합격자가 되는 정리 NOTE

시리즈 파이썬 편

이름

무단전재 및 재배포 금지

© 2024 골든래빗 Corp. All rights reserved.

01장 요약.

- 문제는 쪼개서 읽기
- 제약사항 파악하기
- 테스트 케이스 생각하기
- 입력값 분석하기
- 핵심 키워드 분석하기
- 데이터 흐름 + 구성 파악하기

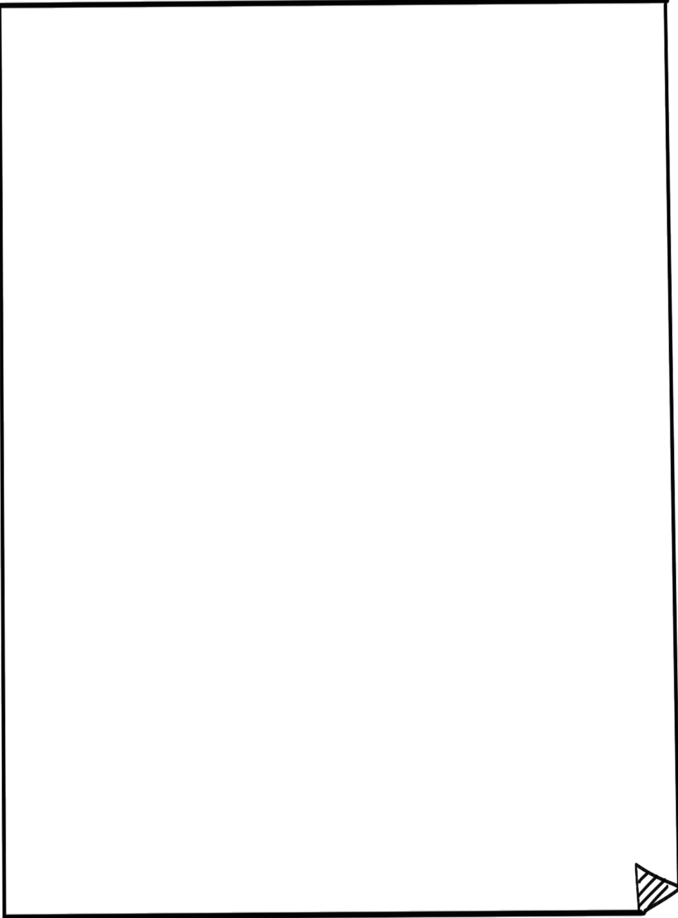
용어 의사 코드란? Pseudo-code 슈도-코드

- 프로그래밍 하듯이 쓰는 것이 아님.
- 자연어로 써야 함.
  + 동작 중심으로
  + 문제 해결 순서로

good) 국어, 영어, 수학 점수를 입력받는다.
bad) 크기가 256 바이트인 문자열을 ...

결론 반드시 분석부터 할 것!

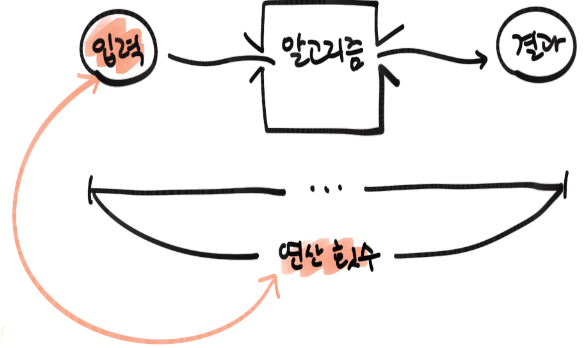
Note



03장 요약.

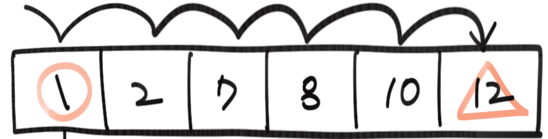
용어 시간 복잡도란? Time-complexity

- 알고리즘이 문제를 푸는데 연산횟수와 입력의 관계를 나타냄



예 1차원 배열에서 수 찾기

- 입력 = 6개
- 알고리즘 = 순서대로 찾기



1 찾기 → 한 번에 찾음 = 연산 1
12 찾기 → 마지막에 찾음 = 연산 6

순서대로 찾기 알고리즘의 시간 복잡도를 연산 횟수로만 말하면 1~6? → 애매 + 상황에 따라 달라짐

그래서 필요한 빅오 표기법 → 상황이 달라져도 표기법은 하나!

to be continue...

03장 요약. cont

용어 빅오 표기법 Big-O notation

- 최악의 경우를 고려한 시간 복잡도 표기법
- x에 대하여 연산 횟수를 $f(x)$라 할 때, 최고차항에서 수를 뗄 때 $O(x)$와 같이 표기

예 $f(x) = 2x^2+3x+5$ 면, $O(x^2)$

* 최고차항 참고

로그함수 < 다항함수 < 지수함수

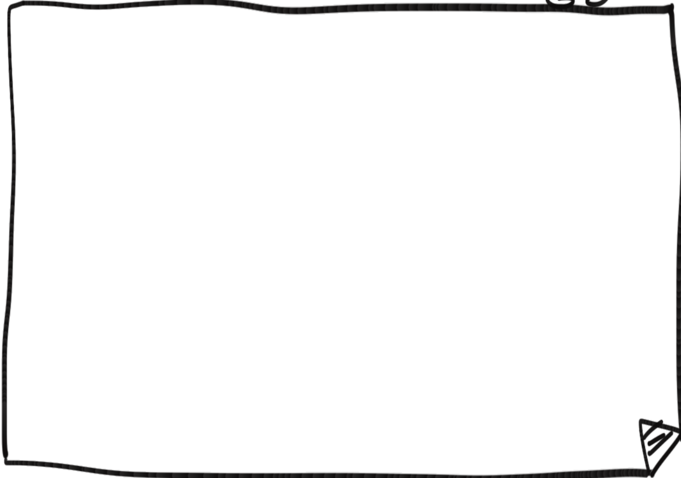
- 시간 복잡도는 알고리즘의 선택 기준이 된다!

why? 파이썬은 1초에 1000만번 연산
↳ 시간 복잡도에 따라 연산 횟수가 정해짐
↳ 문제 제한 시간이 1초면?
↳ 1000만번 연산 이상인 알고리즘!

시간 복잡도	최대 연산 횟수
$n!$	10
2^n	20
n^3	200
n^2	3000
$n\log n$	100만
n	1000만
$\log n$	10억

↓ 효율이 점점 좋아짐

Note



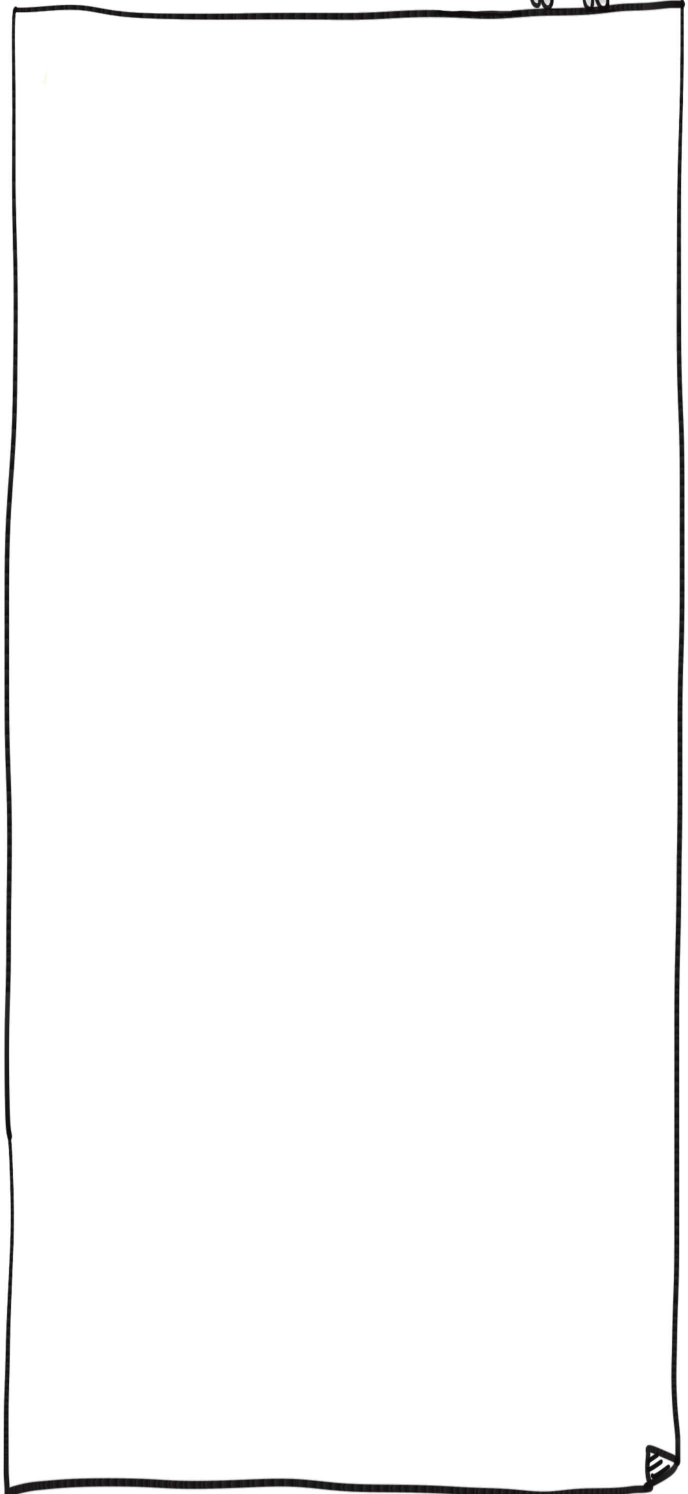
예 별찍기 문제의 시간 복잡도?

```
*
**
***
⋮
```

⇒ 출력 횟수 $f(x) = \frac{x(x+1)}{2}$ (등차수열 합)

$\therefore O(x^2)$

Note



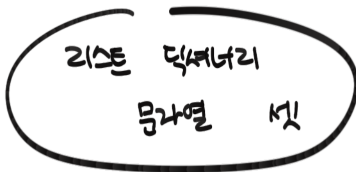
04장 요약.

용어 빌트인 데이터 타입 built-in data type
- 언어 자체에서 제공하는 데이터 타입

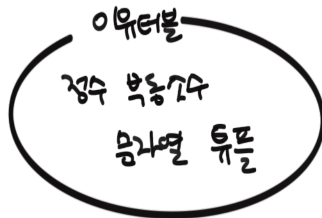
- 부동소수는 오차가 항상 있으므로 주의 (epsilon)

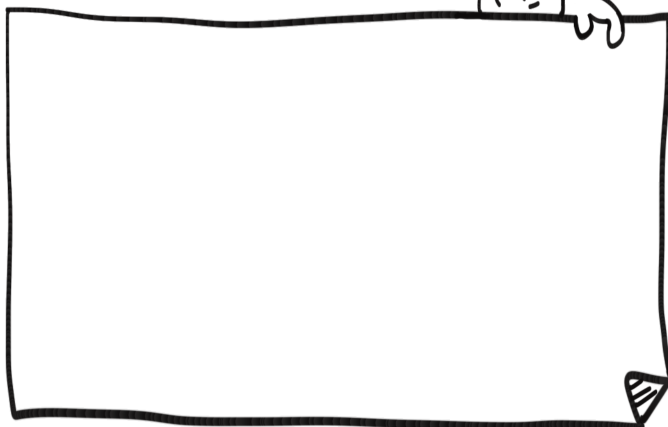
예
a = 0.1 + 0.1 + 0.1
b = 0.3
a - b = 5.5511151 ... e-17 → 0이 아님!

용어 컬렉션 데이터 타입 collection data type
- 여러 값을 담는 데이터 타입



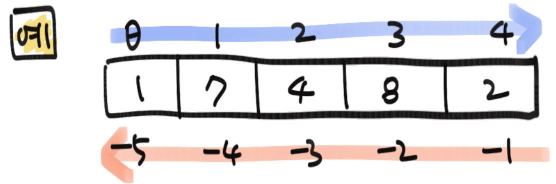
- 데이터 타입은 뮤터블(→ 수정 가능), 이뮤터블(→ 수정 불가능)로 나뉨



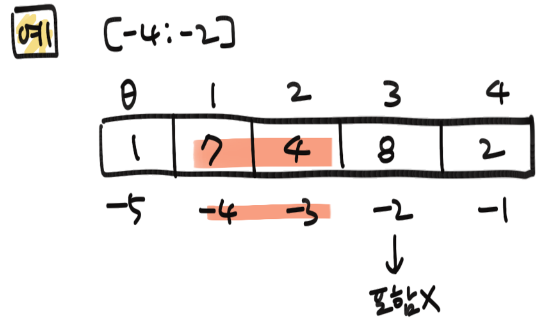


- 리스트 슬라이싱
  - ⊕ 방향, ⊖ 방향의 인덱스가 다름.

예



예 [-4:-2]



- 문자열은 이뮤터블하므로 수정하면 새 객체를 만들어 참조함.

예 a = "He"

04장 요약. cont.

- 람다식 lambda expression
  - 함수를 간단히
  - 이름 없는 함수를 만드는데 사용

예 lambda x, y : x + y
(x, y → 매개변수, x + y → 반환값)

- 람다식 유스케이스
  - 변수가 람다식 참조

예 add = lambda x, y : x + y

  - 인수로 람다식 넘기기

예 list (map(lambda x : x**2, num))
→ num에 있는 값 순회하며 x**2 연산 후
→ 리스트로 반환

- 조기 반환 early return
  - 코드 실행 과정이 함수 끝까지 도달하기 전에 반환하는 기법
  - 코드 가독성 ↑, 예외처리 good

- 보호 구문 guard clauses
  - 본격적인 로직을 진행하기 전에 예외처리를 하는 기법
  - 구현부에서 예외처리 안해도 됨 = 구현 집중
  - 보기 좋음, 안전성 ↑

- 합성 함수 composite method
  - 2개 이상의 함수로 함수를 만드는 방법
  - 보통 람다식으로 많이 함

예 lambda x : square(add(x))
* add, square 합치기 → ②+①

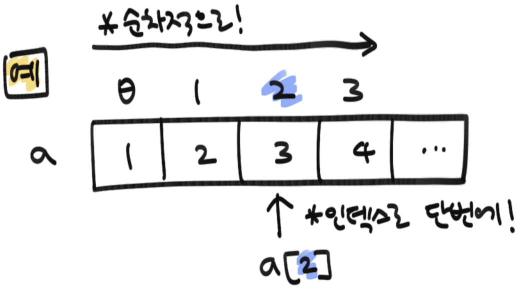
Note

05장 요약.

- 배열 array

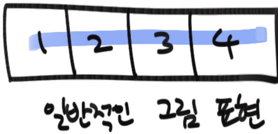
파이썬은 배열이라는 자료구조 자체를 제공하지는 않음.
이 책은 동등한 자료구조인 리스트를 사용.

- 인덱스에 값을 1:1 대응으로 순차적으로

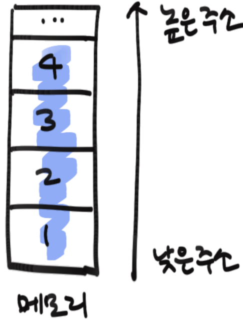


- 1차원 배열

[1, 2, 3, 4]



일반적인 그림 표현

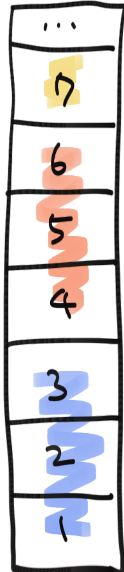


메모리

- 2차원 배열

예 [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

일반적인 그림 표현



메모리

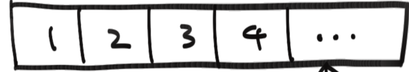
실제 메모리에는 일렬로 저장되는 구조임.

- 배열의 시간 복잡도

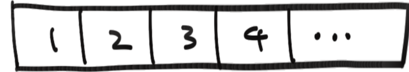
  - 배열 접근 = O(1)

단번에 인덱스로 접근하니까!

  - 배열 삽입
    - 맨 뒤 = O(1)
    - 맨 앞 = O(N)



맨뒤는 한번에!

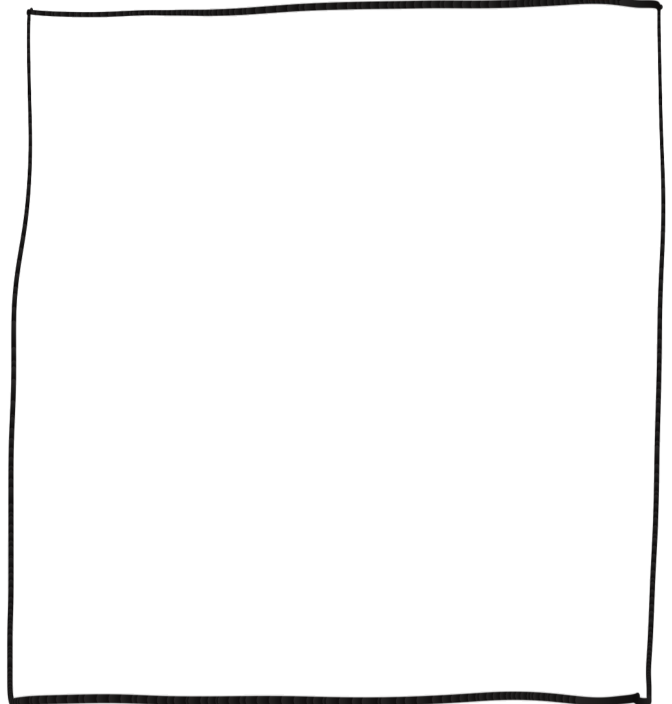


맨앞은 하나씩 밀어야 함! 총N번!

- How to use 배열

  - 데이터를 자주 임의 접근하는 경우 good
  - but 배열 선언 공간을 고려해야
    - 1차원 배열 = 1,000만개
      2차원 배열 = 3,000 x 3000 크기

Note



05장 요약. cont.

- 유용한 리스트 메서드
  - len() = 리스트 길이 반환
  - index() = 특정 값이 처음으로 등장한 위치 반환
  - sort() = 기준에 따라 정렬
  - count() = 특정 값의 개수를 반환

예 sort() 활용, reverse

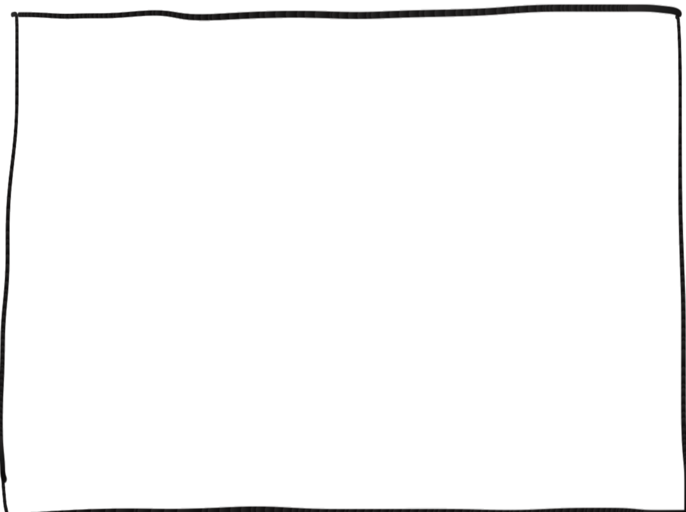
```
fruits = ["banana", "apple", "kiwi"]

fruits.sort()  #기준이 없다면 = 오름차순
# ["apple", "banana", "kiwi"]
fruits.sort(reverse = True)  # 내림차순
# ["kiwi", "banana", "apple"]
```

- reverse() = 배열 뒤집기
  - 일일이 뒤집는 것보다 이 함수가 더 빠름!

- for _ in range 구문
  - 반복문 내에서 변수를 사용할 필요가 없으면 보통 _로 표시하여 사용

Note



- 리스트 컴프리헨션 list comprehension

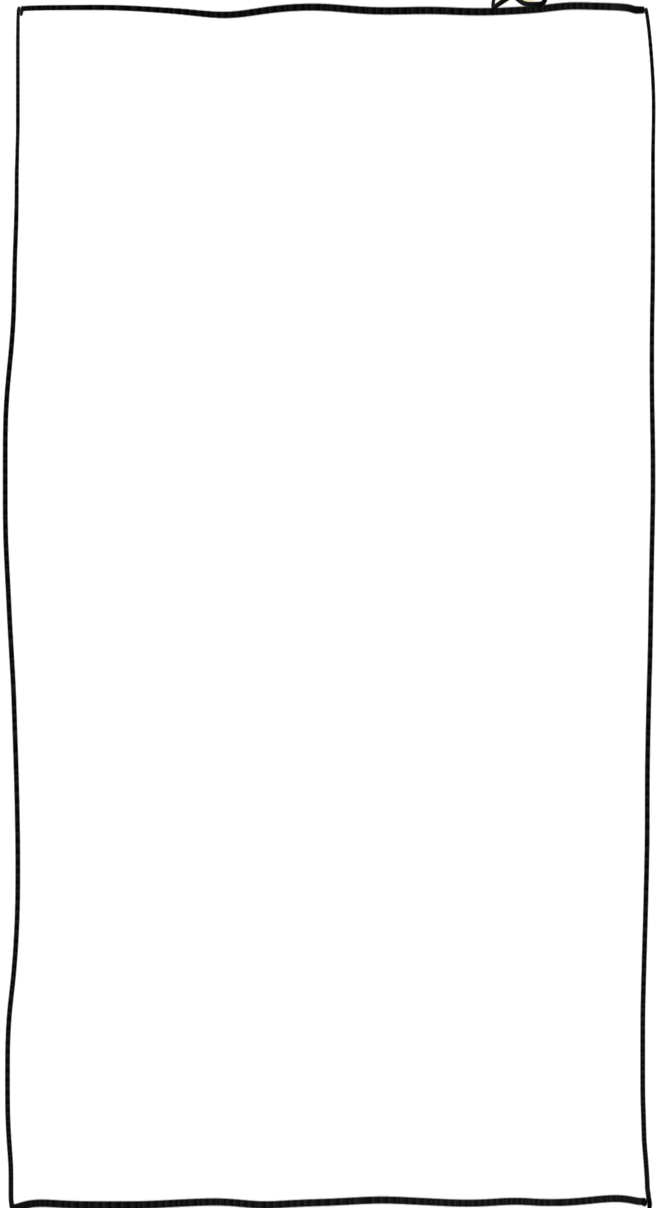
기존 리스트를 기반하여 새 리스트를 만들거나 반복문, 조건문과 조합하여 복잡한 리스트를 만들 수 있음

예 리스트에 제곱 연산하기

```
nums = [1, 2, 3, 4, 5]   ← 기반
sqrs = [num**2 for num in nums]
```

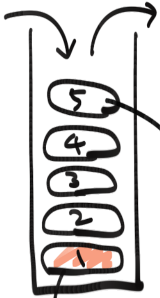
① ② ③

Note



06장 요약.

용어 스택 stack



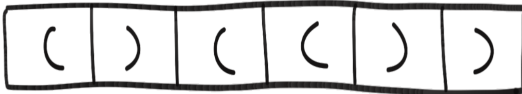
- 데이터가 쌓이는 구조
- First in Last out

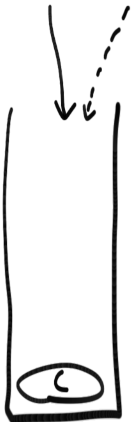
가장 나중에 들어간 것이
가장 먼저 나온다!

제일 먼저 들어갔지만
가장 나중에 나온다!

스택 대표 문제

1. 괄호 상쇄 문제

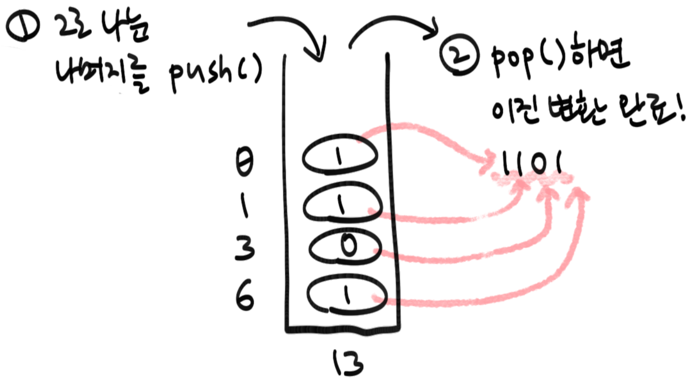




① 리스트에서 순서대로 push()

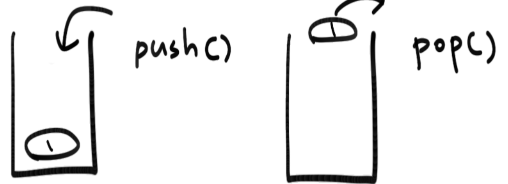
② 다음 push() 대상과 스택의 맨위가 짝이 맞으면 pop()

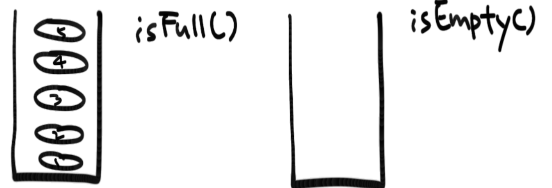
2. 이진수 변환 문제

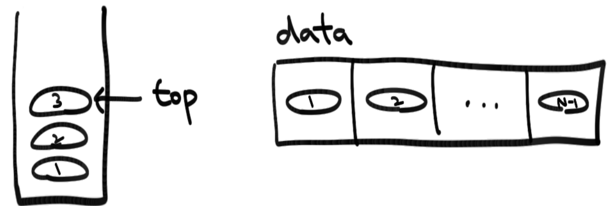


- 스택의 ADT

이러한 연산과 상태가 정의된 자·구를 스택이라 하겠다라는 것임

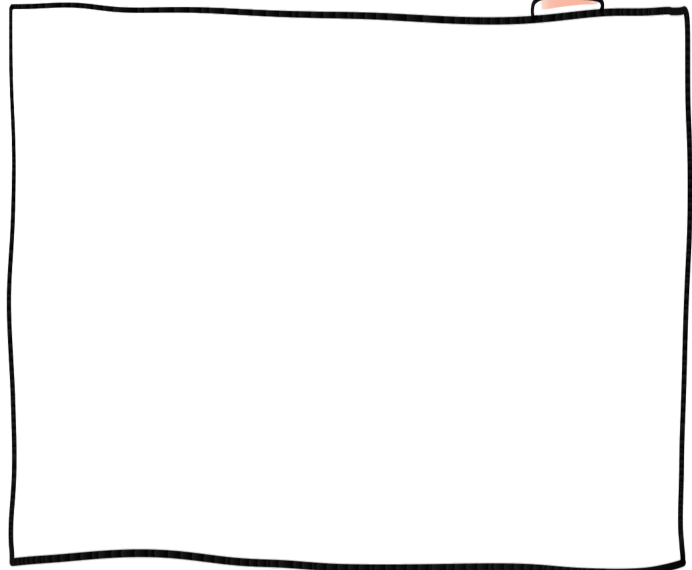






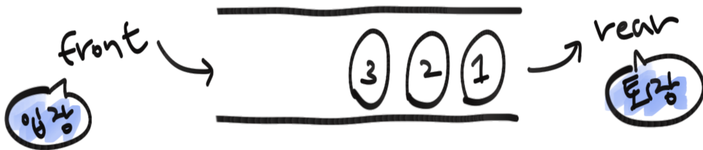
여기서 잠깐 추상 자료형 abstract data type ?

- ADT라고 줄여 부름!
- 인터페이스만 제공!



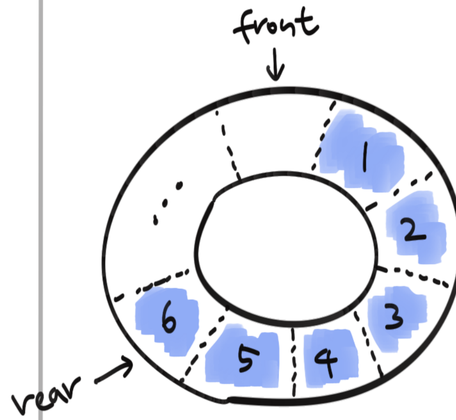
07강 요약.

용어 큐 queue



- 데이터가 줄을 서는 구조
- First In First Out
- 큐의 ADT
  - 연산은 스택과 비슷
  - front, rear를 관리한다는 점이 다름

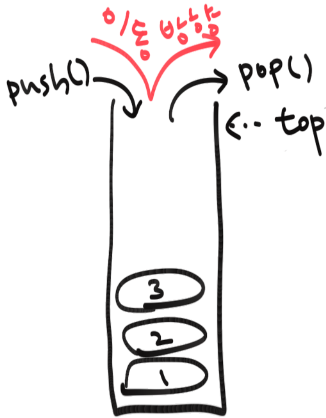
+ **용어** 원형 큐 circular queue

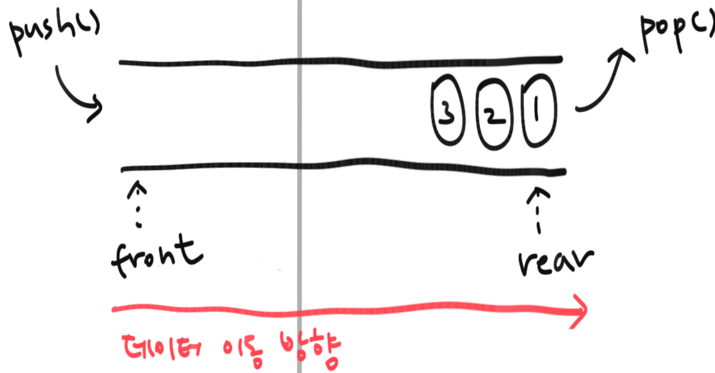


- front == rear
  ⇨ 비어있음
- (rear+1)% N == front
  ⇨ 가득 참

- 큐는 pop() 연산으로 비는 공간 활용이 생겨 비효율적인데 반해 원형큐는 front와 rear가 움직이면서 효율적으로 공간을 활용할 수 있음.

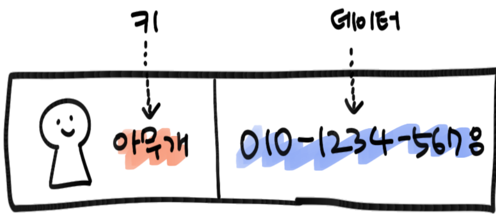
정리 스택과 큐

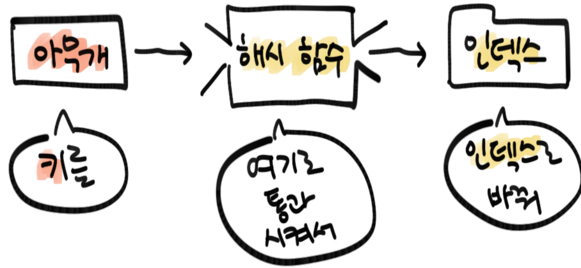


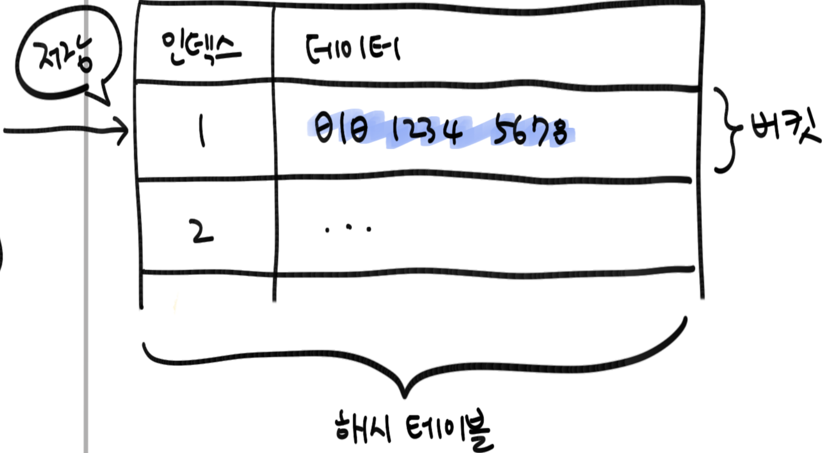


- 공통점
  - 모두 push(), pop() 연산으로 데이터를 넣고 뺀다.
  - 모두 isFull(), isEmpty() 연산으로 상태를 체크한다.
- 차이점
  - 스택은 top에서 데이터가 왔다갔다!
  - 큐는 front로 입장, rear로 퇴장!

8장 요약.







- 해시의 특징

  1 단방향으로 동작한다

  2 탐색 시간은 O(1)

- 해시가 쓰이는 분야

  1 비밀번호 관리 (단방향)

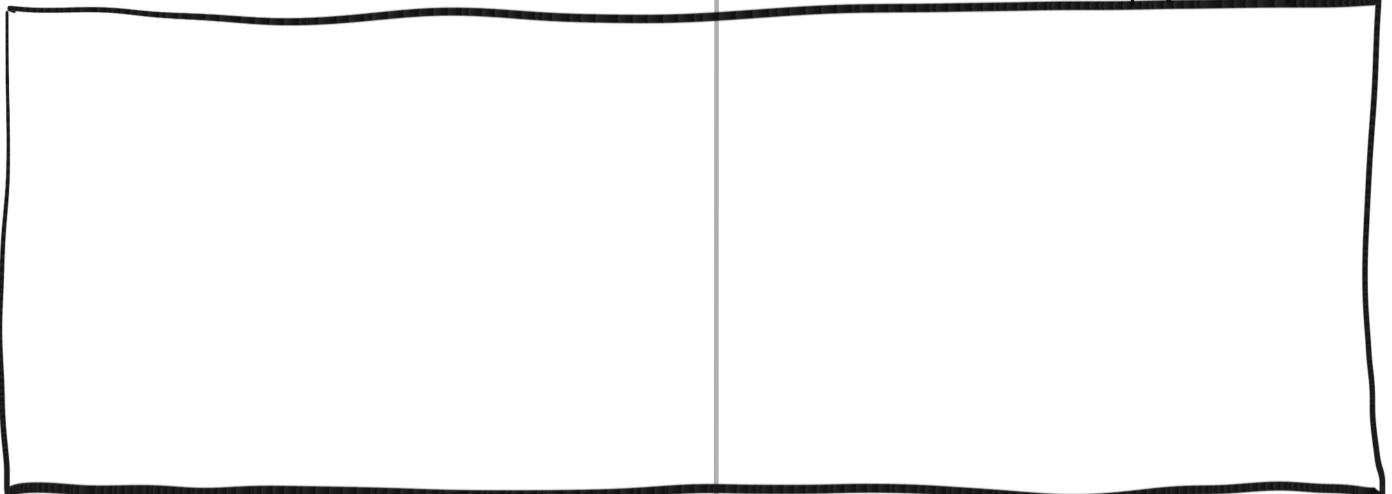
  2 데이터베이스 인덱싱

  3 블록체인

해시는 딕셔너리 활용법을 아는게 중요!

코드 보러가기

Note



08장 cont.

용어 해시 함수 hash function

해시 함수는 인덱스를 만드므로 인덱스의 적절성이 중요.

원칙 1 : 인덱스는 범위 내의 값을 만들 것

원칙 2 : 인덱스의 충돌을 최소화할 것

- 나눗셈법

key % prime

키를 소수로 나눈 나머지를 인덱스로 쓰는 방법

→ 소수가 해시 테이블의 크기를 정함
아주 큰 소수는 구하기 어려움
해시 테이블의 크기는 prime을 따라감

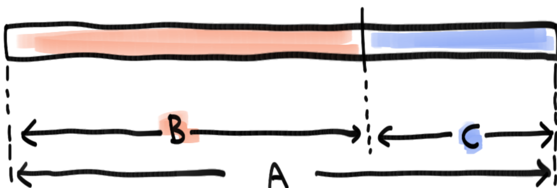
- 곱셈법

((key % gold) % 1) * m

키를 황금비로 나눈 나머지에서
소수부를 취한 다음
최대 버킷수를 곱한 값을 인덱스로 쓰는 방법

용어 황금비 gold ratio

임의의 길이를 두 부분으로 나누었을때
전체와 긴 부분의 비율이
긴 부분과 짧은 부분의 비율과 같은 값을 말함



A : B = B : C

- 문자열 해싱

문자열을 숫자로 변환하는 방법

과정 1. 1~26 숫자들에 문자열의 문자를 매칭

1	2	3	4	5	6	...	26
a	b	c	d	e	f	...	z

apple ⟶ 1, 16, 16, 12, 5

과정 2. 변환한 수와 31^{N-1}을 곱한 다음 더하기

$1 \times 31^0 + 16 \times 31^1 + 16 \times 31^2 + 12 \times 31^3 + 5 \times 31^4$

$= 4990970$

과정 3. 수를 m으로 모듈러 연산하기

m은 해시 테이블의 크기!

단점!

아주 간단한 문자열도 매우 큰 값이 나오므로
오버플로에 주의해야 함

why? 왜 31을 곱하는지?

홀수이면서 메르센 소수이기 때문

용어 메르센 소수

$2^N - 1$로 표현할 수 있는 수
해시 충돌을 줄이는 데 효과가 있다고 함

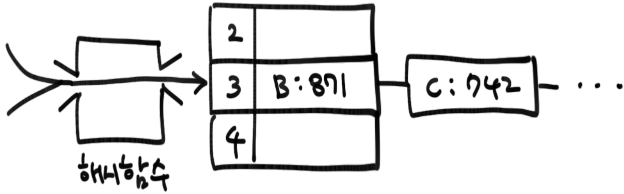
08장 cont.

- 충돌 처리 : 체이닝

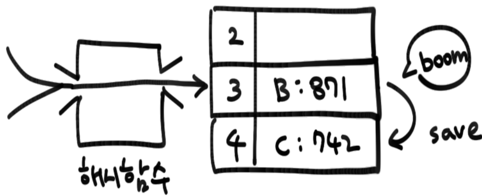
해싱한 값이 같으면 리스트로 연결하여 저장

ⓥ 단점 1 : 해시테이블 공간 효율이 떨어짐

ⓥ 단점 2 : 검색 성능이 떨어짐



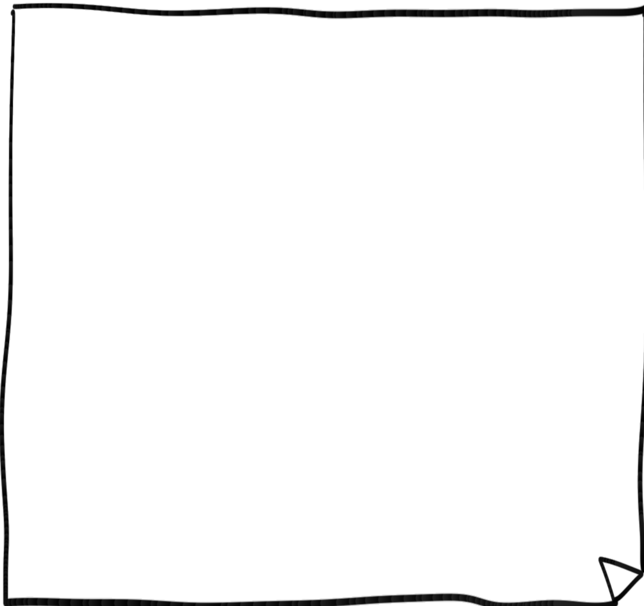
- 충돌 처리 : 개방 주소법
  - 선형 탐사 방식 : 충돌이 발생하면 빈 버킷 찾음
    보통 간격은 1



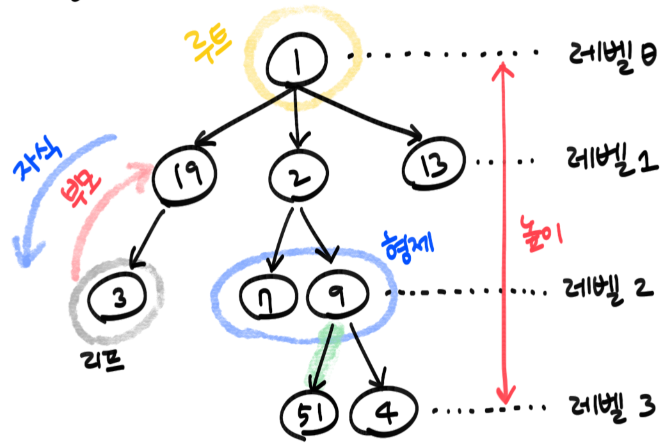
  - 이중 해시 방식 : 충돌이 발생하면
    두 번째 해시 함수 동작

"실제 코딩 테스트에서는 키-값을 매핑하는 것에 집중하는 것이 좋음"

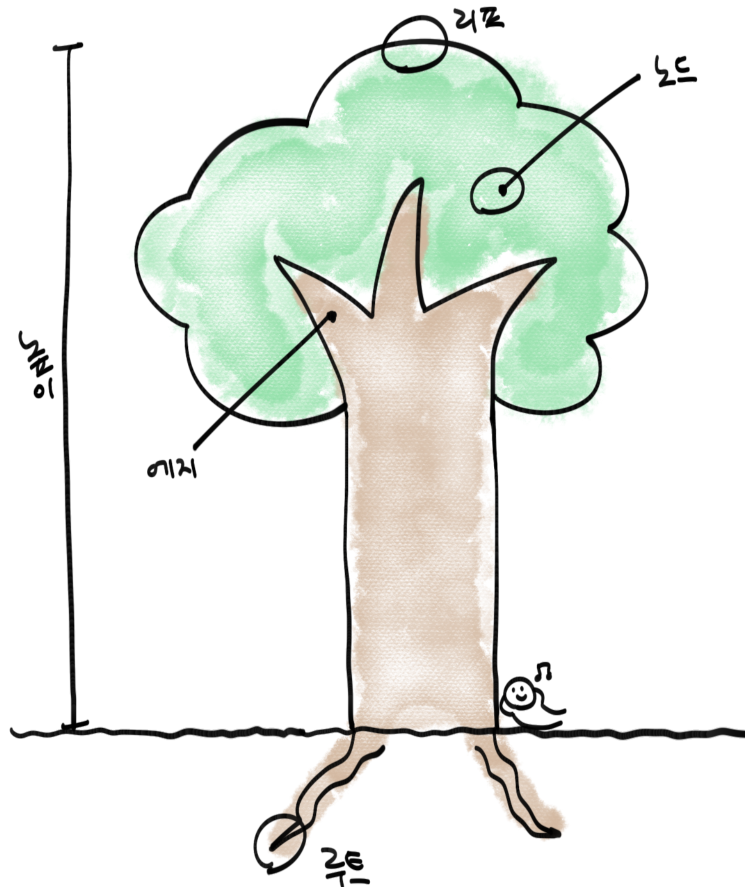
Note



09장 트리

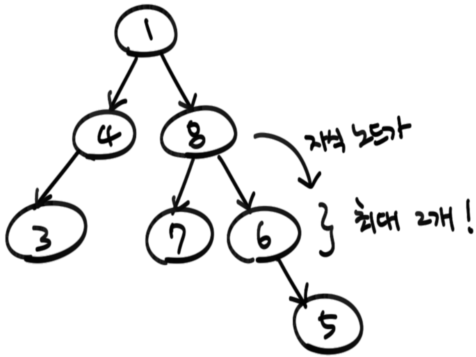


- : 노드 : 트리를 구성하는 기본 단위
- : 루트노드 : 트리 상위에 있는 노드
- : 부모관계 : 노드 간의 관계, 상위
- : 자식관계 : 노드 간의 관계, 하위
- : 형제관계 : 노드 간의 관계, 동위
- : 높이 : 트리의 높이 (지금은 3)
- 레벨 : 노드의 깊이 (0부터 시작)
- : 에지 : 노드를 잇는 선
- : 리프 : 맨 끝에 있는 노드



09강 트리 cont.

- 이진트리 : 자식 노드가 최대 2개인 트리

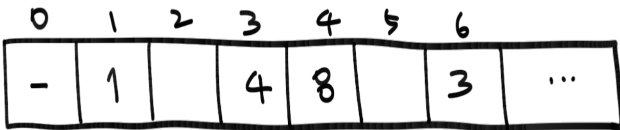


- 이진 트리 배열로 표현하기

① 루트 노드는 인덱스 1

② 왼쪽 자식 노드는 부모 노드 인덱스 X 2

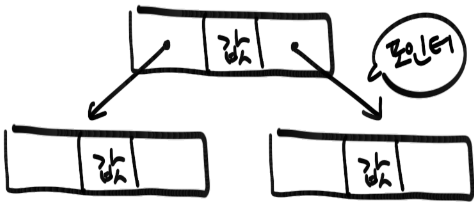
③ 오른쪽 자식 노드는 부모 노드 인덱스 X 2 + 1



0	1	2	3	4	5	6	
-	1		4	8		3	...

단점 : 빈 공간이 생긴다.
= 메모리 효율이 떨어짐.
but 코테 활용 가능

- 이진 트리 포인터로 표현하기



단점 : 구현 난이도가 있음

- 순회 with 이진 트리

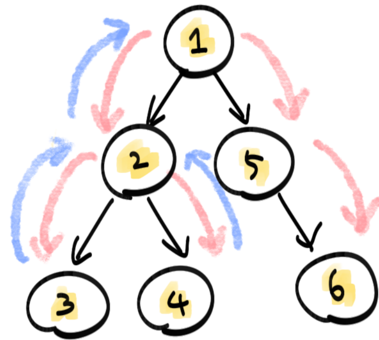
① 전위 순회 Preorder

② 중위 순회 Inorder

③ 후위 순회 postorder

매우 중요!

① 전위 순회

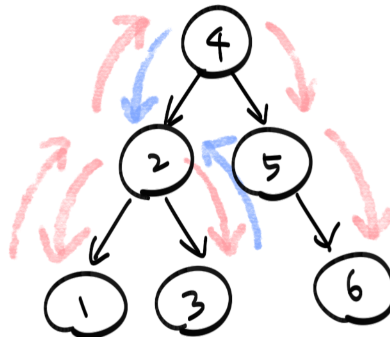


방문 순서

• 부모 → 왼쪽 → 오른쪽

• 자체가 방문 순서

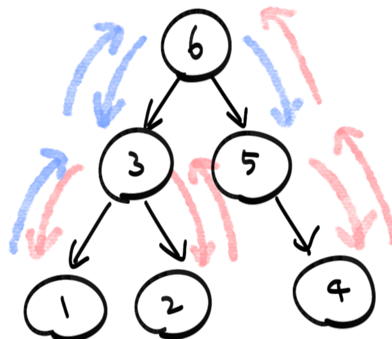
② 중위 순회



방문 순서

• 왼쪽 → 부모 → 오른쪽

③ 후위 순회



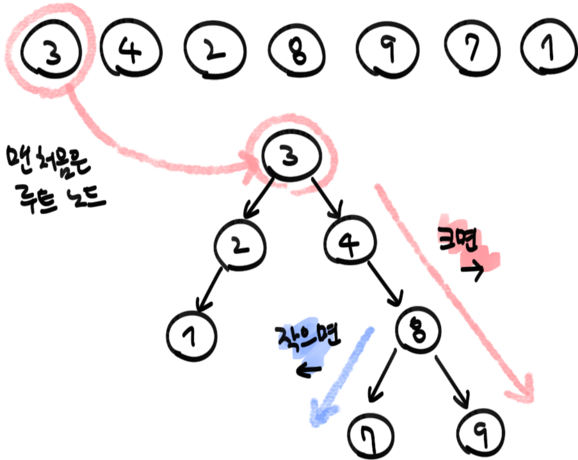
방문 순서

• 왼쪽 → 오른쪽 → 부모

순회의 포인트는 현재의 나를 부모로 생각할 때 어디를 방문할거이다!

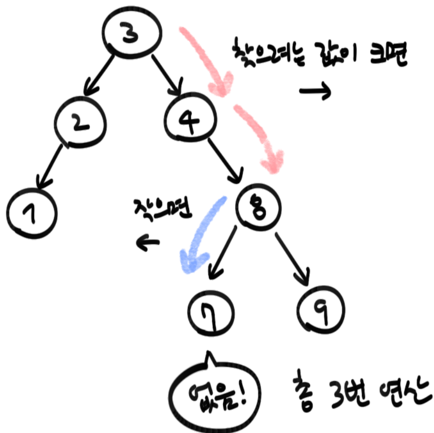
09장 트리 cont.

- 이진 트리 구축 : 크면 →, 작으면 ←

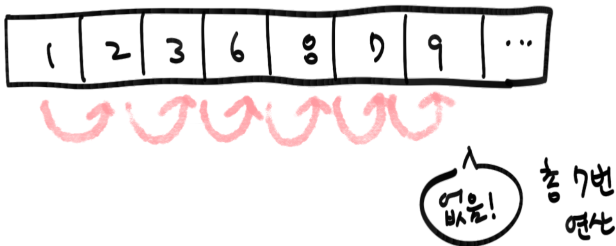


- 이진 트리 탐색

⑤를 찾는다! → 구축 방식으로 찾음



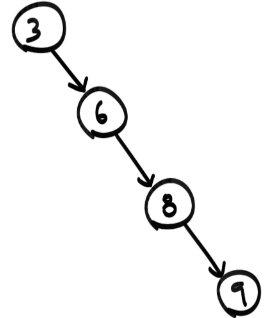
- 배열 탐색과 이진 탐색 비교하기



- 이진 트리의 탐색 성능 O(logN)
- 배열의 탐색 성능 O(N)

- 치우쳐진 이진 트리?

- 배열보다 이진 트리의 탐색 효율이 좋아보이나 이진 트리가 한 쪽으로 치우치면 탐색 효율은 같아진다!



- 결론 : 이진 트리는 균형을 맞추는 것이 중요!

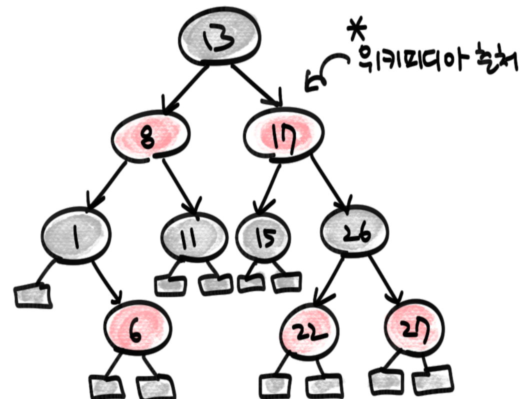
- 균형 이진 트리

- AVL 트리 : 노드를 트리에 추가할 때 균형 인수를 계산하여 왼쪽 서브 트리와 오른쪽 서브 트리의 균형을 맞춤

  용어 균형 인수 : 왼쪽 서브 트리 높이에서 오른쪽 서브 트리 높이를 뺀 값

- Red-Black 트리 : 다음 조건을 갖춘 트리

  ① 모든 노드는 레드나 블랙 중 하나
  ② 루트는 블랙
  ③ 모든 리프도 블랙
  ④ 레드의 자식 노드는 모두 블랙
  ⑤ 어떤 서브 트리를 보아도 리프를 제외하면 블랙의 수는 같음



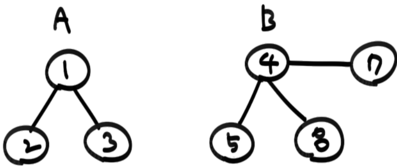
※ 모든 값이 있는 노드는 빈 값 노드(□)를 자식으로 가지고 있음을 가정함

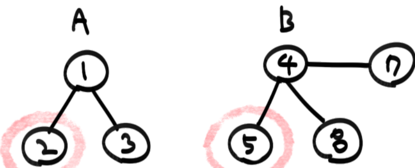
10장 집합

- 집합이란? : 순서와 중복이 없는 자료구조

- 상호배타적 집합이란? : 교집합이 없는 집합 관계

↓ 코테에서는 그래프로 집합 표현



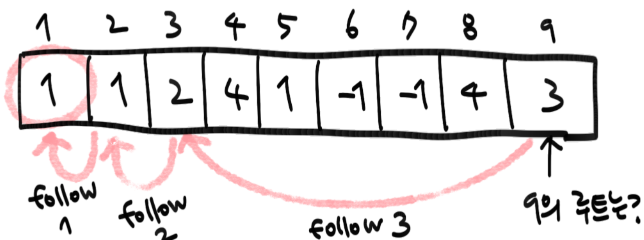


- 배열로 집합 표현하는 법

① 배열의 인덱스 = 자신

② 배열의 값 = 부모

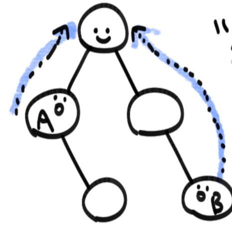
- 배열로 표현하는 집합에서 루트 찾기



- 유니온-파인드 알고리즘 : 집합 연산에 쓰이는 유니온(합치기), 파인드(찾기) 연산을 말함

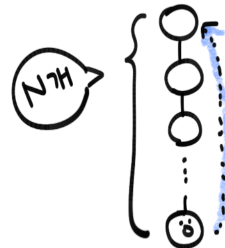
- 파인드 알고리즘

  - 특정 노드의 루트 노드가 무엇인지 찾음
  - 같은 집합에 속해 있는지 찾을 때 유용



"A, B의 파인드 연산 결괏값이 같으면 두 노드는 같은 집합!"

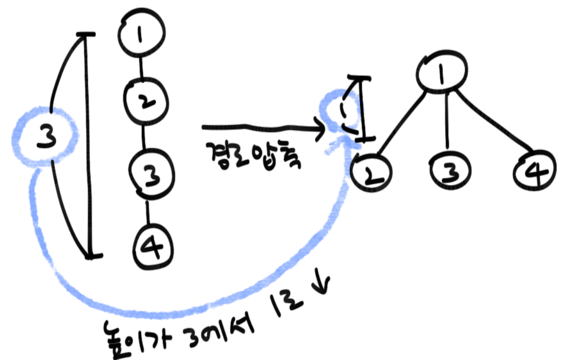
- 파인드 연산은 최악의 경우 O(N)이다.

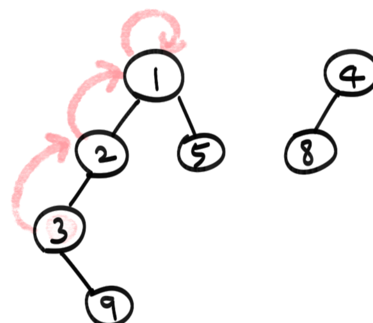


"일자로 구성된 그래프면 모든 노드를 거쳐야 루트 노드를 찾게 되니까!"

- 파인드 알고리즘 : 효율 올리기

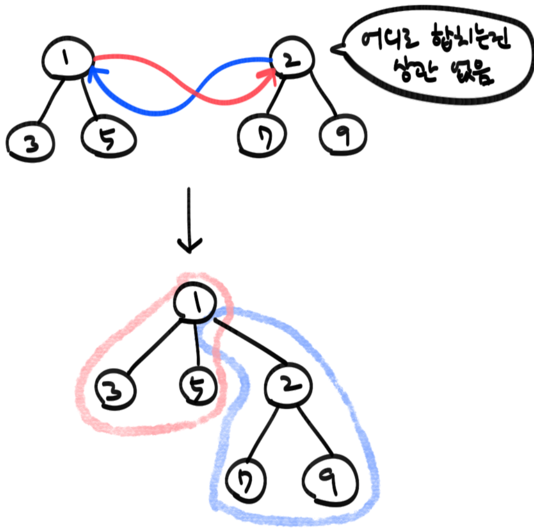
  - 연산 수행 도중에 트리 높이를 줄여 연산 효율을 올릴 수도 있음



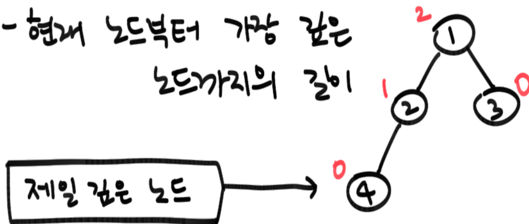


10장 집합 cont.

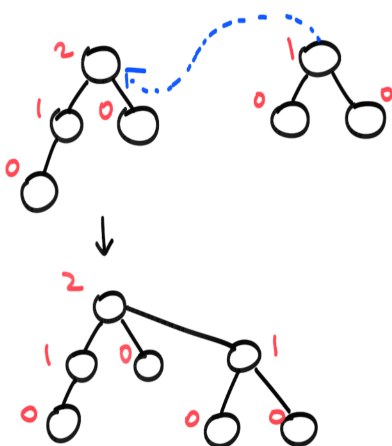
- 합치기 알고리즘
  - 루트 노드가 같도록 합치기



- 합치기 알고리즘 : Rank?
  - 현재 노드부터 가장 깊은 노드까지의 길이

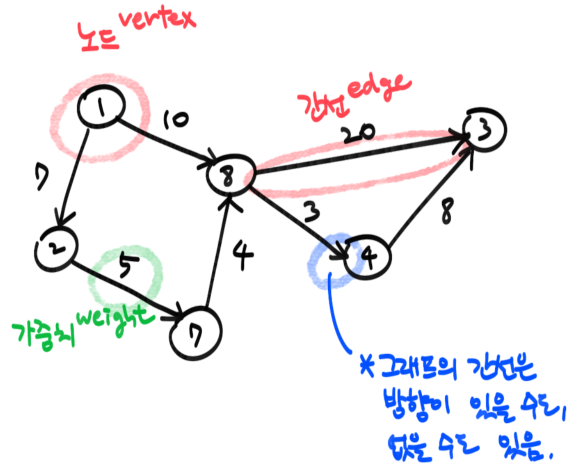


- 랭크 기반의 합치기 알고리즘
  - 루트 노드의 랭크를 비교하며 랭크가 큰 쪽으로 합치는 방법

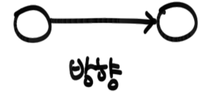


11장 그래프

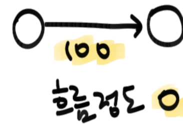
- 그래프 기초 구성

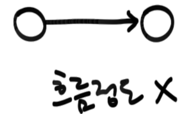


- 방향성의 유무



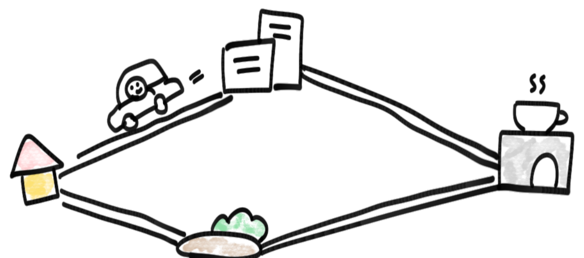
- 가중치로 흐름의 정도를 표현





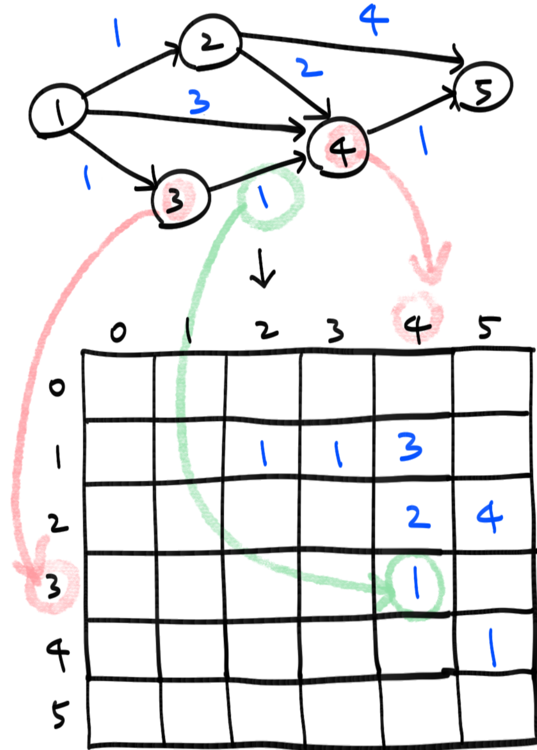
- 순환의 유무 : 다 연결되었다고 순환이 아님

- 순환이란? 시작점과 종료점이 같은 경로가 있으면 순환이라 함



11장 그래프 cont.

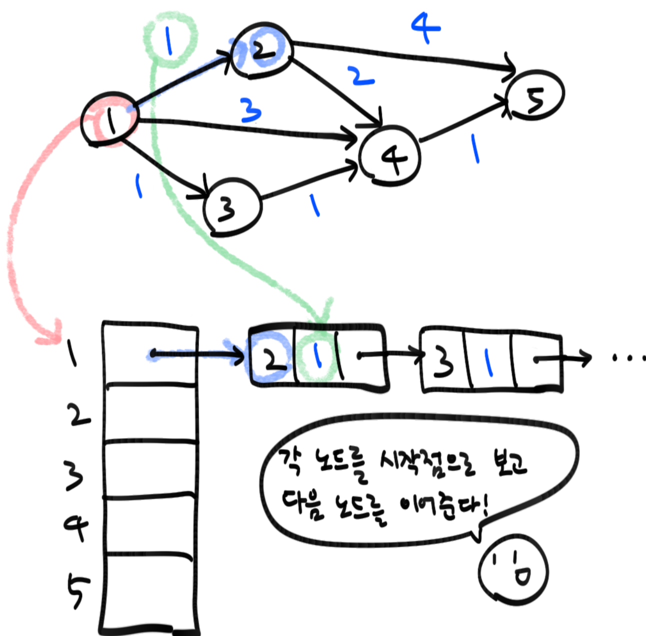
• 인접 행렬로 그래프 표현하기



	0	1	2	3	4	5
0						
1			1	1	3	
2					2	4
3					1	
4						1
5						

- 노드의 값을 인접 행렬의 인덱스로 하여 가중치를 채우는 식으로 구현

• 인접 리스트로 그래프 표현하기



• 인접 행렬 pros & cons

- 희소 그래프 표현에 약하다
  → 노드 수는 많은데 간선은 적음
  = 인접 행렬 공간만 차지하고 값은 채우지 않음
- 값 확인은 O(1)으로 빠르다
- 구현 난이도가 낮다

• 인접 리스트 pros & cons

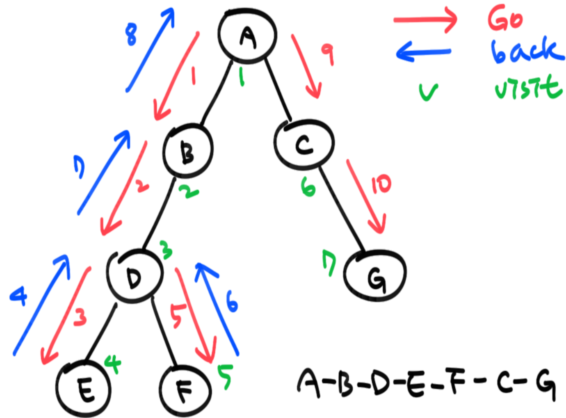
- 메모리 사용 효율이 좋다
- 탐색 시간이 O(N)이다

N개면 탐색에 O(N)이 듦

결론은 노드 개수가 1000개 미만이면 인접 행렬, 이상이면 인접 리스트로!

11장 그래프 cont.

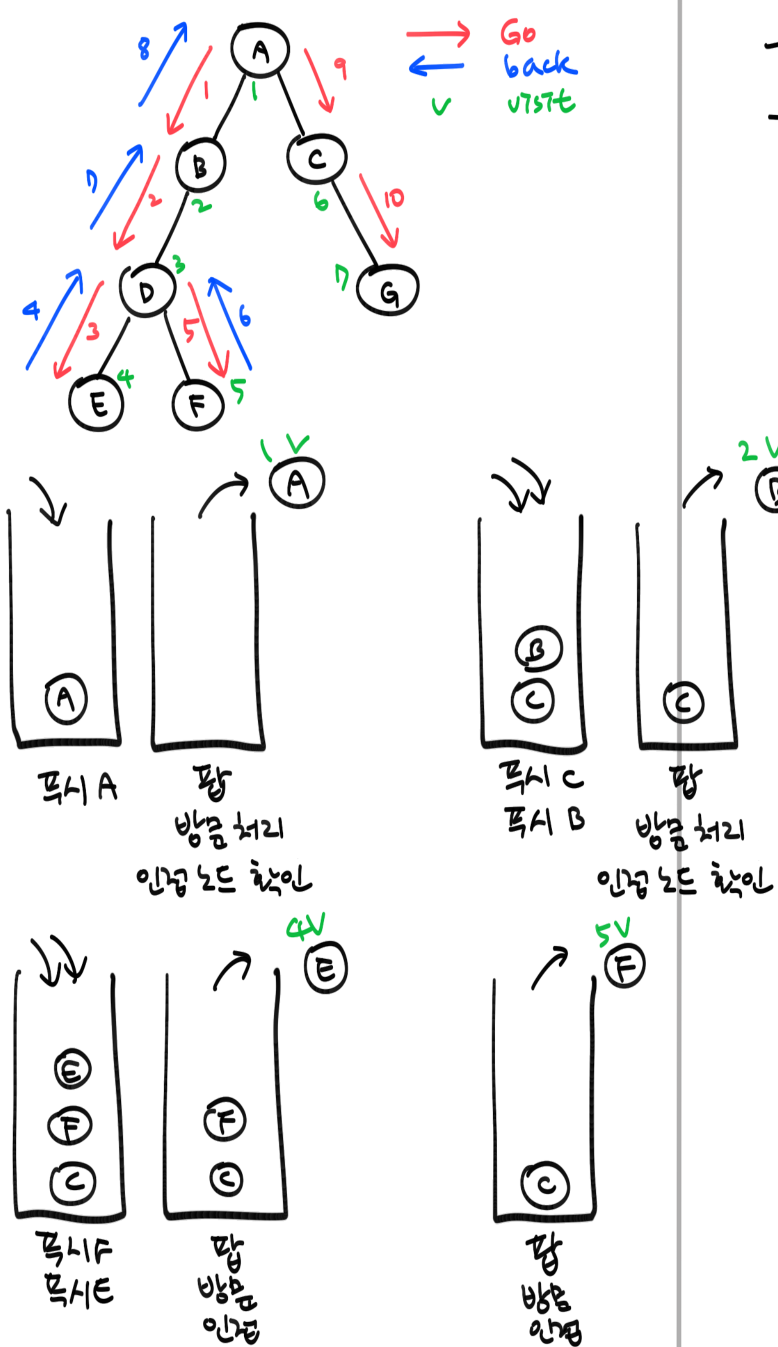
• 깊이 우선 탐색



용어 방문 : 노드 확인이 끝났음을 체크하는 것
visit이라고 이야기하는 경우도 많음

- 방문 순서 : A→B→D→E→F→C→G
- 방문 요약 : 최대 깊이 까지 가면서 방문하고 막히면 직전으로 거슬러간다.

• 깊이 우선 탐색 : 스택으로 구현하기



- 스택의 선입후출을 이용하여 구현.
- 팝하며 방문하고 인접 노드를 푸시.

즉, 푸시는 방문 예정인 노드라고 생각!

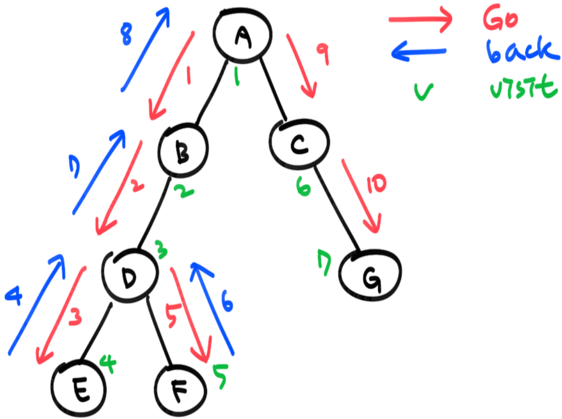
11강 그래프 cont.

- 깊이우선 탐색 : 재귀로 구현하기

  - dfs() 함수 정의

    ① 현재 노드 방문 처리

    ② 인접한 노드 중 아직 방문하지 않은 노드는 모두 dfs 호출



※이해가 어려우면 깃허브 코드 참고

함수 호출 시 남아 있는 다음 dfs 호출을 고려하여 그림을 따라 이해하기

① A 방문 처리

② dfs(B) 호출

②' dfs(C) 호출

① B 방문처리

② dfs(D) 호출

① D 방문처리

② dfs(E) 호출

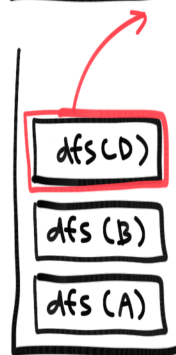
②' dfs(F) 호출

① E 방문처리

② dfs(E) 종료

① F 방문처리

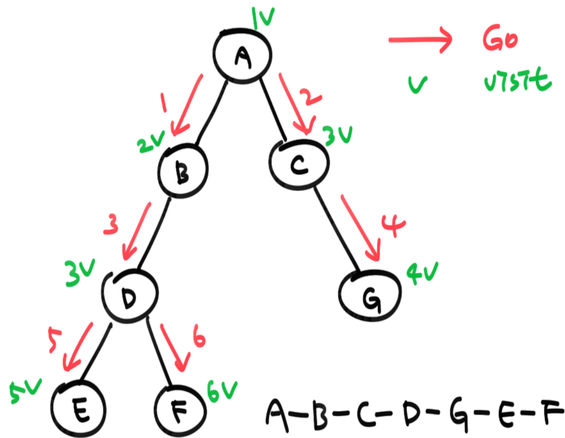
② dfs(F) 종료



① dfs(D) 종료

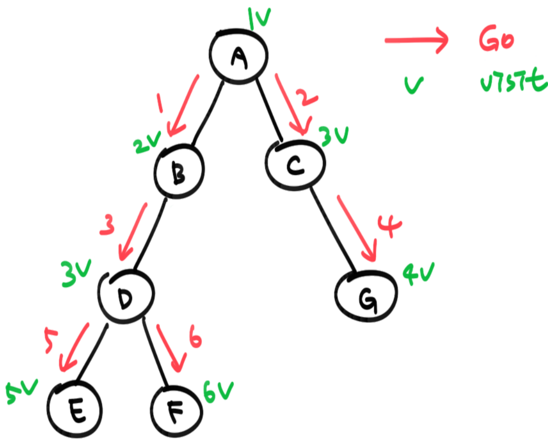
11장 그래프 cont.

- 너비 우선 탐색

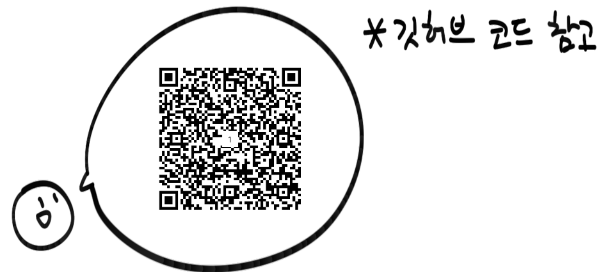


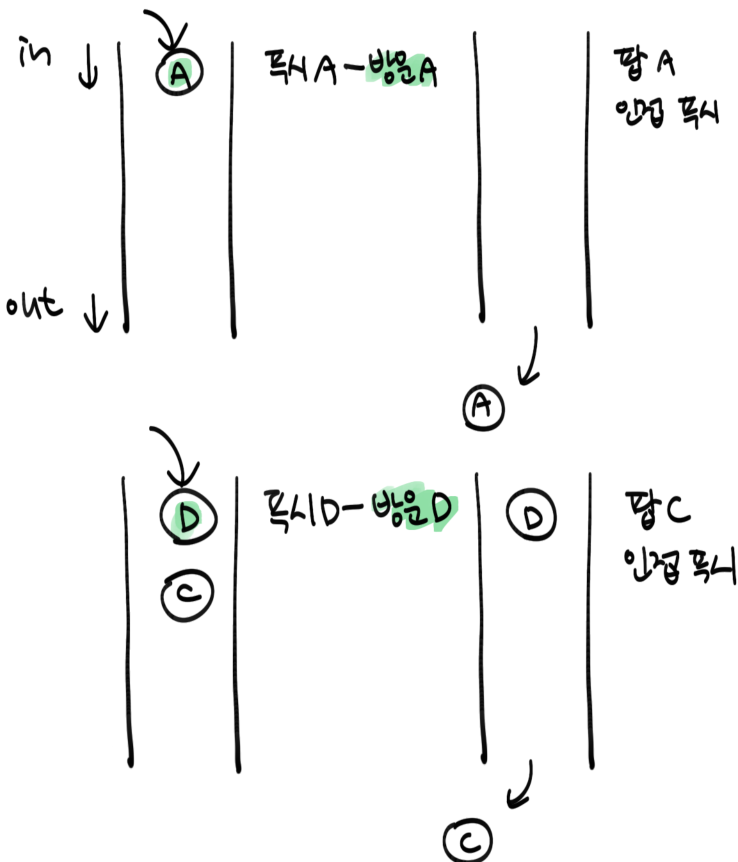
- 방문 순서: A→B→C→D→G→E→F
- 방문 요약: 가장 가까운 노드를 우선 방문

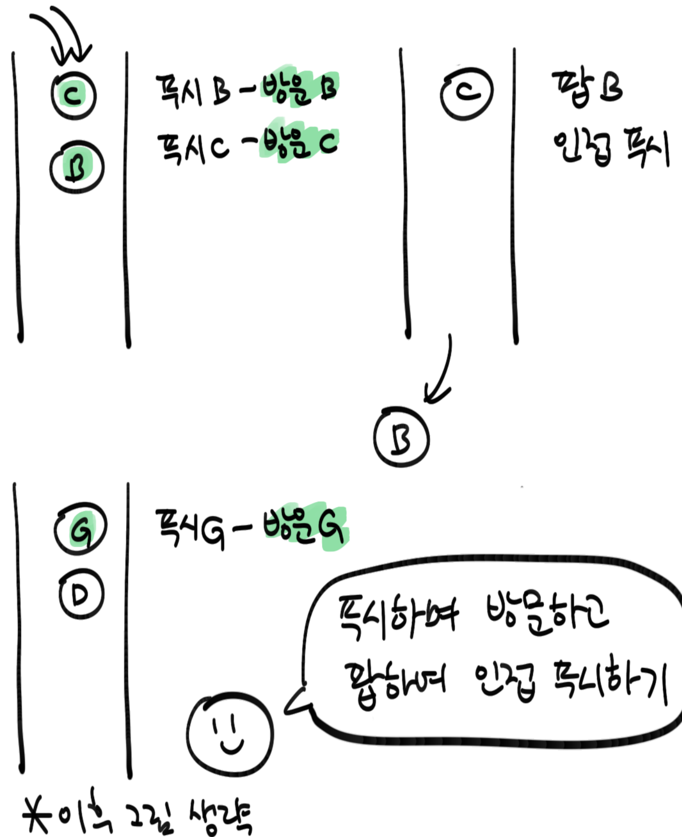
- 너비우선 탐색 : 큐로 구현하기



- 큐의 선입선출을 이용하여 구현
- 푸시하며 방문하기







11장 그래프 cont.

- 다익스트라 algorithm

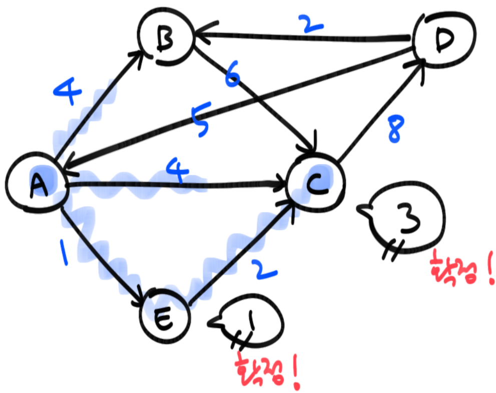
  1. 시작 노드를 설정, 최소 비용과 직전 노드를 저장할 공간 초기화
     a. 초기화값은 무한(∞)으로

  2. 해당 노드를 통해 방문할 수 있는 노드 중 현재까지 구한 최소 비용이 가장 적은 노드 선택
     a. 해당 노드를 거쳐서 각 노드까지 가는 최소 비용과 현재까지의 최소 비용을 비교하여 작은 값을 최소 비용으로 갱신

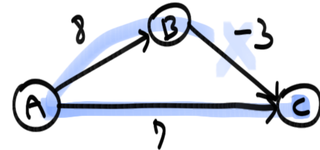
꽤 복잡해서 글보다는 본문 그림을 보길 추천!

- 파이프에 물을 흘려보내는 느낌의 알고리즘

what? 물을 흘려 보낸다?



- 음의 가중치가 있는 그래프에서는 동작 X



다익스트라 알고리즘은 그리디 선택을 하므로 A→C를 우선시하여 최단 경로를 찾지 못함

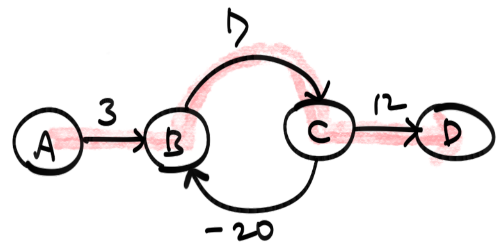
- 벨만-포드 algorithm

  1. 시작 노드 설정 (초기 노드 0, 나머지 ∞)
  2. 노드 개수 -1만큼 다음 연산 반복
     a. 시작 노드에서 갈 수 있는 인접 노드에 대하여 전체 노드를 거쳐갈 때 현재까지 구한 최소 비용보다 더 적은 최소 비용이 있는지 확인하여 갱신
  3. 2를 마지막으로 한 번 더 수행하여 음의 순환 확인.

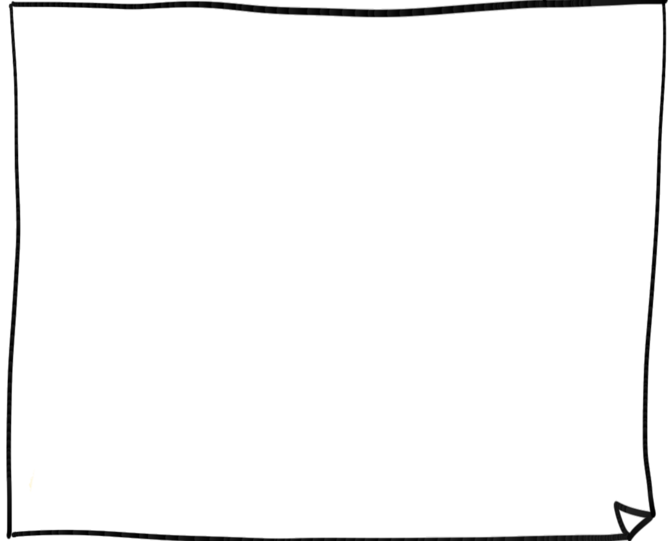
그림으로 표현하기는 지면이 부족! 본문 '11-3절 그래프 최단 경로 구하기'를 읽을 것!

- 다익스트라는 음의 순환에 안 빠진다?

  - 그게 아니라 다익스트라는 음의 순환을 알아차리지도 못함
  - 다익스트라는 확정한 노드를 재방문하지 않으므로 오른쪽 그래프를 A→B→C→D로 탐색하고 종료함.
  - 벨만-포드는 음의 순환에 빠진다기보다는 감지한다고 이해해야 함



12강 백트래킹



[용어] 백트래킹?

답을 찾는 과정에서 가능성이 없는 곳에서 back

[용어] 백트래킹 알고리즘?

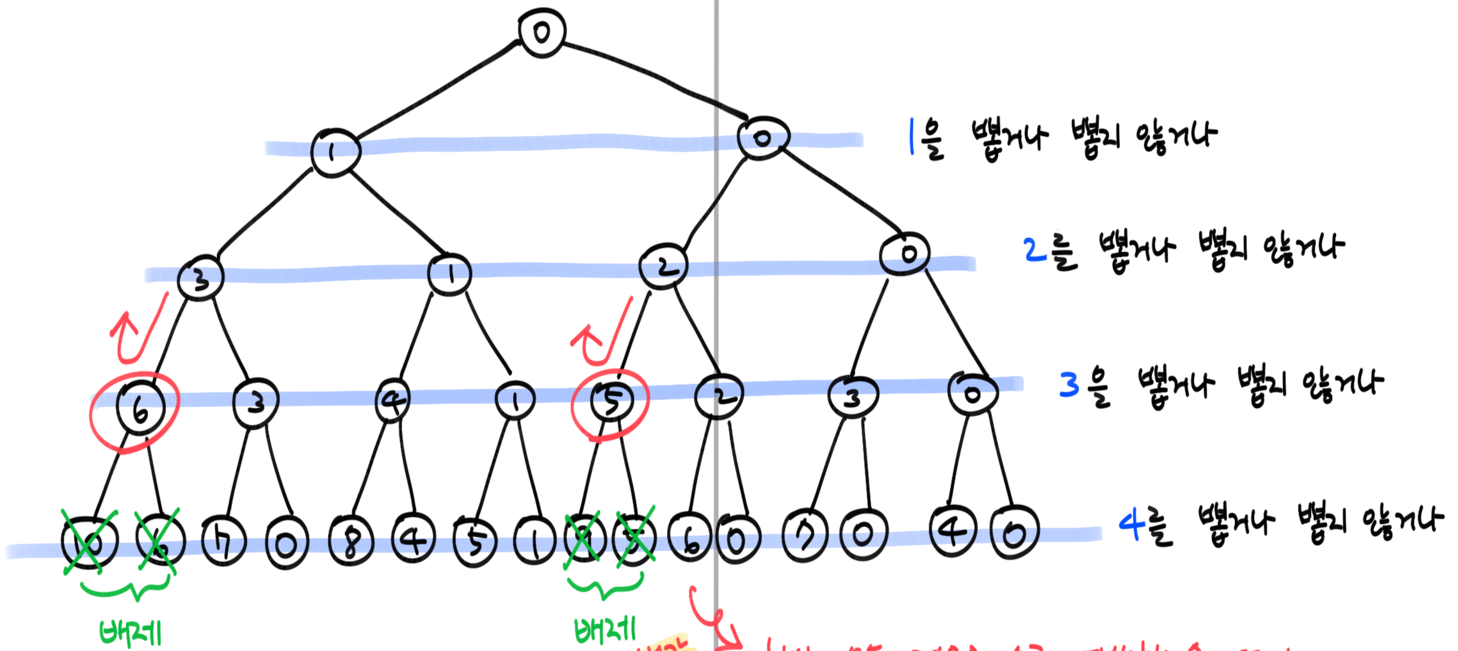
백트래킹을 활용하는 알고리즘

[용어] 유망 함수?

답의 가능성을 판단하는 함수

정리) 백트래킹 알고리즘은 가능성이 있는 모든 해를 탐색하되, 유망 함수로 탐색 효율을 올림.

- 1부터 4까지의 숫자를 조합하여 합이 5가 되는 경우는?

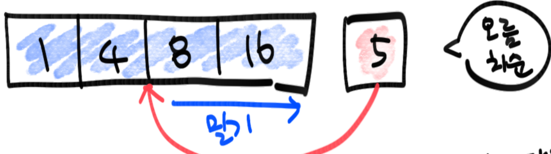


생각해보기 → 항상 모든 경우의 수를 탐색할 순 없다. ○로 표시한 것처럼 5 이상인 경우 백트래킹하는 유망함수를 사용하면?

생각해보기 → 숫자를 1부터 체크하는 특성을 이용해서 1~3을 조합했을때 합이 0이면 4를 제외하면?

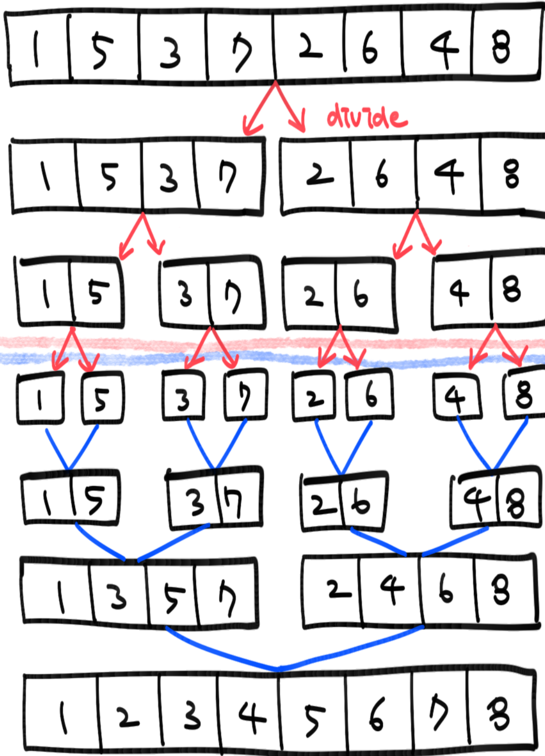
13강 정렬

- 삽입 정렬 insertion sort
  - 정렬 영역으로 적절한 값을 자리 잡는 방식

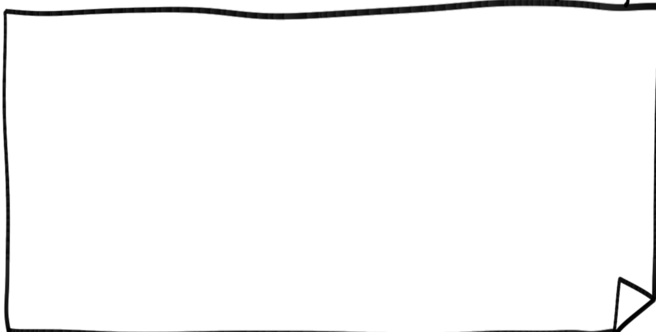


  - 최악의 경우 : 의도한 정렬과 정반대인 경우 $O(N^2)$
    - 오름차순을 기대했지만
    - 내림차순으로 주어질 때

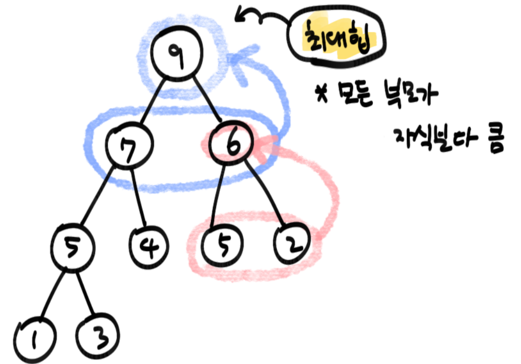
- 병합 정렬 merge sort
  - 최소 단위로 쪼개어 합치면서 정렬하는 방식

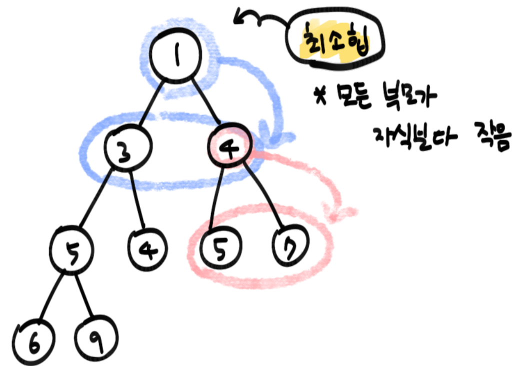


  - 시간 복잡도 : N개의 데이터를 logN번 나누고 병합하므로 $O(N\log N)$

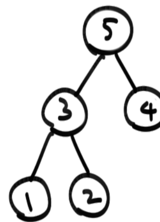


- 힙 heap
  - 이진 트리에 어떤 규칙을 부여한 방식
    - 최대 힙 : 부모 > 자식
    - 최소 힙 : 부모 < 자식





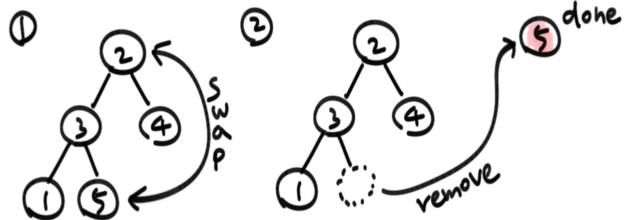
- 힙 정렬 heap sort
  - 힙을 이용한 정렬 : 최대힙, 내림차순

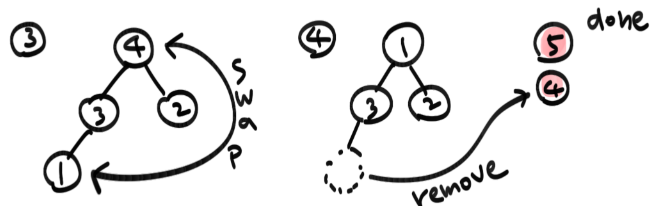


① swap (루트, 마지막 노드)

② 최대힙 유지를 위한 max_heapify() 실행

③ 힙 크기 -1 (remove)





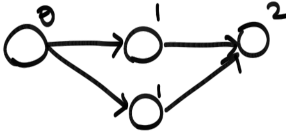
13장 정렬 cont.

• 위상 정렬 topological sort

- 진입 차수를 기준으로 정렬

용어 진입 차수?

그래프에서 자신을 향하는 간선의 개수



① 진입 차수가 0인 노드를 큐에 푸시
② 팝한 노드의 인접 노드 진입 차수 -1 } 반복
③ 팝하는 순서 자체가 정렬

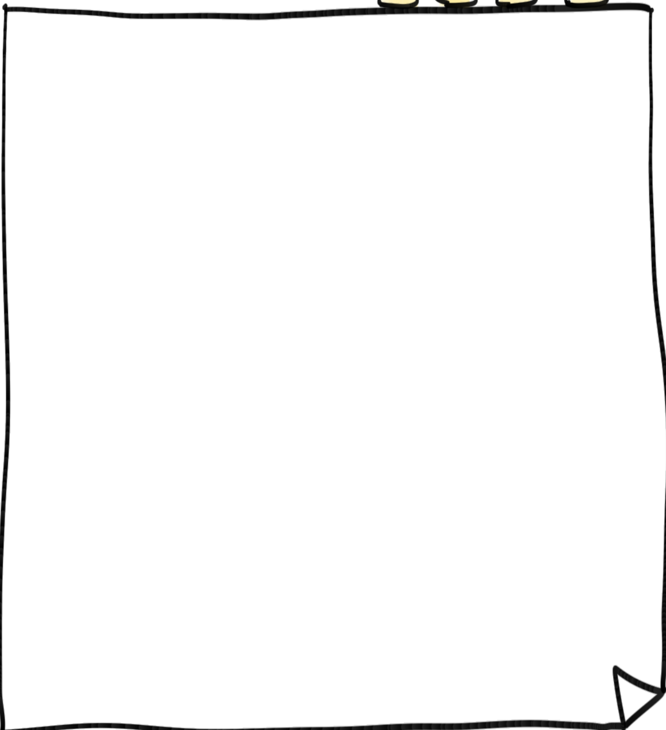
• 계수 정렬 counting sort

- 데이터의 빈도수로 정렬

1	1	2	3	1	2

index	value
1이	3개
2는	2개
3은	1개

- 단점은 음수나 간격이 큰 경우 활용하기가 애매

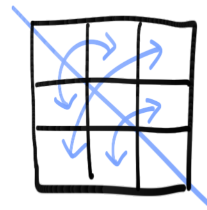


14장 시뮬레이션

• 행렬곱

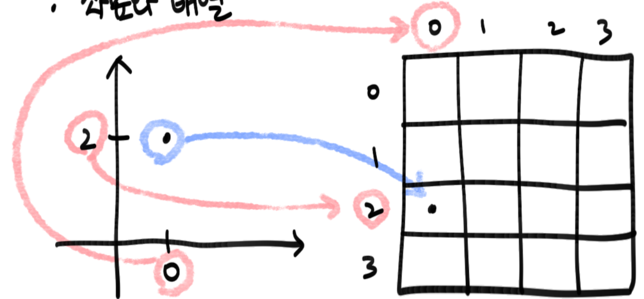
① = 1×1+2×2+3×3
② = 1×4+2×5+3×6
③ = 1×7+2×8+3×9

• 전치 행렬



* 대각선 기준으로 반대값을 swap

• 좌표와 배열



• 좌표 이동 with offset

- 1씩 이동하는 상황 (상하좌우)의 offset

for →

dy	1	0	-1	0
dx	0	1	0	-1
	상	우	하	좌

- 1씩 이동하는 상황 (8방향)의 offset

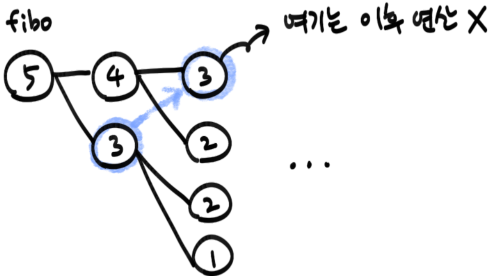
dy	1	0	-1	0	1	-1	-1	1
dx	0	1	0	-1	1	-1	1	-1
	상	우	하	좌	상우	하좌	하우	상좌

15강 동적 계획법

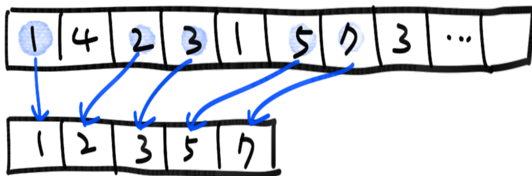
- 이미 해결한 문제로 반복되어 나오는 문제로 → 큰 문제를 해결

- 메모이제이션
  - 이미 해결한 문제를 기록
  - 메모이제이션 with 피보나치 수열

    If fibo(5)?



- 최장 증가 부분 수열 LIS
  - 원소가 오름차순을 유지하면서 길이가 가장 긴 수열



  - 동적 계획법으로 LIS 구하기

    dp[N] = arr[N]을 마지막 원소로 하는 LIS의 길이

    라고 할 때...

    $dp[N] = max(dp[K]) + 1$

    (단, K는 $1 <= K < N$, $arr[K] < arr[N]$)

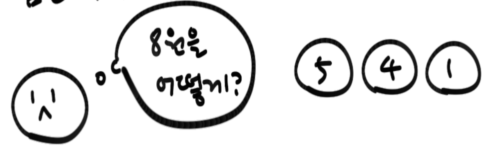
    $dp[1] = 1$

이외의 내용은 문제를 직접 풀며 동적계획법을 알아가는 것이 좋음!

16강 그리디

- 결정 순간마다 보이는 해 중 가장 좋은 것을 고름

- 거스름돈 주기



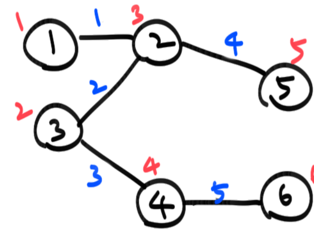
  - think greedy ...

  - but...

    is better

    적절한 곳에 써야 함

- 최소 신장 트리 minimum spanning tree
  - 신장 트리 spanning tree



모든 정점이 연결 + 사이클이 없음

  - so... 최소 신장 트리?

    신장 트리 중 간선 가중치 합이 최소!

- 최소 신장 트리 with 프림 알고리즘