합격자가 되는 정리 NOTE

시리즈 C++ 편

이름

Coding Test Study Note

* 0장

타인의 풀이 보기 → 질문↔답변, 커뮤니티, 저자 소통 ... etc

옷코딩에 집착 X

나만의 TC 추가하는 연습하기 → 예외처리 예방

공부 습관 + 꿀팁

① 기록하기

② 시험 연습하기 → 타이머 필수

③ 짧은 시간 공부해서 합격한다는 무리한 마인드 버리기

④ 나만의 언어로 배운 내용 요약하기

Coding Test Study Note

* 1장

프로그래밍 언어, 이것만 기억하라

변수 함수 자료형 조건문 + 반복문

문제 욱노과 시간 낭비하기 X ⟶ 60%의 시간은 문제 분석에 쓰기

"코테 시간은 보통 2~4시간 주어짐"

① 문제를 쪼개서 분석하기

② 제약 사항 파악 + TC 추가하기

③ 입력값 분석하기 ex) 입력 데이터 개수 100만개 ⟶ $O(N^2)$ X

④ 그리디하게 접근할 때는 근거를 명확히 ⟶ 논리로 따져보기

ex) 동전 거스름돈 문제 ⟶ 그리디하게 생각하기 쉬움 < 틀림

⑤ 데이터 흐름이나 구성 파악하기

ex) 데이터 삽입이나 삭제가 빈번 + Max / Min 구하기 ⟶ Heap

ex) 데이터 범위의 격차가 크다면 ⟶ 데이터 값을 인덱스로

사용 시 메모리 초과 주의!

* 1강

의사 코드 (Pseudo code) — 이렇게 써라

영어 점수 입력

60점이 넘는지?

넘으면 → PASS

아니면 → FAIL

① 프로그래밍 언어로 적기 X

② 문제 해결 순서로

③ 충분히 Test

* 3장

시간 복잡도 (Time Complexity)

- 알고리즘의 성능을 나타내기 위한 방법 → 입력 ∞ 연산 횟수
- 결론만 말하자면 Big-O Notation으로 적고

  Big-O Notation은 최악의 연산 횟수를 가정한 표기법임
- 또 결론만 말하자면 코드에서 연산 포인트를 수식화해서 최고차항만 남기면

  Big-O Notation 끝이 끗!

ex)

```cpp
#include <iostream>

using namespace std;

void solution(int n) {
  int count = 0;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      count += 1;
    }
  }
  for (int k = 0; k < n; ++k) {
    count += 1;
  }
  for (int i = 0; i < 2 * n; ++i) {
    count += 1;
  }
  for (int i = 0; i < 5; ++i) {
    count += 1;
  }
  cout << count << endl;
}

int main() {
  solution(6);
  return 0;
}
```

solution 함수의 연산 point

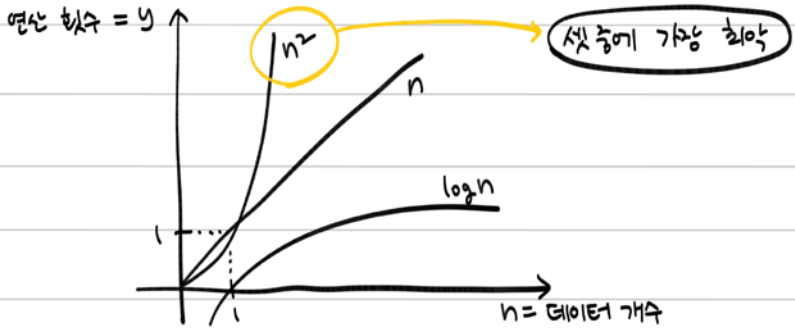
n^2 + n + $2n$ + 5

→ 최고차항 n^2만 남겨 $O(n^2)$

* 3장

Why 최고차항만 남길까?

'수식의 어떤 요소가 최악의 연산에 가장 많이 기여하는지 고려한 것'



시간 복잡도와 연산 횟수의 관계

O	N의 가능 범위
O(N!)	10
O(2^N)	20~25
O(N^3)	200~300
O(N^2)	3,000~5,000
O(NlogN)	100만
O(N)	1,000~2,000만
O(logN)	10억

비교

*보는 법

O(N) 알고리즘을 생각…

데이터 개수가 1000만이면?

→ OK

∴ O(N)은 10000만 수용

* 4장

컨테이너 & 알고리즘 정리 (5)

컨테이너

	vector	배열과 유사 연속된 메모리 공간에 원소 저장	push_back → O(1) pop_back, at, operator[] } O(1) insert, erase → O(N)
	set	키만 저장하는 정렬된 트리 중복키 없음	insert, erase, find } O(logN)
	map	키,값을 저장하는 정렬된 트리	
unordered	set	해시 테이블을 사용 키만 저장 정렬 X	insert, erase, find } O(1) 평균, O(N) 최악
unordered	map	해시 테이블을 사용 키,값 저장 정렬 X	

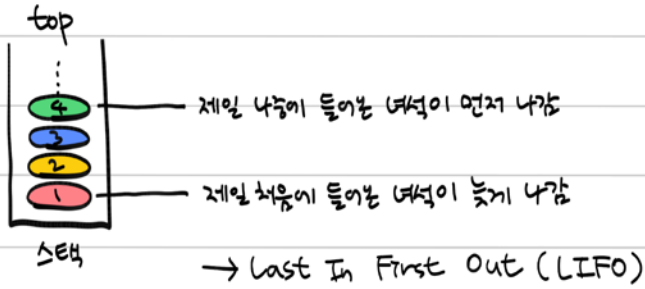
컨테이너 & 알고리즘 정리 (5) cont.

알고리즘

count	컨테이너의 특정 값이 등장하는 횟수 계산	O(N)
sort	지정된 범위 정렬	O(NlogN)
next_permutation	지정된 범위의 다음 순열 생성	O(N×N!)
unique	중복 요소 제거, 실제 컨테이너는 그대로 두고 해당 범위를 반환	O(N)
binary_search	정렬된 범위에서 특정 값을 이진 탐색으로 찾기	O(logN)

* 6장

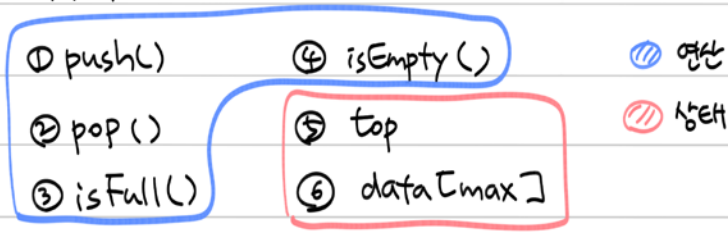
스택 (stack)



ADT란? (Abstract Data Type)

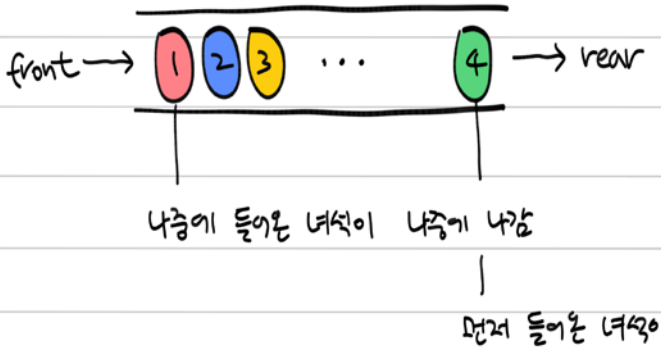
- 추상 자료형이라 부름
- 구현은 되어 있지 않고, 인터페이스만 있는 자료형

스택의 ADT



* 7장

큐 (Queue)



큐의 ADT

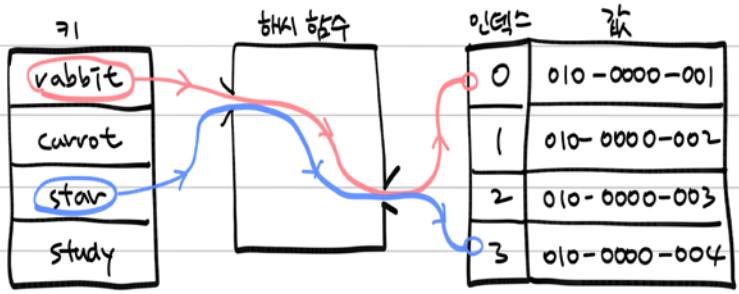
① push ( )
② pop ( )
③ isFull( )
④ isEmpty( )
⑤ front
⑥ rear
⑦ data[max]

연산
상태

* 8장

해시 (Hash)

- 키를 해시 함수에 통과시켜 인덱스를 얻은 다음 값에 접근



① 단방향으로 동작 (키 → 값)

② O(1)으로 값에 접근 (충돌이 없으면)

해시 함수 (Hash function)

① 해시 함수가 반환한 값은 해시 테이블의 크기를 넘으면 X

② 충돌을 최소화해야 함

* 8장

나눗셈법 (division method)

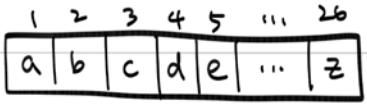
- 키를 소수로 나누는 방식
- 소수를 이용해야 충돌이 적음
- 해시 테이블의 크기는 나누는 소수 K

곱셈법 (multiplication method)

- $hash(x) = (((x \times A) \bmod 1) \times m)$
  - A: 황금비
  - m: 최대 버킷수

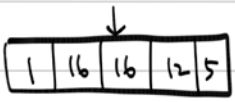
문자열 해싱

- 문자열을 해싱하기 위해 만든 방법



문자표

키



변환값

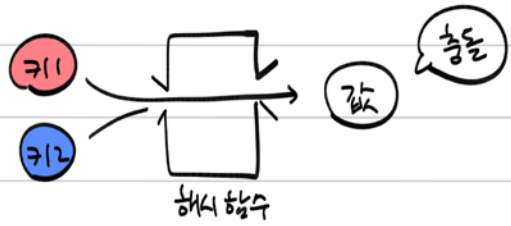
$1 \times 31^0 + 16 \times 31^1 + \cdots + 5 \times 31^4 = 4,990,970$ 을 m으로 나눔

* 8장

해시 충돌 해결

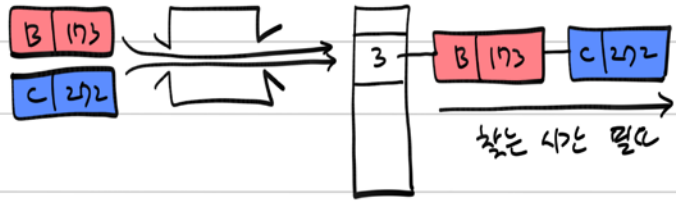
- 충돌? 키가 해시 함수를 통과했을 때 서로 같은 경우

  → 해결하는 방법이 다양함



① 체이닝

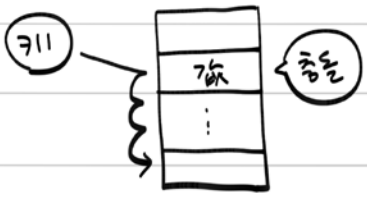
- 충돌난 값을 리스트 형식으로 저장



* 8장

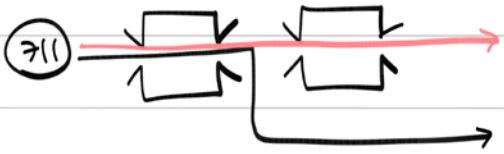
② 개방 주소법

- 빈 버킷을 찾아 저장하는 방법 (군집이 생길 수 있음)



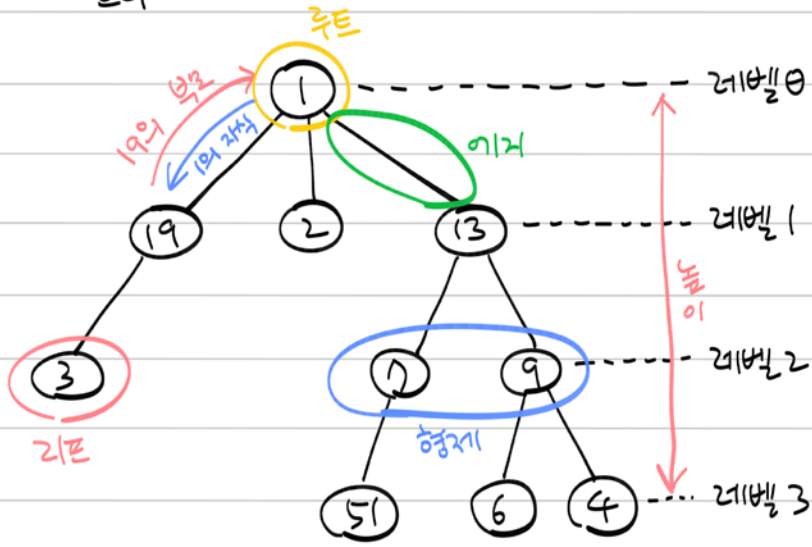
③ 이중 해싱

- 해시 함수를 2개 사용하는 방법
- 2번째 해시 함수가 충돌이 나면 해결하는 역할



✳ 9장

트리



1 : 노드, 트리를 구성하는 기본 단위

루트 : 루트, 최상위 노드

—— : 에지, 노드를 잇는 선

3 : 리프, 최하위 노드 (51, 6, 4 → 리프)

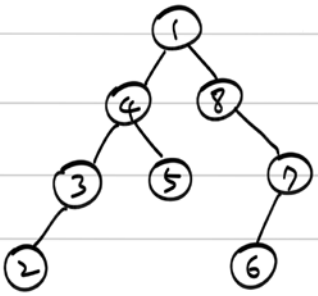
레벨 : 노드의 깊이 (노드 중심 표현)

높이 : 트리의 높이 (트리 중심 표현)

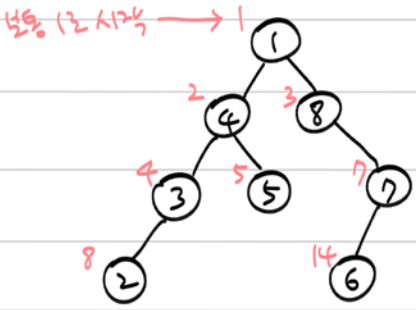
✳ 9장

이진 트리

- 자식 노드의 개수가 최대 2개인 트리



① 배열로 이진 트리 구현하기

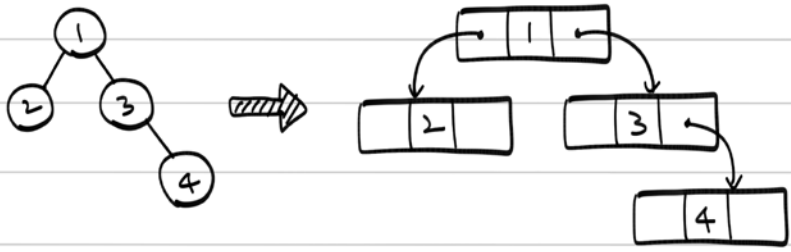


- left child 인덱스
  = parent X 2
- right child 인덱스
  = parent X 2 + 1

✳ 9장

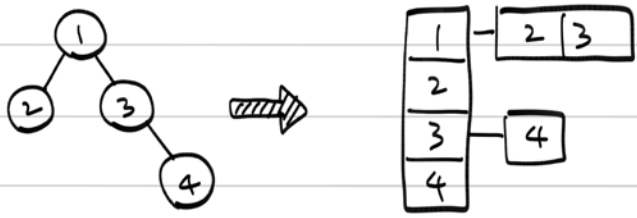
트리 포인터로 구현하기

- 노드가 자식 노드 2개를 가리키는 형태
- 메모리 낭비가 적은 구현 방식



트리 인접 리스트로 구현하기

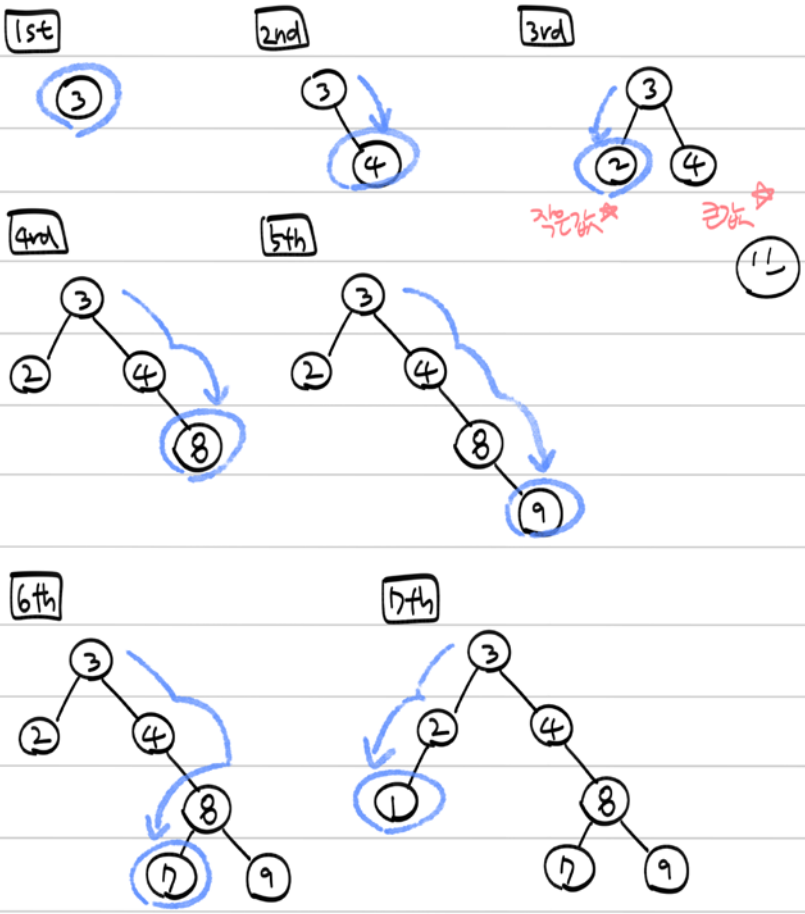
- 자식을 리스트로 갖는 형태
- 2개 이상의 자식도 표현할 수 있음



✳ 9장

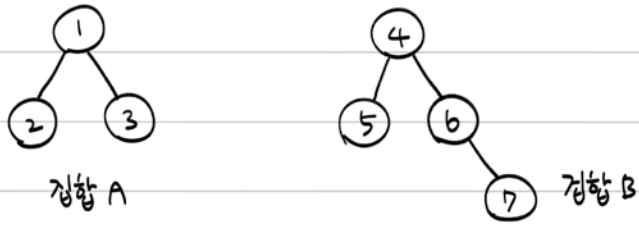
이진 트리 구축 과정 요약 (탐색도 동일함)

3 4 2 8 9 7 1 을 이진 트리로!

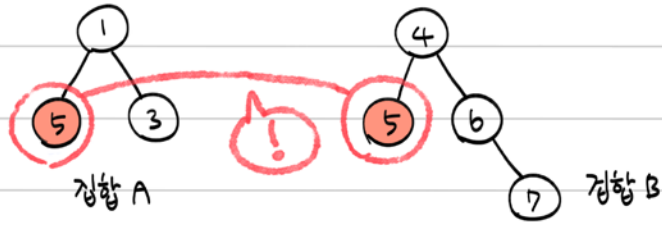


✳ 10장

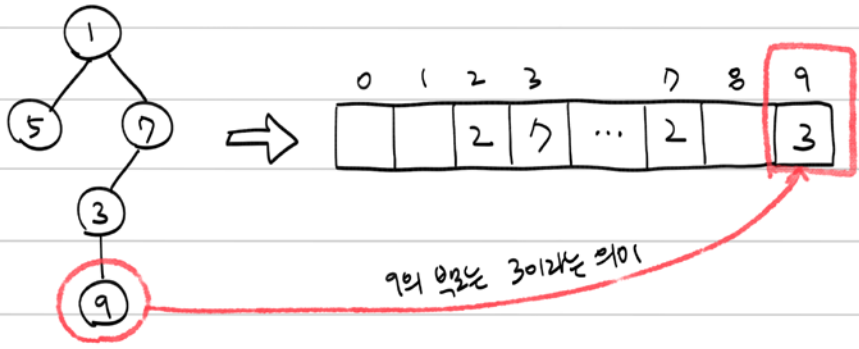
상호배타적 집합



상호배타적이지 않은 집합



배열로 집합 표현하기



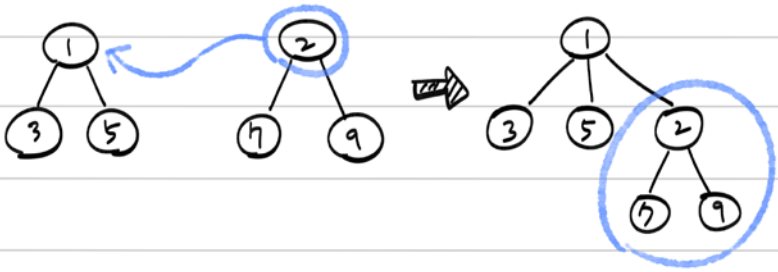
✳ 10장

유니온 파인드 알고리즘

= 유니온 연산 + 파인드 연산

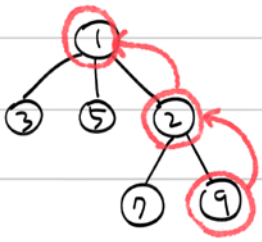
유니온 연산

- 다른 한쪽 집합의 루트를 하위로 포함시킴



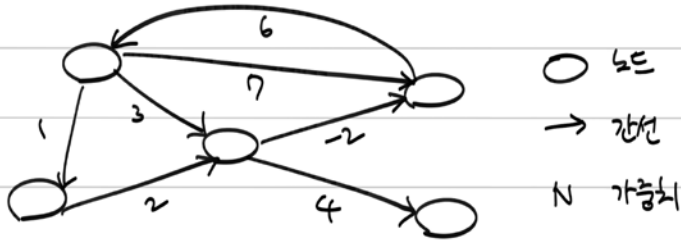
파인드 연산

- 부모 노드가 루트 노드일때까지 탐색
- 의의는 파인드 연산의 결과가 같은가? 같은 집합에 속한 노드!



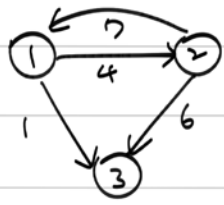
* 11장

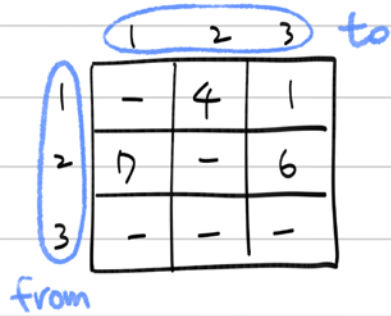
그래프 graph



*지금은 방향이 있는 그래프

인접 행렬 그래프

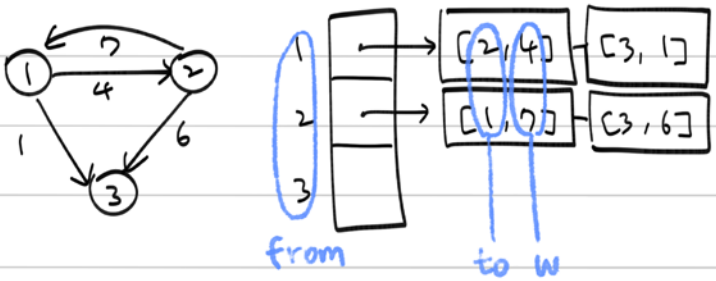




- 단점 1. 희소 그래프 표현 시 공간 낭비
- 단점 2. 노드 값의 차이가 크면 공간 낭비
- 장점 1. A-B 연결하는 간선 정보 O(1)에 알 수 있음

* 11장

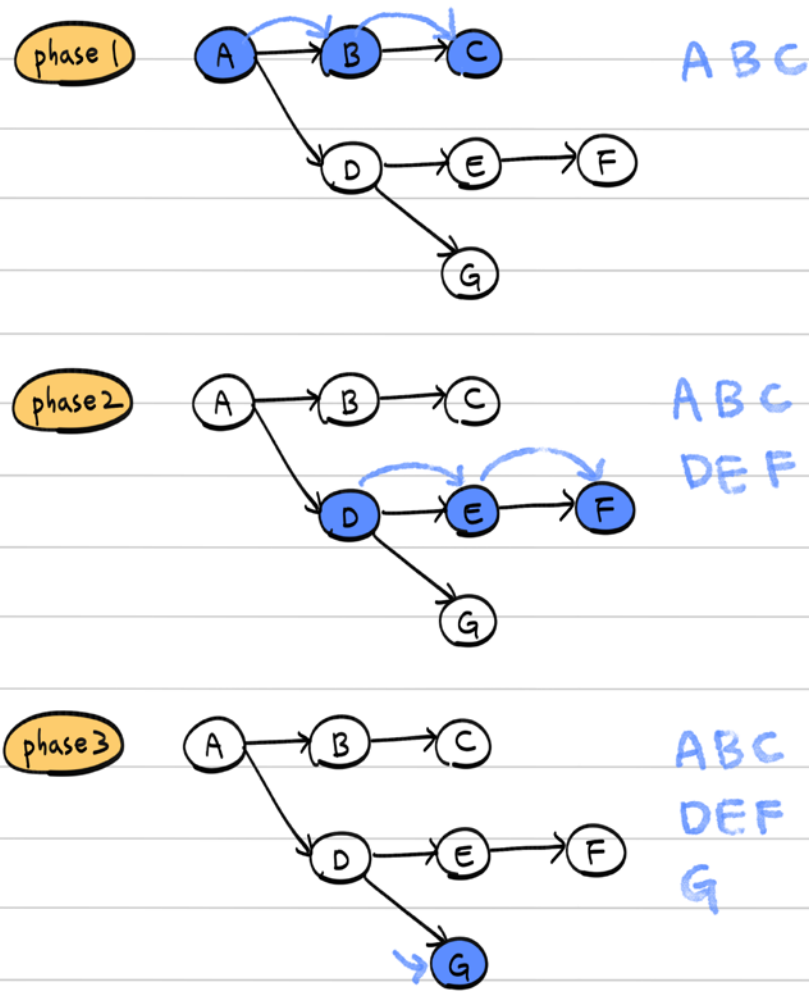
인접리스트 그래프



- 장점 1. 공간 활용 Good!
- 장점 2. 정점과 연결한 간선 정보 빠르게 확인

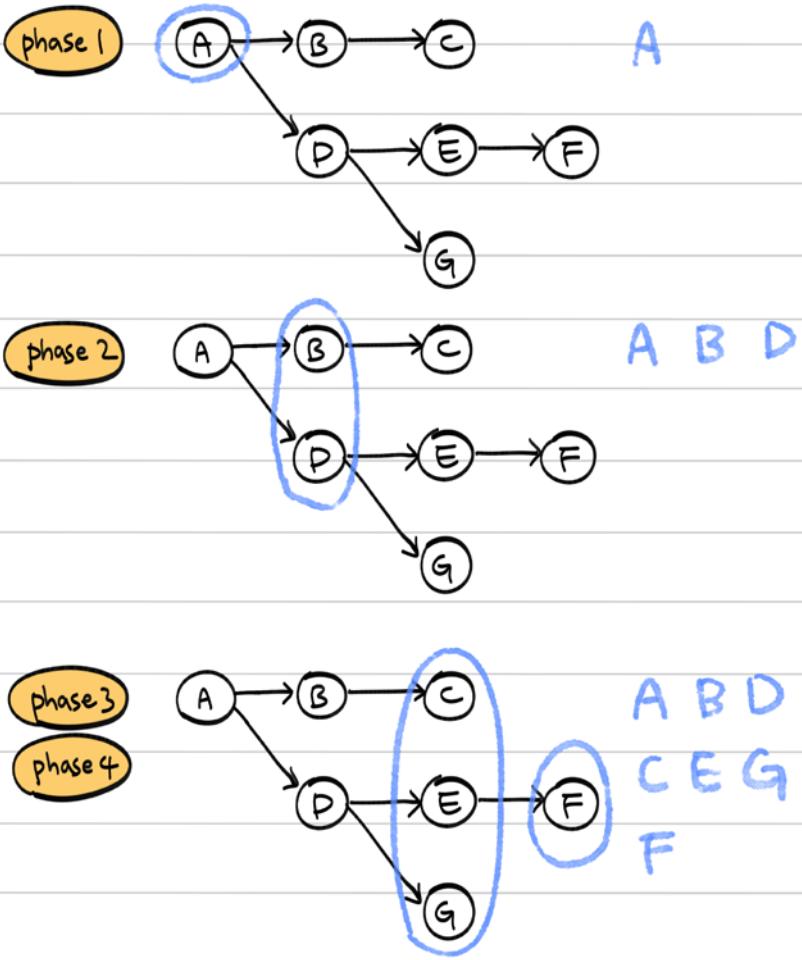
✳ 11장

그래프 탐색 : 깊이 우선 탐색 depth first search



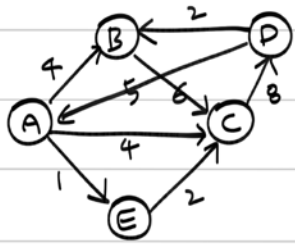
✳ 11장

그래프 탐색 : 너비 우선 탐색 breadth first search



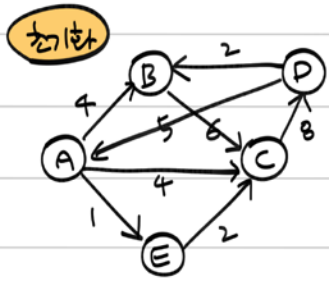
✳ 11장

다익스트라 알고리즘



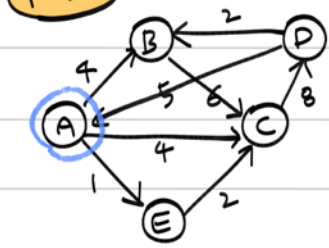
A	B	C	D	E
∞	∞	∞	∞	∞
-	-	-	-	-

초기화



A	B	C	D	E
0	∞	∞	∞	∞
A	-	-	-	-

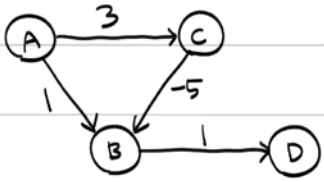
phase 1



A	B	C	D	E
0	4	4	∞	1
A	A	A	-	A

✳ 11장 cont.

다익스트라 알고리즘의 한계? (;)



- 음의 가중치가 있는 그래프
- 다익스트라 알고리즘이 찾는 최단경로

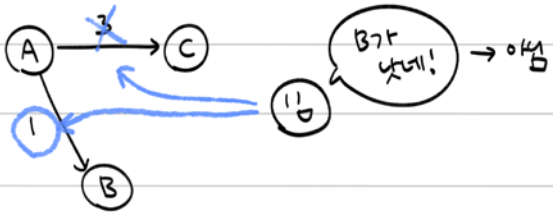
  A → B → D (X)

- 진짜 최단 경로는

  A → C → B → D (O)

why?

- 그리디한 특성 때문!

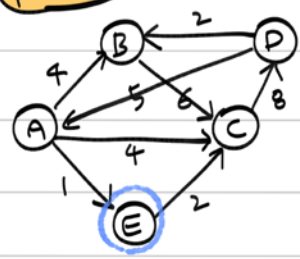


- 그래서 다익스트라 알고리즘은 '음의 가중치'가 있는 그래프에서 최단 경로를 보장할 수 없음!

✳ 11장

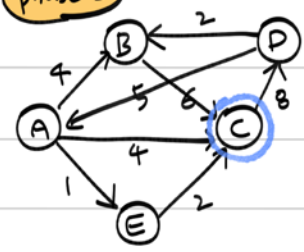
다익스트라 알고리즘

phase 2



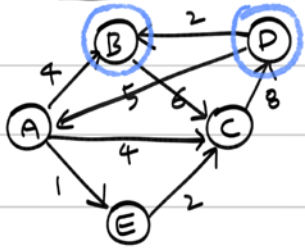
A	B	C	D	E
0	4	3	∞	1
A	A	E	-	A

phase 3



A	B	C	D	E
0	4	3	11	1
A	A	E	C	A

phase 4 → 5



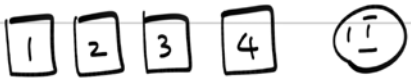
A	B	C	D	E
0	4	3	11	1
A	A	E	C	A

✳ 12장

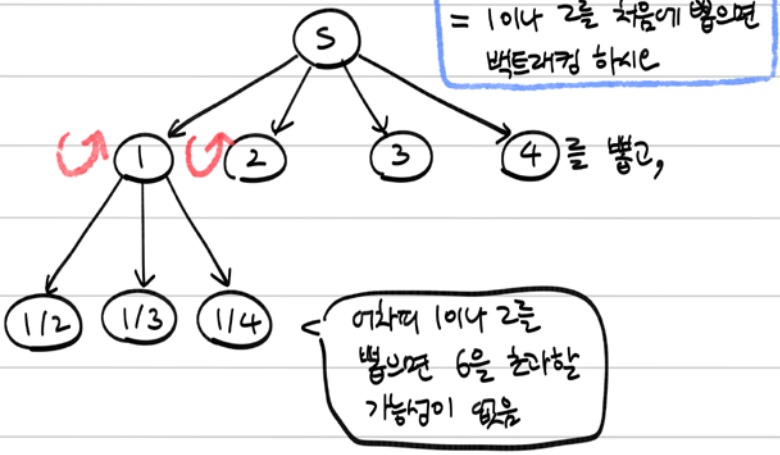
백트래킹

- 유망 함수로 답의 가능성을 체크,
  답이 아니면 되돌아가며 탐색하는 방식

ex 1, 2, 3, 4에서 두 수를 뽑아 합이 6을 초과하는 경우의 수?



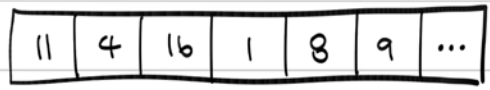
✳유망 함수
= 1이나 2를 처음에 뽑으면 백트래킹 하시오



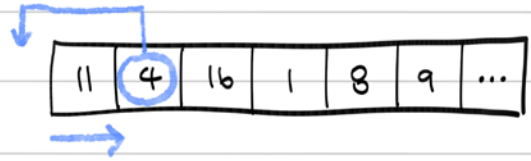
* 13장

삽입 정렬 : 삽입할 위치에 요소 집어넣고 정리하기

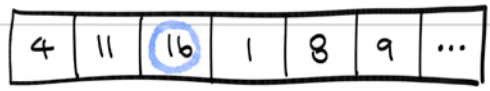
Start 배열 초기 상태



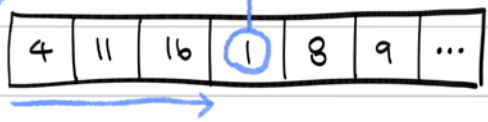
phase 1 4를 11 앞으로 옮기고, 11을 밀어 정리



phase 2 16은 현재 위치 OK



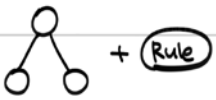
phase 3 1을 4 앞으로 옮기고, 4~16을 밀어 정리



⋮ 반복 진행 (n)

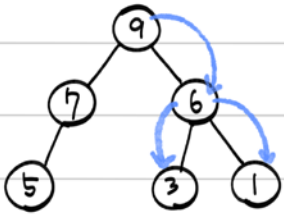
✳ 13장

힙과 힙 정렬



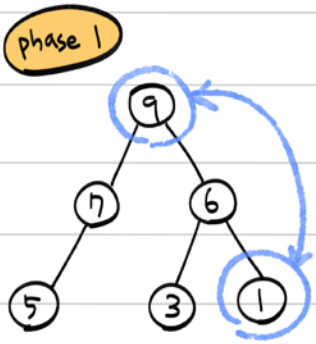
- 힙: 규칙이 있는 이진 트리

- 최대힙: 부모가 자식보다 큰 이진트리 (최소힙은 반대)



- 힙정렬: 힙을 이용한 정렬

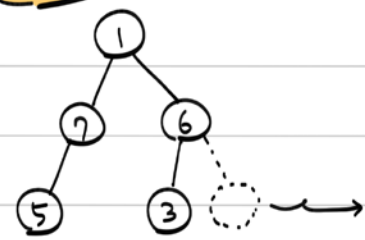
✳여기서는 최대힙의 루트가 가장 크다는 것을 이용.



• 루트와 말단을 swap

* 13장

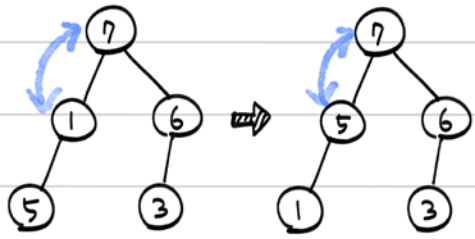
phase 2



- 말단 노드를 '정렬' 상태로 관리
- 이건 트리에서 제외

정렬 완료

phase 3



- max_heapify 진행
  = 힙 구조 유지

phase 4

이후 phase 1 ~ phase 3 반복 진행하면 정렬 완료.

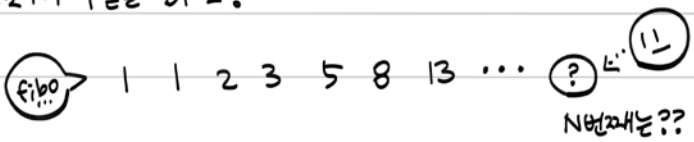
정렬 완료

* 15장

동적 계획법 dynamic programming = DP

- 동일한 작은 문제가 반복되어 나타날 때 고려.
- 작은 문제의 답이 큰 문제의 답을 구성해야 함.

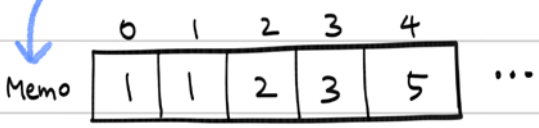
피보나치 수열을 DP로!



- 피보나치 수열 점화식 🔥

$$\begin{cases} fibo(1),\ fibo(2) = 1 \\ fibo(N) = fibo(N-1) + fibo(N-2) \end{cases}$$

* fibo(5) = fibo(4) + fibo(3)을 계산할 때 ···
fibo(4)는 이미 계산한 값이므로
다시 계산한다거나 하지 X
Memo한 값을 사용 ⟶ 연산을 줄임!
핵심!

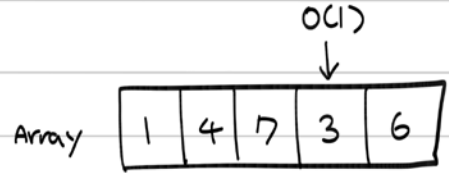


용어 정리 Top 15

1 배열 Array

동일한 자료형의 요소들이 연속으로 메모리에 저장된 구조.

인덱스로 O(1)만에 요소에 접근할 수 있음.

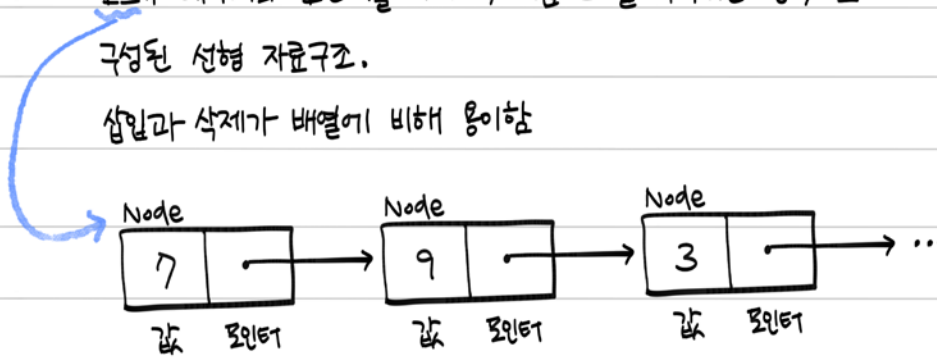


2 연결리스트 Linked List

노드가 데이터와 포인터를 가지며, 다음 노드를 가리키는 방식으로

구성된 선형 자료구조.

삽입과 삭제가 배열에 비해 용이함



3 스택 Stack

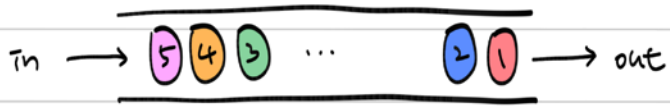
LIFO, 후입선출을 따르는 선형 자료구조.

데이터의 삽입과 삭제가 한쪽에서만 이루어짐.

4 큐 Queue

FIFO, 선입선출을 따르는 선형 자료구조.

데이터의 삽입과 삭제가 서로 다른 쪽에서 이루어짐.



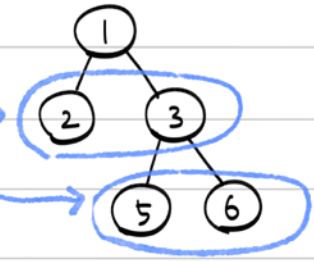
5 우선순위 큐 Priority Queue

요소의 우선순위가 높은 것부터 삭제하는 자료구조

* 자바의 PQ는 Min Heap,
작은 값부터 나옴

6 이진 트리 Binary Tree

각 노드가 최대 2개의 자식 노드를 가지는 트리.



7 이진 탐색 트리 Binary Search Tree

이진 트리의 한 종류.

탐색과 변경을 O(logN)으로 수행할 수 있음.

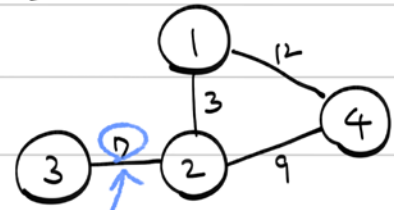
8 그래프 Graph

노드와 이를 연결하는 간선의 집합으로

이루어진 비선형 자료구조.

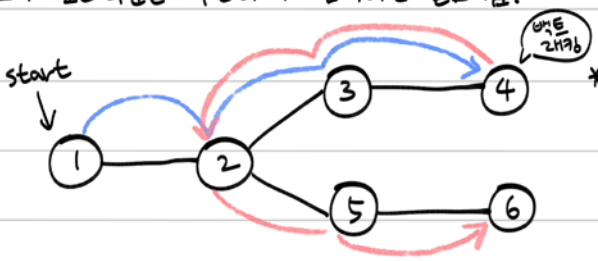
방향의 유무가,

가중치의 유무가 있을 수 있음.



9 깊이 우선 탐색 Depth First Search

그래프의 깊은 부분을 우선하여 탐색하는 알고리즘.

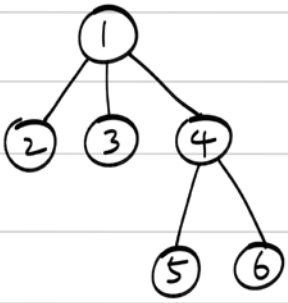


*숫자는 탐색 순서

10 너비 우선 탐색 Breadth First Search

루트 노드에서 가까운 노드를 우선하여 탐색하는 알고리즘

해를 찾았다면 그 해는 루트 노드에서 가장 가까운 해임!

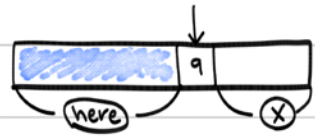


*숫자는 탐색 순서

11 이진 탐색 Binary Search

정렬된 배열에서 특정 값을 찾을 때 …

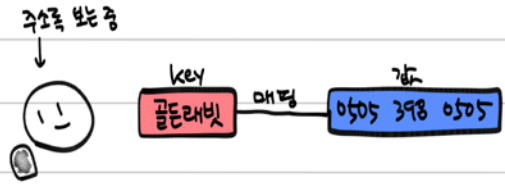
중간값을 기준으로 탐색 범위를 줄이는 알고리즘!



12 해시 테이블

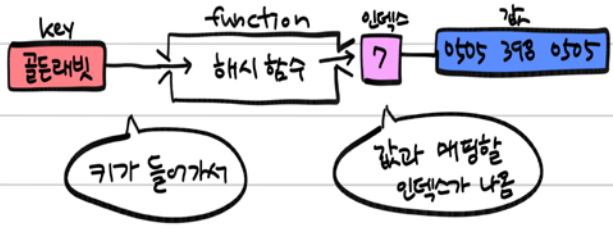
키에 데이터를 매핑하는 자료구조.

검색, 삽입, 삭제가 빠름!



13 해시 함수

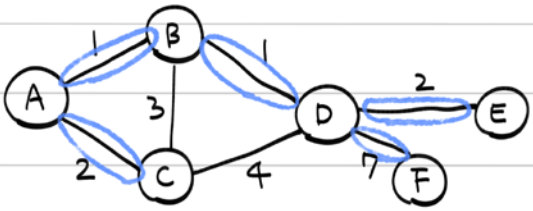
해시 자료구조의 값을 저장할 위치를 반환하는 함수



14 최소 신장 트리 Minimum Spanning Tree MST

가중치 그래프에서 모든 노드를 최소 비용으로 연결하는 트리 tree

크루스칼 알고리즘, 프림 알고리즘으로 만들 수 있음!

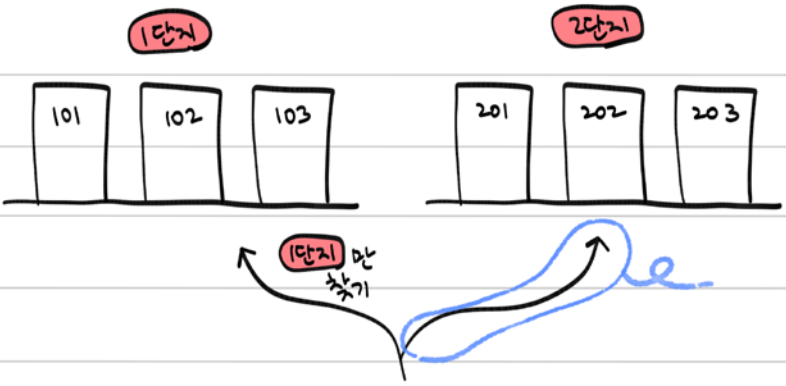


15 백트래킹 Back Tracking

모든 경로를 탐색할 때 …

가능성이 없는 곳은 가지치기하여 시간을 절약하는 방식!

ex 아파트 호수를 보고 친구 집을 찾기



Coding Test Study Note

Coding Test Study Note

Coding Test Study Note