

prepare_frontend_interview

자료구조

프론트엔드 기술 면접을 위한 핸드북 만들기

공부하고 알아야 할 만한 기본적인 자료구조에 대해 정리합니다

python으로 작성된 코드의 예시들이 있지만, JS와는 큰 차이가 없기 때문에 예시 코드로는 파이썬을 넣어 두었습니다

대학 정규 교육 과정으로 오랜 기간 공부해서 배운 내용이 아니기 때문에 깊이 면에서 부족할 수 있지만,

자료구조와 그에 따른 예시를 간단히 작성하여 자료 구조는 어떤 것들이 있는지 알기 위해 정리하고 있습니다

목차

- [자료구조란 무엇인가요](#)
 - 효율적으로 데이터를 관리해야 하는 이유 (예)
- [대표적인 자료구조는 어떤 것들이 있나요](#)
 - 선형 구조
 - 비 선형 구조
- [리스트](#)
- [큐](#)
- [스택](#)
- [링크드 리스트](#)
- [해쉬 테이블](#)
- [트리](#)
- [힙](#)
- [그래프](#)

자료구조란 무엇인가요

자료구조란 대량의 데이터를 효율적으로 관리할 수 있는 데이터의 구조를 의미합니다

코드 상에서 효율적으로 데이터를 처리하기 위해, 데이터 특성에 따라, 체계적으로 데이터를 구조화해야 합니다

어떤 데이터 구조를 사용하느냐에 따라, 코드의 효율이 달라질 수 있습니다

효율적으로 데이터를 관리해야 하는 이유 (예)

우편번호: 5자리 우편번호로 국가의 기초구역을 제공

5자리 우편번호에서 앞 3자리는 시, 군, 자치구를 표기, 뒤 2자리는 일련번호로 구성

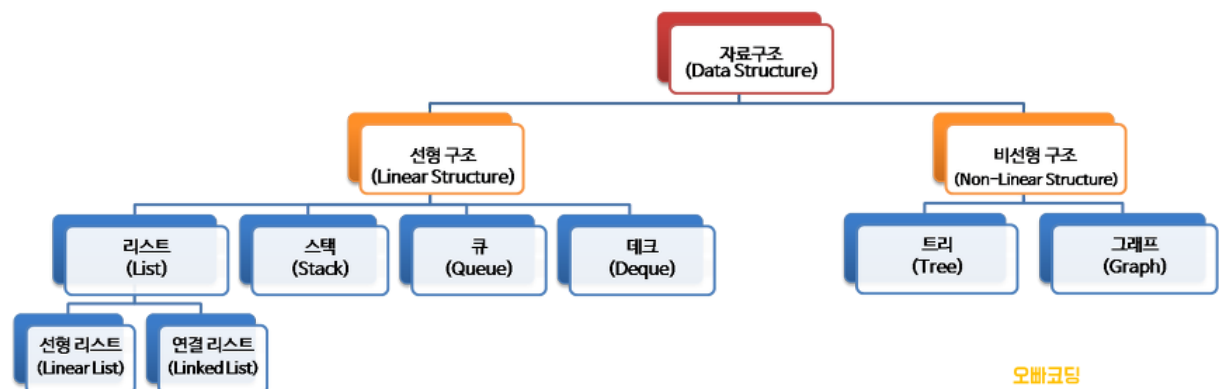
학생 관리: 학년, 반, 번호를 학생에게 부여해서, 학생부를 관리

XX학년, X반, X번 학생

만약 위 관리 기법이 없다면, 3000명 학생 중 특정 학생을 찾기 위해, 전체 학생부를 모두 훑어야 함

대표적인 자료구조는 어떤 것들이 있나요

자료구조	자료구조
선형 구조	비선형 구조
리스트	
스택	트리
큐	그래프
덱	



출처: <https://boycoding.tistory.com/32>

선형 구조란 무엇인가요

선형 구조 (Linear Structure)

데이터들이 일렬로 쭉 저장되어 있는 형태

비 선형 구조란 무엇인가요

비 선형 구조 (Non-Linear Structure)

데이터가 트리 형태로 저장되어 있다고 생각하고 사용하는 자료구조

리스트

- 데이터를 나열하고, 각 데이터를 인덱스에 대응하도록 구성한 데이터 구조
- 파이썬에서는 리스트 타입이 배열 기능을 제공함

리스트는 왜 필요한가요?

- 같은 종류의 데이터를 효율적으로 관리하기 위해
- 같은 종류의 데이터를 순차적으로 저장하기 위해

장점

- 빠른 접근이 가능하다
- 첫 데이터의 위치 [0]에서 상대적인 위치로 데이터에 접근이 가능하다 (인덱스 번호로 접근이 가능하다)

단점

- 데이터 추가/삭제가 어렵다
- 미리 최대 길이를 지정해야 함

make a list using C

- C 언어에서 배열을 사용할 경우 배열의 최대 길이를 사전에 정의해줘야 한다

```
#include <stdio.h>

int main(int argc, char * argv[])
{
    char country[3] = "US";
    printf ("%c%c\n", country[0], country[1]);
    printf ("%s\n", country);
    return 0;
}
```

make a list using python

- Python의 경우 배열(문자열)의 길이를 사전에 정해주지 않아도 동작한다
- JS또한 길이를 사전에 정해주지 않아도 동작한다
- 이러한 특징 때문에 파이썬과 JS의 배열이 자료구조의 배열과 같다고 볼 수는 없다

```
country = 'US'
print (country)

>>> US

country = country + 'A'
print(country)

>>> USA
```

1 차원 배열

```
# 1차원 배열: 리스트로 구현시
data_list = [1, 2, 3, 4, 5]

print(data_list)
>>>

[1,2,3,4,5]
```

2 차원 배열

```
# 2차원 배열: 리스트로 구현시
data_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(data_list)

[
  [1,2,3],
  [4,5,6],
  [7,8,9]
]
```

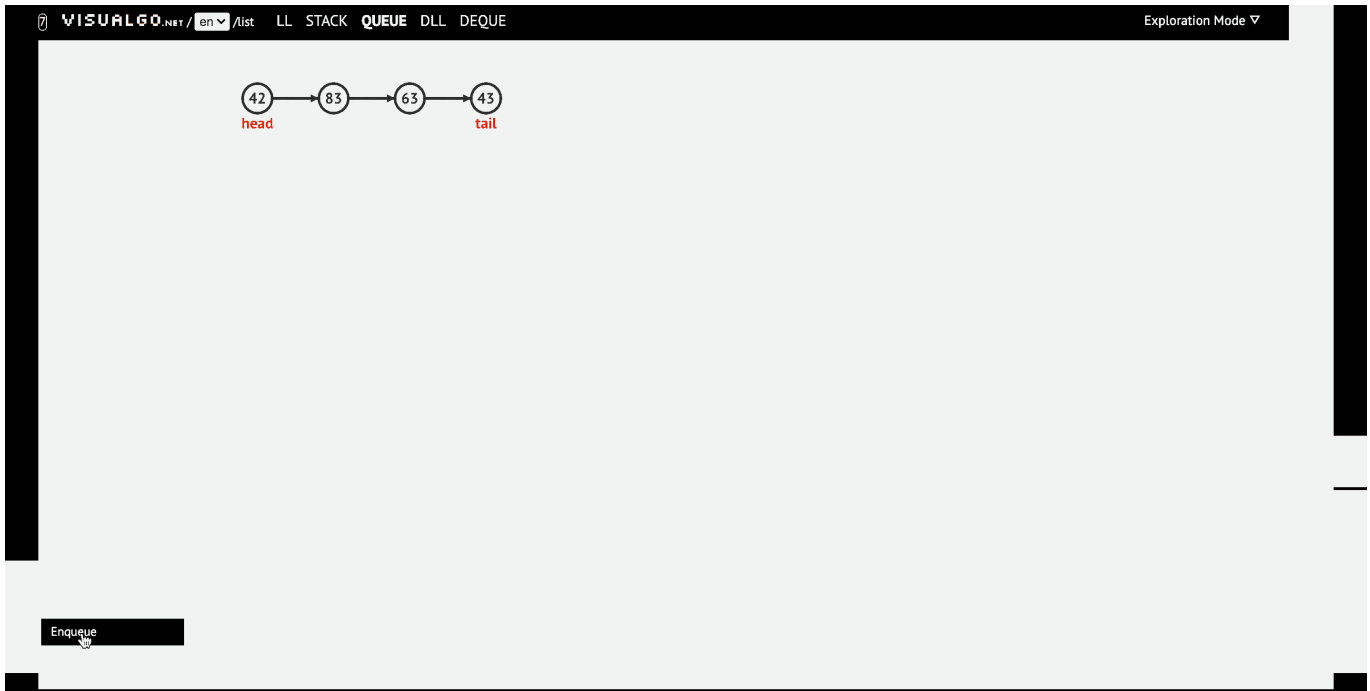
큐

Queue 구조

- 줄을 서는 행위와 유사
- 가장 먼저 넣은 데이터를 가장 먼저 꺼낼 수 있는 구조
- 음식점에서 가장 먼저 줄을 선 사람이 제일 먼저 음식점에 입장하는 것과 동일
- FIFO(First-In, First-Out) 또는 LILO(Last-In, Last-Out) 방식으로 스택과 꺼내는 순서가 반대



- 출처: <http://www.stoimen.com/blog/2012/06/05/computer-algorithms-stack-and-queue-data-structure/>



알아둘 용어

- Enqueue: 큐에 데이터를 넣는 기능 (JS에서 push)
- Dequeue: 큐에서 데이터를 꺼내는 기능 (JS에서 shift)

파이썬 queue 라이브러리 활용해서 큐 자료 구조 사용하기

queue 라이브러리에는 다양한 큐 구조로 ① Queue(), ② LifoQueue(), ③ PriorityQueue() 제공

프로그램을 작성할 때 프로그램에 따라 적합한 자료 구조를 사용

Queue(): 가장 일반적인 큐 자료 구조

▶ 세부정보

```
# 큐 (Queue)

import queue

# FIFO QUEUE (일반적인 구조의 큐) 사용하기

data_queue = queue.Queue()

print('data_queue:', data_queue)
print('type:', type(data_queue))

print()

# 데이터 삽입 (put)

data_queue.put(1)
```

```

data_queue.put(2)
data_queue.put(3)

print('data_queue.qsize():', data_queue.qsize()) >>> 3

print()

# 데이터 추출 (get)

print('data_queue.get():', data_queue.get()) >>> 1
print('data_queue.get():', data_queue.get()) >>> 2
print('data_queue.get():', data_queue.get()) >>> 3

```

LifoQueue(): 나중에 입력된 데이터가 먼저 출력되는 구조 (스택 구조라고 보면 됨)

▶ 세부정보

```

import queue

# LIFO QUEUE (스택과 같은 구조의 큐) 사용하기

data_queue = queue.LifoQueue()

print('data_queue:', data_queue)
print('type:', type(data_queue))

print()

# 데이터 삽입 (put)

data_queue.put(1)
data_queue.put(2)
data_queue.put(3)

print('data_queue.qsize():', data_queue.qsize()) >>> 3

print()

# 데이터 추출 (get)

print('data_queue.get():', data_queue.get()) >>> 3
print('data_queue.get():', data_queue.get()) >>> 2
print('data_queue.get():', data_queue.get()) >>> 1

```

PriorityQueue(): 데이터마다 우선순위를 넣어서, 우선순위가 높은 순으로 데이터 출력

▶ 세부정보

```
# PriorityQueue() 우선 순위가 있는 큐 만들기

import queue

data_queue = queue.PriorityQueue()

# - 튜플 형식 ( , )으로 데이터를 삽입한다
# - 숫자가 낮을수록 우선순위가 높다 (1 <<<< 우선 순위가 높음 , .... , 15 <<<< 우선 순위가 낮음)
# - 우선순위가 높은 튜플이 먼저 Dequeue 된다

data_queue.put((10, "korea"))
data_queue.put((5, 1))
data_queue.put((15, "china"))

print()

print(data_queue.qsize()) # >>> 3

print()

print(data_queue.get()) # >>> (5,1)
print(data_queue.get()) # >>> (10, 'korea')
print(data_queue.get()) # >>> (15, 'china')
```

어디에 큐가 많이 쓰일까?

멀티 태스킹을 위한 프로세스 스케줄링 방식을 구현하기 위해 많이 사용된다

queue 직접 구현하기

```
queue_list = list()

def enqueue(data):
    queue_list.append(data)

def dequeue():
    data = queue_list[0]
    del queue_list[0]
    return data

for index in range(10):
    enqueue(index)

print('queue_list:', queue_list)
# >>> queue_list: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
dequeue()

print('queue_list:', queue_list)
# >>> queue_list: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

스택

Stack 구조

- 데이터를 제한적으로 접근할 수 있는 구조
- 한쪽 끝에서만 자료를 넣거나 뺄 수 있는 구조
- 가장 나중에 쌓은 데이터를 가장 먼저 빼낼 수 있는 데이터 구조

큐(Queue)	스택(Stack)
F.I.F.O	L.I.F.O
First In First Out	Last In First Out

스택 구조

스택은 LIFO(Last In, First Out) 또는 FILO(First In, Last Out) 데이터 관리 방식을 따름

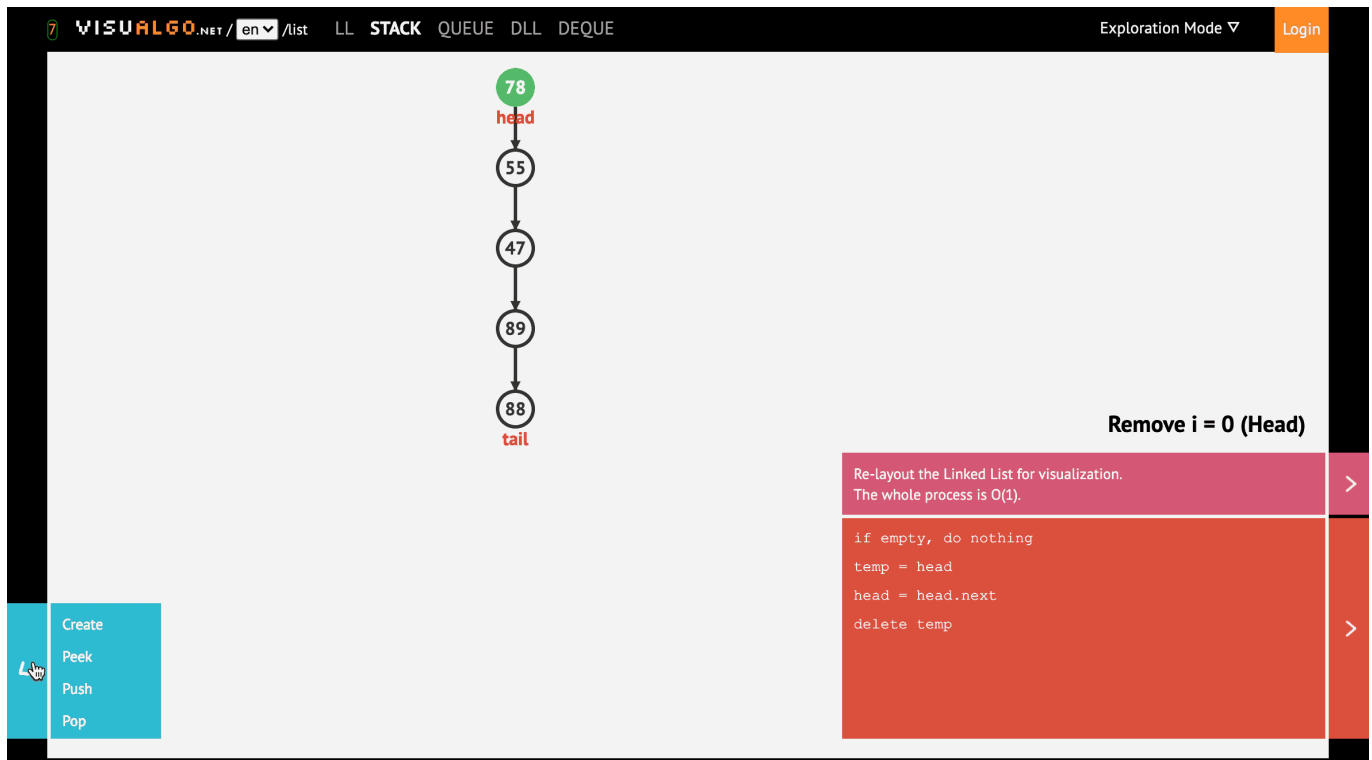
LIFO: 마지막에 넣은 데이터를 가장 먼저 추출하는 데이터 관리 정책 FILO: 처음에 넣은 데이터를 가장 마지막에 추출하는 데이터 관리 정책

대표적인 스택의 활용

컴퓨터 내부의 프로세스 구조의 함수 동작 방식

주요 기능

push(): 데이터를 스택에 넣기 pop(): 데이터를 스택에서 꺼내기



스택의 장단점

- 장점 🔥
 - 구조가 단순해서, 구현이 쉽다
 - 데이터 저장/읽기 속도가 빠르다
- 단점 🔥
 - 데이터 최대 개수를 미리 정해야 한다
 - 파이썬의 경우 재귀 함수 호출은 1000번까지 가능하다
 - 저장 공간의 낭비가 발생할 수 있다
 - 미리 최대 개수만큼 저장 공간을 확보해야 한다
- 스택은 단순하고 빠른 성능을 위해 사용되므로, 보통 배열 구조를 활용해서 구현하는 것이 일반적임

파이썬을 통해 스택 구조 살펴보기

list 자료형의 내장 메서드로 `append(push)`, `pop()` 메서드를 제공한다

```

# 스택 (Stack)

# 큐와 달리 파이썬, JS 에서 기본 제공하는 리스트를 통해서 구현이 가능하다
# 큐의 경우 data = queue.Queue() 와 같이 인스턴스로 만들어서 사용했었음

# append: 삽입하기 (push)
# pop: 꺼내기 (pop)

```

```
data_stack = list()

data_stack.append(1)
data_stack.append(2)

print('data_stack:', data_stack) >>> [1,2]

print()

data_stack.pop()

print('① data_stack.pop():', data_stack) >>> [1]

print()

data_stack.pop()

print('② data_stack.pop():', data_stack) >>> [0]
```

프로그래밍 연습

리스트 변수로 스택을 다루는 pop, push 기능 구현해보기

```
stack_list = list()

def push(data):
    stack_list.append(data)

def pop():
    data = stack_list[-1]
    del stack_list[-1]
    return data

for elem in range(10):
    push(elem)

print('stack_list', stack_list)
# >>> stack_list [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print()

pop()
```

```
print('stack_list', stack_list)
# >>> stack_list [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

링크드 리스트

Linked List 구조

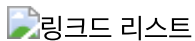
- 연결 리스트라고도 한다
- 배열은 순차적으로 연결된 공간에 데이터를 나열하는 데이터 구조
- 링크드 리스트는 떨어진 곳에 존재하는 데이터를 화살표로 연결해서 관리하는 데이터 구조
- 본래 C언어에서는 주요한 데이터 구조이지만, 파이썬은 리스트 타입이 링크드 리스트의 기능을 모두 지원

링크드 리스트 기본 구조와 용어

노드(Node): 데이터 저장 단위 (데이터값, 포인터) 로 구성

포인터(pointer): 각 노드 안에서, 다음이나 이전의 노드와의 연결 정보를 가지고 있는 공간

일반적인 링크드 리스트 형태



(출처: wikipedia, https://en.wikipedia.org/wiki/Linked_list)

링크드 리스트의 장단점 (전통적인 C 언어에서의 배열과 링크드 리스트)

- 장점
 - 미리 데이터 공간을 할당하지 않아도 된다
 - <-> 배열은 미리 데이터 공간을 할당해야 함
 - 데이터를 추가 할때마다 동적으로 크기가 늘어난다
 - 원소 검색 시 첫 번째 노드부터 마지막 노드까지 일일이 확인하기 때문에 $O(n)$ 의 시간 복잡도를 갖는다
 - 삽입 또는 삭제 연산 시에 해당 원소를 검색한 후 삭제, 삽입 연산이 이루어지므로 $O(n)$ 의 시간 복잡도를 갖는다
 - 즉, 삽입, 삭제가 잦은 경우 Linked List를 사용하는 것이 좋다
 - why? 배열 경우 배열의 크기를 사전에 정해줘야 하기 때문
- 단점
 - 연결을 위한 별도 데이터 공간이 필요하므로, 저장 공간 효율이 높지 않음
 - 연결 정보를 찾는 시간이 필요하므로 접근 속도가 느림
 - 중간 데이터 삭제시, 앞뒤 데이터의 연결을 재구성해야 하는 부가적인 작업 필요
 - 검색이 잦은 경우 배열을 사용하는 것이 좋다

더블 링크드 리스트 구조

더블 링크드 리스트(Doubly linked list) 기본 구조

- 단방향 링크드 리스트의 경우 반드시 우리가 설정한 head 차례로 데이터를 찾아갔음
- 더블 링크드 리스트는 앞 뒤 방향 모두 노드 탐색이 가능함

- 이중 연결 리스트라고도 함
- 장점: 양방향으로 연결되어 있어서 노드 탐색이 양쪽으로 모두 가능

더블 링크드 리스트 구조

(출처: wikipedia, https://en.wikipedia.org/wiki/Linked_list)

해쉬 테이블

해쉬 테이블(Hash Table) 구조

해쉬 테이블(Hash Table): 키(Key)에 데이터(Value)를 저장하는 데이터 구조

- Key를 통해 바로 데이터를 받아올 수 있으므로, 속도가 획기적으로 빨라짐
- (특정 조건에 맞춰 배열을 모두 순회할 필요가 없다는 소리)
- 파이썬 딕셔너리(Dictionary) 타입이 해쉬 테이블의 예: Key를 가지고 바로 데이터(Value)를 꺼낼 수 있음
- 보통 배열로 미리 Hash Table 사이즈만큼 생성 후에 사용 (공간과 탐색 시간을 맞바꾸는 기법)
- 해쉬 테이블의 길이(슬롯의 개수)를 늘리는 방법
- 단, 파이썬에서는 해쉬를 별도로 구현할 이유가 없음 (딕셔너리 타입 { 'key': 'value' } 을 제공하기 때문)
- JS 또한 별도로 구현할 필요가 없다 (객체 타입 { key: 'value' } 을 제공하기 때문)

알아둘 용어

- 해쉬(Hash): 임의 값을 고정 길이로 변환하는 것
- 해쉬 테이블(Hash Table): 키 값의 연산에 의해 직접 접근이 가능한 데이터 구조
- 해싱 함수(Hashing Function): Key를 해싱 함수로 연산해서, 해쉬 값을 알아내고, 이를 기반으로 해쉬 테이블에서 해당 Key에 대한 데이터 위치를 일관성있게 찾을 수 있음
- 슬롯(Slot): 한 개의 데이터를 저장할 수 있는 공간
- 저장할 데이터에 대해 Key를 추출할 수 있는 별도 함수도 존재할 수 있음

해시테이블

해쉬 테이블의 장단점과 주요 용도

- 장점
 - 데이터 저장/읽기 속도가 빠르다. (검색 속도가 빠르다.)
 - 해쉬는 키에 대한 데이터가 있는지(중복) 확인이 쉽다
 - 해쉬는 키를 바탕으로 한 데이터의 여부(유무)를 파악하기 쉽다
- 단점
 - 일반적으로 저장공간이 좀 더 많이 필요하다
 - 여러 키에 해당하는 주소가 동일할 경우 충돌을 해결하기 위한 별도 자료구조가 필요하다
 - 해시 함수를 통해 나눠져 저장되는 해쉬 테이블의 주소가 같은데, 별도의 처리를 하지 않을 경우
 - 키를 바탕으로 한 데이터가 덮어쓰워질 가능성이 있다.
 - 중복이 일어나지 않도록 해쉬 테이블의 공간을 넓게 설계해야 한다
- 주요 용도

- 검색이 많이 필요한 경우
- 저장, 삭제, 읽기가 빈번한 경우
- 캐쉬 구현시 (중복 확인이 쉽기 때문)
- 캐쉬는 동일한 페이지를 불러오는 경우 <https://naver.com>, 변경되는 데이터 이외에는
- 사용자의 캐쉬 메모리에 저장하여 서버로부터 불러오는 데이터의 양을 관리하기 위한 메모리이다

트리

트리(Tree) 구조

트리: 노드(node)와 브랜치(branch)를 이용해서, 사이클을 이루지 않도록 구성된 데이터 구조

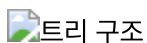
- 사이클은 아래 이미지를 기준으로 5 - 3 - 6 순으로 원을 그리며 탐색하는 것을 의미
- 트리 구조에서 Siblings 끼리는 브랜치로 이어지지 않는다 🔥

실제로 어디에 많이 사용되나?

- 트리 중 이진 트리 (Binary Tree) 형태의 구조로, 탐색(검색) 알고리즘 구현을 위해 많이 사용됨
- 트리 내부에 어떤 값을 가진 데이터가 존재하는지의 유무를 파악하기 쉽다
- 배열을 통해 배열의 길이만큼 전부 순회하는 것 보다 기준(left right)을 가지고 탐색하므로 더욱 빠르다

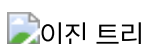
알아둘 용어

- Node: 트리에서 데이터를 저장하는 기본 요소 (데이터와 다른 연결된 노드에 대한 Branch 정보 포함)
- Root Node: 트리 맨 위(최 상단)에 있는 노드
- Level: 최상위 노드를 Level 0으로 하였을 때, 하위 Branch로 연결된 노드의 깊이를 나타냄
- Parent Node: 어떤 노드의 다음 레벨에 연결된 노드
- Child Node: 어떤 노드의 상위 레벨에 연결된 노드
- Leaf Node (Terminal Node): Child Node가 하나도 없는 노드
- Sibling (Brother Node): 동일한 Parent Node를 가진 노드
- Depth: 트리에서 Node가 가질 수 있는 최대 Level (깊이를 나타냄)



이진 트리(Binary Tree)와 이진 탐색트리(Binary Search Tree)

- 이진 트리: 노드의 최대 브랜치가 2개인 트리
 - 이진 트리구조에서 워낙 이진 탐색 트리 형식으로 많이 쓰기 때문에 이진 트리 = 이진 탐색 트리라고 생각하는 경우도 있다
 - 하지만 둘은 같지 않음
 - 이진 트리는 루트 노드의 최대 브랜치가 2개인 트리이며, 사이클을 이루지 않도록 구성된 데이터 구조이지만,
 - 이진 탐색 트리는 해당 이진 트리의 특징을 바탕으로 특정 조건을 붙인 트리이다
- 이진 탐색 트리: 이진 트리에 다음과 같은 추가적인 조건이 있는 트리
 - 왼쪽 노드는 해당 노드보다 작은 값, 오른쪽 노드는 해당 노드보다 큰 값을 가지고 있음




자료 구조 이진 탐색 트리의 장점과 주요 용도

- 주요 용도: 데이터 검색(탐색)
- 장점: 탐색 속도를 개선할 수 있음

이진 트리와 정렬된 배열간의 탐색 비교 🔥

- steps: 단계의 가짓수를 보면 이진 트리가 훨씬 빠르게 검색을 할 수 있다

 이진 탐색 트리

(출처: <https://www.mathwarehouse.com/programming/gifs/binary-search-tree.php#binary-search-tree-insertion-node>)

힙

힙(Heap) 이란?

- 자료구조의 힙: 데이터에서 최대값과 최소값을 빠르게 찾기 위해 고안된 완전 이진 트리(Complete Binary Tree)
- **완전 이진 트리**: 노드를 삽입할 때 최하단의 왼쪽 노드부터 차례대로 삽입하는 트리
- 트리를 기반으로 한 변형된 정책을 쓰고 있다고 생각하면 됨
- js의 실행 컨텍스트에서 객체를 담아두는 공간인 Heap 메모리와 자료구조에서의 Heap은 다름 🔥

메모리의 힙?

출처: <http://sjkitpro.blogspot.com/2018/07/heap.html>

▶ 세부정보

메모리 힙(Heap)

프로그램이 실행되면 아래 그림과 같이 4개의 메모리 영역을 가지게 된다.

이중 Heap 영역은 사용자가 동적할당을 할 경우 메모리에 저장된다.

C언어에서는 malloc, java에서는 new 키워드로 Heap 영역에 메모리를 할당할 수 있다.

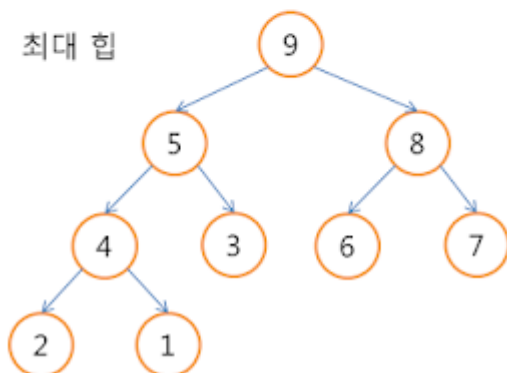
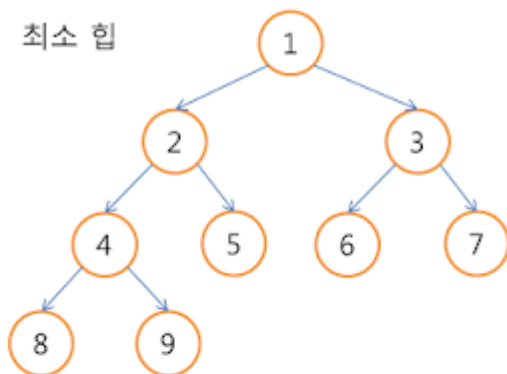
Heap 메모리의 해제는 C에서는 free 함수로 해제하며, java에서는 JVM에서 가비지 컬렉터가 지우는 시점에 해제된다.



자료구조의 힙(Heap)

힙은 완전 이진 트리이다. 최소 값 혹은 최대 값을 찾아내는 연산에 적합하다.

힙에는 최소 힙과 최대 힙이 있다. 최대 힙은 루트노드로 갈수록 값이 커지며, 최소 힙은 루트노드로 갈수록 값이 작아진다.



자료구조 힙

- 힙을 사용하는 이유
 - 배열에 데이터를 넣고, 최대값과 최소값을 찾으려면 $O(n)$ 이 걸린다 (전체를 순회해야 하기 때문)
 - 이에 반해, 힙에 데이터를 넣고, 최대값과 최소값을 찾으려면 $O(\log n)$ 이 걸림
 - 우선 순위 큐와 같이 최대값 또는 최소값을 빠르게 찾아야 하는 자료구조 및 알고리즘 구현 등에 활용됨

시간 복잡도 순서

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

힉(Heap) 구조

- 힉은 최대값을 구하기 위한 구조 (최대 힉, Max Heap)와, 최소값을 구하기 위한 구조 (최소 힉, Min Heap)로 분류할 수 있음
- 힉은 다음과 같이 두 가지 조건🔥 을 가지고 있는 자료구조임

① 각 노드의 값은 해당 노드의 자식 노드가 가진 값보다 크거나 같다. (최대 힉의 경우) - 최소 힉의 경우는 각 노드의 값은 해당 노드의 자식 노드가 가진 값보다 크거나 작음

② 완전 이진 트리 형태를 가진다

힉과 이진 탐색 트리(BST, Binary Search Tree)의 공통점과 차이점 🔥

- 공통점: 힉과 이진 탐색 트리는 모두 이진 트리임 (자식 노드가 최대 2개가 있는 트리 구조)
- 차이점:
 - 힉은 각 노드의 값이 자식 노드보다 크거나 같음(Max Heap의 경우)
 - 이진 탐색 트리는 왼쪽 자식 노드의 값이 가장 작고, 그 다음 부모 노드, 그 다음 오른쪽 자식 노드 값이 가장 큼
 - 힉은 이진 탐색 트리의 조건인 자식 노드에서 작은 값은 왼쪽, 큰 값은 오른쪽이라는 조건은 없음
 - 힉은 왼쪽 및 오른쪽 자식 노드의 값은 오른쪽이 클 수도 있고, 왼쪽이 클 수도 있음
 - 조건을 정해둔 것이 없다는 뜻(힉 구조에 들어오면서 판별한다)
 - 이진 탐색 트리의 목적은 탐색을 위한 구조이다 (값의 유무 판별)
 - 힉은 최대/최소값 검색을 위한 구조 중 하나로 이해하면 됨 (루트 노드에는 항상 최대/최소 값이 존재하므로)

힉과 BST의 차이

그래프

그래프 (Graph) 란?

- 그래프는 실제 세계의 현상이나 사물을 정점(Vertex) 또는 노드(Node)와 간선(Edge)로 표현하기 위해 사용
- 예제 집에서 회사로 가는 경로를 그래프로 표현한 예

그래프

그래프 (Graph) 관련 용어

노드(Node) = 정점(Vertex): 위치를 말함

간선(Edge) = 링크 = 브랜치: 위치 간의 관계를 표시한 선으로 노드를 연결한 선이라고 보면 됨 (link 또는 branch라고도 함)

인접 정점(Adjacent Vertex): 간선으로 직접 연결된 정점(또는 노드)

눈여겨 볼 그래프 종류

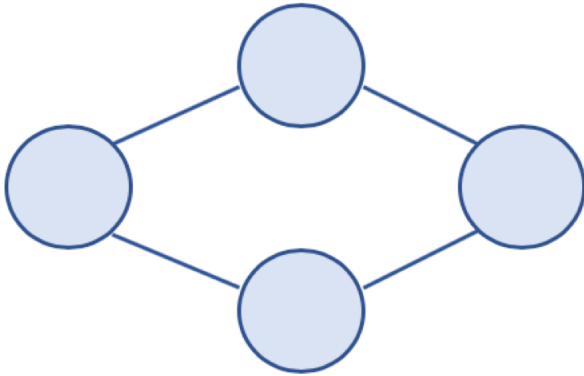
- 무방향 그래프
- 방향 그래프

- 가중치 그래프

그래프 (Graph) 종류

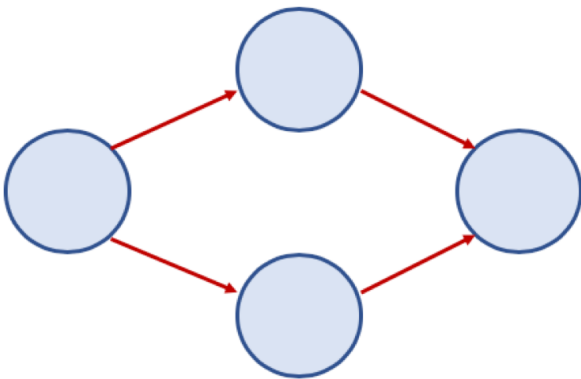
무방향 그래프

- 방향이 없는 그래프
- 간선을 통해, 노드는 양방향으로 갈 수 있음
- 보통 노드 A,B가 연결되어 있을 경우 (A,B) 또는 (B,A)로 표기



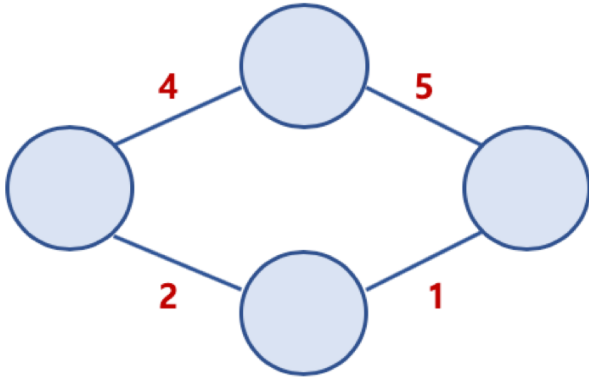
방향 그래프 (Directed Graph)

- 간선에 방향이 있는 그래프
- 보통 노드 A, B가 A -> B 로 가는 간선으로 연결되어 있을 경우, <A, B> 로 표기 (<B, A> 는 B -> A 로 가는 간선이 있는 경우이므로 <A, B> 와 다름)



가중치 그래프 (Weighted Graph) 또는 네트워크 (Network)

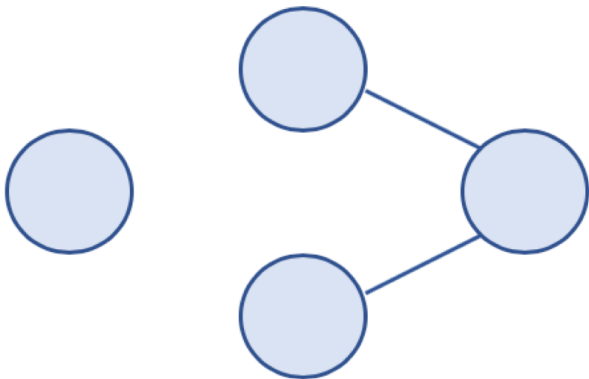
- 간선에 비용 또는 가중치가 할당된 그래프



연결 그래프 (Connected Graph) 와 비연결 그래프 (Disconnected Graph)

- 연결 그래프 (Connected Graph)
 - 무방향 그래프에 있는 모든 노드에 대해 항상 경로가 존재하는 경우
- 비연결 그래프 (Disconnected Graph)
 - 무방향 그래프에서 특정 노드에 대해 경로가 존재하지 않는 경우

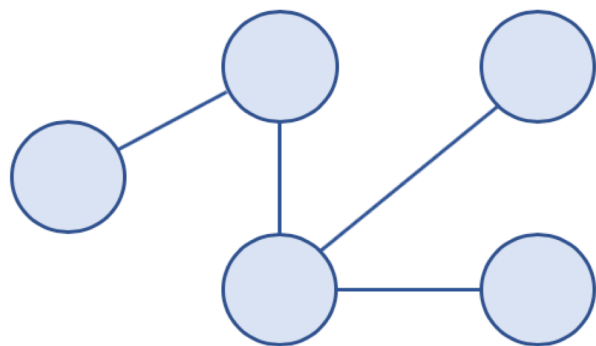
비연결 그래프 예시 이미지



사이클 (Cycle) 과 비순환 그래프 (Acyclic Graph)

- 사이클 (Cycle)
 - 단순 경로의 시작 노드와 종료 노드가 동일한 경우
- 비순환 그래프 (Acyclic Graph)
 - 사이클이 없는 그래프

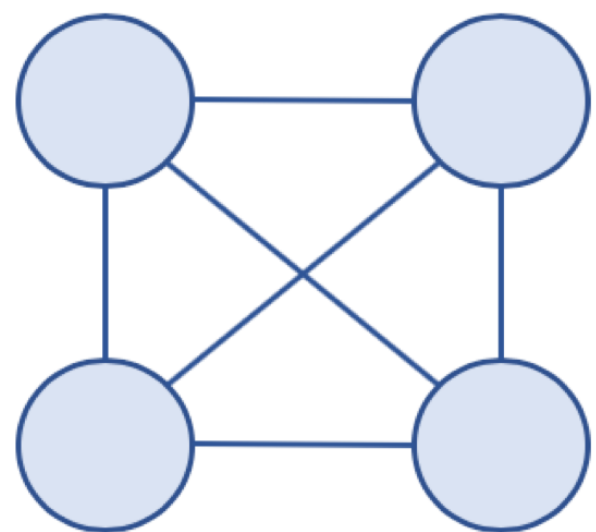
비순환 그래프 예



완전 그래프 (Complete Graph)

- 그래프의 모든 노드가 서로 연결되어 있는 그래프

완전 그래프 예



그래프와 트리의 차이

트리는 그래프 중에 속한 특별한 종류라고 볼 수 있음 (트리는 그래프의 한 종류이다)

	그래프	트리
정의	노드와 노드를 연결하는 간선으로 표현되는 자료 구조	그래프의 한 종류, 방향성이 있는 비순환 그래프
방향성	방향 그래프, 무방향 그래프 둘 다 존재함	방향 그래프만 존재함
사이클	사이클 가능함, 순환 및 비순환 그래프 모두 존재함	비순환 그래프로 사이클이 존재하지 않음
루트 노드	루트 노드 존재하지 않음	루트 노드 존재함
부모/자식 관계	부모 자식 개념이 존재하지 않음	부모 자식 관계가 존재함