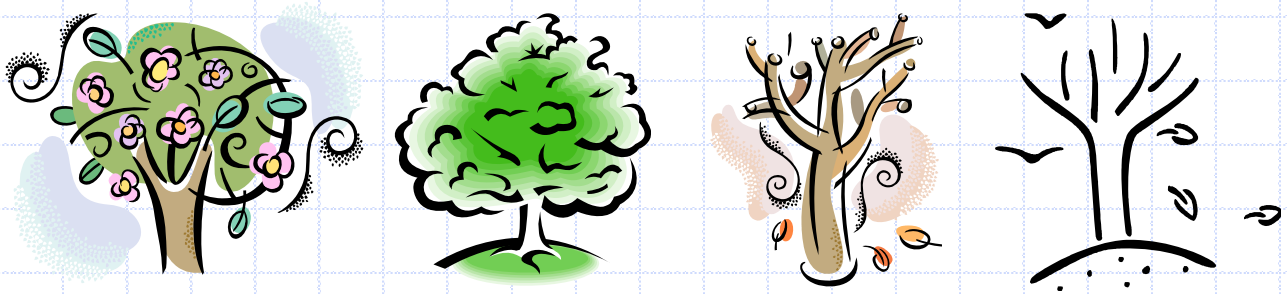
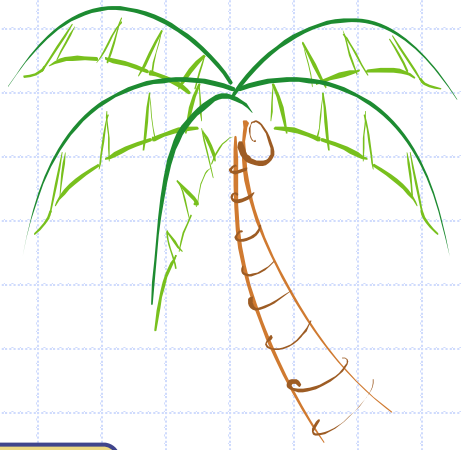
트리



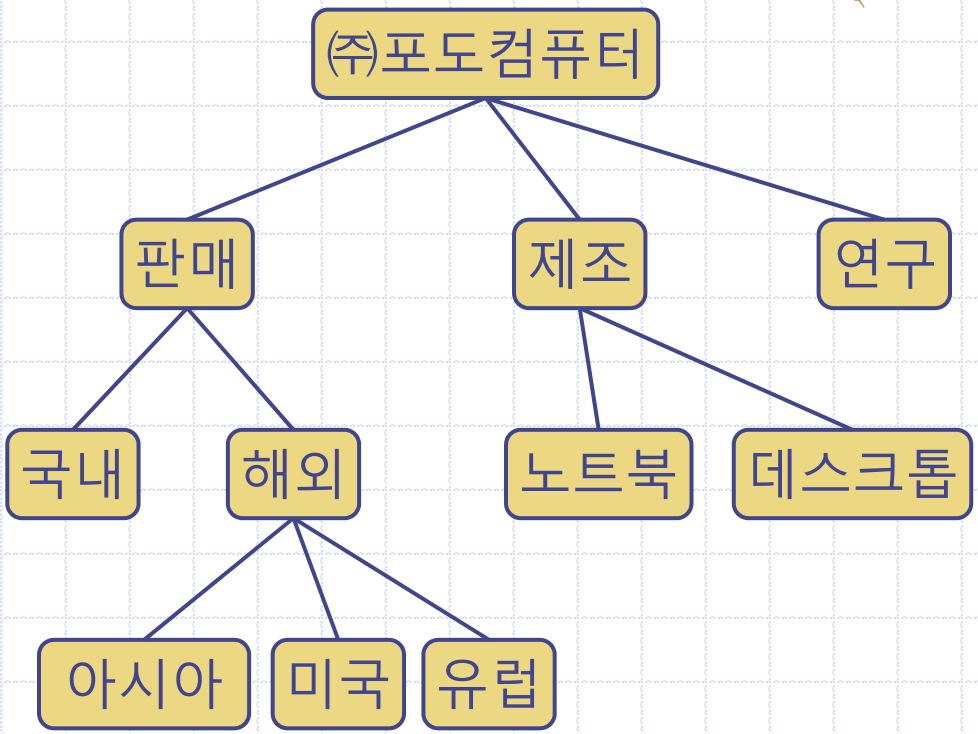
Outline

- 8.1 트리 ADT
- 8.2 트리 용어
- 8.3 트리 ADT 메쏘드
- 8.4 이진트리 ADT
- 8.5 이진트리 ADT 메쏘드
- 8.6 이진트리 ADT 구현과 메쏘드
- 8.7 트리 ADT 구현과 메쏘드
- 8.8 응용문제

트리 ADT

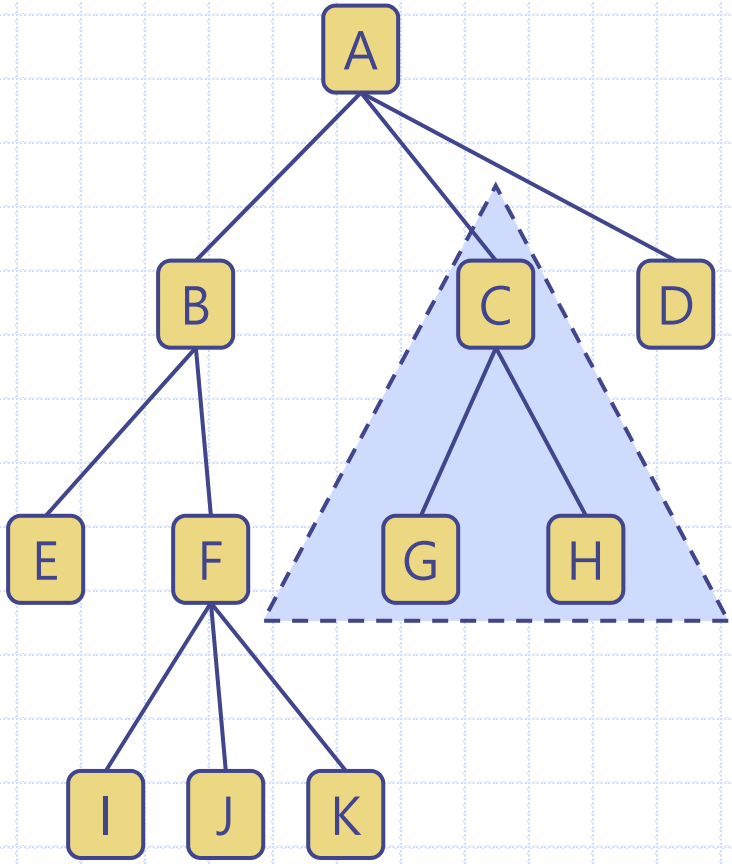


- 트리 ADT는 **계층적**으로 저장된 데이터원소들을 모델링한다
- 맨위의 원소를 제외하고, 각 트리 원소는 **부모**(parent) 원소와 0개 이상의 **자식**(children) 원소들을 가진다
- **전제:** 트리는 비어 있지 않다



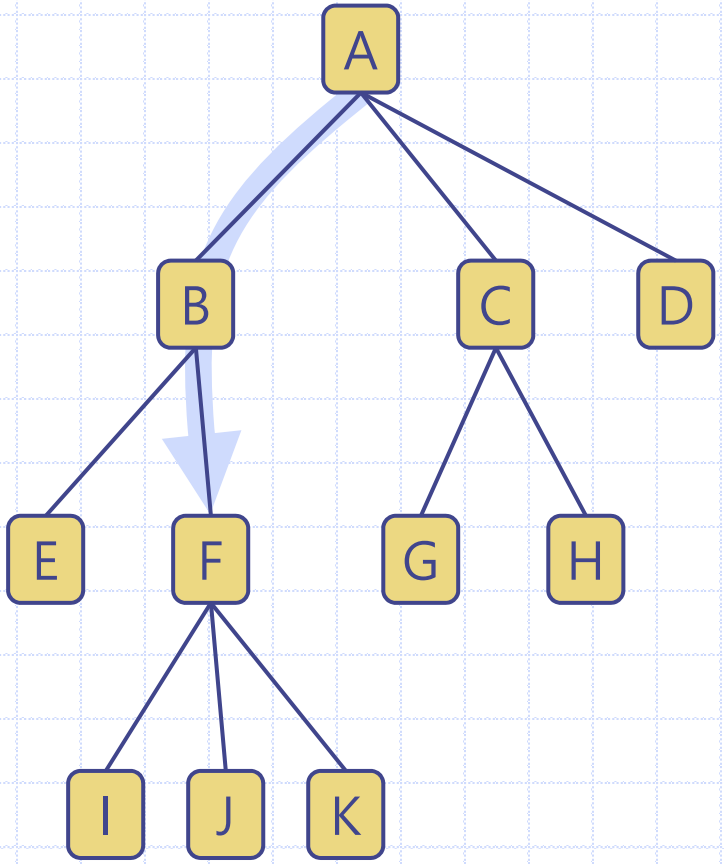
트리 용어

- **루트**(root): 부모가 없는 노드(A)
- **내부노드**(internal node): 적어도 한 개의 자식을 가진 노드(A, B, C, F)
- **외부노드**(external node), 또는 **리프**(leaf): 자식이 없는 노드(E, I, J, K, G, H, D)
- **형제**(siblings): 같은 부모를 가진 노드들(G, H)
- 노드의 **조상**(ancestor): 부모(parent), 조부모(grandparent), 증조부모(grand-grandparent), 등
- 노드의 **자손**(descendant): 자식(child), 손주(grandchild), 증손주(grand-grandchild), 등
- **부트리**(subtree): 노드와 그 노드의 자손들로 구성된 트리



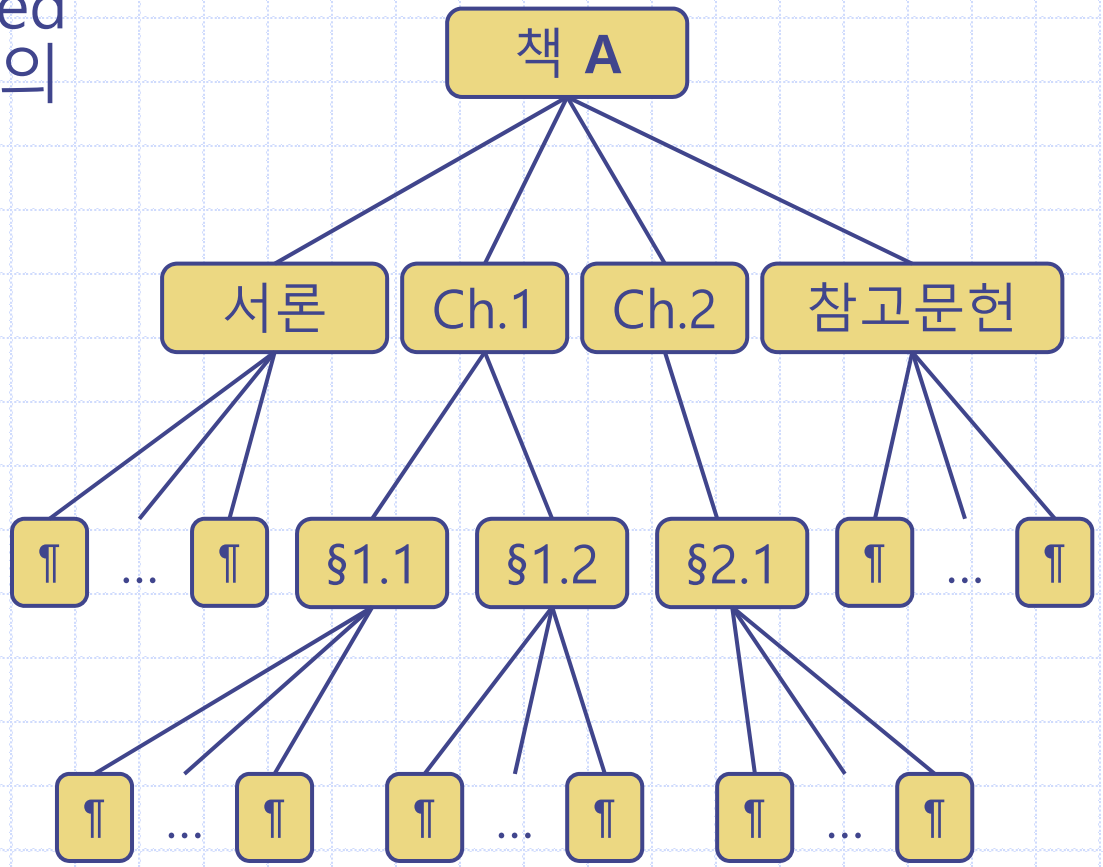
트리 용어 (conti.)

- **경로**(path): 조상 또는 자손을 따라 이어진 노드 시퀀스(**예:** ABF)
- **경로길이**(path length): 경로내 간선(edge)의 수
- 노드의 **깊이**(depth): 루트로부터 노드에 이르는 유일한 경로의 길이
- 노드의 **높이**(height): 노드로부터 외부노드에 이르는 가장 긴 경로의 길이
- **트리의 높이**(height of a tree): 루트의 높이



순서트리

- **순서트리**(ordered tree)는 각 노드의 자식들에 대해 선형 순서가 정의되어 있는 트리를 말한다
- **예:** 구조적 문서



트리 ADT 메쏘드

- 노드를 추상화하기 위해 **위치**를 사용한다
- 일반 메쏘드
  - boolean isEmpty()*
  - integer size()
- 접근 메쏘드
  - position root()
  - position parent(p)
  - position children(p)
  - element element(p)
- 질의 메쏘드
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- 갱신 메쏘드
  - swapElements(p, q)
  - element replaceElement(p, e)
- 예외
  - invalidNodeException(): 불법 노드 접근 시 발령
- 트리 ADT를 구현하는 데이터구조에 따라 추가적인 갱신 메쏘드들(삽입, 삭제 등)이 정의될 수 있다

트리 응용

- 직접 응용
  - 조직구성도
    - 내부노드: 부, 과, 팀 등
    - 외부노드: 직원
  - 파일시스템
    - 내부노드: 폴더(folders 또는 directories)
    - 외부노드: 파일
  - 프로그래밍 환경
    - 내부노드: 프로그램 구조물(programming constructs)
    - 외부노드: 어휘, 상수, 심볼
- 간접 응용
  - 알고리즘을 위한 보조 데이터구조
  - 다른 데이터구조를 구성하는 요소

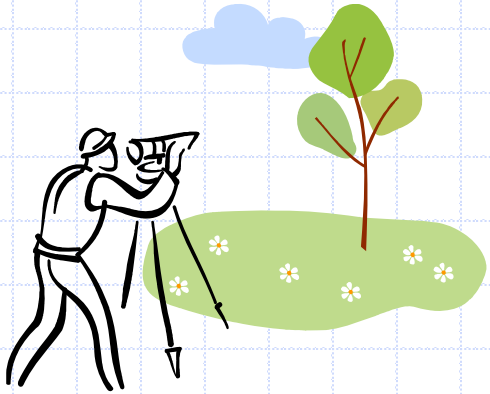
깊이

- 노드 v의 **깊이**(depth)의 재귀적 정의
  - 만약 v가 루트면, v의 깊이는 0
  - 그렇지 않으면, v의 깊이는 v의 부모의 깊이 더하기 1
- 최악의 경우, depth는 $O(n)$ 시간에 수행한다 – 단, n은 트리내 총 노드의 수다

```
Alg depth(v)
1. if (isRoot(v))
       return 0
   else
       return 1 + depth(parent(v))
```

- depth(v)는 일반적(generic)이다 – 즉, 트리의 구현에 독립적이다

높이



- 노드 v의 **높이**(height)의 재귀적 정의
  - 만약 v가 외부노드면, v의 높이는 0
  - 그렇지 않으면, v의 높이는 v의 자식들 중 최대 높이 더하기 1
- 최악의 경우, height는 $O(n)$ 시간에 수행한다 – 단, n은 트리내 총 노드의 수다

```
Alg height(v)
1. if (isExternal(v))
       return 0
   else
       h ← 0
       for each w ∈ children(v)
           h ← max(h, height(w))
       return 1 + h
```

- 루트가 r인 **트리의 높이**(height of a tree)는 height(r)을 호출하여 구할 수 있다

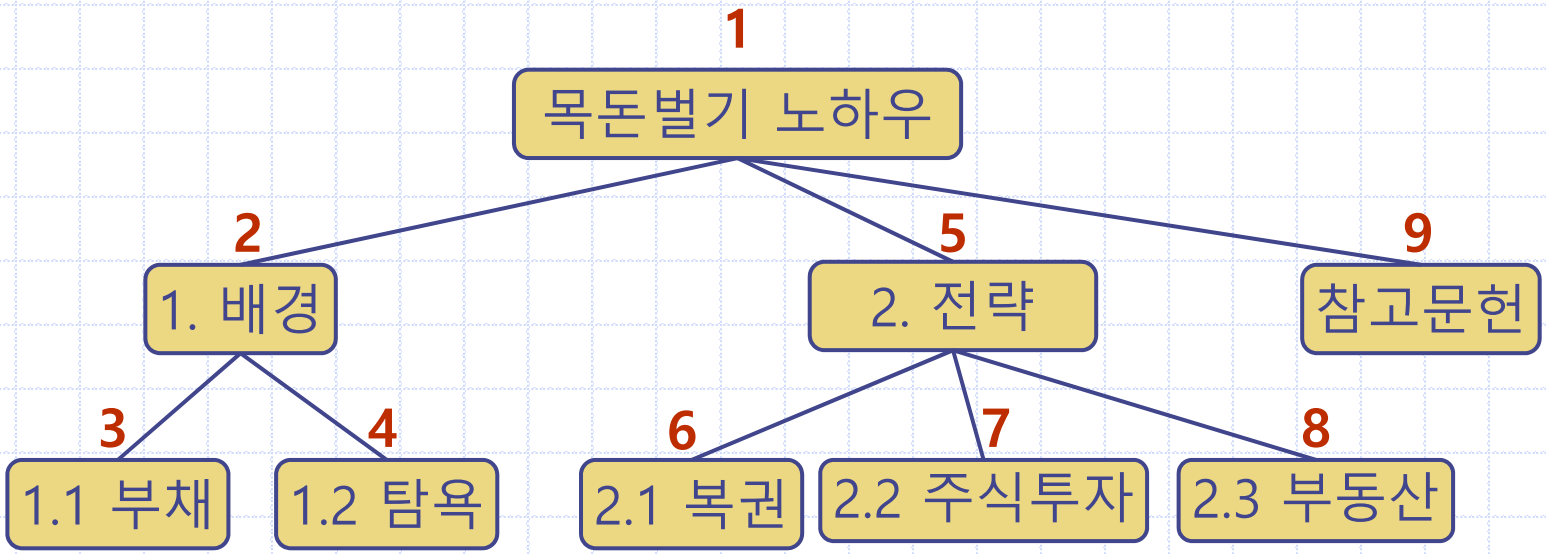
선위순회

- **순회**(traversal)란 트리의 노드들을 체계적인 방식으로 방문하는 것을 말한다
- **선위순회**(preorder traversal)에서는, 노드가 그의 자손들보다 **앞서** 방문된다
- **실행시간:** $O(n)$ – 단, n은 트리내 총 노드의 수
- **응용**
  - 구조적 문서를 인쇄
  - 계층적 파일시스템의 모든 폴더들을 나열

Alg ***preOrder***(v)
1. ***visit***(v)
2. **for each** $w \in$ ***children***(v)
   preOrder(w)

예: 구조적 문서

- 적절한 들여쓰기를 사용하여 구조적 문서의 목차를 인쇄하고자 한다
- **전제:** 문서는 순서트리에 저장되어 있다



예: 구조적 문서 (conti.)

```
Alg printTableOfContents(v)
1. rPrint(v, 0)
2. return

Alg rPrint(v, d)
1. for i ← 1 to d
       write(Tab)
2. write(element(v))
3. for each w ∈ children(v)
       rPrint(w, d + 1)
4. return
```

목돈벌기 노하우
1. 배경
   1.1 부채
   1.2 탐욕
2. 전략
   2.1 복권
   2.2 주식투자
   2.3 부동산

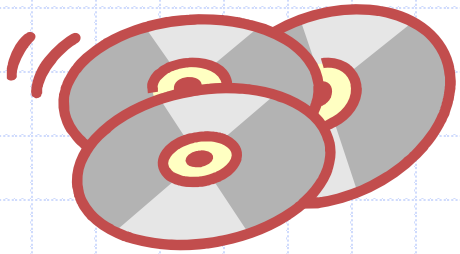
참고문헌

후위순회

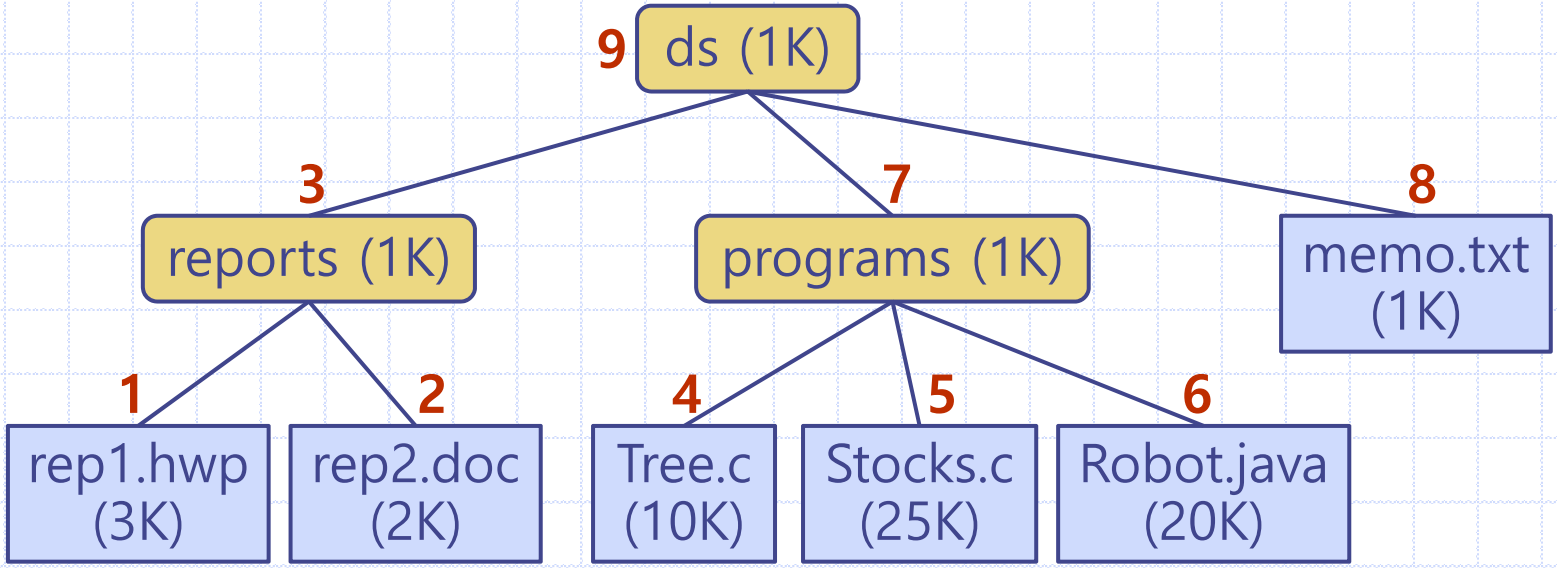
- **후위순회**(postorder traversal)에서는, 노드가 그의 자손들보다 **나중에** 방문된다
- **실행시간:** $O(n)$ – 단, n은 트리내 총 노드의 수
- **응용**
  - 계층적 파일시스템에서 폴더의 디스크 사용량 계산

Alg ***postOrder*(*v*)**
1. **for each** $w \in$ ***children*(*v*)**
   ***postOrder*(*w*)**
2. ***visit*(*v*)**

예: 디스크 사용량



- 폴더의 디스크 사용량을 계산하고자 한다
- 다음 사용량의 재귀적 합을 구해 얻는다
  - 폴더 자체의 사용량 (1KB라고 가정)
  - 폴더내 파일들의 사용량
  - 자식 폴더들의 사용량



예: 디스크 사용량 (conti.)

```
Alg diskUsage(v)
1. sum ← 0
2. for each w ∈ children(v)
       sum ← sum + diskUsage(w)
3. return sum + space(v)
```

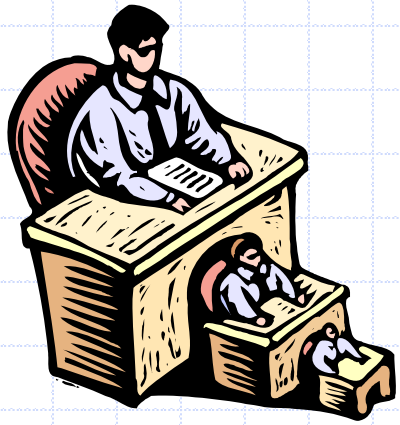
```
        rep1.hwp      3
        rep2.doc      2
    reports           6
        Tree.c        10
        Stocks.c      25
        Robot.java    20
    programs          56
    memo.txt          1
ds                    64
```

레벨순회

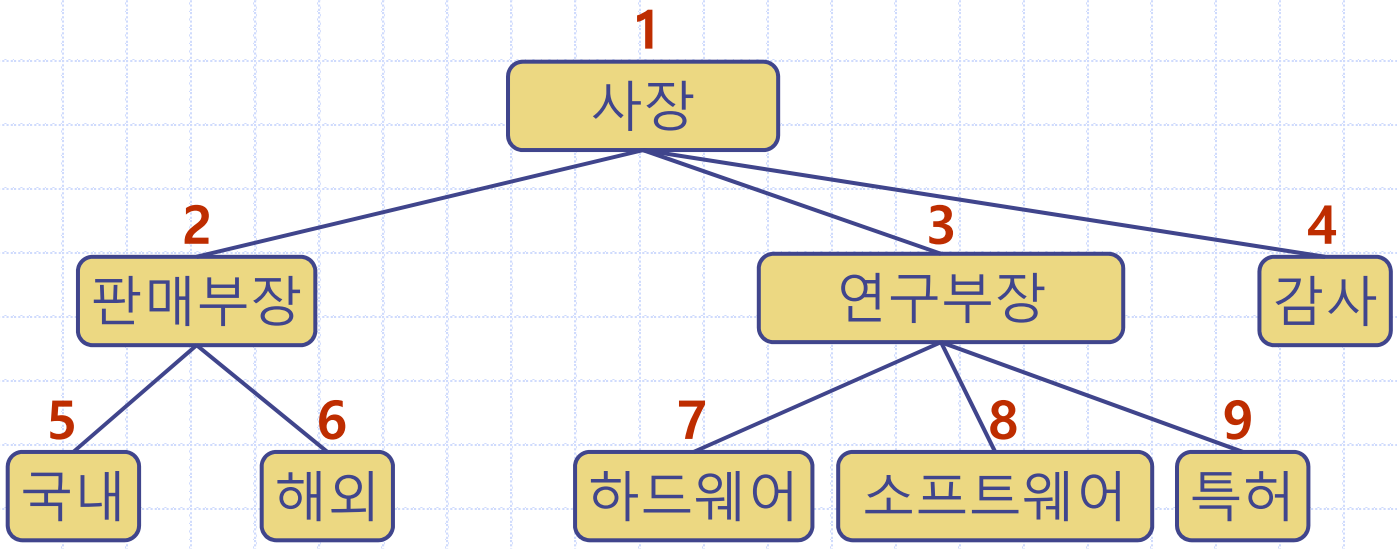
- **레벨**(level) d는 트리의 같은 깊이 d에 존재하는 모든 노드들의 집합을 나타낸다
  - 레벨 0에는 한 개의 노드, 루트만이 존재한다
- **레벨순회**(levelorder traversal)에서는 큐를 이용하여 깊이 d의 모든 노드들이 깊이 $d+1$의 노드들에 앞서 방문된다
- **실행시간: O(n)**
- **응용**
  - 관료적 계층구조 인쇄

```
Alg levelOrder(v)
1. Q ← empty queue
2. Q.enqueue(v)
3. while (!Q.isEmpty())
        v ← Q.dequeue()
        visit(v)
        for each w ∈ children(v)
            Q.enqueue(w)
4. return
```

예: 계층구조 인쇄



- 회사의 **계층구조**를 상위 레벨에서 하위 레벨 순서로 인쇄하고자 한다

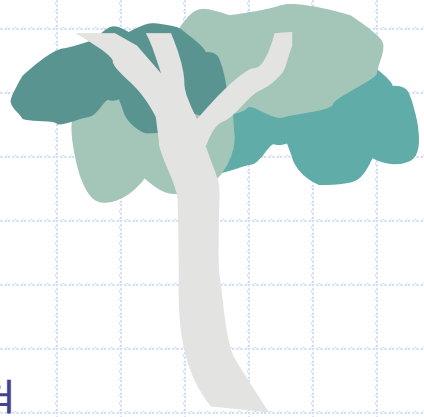


예: 계층구조 인쇄 (conti.)

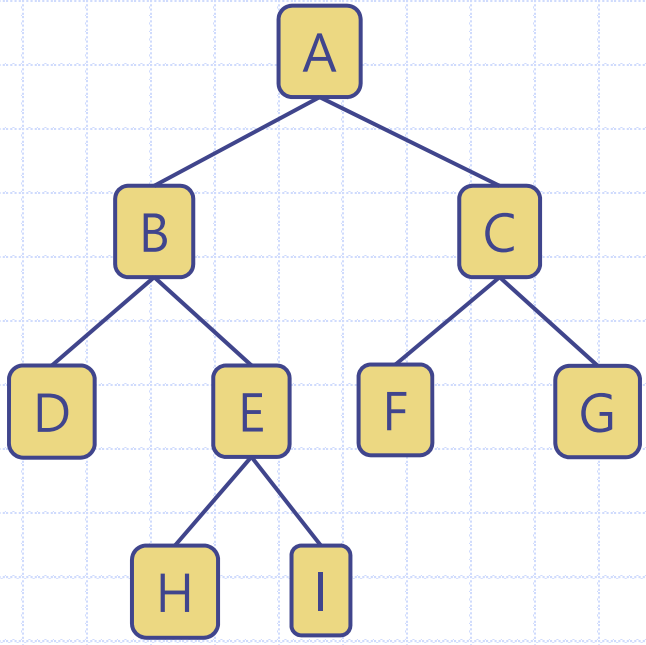
```
Alg printHierarchy(v)
1. Q ← empty queue
2. Q.enqueue(v)
3. while (!Q.isEmpty())
        v ← Q.dequeue()
        write(depth(v), element(v))
        for each w ∈ children(v)
            Q.enqueue(w)
4. return
```

```
0 사장
1 판매부장
1 연구부장
1 감사
2 국내
2 해외
2 하드웨어
2 소프트웨어
2 특허
```

이진트리 ADT

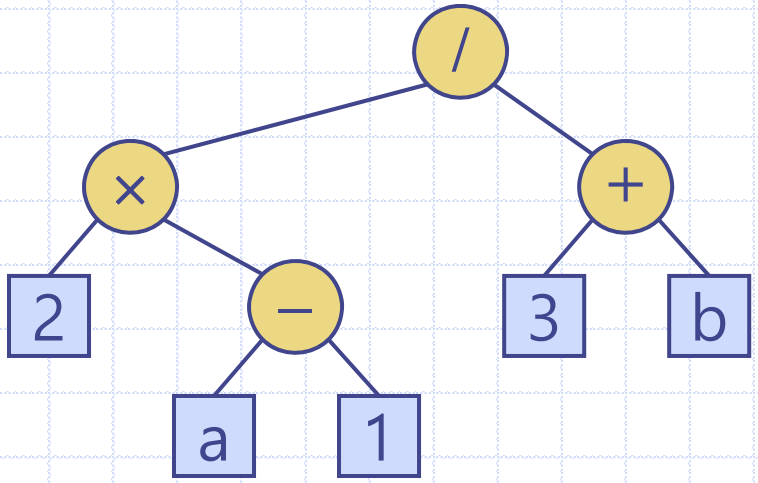


- 이진트리 ADT는 순서트리를 모델링한다
- 트리의 각 내부노드는 두 개의 자식을 가지며, 각각 **왼쪽**(left) 및 **오른쪽**(right) 자식이라 부른다 – 이를 **적정**(proper) 이진트리라고도 한다
- **이진트리**의 재귀적 정의
  - 루트가 자식의 순서쌍을 가지며, 각각의 자식은 내부노드인 경우 이진트리다
- **전제:** 이진트리는 비어있지 않다
- **응용**
  - 수식 표현
  - 의사결정 과정
  - 검색



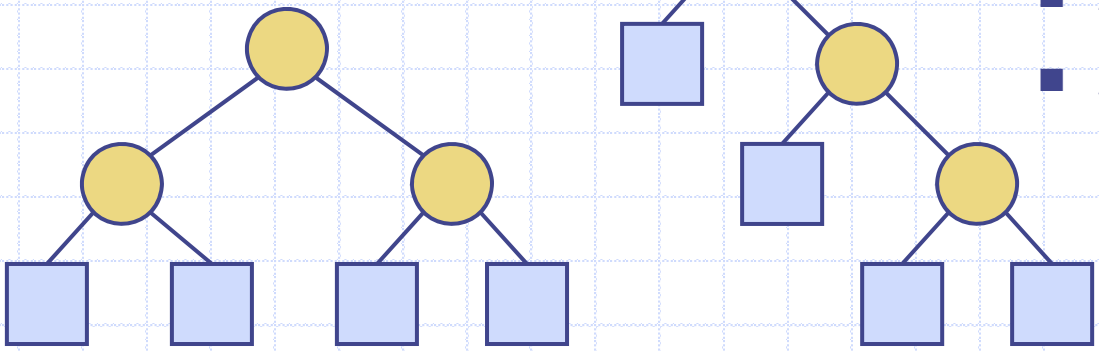
예: 수식 표현

- **수식트리**(expression tree)는 수식을 표현하는 이진트리다
  - 내부노드: **연산자**(operators)
  - 외부노드: **피연산자**(operands)
- **예:** 식 $(2 \times (a - 1) / (3 + b))$을 표현한 수식트리



이진트리의 성질

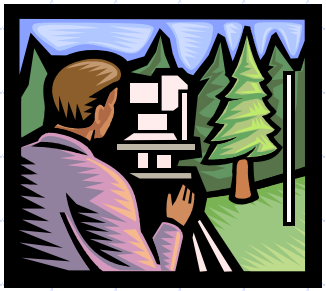
- 표기
  - n 노드 수
  - e 외부노드의 수
  - i 내부노드의 수
  - h 트리의 높이
- 성질
  - $e = i + 1$
  - $n = i + e = 2e - 1$
  - $h \le i$
  - $h \le (n-1)/2$
  - $e \le 2^h$
  - $h \ge \log_2 e$
  - $h \ge \log_2(n+1) - 1$



이진트리 ADT 메쏘드

- 이진트리 ADT는 트리 ADT의 확장이다 – 즉, 트리 ADT의 모든 메쏘드들을 상속한다
- 추가적인 메쏘드
  - position leftChild(p)
  - position rightChild(p)
  - position sibling(p)
- **전제:** 적정이진트리
  - 만약 이진트리가 부적정하다면 작업시 오류가 발생할 수 있다
- 이진트리 ADT를 구현하는 데이터구조에 따라 추가적인 갱신 메쏘드들(삽입, 삭제 등)이 정의될 수 있다

깊이와 높이



- 노드 v의 **깊이**(depth)의 재귀적 정의
  - 만약 v가 루트면, v의 깊이는 0
  - 그렇지 않으면, v의 깊이는 v의 부모의 깊이 더하기 1
- 노드 v의 **높이**(height)의 재귀적 정의
  - 만약 v가 외부노드면, v의 높이는 0
  - 그렇지 않으면, v의 높이는 v의 왼쪽과 오른쪽 자식 중 최대 높이 더하기 1

```
Alg depth(v)
1. if (isRoot(v))
       return 0
   else
       return 1 + depth(parent(v))

Alg height(v)
1. if (isExternal(v))
       return 0
   else
       h ←
           max(height(leftChild(v)),
               height(rightChild(v)))
       return 1 + h
```

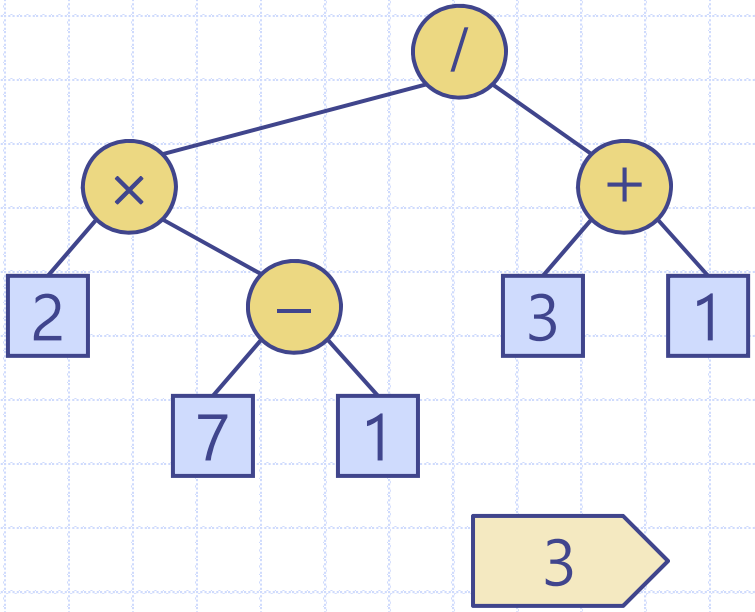
이진트리 순회

- 이진트리의 순회는 트리 순회의 **특화**(specialization)다
- **선위순회**(preorder traversal)에서는 노드가 그의 왼쪽 및 오른쪽 부트리보다 **앞서** 방문된다
- **후위순회**(postorder traversal)에서는 노드가 그의 왼쪽 및 오른쪽 부트리보다 **나중에** 방문된다

```
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)
```

예: 수식 평가

- 수식트리에 저장된 수식을 평가하고자 한다



- **후위순회**의 특화 – 단,
  - 부트리의 값을 반환하는 재귀적 메쏘드
  - 내부노드 방문시에 부트리의 값들을 결합한다

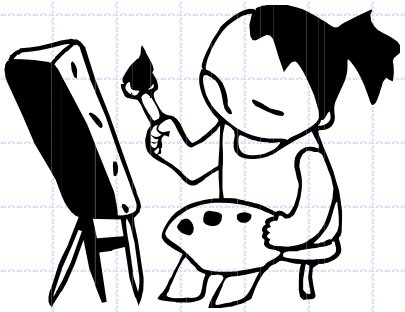
```
Alg evalExpr(v)
1. if (isExternal(v))
       return element(v)
   else
       x ← evalExpr(leftChild(v))
       y ← evalExpr(rightChild(v))
       ◊ ← element(v)
       return x ◊ y
```

중위순회

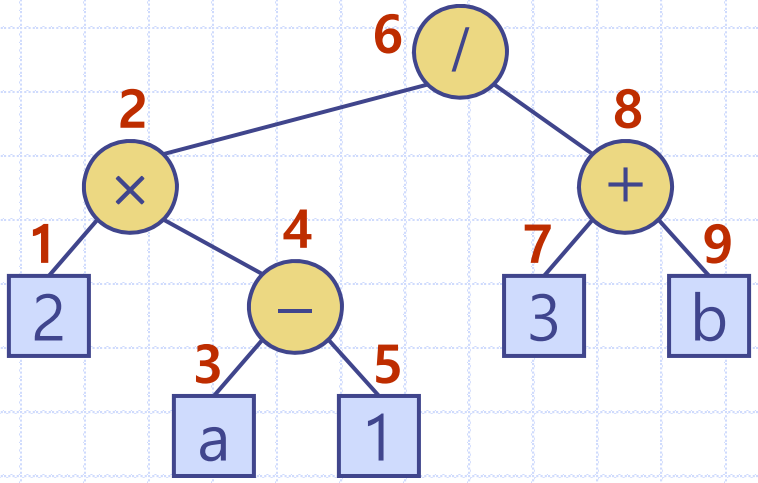
- **중위순회**(inorder traversal)에서는 노드가 그의 왼쪽 부트리보다는 **나중에**, 오른쪽 부트리보다는 **앞서** 방문된다
- **실행시간: O(n)** – 단, n은 이진트리내 총 노드의 수
- **응용**
  - 이진트리 그리기
  - 수식 인쇄

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

예: 이진트리 그리기



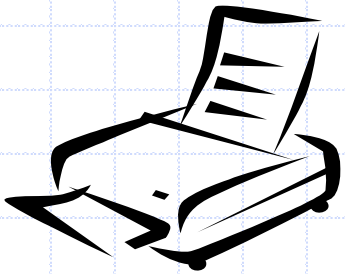
- 이진트리의 노드들을 *xy* 평면에 그리고자 한다
- **중위순회**의 특화 – 단 노드 *v*의 *xy* 좌표에 다음 정보를 이용
  - x_v = *v*의 중위 순위
  - y_v = *v*의 깊이



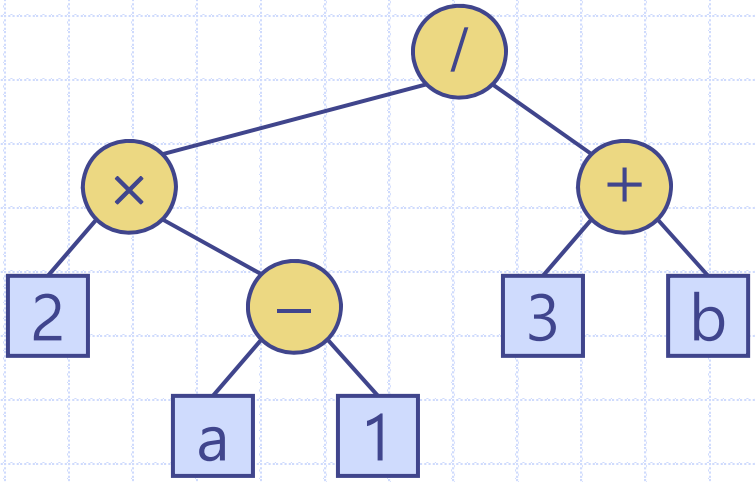
Alg ***drawBinaryTree(v)***
1. ***rank*** ← 0
2. ***rDraw(v, rank)***

Alg ***rDraw(v, rank)***
1. **if** (***isInternal(v)***)
   rDraw(leftChild(v), rank)
2. ***rank*** ← ***rank*** + 1
3. ***drawNodeXY(v, rank, depth(v))***
4. **if** (***isInternal(v)***)
   rDraw(rightChild(v), rank)

예: 수식 인쇄



- 수식트리로부터 괄호쳐진 수식을 인쇄하고자 한다

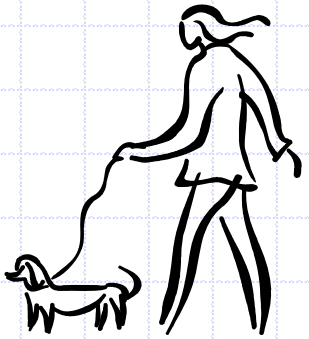


((2 × (a – 1)) / (3 + b))

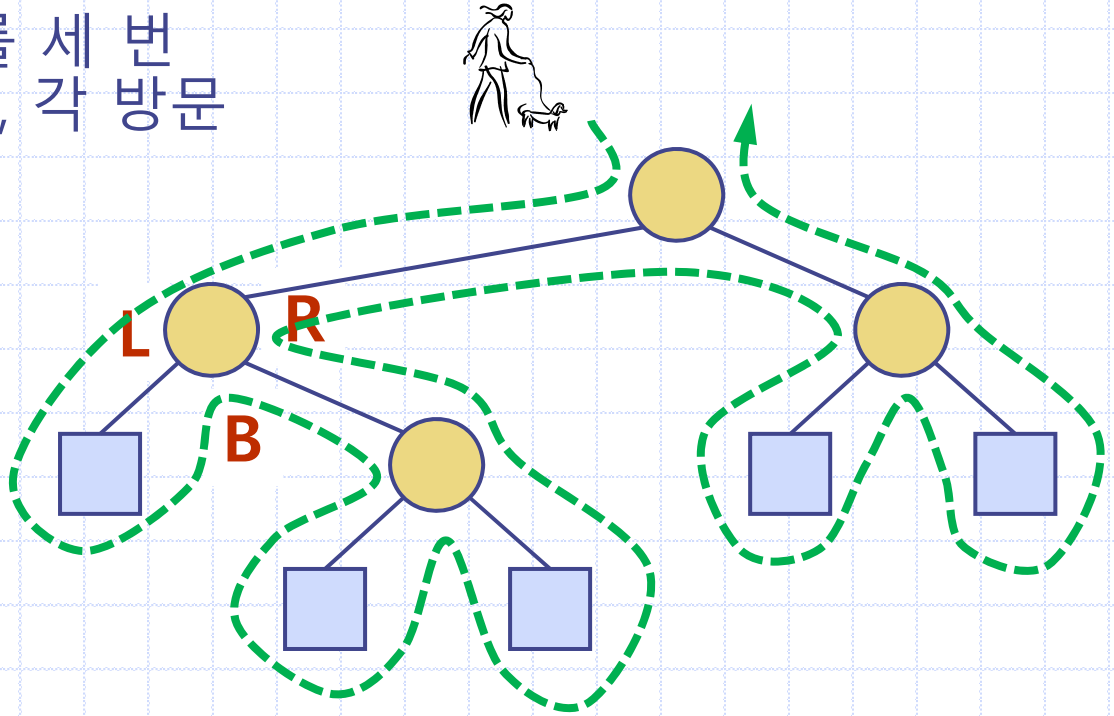
- **중위순회**의 특화 – 단,
  - 노드를 방문시에 인쇄
  - 왼쪽 부트리를 순회하기 전에 "("를 인쇄
  - 오른쪽 부트리를 순회하고 나서 ")"를 인쇄

```
Alg printExpr(v)
1. if (isInternal(v))
       write("(")
       printExpr(leftChild(v))
2. write(element(v))
3. if (isInternal(v))
       printExpr(rightChild(v))
       write(")")
```

오일러 투어 순회



- **오일러 투어**(Euler Tour)는 이진트리에 대한 **일반순회**(generic traversal)다
- 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**)

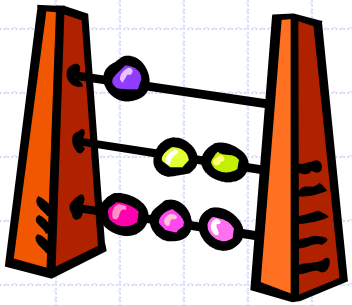


오일러 투어 순회 (conti.)

- **선위**, **중위**, **후위** 순회를 모두 포함한다
- **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: 부트리의 노드 수



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

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

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

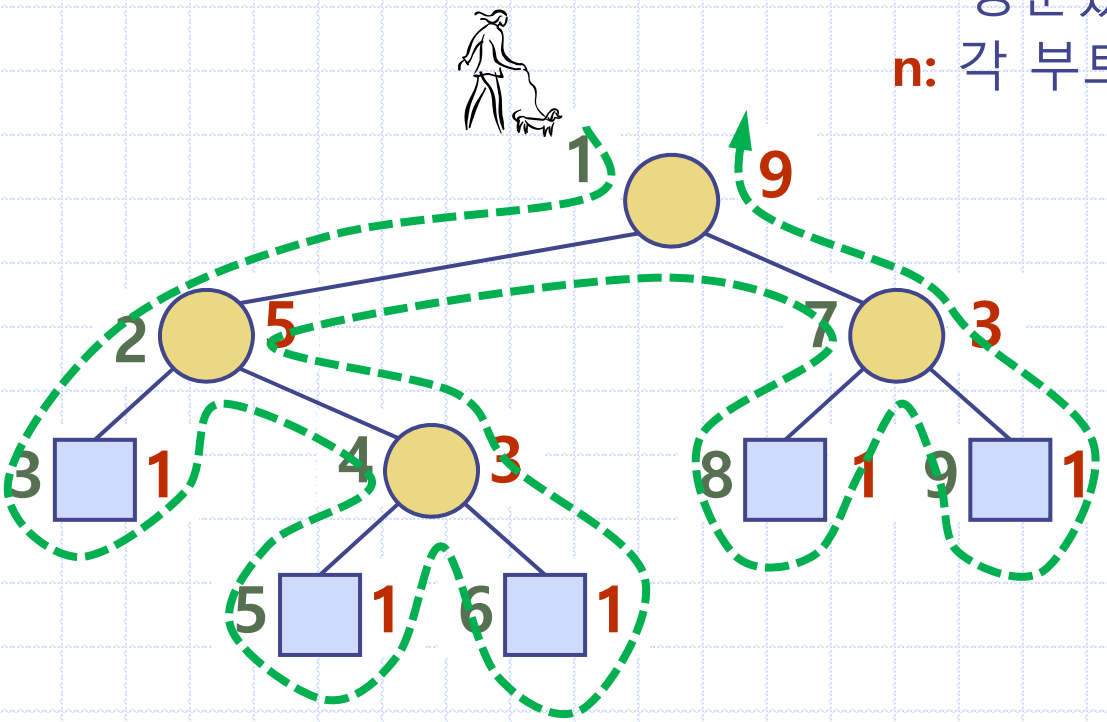
Alg *visitBelow*(v)
1. **return**

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

작동 원리

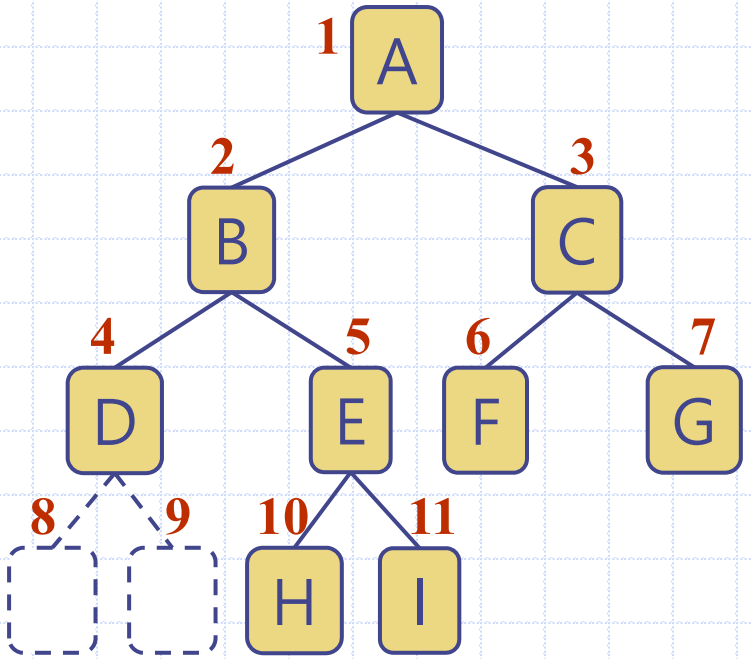
k: 각 노드를 왼쪽에서
방문했을 때의 카운터 값

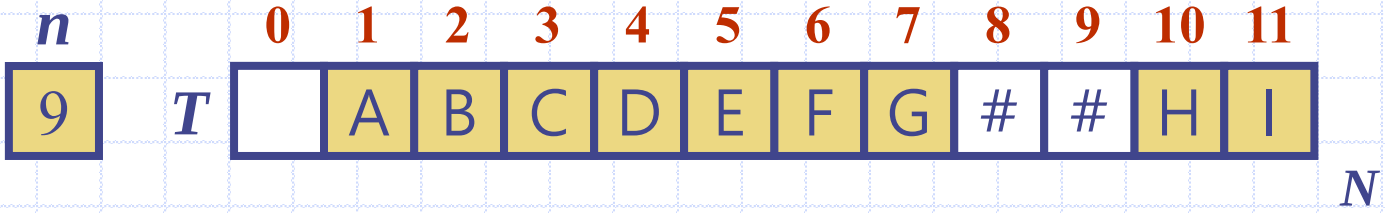
n: 각 부트리의 노드 수



배열에 기초한 이진트리

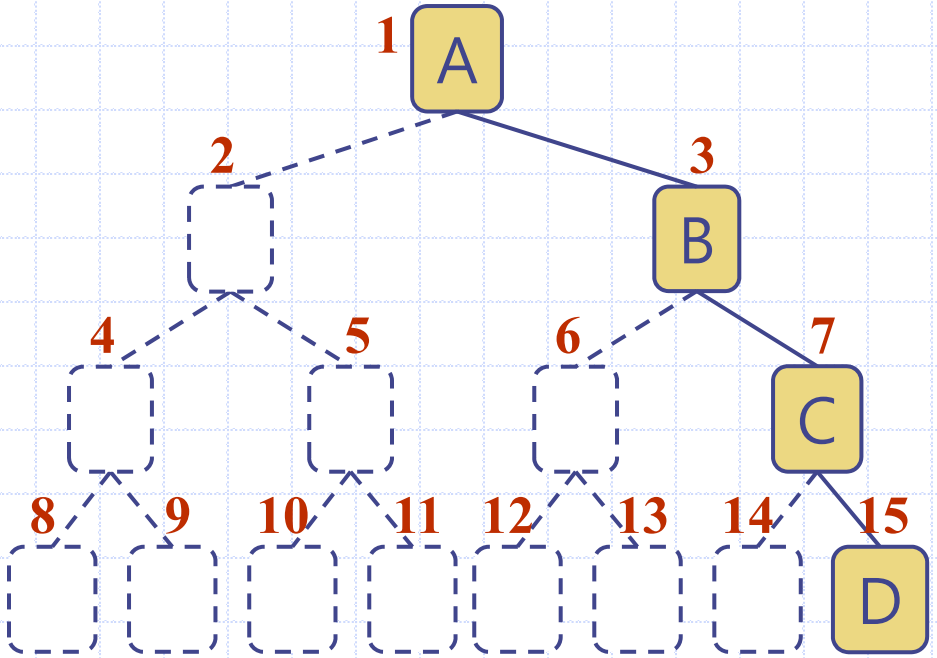
- **1D 배열**을 이용하여 이진트리를 표현할 수 있다
- 랭크 i의 노드에 대해:
  - **왼쪽 자식**의 위치는 순위 $2i$
  - **오른쪽 자식**의 위치는 순위 $2i + 1$
  - **부모**의 위치는 순위 $\lfloor i/2 \rfloor$
- 노드간의 링크 저장 불필요
- 랭크 0 셀은 사용하지 않음
- 미사용 셀은 특별값을 저장
  - 널마커(**예:** '#'), 또는
  - 널포인터(포인터배열인 경우)

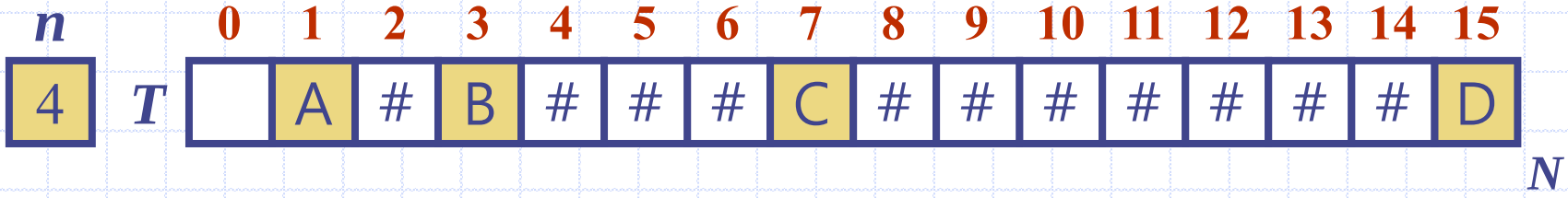




최선과 최악의 경우

- P_{MAX}를 노드 위치 중 최대값이라 하면, 배열크기 $N = P_{MAX}$
- **최선**의 경우, $N = n$
  - 이런 이진트리를 **완전이진트리**(complete binary tree)라고 부른다
- **최악**의 경우, $N = 2^n - 1$ (단, 부적정이진트리 경우임)





배열에 기초한 이진트리 메쏘드

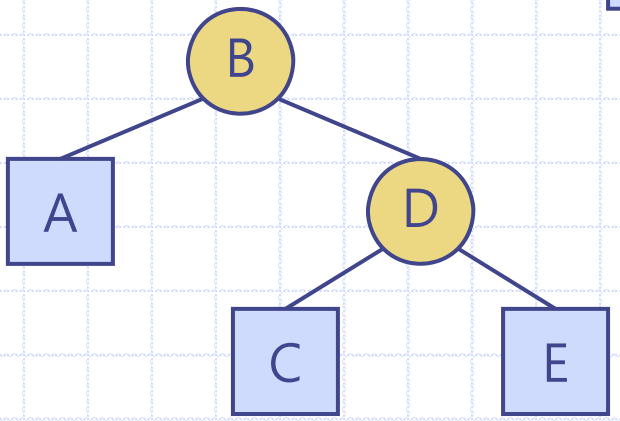
```
Alg element(v)
1. return T[v]

Alg root()
1. return 1

Alg isRoot(v)
1. return v = 1
```

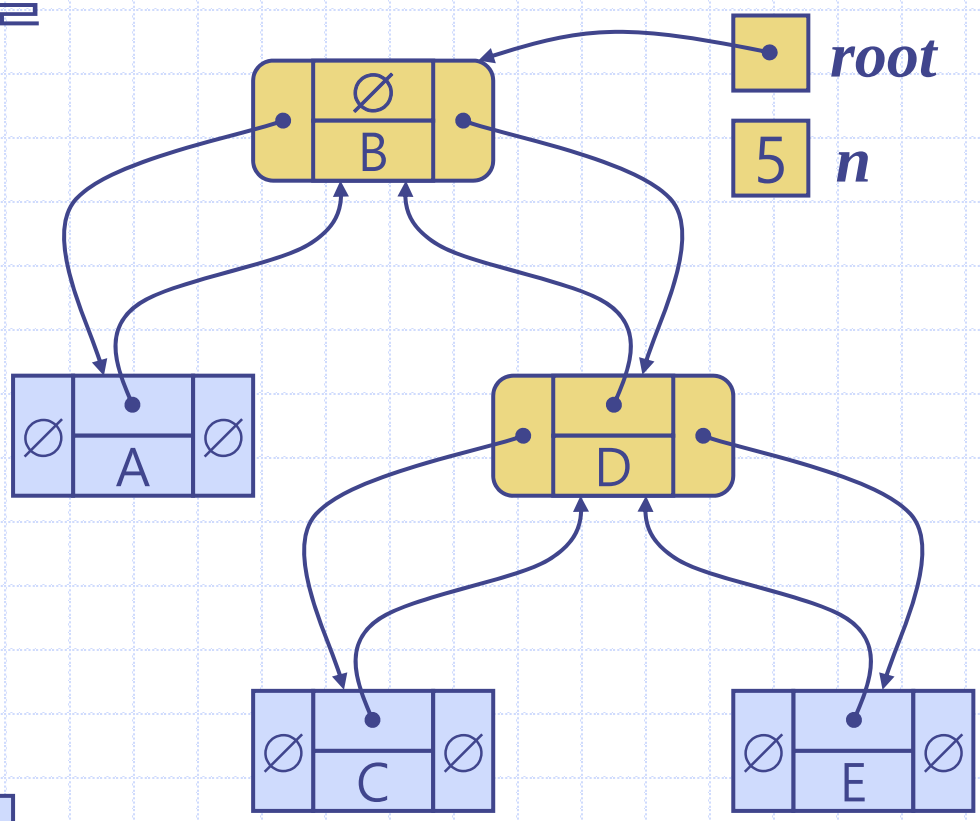
```
Alg parent(v)
1. return ⌊v/2⌋

Alg leftChild(v)
1. return 2v

Alg rightChild(v)
1. return 2v + 1

Alg sibling(v)
1. if even(v)
       return v + 1
   else
       return v – 1
```

```
Alg isInternal(v)
1. return (2v ≤ N – 1) & (T[2v] ≠ Null)

Alg isExternal(v)
1. return (2v > N – 1) || (T[2v] = Null)

Alg replaceElement(v, e)
1. T[v] ← e
2. return e

Alg swapElements(v, w)
1. tmp ← T[v]
2. T[v] ← T[w]
3. T[w] ← tmp
4. return
```

연결이진트리

- 노드 개체들은 **위치**를 구현
- 노드 저장내용
  - 원소
  - 부모노드
  - 왼쪽 자식노드
  - 오른쪽 자식노드





연결이진트리 메쏘드

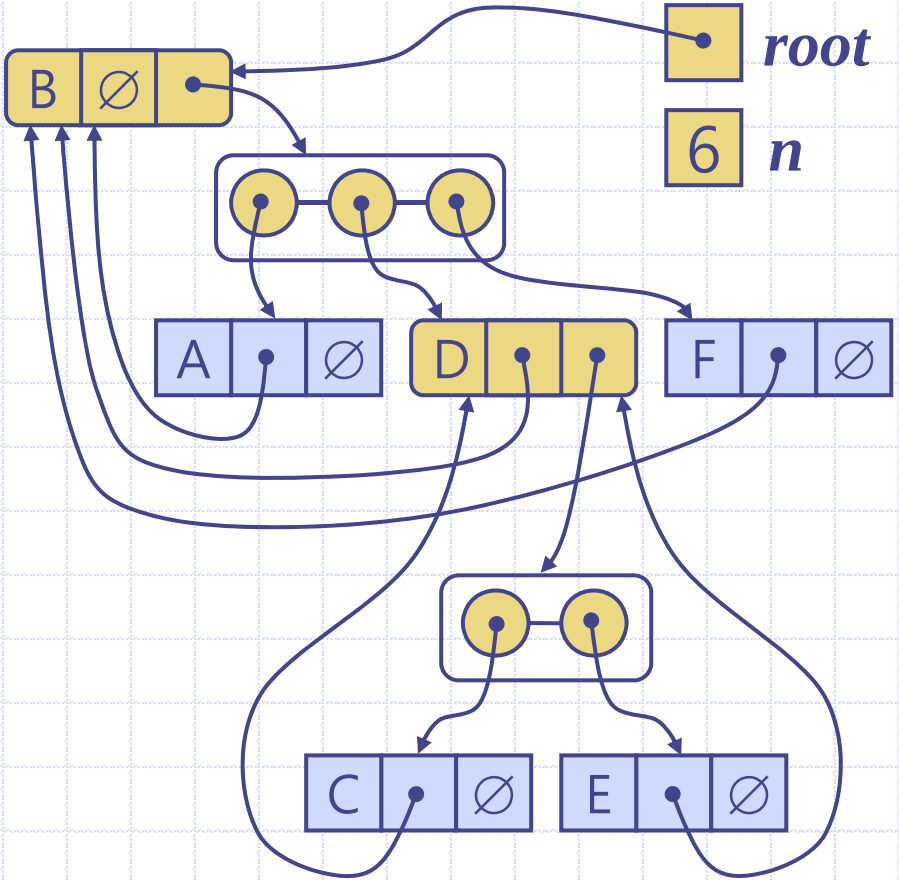
```
Alg element(v)
1. return v.elem

Alg root()
1. return root

Alg isRoot(v)
1. return v = root
```

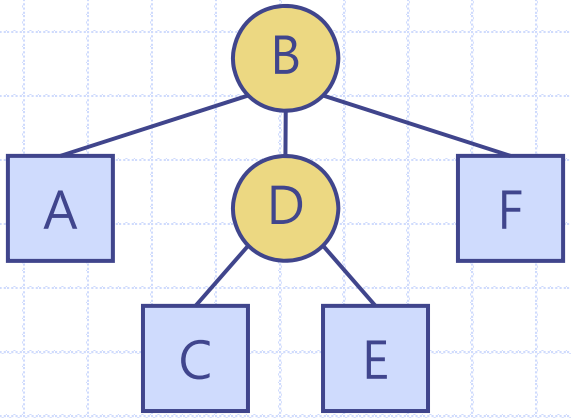
```
Alg parent(v)
1. return v.parent

Alg leftChild(v)
1. return v.left

Alg rightChild(v)
1. return v.right

Alg sibling(v)
1. p ← v.parent
2. if (p.left = v)
       return p.right
   else
       return p.left
```

```
Alg isInternal(v)
1. return (v.left ≠ ∅) & (v.right ≠ ∅)

Alg isExternal(v)
1. return (v.left = ∅) & (v.right = ∅)

Alg replaceElement(v, e)
1. v.elem ← e
2. return e

Alg swapElements(v, w)
1. tmp ← v.elem
2. v.elem ← w.elem
3. w.elem ← tmp
4. return
```

연결트리 (Ver.1)

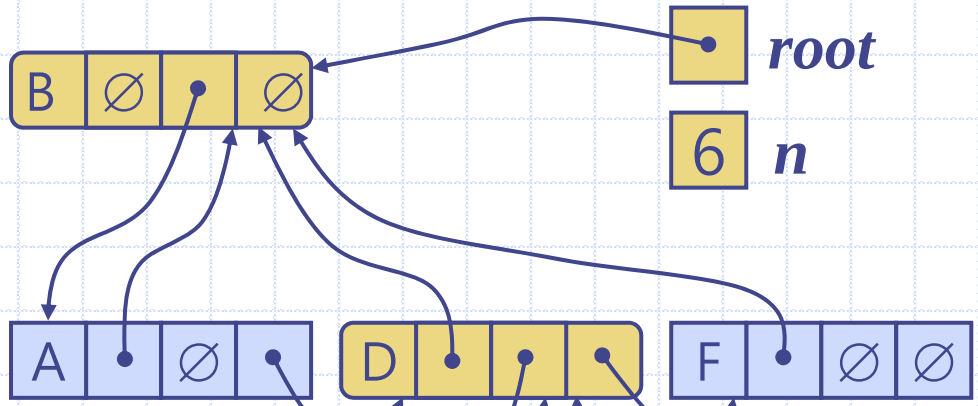
- 노드 개체들은 **위치**를 구현
- 노드 저장내용
  - 원소
  - 부모노드
  - 자식노드들의 리스트

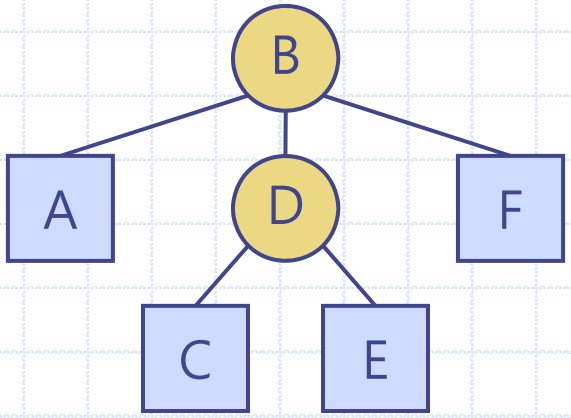




연결트리 (Ver.2)

- 노드 개체들은 **위치**를 구현
- 노드 저장내용
  - 원소
  - 부모노드
  - 첫째 자식노드
  - 바로 아래 동생노드





연결트리 메쏘드 (Ver.2)

Alg *element*(*v*)
1. **return *v*.elem**

Alg *root*()
1. **return *root***

Alg *isRoot*(*v*)
1. **return *v* = *root***

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

Alg *children*(*v*)
1. $C \leftarrow \emptyset$
2. $c \leftarrow v$.first
3. **while** ($c \neq \emptyset$)
   $C \leftarrow C \cup \{c\}$
   $c \leftarrow c$.next
4. **return** C

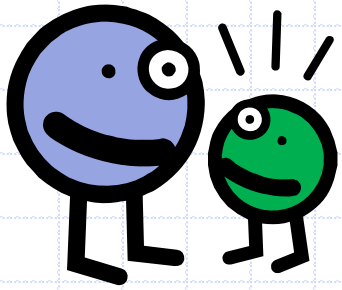
Alg *isInternal*(*v*)
1. **return** v.first $\neq \emptyset$

Alg *isExternal*(*v*)
1. **return** v.first $= \emptyset$

Alg *replaceElement*(*v*, *e*)
1. v.elem $\leftarrow e$
2. **return** e

Alg *swapElements*(*v*, *w*)
1. *tmp* $\leftarrow v$.elem
2. v.elem $\leftarrow w$.elem
3. w.elem $\leftarrow$ *tmp*
4. **return**

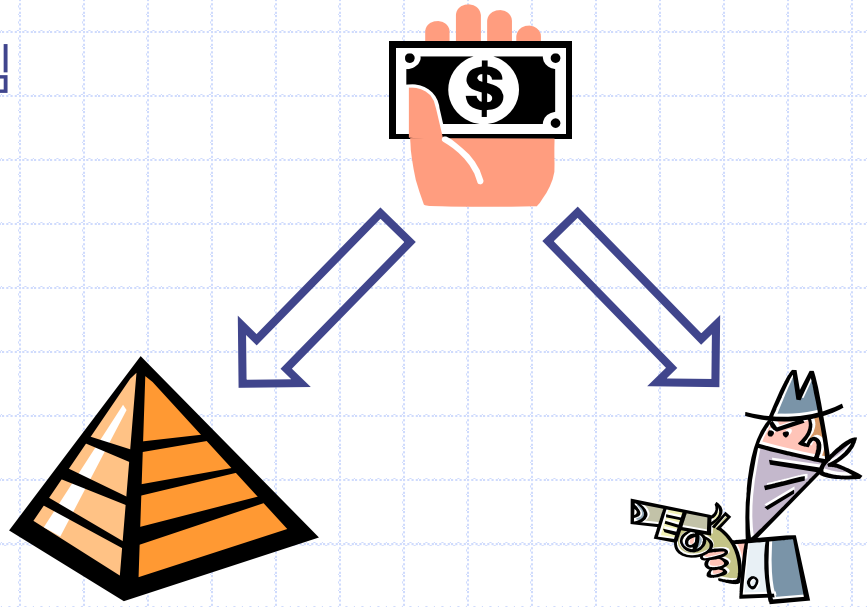
성능



작업	이진트리		트리
	배열	연결	연결
size, isEmpty*	1	1	1
root, parent	1	1	1
children(v)	1	1	c_v
leftChild, rightChild, sibling	1	1	N/A
isInternal, isExternal, isRoot	1	1	1
replaceElement, swapElements	1	1	1

응용문제

- 흥미로운 설계 문제를 통해 어떻게 **이진트리**를 주요 데이터구조로 사용하는지 공부한다
- 설계 문제
  - 계승자
  - 로만노드
  - 양자택일식 문답시스템



응용문제: 계승자

- 이진트리 T의 노드 v에 대한 아래의 **일반** 알고리즘들을 작성하라
  - preOrderSucc(v): **선위순회 계승자**(즉, T를 선위순회할 경우 노드 v 직후에 방문되는 노드)를 반환
  - inOrderSucc(v): **중위순회 계승자**(즉, T를 중위순회할 경우 노드 v 직후에 방문되는 노드)를 반환
  - postOrderSucc(v): **후위순회 계승자**(즉, T를 후위순회할 경우 노드 v 직후에 방문되는 노드)를 반환
- **주의:** 계승자 노드가 존재하지 않을 경우 invalidNodeException을 발령해야 함
- **힌트:** 이진트리 ADT의 기본 메쏘드들 사용 가능

해결

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

1. if (isInternal(v))
        return leftChild(v)
2. p ← parent(v)
3. while (leftChild(p) ≠ v)
        if (isRoot(p))
            invalidNodeException()
        v ← p
        p ← parent(p)
4. return rightChild(p)
```

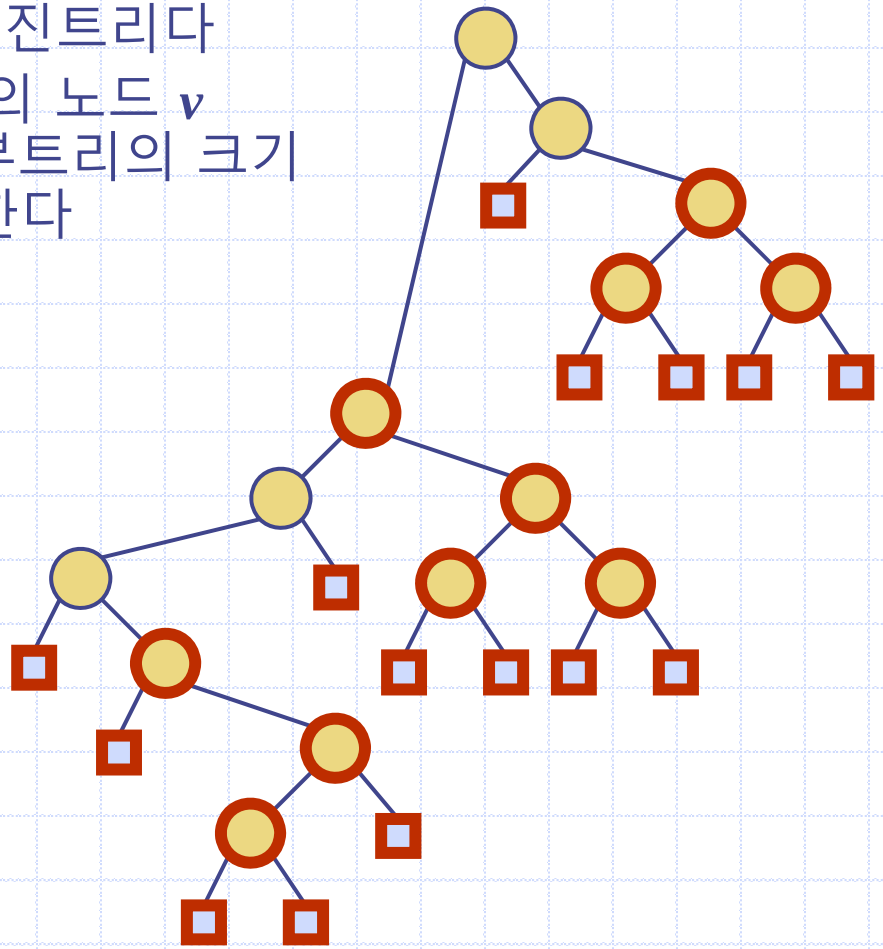
```
Alg inOrderSucc(v)
    input node v
    output node

1. if (isInternal(v))
        v ← rightChild(v)
        while (isInternal(v))
            v ← leftChild(v)
        return v
2. p ← parent(v)
3. while (leftChild(p) ≠ v)
        if (isRoot(p))
            invalidNodeException()
        v ← p
        p ← parent(p)
4. return p
```

해결 (conti.)

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

1. if (isRoot(v))
        invalidNodeException()
2. p ← parent(v)
3. if (rightChild(p) = v)
        return p
4. v ← rightChild(p)
5. while (!isExternal(v))
        v ← leftChild(v)
6. return v
```

응용문제: 로만노드

- T는 n개의 노드로 구성된 이진트리다
- **로만노드**(roman node)란 T의 노드 v 가운데 v의 왼쪽과 오른쪽 부트리의 크기 차이가 5 이내인 노드를 말한다
- 주어진 노드 v와 v의 모든 자손들이 로만인 경우 v를 루트로 하는 부트리의 크기를, 그렇지 않으면 0을 반환하는 **선형시간**의 **일반** 알고리즘 romanSize(v)를 작성하라
- **주의:** 노드를 중복 방문하면 **선형시간** 조건을 만족하기 어렵다



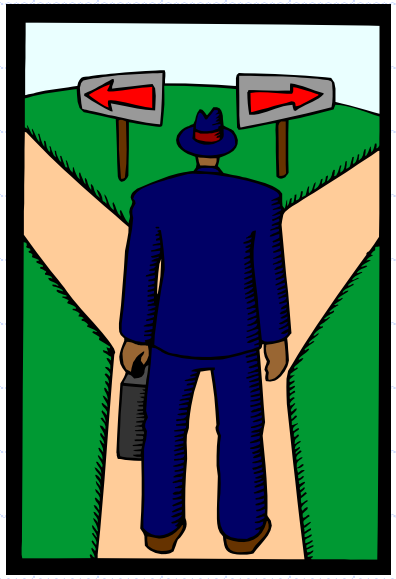
해결

- romanSize(v)는 노드 v와 v의 모든 후손이 **로만**이면 루트를 v로 하는 부트리의 모든 노드의 수를, 그렇지 않으면 0을 반환한다
- 어느 노드도 한 번 이상 방문하지 않으므로 $O(n)$ 시간에 실행한다

```
Alg romanSize(v)
1. if (isExternal(v))                    {roman}
       return 1
2. l ← romanSize(leftChild(v))
3. if (l = 0)
       return 0
4. r ← romanSize(rightChild(v))
5. if ((r > 0) & (|l – r| ≤ 5))
       return l + r + 1                  {roman}
   else
       return 0
```

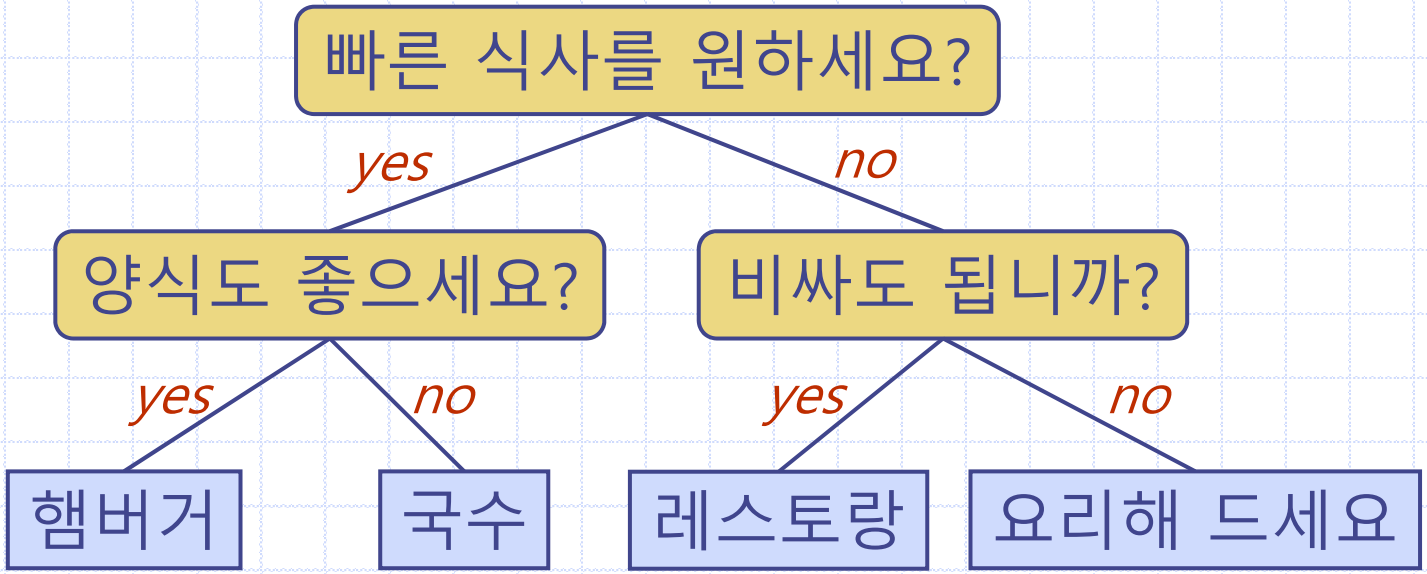
응용문제: 양자택일식 문답시스템

- 대답에 따라 다양한 결정 가운데 하나를 제공하는 **양자택일식 문답시스템**을 구축하고자 한다
  - 예/아니오 응답을 요하는 질문들
  - 결정들
- **예**
  - 스무고개
  - 해결사(troubleshooters)
  - 분류기(classifiers)
  - 추천기(recommenders)



해결: 결정트리

- **결정트리**(decision tree)란 의사결정 과정과 연관된 이진트리를 말한다
  - 내부노드: 질문(yes/no 응답을 요함)
  - 외부노드: 결정
- **예:** 식사 결정



해결: 배열로 결정트리 구축

```
Alg buildDecisionTree()
    input questions and decisions
    output a decision tree

1. write("***Let's build a dichotomous
    QA system")
2. makeInternalNode(1)
3. return

Alg makeExternalNode(i)
1. write("Enter decision:")
2. T[i] ← read()
3. if (2i < N)
        T[2i] ← Null
        T[2i + 1] ← Null
4. return
```

```
Alg makeInternalNode(i)
1. write("Enter question:")
2. q ← T[i] ← read()
3. write("Question if yes to", q, "?")
4. if (read() = "yes")
        makeInternalNode(2i)
    else
        makeExternalNode(2i)
5. write("Question if no to", q, "?")
6. if (read() = "yes")
        makeInternalNode(2i + 1)
    else
        makeExternalNode(2i + 1)
7. return
```

해결: 연결리스트로 결정트리 구축

```
Alg buildDecisionTree()
    input questions and decisions
    output a decision tree

1. write("***Let's build a dichotomous
    QA system")
2. return makeInternalNode()

Alg makeExternalNode()
1. v ← getnode()
2. write("Enter decision:")
3. v.elem ← read()
4. v.left ← ∅
5. v.right ← ∅
6. return v
```

```
Alg makeInternalNode()
1. v ← getnode()
2. write("Enter question:")
3. q ← v.elem ← read()
4. write("Question if yes to", q, "?")
5. if (read() = "yes")
        v.left ← makeInternalNode()
   else
        v.left ← makeExternalNode()
6. write("Question if no to", q, "?")
7. if (read() = "yes")
        v.right ← makeInternalNode()
   else
        v.right ← makeExternalNode()
8. return v
```

해결: 결정트리를 구축하기 위한 실행 예

***Let's build a dichotomous QA system
Enter question: *빠른 식사를 원하세요?*
Question if yes to ''빠른 식사를 원하세요?'? *yes*
Enter question: *양식도 좋으세요?*
Question if yes to '양식도 좋으세요?'? *no*
Enter decision: *햄버거*
Question if no to '양식도 좋으세요?'? *no*
Enter decision: *국수*
Question if no to '빠른 식사를 원하세요?'? *yes*
Enter question: *비싸도 됩니까?*
…

해결: 결정트리를 사용한 실행 예

```
Alg runDecisionTree(v)
    input decision tree v
    output decision

1. write("***Please answer questions")
2. processNode(v)

Alg processNode(v)
1. write(element(v))
2. if (isInternal(v))
        if (read() = "yes")
            processNode(leftChild(v))
        else
            processNode(rightChild(v))
```

***Please answer questions
빠른 식사를 원하세요? *yes*
양식도 좋으세요? *no*
국수

***Please answer questions
빠른 식사를 원하세요? *no*
비싸도 됩니까? *no*
요리해 드세요