트리 이론

트리 구조

- Gragh
  - Node(정점, 교점, 노드, vertex)와 (호, edge, arc)으로 이루어진 자료구조

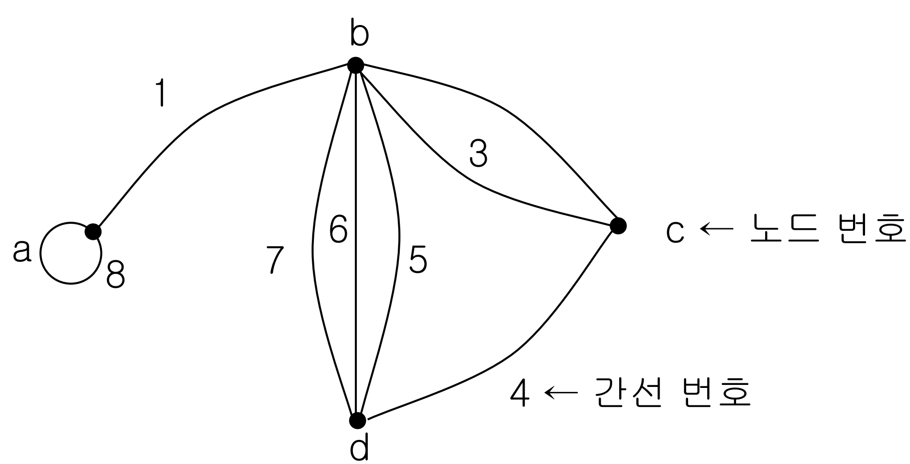


그림 1 그래프의 예

- 단순(Simple)그래프
  - Loop가 없고 두 노드를 연결하는 간선이 하나뿐일 때

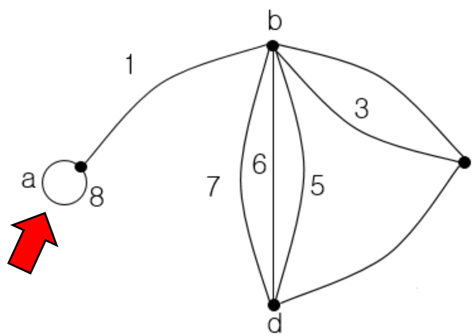


그림1

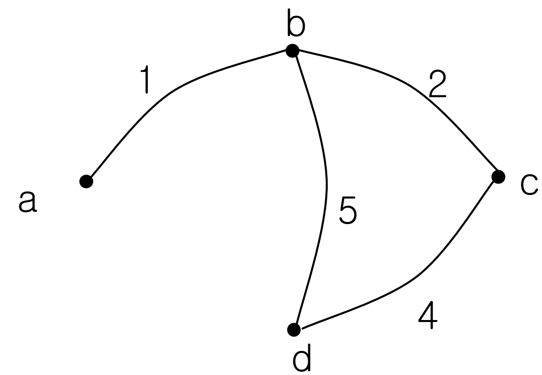


그림 2 그림1을 단순그래프로 변환

- 비순환(Acyclic)그래프
  - 순환(cycle)이 없는 그래프

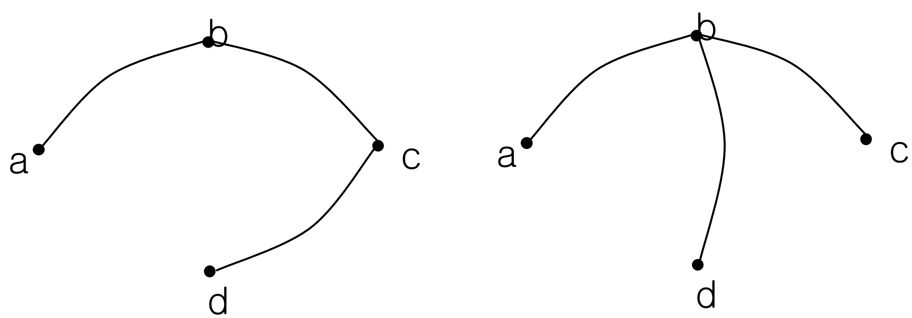


그림 3 Acyclic 그래프의 예

- Connected 그래프
  - 모든 두개의 노드 간에 경로가 있는 그래프

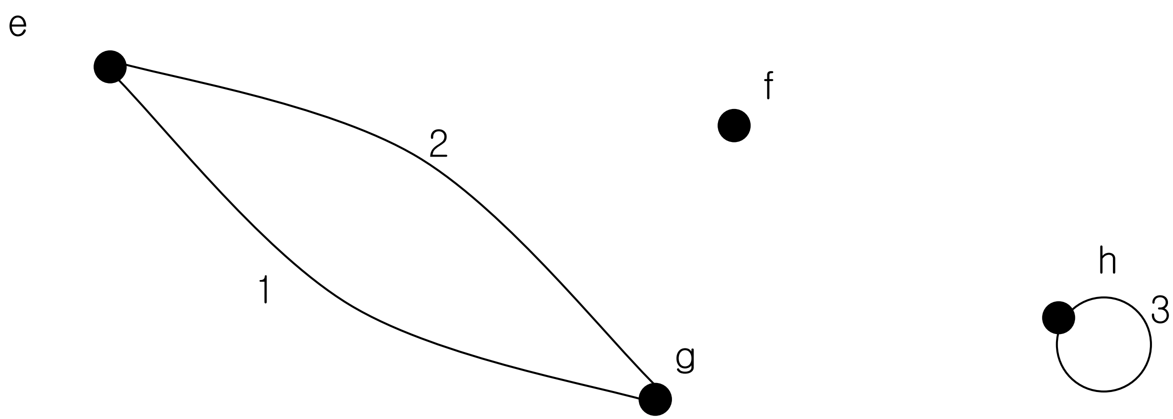


그림 4 연결되지 않은 그래프

- 방향 그래프
  - 간선에 방향이 있는 그래프

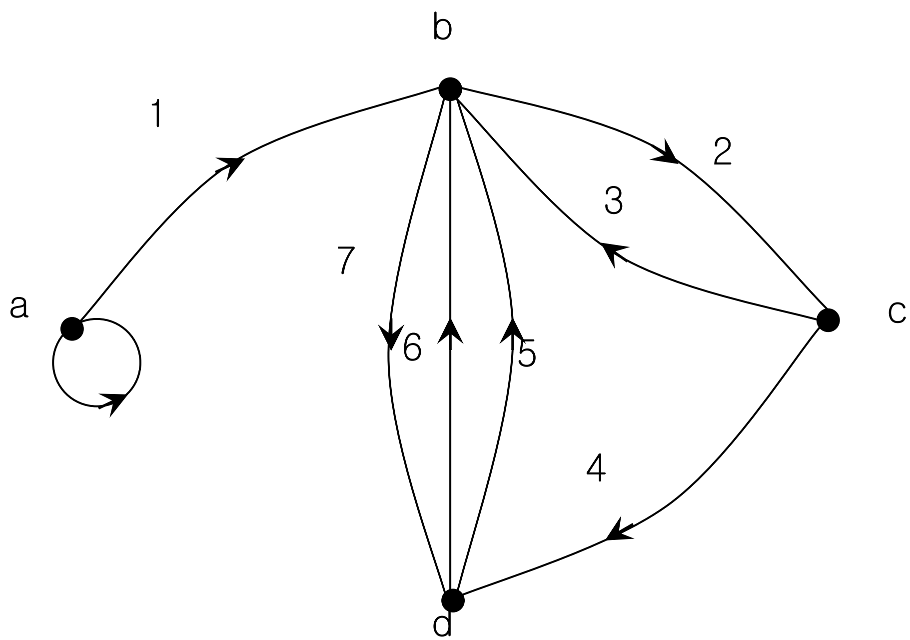


그림 5 방향그래프의 예

- 트리(Tree)
  - Simple & Acyclic, Connected Graph
  - 트리의 간선은 Branch(가지)라고 함

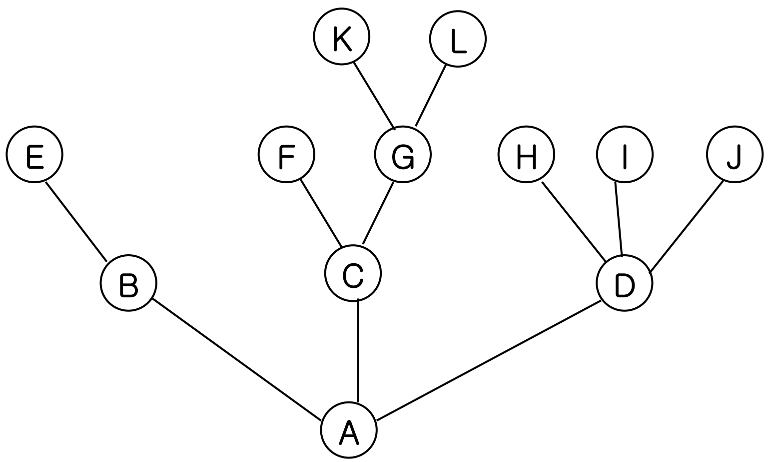


그림 6 트리의 예

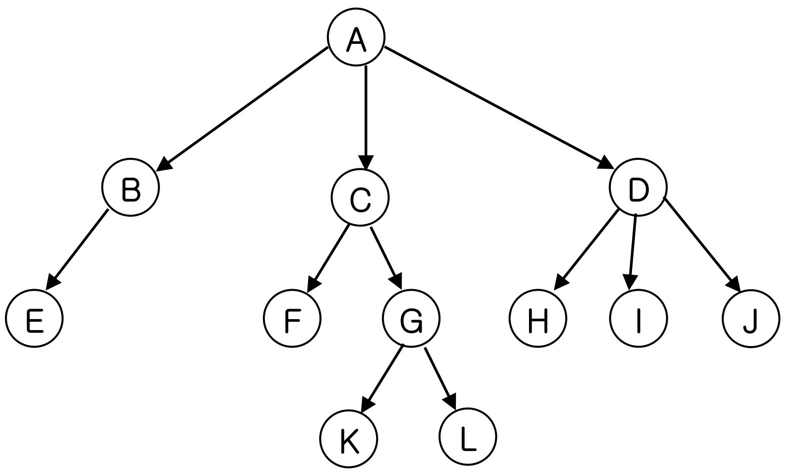


그림 7 루트노드가 위에 오도록 표현한 트리

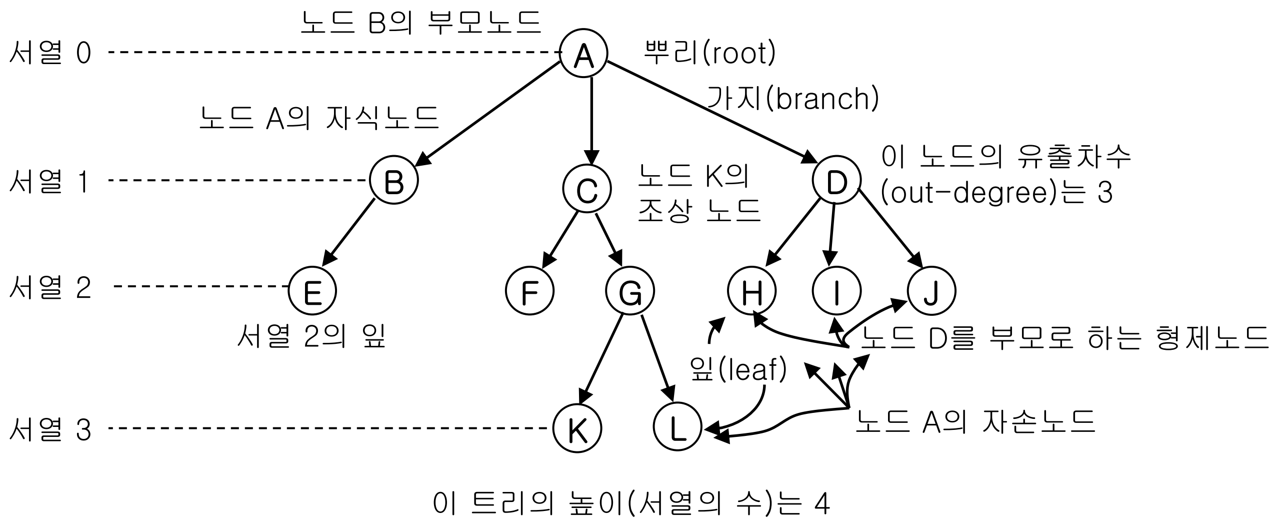


그림 9 트리의 용어설명

- 종속트리(Subtree)의 개념

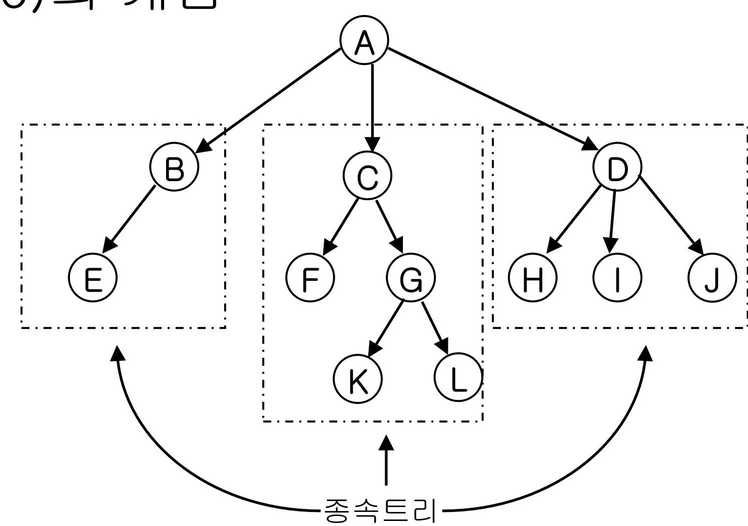
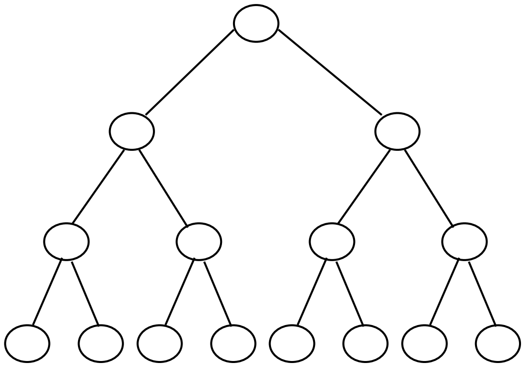
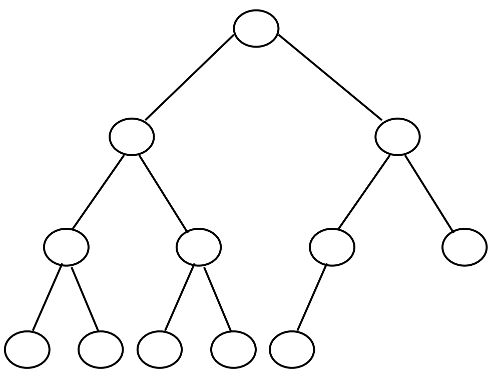


그림 8 종속트리



- 이진트리(Binary Tree)
  - 한 노드에 최대 두개의 종속트리가 있는 노드만으로 구성된 트리
- 완전이진트리(Complete Binary Tree, 혹은 포화이진트리(Full Binary Tree)라고도 함)
  - 이진 트리가 자기 높이에서 가질 수 있는 최대 개수의 노드를 갖는 트리
  - 높이가 H인 완전이진트리의 노드 개수는?

$$\sum_{i=1}^{H} 2^i = 2^H - 1$$



- 유사완전이진트리(Almost Complete Binary Tree)
  - 높이가 H인 이진트리가 서열0부터 H-2까지는 포화되어 있고 마지막 서열 H-1에서 왼쪽부터 오른쪽으로 채워져 있는 경우
  - 높이가 H인 유사완전이진트리에서 노드의 최대 개수가 2^H-1
  - 이진트리의 노드개수 $n \le 2^H-1$이 성립
  - 이를 H에 대해서 풀면 노드가 n개인 이진트리의 최소 높이

    $H = \{ \log_2(n+1) \}$

- 이진트리의 저장 방법
  - 각 노드는 data field와 link field로 구성

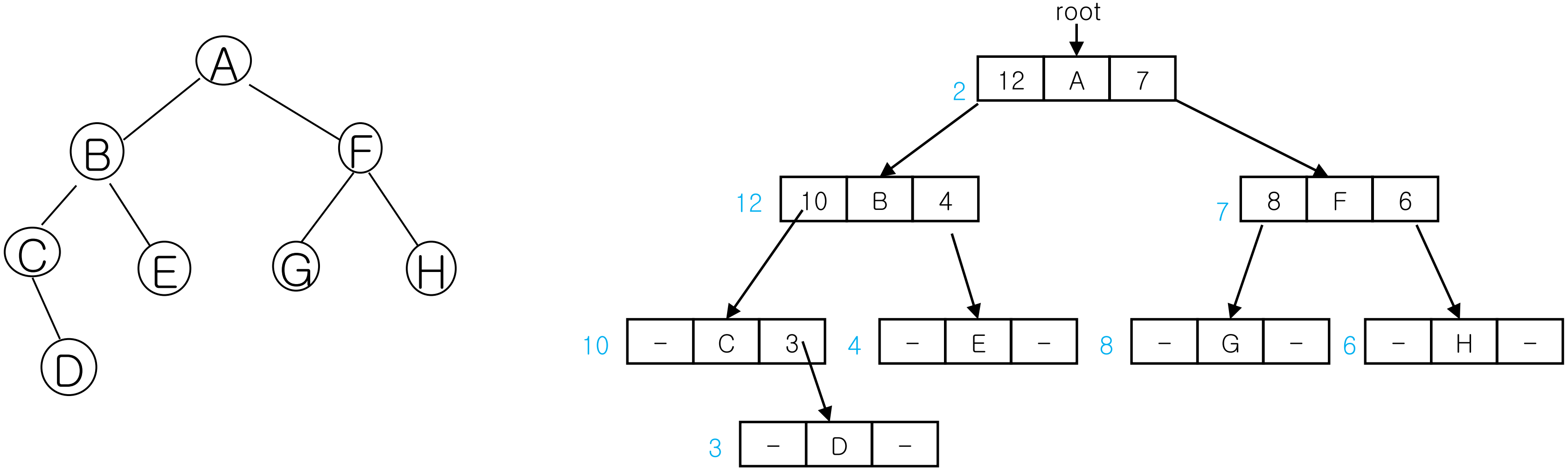
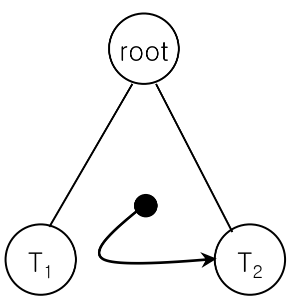
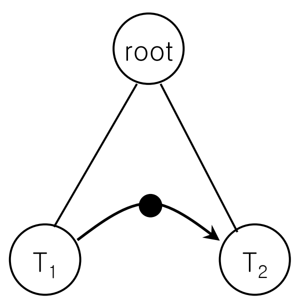


그림 11 이진트리의 저장

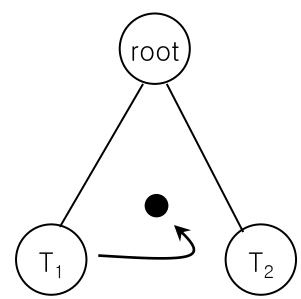
- 이진트리의 순회(Traversal) 혹은 운행
  - 트리의 전체 노드를 처리할 때 각 노드를 반드시 한번씩만 방문하면서 트리를 통과해 가는 과정
  - 순회의 예
    - 전체자료를 인쇄하기 위하여 전체 노드를 인쇄한다.
    - 특정 노드를 탐색하기 위하여 노드를 검색한다.
  - 순회의 종류
    - 전위 순회
      - 루트를 방문
      - 왼쪽 종속 트리를 전위순회
      - 오른쪽 종속트리를 전위순회
    - 중위 순회
      - 왼쪽 종속트리를 중위순회
      - 루트를 방문
      - 오른쪽 종속트리를 중위순회
    - 후위 순회
      - 왼쪽 종속트리를 후위순회
      - 오른쪽 종속트리를 후위순회
      - 루트를 방문



(a)전위순회

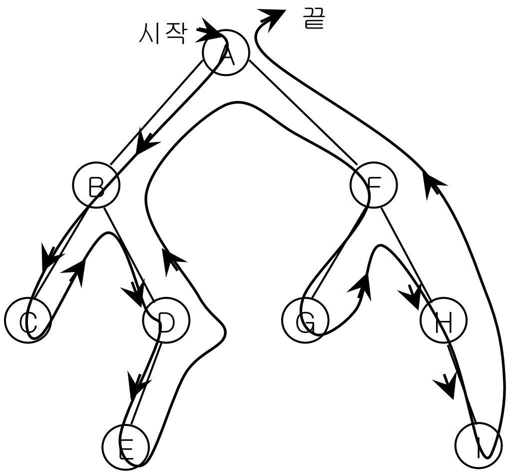


(a)전위순회



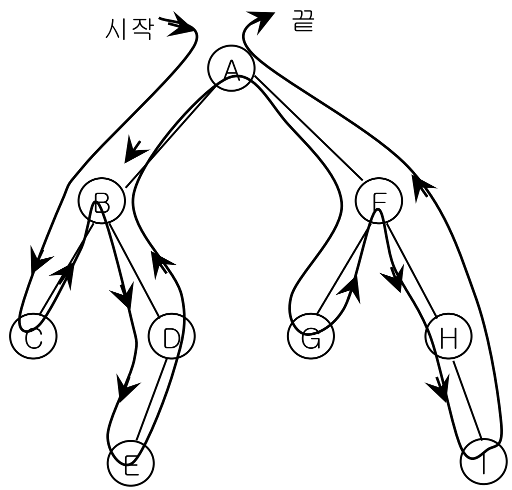
(a)전위순회

그림 13 전위순회의 예



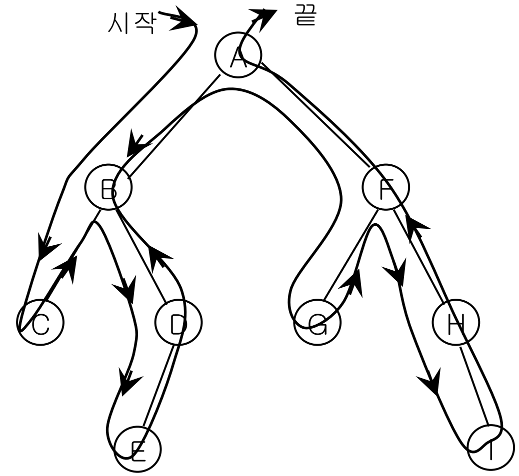
방문된 노드순서 : ABCDEFGHI

그림 14 중위순회의 예



방문된 노드순서 : CBEDAGFHI

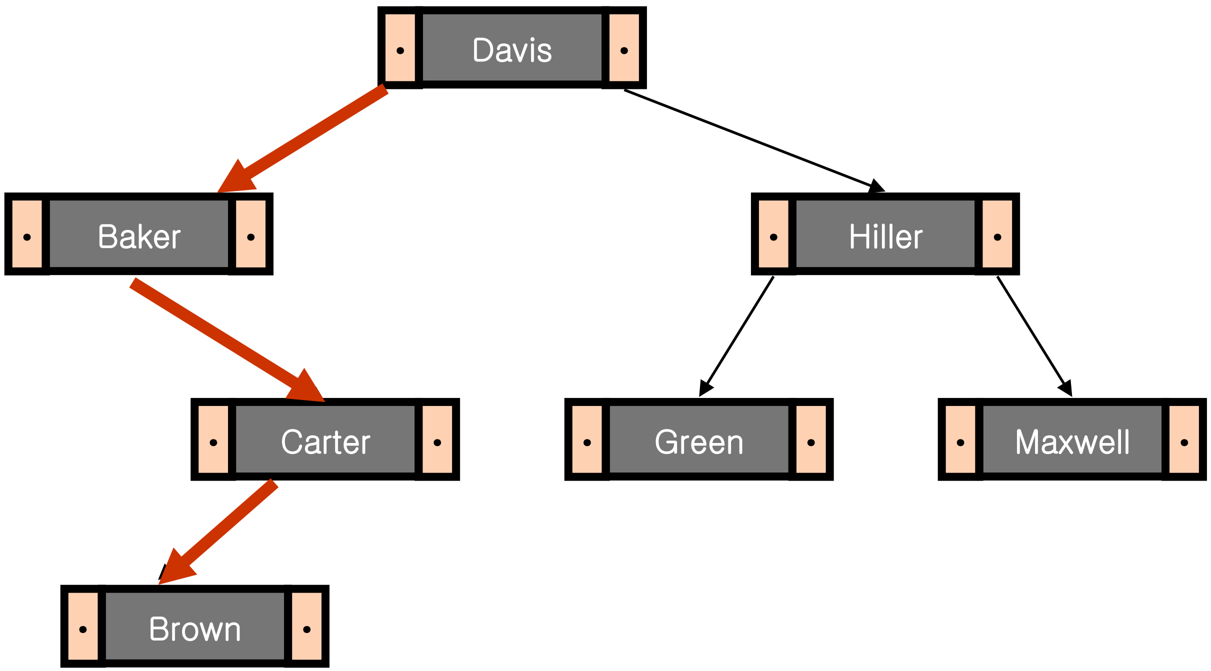
그림 15 후위순회의 예



방문된 노드순서 : CEDBGIHFA

이진탐색트리 (Binary Search Tree)

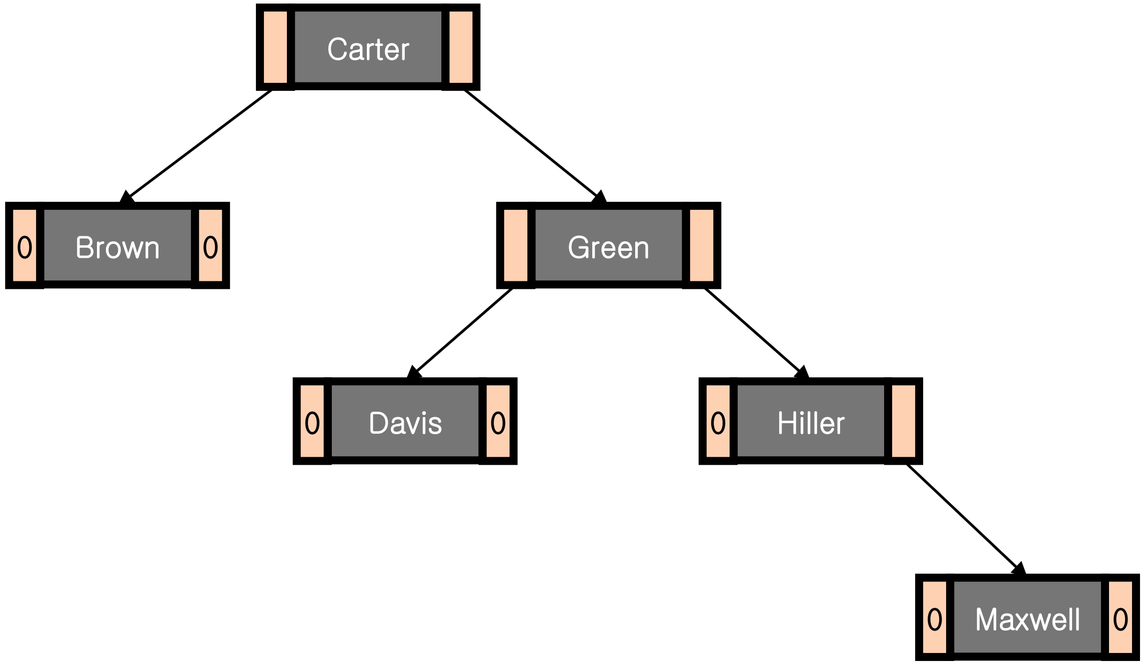
- 자료를 특별한 순서로 구성한 이진트리
  - 각 노드의 왼쪽 종속트리에는 그 노드의 자료 값보다 작은 자료
  - 각 노드의 오른쪽 종속트리에는 그 노드의 자료 값보다 큰 자료



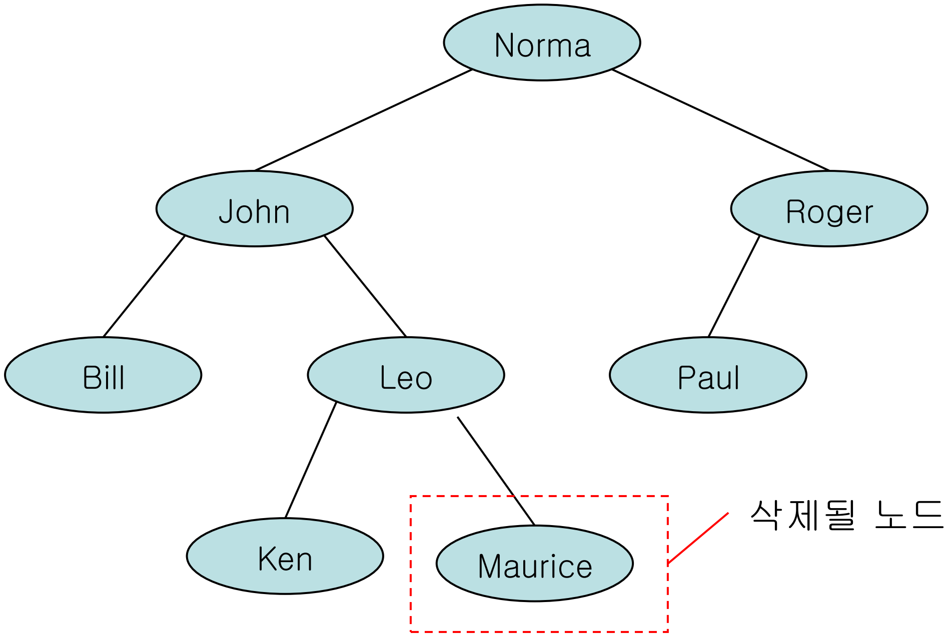
이진탐색 tree의 예 및 Brown을 탐색하는 경로

<삽입과정>

Carter, Brown, Green, Davis, Hiller, Maxwell, Baker

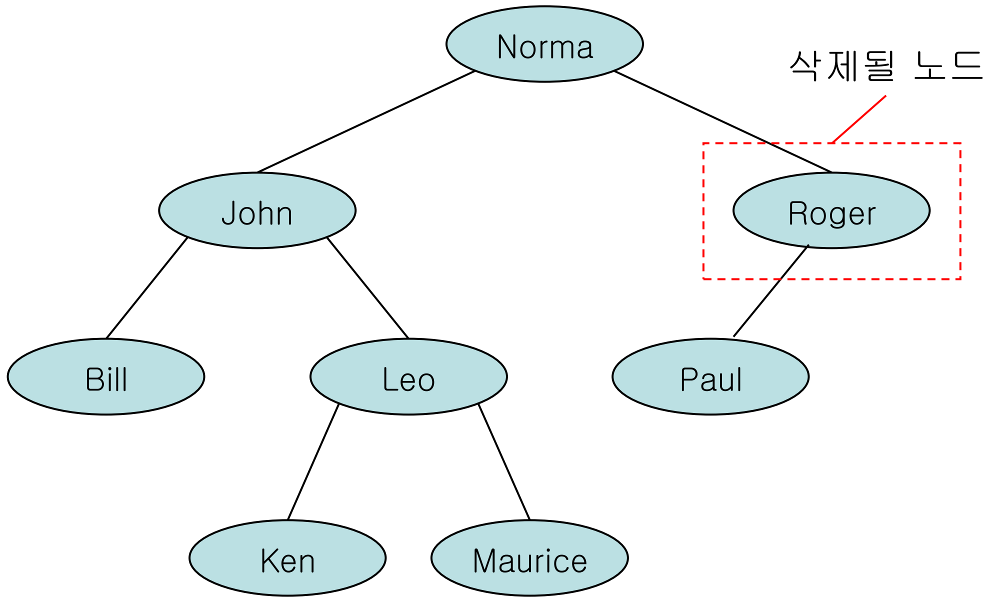


<삭제과정, 잎새노드의 경우>



- 이진탐색트리에서 노드의 삭제(자식노드가 없는 경우)
  - 즉, Leaf Node
  - 바로 삭제
    - 부모노드의 링크 삭제

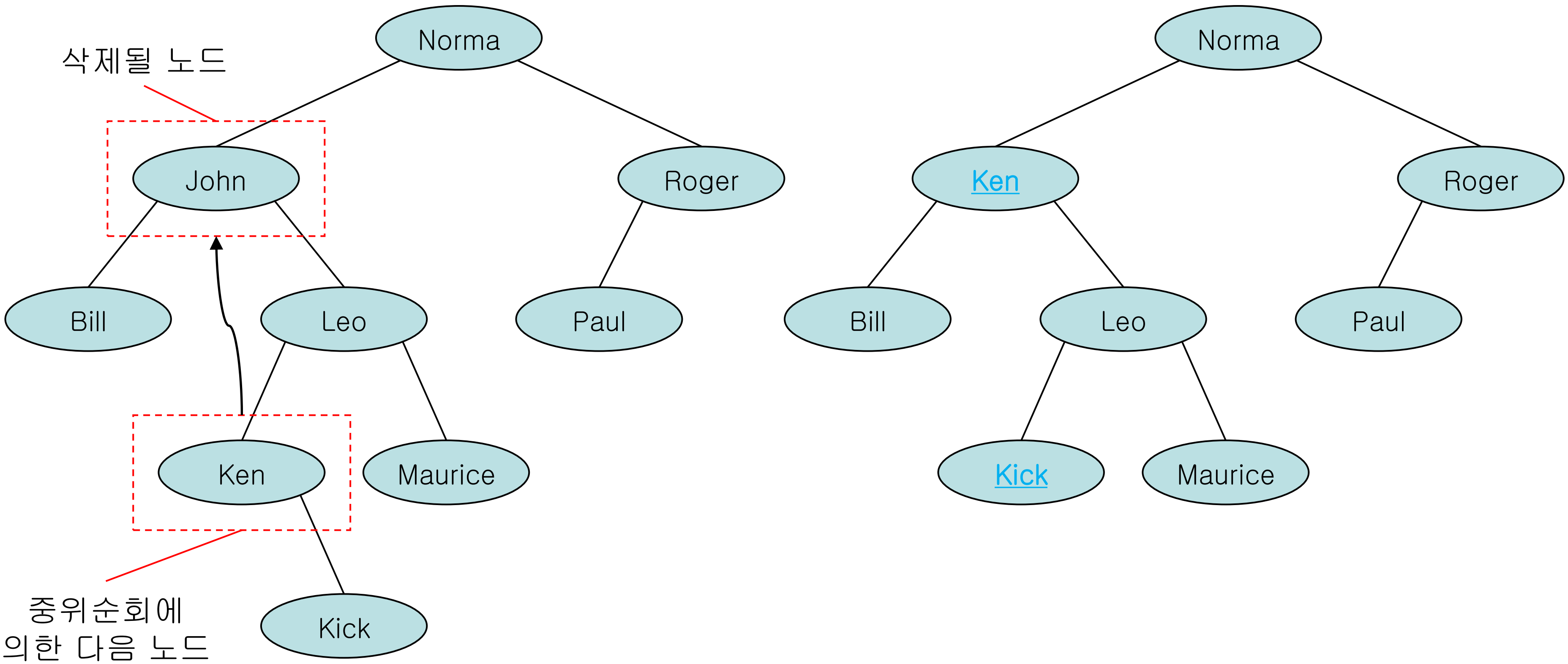
<삭제과정, 중간노드의 경우1>



=>불균형

- 이진탐색트리에서 노드의 삭제(자식노드가 하나인 경우)
    - 부모와 자식을 이어 줌

<삭제과정, 중간노드의 경우2>



삭제이전 삭제이후

- 이진탐색트리에서 노드의 삭제(두개의 자식을 가지는 경우)
  - 중위순회의 직후 노드 탐색
    - 오른쪽 자식 중 왼쪽자식을 가지지 않는 최초 노드
  - 이 노드를 삭제 위치로 옮기고
    - 부모의 왼쪽 자식이면 오른쪽 자식을 부모의 왼쪽 자식으로 수정
    - 부모의 오른쪽 자식이면 오른쪽 자식을 부모의 오른쪽 자식으로 수정

- 균형이진탐색트리 (AVL Tree)
  - 균형 잡힌(balanced) 이진 탐색 트리
  - 1962년 G.M. Adelson-Velskii와 E.M. Landis 가 그들의 논문 "An algorithm for the organization of information” 에서 발표함.
  - 지금까지 사용한 노드의 구조에 노드의 균형상태를 표시하는 균형계수 (balance factor) 필드를 하나 더 첨가시킨다.
  - 이 균형계수의 값은 그 노드의 왼쪽 종속트리에서 가장 긴 경로의 길이(LP)와 오른쪽 종속트리에서 가장 긴 경로의 길이(RP)에 따라 다음과 같은 네가지 값, 즉 L, E, R, C 중의 하나임.

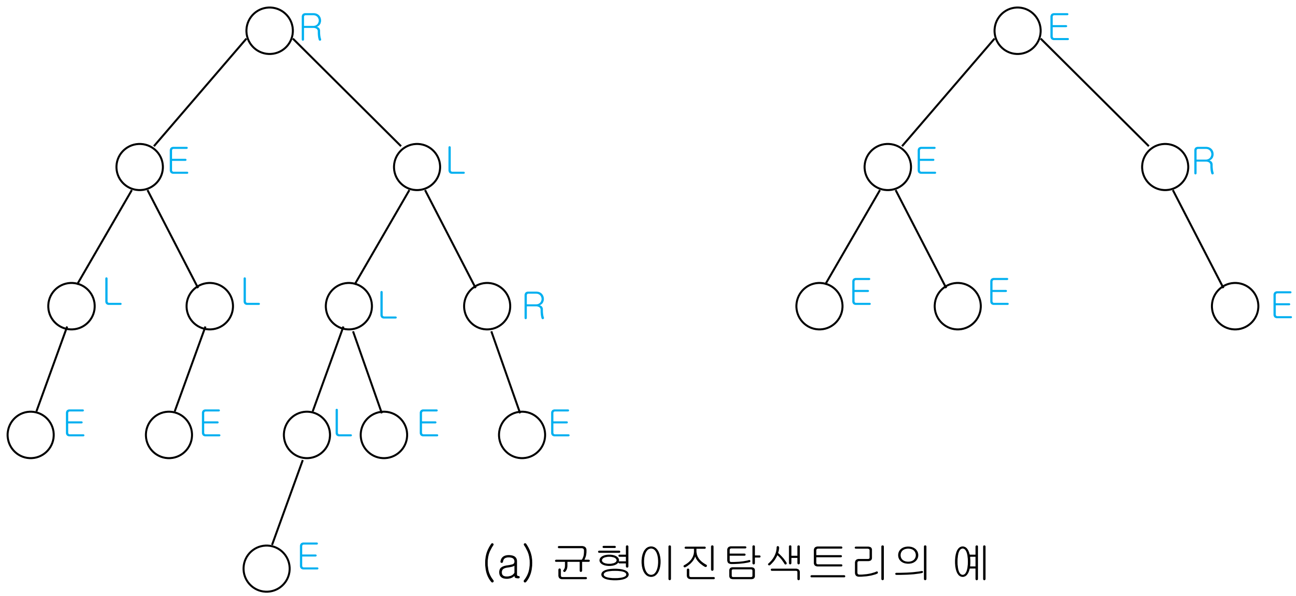
  L : LP=RP+1 일 때 : 이 노드는 왼쪽이 높다(left high)라고 한다.

  E : Ll=RP일 때 : 이 노드의 양쪽 높이가 같다(equal height)라고 한다.

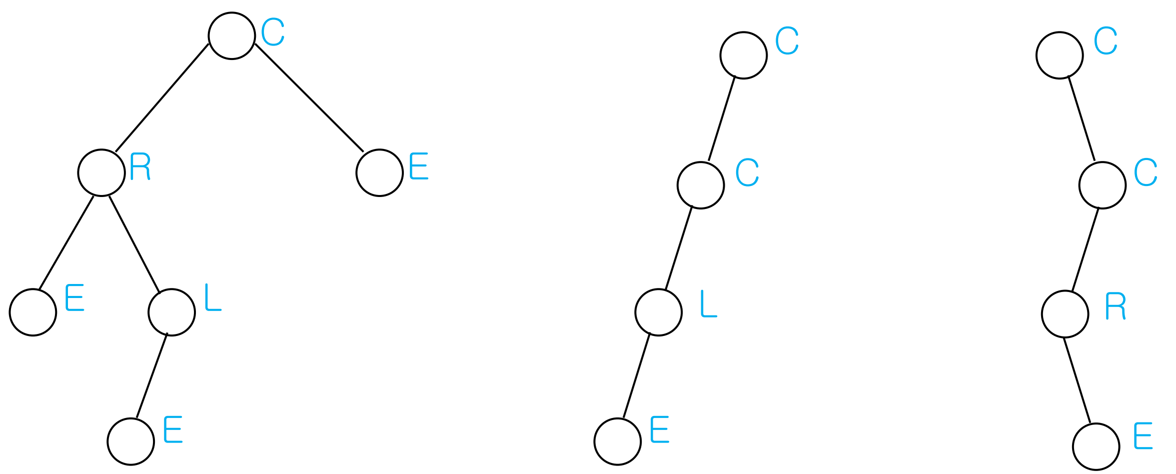
  R : LP+1=RP 일 때 : 이 노드는 오른쪽이 높다(right high)라고 한다.

  만약 이진탐색트리의 모든 노드가 L, E, R 상태만으로 구성

  ➔ AVL Tree



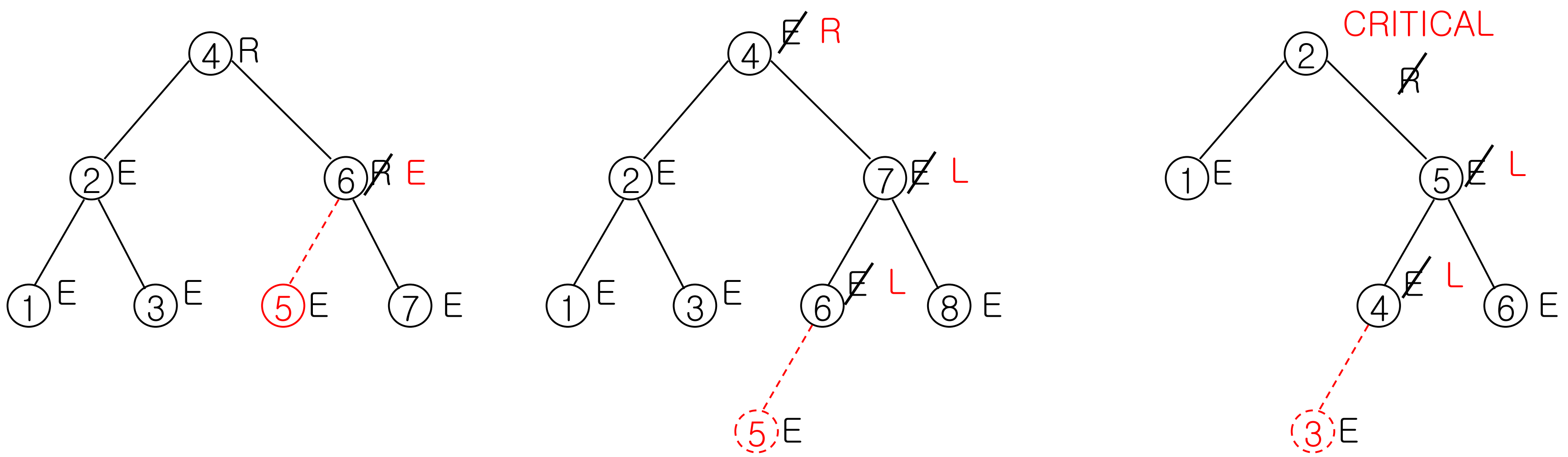
(a) 균형이진탐색트리의 예



(b) 불균형된 이진탐색트리의 예

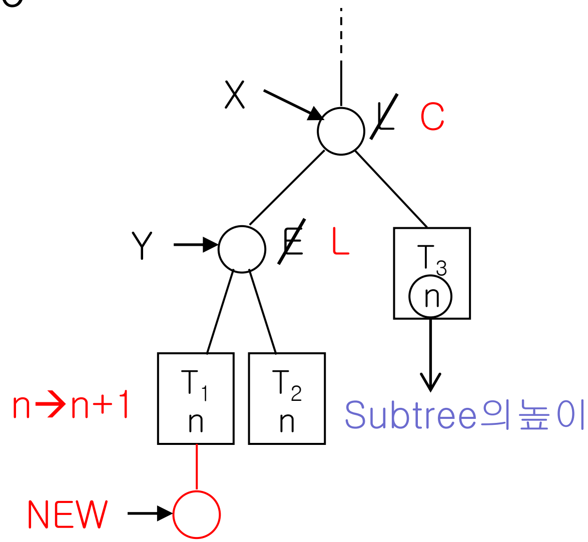
이진탐색트리의 균형상태를 표시한 예

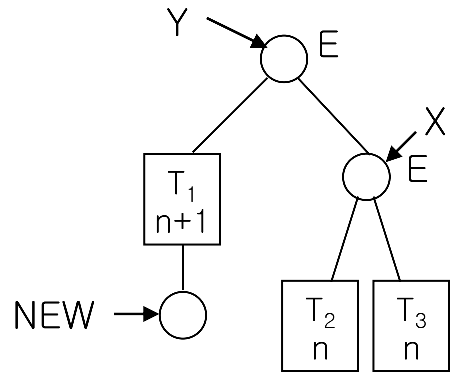
균형이진탐색트리에 노드를 입력할 때의 균형상태 변화의 종류



(a) 종류 1 (b) 종류 2 (c) 종류 3

LL Case





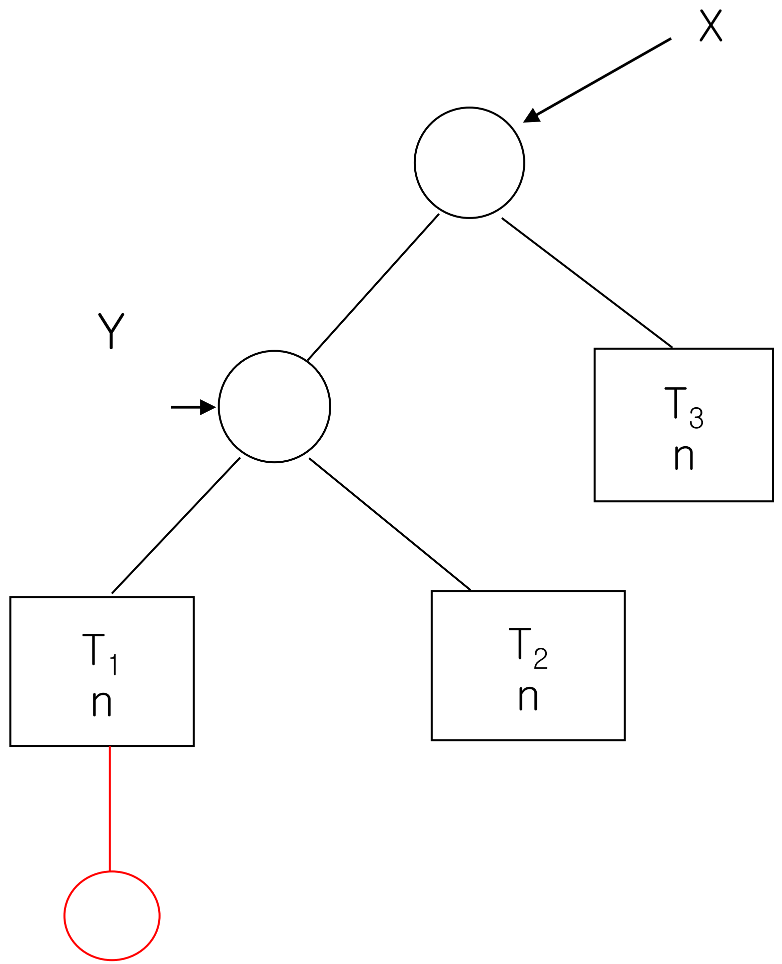
X가 Critical 노드로 되어
불균형된 트리

균형된 트리로 변환

Y가 L인 경우 (오른쪽회전)

오른쪽 회전(right rotation) Algorithm
X a node로 부터 가장 가까운 Critical node

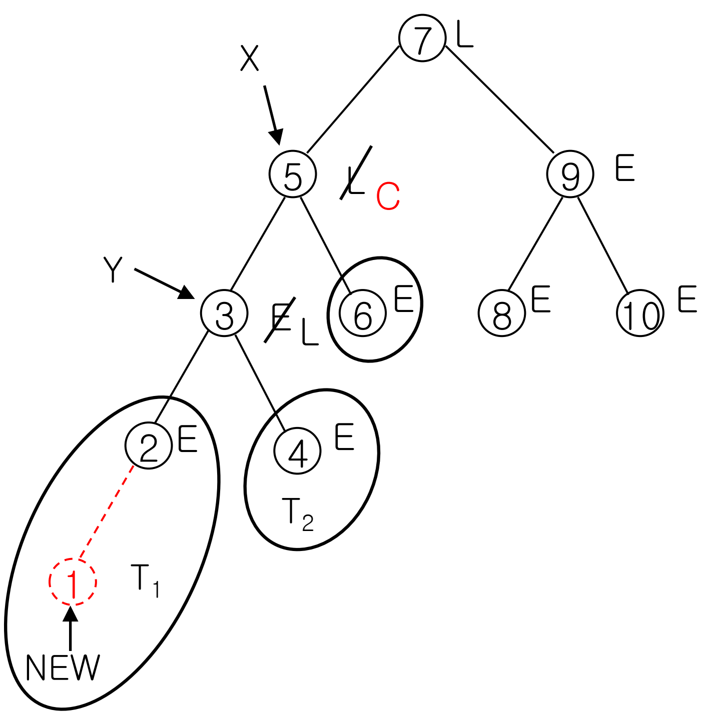
```
y = x -> left;
temp = y -> right;
y -> right = x;
x -> left = temp;
```



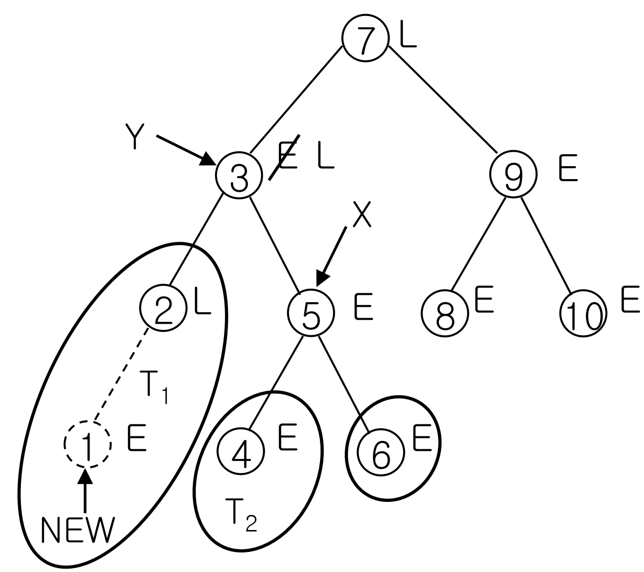
(오른쪽회전)

<Example>

LL Case

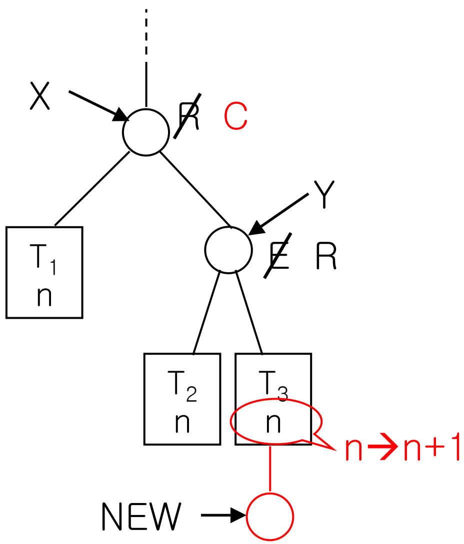


(a) 불균형된 트리

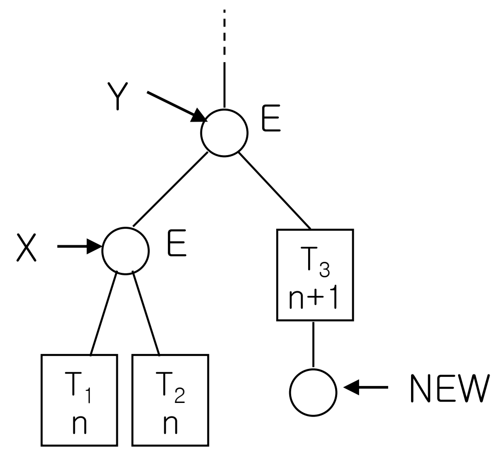


(b) 균형된 트리

RR Case

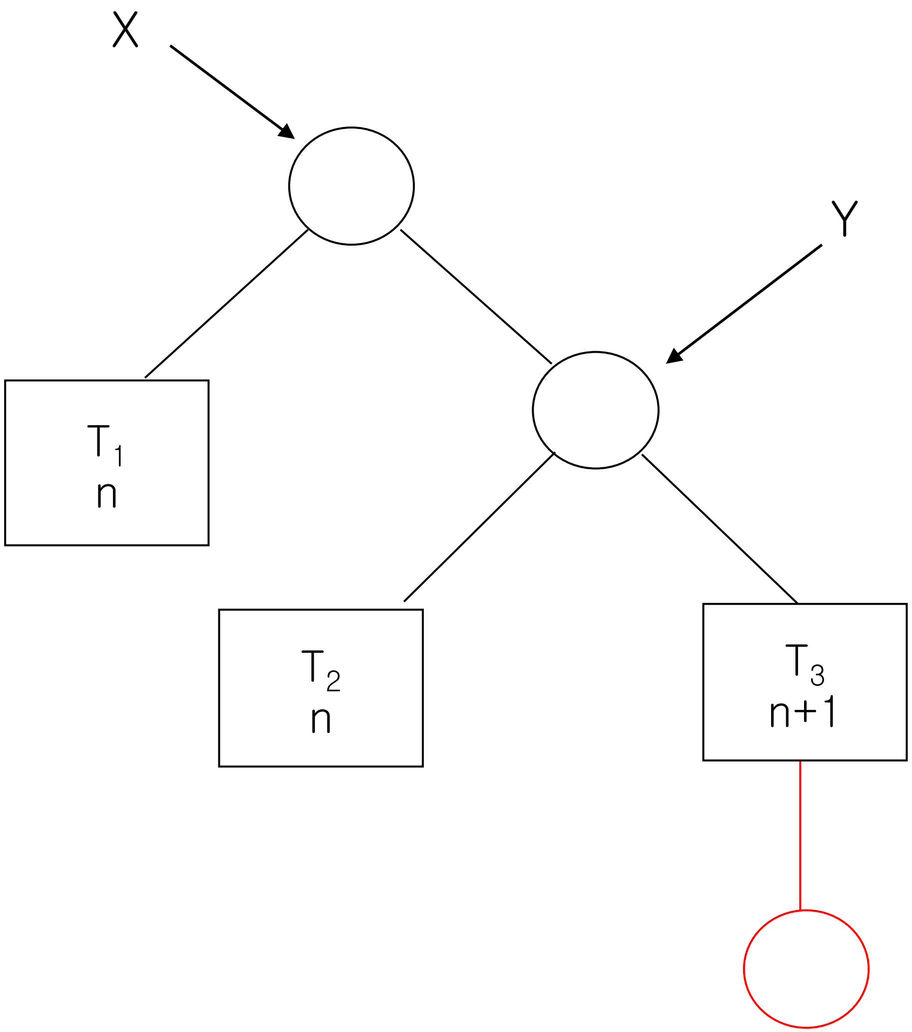


불균형된 트리



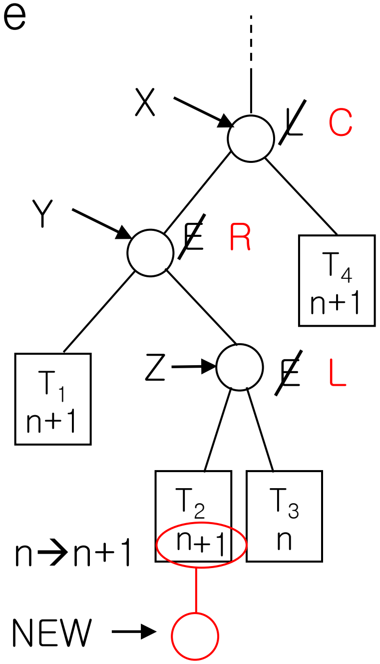
균형된 트리로 변환

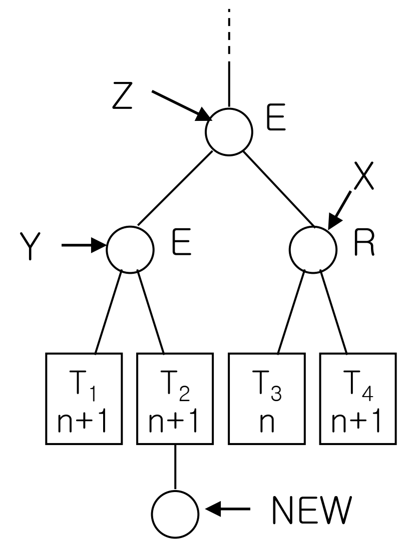
Y가 R인 경우 (왼쪽회전)



Y가 R인 경우 (왼쪽회전)

LR Case

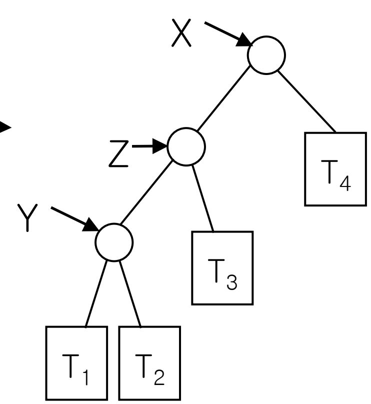




불균형된 트리

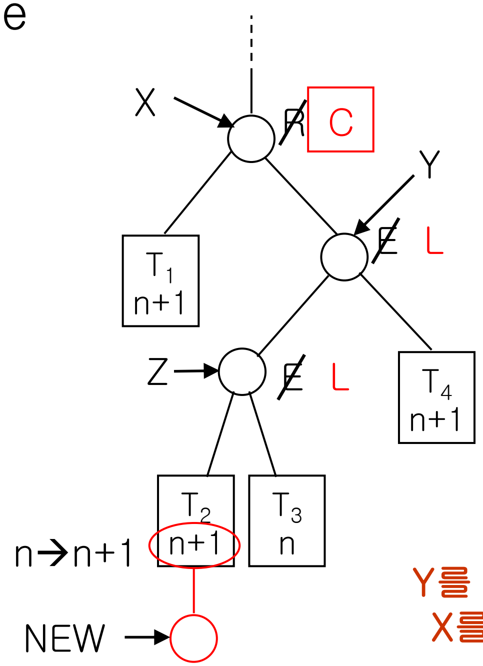
균형된 트리로 변환

1단계 : Y를 축으로한 왼쪽회전

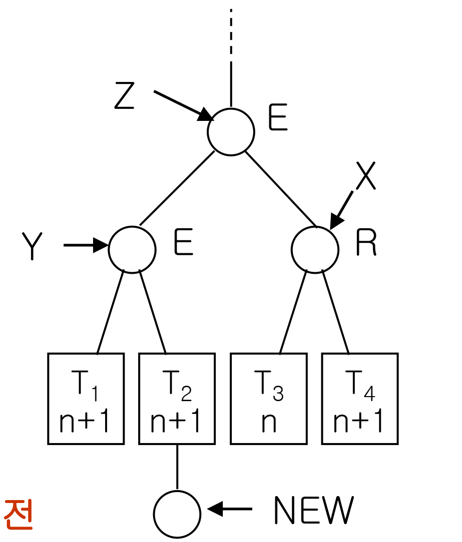


2단계 : X를 축으로한 오른쪽회전

RL Case

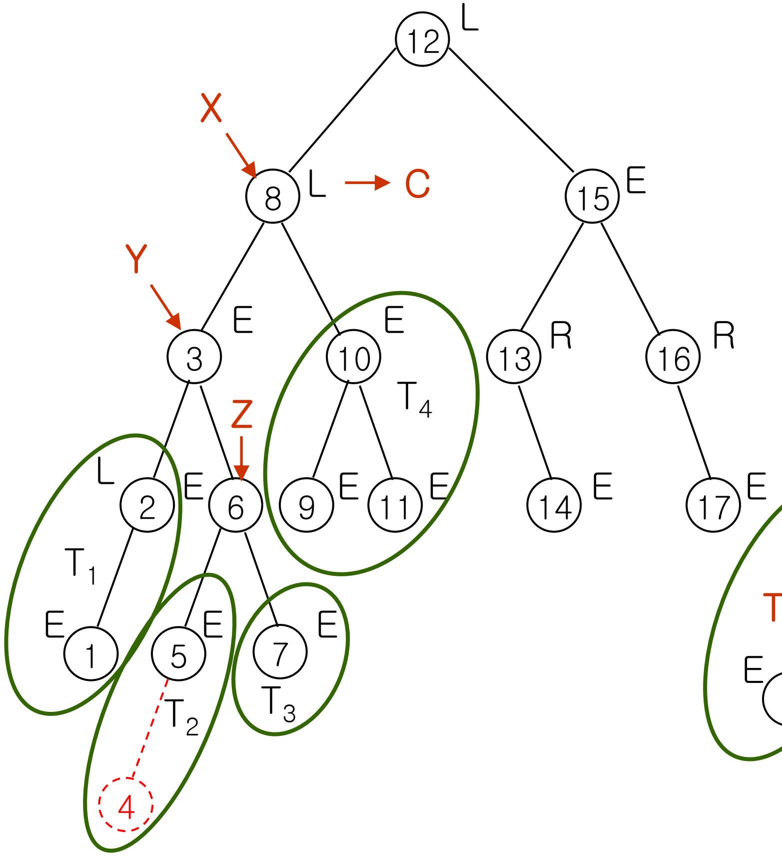


Y를 축으로한 오른쪽회전
X를 축으로한 왼쪽회전

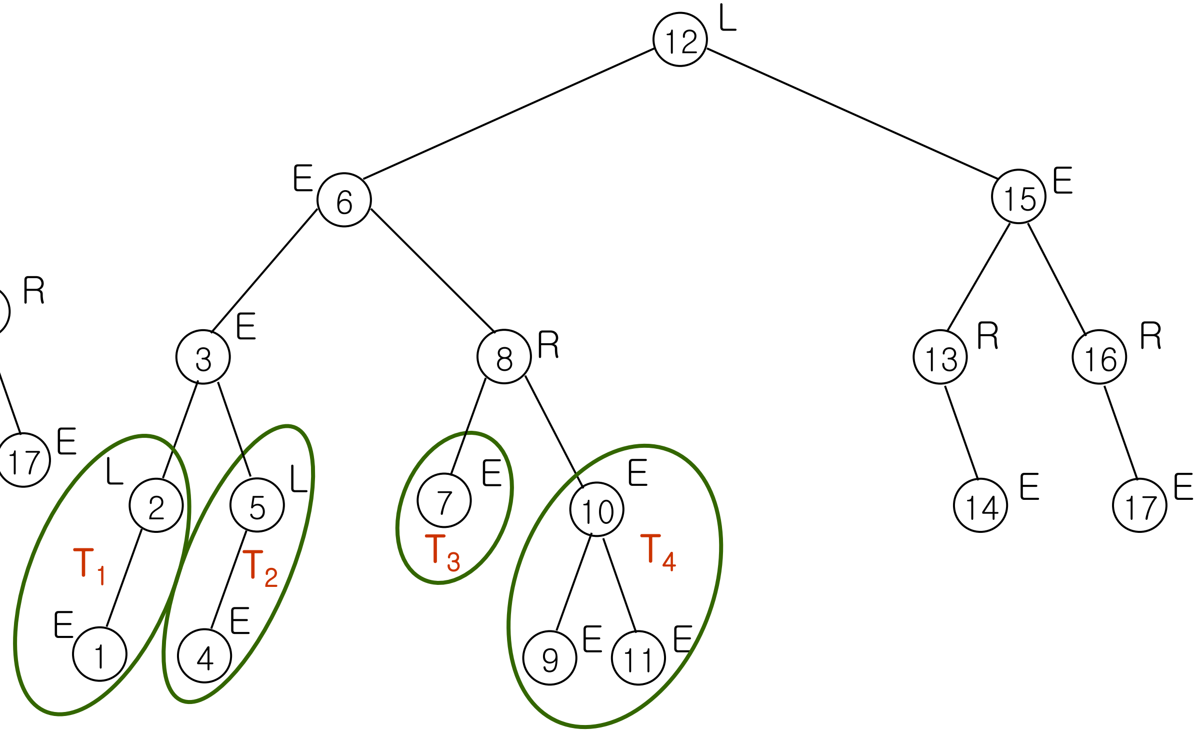


불균형된 트리

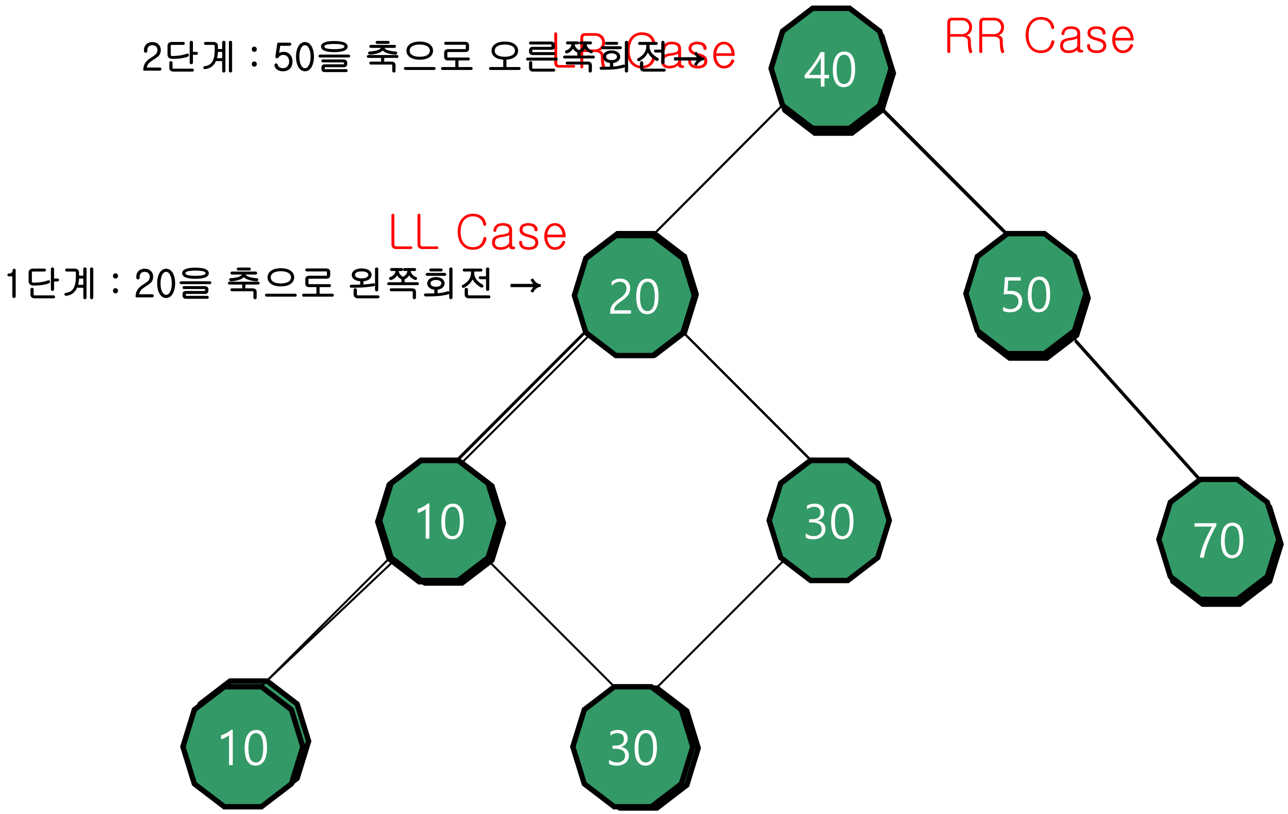
균형된 트리로 변환



(a) 불균형된 트리(LR Case)



(b) 균형된 트리



균형된 트리로 변환

<균형이진탐색트리>

Binary Search Tree(AVL Tree)

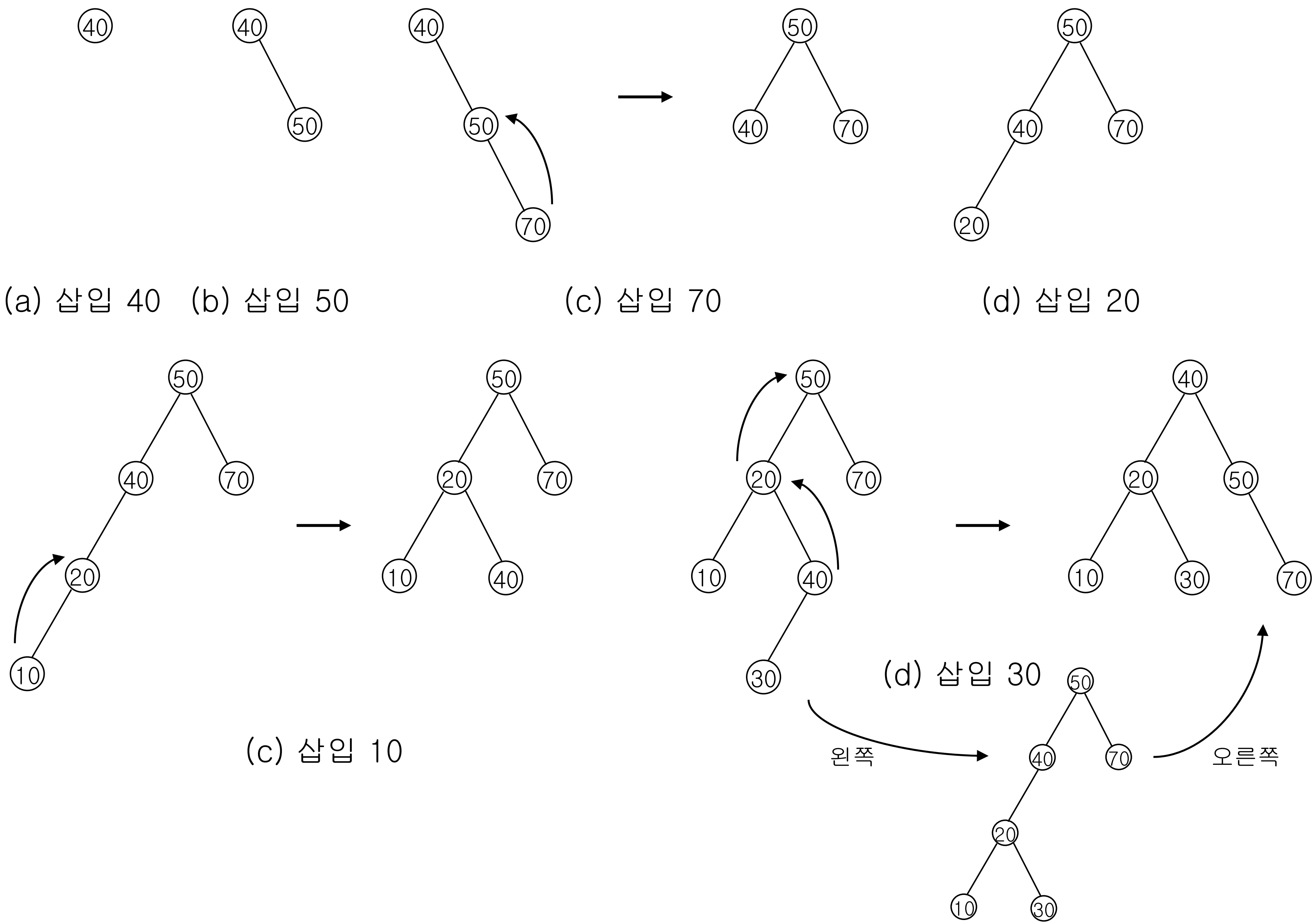


그림 12-13 균형이진탐색트리의 구성 예

M원탐색트리

- 한 개의 노드에 (m-1)개의 키와 m개의 종속트리를 가짐
- 2원탐색트리에 비하여
  - 높이 감소
  - 탐색시간 단축
  - 삽입, 삭제의 어려움
- M원탐색트리의 구조

n	P_1	K_1	P_2	K_2	…	P_{m-1}	K_{m-1}	P_m

- n=# of sub tree
- P_i =sub tree에 대한 포인터
- K_i = Key
- Key값은 반드시 오름차순
- P_i 가 지시하는 서브트리 내의 키값 $< K_i$
- P_{i+1} 가 지시하는 서브트리 내의 키값 $> K_i$

- B-Tree

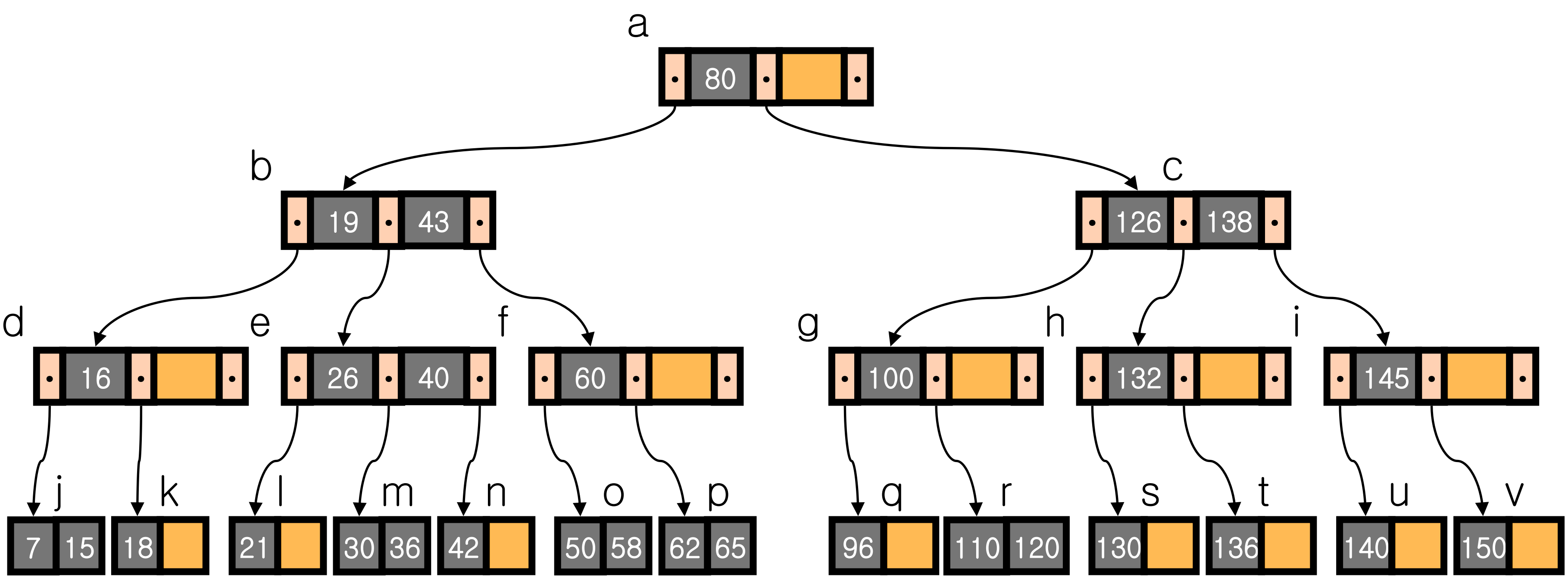
B-트리는 주로 보조기억장치에 저장된 데이타의 탐색에 이용
Bayer와 Mc Creight에 의해 제안된 것으로 균형된 m-원 탐색트리

B-Tree 정의

- 트리에 있는 각 노드는 최대 m개, 최소 $\lceil m/2 \rceil$개의 종속 트리를 가져야 한다(루트와 leaf 노드는 제외. 루트는 성질2 참조. leaf노드는 종속트리가 없음)
- 루트노드는 최소한 두개의 종속트리를 가져야 한다

  (단, 트리가 루트만 있는 경우에는 종속트리가 없음)
- 모든 leaf노드는 같은 level에 있어야 한다

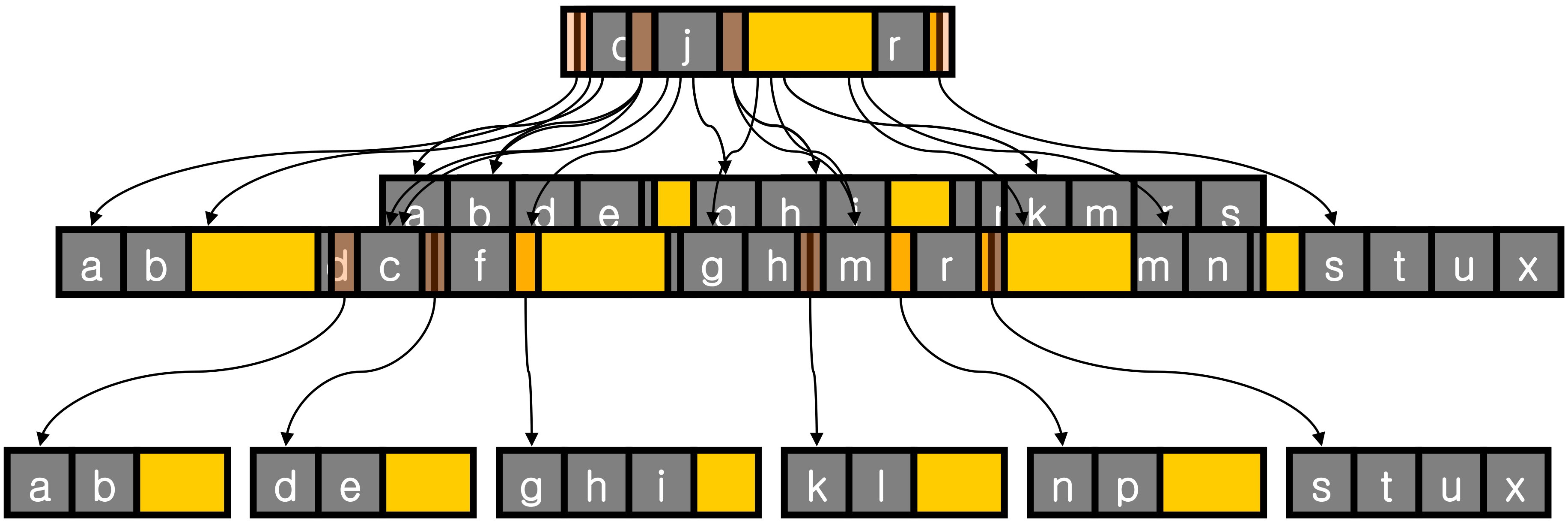
  (즉 모든 leaf노드는 루트로부터 같은 거리에 있음)
- 노드의 키값 갯수는 종속트리수보다 하나 적으며 최소 $\lceil m/2 \rceil - 1$ 개, 최대 (m−1)개이다 (루트노드 제외)
- 키값은 트리 내에서 유일하다
- 정렬된 상태로 저장된다.



차수3의 B-Tree

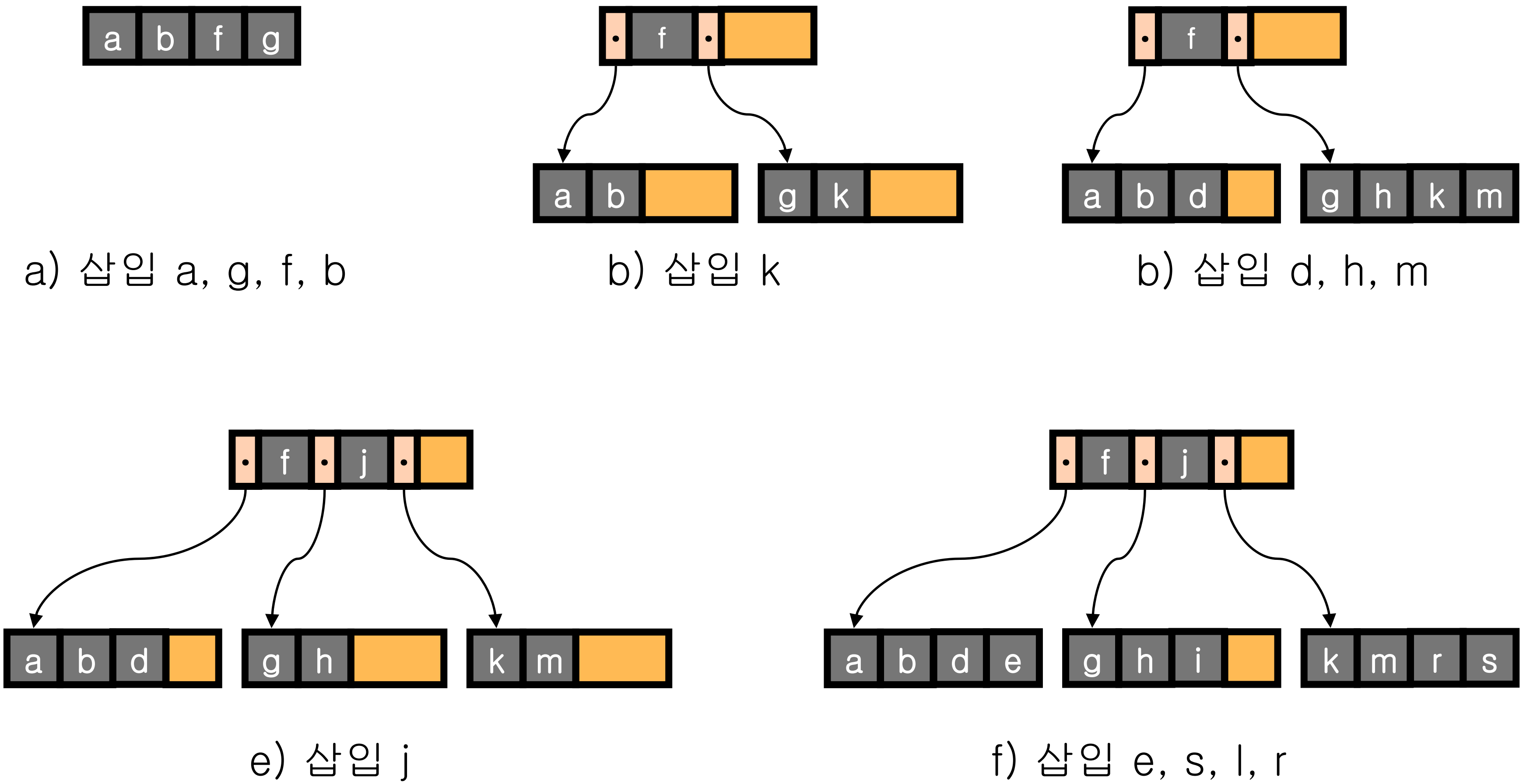
- B-tree에서의 탐색(search)
  - 특정 key값 탐색
  - 순차탐색 : 중위 순회 = B+ Tree
    (방문의 중복으로 순차 file보다는 성능이 떨어짐)

- 삽입 Algorithm
  - 빈공간이 있는 경우 : 단순 삽입
  - overflow : split(기존 Key값, 새로운 Key값) 中
    Median값([m/2]번째값) : Parent node로
    나머지 반씩 두 node로

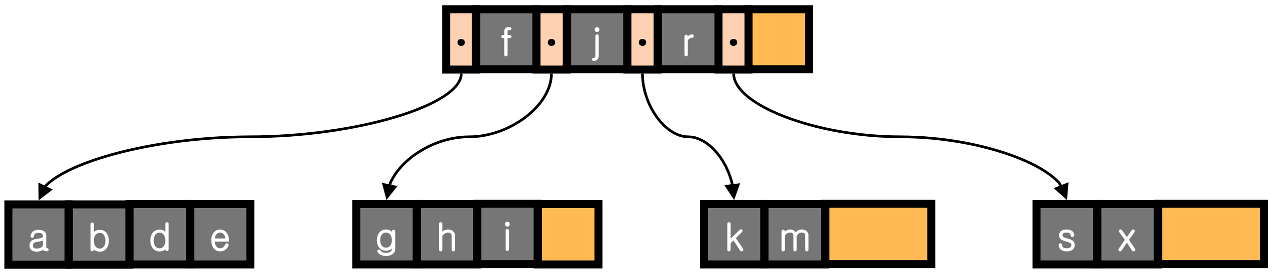


<B – Tree>

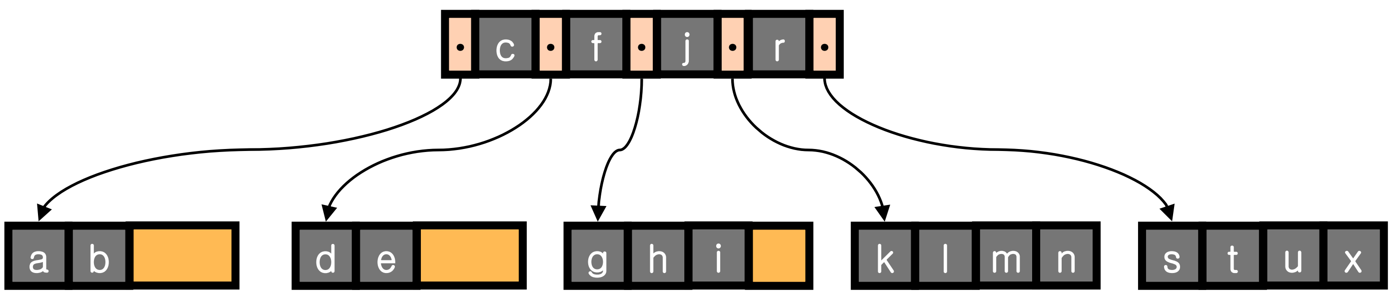
차수5인 B-Tree의 생성 과정



차수5인 B-Tree의 생성 과정

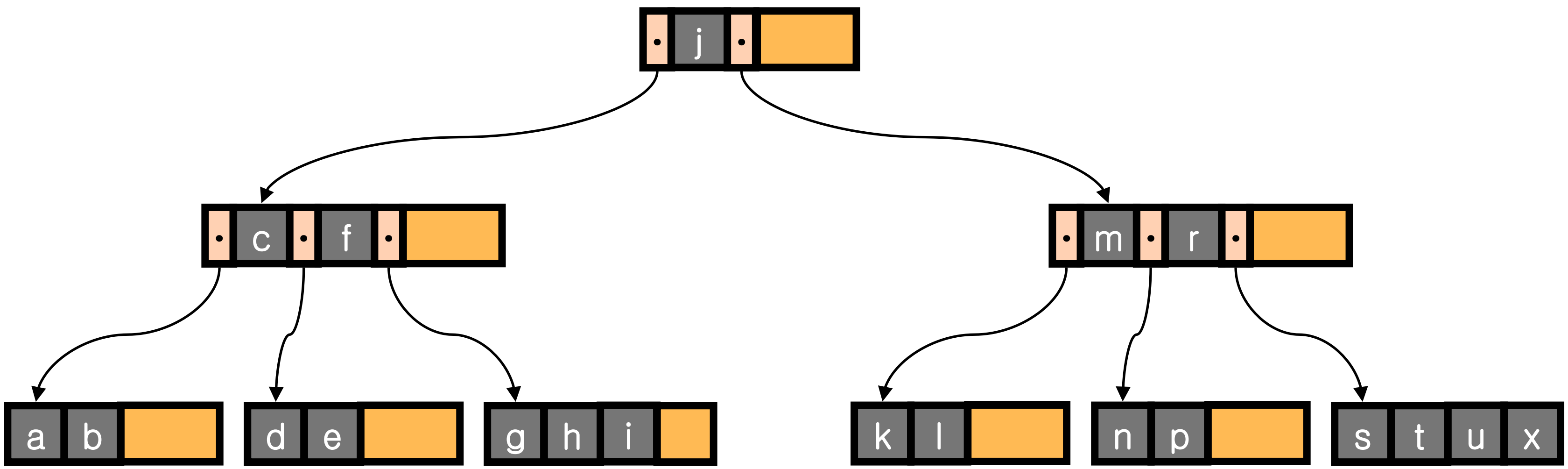


f) 삽입 x



g) 삽입 c, l, m, t, u

차수5인 B-Tree의 생성 과정



h) 삽입 p

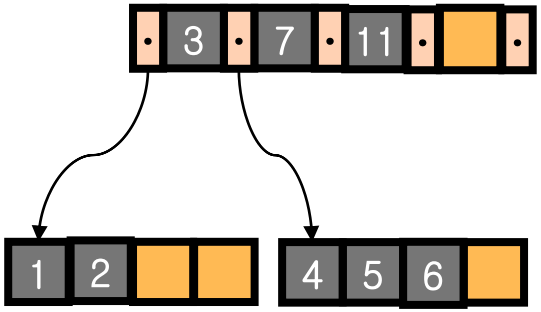
차수5인 B-Tree의 생성 과정

- 삽입 Algorithm
  - 빈공간이 있는 경우 : 단순 삽입
  - overflow : split(기존 Key값, 새로운 Key값) 中
    Median값([m/2]번째값) : Parent node로
    나머지 반씩 두 node로

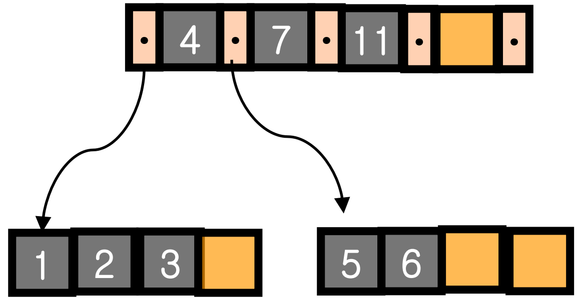
B트리의 삭제 알고리즘

- 삭제하려는 키 값을 가진 노드가 가지고 있는 키 갯수를 해당 키 값 삭제 후에 M/2 개 이상이 되도록 해야한다.
  - 만약에 부족하면(언더플로우)
    - 형제한테 빌리기
  - 형제도 부족하면
    - 형제와 결합하기
- 삭제 키가 있는 노드가 내부 노드인 경우
  - 대체 키를 찾아 대체
    - 왼쪽 서브트리 중 가장 큰 값
    - 혹은 오른쪽 서브트리 중 가장 작은 값.)

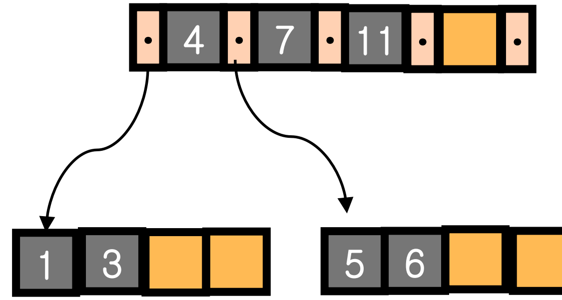
언더플로우 처리



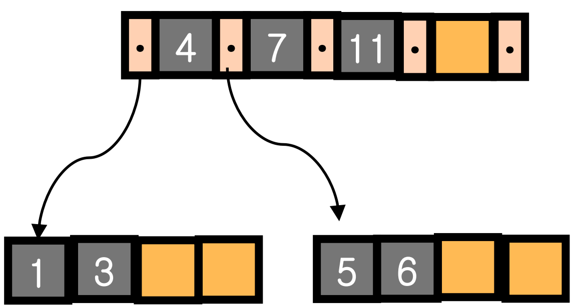
2 삭제
- (언더플로우)



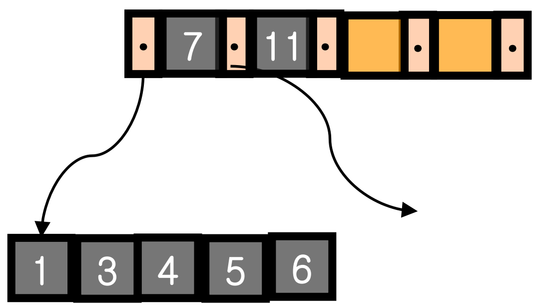
형제노드에서 빌려옴



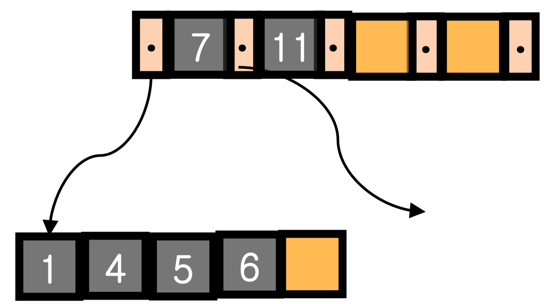
삭제



3삭제
빌려오면 형제노드도 언더
플로우 발생

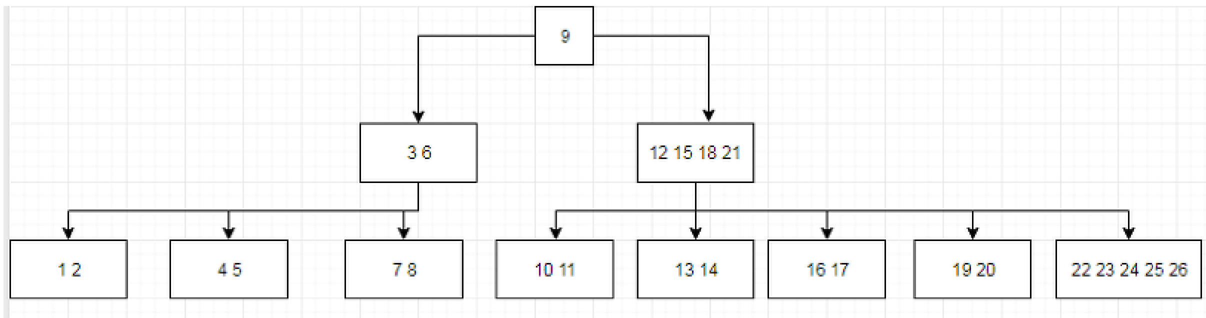


형제노드와 합병

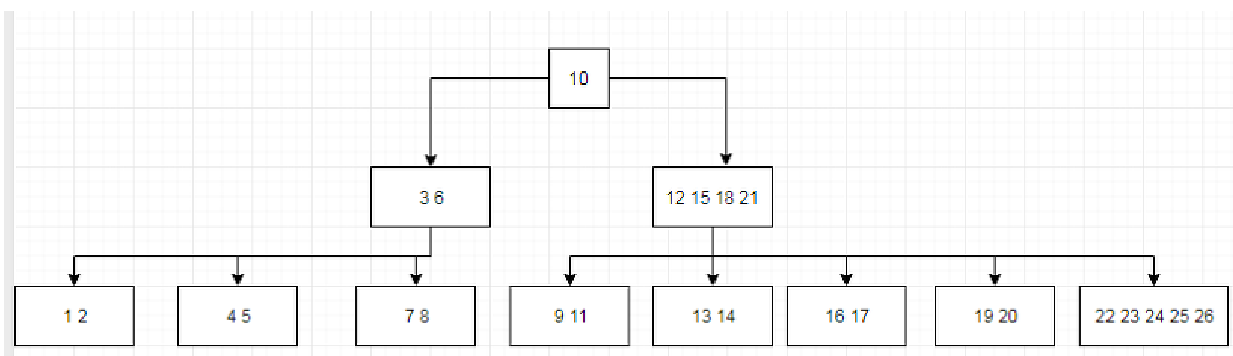


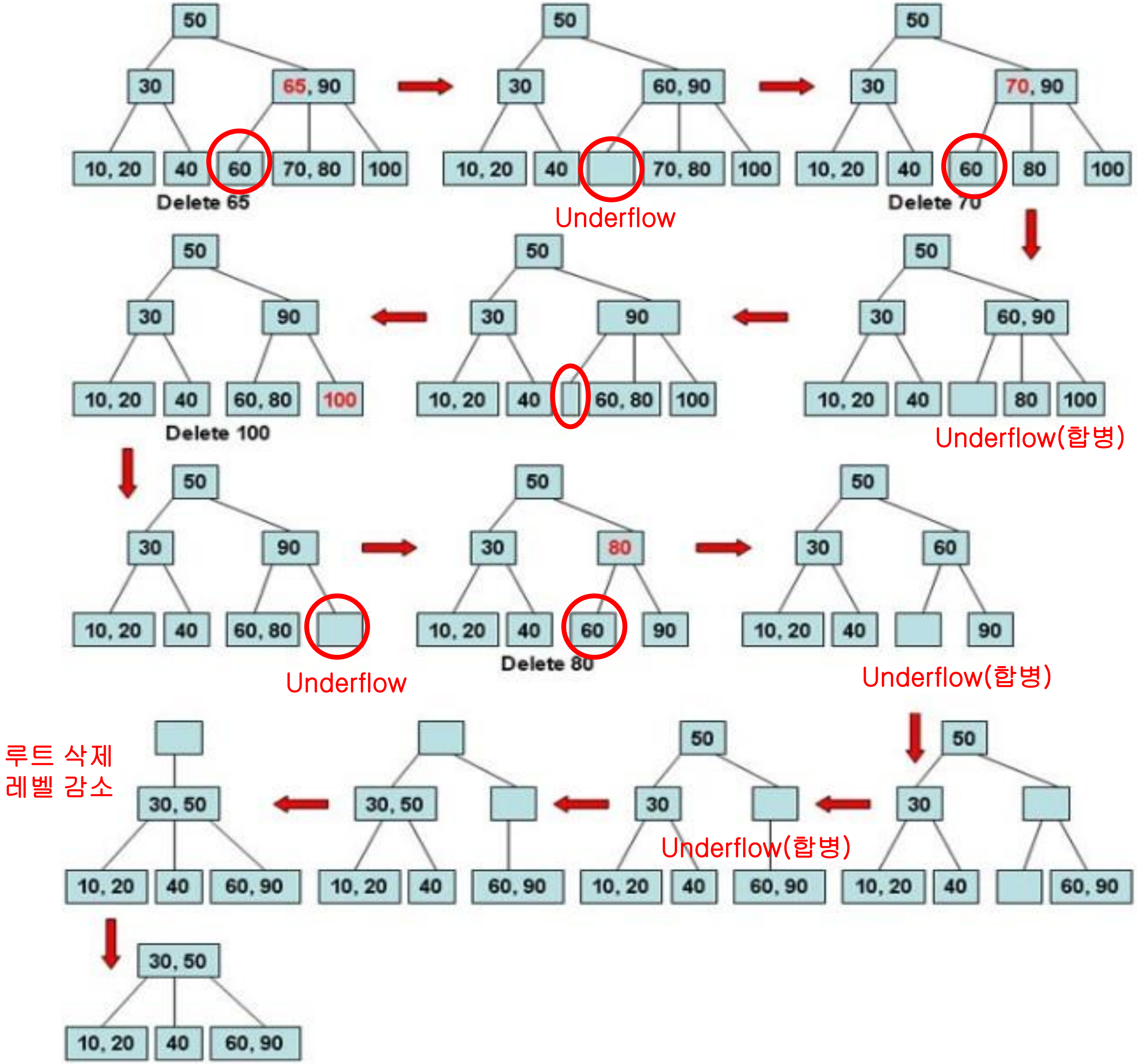
삭제

내부노드 삭제



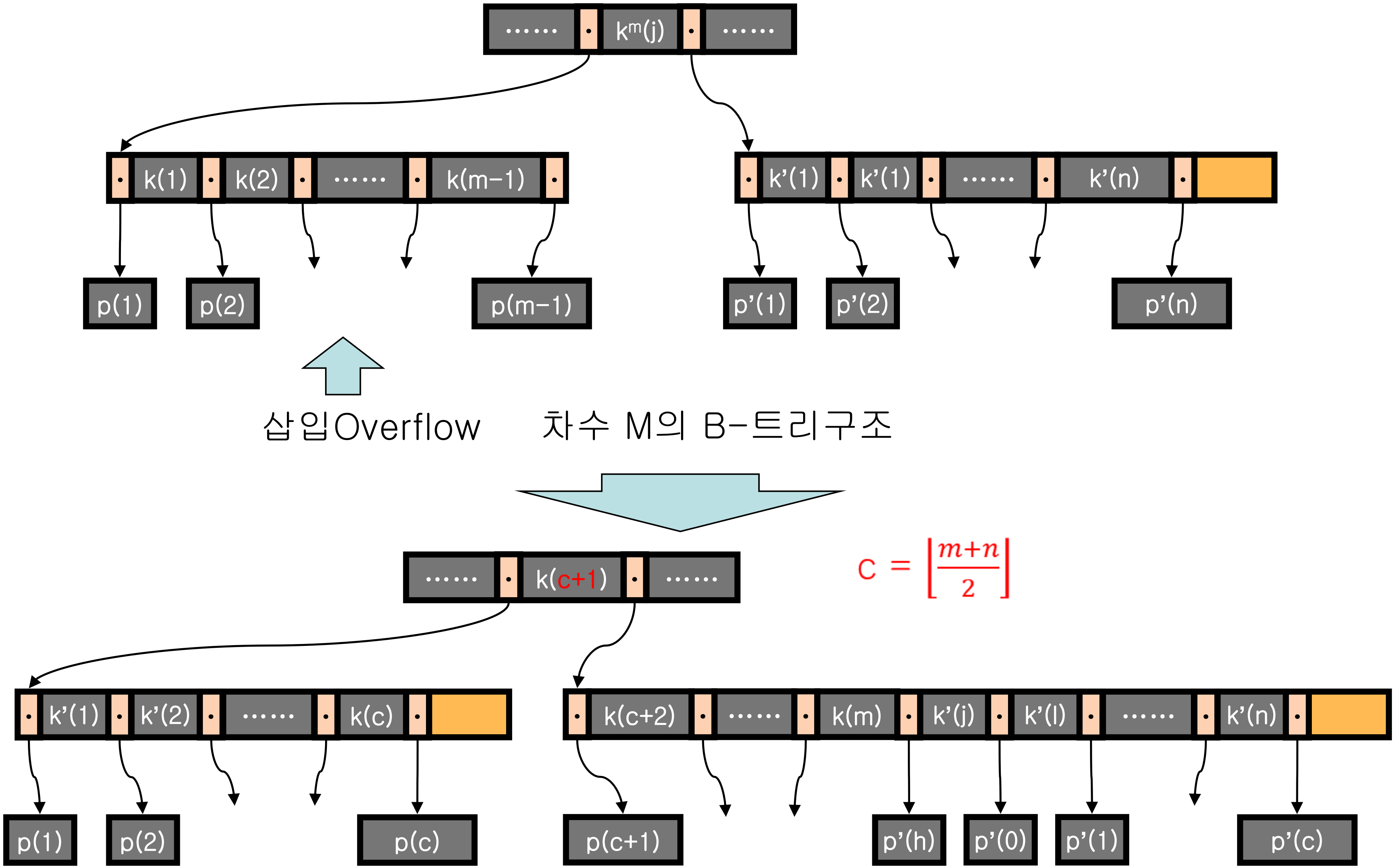
- 위 트리에서 9를 삭제
  - 노드의 왼쪽 서브 트리에서 가장 큰 값인 8
  - 혹은 오른쪽 서브 트리에서 가장 작은 값은 10으로 바꿔주고 9를 삭제




Delete 65
Underflow
Delete 70
Underflow(합병)
Delete 100
Underflow
Delete 80
Underflow(합병)
Underflow(합병)
루트 삭제
레벨 감소

B* Tree

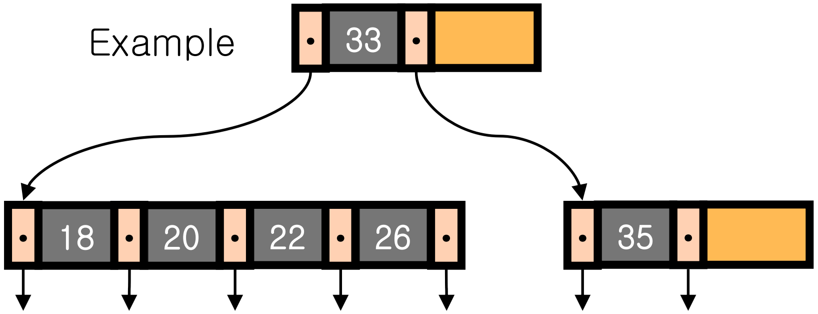
- B*-Tree (Knuth 제안)
  - B-Tree의 단점
    - 분열, 재분배, 합병 연산이 빈번 -> node 수 증가
    - 각 node는 절반 정도가 Key값으로 채워짐
  - B*-Tree의 기본 Idea
    - Overflow 발생시 Split 대신 형제 node로 재분배
    - 형제 node가 모두 Overflow이면 두 node를 세 node로 분할
      => 각 node는 2/3 정도가 채워짐



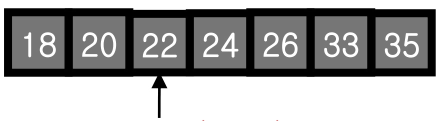
······
$k^m(j)$
······
k(1)
k(2)
······
k(m−1)
k'(1)
k'(1)
······
k'(n)
p(1)
p(2)
p(m−1)
p'(1)
p'(2)
p'(n)
삽입Overflow
차수 M의 B-트리구조
······
k(c+1)
······
$c = \left\lfloor \frac{m+n}{2} \right\rfloor$
k'(1)
k'(2)
······
k(c)
k(c+2)
······
k(m)
k'(j)
k'(l)
······
k'(n)
p(1)
p(2)
p(c)
p(c+1)
p'(h)
p'(0)
p'(1)
p'(c)

B-트리를 재배치한 결과

Example

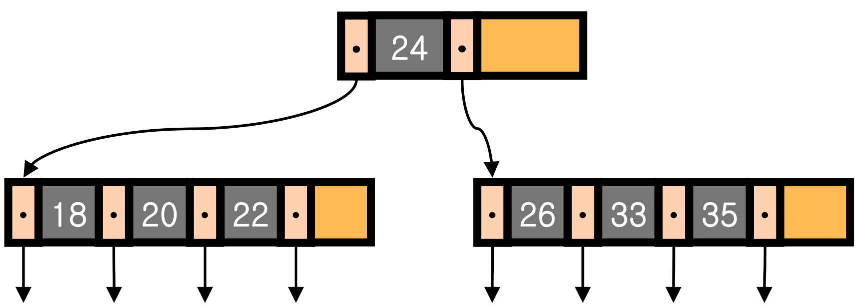


24를 삽입 => Overflow

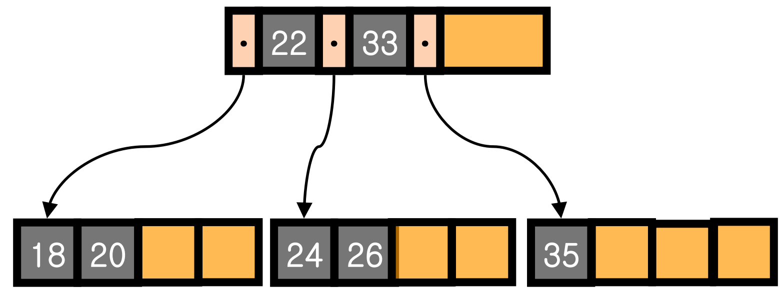


c=(5+1)/2=3

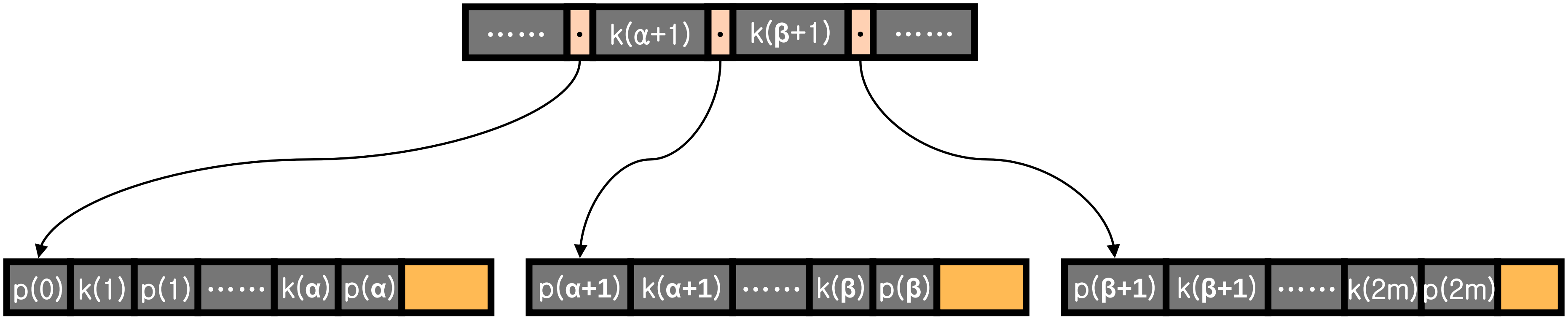
재분배



B-Tree의 경우



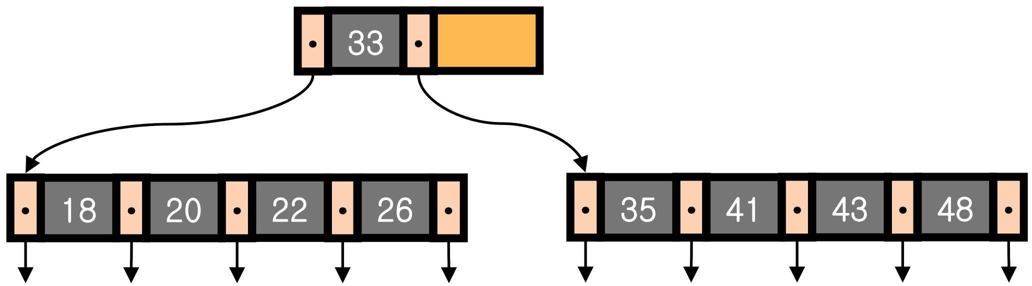
두 형제 node가 가득찬 경우



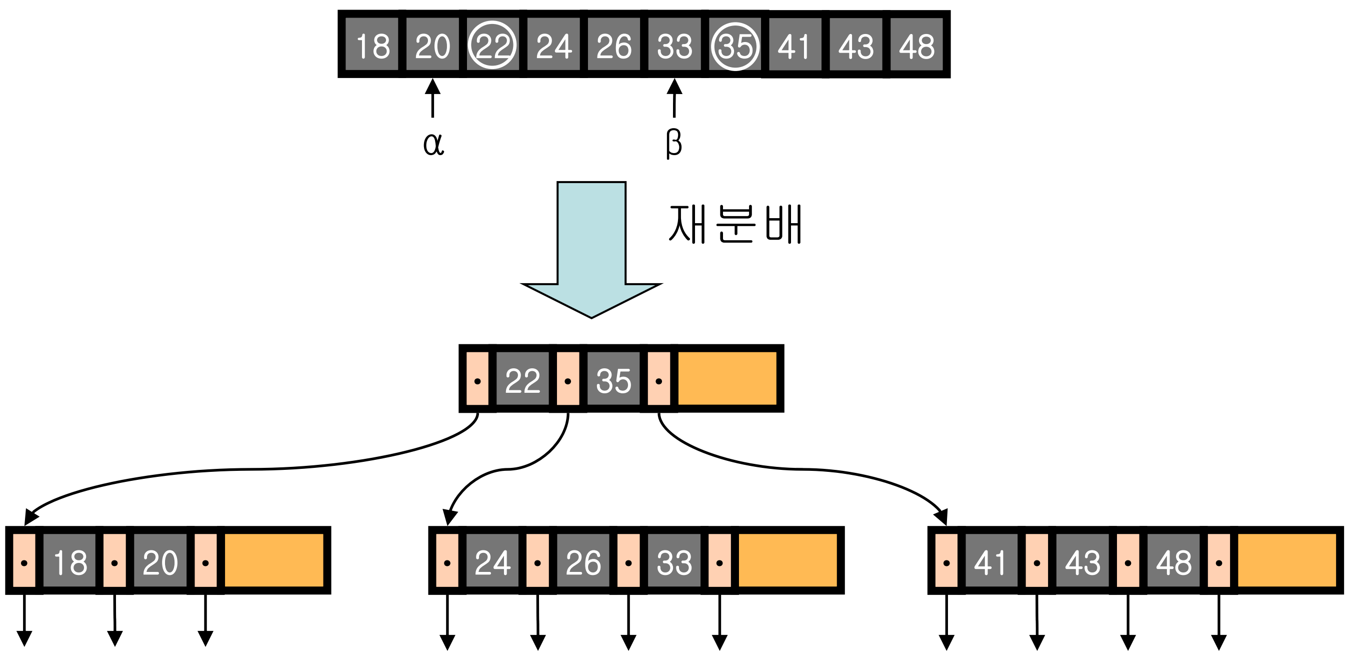
여기서 $$\alpha = \left\lfloor \frac{2m-2}{3} \right\rfloor$$

$$\beta = \left\lfloor \frac{2(2m-2)}{3} \right\rfloor + 1$$

Example



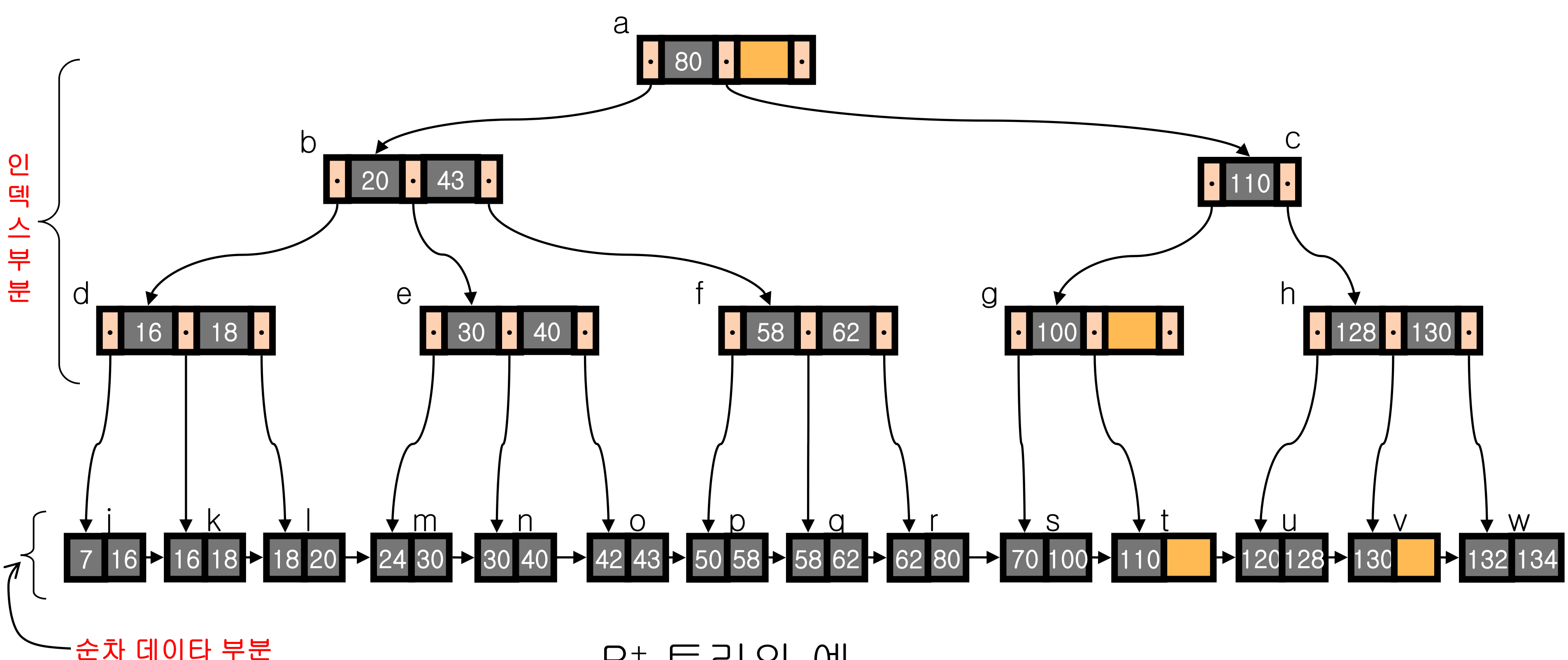
24를 삽입 => Overflow



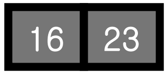
B^+ Tree

B^+-Tree (Knuth 제안)

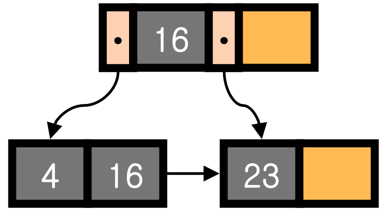
- Index Part와 Sequence Set로 구성
  - Index Part
    - Leaf node에 대한 경로 정보, Leaf node에 다시 나타남
    - Key 값만 수록 (주소는 없음)
  - Sequence Set
    - Leaf node로 구성
    - Key값과 주소쌍으로 구성
  - 순차, 직접 access가 모두가능
- 삽입, 삭제는 B-Tree와 거의 유사



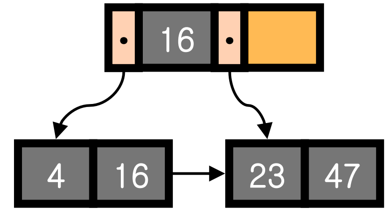
B^+ 트리의 예



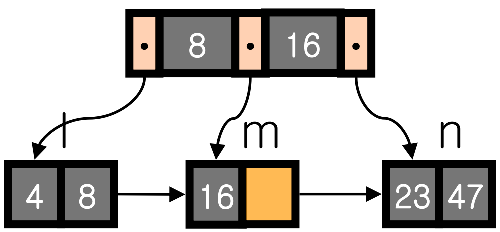
(a) 삽입 16, 23



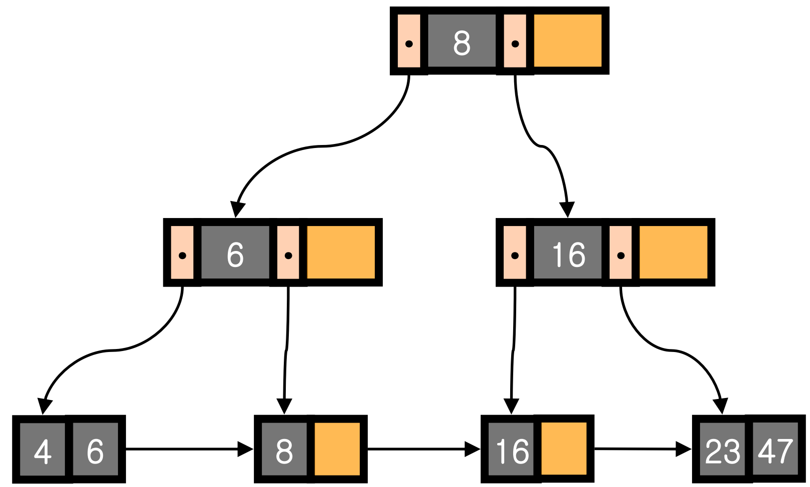
(b) 삽입 4



(c) 삽입 47

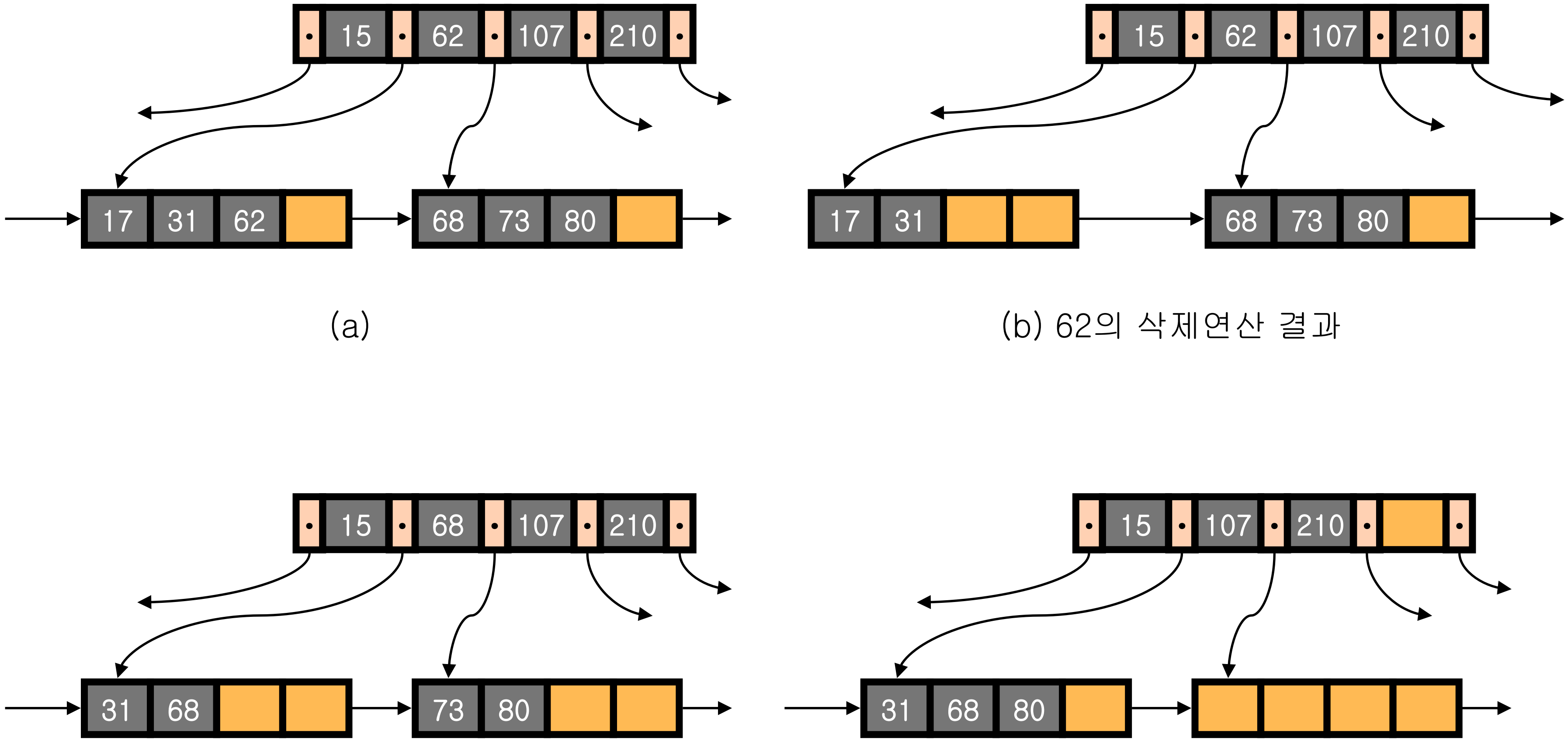


(d) 삽입 8



(d) 삽입 6

차수 3의 B⁺트리에서의 삽입 과정 예



(a)

(b) 62의 삭제연산 결과

(c) 17의 삭제연산(배치) 결과

(d) 73의 삭제연산(합병) 결과

차수 5의 B+트리에서의 삭제 과정 예