Ch 12. Balanced Search Tree (균형 검색트리)

1. AVL Tree - BBST (균형 이진검색트리) - rotation
2. Red-Tree - BBST 〃
3. B-Tree - 균형이진탐색
4. 2-3 Tree, 2-3-4 Tree

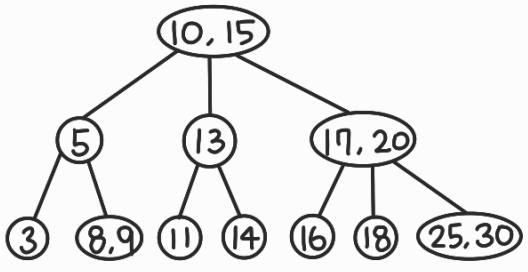
2. 2-3 Tree (Hopcroft, 1970)

: slow memory에 최적화된 검색트리로서 B-tree로 일반화

1) 정의

- B-tree의 n=2인 간단한 형태
- 각 node의 원소는 1~2개 (좌≤우)이며 그에 따르는 2~3개의 자식 node를 갖고있다.
- 자식 node는 두 원소의 중간값들로 구성된다.
- 삽입/삭제 후에도 모든 leaf는 항상 동일 level로 유지된다. (자동 균형)

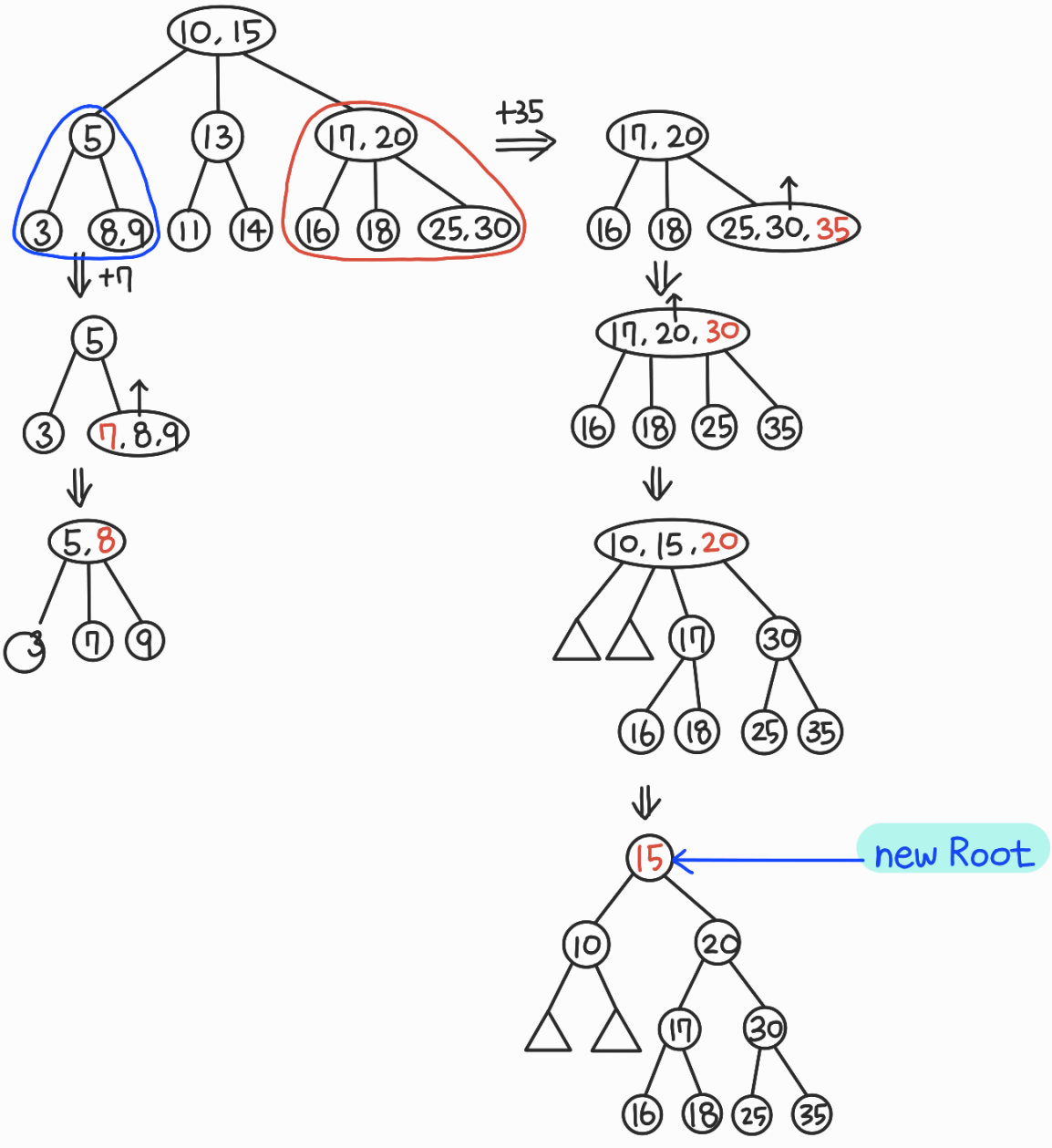
ex)



2) Insert(k) // data k를 삽입

```
{  1. data k는 leaf X에 추가됨.
   2. if(|X|≥3) Overflow(X)      node 원소 개수가 초과했을 경우
}
```

```
Overflow(X)   // |X|≥3
{  p ← X의 부모
   if(p=∧)  p = new empty Root
   X의 중앙원소를 p로 보내고, 양 끝을 그의 자식으로 한다.
   if(|p|=3)  Overflow(p)
}
```



3) Delete(k) // data k 제거

{ 1. X = data k가 속한 node

2. if (X ≠ leaf)

{ K의 후속자 (successor) S∈Y를 찾아 K↔S 교환

X←Y Y의 이름을 X로 바꿈 (Y: 단말노드)

}

3. k∈X 삭제 // X = leaf

4. if (|X| = 0) Underflow(X) X가 빈 노드일 때

}

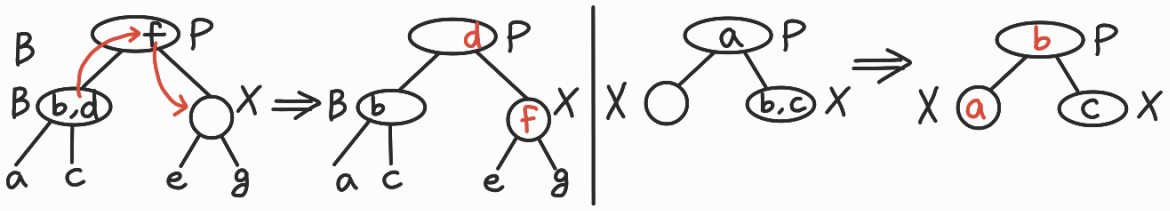
Underflow(X) // X = ∅

{ 1. P←X의 부모 (empty tree 포함)

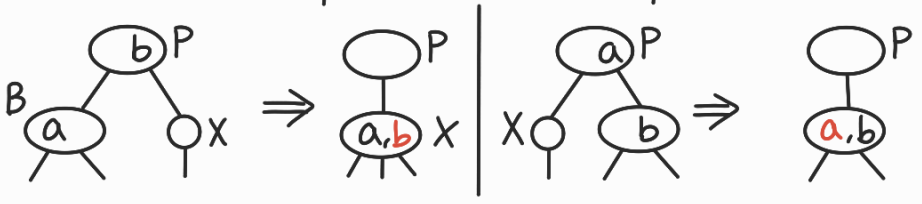
if (P = ⊥) Root ← X의 자식 : 종료

2. B←X의 형제 node

3. B에 여분이 있으면 옮겨 받고 종료

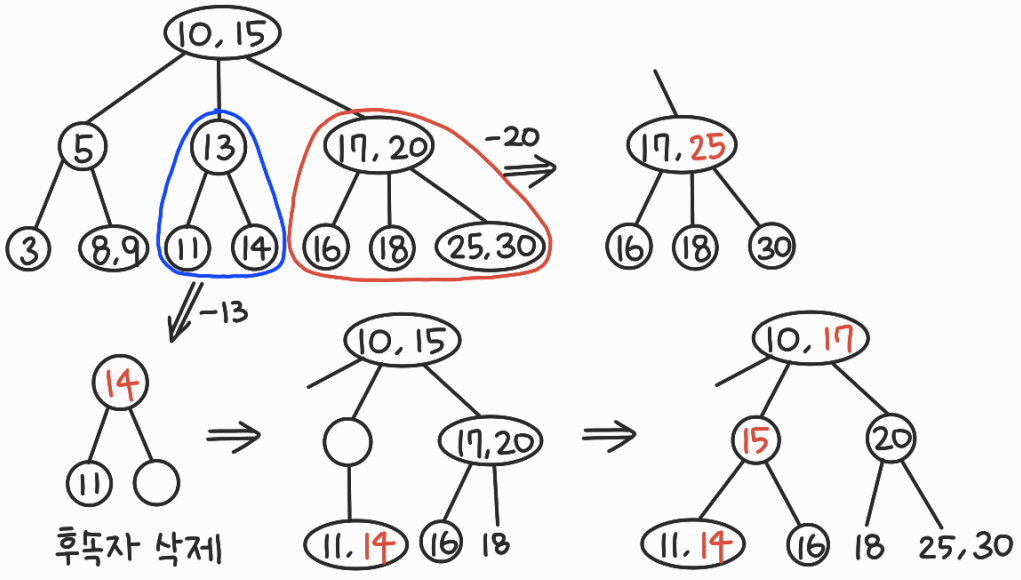


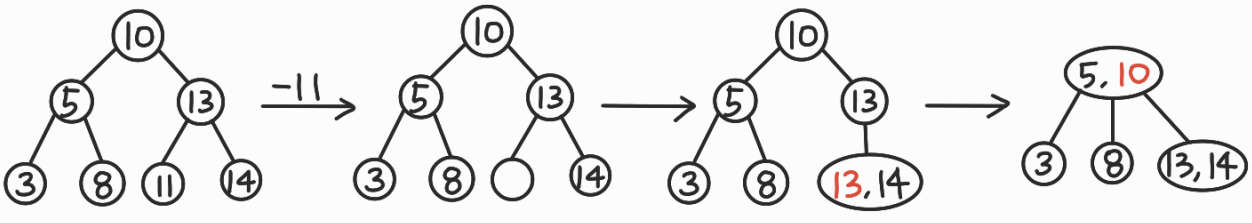
4. X를 B와 합병 (p에서 내려 받음 → p = ∅이 됨)



5. Underflow(p)

}





검색 / 삽입 / 삭제가 모두 O(lg N)에 가능

3. B-tree (Bager, McCreight, 1972)

"Symmetric Binary B-trees"

- 2-3 tree, 2-3-4 tree 등을 일반화 시범
- 1993, 99, 2001, 2008 … ,? upgrade

① 정의

- Root를 제외한 모든 node는 좌→우 증가순으로 K의 data를 갖는다.

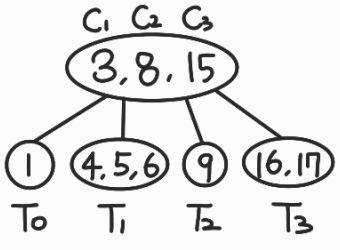
  단 $\lfloor \frac{n}{2} \rfloor \le K \le n$ for some n

  ex)
  - n=1 → 일반 이진트리
  - n=2 → 2-3 tree
  - n=3 → 2-3-4

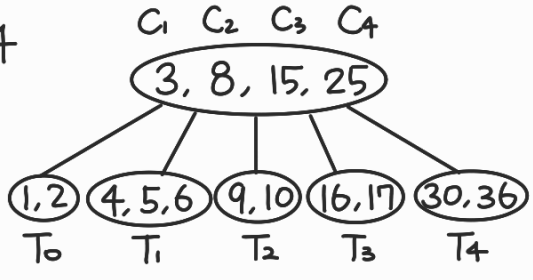
- k개의 원소($C_1 \le C_2 \le \cdots \le C_K$)를 가진 internal node는 중간값들로 구성된 k+1개의 자식 node ($T_0 \le T_1 \le \cdots \le T_K$)를 갖는다.

  (i.e $T_{i-1} \le C_i \le T_i$, $i = 1 \sim k$)

ex) n=3



n = 4



- 모든 leaf는 동일 level에 유지

② 특징

1) 이진 검색 트리 (BST)의 향상된 형태

2) 최악의 경우 검색/삽입/삭제가 $O(\lg n)$에는 가능 ← 자동균형

3) rotation에 의한 재균형 (AVL, BB) 보다 효율적으로 평가

4) slow memory에 최적화

5) 왜 B-tree? X

③ 삽입 → overflow()

삭제 → Underflow()

4. Red-Black tree (Guibas, Sedgewick, 1978)

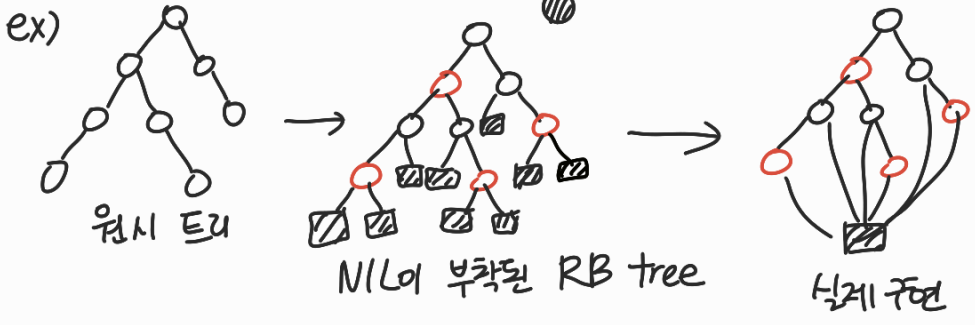
"A Dichromatic Framework for Balanced Tree" 에서 RB 소개

· self-balancing binary search tree 실제 사용에 효율적

① 정의

- 또 다른 균형 이진검색트리 (BBST) 로서 삽입/삭제 후 회전에 의해 재균형
- 모든 node는 Red or Black의 color를 갖는다. (color-bit 추가)
- Root ⇝ 각 leaf 경로상의 Black node 갯수는 항상 동일 (=black height)
- 모든 leaf는 data가 없는 가상의 node (=NIL) 로서 data node 에 부착됨
- Root의 모든 leaf (=NIL)은 Black
- Red의 자식은 항상 Black

ex)



② Insert (x): node X

{ 1. 삽입 X를 ○ 하여 정상 BST 삽입 후 2개의 Black NIL 부착

2. p ← X의 부모가 ◍ 이면 종료

3. Now p = ○ ≠ root ∧ p_2 ← p의 부모 ◍

→ p의 형제 Q의 color에 따라

Q = { ○ : case ⓘ
◍ : case ⓘⓘ ⓘⓘⓘ } 로 분류하여 재균형

③ Delete (X)

{ 1. 삭제할 X를 정상 BST로 제거 → Z = 실제 삭제된 node (자신 or 후속자)

2. Z=○ 이면 종료:

Why? ① black height에 무관

② Red는 상하로 연결되지 않는다

③ Z ≠ Root (=black)

3. Z = ◍ 이면 가능한 violations