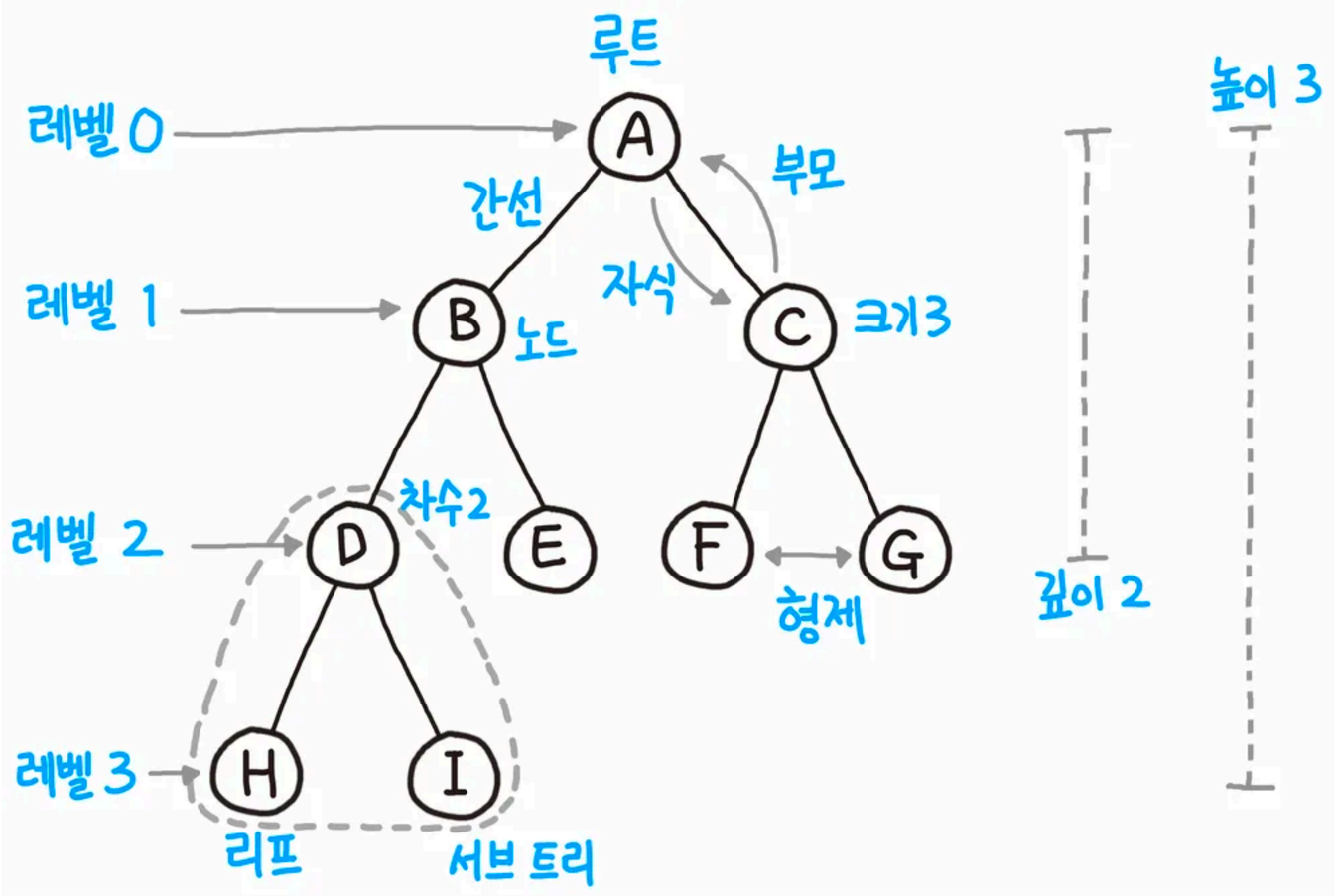
Binary Tree & B-tree

Data structure with C study

Seongyun Jang / 2023.05.09

이진 트리

자식이 두개인 트리

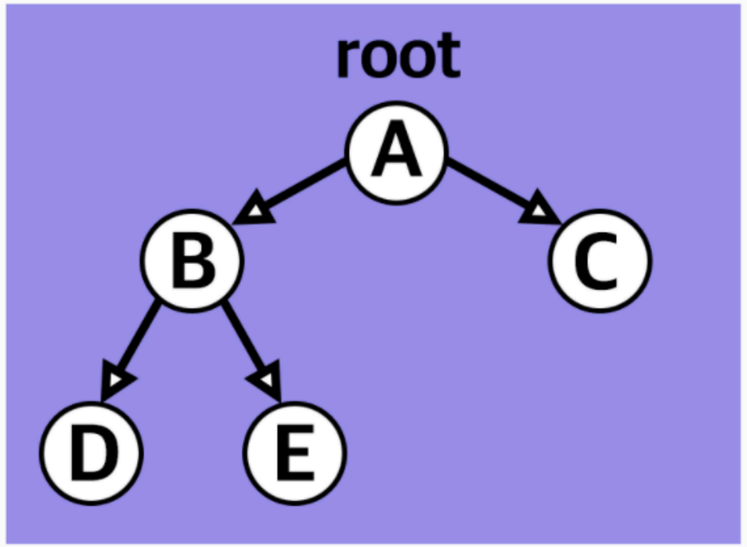


이진 트리 종류

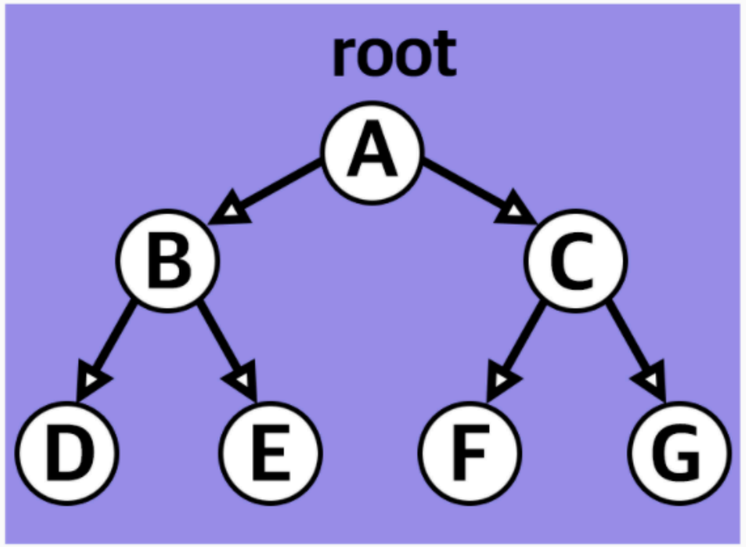
- 정 이진 트리: 모든 트리의 자식은 0개이거나 2개다
- 포화 이진 트리: 모든 리프 노드의 높이가 같고 리프 노드가 아닌 노드는 모두 2개의 자식을 갖는다.
- 완전 이진 트리: 모든 리프노드의 높이가 최대 1 차이가 나고, 모든 노드의 오른쪽 자식이 있으면 왼쪽 자식이 있는 이진트리이다.

이진 트리 종류

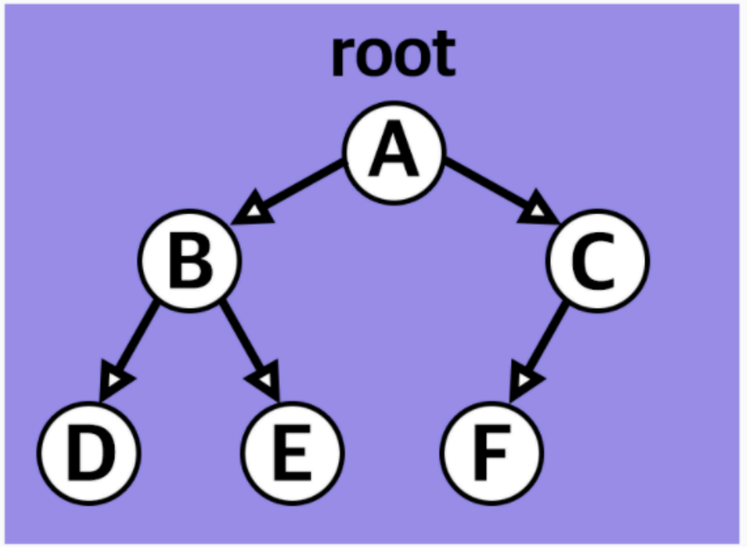
정 이진 트리



포화 이진 트리



완전 이진 트리



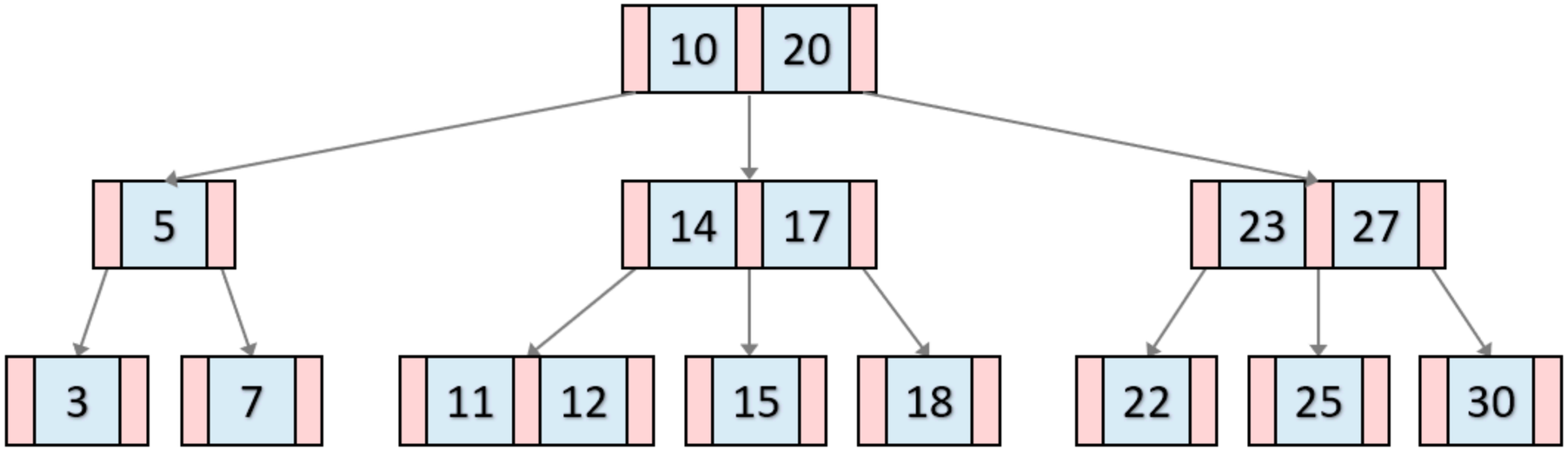
이진 트리 종류

- 이진 탐색 트리 (BST)
  - AVL-tree
  - Red-black tree
- 힙 (Heap)

이진 트리 순회

- 중위 순회: 왼쪽 -> 자신 -> 오른쪽
- 전위 순회: 자신 -> 왼쪽 -> 오른쪽
- 후위 순회: 왼쪽 -> 오른쪽 -> 자신
- 레벨 순서 순회

B-tree



B tree 특징

B트리는 이진트리와 다르게 하나의 노드에 많은 수의 정보를 가지고 있을 수 있습니다.
최대 M개의 자식을 가질 수 있는 B트리를 M차 B트리라고 한다.
다음과 같은 특징을 갖습니다.

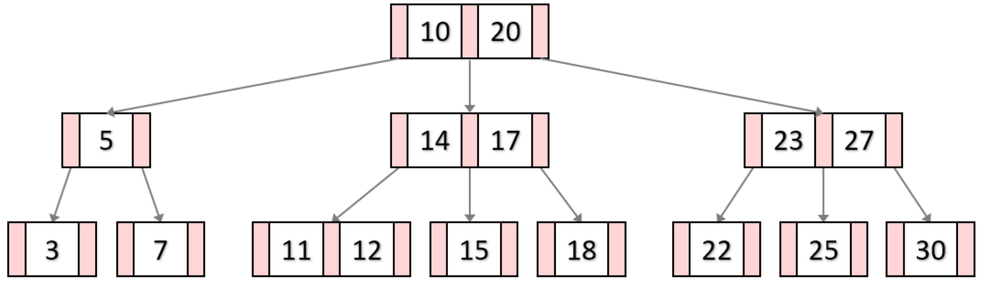
- 노드는 최대 M개 부터 $M/2$개 까지의 자식을 가질 수 있습니다.
- 노드에는 최대 $M-1$개 부터 $[M/2]-1$개의 키가 포함될 수 있습니다.
- 노드의 키가 x개라면 자식의 수는 $x+1$개 입니다.
- 최소차수는 자식수의 하한값을 의미하며, 최소차수가 t라면 $M=2t-1$을 만족합니다.
  (최소차수 t가 2라면 3차 B트리이며, key의 하한은 1개입니다.)

B-tree 용어

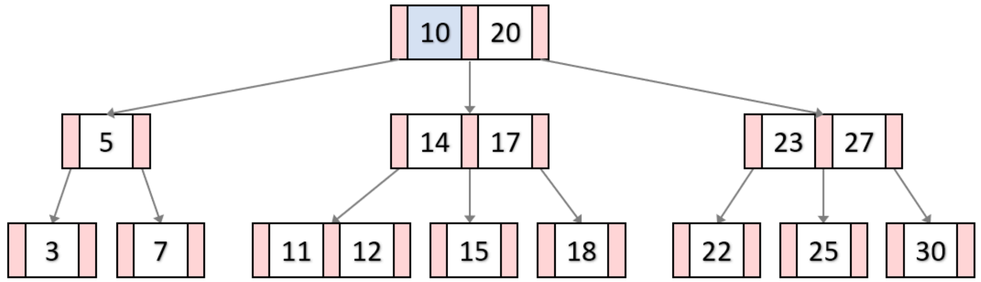
- inorder predecessor : 노드의 **왼쪽 자손**에서 가장 **큰 key**
- inorder successor : 노드의 **오른쪽 자손**에서 가장 **작은 key**
- 부모key: 부모노드의 key들 중 왼쪽 자식으로 본인 노드를 가지고 있는 key값입니다. 단, 마지막 자식노드의 경우에는 부모의 마지막 key입니다.

B-tree 검색

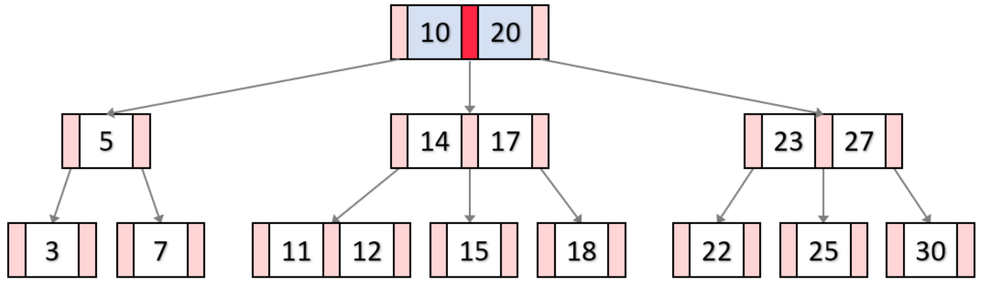
1 18 검색 시작, 루트노드의 key를 순회하면서 검색 시작



2 18은 10보다 크기 때문에 다음 key를 검사

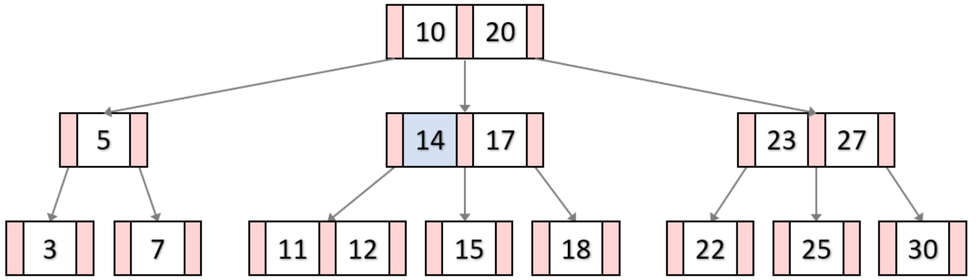


3 18은 10보다 크고, 20보다 작기 때문에 10과 20 사이의 자식 노드로 이동

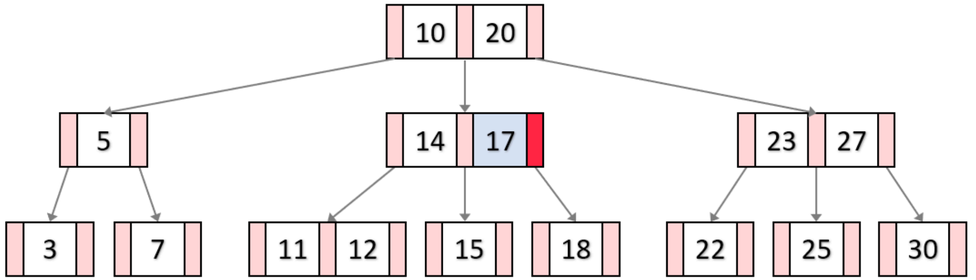


B-tree 검색

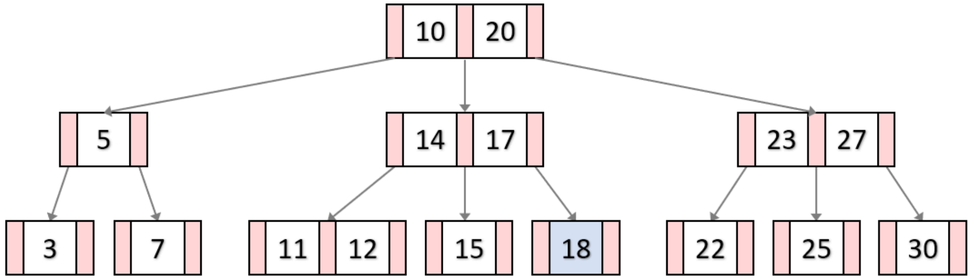
4 자식노드에서 다시 검색 시작. 18은 14보다 크기 때문에 다음 key를 검사



5 18은 노드의 가장 마지막 key인 17보다 크기 때문에 노드의 가장 마지막 자식노드로 이동

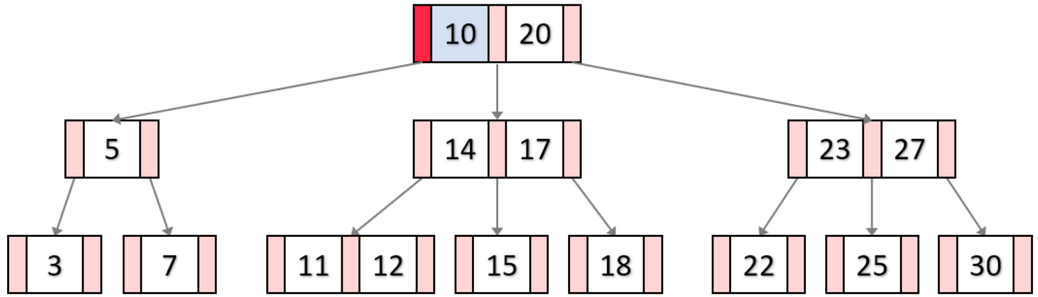


6 18 검색 완료

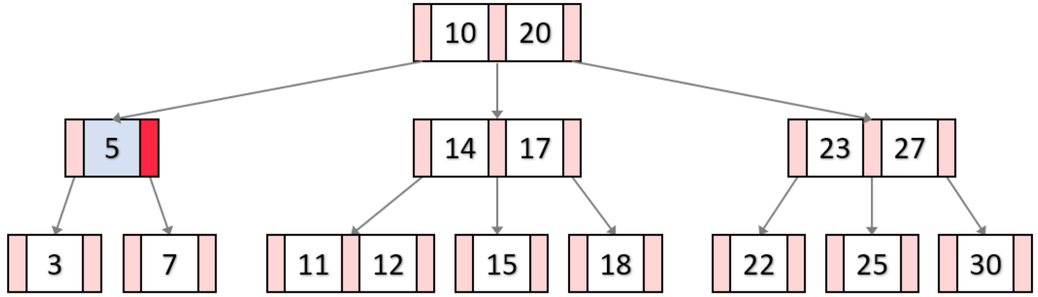


B-tree 삽입 (분할 X)

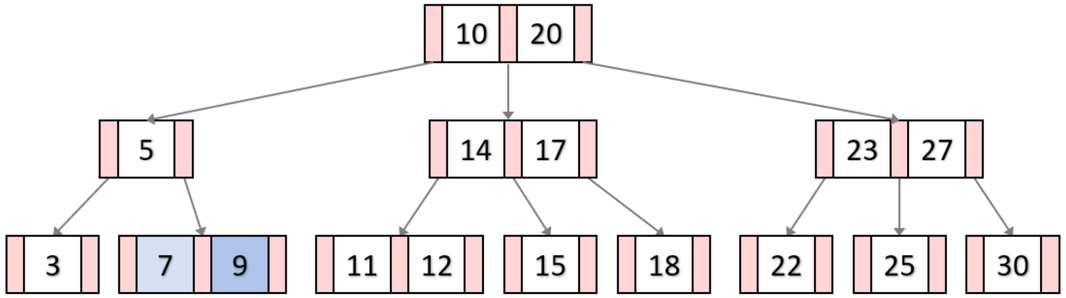
1 9 삽입 시작, 루트노드에서 key 10이 9보다 크기 때문에 가장 왼쪽 자식노드로 이동



2 9는 5보다 크기 때문에 가장 오른쪽 자식노드로 이동

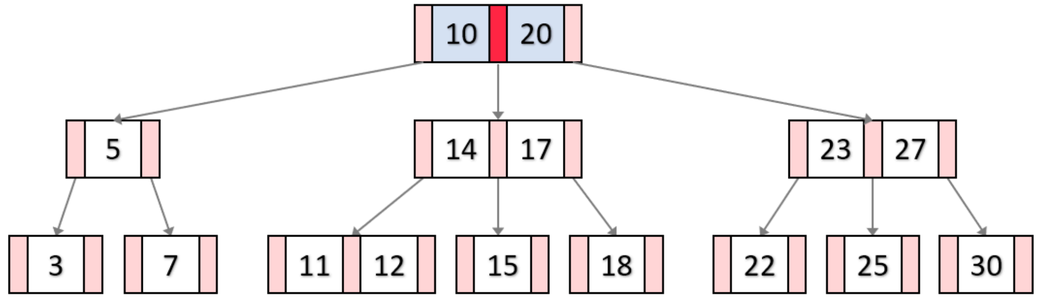


3 리프노드에 도달, 9는 7보다 크기 때문에 7 오른쪽에 9 삽입 완료

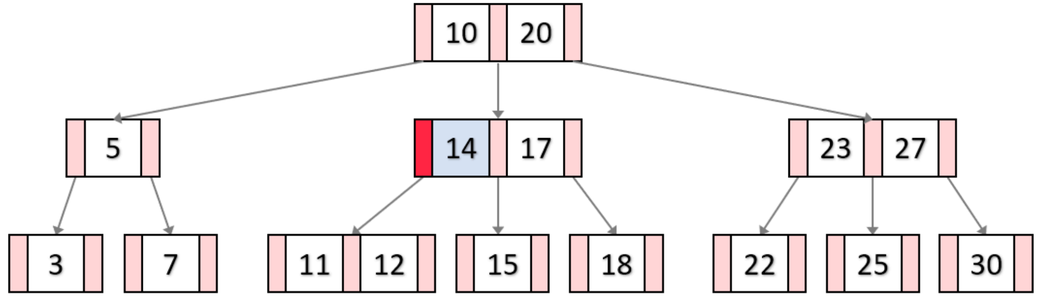


B-tree 삽입 (분할 O)

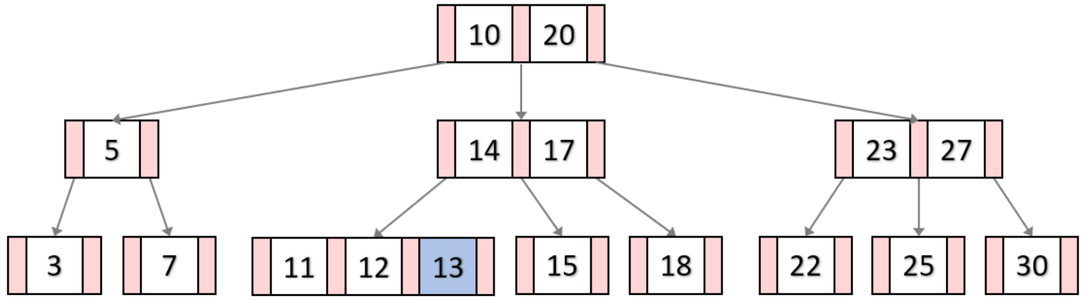
1 13 삽입 시작, 루트노드의 key 중 10보다 크고 20보다 작기 때문에 중간 자식노드로 이동



2 13은 14보다 작기 때문에, 가장 왼쪽 자식노드로 이동



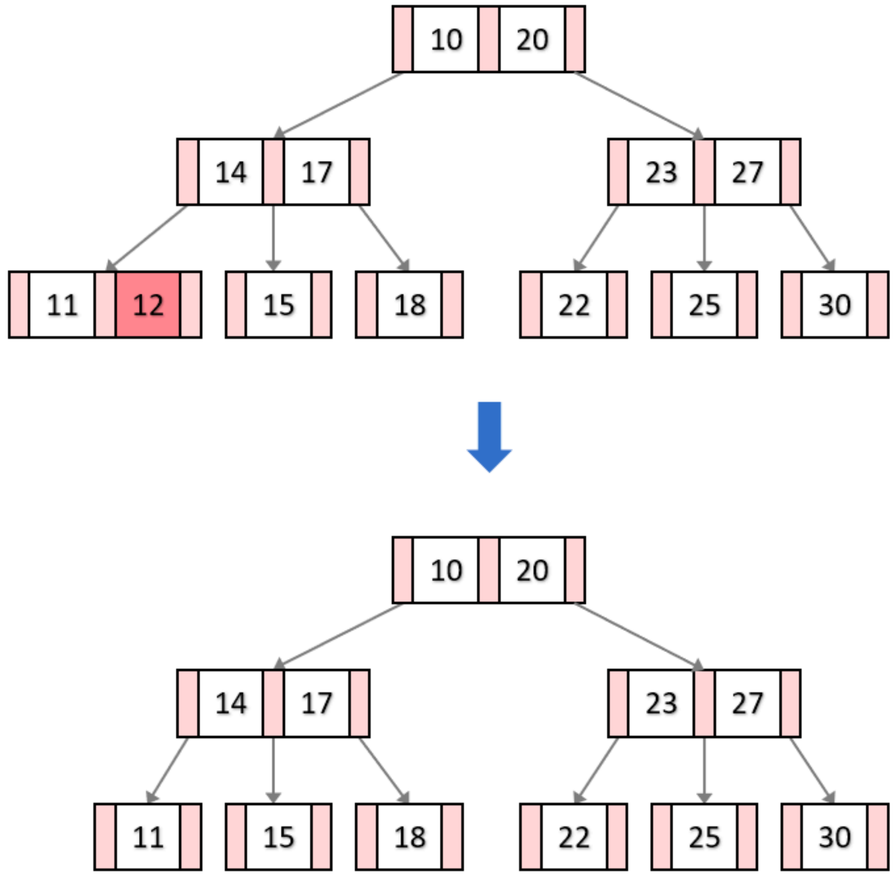
3 13은 12보다 크기 때문에 가장 마지막에 삽입. 해당 노드가 최대로 가질 수 있는 key의 개수를 초과했기 때문에 분할을 수행



B-tree 삭제

- 리프에 있는 경우

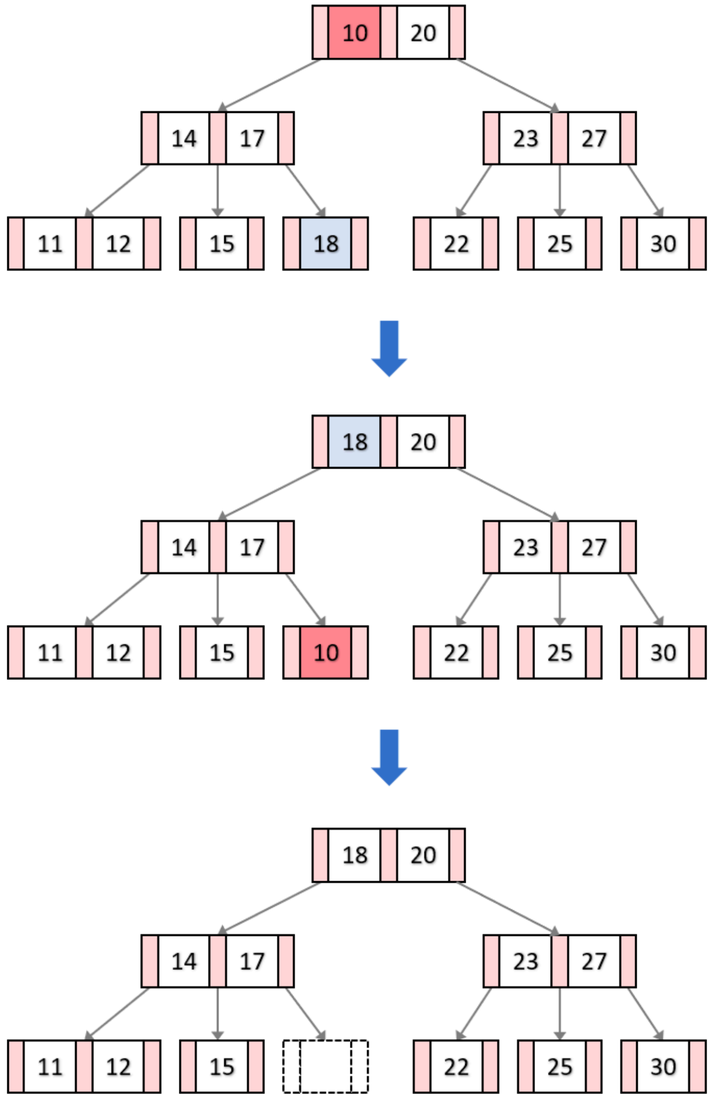
Case 1.1) 12 삭제, 해당 노드의 key 개수는 최소 key의 개수보다 크므로 바로 삭제



B-tree 삭제

- 내부 노드에 있는 경우
- (노드나 자식에 키가 최소 키수보다 많을 경우)

Case 2) 10 삭제, inorder predecessor인 18을 찾아 서로의 key를 교환,

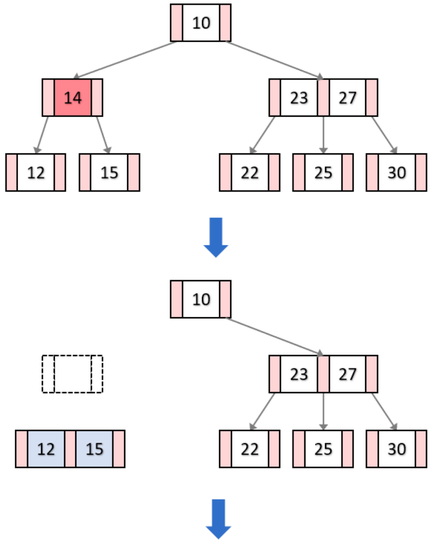


리프노드가 삭제되었을 때의 조건으로 변환

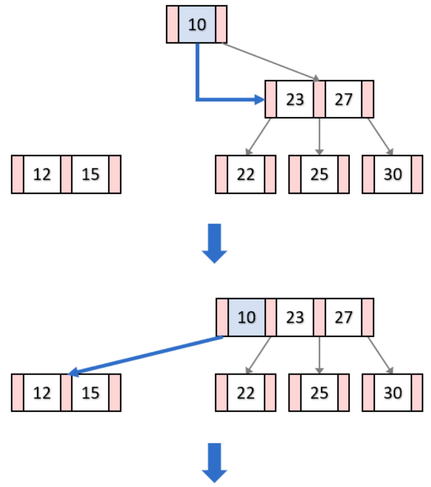
B-tree 삭제

- 내부 노드에 있는 경우
- (노드나 자식에 키가 최소 키수일 경우)

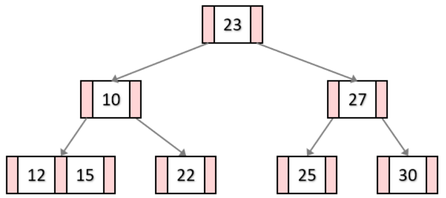
Case 3-1) 14 삭제, 양쪽 자식노드를 병합하고, 14는 삭제



Case 3-2) 부모 key인10을 형제 노드에 붙이고, 이전에 병합한 자식노드를 왼쪽 자식으로 가져옴



Case 3-3-1) 최대 key수를 넘어간 노드가 생겼기 때문에 노드분할 수행



B-tree 응용

- B+tree
  - B-tree의 확장형.
  - 루트 노드와 중간의 노드는 키를 이용하여 위치를 찾아가는 인덱스 역할만을 하며데이터 자체는 모두 리프 노드에 저장한다.
  - 데이터를 정렬하여 리프 노드에 저장했고, 그 위에 B-tree의 규칙으로 키를 저장해 트리를 구성했다고 생각하면 편하다.
  - 이때 리프 노드는 이중 연결 리스트로 구성하여 데이터를 순차적으로 검색하기 용이하도록 한다.