탐색인 경우

최악의 경우 n번을 탐색해야 됨

$O(n)$

이진탐색인 경우

첫번째 탐색 범위

1/2

두번째 탐색 범위

1/2

세번째 탐색 범위

n이 1이 될 때까지 2로 k회 나눔

$\rightarrow n \times (\frac{1}{2})^k = 1$

$\rightarrow k = \log_2 n$

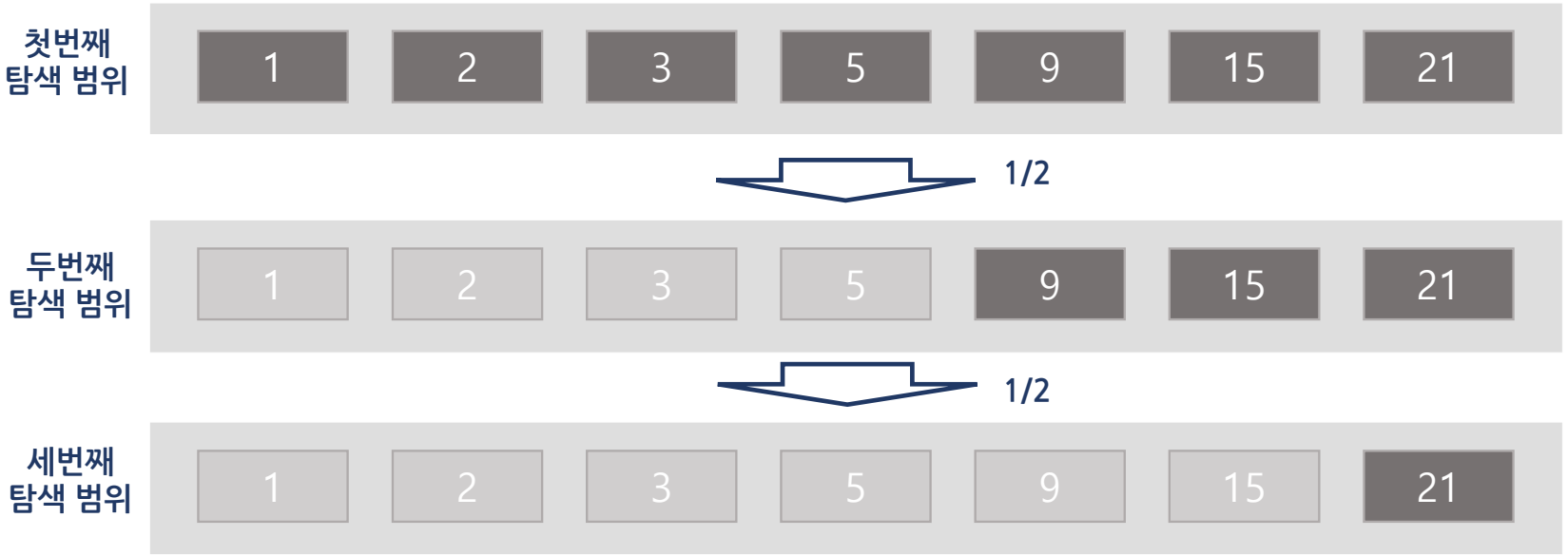
$\therefore O(\log_2 n)$

이진 탐색(Binary Search)

전체 데이터를 두 개의 서브 데이터 세트로 분리해 가면서 찾고자 하는 데이터를 탐색하는 방식

특징

- 비교 횟수를 거듭할수록 검색 대상이 되는 데이터 수가 절반으로 줄어들기에 탐색 효율이 가장 좋음
- 중간 레코드 번호 $M = \frac{(F+L)}{2}$ (F: 첫번째 데이터 번호, L : 마지막 데이터 번호)



n이 1이 될 때까지 2로 k회 나눔

$\rightarrow n \times (\frac{1}{2})^k = 1$

$\rightarrow k = \log_2 n$

$\therefore O(\log_2 n)$

탐색(Search)

아답터 IT의 답을 터득하다

기억공간에 저장된 특정 데이터를 찾아내는 작업

- **순차 탐색** 순서화 되어 있지 않은 데이터를 첫 번째 부터 차례대로 탐색하는 방식

- **제어 탐색** 반드시 순서화 되어있는 데이터를 탐색하는 방식

  (1) 이진 탐색(Binary Search)

  전체 데이터를 두 개의 서브 데이터 세트로 분리해 가면서 찾고자 하는 데이터를 탐색하는 방식, 가장 뛰어난 성능

  (2) 피보나치 탐색(Fibonacci Search)

  피보나치 수열에 따라 다음 비교할 대상을 선정하며, 가감산만을 활용하기에 효율이 우수

  (3) 보간 탐색(Interpolation Search)

  ※ 피보나치 수열 : 앞의 두 수의 합이 바로 뒤의 수가 되는 배열 ($F_n = F_{n-1} + F_{n-2}$)

  찾으려는 데이터가 있을법한 부분의 키를 택하여 탐색하는 방식이나 예측을 해야 하므로 프로그래밍 불가

  (4) 블록 탐색(Block Seach)

  여러 데이터를 Block 단위로 구성하여 저장해놓고, 어느 Block에 있는지 확인하여 탐색

  (5) 이진 트리 탐색(Binary Tree Search)

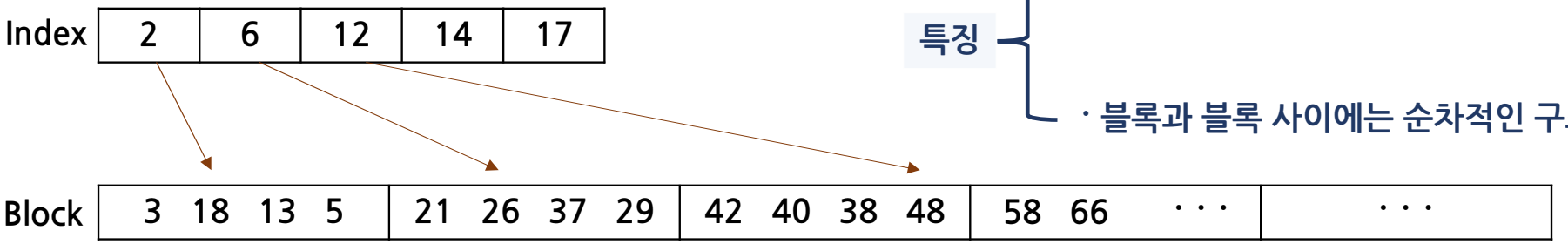
  이진 검색 트리로 구성하여 탐색하는 방식

  ※ 이진 검색 트리 : 왼쪽 자식 노드의 값은 부모 노드 값보다 작고 오른쪽 자식 노드 값은 부모 노드 값보다 큰 값을 갖도록 구성한 트리

블록 탐색과 이진 트리 탐색

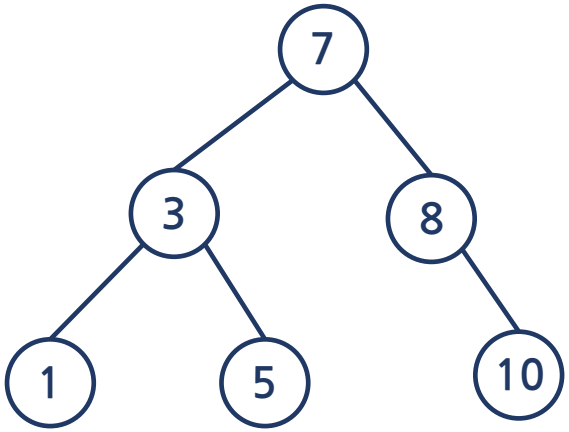
아답터
IT의 답을 터득하다

숫자 38를 블록 탐색으로 찾아보자



특징
- 각 블록 내 데이터들이 순차적으로 저장될 필요는 없음
- 블록과 블록 사이에는 순차적인 구조로 저장

숫자 5를 이진 트리 탐색으로 찾아보자



알고리즘
(1) 찾고자 하는 데이터를 Root Node와 비교
(2) 찾고자 하는 데이터가 Root Node보다 작으면 왼쪽 Sub Tree 검색
(3) 찾고자 하는 데이터가 Root Node보다 크면 오른쪽 Sub Tree 검색
(4) 같으면 검색을 종료

탐색(Search)

기억공간에 저장된 특정 데이터를 찾아내는 작업

- **순차 탐색** 순서화 되어 있지 않은 데이터를 첫 번째 부터 차례대로 탐색하는 방식

- **제어 탐색** 반드시 순서화 되어있는 데이터를 탐색하는 방식

  - (1) 이진 탐색(Binary Search)
  - (2) 피보나치 탐색(Fibonacci Search)
  - (3) 보간 탐색(Interpolation Search)
  - (4) 블록 탐색(Block Seach)
  - (5) 이진 트리 탐색(Binary Tree Search)

제어탐색 방식은 아무리 빨라도 시간복잡도가 $O(logN)$

메모리를 많이 사용하더라도 시간복잡도를 더 줄일 수 있다면?

해싱(Hashing)

단기간에 **핵심**만!

기초 자료구조 특강

아답터
IT의 답을 터득하다

9강. 탐색 - 해싱(Hashing)

- 해시 함수, 해시 테이블, 해시 충돌
- Open Hashing, Close Hashing

대학생, 직장인, 수험생을 위한

필요 내용만 간략히 배우는 자료구조

자료구조의 이용

정렬

기억장치에 저장된 데이터를 일정한 순서로 나열하는 것

탐색

기억장치에서 데이터를 찾는 것

파일 편성

데이터를 기억 장치에 저장할 때의 파일 구조

인덱스

특정 데이터를 빠르게 찾기 위한 것

제어탐색 방식은 아무리 빨라도 시간복잡도가 $O(logN)$

메모리를 많이 사용하더라도 시간복잡도를 더 줄일 수 있다면?

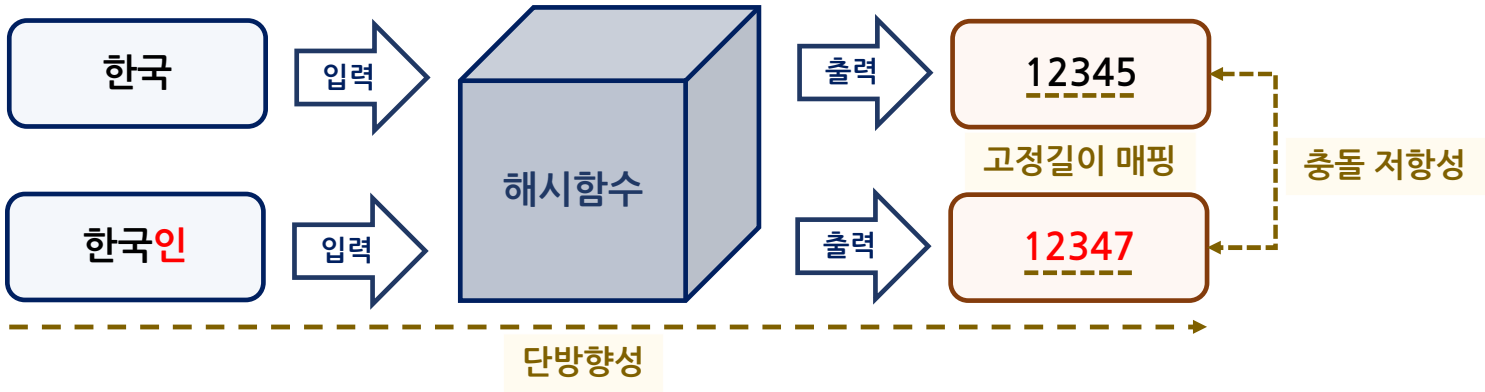
해싱(Hashing)

해싱(Hashing)

각각의 데이터를 고유한 숫자 값으로 표현하고 이를 통하여 기억장소에 저장하거나 검색 작업을 수행하는 방식

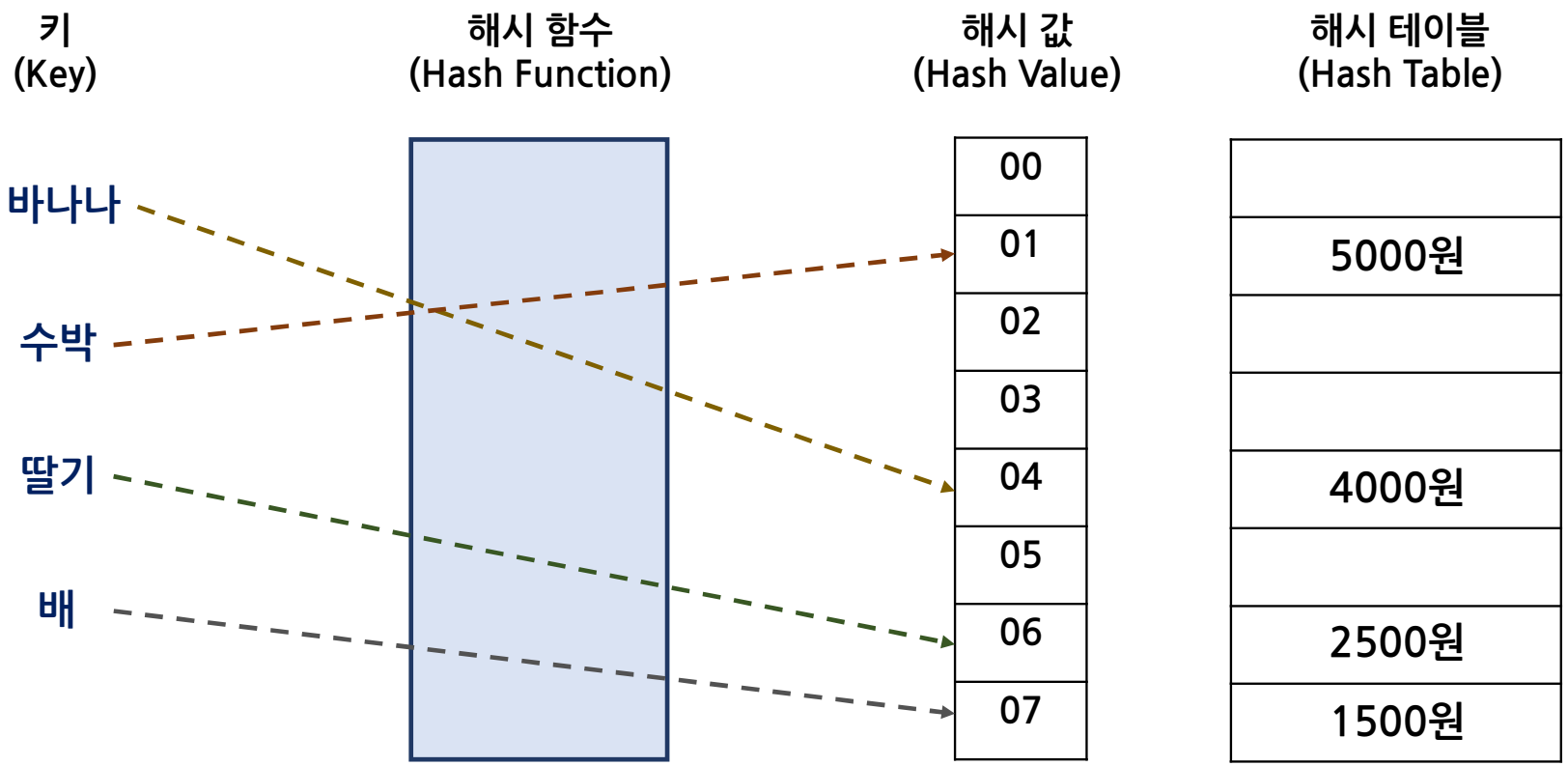
- 해시 함수(Hash Function)을 이용하여, 해시 테이블(Hash Table)에 인덱스를 계산
- 다른 방식에 비해 검색 속도가 가장 빠름
- 삽입, 삭제의 빈도가 많을 때 유리한 방식
- Key-Value 변환 방법
- 데이터 검색과 저장 뿐만 아니라 데이터 무결성 검사, 암호화에도 활용

해시함수란? 임의의 길이를 가진 데이터를 입력 받아 고정된 길이의 값(해시 값)을 출력하는 함수



해시 테이블(Hash Table)

데이터가 저장되는 곳으로, 해시 함수를 통해 산출된 해시 값(Hash Value)을 인덱스로 하여 데이터를 저장



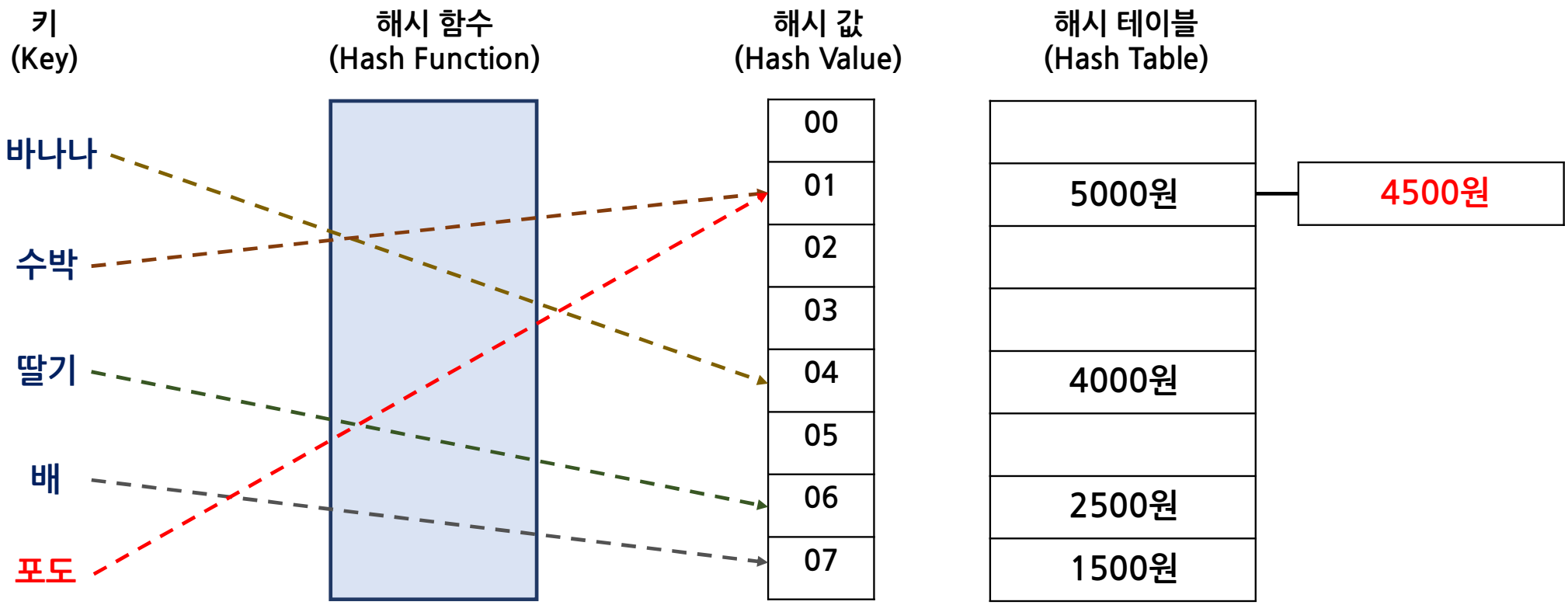
해시 충돌(Collision)

해시 함수가 서로 다른 Key에 대하여 같은 값을 반환하여 같은 위치에 두 개 이상 데이터가 저장되는 현상

Open Hashing

Close Hashing

Chaining, 혹은 Close Addressing 기법이라고도 하며 해시 테이블 저장공간 이외 공간 활용



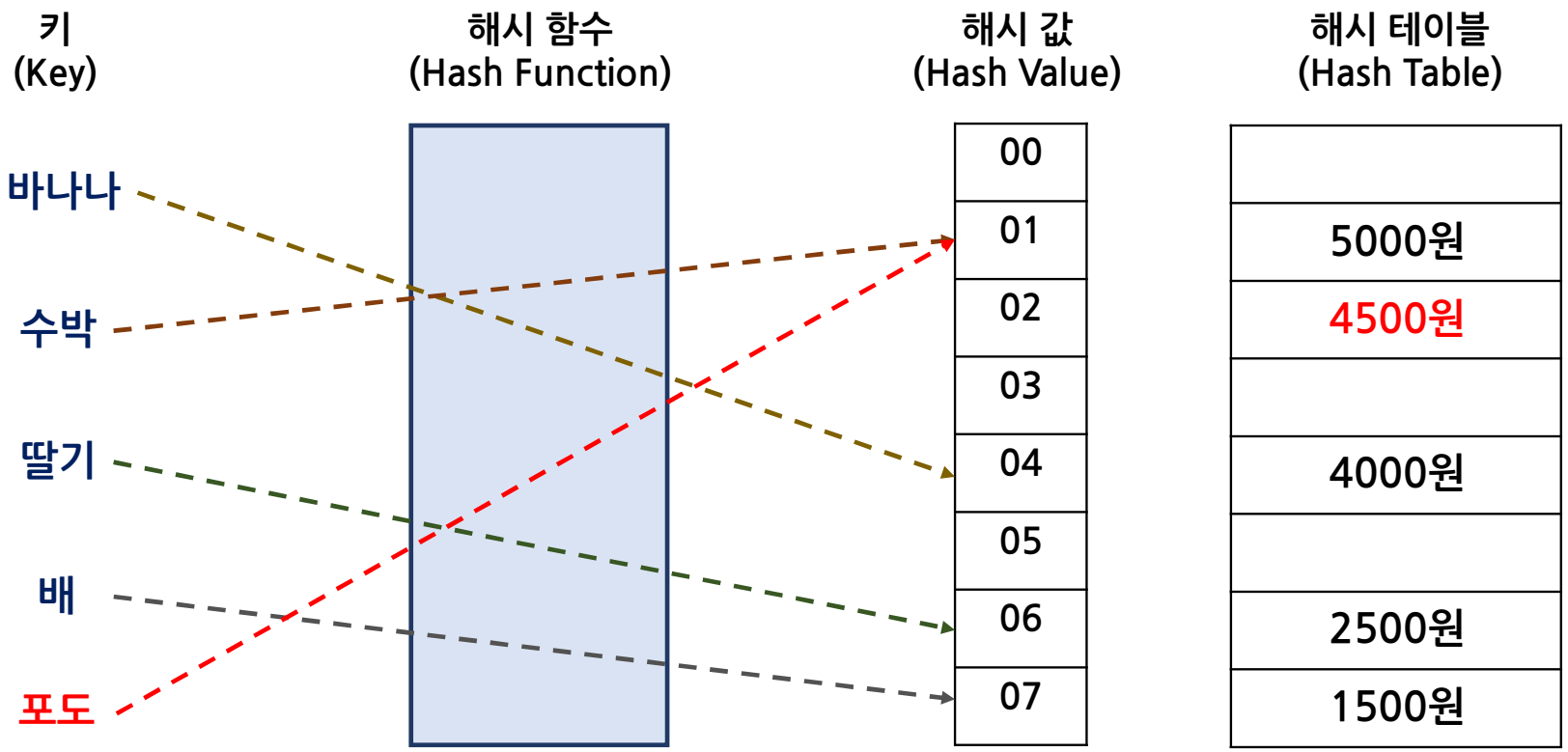
해시 충돌(Collision)

해시 함수가 서로 다른 Key에 대하여 같은 값을 반환하여 같은 위치에 두 개 이상 데이터가 저장되는 현상

Open Hashing

Close Hashing

Linear Probing, 혹은 Open Addressing 기법이라고도 하며 해시 테이블 공간 안에서 순차적으로 다음 빈 공간에 저장



단기간에 **핵심**만!

기초 자료구조 특강

아답터
IT의 답을 터득하다

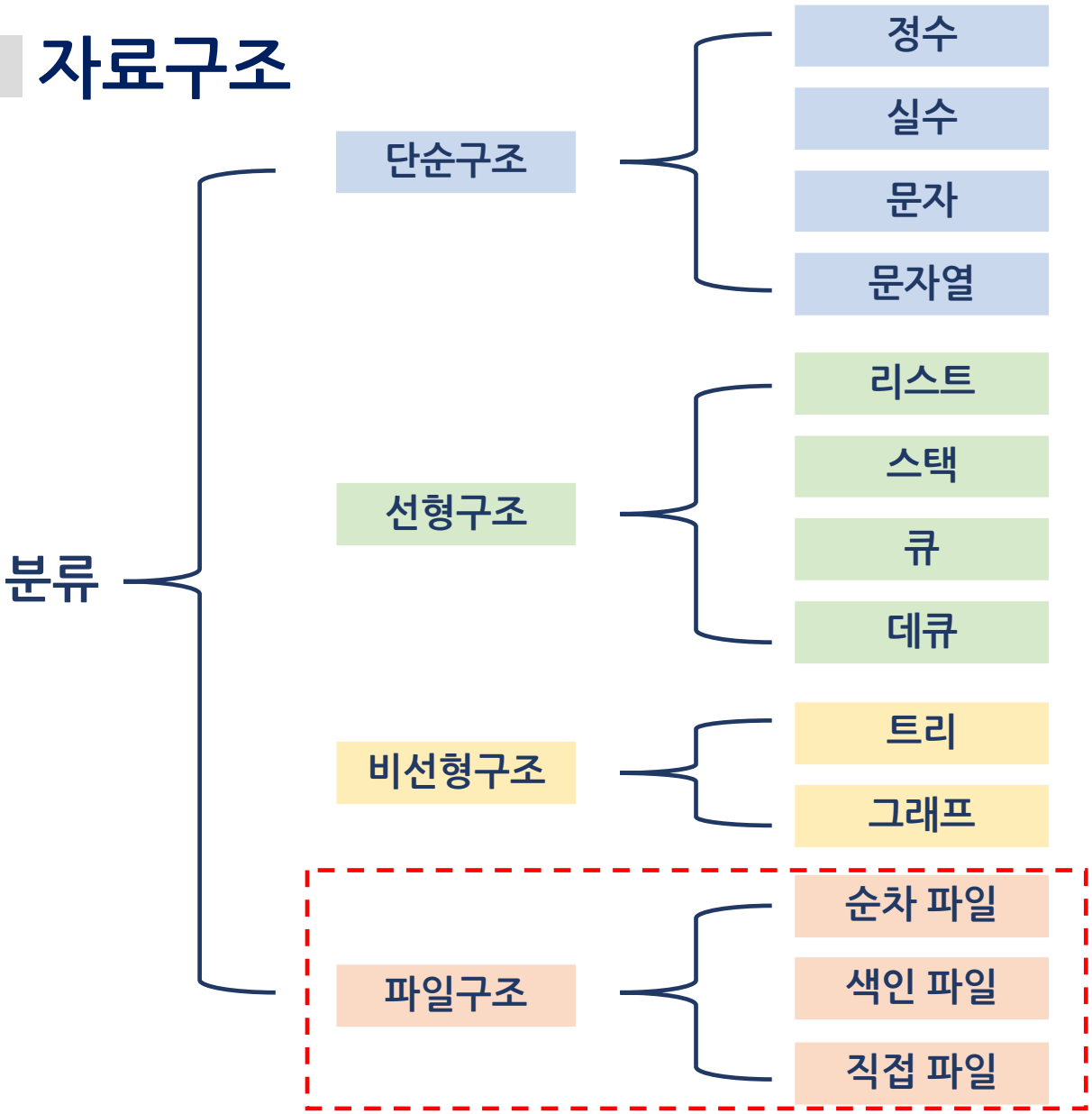
10강. 인덱스와 파일구조

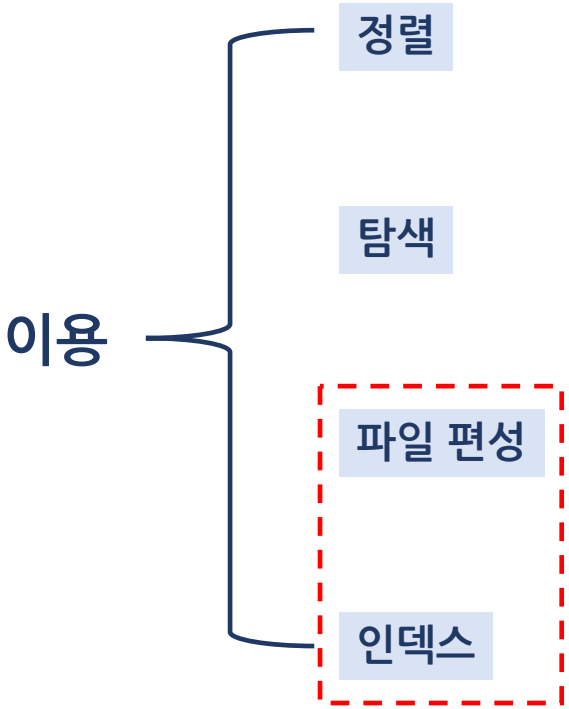
- B-트리, B+트리
- 순차 파일, 색인 파일, 직접 파일

대학생, 직장인, 수험생을 위한

필요 내용만 간략히 배우는 자료구조

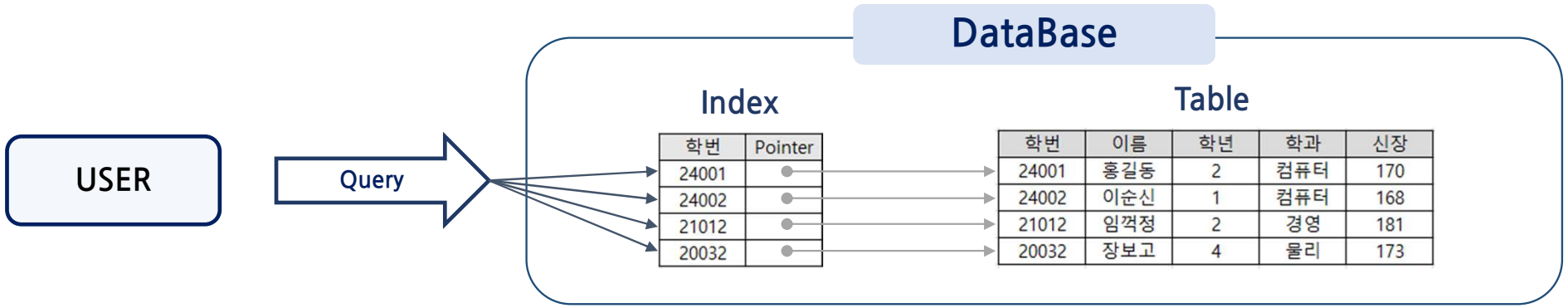
자료구조


분류
단순구조
정수
실수
문자
문자열
선형구조
리스트
스택
큐
데큐
비선형구조
트리
그래프
파일구조
순차 파일
색인 파일
직접 파일


이용
정렬
탐색
파일 편성
인덱스

인덱스

데이터베이스의 테이블에 대한 검색 속도를 향상시켜주는 자료구조



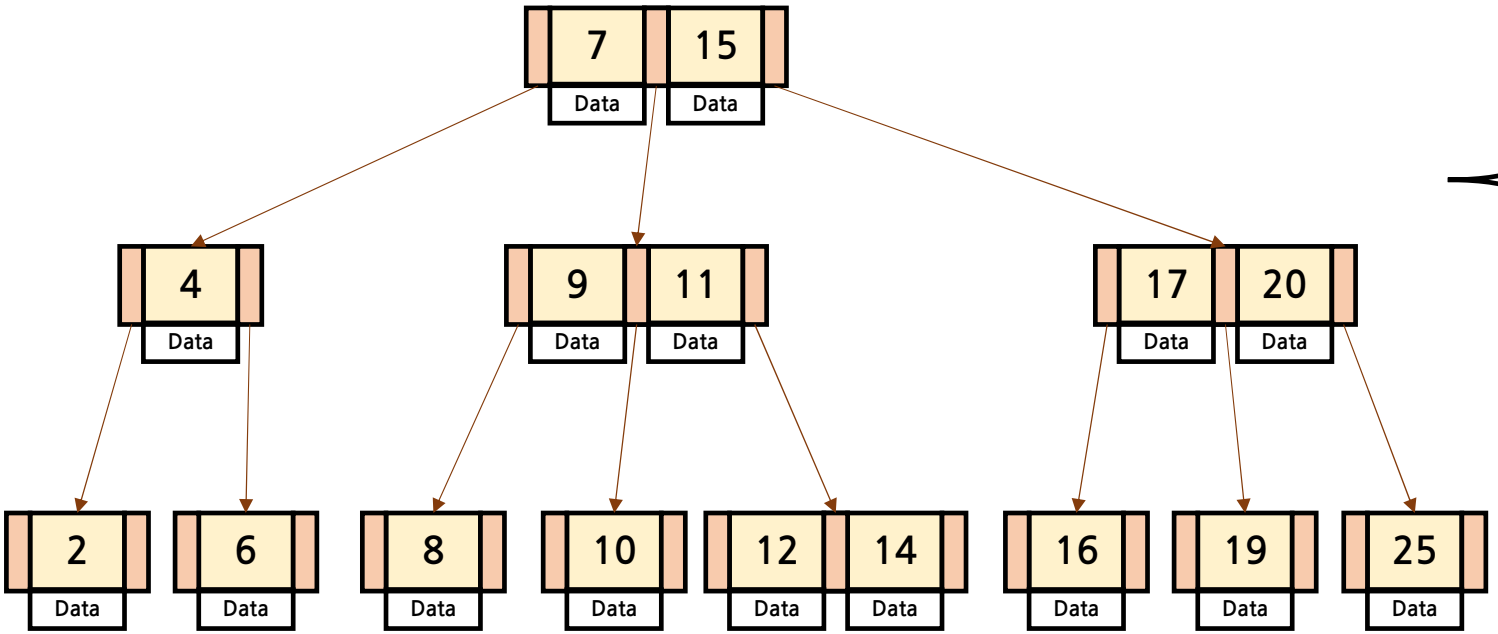
- 테이블을 검색하는 속도와 성능이 향상
- 데이터가 저장된 물리적 구조와 밀접한 관계
- 인덱스를 관리하기 위한 추가 작업 및 저장 공간 필요
- 잘못 사용하는 경우 오히려 성능 저하 발생

인덱스

데이터베이스의 테이블에 대한 검색 속도를 향상시켜주는 자료구조

B-Tree | B+Tree | Hash Table

탐색 성능을 높이기 위해 균형 있게 높이를 유지하는 검색 트리



- Node의 Key의 수가 k개라면, 자식 Node의 수는 k+1개
- Node의 key는 오름차순을 유지
- 모든 Leaf는 같은 Level에 존재
- Root Node는 항상 2개 이상의 자식 Node를 가짐

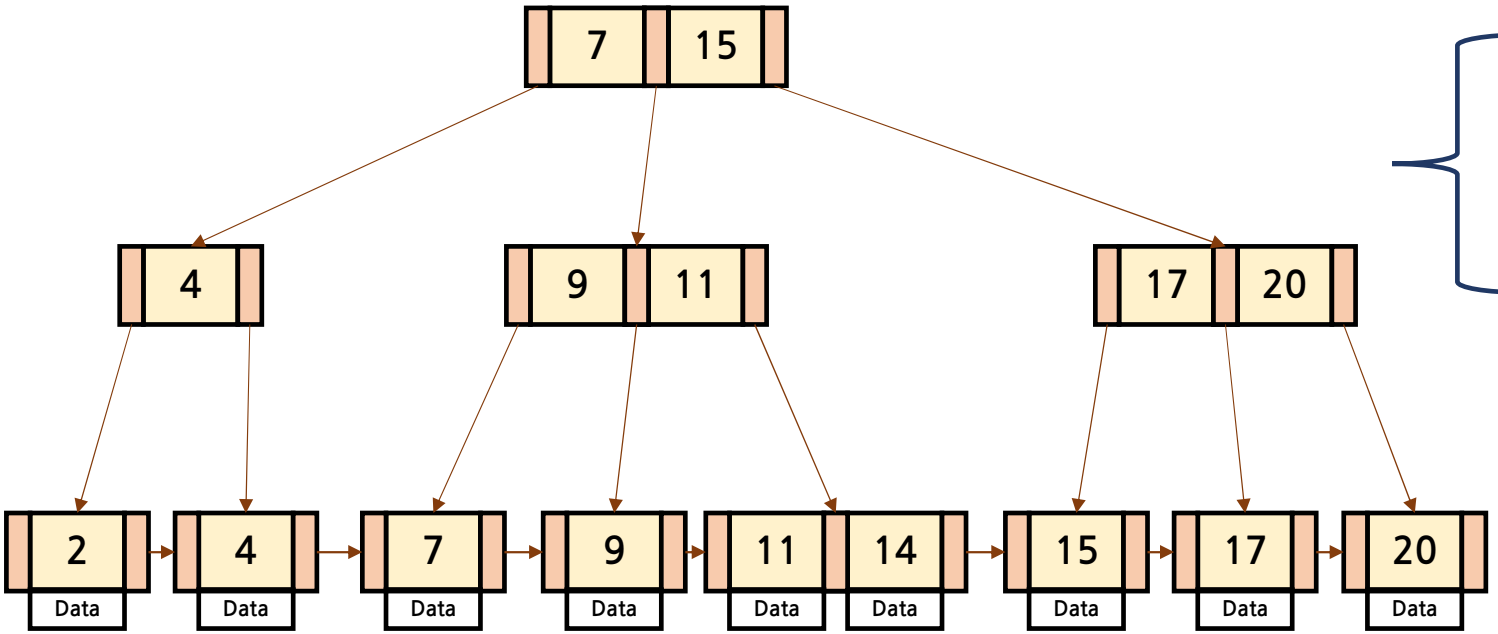
탐색 시 모든 노드를 방문해야 하므로 비효율적

인덱스

데이터베이스의 테이블에 대한 검색 속도를 향상시켜주는 자료구조

B-Tree | **B+Tree** | Hash Table

B-Tree에서 확장된 트리로서, Leaf 노드를 제외하고는 데이터를 저장하지 않는 트리



- Internal Node는 Key와 Pointer만을 저장하여 메모리 효율이 좋음
- 탐색 시 Leaf Node간 Linked List로 연결되어 있어 선형 시간이 소요

파일구조

보조기억장치에 파일을 구성하는 데이터들을 저장하는 방식

순차 파일(Sequential File)

데이터를 물리적 연속 공간에 순차적으로 기록하는 방식

- 연속적인 공간에 저장하므로 메모리 효율이 높음
- 가장 간단하여 어떤 매체에서도 용이하게 사용
- 삽입/삭제 시 전체를 복사하므로 시간이 많이 소요
- 순차적으로 검색 하기에 검색 효율 및 응답 시간이 낮음

직접 파일(Direct File)

데이터를 특정 순서 없이 해시함수를 활용하여 물리적 저장공간에 임의로 기록하는 방식

- 접근 시간이 빠르고 삽입, 삭제, 갱신이 용이
- 어떤 데이터라도 평균 접근 시간내 검색이 가능
- 데이터 주소 변환을 위한 메모리 효율 저하
- 기억장치의 물리적 구조를 알아야 하며, 프로그래밍 복잡

색인 순차 파일(Indexed Sequential File)

데이터를 키 값 순으로 정렬(Sort)시켜 기록하고, 색인(Index)을 구성하여 편성하는 방식

파일구조

보조기억장치에 파일을 구성하는 데이터들을 저장하는 방식

색인 순차 파일(Indexed Sequential File)

데이터를 키 값 순으로 정렬(Sort)시켜 기록하고, 색인(Index)을 구성하여 편성하는 방식

구성 요소

- 기본 구역(Prime Area)
  실제 데이터가 기록되는 구역
- 색인 구역(Index Area)
  기본 구역을 찾아가기 위한 Index로 구성 (Track Index < Cylinder Index < Master Index)
- 오버플로 구역(Overflow Area)
  기본 구역에 빈 공간이 없어 새로운 데이터 삽입이 불가할 때를 대비한 공간

특징

- 순차처리와 랜덤처리가 모두 가능하여, 융통성 있는 활용 가능
- 효율적인 검색, 삽입, 삭제, 갱신이 가능
- 색인 구역과 오버플로 구역을 구성하기 위한 추가 기억공간이 필요