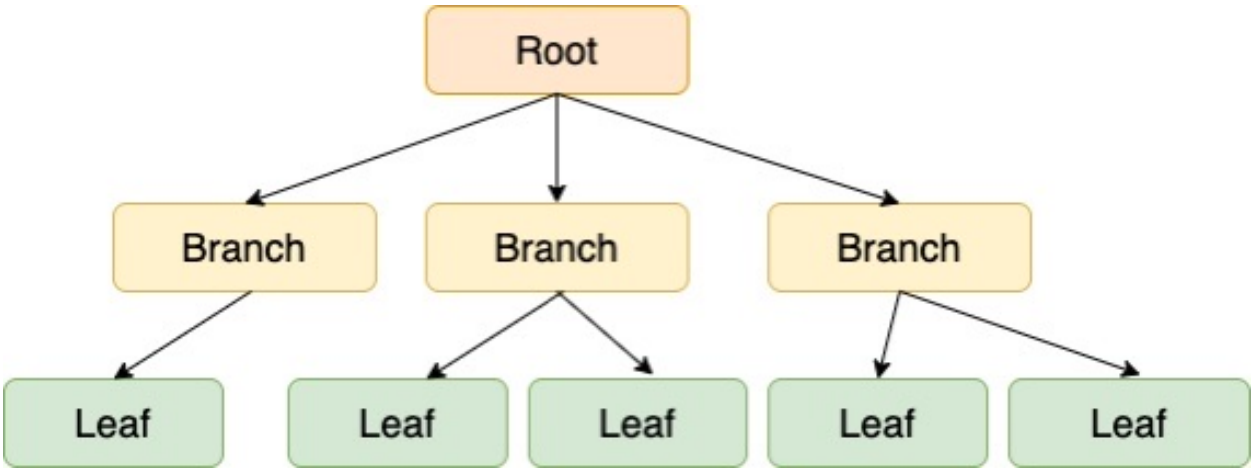
먼저 **B-tree**를 살펴보자.

트리 구조의 우위성

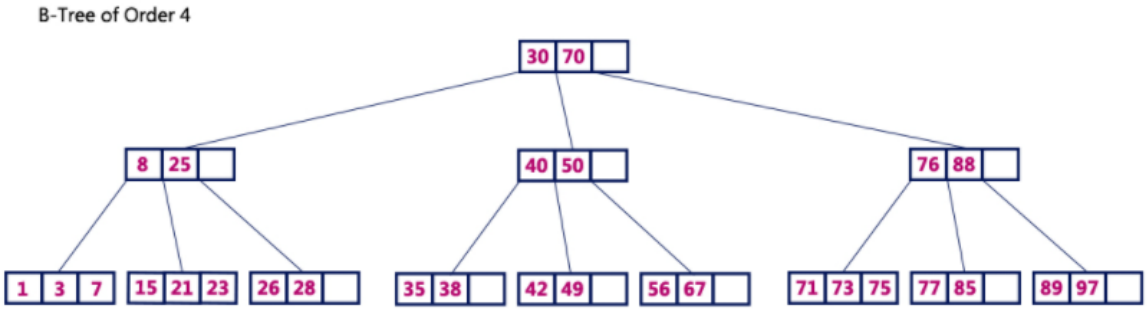
트리 구조는 꼭 데이터베이스에 한정하지 않더라도 시스템 세계에서는 데이터를 유지하기 위해 자주 사용하는 구조이다. '탐색' 시, 단시간 내에 실행할 수 있기 때문이다.

B-tree의 핵심은 데이터가 정렬된 상태로 유지되어 있다는 것이다.



그림에 표시된 사각형으로 표시된 한 개 한 개의 데이터를 '노드(Node)'라고 한다.

가장 상단의 노드를 '루트 노드(Root Node)', 중간 노드들을 '브랜치 노드(Branch Node)', 가장 아래 노드들을 '리프 노드(Leaf Node)'라고 한다.



출처: http://www.btechsmartclass.com/data_structures/b-trees.html

B-tree는 (Binary search tree와 유사하지만,) 한 노드 당 자식 노드가 2개 이상 가능하다.

key 값을 이용해 찾고자 하는 데이터를 트리 구조를 이용해 찾는 것이다.

왜 B-tree는 빠른가

B-tree의 장점 한 가지는 '어떤 값에 대해서도 같은 시간에 결과를 얻을 수 있다'인데, 이를 '균일성'이라고 한다.

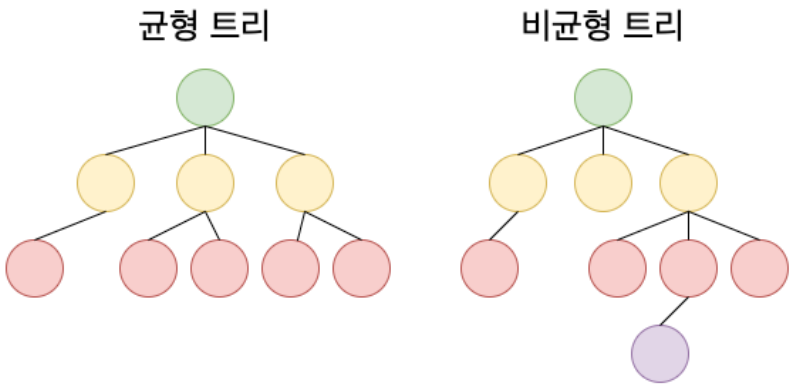
위의 예시에서 리프노드에 있는 '15'나 '28'을 찾는 시간은 동일할 것이다.(트리 높이가 다른 경우, 약간의 차이는 있겠지만 O(logN)이라는 시간 복잡도를 구할 수 있다.)

만약 선형탐색일 경우 어떨까?

리프노드에 있는 값들만 따져보면,
[1, 3, 7, 15, 21 .... 85, 89, 97]

'15', '28'을 찾기 위해서는 배열을 하나씩 체크하는 수 밖에 없고 시간은 더욱 소요된다. (시간복잡도 : O(n))

B-tree는 균형트리



'균형 트리'란 루트로부터 리프까지의 거리가 일정한 트리 구조를 뜻하는 것으로, 트리 중에서 특히 성능이 안정화 되어있다.

그러나, B-tree 처음 생성 당시는 균형 트리이지만 테이블 갱신(INSERT/UPDATE/DELETE)의 반복을 통해 서서히 균형이 깨지고, 성능도 악화된다.

어느 정도 자동으로 균형을 회복하는 기능이 있지만, 갱신 빈도가 높은 테이블에 작성되는 인덱스 같은 경우 인덱스 재구성을 해서 트리의 균형을 되찾는 작업이 필요하다.