Chapter 01
기본 자료구조

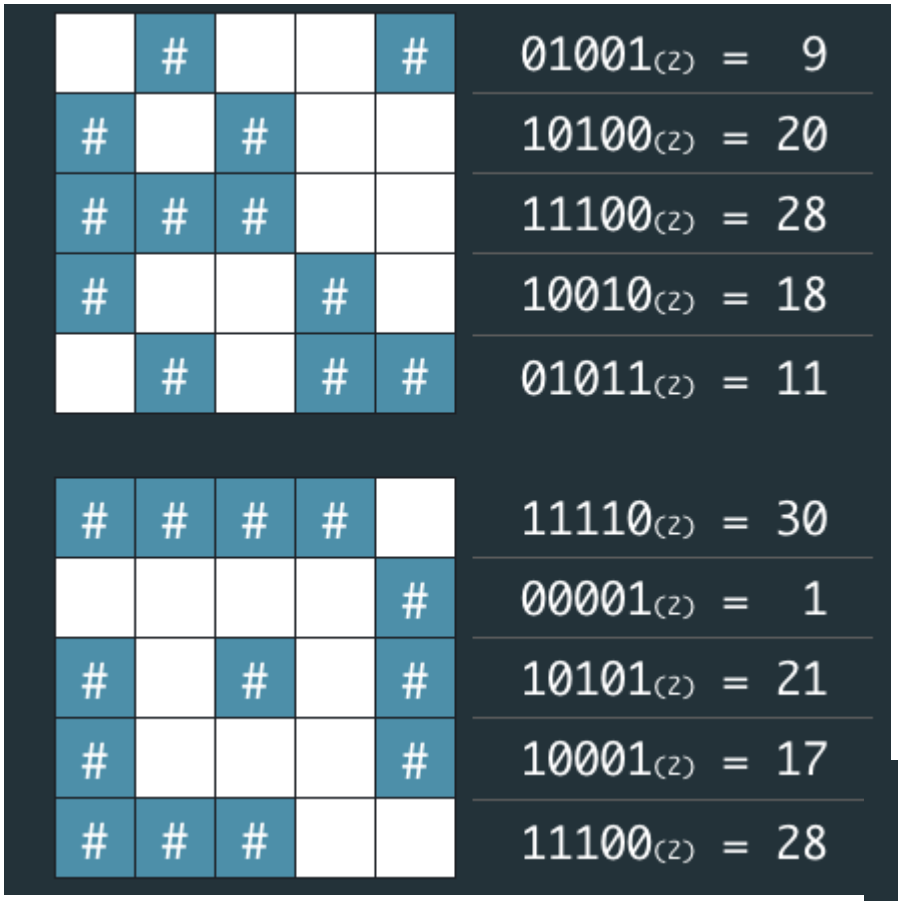
문제

※ 비밀 지도(난이도: 하)

네오는 평소 프로도가 비상금을 숨겨놓는 장소를 알려줄 비밀지도를 손에 넣었다. 그런데 이 비밀지도는 숫자로 암호화되어 있어 위치를 확인하기 위해서는 암호를 해독해야 한다. 다행히 지도 암호를 해독할 방법을 적어놓은 메모도 함께 발견했다.

- 지도는 한 변의 길이가 n인 정사각형 배열 형태로, 각 칸은 “공백”(“ “) 또는 “벽”(“#”) 두 종류로 이루어져 있다.
- 전체 지도는 두 장의 지도를 겹쳐서 얻을 수 있다. 각각 “지도 1”과 “지도 2”라고 하자. 지도 1 또는 지도 2 중 어느 하나라도 벽인 부분은 전체 지도에서도 벽이다. 지도 1과 지도 2에서 모두 공백인 부분은 전체 지도에서도 공백이다.
- “지도 1”과 “지도 2”는 각각 정수 배열로 암호화되어 있다.
- 암호화된 배열은 지도의 각 가로줄에서 벽 부분을 1, 공백 부분을 0으로 부호화했을 때 얻어지는 이진수에 해당하는 값의 배열이다.

문제



네오가 프로도의 비상금을 손에 넣을 수 있도록, 비밀지도의 암호를 해독하는 작업을 도와줄 프로그램을 작성하라.

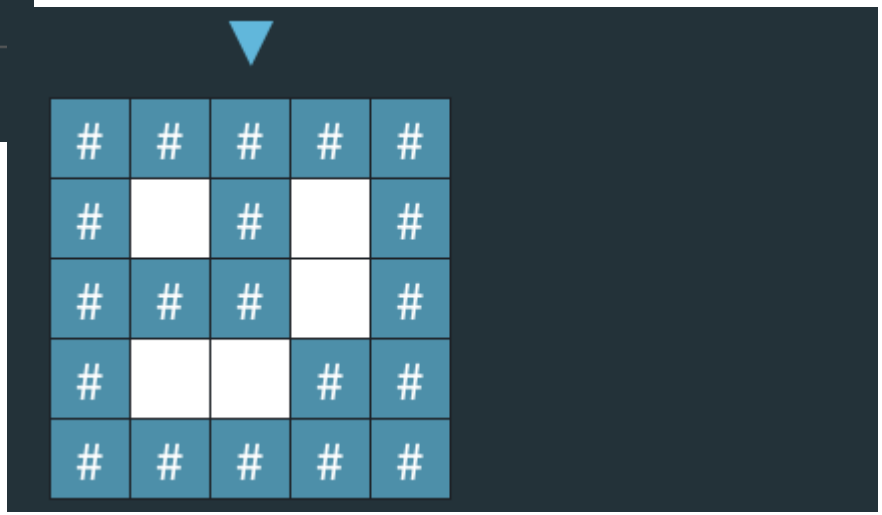
▶입력 형식

입력으로 지도의 한 변 크기 n 과 2개의 정수 배열 arr1, arr2가 들어온다.

- 1 ≦ n ≦ 16
- arr1, arr2는 길이 n인 정수 배열로 주어진다.
- 정수 배열의 각 원소 x를 이진수로 변환했을 때의 길이는 n 이하이다. 즉, 0 ≦ x ≦ 2^n – 1을 만족한다.

▶ 출력 형식

원래의 비밀지도를 해독하여 "#", 공백으로 구성된 문자열 배열로 출력하라.

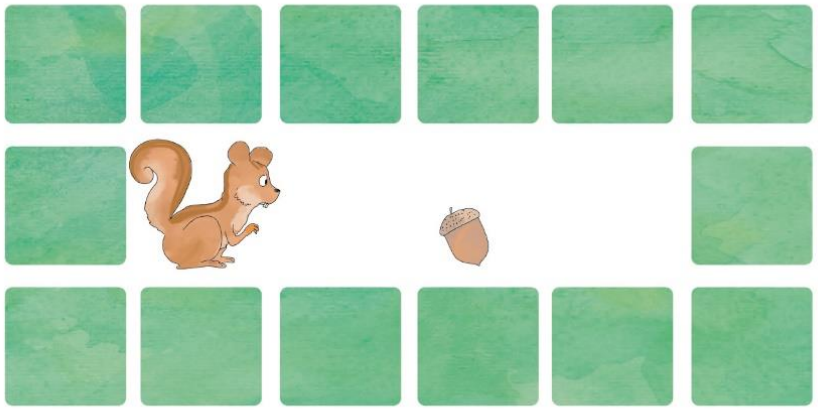


2017년 카카오공채 코딩테스트 문제로

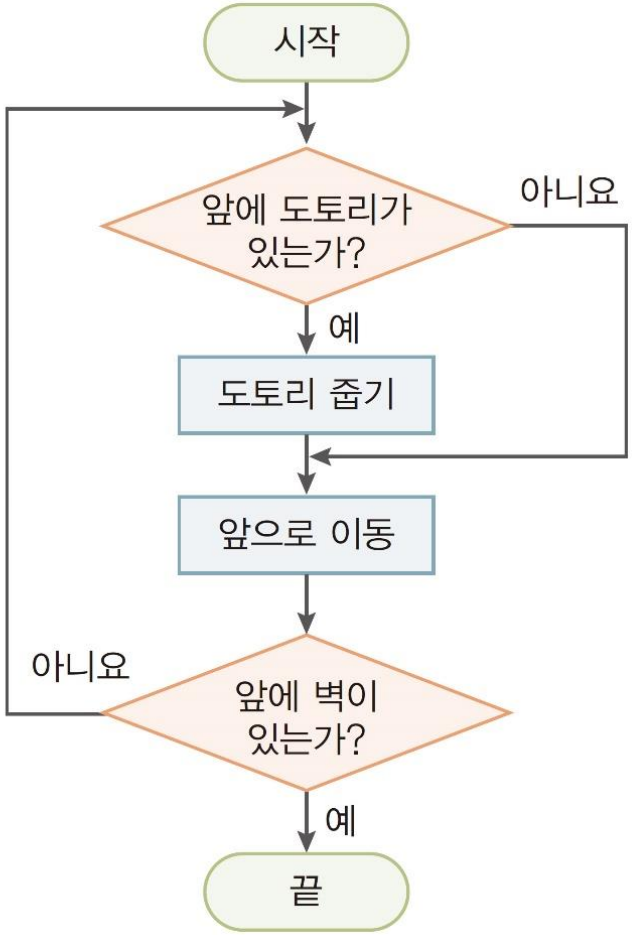
비트연산을 묻는 문제임.

정답률 81.78%

■ 도토리 줍기 알고리즘



(a) 밀폐된 공간의 다람쥐와 도토리



(b) 순서도 구현

그림 7-29 순서도로 도토리 줍기 알고리즘 설계

알고리즘 분석

알고리즘의 성능

- 알고리즘의 실행 단계가 복잡하거나 처리해야 하는 자료가 많을 경우

  알고리즘의 효율성은 프로그램의 성능에 중요한 역할을 함

- 알고리즘 성능 분석 방법
  - 알고리즘을 구현한 프로그램을 직접 실행하는 방법
  - 알고리즘의 실행 횟수 등 복잡도를 분석하는 방법
- 알고리즘의 복잡도
  - 알고리즘이 특정 기준에 따라 얼마나 빠르게 또는 느리게 실행되는지 나타내는 것
  - 시간 복잡도와 공간 복잡도를 분석하면 가장 효율적인 알고리즘을 선택할 수 있음

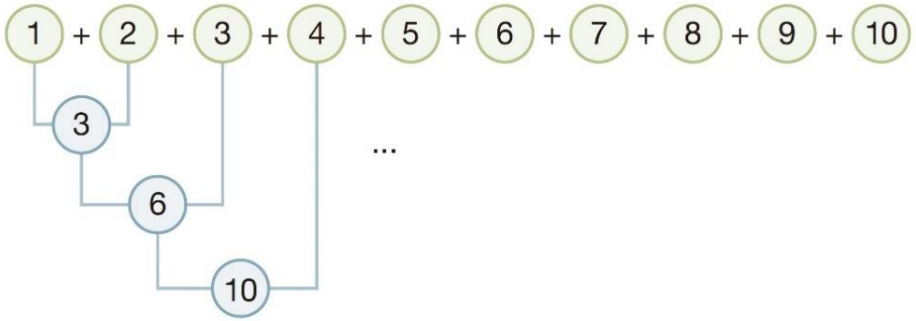
알고리즘 분석

시간 복잡도

- 알고리즘이 실행되어 종료될 때까지 어느 정도의 시간이 필요한지 측정하는 방법
- 실제 컴퓨터의 실행 시간을 측정하기는 어렵기 때문에 알고리즘의 실행문이 몇 번 실행되는지 횟수를 표시하여 측정

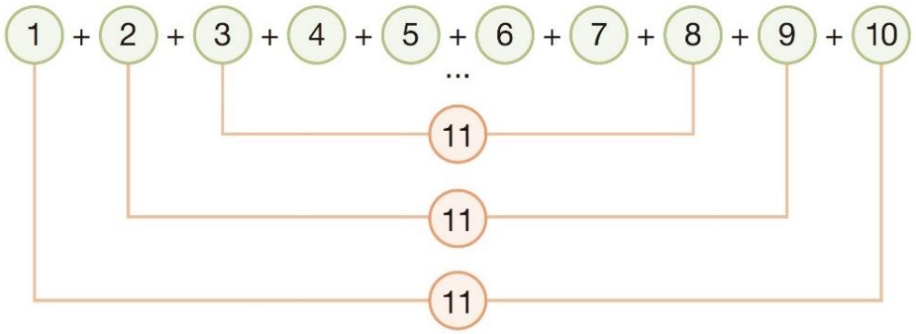
시간 복잡도

- '방법 2'가 '방법 1'보다 더 적은 연산으로 문제 해결하므로 시간 복잡도가 낮음



1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55

(a) 방법 1



11 × 5 = 55

(b) 방법 2

그림 7-31 1부터 10까지 수를 더하는 두 가지 알고리즘

여기서 잠깐 **빅오 표기법**

빅오 표기법(Big O notation)은 알고리즘의 시간 복잡도를 표현하는 방법이다. 수학에서는 함수 성질을 나타낼 때 사용하며, 컴퓨터에서는 입력한 값의 크기에 따라 알고리즘 처리 횟수가 얼마나 증가하는지 나타낸다.

다음 두 경우를 비교하면 O(n)이 $O(n^2)$보다 시간 복잡도가 더 좋은 알고리즘이라는 것을 알 수 있다.

- O(n): 알고리즘의 수행 횟수도 n만큼 커진다.
- $O(n^2)$: 알고리즘의 수행 횟수도 n^2만큼 커진다.

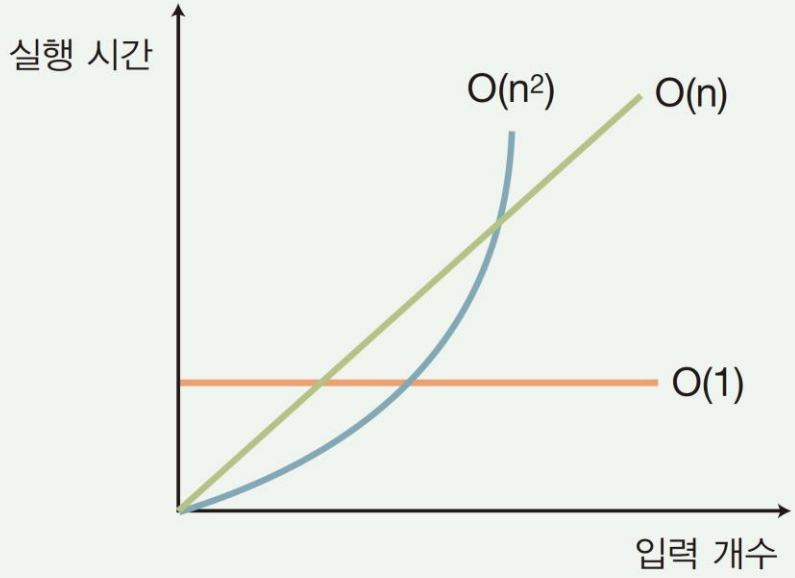


그림 7-32 빅오 표기법 그래프

알고리즘 분석

- **O(1)**
  - 알고리즘의 실행 시간이 입력되는 데이터의 크기에 상관없이 항상 일정한 상수 값(C)을 갖는 경우
- O(n)
  - 데이터의 크기에 비례하여 처리 시간이 증가하기에 선형(Linear) 함수
  - 배열에 있는 데이터를 순차적으로 검색하는 경우
- O(lon n)
  - 데이터의 크기가 두 배 늘어나더라도 복잡도가 1씩, 아주 조금씩 증가
  - 가장 효율적(Binary Search Tree)
- $O(n^2)$
  - 자료의 크기가 커지면 커질 수록 복잡도가 급격하게 상승(비효율적)
    (대표적인 예로 이중 for문)

알고리즘 분석

공간 복잡도

- 알고리즘이 문제를 해결하는 데 어느 정도의 저장 공간을 필요로 하는지 측정하는 방법
- 기억 장치 내의 공간을 얼마나 적게 사용하는지가 중요
- 저장 공간은 알고리즘이 사용하는 공간과 알고리즘에 입력되어 처리되는 자료 공간을 모두 포함

공간 복잡도

- '방법 2'의 알고리즘이 공간 복잡도가 더 낮아서 기억 장치의 공간을 효율적으로 활용

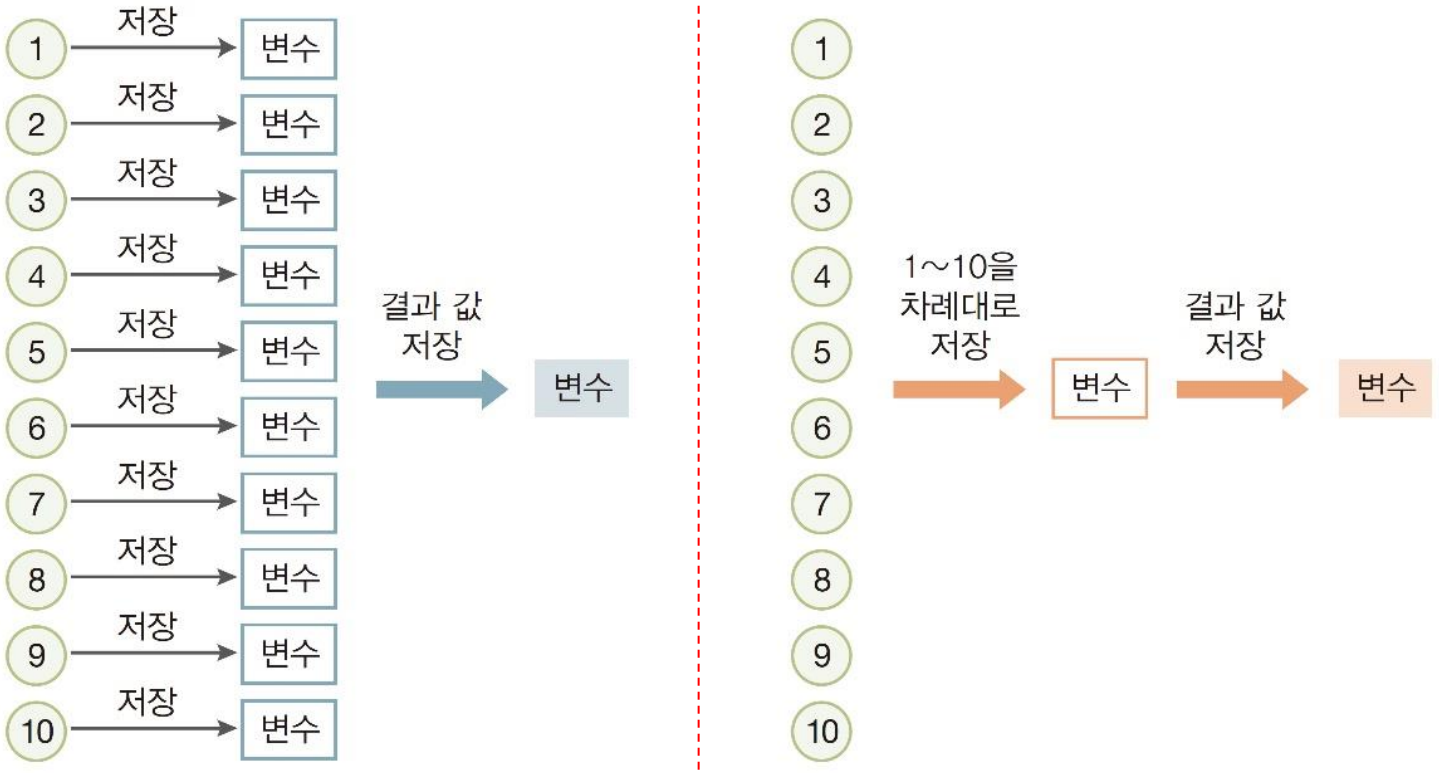


그림 7-33 1부터 10까지 수를 더하는 알고리즘에서 수를 저장하는 방법

알고리즘을 이용한 문제 해결 과정

■ 알고리즘을 이용한 문제 해결 과정

❶ 코드 작성 : 의사코드 형태의 알고리즘 설계

❷ 코드 검토 : 의사코드를 머릿속에서 시뮬레이션

❸ 입력 및 실행 : 프로그래밍 언어로 프로그램 작성 후 실행,
오류가 발생하면 ❶ 단계로 돌아가 코드 수정, ❶~❸ 단계 반복

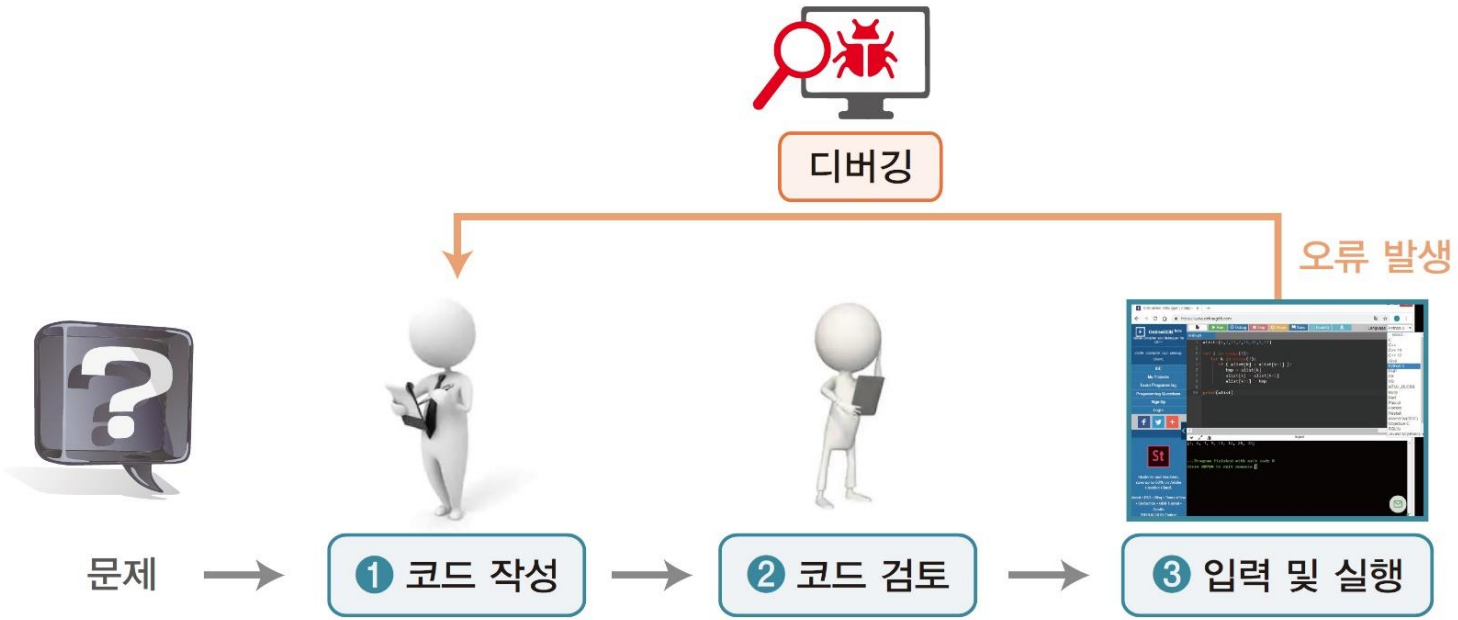


그림 7-36 알고리즘을 작성하여 문제를 해결하는 과정

알고리즘을 이용한 문제 해결 과정

오류

- 문법 오류(syntax error)
  - 여러 가지 오류 중 가장 간단, 어떤 지점에서 오류가 발생했는지 알려 주기 때문
- 논리 오류
  - 문법적인 오류가 없는데도 원하는 답이 나오지 않는 오류가 가장 고치기 어려움
  - 알고리즘을 잘못 만들었기 때문에 발생

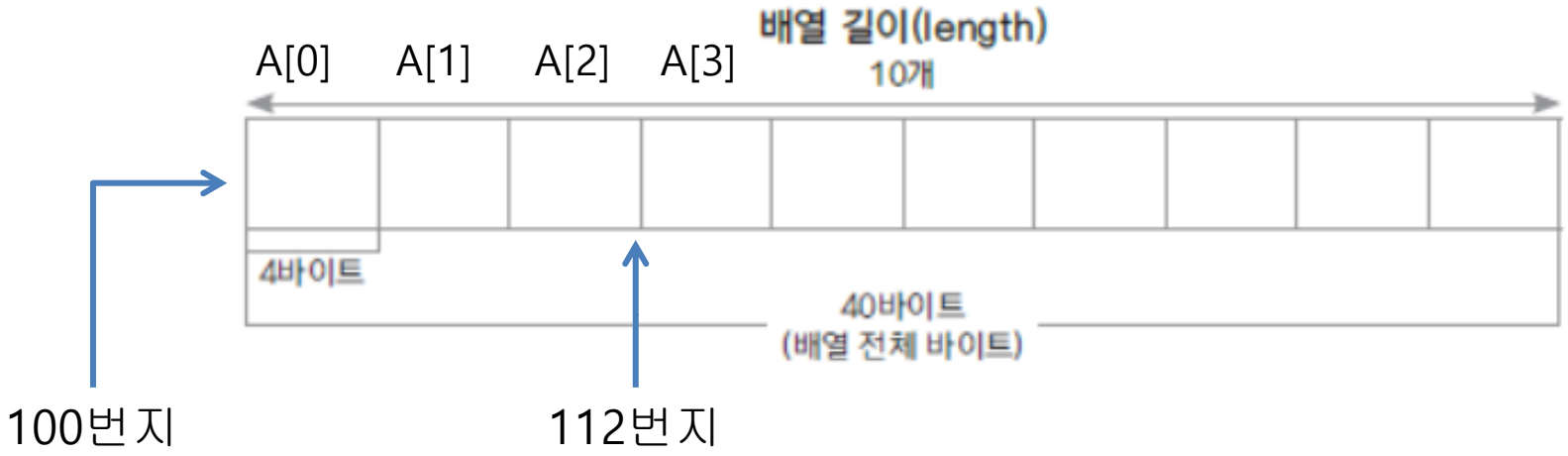
01. 선형자료구조

■ 자료구조란 무엇인가? (Data Structure)

- 프로그램에서 사용할 많은 데이터를 메모리 상에서 관리하는 여러 구현방법들
- 효율적인 자료구조가 성능 좋은 알고리즘의 기반이 됨
- 자료의 효율적인 관리는 프로그램의 수행속도와 밀접한 관련이 있음
- 여러 자료 구조 중에서 구현하려는 프로그램에 맞는 최적의 자료구조를 활용해야 하므로 자료구조에 대한 이해가 중요함

1. 배열(Array)

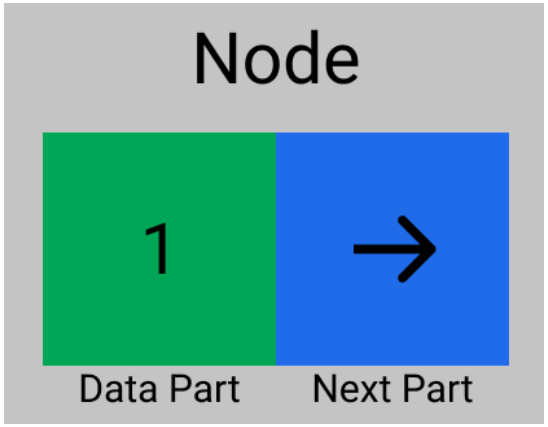
- ✓ **배열 (Array)** : 선형으로 자료를 관리, 정해진 크기의 메모리를 먼저 할당 받아 사용하고, 자료의 물리적 위치와 논리적 위치가 같음
- ✓ 수행속도 O(n)



100번지 + (4byte × 3)

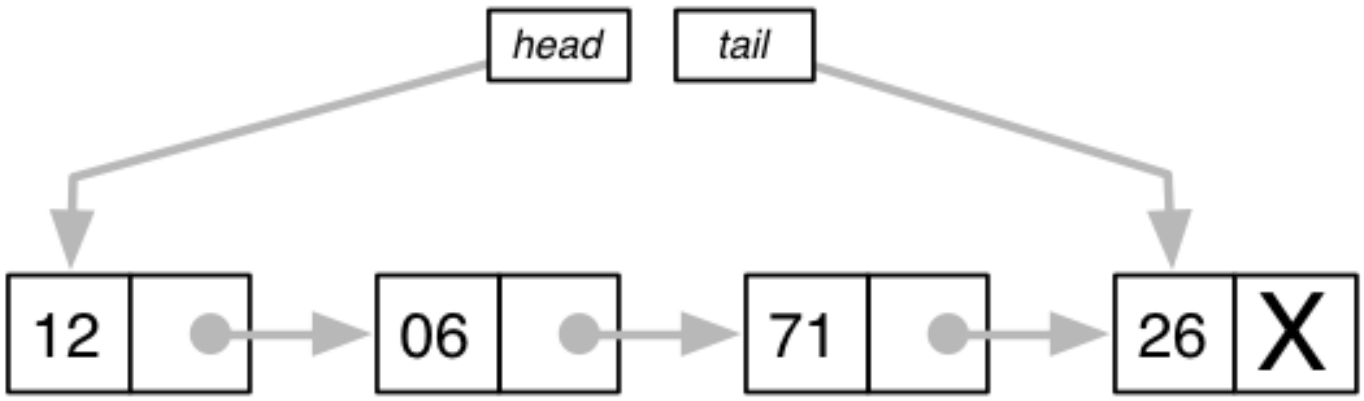
2. 링크드리스트(LinkedList)

- ✓ **연결 리스트 (LinkedList)** : 선형으로 자료를 관리, 자료가 추가될 때마다 메모리를 할당 받고, 자료는 링크로 연결됨. 자료의 물리적 위치와 논리적 위치가 다를 수 있음
- ✓ 수행시간 : O(1), 검색시간 : O(n)



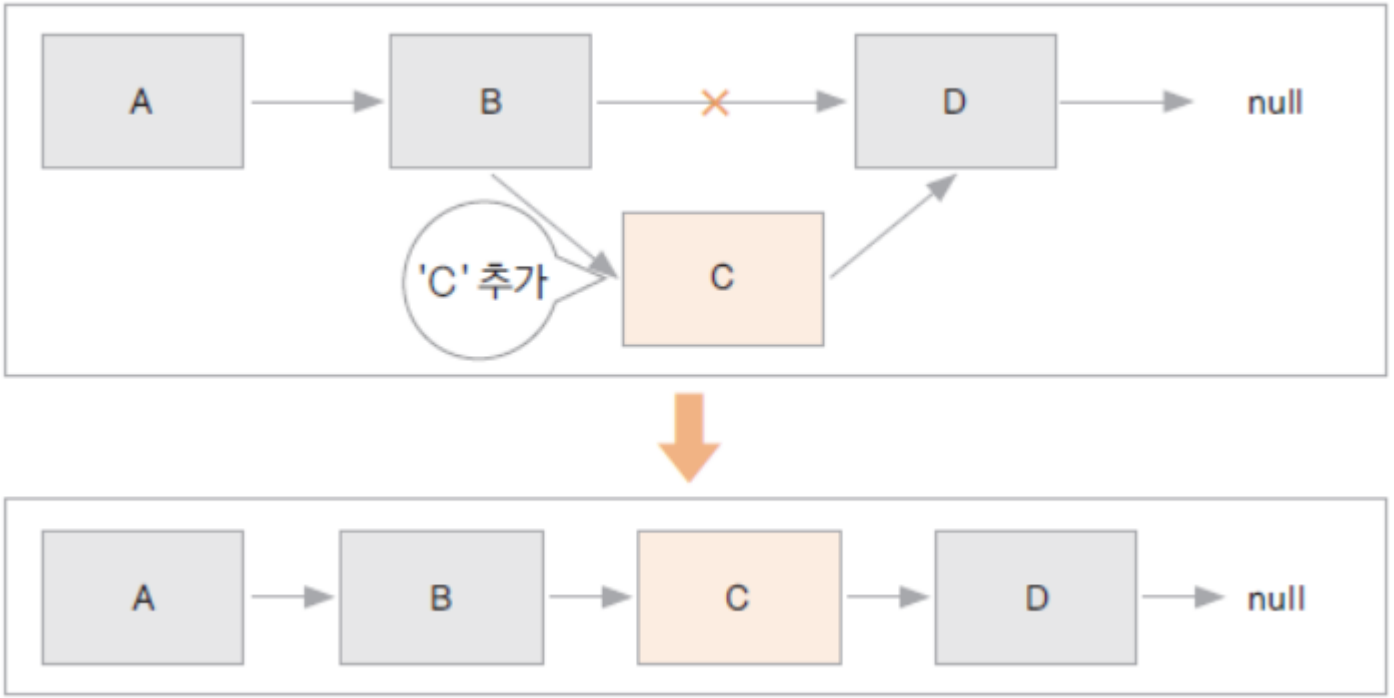
2. 링크드리스트(LinkedList)

자료구조	가져오기	추가하기	삭제하기
Linked List	O(n)	O(1)	O(1)



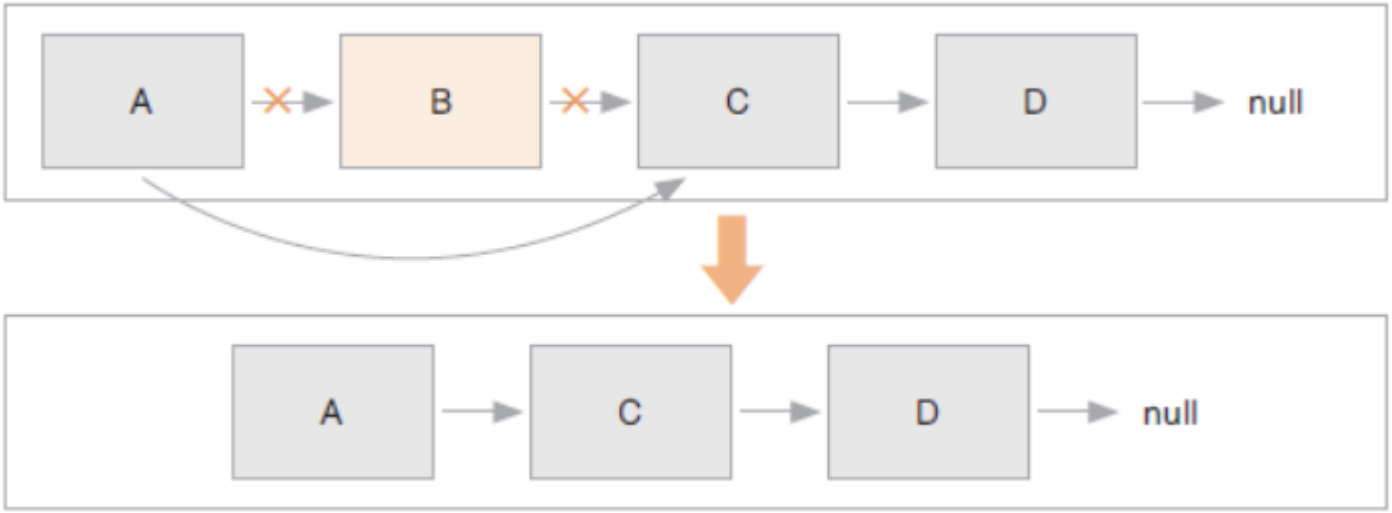
2. 링크드리스트(LinkedList)

리스트에 자료 추가하기



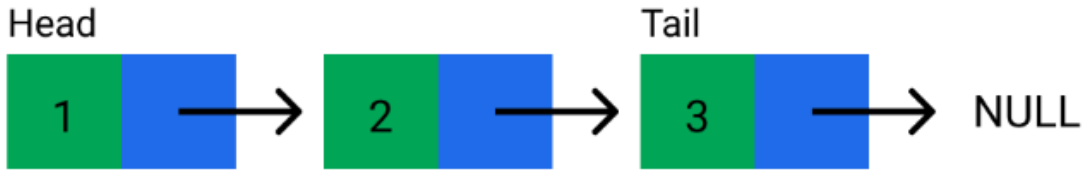
2. 링크드리스트(LinkedList)

리스트에 자료 삭제하기



2. 링크드리스트(LinkedList)

단일 연결 리스트(Singly-Linked List) : 요소(노드)가 오직 다음 요소로 연결된 것



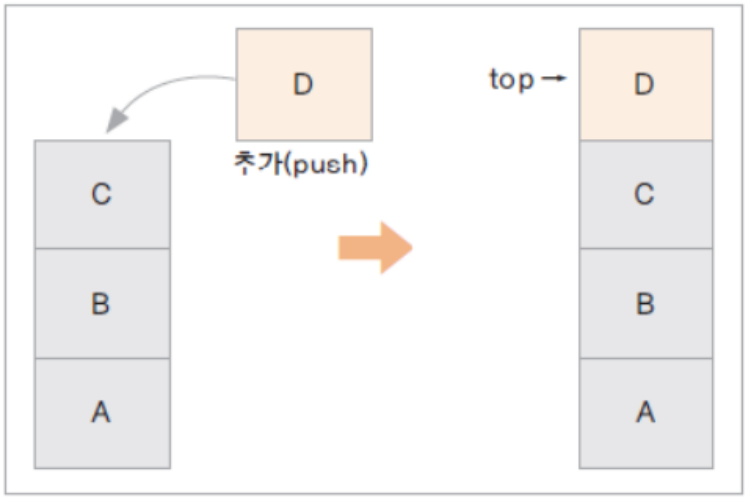
이중 연결리스트(Doubly-Linked List) : 이전 노드와 다음 노드가 레퍼런스를 포함하고 있는 리스트

3. 스택(Stack)

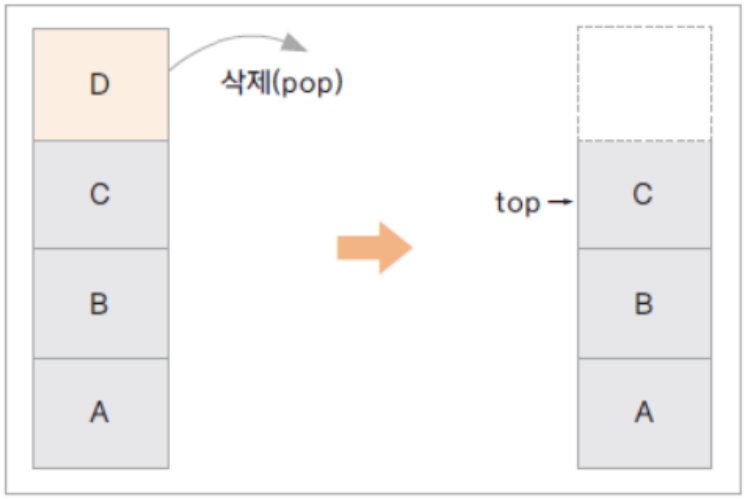
✓ 스택 (Stack) : 가장 나중에 입력 된 자료가 가장 먼저 출력되는 자료 구조

✓ (Last In First Out)

- 사용처 : 바둑/체스/장기(무르기), 실행 취소, 연산 등...

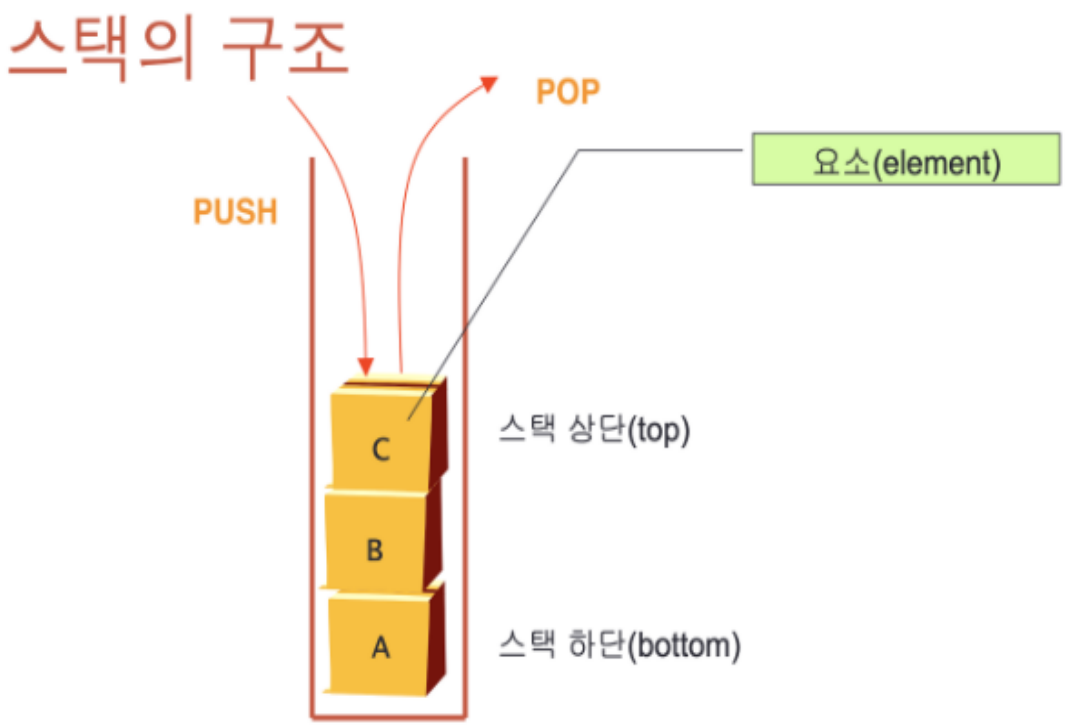


스택에 요소 추가(push)하기



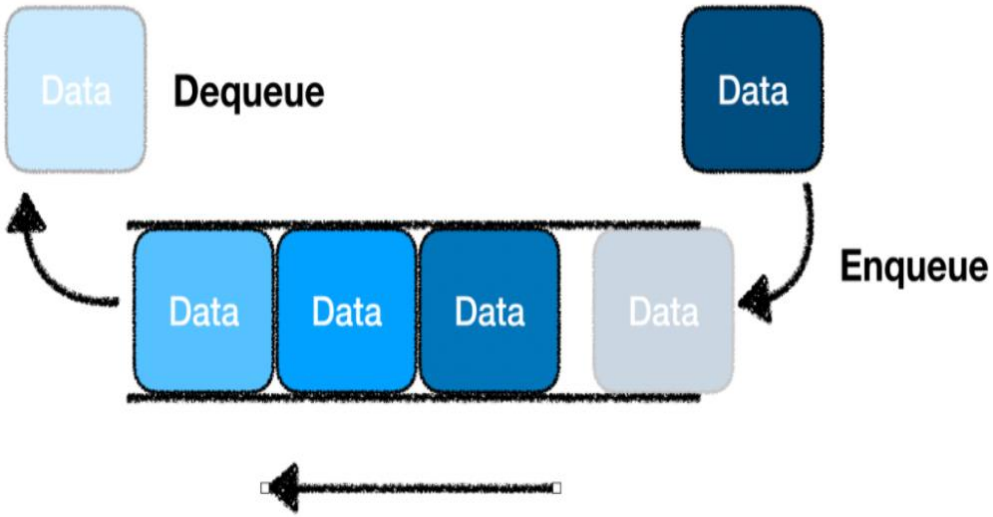
스택에서 요소 꺼내어(pop) 삭제하기

01. 선형자료구조

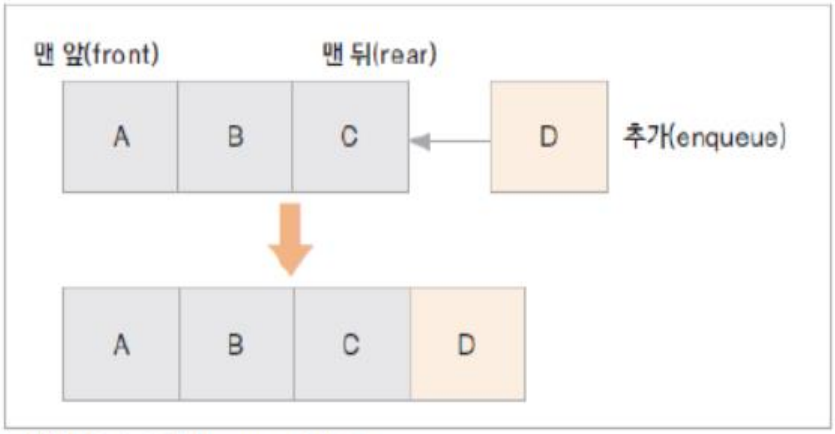


01. 선형자료구조

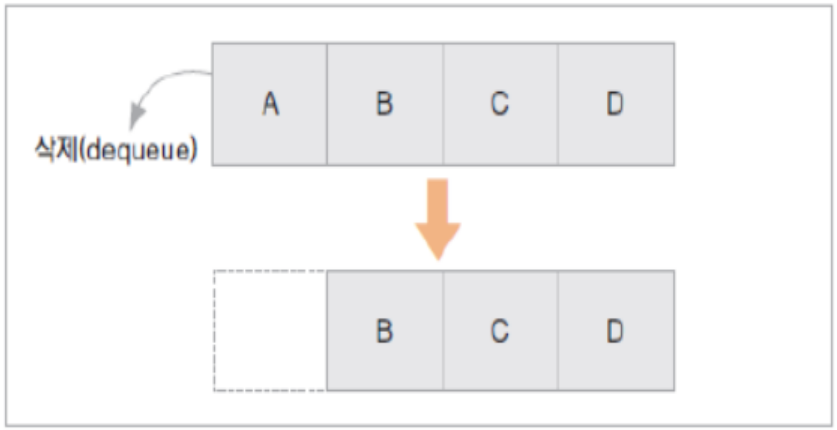
- ✓ 큐 (Queue) : 가장 먼저 입력 된 자료가 가장 먼저 출력되는 자료 구조 (First In First Out)
- ✓ 사용처 :
  - 콜센터 상담 대기 혹은 은행 대기 번호표
  - 지하철 탈 때 카드 찍는 경우(놀이공원 등 유사)



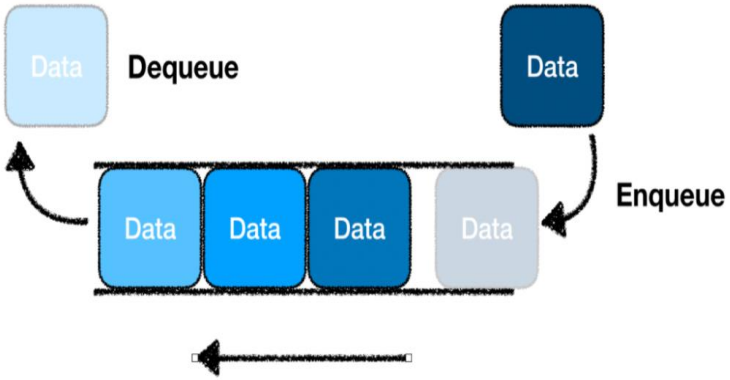
01. 선형자료구조



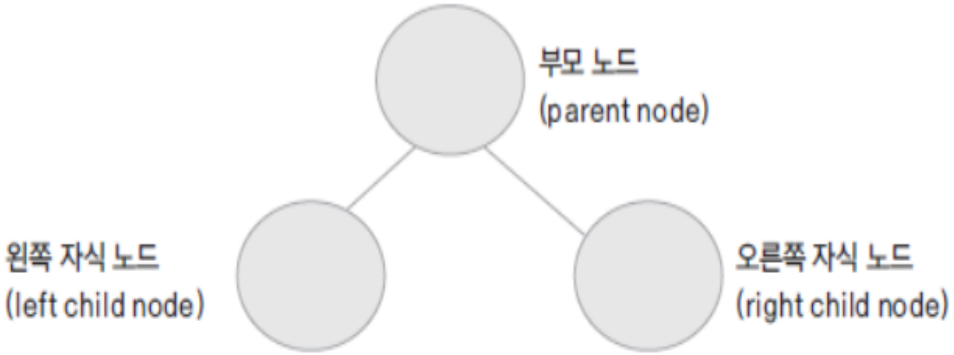
큐에서 요소 추가(enqueue)하기



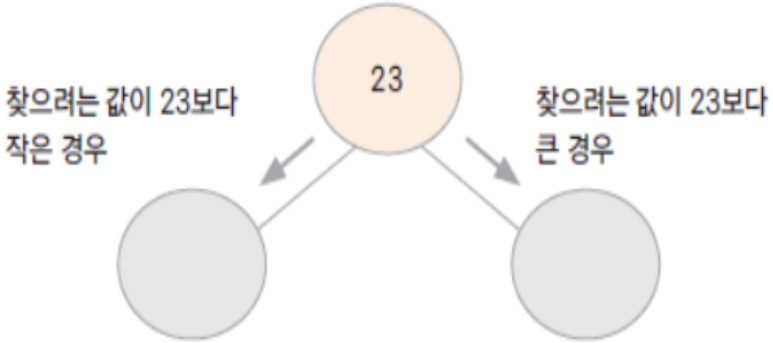
큐에서 요소 삭제(dequeue)하기



이진 트리 (binary tree) : 부모노드에 자식노드가 두 개 이하인 트리



이진 검색 트리 (binary search tree)



이진 트리 (binary tree) : 부모노드에 자식노드가 두 개 이하인 트리

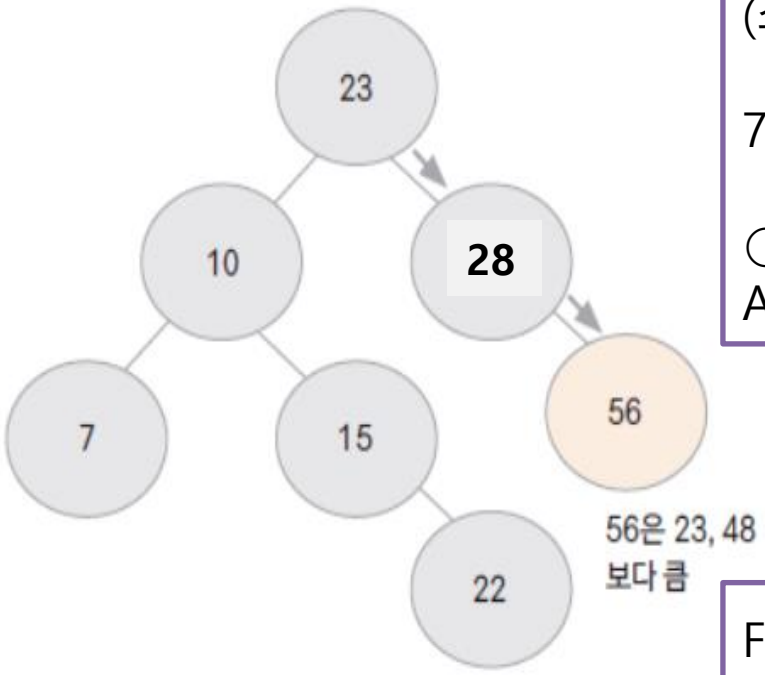
- 자료(key)의 중복을 허용하지 않음
- 왼쪽 자식 노드는 부모 노드보다 작은 값, 오른쪽 자식 노드는 부모 노드보다 큰 값을 가짐
- 자료를 검색에 걸리는 시간이 평균 log(n) 임
- inorder traversal 탐색을 하게 되면 자료가 정렬되어 출력됨
  (왼쪽 자식 노드 → 부모노드 → 오른쪽 자식노드 순으로 탐색)
  - 트리의 좌우가 바뀌면 역순

- jdk 클래스 : TreeSet, TreeMap (Tree로 시작되는 클래스는 정렬을 지원 함)

02. 비선형구조

■ 이진 트리 (binary tree) : 부모노드에 자식노드가 두 개 이하인 트리

예) [23, 10, 28, 15, 7, 22, 56] 순으로 자료를 넣을때 BST



in-Order 순회 시 정렬(부모노드가 가운데 있음)
(좌측자식노드 → 부모 → 우측자식노드

7 → 10 → 15 → 22 → 23 → 48 → 56

○ TreeSet, TreeMap Collection Framework 의 Algorithm으로 사용

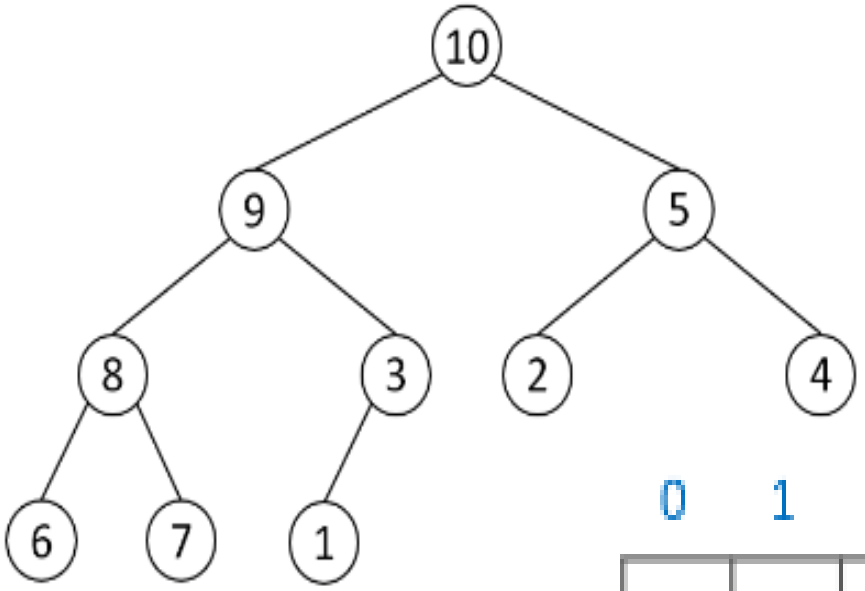
Full Binary Tree(정이진트리) 인 경우

노드의 수는 레벨 n을 기준으로 2^n-1

검색속도는 $\log_2 n$

- **힙(heap) : Priority queue를 구현 (우선 큐)**
  - 완전이진트리를 기본으로 항상 좌측 노드부터 채워진다. 중복 가능
  - Max heap : 부모 노드는 자식 노드보다 항상 크거나 같은 값을 갖는 경우
  - Min heap : 부모 노드는 자식 노드보다 항상 작거나 같은 값을 갖는 경우
  - heap 정렬에 활용 할 수 있음
  - 배열로 구현하기 쉬움.

02. 비선형자료구조

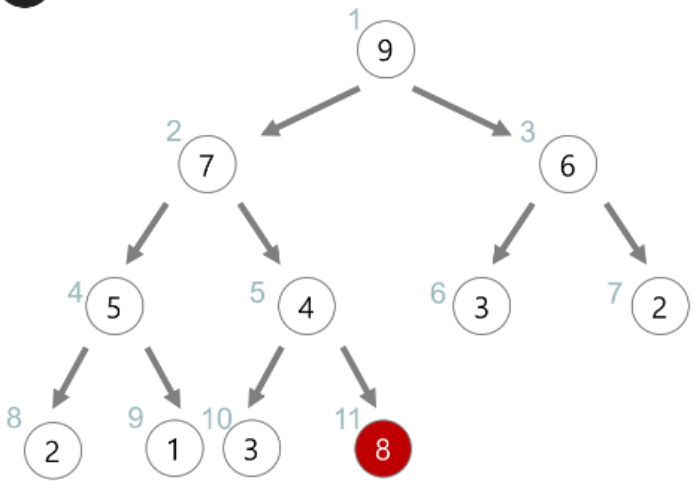


0	1	2	3	4	5	6	7	8	9
10	9	5	8	3	2	4	6	7	1

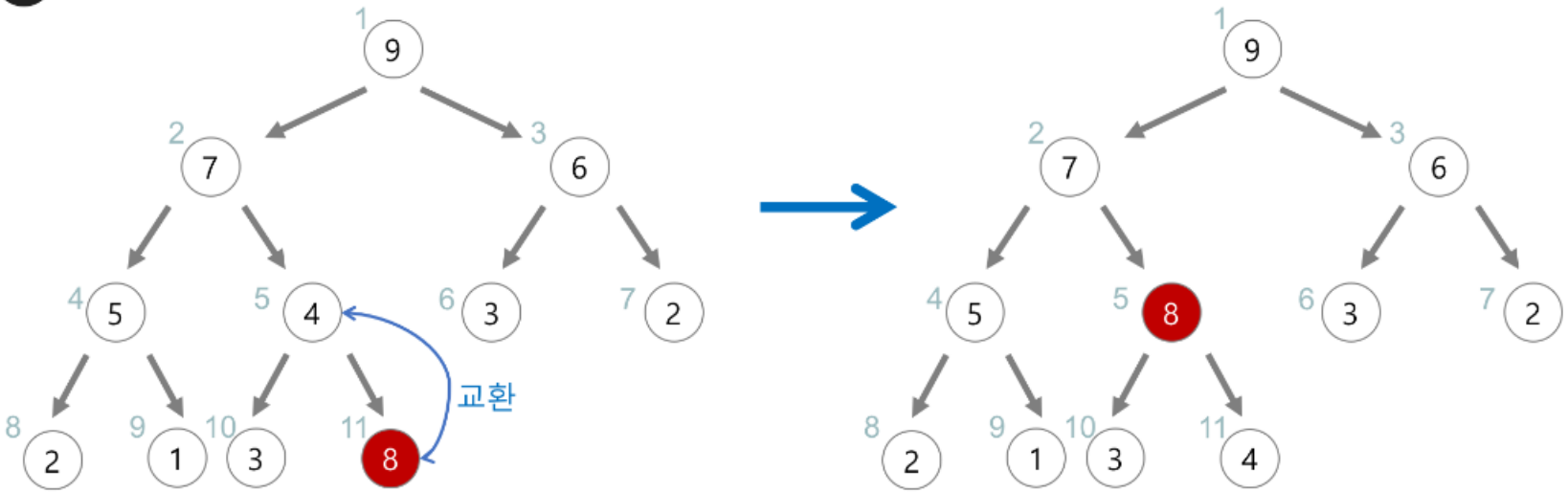
부모 노드	a[(i - 1) // 2]
왼쪽 자식 노드	a[i * 2 + 1]
오른쪽 자식 노드	a[i * 2 + 2]

Max Heap 의 삽입

1 인덱스순으로 가장 마지막 위치에 이어서 새로운 요소 8을 삽입

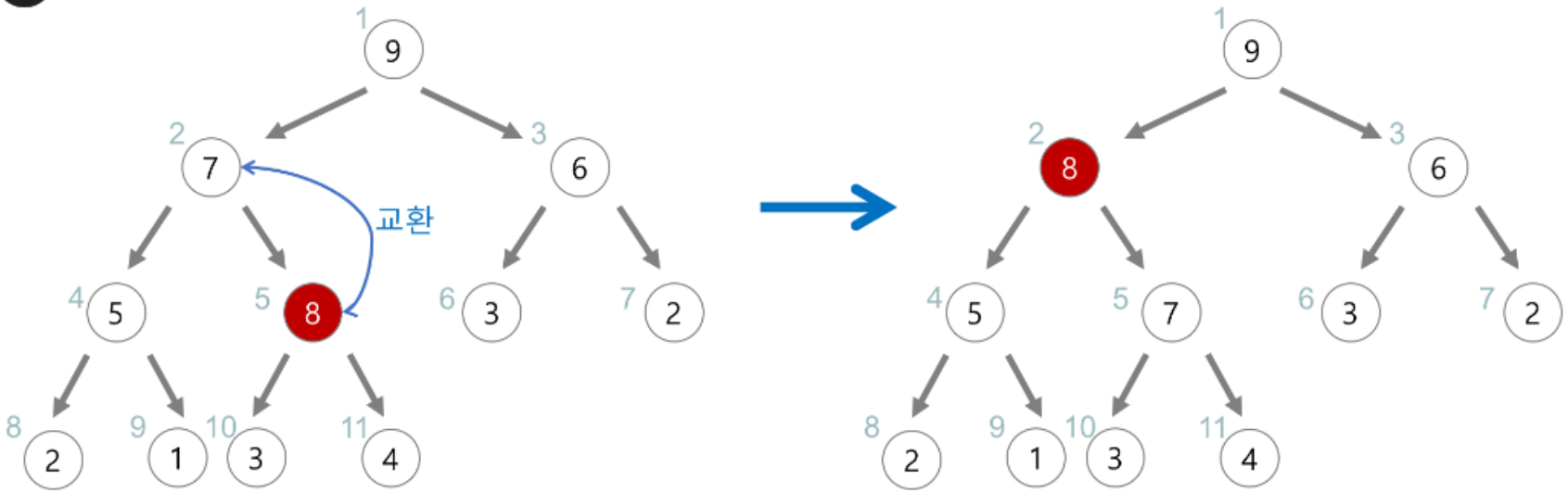


2 부모 노드 4 < 삽입 노드 8 이므로 서로 교환



Max Heap 의 삽입

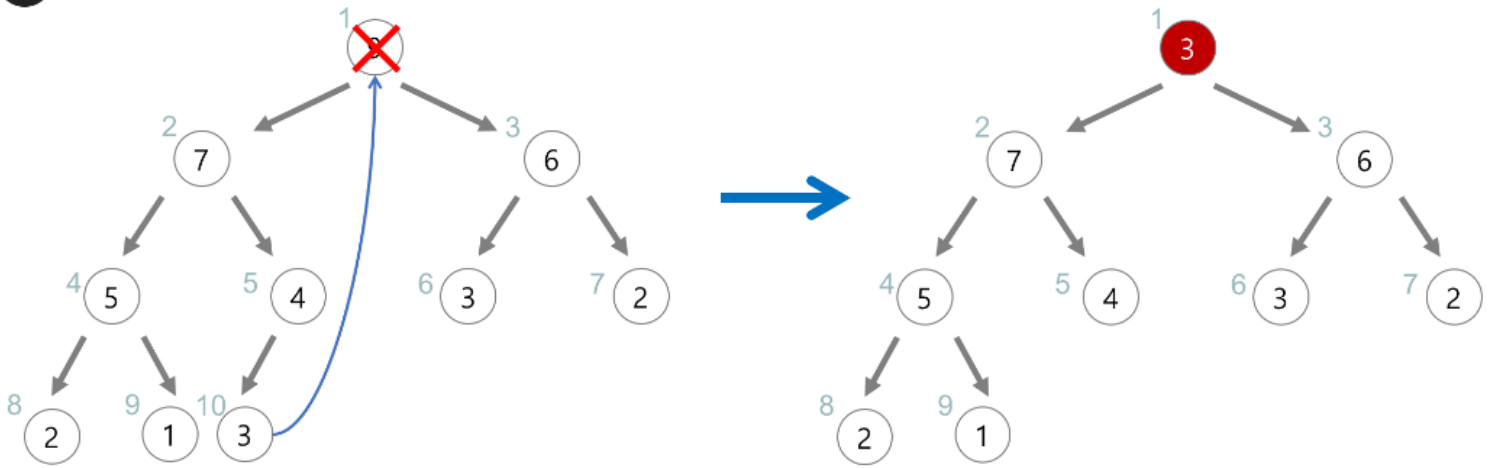
3 부모 노드 7 < 삽입 노드 8 이므로 서로 교환



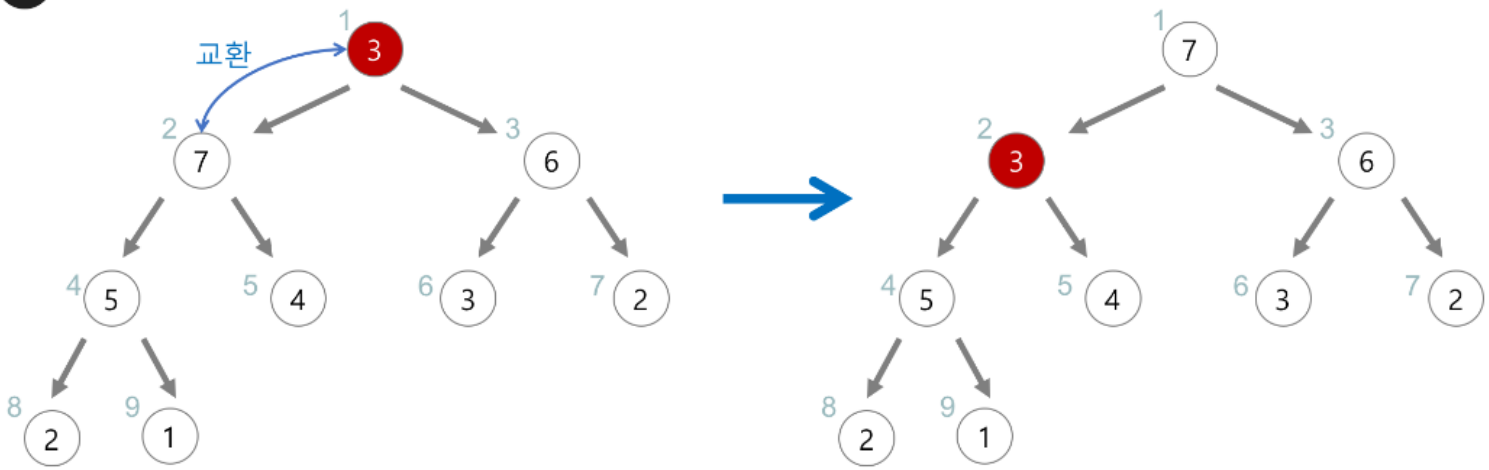
4 부모 노드 9 > 삽입 노드 8 이므로 더 이상 교환하지 않는다.

Max Heap 의 삭제

1 최댓값인 루트 노드 9를 삭제. (빈자리에는 최대 힙의 마지막 노드를 가져온다.)

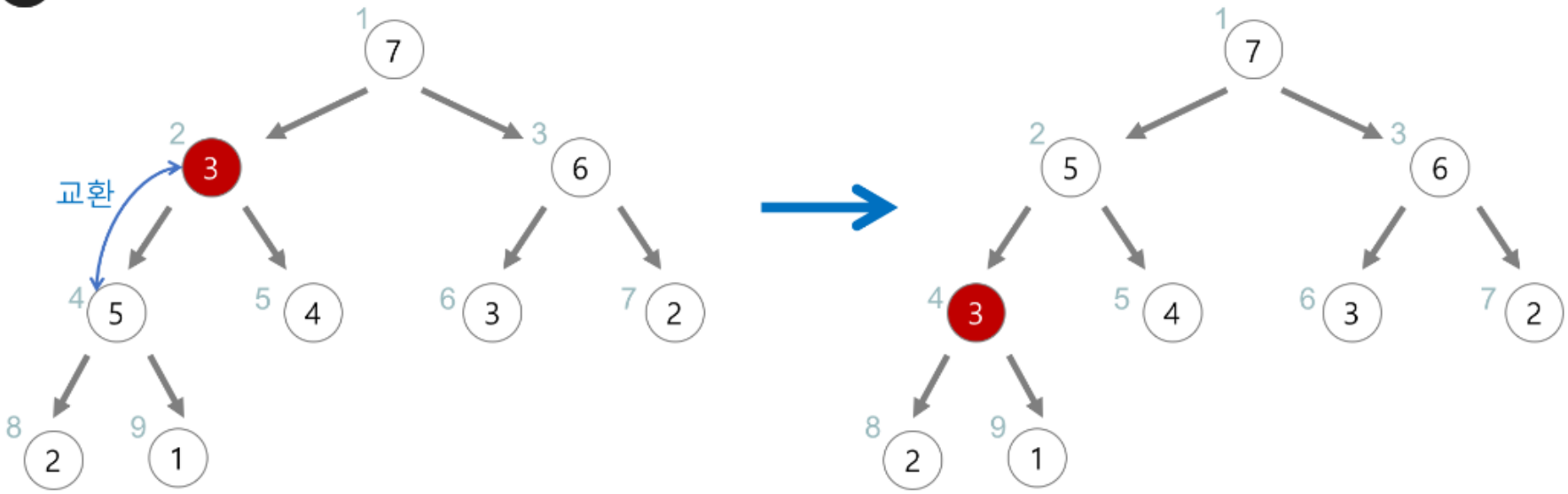


2 삽입 노드와 자식 노드를 비교. 자식 노드 중 더 큰 값과 교환. (자식 노드 7 > 삽입 노드 3 이므로 서로 교환)



Max Heap 의 삭제

3 삽입 노드와 더 큰 값의 자식 노드를 비교. 자식 노드 5 > 삽입 노드 3 이므로 서로 교환



4 자식 노드 1, 2 < 삽입 노드 3 이므로 더 이상 교환하지 않는다.

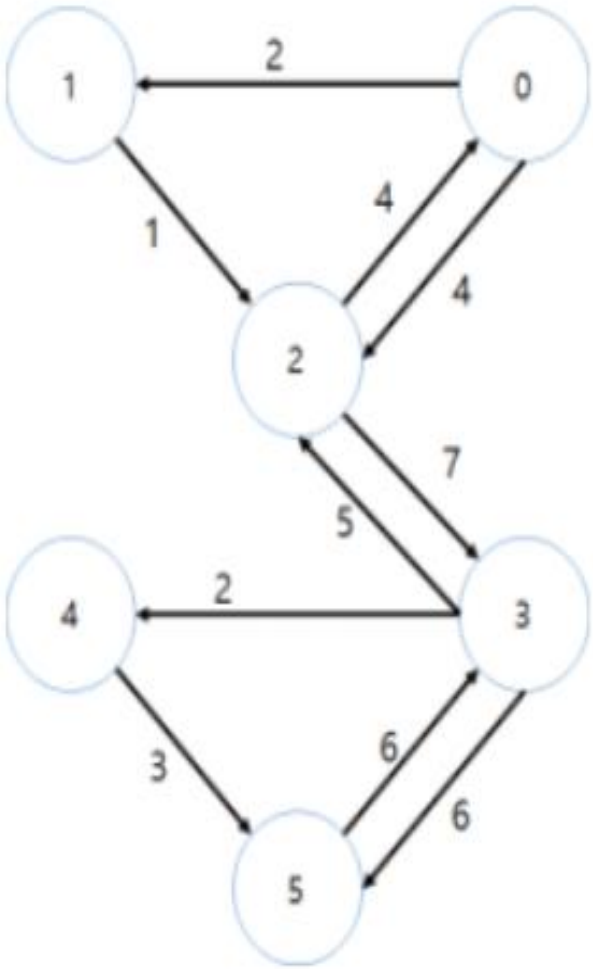
02. 비선형자료구조

■ 그래프 (Graph) : 정점과 간선들의 유한 집합 G = (V,E)

- 정점(vertex) : 여러 특성을 가지는 객체, 노드(node)
- 간선(edge) : 이 객체들을 연결 관계를 나타냄. 링크(link)
- 간선은 방향성이 있는 경우와 없는 경우가 있음
- 그래프를 구현하는 방법 : 인접 행렬(adjacency matrix), 인접 리스트(adjacency list)
- 그래프를 탐색하는 방법 : BFS(bread first search), DFS(depth first search)

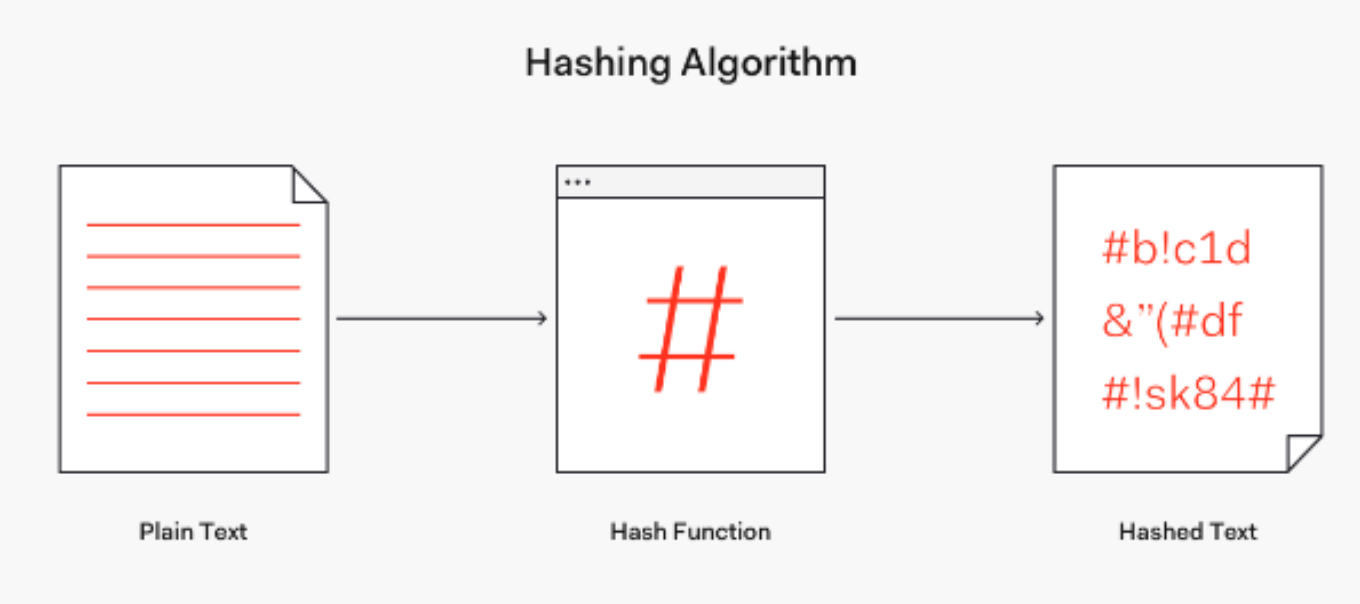
그래프

그래프의 예)



02. 비선형자료구조

- **해싱 (Hashing) : 자료를 검색하기 위한 자료 구조**



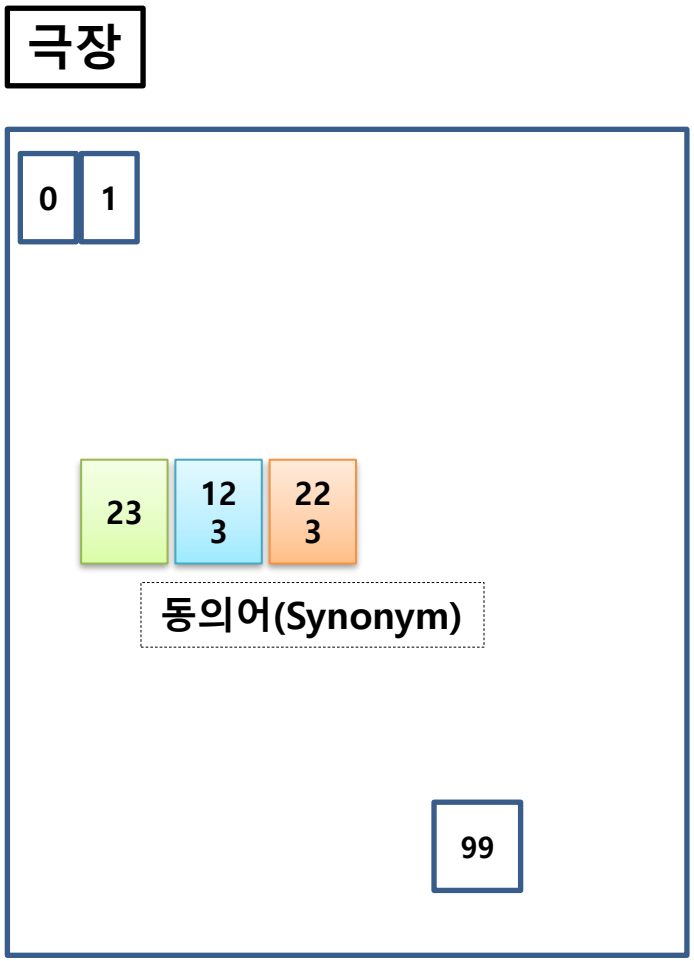
02. 비선형자료구조

■ 해싱 (Hashing) : 자료를 검색하기 위한 자료 구조

- 검색을 위한 자료 구조
- 키(key)에 대한 자료를 검색하기 위한 사전(dictionary) 개념의 자료 구조
- key는 유일하고 이에 대한 value를 쌍으로 저장
- index = h(key) : 해시 함수가 key에 대한 인덱스를 반환해줌 해당 인덱스 위치에 자료를 저장하거나 검색하게 됨
- 해시 함수에 의해 인덱스 연산이 산술적으로 가능 O(1)
- 저장되는 메모리 구조를 해시테이블이라 함
- 보통 해시테이블이 75% 이상 차 있을 시에 재구성 함.
- jdk 클래스 : HashMap, Properties

01. 여러 가지 자료구조에 대해 알아봅시다.

■ 해싱 (Hashing) : 자료를 검색하기 위한 자료 구조



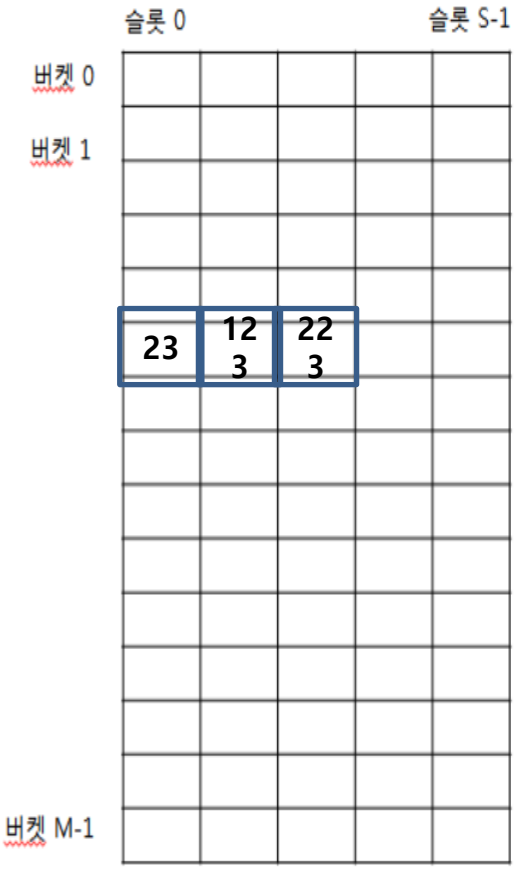
- 극장 좌석 자리는 총 100개
- 키 값 : 극장 좌석번호(0번~99번)
- 극장 주인이 표를 300장 팖
- 특정 위치를 소유한 사람이 온다면,
- 표 자리를 찾을 때 100으로 나눈 나머지를 구해 자리를 배정 함.

- 이 때 사용하는 나머지를 구하는 알고리즘이 Hash function임.
- 동일한 값이 들어오면 충돌(Collision)
- 여러 가지 알고리즘이 있으나, 그냥 옆에 앉으세요... 찾을 때 순서대로 물어봄.
- 충돌 난 값의 집합은 synonym(동의어)

01. 여러 가지 자료구조에 대해 알아봅시다.

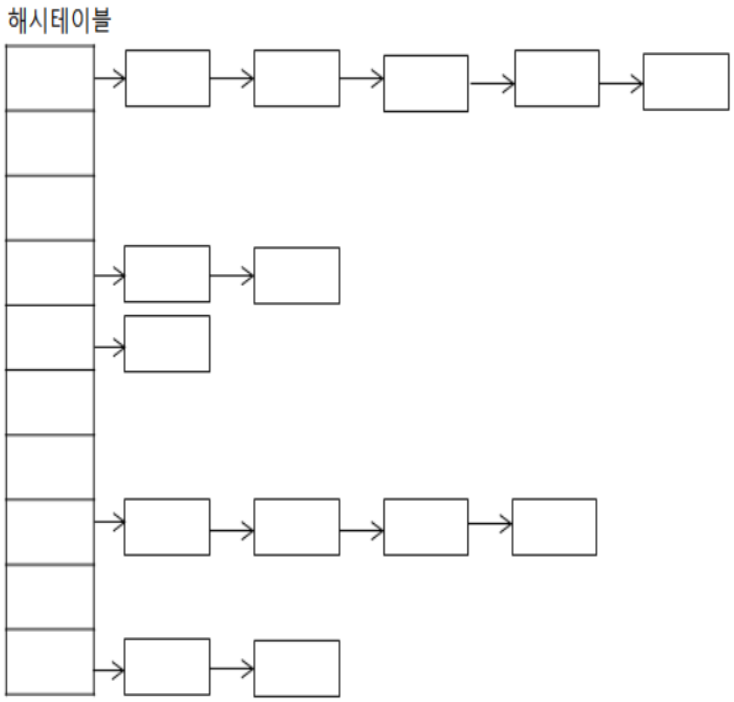
■ 해싱 (Hashing) : 자료를 검색하기 위한 자료 구조

배열을 사용한 해시테이블



링크드리스트를 사용한 해시테이블

- 체이닝 -



Thank you!