1. 그래프, 트리의 자료구조 정의와 비교

	그래프	트리
정의	노드와 그 노드를 연결하는 간선을 하나로 모아놓은 자료구조	그래프의 한 종류로, 방향성이 있는 비순환 그래프 중 하나이다 (Directed Acyclic Graph-DAG). 요소를 계층적으로 저장하기 위한 추상자료형
방향성	방향 그래프 (Directed), 무방향 그래프 (Undirected)가 모두 존재	방향 그래프 (Directed)
사이클	사이클 (Cycle) 가능. 자체간선 (self-loop)도 가능, 순환그래프 (Cyclic), 비순환 그래프 (Acyclic) 모두 존재	사이클 (Cycle) 불가능, 자체 간선 (self-loop) 불가능, 비순환 그래프 (Acyclic)
루트 노드	루드 노드의 개념	한개의 루트 노드만이 존재, 모든 자식 노드는 한 개의 부모 노드만을 가짐
부모-자식	부모-자식 개념이 없음	부모-자식 관계가 있음. Top-bottom 또는 bottom-top 으로 이루어짐
모델	네트워크 모델	계층 모델
순회	DFS, BFS	DFS, BFS
에지	그래프에 따라 에지의 수가 다름. 에지가 없을 수도 있다.	노드가 N인 트리는 항상 N-1의 에지를 가짐
경로	임의의 두 노드 간 경로는 여러가지	임의의 두 노드 간 경로는 유일
종류	지도, 지하철 노선도, 전기회로 소자, 도로	이진트리, 이진탐색트리, 이진힙

2. Big-O, Big-Omega, Big-Theta

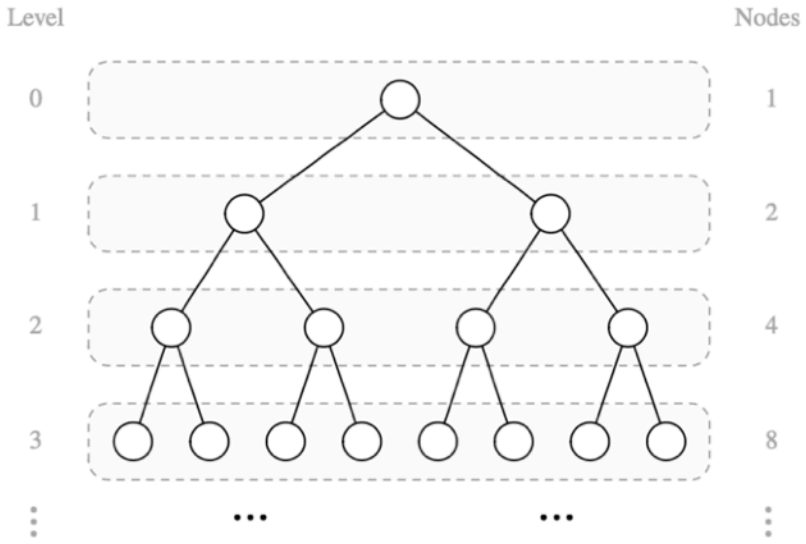
- Big-O: 두 함수 $f(n), g(n)$이 있을 때, $n_1 \leq n, f(n) \leq C \times g(n)$이 성립하는 상수 C, n_1이 존재하면 $f(n) = O(g(n))$이다.
- Big-Omega: 두 함수 $f(n), g(n)$이 있을 때, $n_1 \leq n, f(n) \geq C \times g(n)$이 성립하는 상수 C, n_1이 존재하면 $f(n) = \Omega(g(n))$이다.
- Big-Theta: 두 함수 $f(n), g(n)$이 있을 때, $n_1 \leq n, C_1 \times g(n) \leq f(n) \leq C_2 \times g(n)$이 성립하는 상수 C_1, C_2, n_1이 존재하면 $f(n) = \Theta(g(n))$이다.

4. Formal definition of "Big-Oh" notation을 쓰고, 성능 평가에서 실제 실험과 비교할 때 강점과 약점을 설명

- Big-Oh는 실제 알고리즘이 가지는 최악의 경우를 보여주기 때문에, 알고리즘의 Big-Oh를 계산하게 되면 알고리즘의 시간은 그 시간보다는 작다는 보장을 받고 예상 시간이 초과되지 않는다는 장점이 있음. 그러나 Big-Oh의 값으로 무조건적으로 큰 값을 할당해도 정의상 문제가 되지 않기 때문에 가능한 Big-Oh들 중에서 알고리즘에 가장 가까운 상한만이 가장 의미를 갖는다.

5. Binary tree

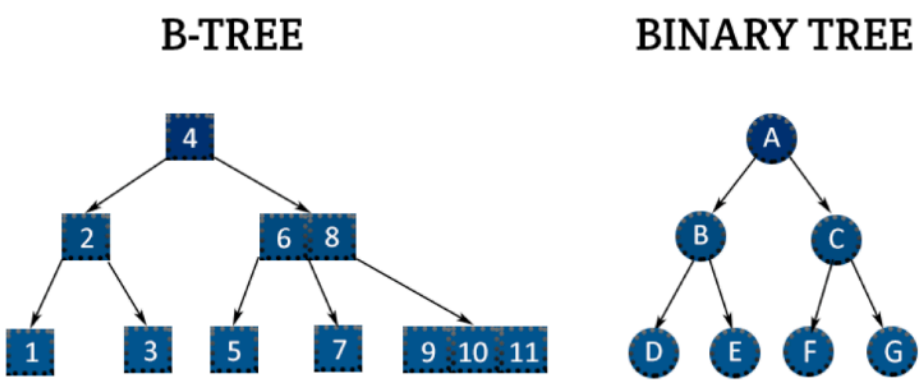
- Binary tree는 트리 자료구조 중 자식 노드가 최대 2개인 노드들로 구성된 트리를 말한다. 이진트리 중 자식노드가 0개 또는 2개만으로 구성된다면 'proper binary tree' 혹은 'full binary tree'라고 한다.



- 이진 트리의 노드 수와 레벨의 관계: d 레벨에서의 노드의 수 $\leq 2^d$

6. B tree

- Balanced binary tree: Binary tree에서 발전되어 트리의 높이에 불균형이 생길 때 자동으로 밸런스를 맞추는 tree 자료구조로, AVL tree, red-black tree, B tree, B+ tree, B* tree 등이 있다.
- B tree는 하나의 노드가 가질 수 있는 key의 최대 개수가 1보다 큰 트리구조이다.
- B tree의 특징:
  - 각 노드의 key는 정렬되어있고 중복되지 않는다.
  - 모든 leaf node는 같은 레벨에 있다.
  - root node는 자신이 leaf node가 되지 않는 이상 적어도 2개 이상의 자식을 가진다.
  - root node와 leaf node를 제외한 노드들은 최대 M개부터 최소 [M/2]개 까지의 자식을 가질 수 있다.
  - 특정 노드의 key가 k개면 자식 노드의 수는 k+1개가 된다.
  - Binary search tree와 같이 특정 노드의 왼쪽 서브 트리는 특정 노드의 데이터보다 작은 값들로, 오른쪽 서브 트리는 큰 값들로 구성된다.



- Binary tree구조에서 preorder traverse, inorder traverse, postorder traverse, level order traverse를 보면 아래와 같다.
- Preorder traverse: A B D E C F G (부모 먼저)
- Inorder traverse: D B E A F C G (왼자식 부모 오른자식)
- Postorder traverse: D E B F G C A (왼자식 오른자식 부모)
- level order traverse: A B C D E F G

7. B+ tree와 B*의 차이

- B* tree: B tree가 구조를 유지하기 위해서 추가적인 연산을 수행하거나 새로운 노드를 생성해야 한다는 단점을 최소화하기 위해 B tree에 몇 가지 규칙이 추가된 tree 구조
  - 최소 M/2개의 key 값을 가져야 했던 기존 노드의 자식 노드 수 최소 제약 조건이 2M/3개로 늘어남
  - 삽입 시에 노드가 가득 차있으면 분리하지 않고 이웃한 형제 노드로 재배치
- B+ tree: B tree가 탐색을 위해 노드를 찾아서 이동해야한다는 단점이 있는데, 이를 해소하고자 같은 레벨의 모든 키값들이 정렬되어 있고, 같은 레벨의 sibling node들은 Linked list로 이어져 있는 tree 자료구조
  - leaf 노드에 모든 자료가 존재함. 데이터 노드라고도 불림. 그 외의 노드들은 인덱스 노드라고 부름
  - 인덱스 노드의 value에는 다음 노드를 가리킬 수 있는 포인터 주소가 존재
  - 데이터 노드의 value에는 데이터가 존재
  - B tree와 다르게 key 값이 중복될 수 있다.
  - 데이터 검색을 위해 반드시 leaf node까지 내려가야 한다.
  - B tree, B* tree와 비교할 때 가장 빠른 탐색 속도를 가진다.

8. Max-heap 자료구조

- 최대 힙: 최대 트리이면서 완전 이진 트리를 말함. 최대 트리는 각 노드의 키 값이 자식의 키 값보다 크거나 같은 트리이다. -> 항상 부모 노드의 키 값이 자식의 키 보다 같거나 큼
- 최소 힙: 최소 트리이면서 완전 이진 트리를 말함. 최소 트리는 각 노드의 키 값이 자식의 키 값보다 작거나 같은 트리이다. -> 항상 부모 노드의 키 값이 자식의 키 보다 같거나 작음
- 완전 이진 트리 (Complete Binary Tree): 노드를 삽입할 때 왼쪽부터 차례대로 삽입하는 트리이다. 자식 노드가 최대 2개가 가능하지만 가장 마지막에 삽입된 자식 노드가 1개인 경우는 1개도 가능하다.
- 삽입: 리프 노드에서부터 삽입하여 부모 노드와 비교해가며 정렬

• 제거: 루트 노드부터 삭제하고 가장 마지막 리브 노드를 루트 노드로 올림. 이후 자식 노드들과 비교해가며 정렬

9. AVL tree 자료구조

스스로 균형을 잡는 balanced binary search tree.

Binary search tree의 시간 복잡도는 트리의 높이가 h 일 때 O(h)이다. 한쪽으로 치우친 편향 이진 트리가 되면 시간복잡도가 늘어나기 때문에 이를 완화하기위해 사용됨.

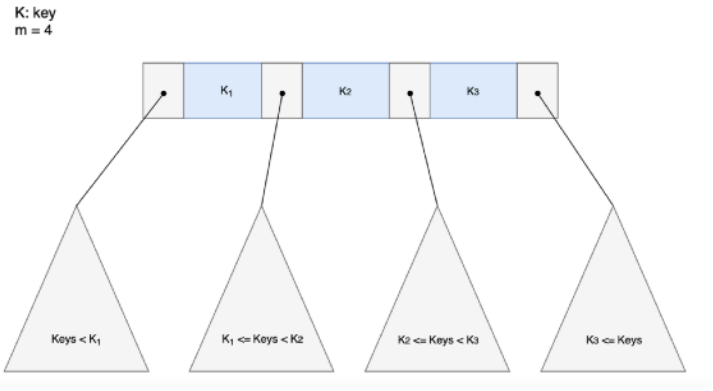
특징:

- 이진탐색트리의 속성을 가짐
- 왼쪽, 오른쪽 서브트리의 높이 차이가 최대 1 (balance factor = 0, 1, -1)
- 높이 차이가 1보다 커지면 rotation을 통해 높이 차이를 줄임
- 시간 복잡도는 O(logN)

10. M-way tree 자료구조

Multiway tree, Multiway search tree 라고도 한다.

한 노드 안에 최대 m-1개의 요소와 m개의 자식을 가질 수 있다. 이진탐색트리는 m=2인 다원 탐색 트리인 셈이다.



• 특징

- $0 \leq$ 서브트리 개수 $\leq m$
- m개의 자식 노드를 가지는 m-1개의 요소를 가짐
- 각 노드의 key는 오름차순으로 정렬
- 이진 탐색 트리의 배치처럼 각 요소의 왼쪽은 모두 해당 요소보다 작은 key값을 가지고, 오른쪽은 모두 해당 요소보다 크거나 같은 key 값을 가진다
- 낮은 트리 높이를 유지할 수 있다
- 스스로 균형을 유지하지 못해서 불균형이 발생할 수 있다 -> B tree 가 이를 보완할 수 있음

11. Red-black tree

- 모든 노드는 빨강 혹은 검정
- 루트 노드와 리프 노드는 검정
- 빨강 노드의 자식은 검정 노드 -> 빨강 노드는 연속 연결이 불가능하다
- 모든 리프 노드부터 루트 노드까지의 black depth 는 같다
- 새로운 노드는 항상 빨강 -> 연속 빨강 노드가 되면 삼촌 노드의 색을 보고 reconstruct하거나 recoloring 진행
  - 삼촌 노드가 검정이면 새로운 노드, 부모, 조부모 노드를 reconstruct
  - 삼촌 노드가 빨강이면 부모, 삼촌 노드 검정, 조부모 노드를 빨강으로 바꾼 담에 조부모 노드에 빨강이 문제가 되면 검정으로 변경

12. Splay tree

- 자주 탐색하는 key를 가진 노드를 루트에 가깝게 위치하도록 한 이진탐색트리이다. 비교연산이 줄어 탐색성능이 올라간다.
- Heuristic 알고리즘: 자주 탐색하는 key를 가진 노드를 트리의 루트에 가깝게 위치시킨다. 트리의 균형이 유지되도록 각 노드의 왼쪽, 오른쪽 서브트리가 비슷한 수의 노드를 갖게 한다.
- Splay 연산: 최근에 접근한 노드 x를 루트 노드에 두어 트리를 재구성하는 연산

13. Stack

- Last-in First-out (LIFO) 원리를 따르는 객체 집합의 추상자료형으로, 먼저 들어간 객체 위에 다음에 들어간 객체가 쌓이다가 꺼낼 때는 마지막에 넣은 객체부터 빠진다.
- 삽입, 삭제는 일반적으로 O(1)이지만 배열이 다 찬경우 배열을 늘려야 할 때 등의 특정한 경우에 O(n)이 걸리기도 한다. 그러나 특정 값 만을 찾아서 삭제하는건 불가
- 활용: 수학표현에서 괄호 여닫기, HTML 경계기호 여닫기

14. Queue

- First-in First-out (FIFO) 원리를 따름.
- 활용: 콜센터에 전화가 온 순서대로 상담사에게 매칭
- 삭제의 경우 앞에서부터 제거하고 다음 요소를 앞으로 한칸씩 이동시켜야하기 때문에 O(n)이 소요됨. 따라서 제거하는 것이 아니라 앞에서 포인팅하는 주소를 None으로 변경하는 방식으로 O(1)만 소요하도록 구조화된다. 그러나 이럴 경우 큐의 크기가 계속 증가한다는 문제가 발생. 따라서 적당한 사이즈를 다룰 때 주로 이용됨

- 결과적으로 삽입, 삭제가 모두 O(1)이라 할 수 있음. 마찬가지로 특정 값 만을 삭제하는건 불가

15. Circularly Queue

- 선형 큐의 단점을 보완하기 위함
- N개로 구성된 큐가 있을 때, 0부터 N-1까지의 인덱스를 지나고 나면 다시 0으로 이어진다. modulo 연산자가 이용됨

16. Linked Lists

- 첫 번째 노드를 헤드로 두고 마지막을 테일로 지정하여 노드를 계속 연결한 자료구조형
- 방향이 하나인 singly linked list, 두 방향인 doubly linked list가 있다.
- 싱글리 링크드 리스트의 경우는 이전 방향으로 접근이 안되고 헤드로만 접근이 가능하다.
- 링크드 리스트 스택과 링크드 리스트 큐가 있는데, 스택과 다르게 큐는 테일이 존재하여 삽입할 때 테일로 삽입하고 삭제할때 헤드를 삭제한다. 반면, 스택은 헤드에 삽입, 헤드를 삭제.

17. 서치, 삽입

- 정렬되지 않은 array: 서치 - O(n)
- 정렬된 array: 삽입 - O(n) -> 삽입하면서 뒤쪽을 재정렬 해야하기 때문
- 힙: 자식노드가 부모노드보다 숫자가 작기만 하면 되므로 완전 정렬 상태가 아니다. 서치 - O(n)
- 이진탐색트리: 서치, 삽입까지 O($\log n$)

18. Priority Queue

- 우선순위 원소가 존재해서 임의의 원소가 들어왔을 때, 그 원소보다 우선순위 원소들을 먼저 제거하는게 허용되는 데이터 구조
- 키-값 으로 키에 우선순위가 값에 원소가 들어가도록 모델링됨
- 정렬되지 않은 더블리 링크드 리스트, 정렬된 더블리 링크드 리스트, 힙 기반으로 PQ를 구현할 수 있다.
  - 정렬되지 않은 더블리 링크드 리스트의 경우 삭제에서 O(n)
  - 정렬된 링크드 리스트는 삭제에서 O(1)이지만 삽입에서 O(n)
  - 힙에서는 삽입 삭제에 O($\log n$)이 걸린다.

19. Map

- 유니크한 key가 연관된 value로 매핑되는 추상표현을 map이라고함
- Hash table은 맵의 구현 중 하나이며 hash function이라는 새로운 개념을 사용해서 hash table을 일반화 할 수 있다.
  - 0부터 N-1까지 N개의 정수로 된 숫자에 값이 분포되어 있다고 할 때, 해시함수를 통해 0부터 N-1까지가 키로 나오도록 분포된다.
  - 종종 하나의 키가 여러번 중복될 수 있는데, 테이블을 bucket array로 개념화할 수 있다.
  - 해시 함수의 목표는 각 키를 [0, N-1]까지의 숫자로 매핑하는 것이다. 그리고 이렇게 만들어진 h(k) 를 새로운 키로써 실제 맵에서 값을 찾는 것.
  - 해시함수 구성요소: Hash code & compression function
  - 일반적인 방법은 hash code를 이용해서 정수로 매핑하고, compression function을 이용하여 해당 정수를 [0, N-1]까지의 정수로 매핑하는 것.
- . Key collision in Hash function
  - 서로다른 두 개의 키에 대하여 해시함수의 아웃풋이 같으면 충돌이 일어난다.
  - Separate Chaining: 각 버켓의 요소가 리스트같은 두 번째 컨테이너를 가지도록 하는것
  - 최악의 경우 리스트 끝까지 읽어야 한다. n개의 아이템을 N개의 공간을 가진 버켓에 연결할 때 각 버켓에서 리스트 크기의 기대값은 n/N이 된다. 따라서 주요 매핑 연산은 O([n/N]).
  - 공간이 작을 경우 충돌을 피하기가 어렵고, 리스트같은 보조 데이터구조가 필요하다는 단점이 있다. 이에 대한 대안은 Open Addressing으로 linear probing, quadratic probing 등이 있다.

20. Sorted map

- 키가 증가하는 순서대로 요소가 저장됨.
- Binary search 알고리즘으로 mid = [(low + high) / 2]으로 두고 target과 비교하며 search 가능. O($\log n$). 그러나 타겟 값이 맵에 저장되어있지 않은 경우 삭제에는 O(n)이 걸림.