분류	질문	답변	중요도	출처
트리	트리에 대해 설명해주세요	트리는 스택이나 큐와같은 선형 구조가 아닌 비선형 자료구조 이다. 트리는 계층적 관계(Hierarchical Relationship)을 표현하는 자료구조이다. 이 트리 라는 자료구조는 표현에 집중한다. 무엇인가를 저장하고 꺼내야한다는 사고에서 벗어나 트리라는 자료구조를 바라봐야한다. 트리를 구성하고있는 용어 Node(노드) : 트리를 구성하고 있는 각각의 요소를 의미한다. Edge(간선) : 트리를 구성하기 위해 노드와 노드를 연결하는 선을 의미한다. Root Node(루트노드) : 트리 구조에서 최상위에 있는 노드를 의미한다. Terminal Node (= leaf Node, 단말노드) : 하위에 다른 노드가 연결되어 있지않은 노드를 의미한다. Internal Node (내부 노드, 비단말 노드) : 단말노드를 제외한 모든 노드로 루트노드를 포함한다.	상	
	BST와 Binary Tree에 대해서 설명하세요. (N사 전화면접)	이진탐색트리 (Binary Search Tree)는 이진 탐색과 연결 리스트를 결합한 자료구조이다. 이진 탐색의 효율적인 탐색 능력을 유지하면서, 빈번한 자료 입력과 삭제가 가능하다는 장점이 있다. 이진 탐색 트리는 왼쪽 트리의 모든 값이 반드시 부모 노드보다 작아야 하고, 반대로 오른쪽 트리의 모든 값이 부모 노드보다 커야 하는 특징을 가지고 있어야 한다. 이진 탐색 트리의 탐색, 삽입, 삭제의 시간복잡도는 O(h)이다. 트리의 높이에 영향을 받는데, 트리가 균형이 맞지 않으면 워스트 케이스가 나올 수 있다. 그래서 이 균형을 맞춘 구조가 AVL Tree이다.	상	https://devowen.com/285 <코딩 인터뷰 완전분석(Cracking the coding interview)> 게리 라크만 맥도웰 저 https://medium.com/@audrl1010/linked-list-와-array-차이점-4ba873c2e5f5 https://blog.naver.com/PostView.nhn?blogId=kks227&logNo=220791188929 https://baeharam.github.io/posts/data-structure/hash-table/ https://ratsgo.github.io/data structure&algorithm/2017/10/22/bst/ https://gmlwjd9405.github.io/2018/08/28/algorithm-mst.html
	최소 스패닝 트리(Minimum Spanning Tree)에 대해서 설명해주세요.	그래프 G의 스패닝 트리 중 edge weight 값이 최소인 스패닝 트리를 말한다. 스패닝 트리란 그래프 G의 모든 vertex가 cycle 없이 연결된 형태를 말한다. n개의 vertex를 가지는 그래프에서 반드시 (n-1)개의 edge만을 사용해야 하며 사이클이 포함되어서는 안 된다. Kruskal과 Prim을 통해서 MST를 구현할 수 있다. Kruskal의 경우 그래프의 간선들을 오름차순으로 정렬하여 가장 낮은 가중치의 간선부터 차례로 MST에 집합체 추가하는 그리디 알고리즘 방식을 사용한다. Prim의 경우는 시작 정점부터 단계적으로 트리를 확장하는 방법이다.	하	
	Binary Tree(이진 트리)에 대해 설명해주세요	루트 노드를 중심으로 두 개의 서브트리(큰 트리에 속하는 작은 트리)로 나뉘어 진다. 또한 나뉘어진 두 서브 트리도 모두 이진 트리어야 한다. 재귀적인 정의라 맞는듯 하면서도 이해가 쉽지 않을 듯 하다. 한가지 덧붙이자면 공집합도 이진트리로 포함시켜야한다. 그래야 재귀적으로 조건을 확인해갔을때, Leaf Node에 다다랐을때, 정의가 만족되기 때문이다. 자연스럽게 노드가 하나뿐인 것도 이진트리 정의에 만족하게된다. 트리에서는 각 층별로 숫자를 매겨서 이를 트리의 Level(레벨)이라고 한다. 레벨의 값은 0 부터 시작하고, 따라서 루트노드의 레벨은 0 이다. 그리고 트리의 최고레벨을 가리켜 해당 트리의 height(높이)라고 한다.	상	
	Perfect Binary Tree(포화 이진트리), Complete Binary Tree(완전 이진 트리), Full Binary Tree(정 이진 트리)	모든 레벨이 꽉 찬 이진 트리를 가리켜 포화 이진트리라고 한다. 위에서 아래로 왼쪽에서 오른쪽으로 순서대로 차곡차곡 채워진 이진트리를 가리켜 완전 이진트리라고 한다. 모든 노드가 0개 혹은 2개으 ㅣ자식노드만 갖는 이진트리를 가리켜 정 이진트리 라고한다. 배열로 구성된 Binary Tree는 노드의 개수가 n개 이고 root가 0이 아닌 1에서 시작할때, i번째 노드에 대해서 parent (i) = i/2, left_child(i)=2i, right_child(i) = 2i+1의 index값을 갖는다.	상	
	BST(Binary Search Tree)	효율적인 탐색을 위해서는 어떻게 찾을까만 고민해서는 안된다. 그보다는 효율적인 탐색을 위한 저장방법이 무엇일까를 고민해야 한다. 이진 탐색 트리는 이진 트리의 일종이다. 단 이진 탐색 트리에는 데이터를 저장하는 규칙이 있다. 그리고 그 규칙은 특정 데이터의 위치를 찾는데 사용할 수 있다. 1. 이진 탐색 트리의 노드에 저장된 키는 유일하다. 2. 부모의 키가 왼쪽 자식 노드의 키보다 크다. 3. 부모의 키가 오른쪽 자식 노드의 키보다 작다. 4. 왼쪽과 오른쪽 서브트리도 이진 탐색 트리이다. 이진 탐색트리의 탐색연산은 O(log n)의 시간복잡도를 갖는다. 사실 정확히 말하면 O(h)라고 표현하는것이 맞다. 트리의 높이를 하나씩 더해갈수록 추가할 수 있는 노드의 수가 두배씩 증가하기 때문이다. 하지만 이러한 이진 탐색트리는 Skewed Tree(편향 트리가)가 될수있다. 저장 순서에 따라 계속 한 쪽으로만 노드가 추가되는 경우가 발생하기 때문이다. 이럴 경우 성능에 영향으 미치게 되며, 탐색의 Worst Case가 되고 시간 복잡도는 O(n)이 된다. 배열보다 많은 메모리를 사용하며 데이터를 저장했지만 탐색에 필요한 시간 복잡도가 같게 되는 비효율적인 상황이 발생한다. 이를 해결하기 위해 Rebalancing 기법이 등장하였다. 균형을 잡기 위한 트리 구조의 재조정을 Rebalancing이라 한다. 이 기법을 구현한 트리에는 여러 종류가 존재하는데 그 중 하나가 Red-Black Tree이다.	중	https://velog.io/@gatsukichi/%EB%A9%B4%EC%A0%91-%EC%A0%95%EB%A6%AC%ED%8E%B8%EC%A7%91%EC%A4%91

	Red-Black Tree의 특징	binary Search Tree 이므로 BST의 특징을 모두 갖는다. 2. Root node 부터 leaf node까지의 모든 경로 중 최소 경로와 최대 경로의 크기 비율은 2보다 크지않다. 이러한 상태를 balanced 상태라고 한다. 3. 노드의 child가 없을 경우 child를 가리키는 포인터는 NIL 값을 저장한다. 이러한 NIL들은 leaf node로 간주한다. RBT는 BST의 삽입, 삭제 연산과정에서 발생할 수 있는 문제점을 해결하기 위해 만들어진 자료구조이다. 이를 어떻게 해결한 것인가? 삽입 우선 BST의 특성을 유지하면서 노드를 삽입을 한다. 그리고 삽입된 노드의 색깔을 RED로 지정한다. Red로 지정한 이유는 Black-Height변경을 최소화 하기 위함이다. 삽입결과 RBT의 특성 위배(violation)시 노드의 색깔을 조정하고, Black-Height가 위배되었다면 rotation을 통해 height를 조정한다. 이러한 과정을 통해 RBT의 동일한 height에 존재하는 Internal node들의 Black-Height가 같아지게 되고 최소 경로와 최대 경로의 크기 비율이 2이만으로 유지된다. 삭제 삭제도 마찬가지로 BST의 특성을 유지하면서 해당 노드를 삭제한다. 삭제될 노드의 child의 개수에 따라 rotation방법이 달라지게된다. 그리고 만약 지워진 노드의 색깔이 Black이라면 Black-Height가 1 감소한 경로에 Black node가 1개 추가되도록 rotation하고 노드의 색깔을 조정한다. 지워진 노드의 색깔이 red라면 Violation이 발생하지 않으므로 RBT가 그대로 유지된다. Java Collection에서 ArrayList도 내부적으로 RBT로 이루어져 있고, HashMap에서의 separate Chaining에서도 사용된다. 그만큼 효율이 좋고 중요한 자료구조이다.	하	
우선순위 큐	PriorityQueue의 동작 원리가 어떻게 되나요? (N사 전화면접)	우선순위 큐는 힙이라는 자료구조를 가지고 구현한다. top이 최대면 최대힙, top이 최소면 최소힙으로 표현한다. 힙으로 구현된 이진 트리는 모든 정점이 자신의 자식 요소보다 우선순위가 높다는 성질을 가지고 있다. 이 성질을 통해 삽입과 삭제 연산을 모두 O(logN)으로 수행할 수 있다.	상	
	우선순위 큐	Queue는 먼저 넣은 데이터가 먼저 나오는 FIFO(First In First Out) 특징을 가진 선형 자료구조입니다. 우선순위 큐(Priority Queue) 또한 큐와 비슷하지만 다른 점은 들어간 순서에 상관없이 우선순위가 높은 데이터가 먼저 나오는 자료구조입니다. **우선순위 큐 구현 방법** 배열 연결 리스트 힙(Heap) 구현방법 삽입 삭제 배열 O(1) O(N) 연결 리스트 O(1) O(N) 힙 O(logN) O(logN) 배열이나 연결 리스트로 우선순위 큐를 구현 시 쉽게 구현이 가능합니다. 하지만 배열과 연결 리스트는 데이터 삽입, 삭제 과정에서 데이터들을 한 칸 밀거나 당겨야 하는 연산이 필요하고 또한 삽입 위치를 찾기 위해 모든 데이터와의 비교가 필요하다는 단점이 있습니다. 그렇기 때문에 우선순위 큐는 일반적으로 힙(Heap)을 이용해서 구현합니다.	상	
힙	Binary Heap	자료 구조의 일종으로 Tree의 형식을 하고있으며, Tree중에서도 배열에 기반한. Complete Binary Tree이다. 배열에 트리의 값들을 넣어줄 때, 0번째는 건너뛰고 1번 index부터 루트노드가 시작된다. 이는 노드의 고유번호 값과 index를 일치시켜 혼동을 줄이기 위함이다. 힙(Heap)에는 최대힙(max heap), 최소힙(min heap) 두종류가 있다. Max Heap이란, 각 노드의 값이 해당 children의 값보다 크거나 가은 complete binary tree를 말한다. (Min heap은 그 반대이다.) Max heap에서는 Root node에 있는 값이 젱일 크므로, 최대값을 찾는ㄷ ㅟ소요되는 연산의 시간 복잡도는 O(1)이다. 그리고 complete binary tree이기 때문에 배열을 사용하여 효율적으로 관리할수 있다. (즉, random access 가 가능하다. Min heap에서는 최소값을 찾는데 소요되는 연산의 time complexity 가 O(1)이다) 하지만 heap의 구조를 계속 유지하기 위해서는 제거된 루트노드를 대체할 다른 노드가 필요하다. 여기서 heap은 맨 마지막 노드를 루트노드로 대체시킨 후, 다시 heapify과정을 거쳐 heap 구조를 유지한다. 이런 경우에는 결국 O(log n)의 시간복잡도로 최대값 또는 최소값에 접근할 수 있게된다.	상	