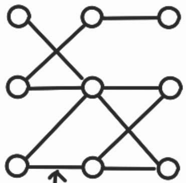
Ch 7 트리 (Tree)

Graph $G = (V, E)$

V : set of vertex

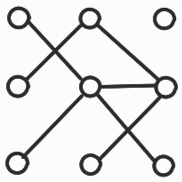
E : 〃 edges

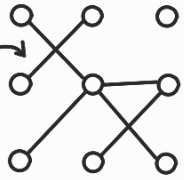


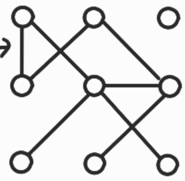
Tree는 graph의 특수한 경우로서

- ① connected (연결)
- ② acyclic (비순환)

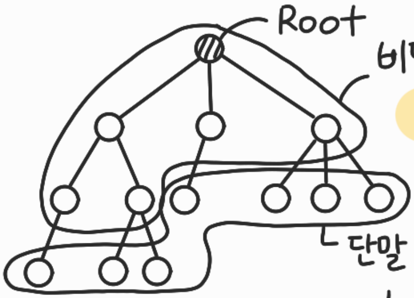
tree $T = (V, E) \underset{\not\Leftarrow}{\Rightarrow} |V| = |E| + 1$

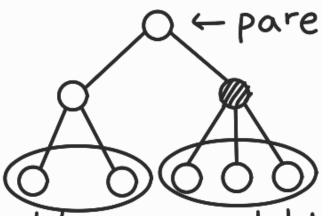


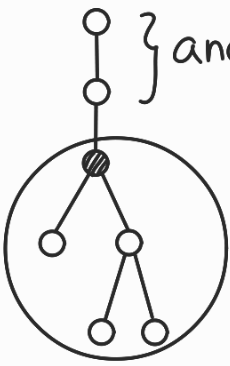


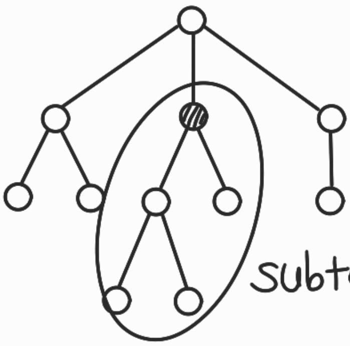


Tree의 용어들





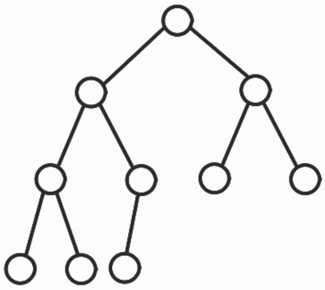




K-ary tree : 모든 node의 자식이 ≤k인 tree
(k진 트리)
$k=2$: 이진 tree (binary tree)
$k=4$: 4진 tree (quater tree)
$k=8$: 8진 tree (oct tree)

1. 이진트리 (binary tree)

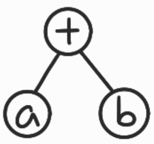
: 모든 node의 자식이 둘 이하인 tree → left, right 로 구분



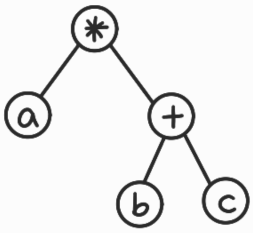
Node 구성

<table>
<tr><td colspan="2">Data</td></tr>
<tr><td>left link</td><td>right link</td></tr>
</table>

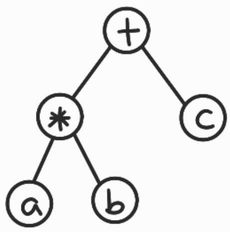
ex) expression tree (수식 트리)



a+b



a * (b+c)

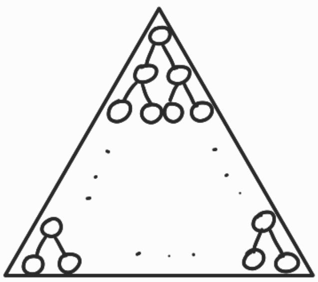


a*b+c

2. 이진트리 (Binary Tree : BT)의 종류

① 포화 이진트리 (Full BT)

⇔ 1) 모든 non-terminal node (leaf 제외)가 두 개의 자식을 갖는다.
2) 모든 leaf node의 level이 동일

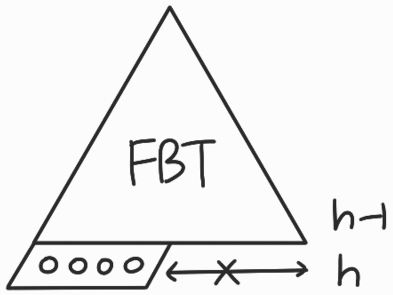


→ level 0 : 1
→ level 1 : 2
→ level 2 : 2^2
⋮
→ level h : 2^h

total # of node $N = 2^0+2^1+2^2+\cdots+2^h = 2^{h+1}-1$

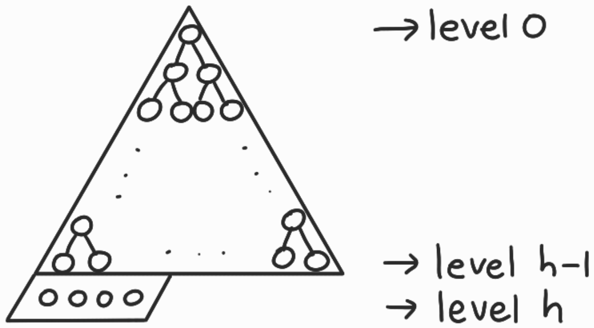
height $= \lg(N+1)-1 \approx \lg N$

② 완전 이진트리 (Complete BT)



$FBT \underset{\not\Leftarrow}{\Rightarrow} CBT$

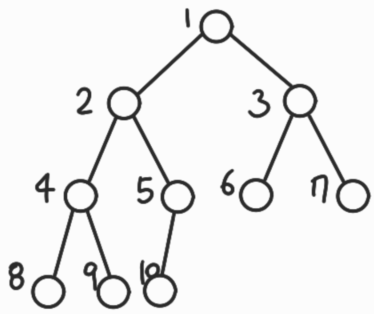
1) 높이 h인 BT에서 level h-1 까지는 FBT이다.
2) 마지막 level h 에는 좌→우로 빈틈없이 leaf들이 채워진다.

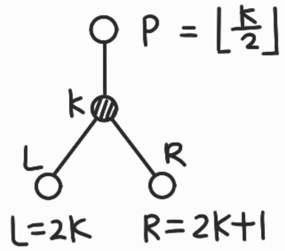


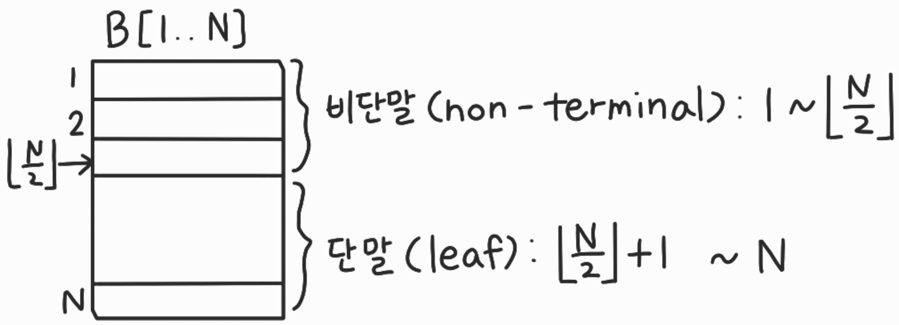
$$2^h - 1 \le N < 2^{h+1} - 1$$
$$2^h \le N < 2^{h+1}$$
$$h \le \lg N < h+1$$
$$h = \lfloor \lg N \rfloor \approx \lg N$$

③ 완전이진트리 (CBT)의 구현

1차원 배열 B[1...N]에 구현 가능 → link 불필요

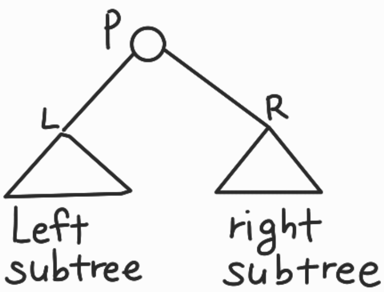






④ 이진트리의 순회 (Traversal)

: 재귀적 방문 (stack이 사용됨)

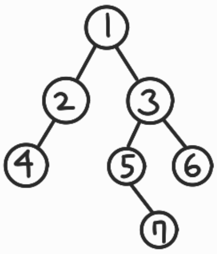


- 전위 순회 (pre-order tr) : P - L - R
- 중위 " (in-order tr) : L - P - R
- 후위 " (post-order tr) : L - R - P
- 레벨 " (level-order tr) : 비재귀적 (Q)

```
preorder (Node *p)
{   if(p ≠ NULL)
    {  print p→Data
       preorder (p→Left)
       preorder (p→Right)
    }
}
```

```
inorder (Node *p)
{   if(p ≠ NULL)
    {  inorder (p→Left)
       print p→Data
       inorder (p→Right)
    }
}
```

ex)



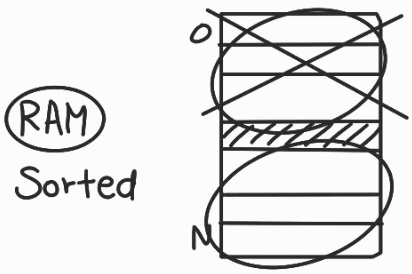
preorder (PLR): 1-2-4-3-5-7-6

inorder (LPR): 4-2-1-5-7-3-6

postorder (LRP): 4-2-7-5-6-3-1

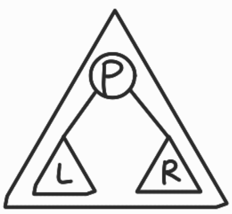
levelorder : 1-2-3-4-5-6-7
(Q가 사용됨)

⑤ 이진 탐색 트리 (Binary Search Tree : BST)



$N \quad \frac{N}{2} \quad \frac{N}{2^2} \quad 1 \; \cdots \; 1$

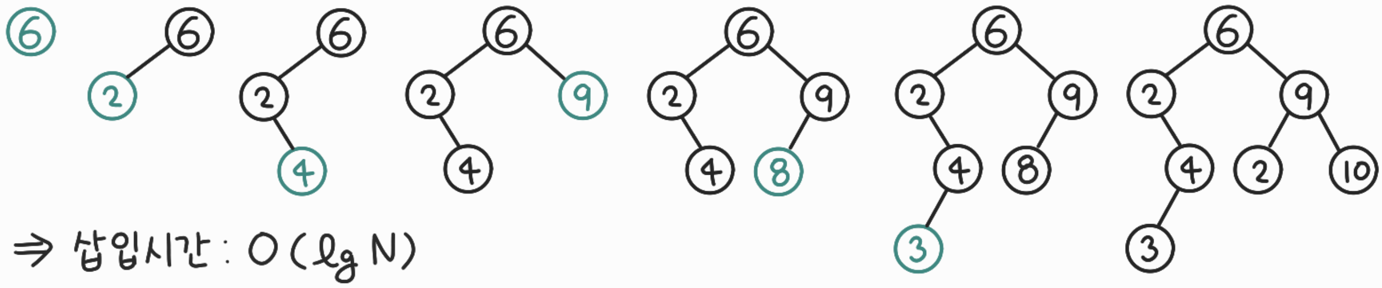
$\lg N$



규칙 : $L \leq P \leq R$: BST

$P \geq L, R$: Heap

ex) input : 6 - 2 - 4 - 9 - 8 - 3 - 10



⇒ 삽입시간 : $O(\lg N)$

생성시간 : $O(N \lg N)$

검색시간 : $O(\lg N)$ → 이진 검색

1) 검색

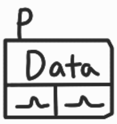
검색중인 node

찾는 data

← $O(\lg N)$

```
Search (Node *p, Data k)
{   if (p == NULL) ret NULL        // not found !!
    if (k == p→data) ret p         // found
    else if (k < p→data) ret Search (p→Left, k)
    else   ret   Search (p→Right, k)
}
```

2) 삽입




```
if (Root == NULL)  Root = p
else  Insert (Root, p)
```

방문 node (≠NULL)

삽입할 node

```
Insert (Node *X, P)
{   if (p→data < X→data)   // 왼쪽에 삽입
    {  if (X→Left = NULL)     X→Left = p
       else    Insert (X→Left, p)
    }
    else   // 오른쪽 삽입 예정
    {  if (X→Right = NULL)  X→Right = p
       else  Insert (X→Right, p)
    }
}
```

3) 삭제

삭제할 node의 3가지 경우

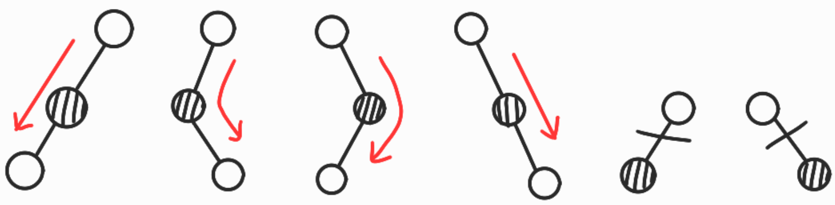
i) leaf node ⎫
ii) 1개 자식 node ⎭ 간단

iii) 2 " — 복잡

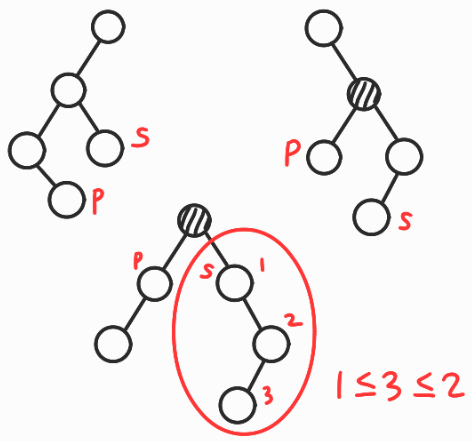
i) leaf node

i) leaf node

ii) 1개의 자식 node



iii) 2개 자식 node (복잡)

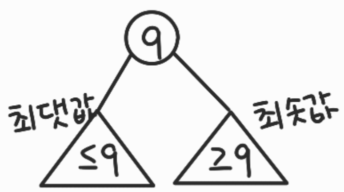


의 후속자(successor : 다음 원소)를

에 복사한 후 그 후속자를 삭제

↓

case ⓘⓘⓘ 뿐



작업	평균	최악
검색 :	$O(\lg N)$	$O(N)$
삽입 :	$O(\lg N)$	$O(N)$
삭제 :	$O(\lg N)$	$O(N)$

↓

균형이진탐색트리 (Balanced BST)가 필요

(ch 12)

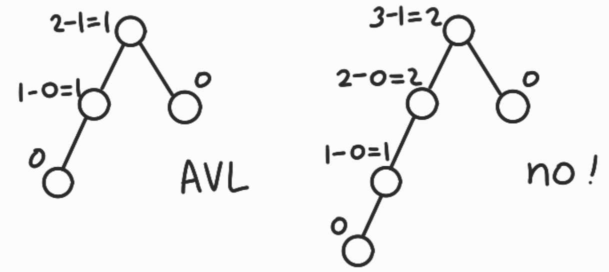
⑥ AVL Tree (Adelson-Velskii & Landis tree, 1962)

: 모든 node의 좌우 subtree의 높이차이가 1 이내인 균형이진탐색트리 (Balanced BST)로서 검색, 삽입, 삭제가 모두 $O(\lg N)$에 가능

· 균형인수 (balance factor) = 좌우 높이차

= 왼쪽 subtree의 높이 − 오른쪽 subtree의 높이 = −1, 0, 1

e.g.

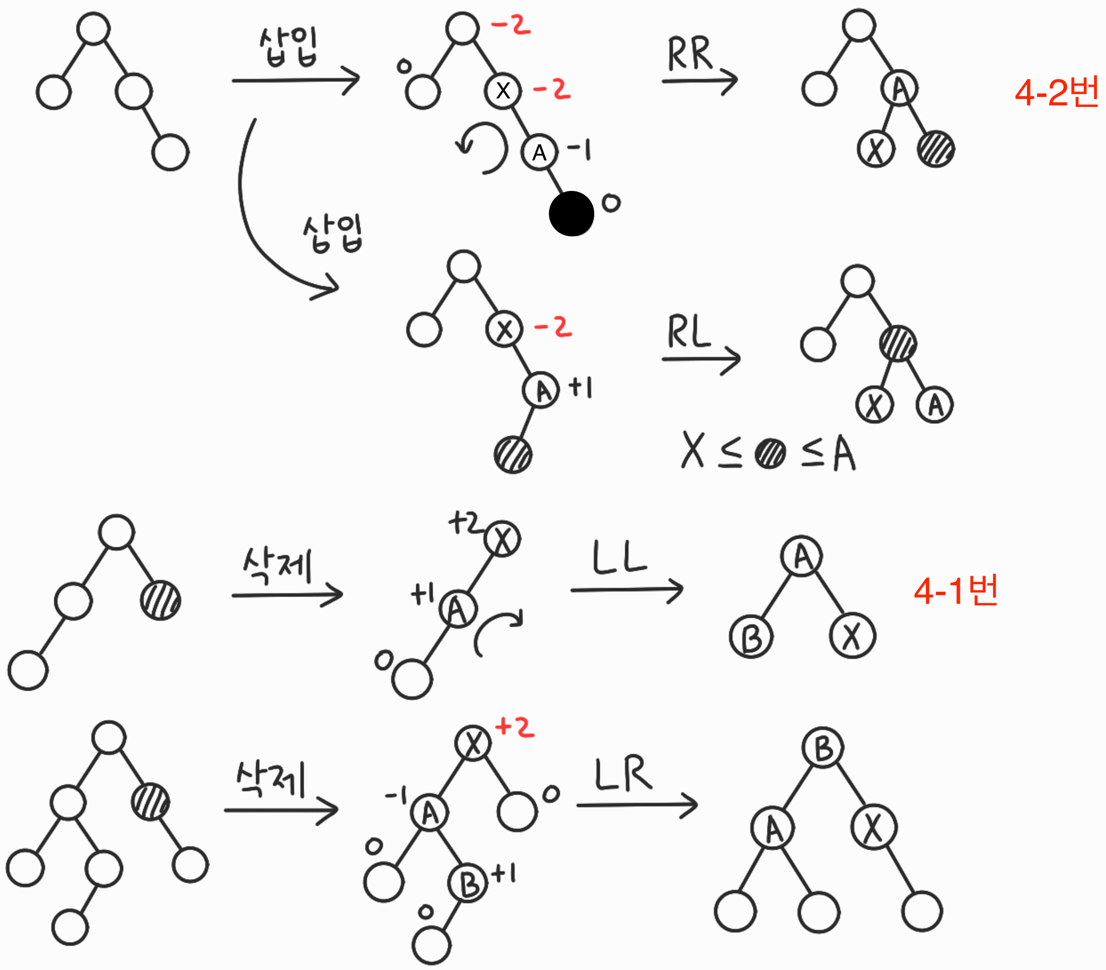


㊉ 초기 AVL은 삽입/삭제 후 균형이 깨질 수 있다.

⇒ 4가지의 rotation으로 rebalancing (재균형) 한다.

(LL, LR, RL, RR)

e.g.



→ 70 72

Red/Black Tree

B-tree

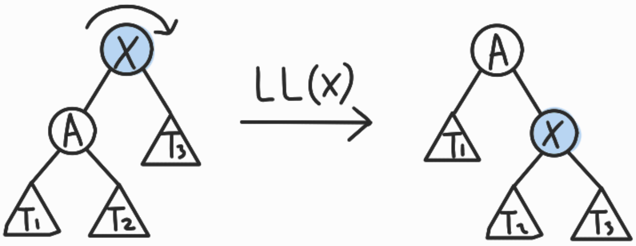
2-3 tree

2-3-4 tree

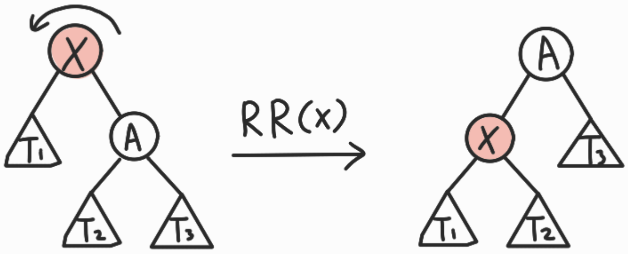
ch12

4 Rotations

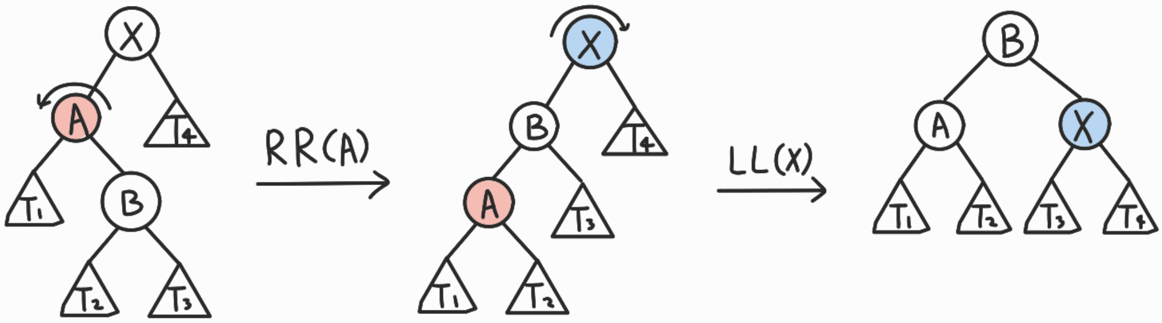
① LL (X)



② RR(X)



③ LR(X) { RR (X→L); LL(X)}



④ RL(X) { LL (X→R); RR(X)}