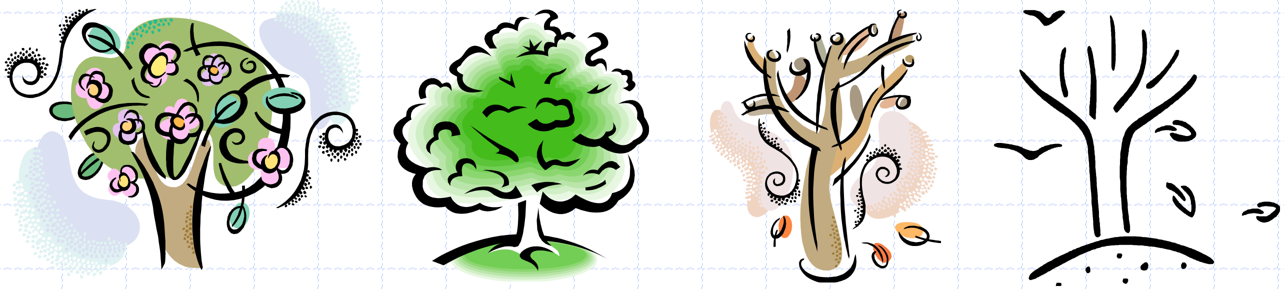
트리

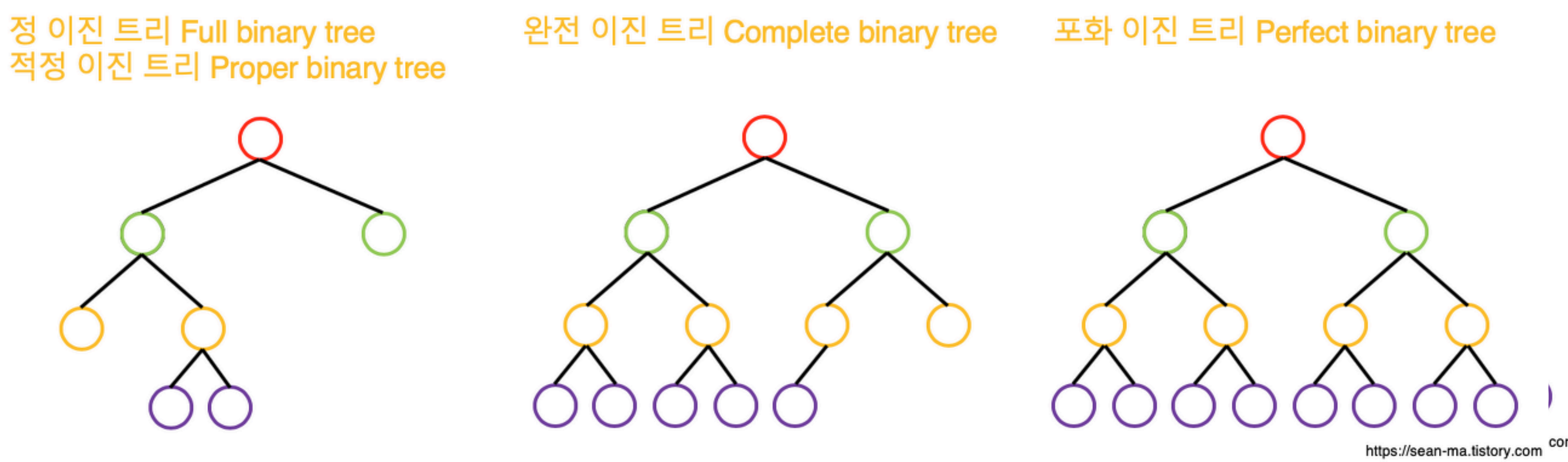


이진트리의 종류

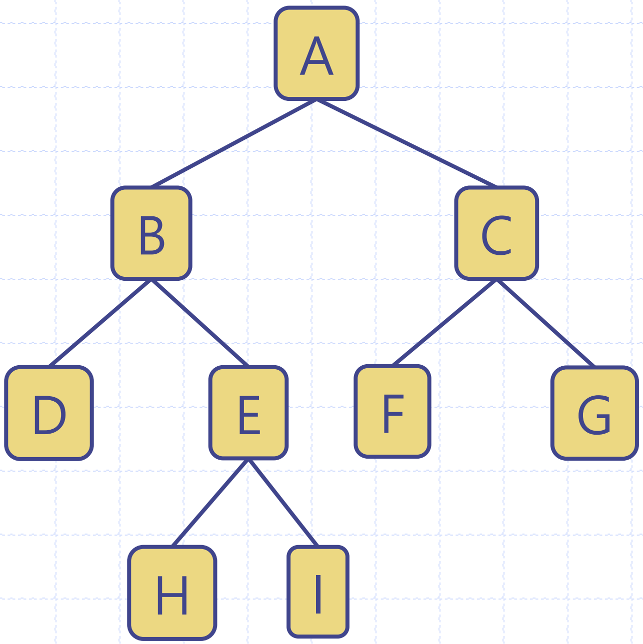
***적정 이진 트리* (Proper binary tree)**는 각 내부 노드가 두 개의 자식 노드를 갖는 순서화된 트리입니다. (홀수 개의 자식 노드를 가질 수 없습니다.)

***완전 이진 트리* (Complete binary tree)** 는 부모, 왼쪽 자식, 오른쪽 자식 순으로 채워지는 트리를 말하며, 마지막 레벨을 제외하고 모든 노드가 가득 차 있어야 합니다. 또한, 마지막 레벨의 노드도 왼쪽으로 몰려 있어야 합니다. (중간에 빈 곳이 없어야 합니다.)

***포화 이진 트리* (Perfect binary tree)** 는 모든 리프 노드의 레벨이 동일하고 모든 레벨이 가득 채워져 있는 이진 트리를 의미합니다.



적정이진트리



Alg ***insertNode***(e)
// input element, left node, right node
// output new node

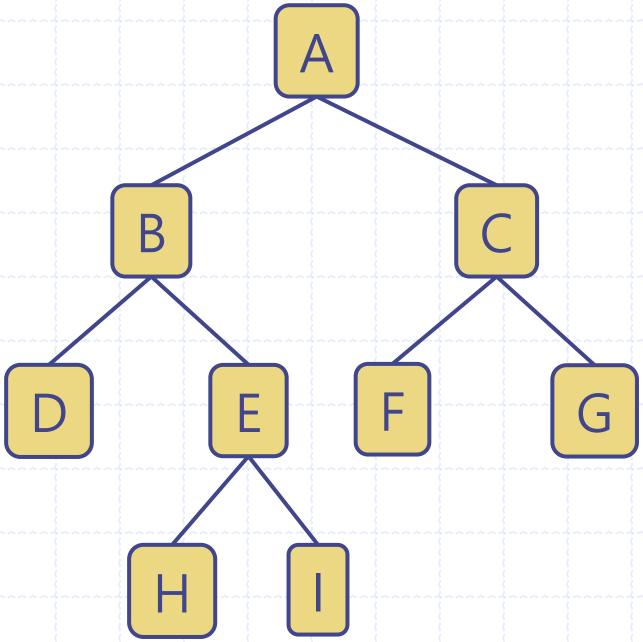
newNode ← *elem* = *e*
newNode ← *left* = *left*
newNode ← *right* = *right*

Alg ***treeBuild***()
// output rootNode

n1 ← ***insertNode***(***'H',NULL,NULL***)
n2 ← ***insertNode***(***'I',NULL,NULL***)
n3 ← ***insertNode***(***'E',n1,n2***)
n4 ← ***insertNode***(***'D',NULL,NULL***)
n5 ← ***insertNode***(***'B',n4,n3***)
n6 ← ***insertNode***(***'F',NULL,NULL***)
n7 ← ***insertNode***(***'G',NULL,NULL***)
n8 ← ***insertNode***(***'C',n6,n7***)
root ← ***insertNode***(***'A',n5,n8***)

return ***root***

적정이진트리



Alg *leftChild(v)*
1. **return** v.left

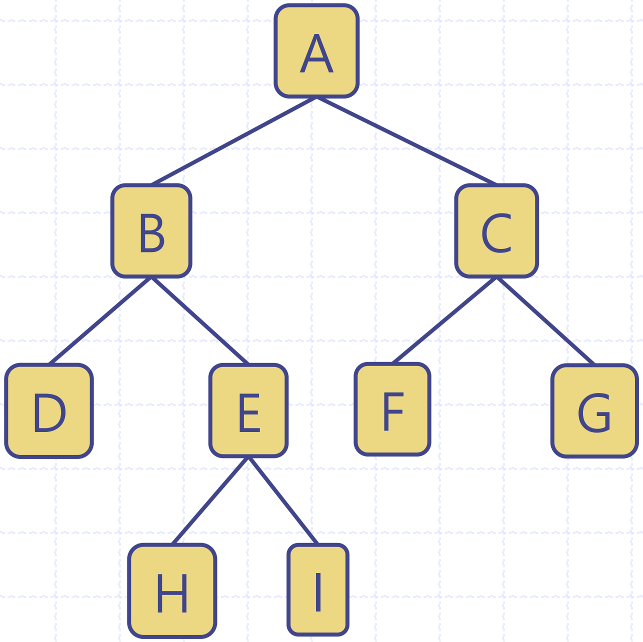
Alg *rightChild(v)*
1. **return** v.right

Alg *isInternal(v)*
1. **return** (v.left $\neq \emptyset$) & (v.right $\neq \emptyset$)

Alg *isExternal(v)*
1. **return** (v.left $= \emptyset$) & (v.right $= \emptyset$)

실습 1

적정이진트리

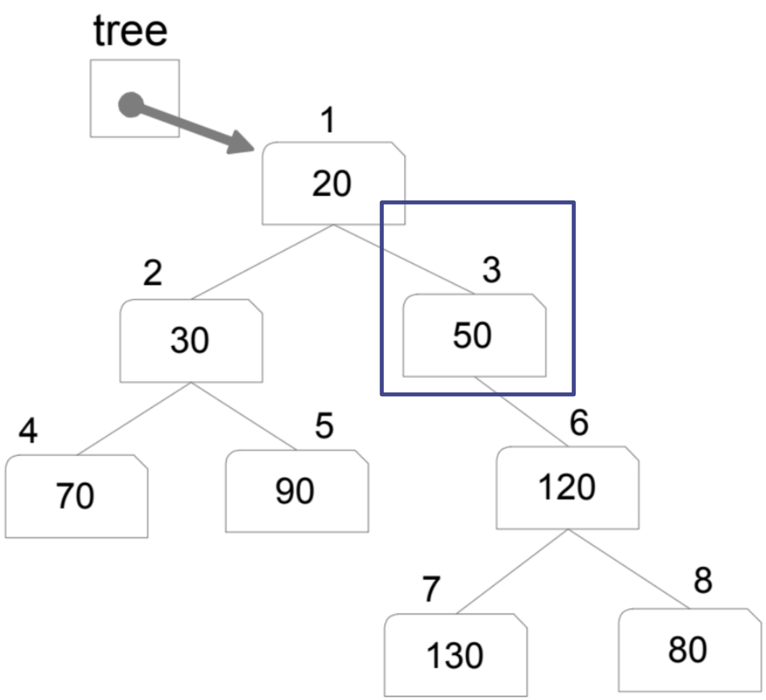


Alg *binaryPreOrder(v)*
1. *visit*(*v*)
2. **if** (*isInternal*(*v*))
    binaryPreOrder(*leftChild*(*v*))
    binaryPreOrder(*rightChild*(*v*))

Alg *binaryPostOrder(v)*
1. **if** (*isInternal*(*v*))
    binaryPostOrder(*leftChild*(*v*))
    binaryPostOrder(*rightChild*(*v*))
2. *visit*(*v*)

Alg *inOrder(v)*
1. **if** (*isInternal*(*v*))
    inOrder(*leftChild*(*v*))
2. *visit*(*v*)
3. **if** (*isInternal*(*v*))
    inOrder(*rightChild*(*v*))

이진트리(적정트리x)



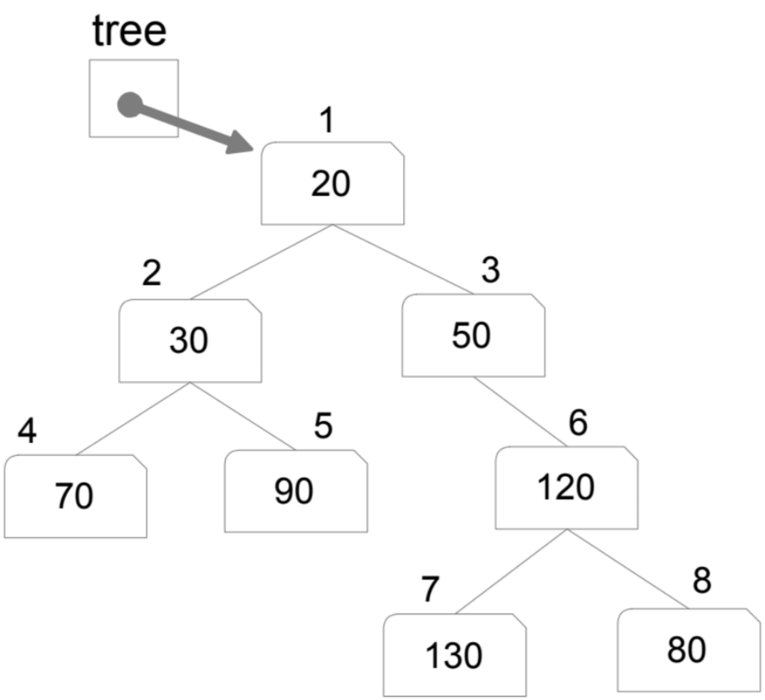
Alg ***insertNode***(id, e)
// input id, e, left node, right node
// output new node
newNode ← id = id
newNode ← elem = e
newNode ← left = left
newNode ← right = right

Alg ***treeBuild***()
// output rootNode

n7 ← ***insertNode***(***7,130,NULL,NULL***)
n8 ← ***insertNode***(***8,80,NULL,NULL***)
n6 ← ***insertNode***(***6,120,n7,n8***)
n4 ← ***insertNode***(***4,70,NULL,NULL***)
n5 ← ***insertNode***(***5,90,NULL,NULL***)
n3 ← ***insertNode***(***3,50,NULL,n6***)
n2 ← ***insertNode***(***2,30,n4,n5***)
n1 ← ***insertNode***(***1,20,n2,n3***)

return *n1*

이진트리(적정트리x)



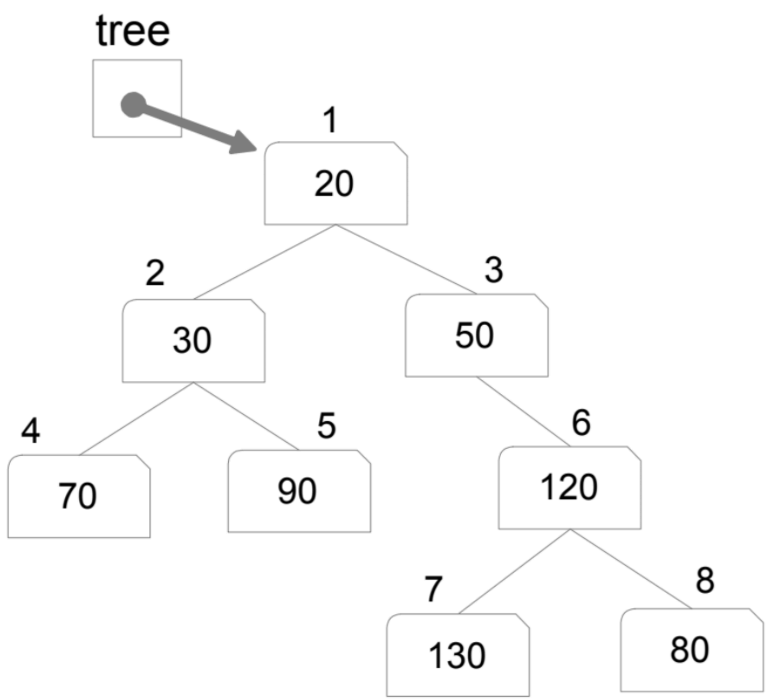
Alg *binaryPreOrder(v)*
if (*v!=NULL*)
  visit(v)
  binaryPreOrder(leftChild(v))
  binaryPreOrder(rightChild(v))

Alg *binaryPostOrder(v)*
if (*v!=NULL*)
  binaryPostOrder(leftChild(v))
  binaryPostOrder(rightChild(v))
  visit(v)

Alg *inOrder(v)*
if (*v!=NULL*)
  inOrder(leftChild(v))
  visit(v)
  inOrder(rightChild(v))

이진트리(적정트리x)

Preorder 스타일로
순회하며 ID 검사



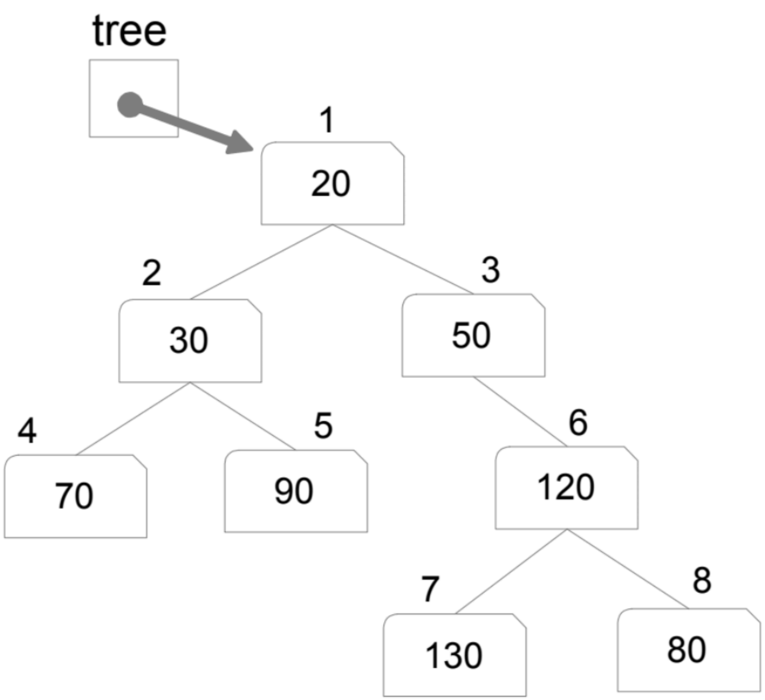

```
Alg binaryPreOrder(v)
    if (v!=NULL)
        visit(v)
        binaryPreOrder(leftChild(v))
        binaryPreOrder(rightChild(v))


Alg findID(v)
    if(v != NULL)
        if(v.id = id) return v
        p ← findID(leftChild(v))
        if (p!=NULL) return p
        p ← findID(rightChild(v))
        if (p!=NULL) return p

1.  return NULL
```

이진트리(적정트리x)

postorder 스타일로
순회하며 디스크용량 누적



Alg *binaryPostOrder(v)*
1. **if** (*v!=NULL*)
   binaryPostOrder(leftChild(v))
   binaryPostOrder(rightChild(v))
2. *visit_sum(v)*

Alg visit_sum(v)

1. gFolderSize += v.size

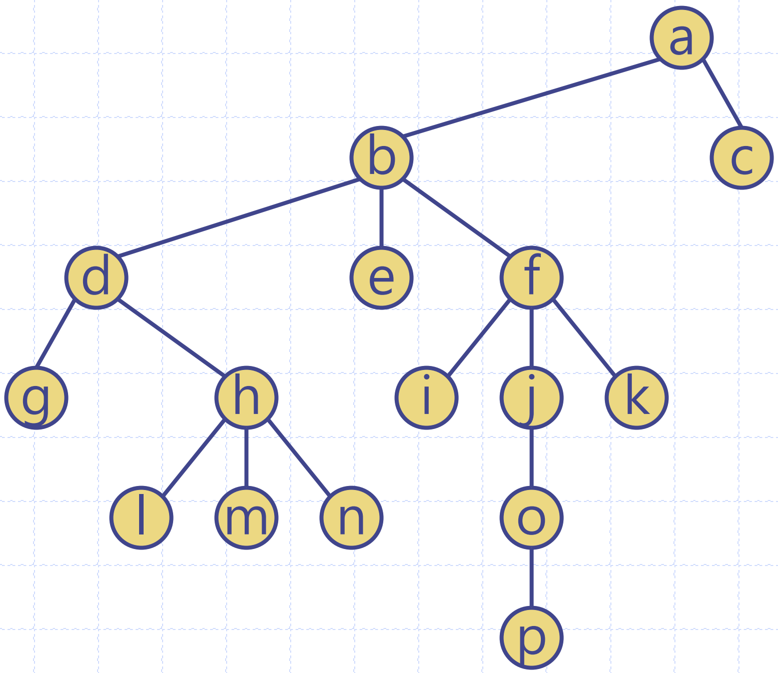
트리의 경로 길이

- 트리 T의 **경로길이**란 T의 모든 노드들의 깊이의 합을 말한다
- 일반(generic) 트리 T의 **경로길이**를 구하기 위한 선형시간 알고리즘을 작성하라
  - pathLength(v): 루트가 v인 트리의 경로길이를 반환
- **힌트:** depth를 반복적으로 사용하면 선형시간 조건을 만족하기 어렵다

Alg ***pathLength(v)***
1. **return** ***rPathLength***(*v*, 0)

Alg ***rPathLength(v, d)***
1. **if** (***isExternal***(*v*))
   return ***d***
2. ***sum*** ← ***d***
3. **for each** ***w*** ∈ ***children***(*v*)
   sum ← ***sum*** + ***rPathLength***(*w*, ***d*** + 1)
4. **return** ***sum***

트리의 경로 길이



Alg *pathLength(v)*
1. **return** ***rPathLength***$(v, 0)$

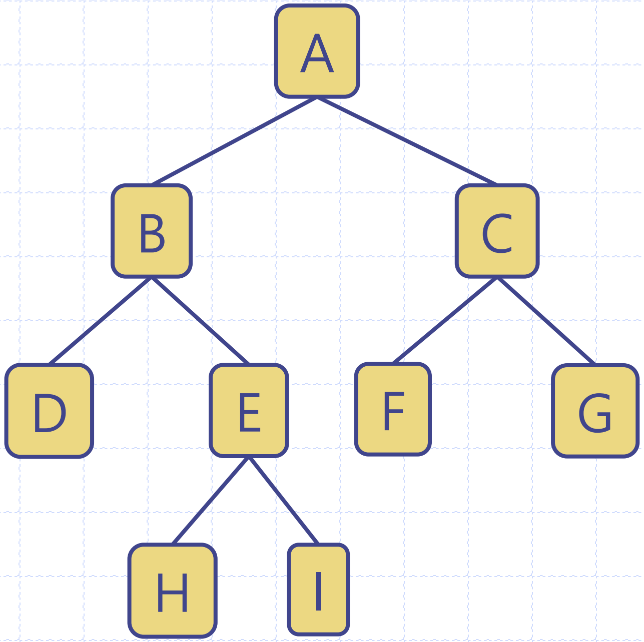
Alg *rPathLength(v, d)*
1. **if** (***isExternal***(v))
   return d
2. $sum \leftarrow d$
3. **for each** $w \in children(v)$
   $sum \leftarrow sum + rPathLength(w, d + 1)$
4. **return** sum

A. 경로길이는 44
B. 내부경로길이는 15
C. 외부경로길이는 29

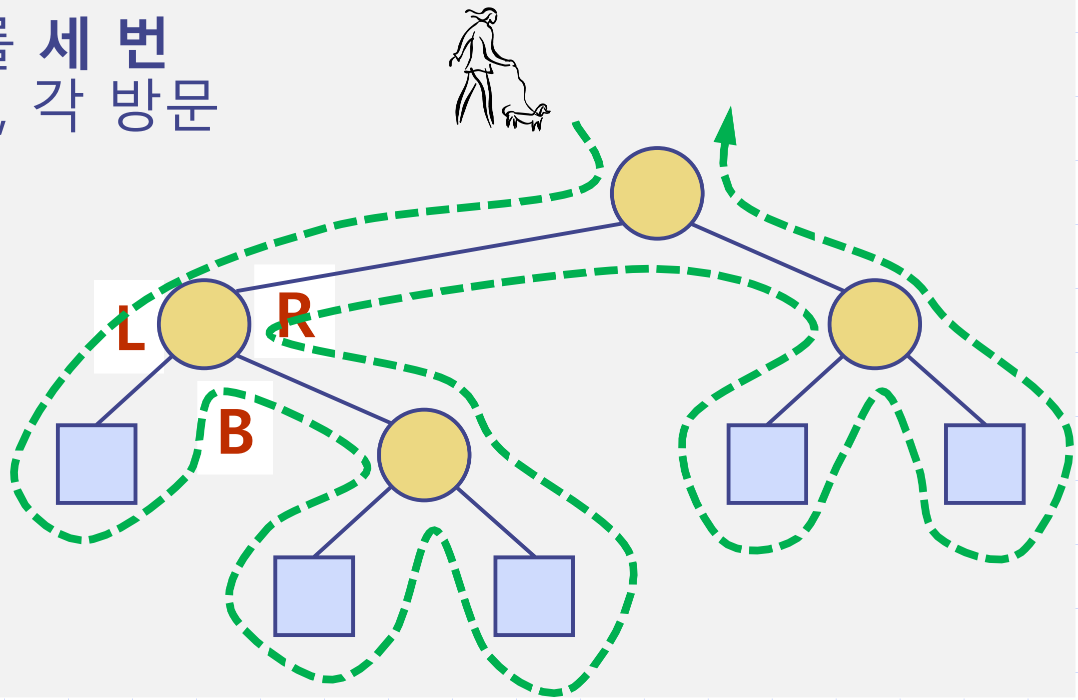
계승자

- 이진트리 T의 노드 v의,
  - **선위순회 계승자**(preorder successor)란 T의 선위순회에서 v 직후에 방문되는 노드를 말한다
  - **중위순회 계승자**(inorder successor)란 T의 중위순회에서 v 직후에 방문되는 노드를 말한다
  - **후위순회 계승자**(postorder successor)란 T의 후위순회에서 v 직후에 방문되는 노드를 말한다
- **주의:** 마지막 방문노드의 계승자는 존재하지 않는다

- 아래 트리에서 다음을 구하라
  - 노드 I의 선위순회 계승자
  - 노드 A의 중위순회 계승자
  - 노드 C의 후위순회 계승자

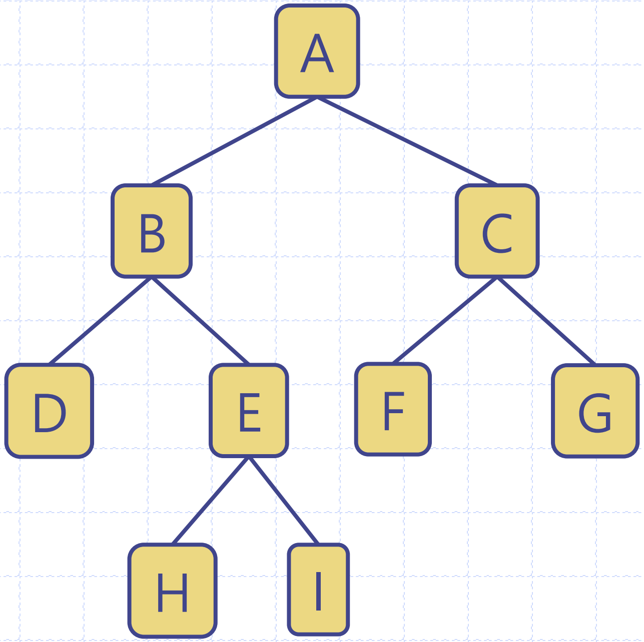


- 그러면 각 노드를 **세 번** 방문하게 되는데, 각 방문 위치는 노드의:
  - 왼쪽에서 (**L**)
  - 아래에서 (**B**)
  - 오른쪽에서 (**R**)

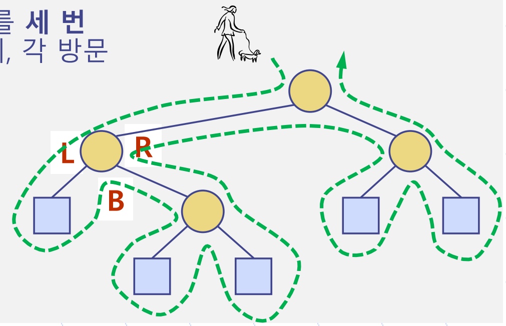


답

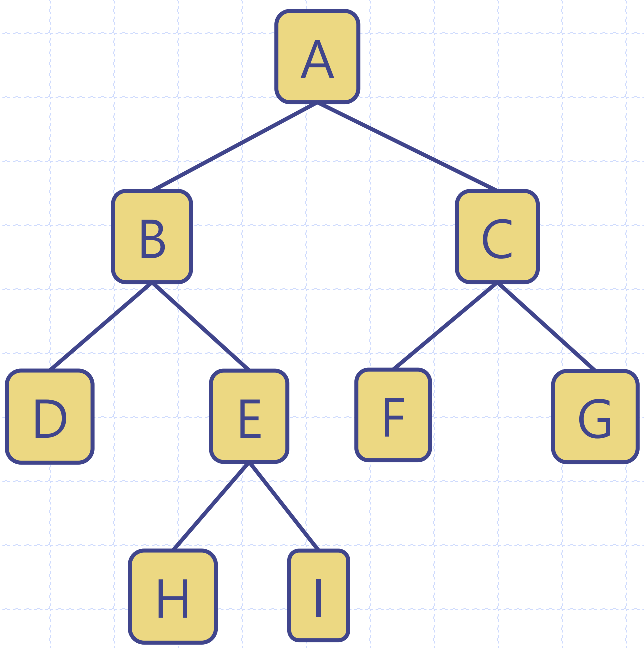
- 노드 I의 선위순회 계승자는 C
- 노드 A의 중위순회 계승자는 F
- 노드 C의 후위순회 계승자는 A



- 그러면 각 노드를 **세 번** 방문하게 되는데, 각 방문 위치는 노드의:
  - 왼쪽에서 (**L**)
  - 아래에서 (**B**)
  - 오른쪽에서 (**R**)



- **선위순회 계승자**(preorder successor)란 T의 선위순회에서 v 직후에 방문되는 노드를 말한다
- **왼쪽에서 (L)**



Alg ***preOrderSucc*(*v*)**

  input node ***v*, output** node

// case1: v가 내부 노드이고, 왼쪽 탐색이 남은 경우

1. **if** (***isInternal*(*v*)**)

    return ***leftChild*(*v*)**

2. ***p*** ← ***parent*(*v*)**

// case2: v가 외부노드이나, 본인의 부모 노드의 오른쪽 자식 탐색이 남은 경우 (while 스킵)

// case3: v가 외부노드이나 본인이 부모노드의 오른쪽 자식인 경우 조상 중 오른쪽 탐색이 남은 노드를 탐색

3. **while** (***leftChild*(*p*)** ≠ ***v***)

    if (***isRoot*(*p*)**) // 오른쪽 탐색이 남은 경우가 없는 경우

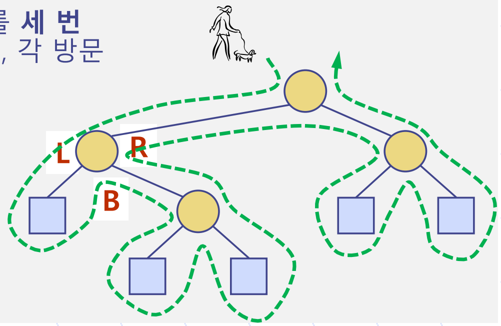
      ***invalidNodeException*()**

    v ← ***p***

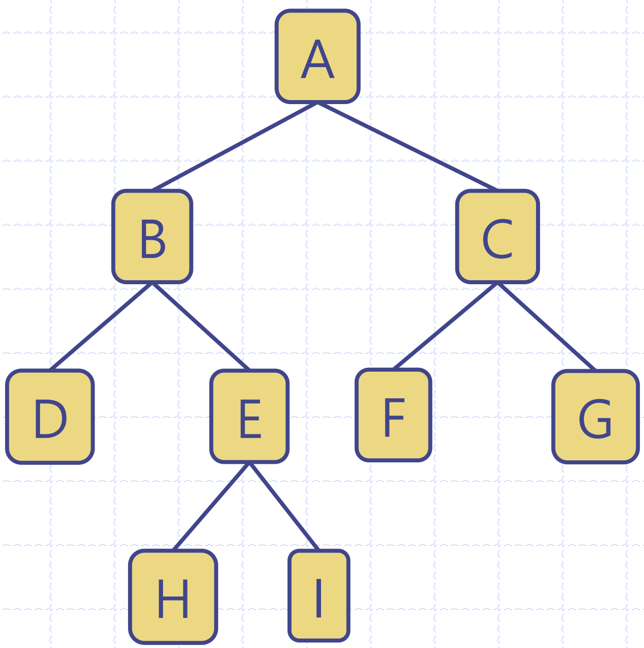
    p ← ***parent*(*p*)**

4. **return** ***rightChild*(*p*)**

◆ 그러면 각 노드를 **세 번** 방문하게 되는데, 각 방문 위치는 노드의:
- 왼쪽에서 (**L**)
- 아래에서 (**B**)
- 오른쪽에서 (**R**)



- **후위순회 계승자**(postorder successor)란 *T*의 후위순회에서 *v* 직후에 방문되는 노드를 말한다
- 오른쪽에서 (R)



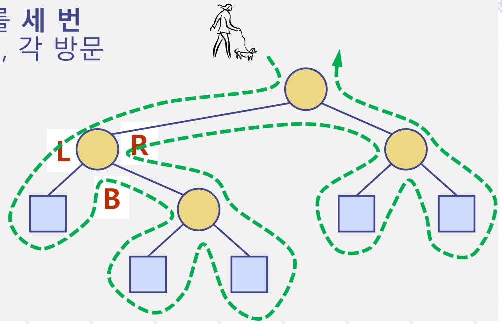
Alg ***postOrderSucc(v)***
    input node ***v***
    output node

// 후위순회에서 root는 최종 순회임
1. **if** (***isRoot(v)***)
        invalidNodeException()

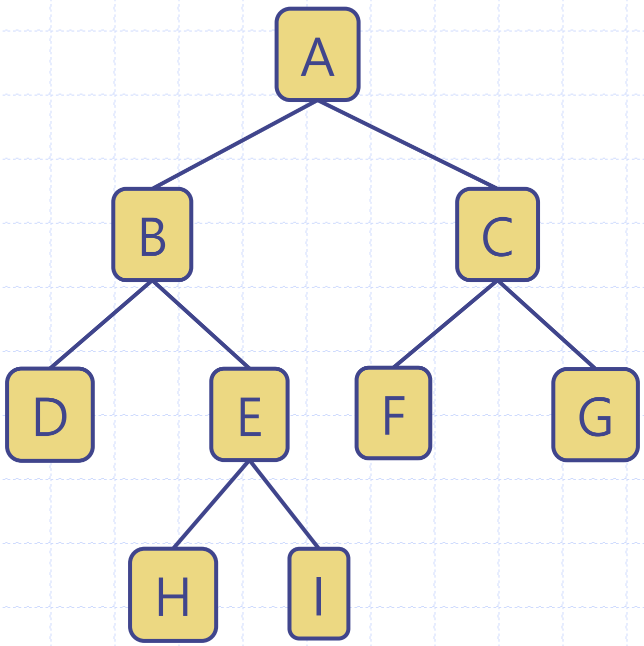
2. ***p*** ← ***parent(v)***
// R-Node (Node G/NodeE) – External (RR/LR)
3. **if** (***rightChild(p)*** = ***v***)
        return ***p***
// L-Node (Node F) – 오른쪽형제노드 External
4.   ***v*** ← ***rightChild(p)***

// L-Node (Node D) – 오른쪽형제노드 Internal
5. **while** (!***isExternal(v)***)
        v ← ***leftChild(v)***
6. **return** ***v***

- 그러면 각 노드를 **세 번** 방문하게 되는데, 각 방문 위치는 노드의:
  - 왼쪽에서 (**L**)
  - 아래에서 (**B**)
  - 오른쪽에서 (**R**)



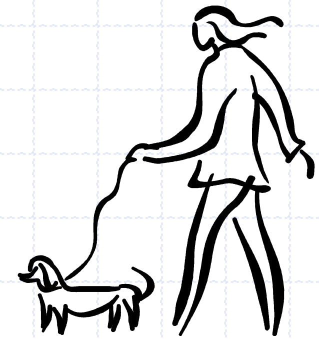
- **중위순회 계승자**(inorder successor)란 *T*의 중위순회에서 *v* 직후에 방문되는 노드를 말한다
- 아래에서 **(B)**



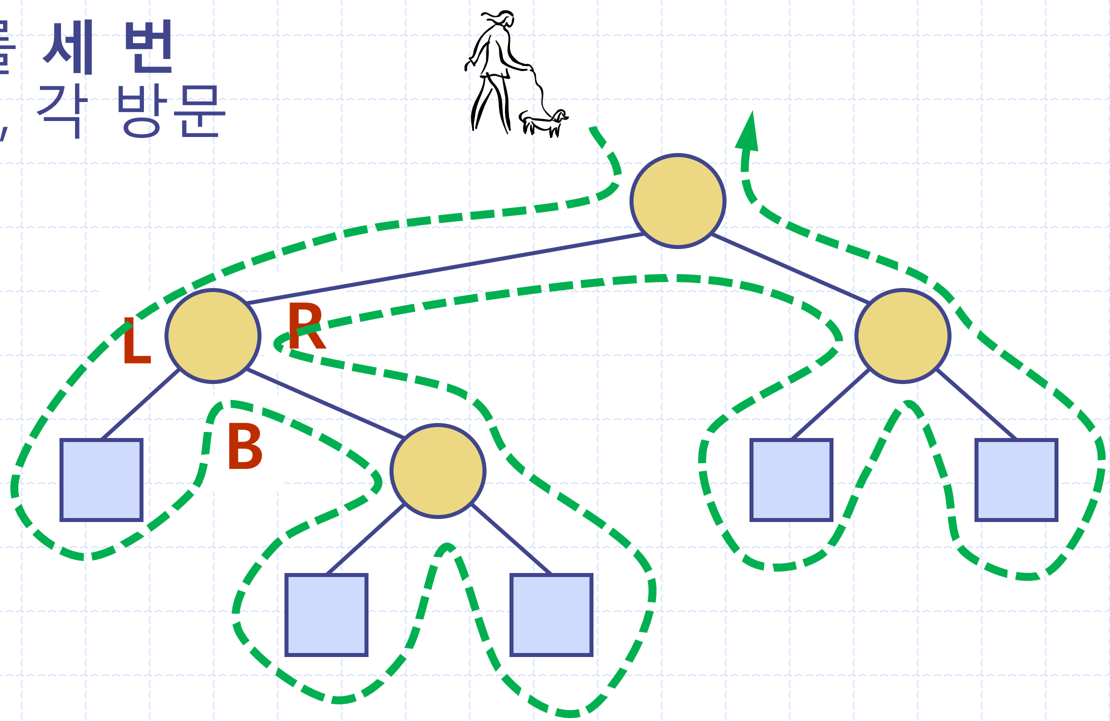

```
Alg inOrderSucc(v)
    input node v
    output node

// Node B, Node C, Node E
1. if (isInternal(v))
        v ← rightChild(v)
        while (isInternal(v))
            v ← leftChild(v)
        return v
// External – Node D, F, H  / I, G
2. p ← parent(v)
3. while (leftChild(p) ≠ v)
        if (isRoot(p)) // Node G
            invalidNodeException()
        // Node I
        v ← p
        p ← parent(p)
4. return p
```

오일러 투어 순회



- **오일러 투어**(Euler Tour): 이진트리에 대한 **일반순회**(generic traversal)
- 왼쪽 자식 방향으로 루트를 출발하여, 트리의 간선들을 항상 왼쪽 벽으로 두면서 트리 주위를 걷는다
- 그러면 각 노드를 **세 번** 방문하게 되는데, 각 방문 위치는 노드의:
  - 왼쪽에서 (**L**)
  - 아래에서 (**B**)
  - 오른쪽에서 (**R**)

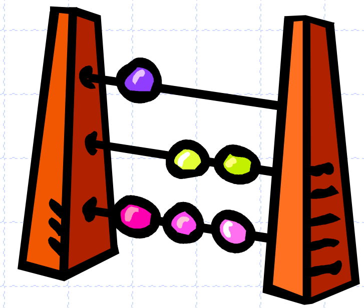


오일러 투어 순회 (conti.)

- **선위**, **중위**, **후위** 순회를 모두 포함한다
- 각 노드를 **세 번** 방문하므로 위 셋 중 한 가지 순회만으로는 성취하기 어려운 작업 수행 가능
- **응용**
  - 이진트리내 각 부트리의 노드 수 계산

Alg *eulerTour(v)*
1. ***visitLeft***(*v*) {preorder}
2. **if** (***isInternal***(*v*))
   eulerTour(***leftChild***(*v*))
3. ***visitBelow***(*v*) {inorder}
4. **if** (***isInternal***(*v*))
   eulerTour(***rightChild***(*v*))
5. ***visitRight***(*v*) {postorder}

예: 부트리들의 크기



- 카운터 k를 0으로 초기화한 후 **오일러 투어**를 시작
- 노드를 왼쪽에서 방문할 때마다 k를 하나씩 증가
- 루트가 v인 부트리의 크기는, v를 왼쪽에서 방문했을 때의 k값과 오른쪽에서 방문했을 때의 k값의 차이에 1을 더한 값
- **실행시간: O(n)**

Alg ***findSizeOfSubtrees(v)***
1. $k \leftarrow 0$
2. *eulerTour*(v)

Alg ***visitLeft(v)***
1. $k \leftarrow k + 1$
2. v.kleft $\leftarrow k$

Alg ***visitBelow(v)***
1. **return**

Alg ***visitRight(v)***
1. v.size $\leftarrow k - v$.kleft $+ 1$

작동 원리

k: 각 노드를 왼쪽에서 방문했을 때의 카운터 값

n: 각 부트리의 크기