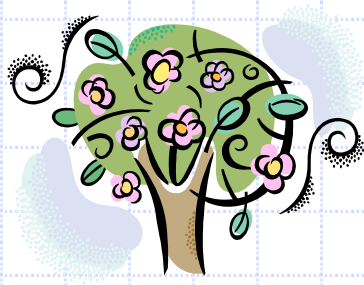


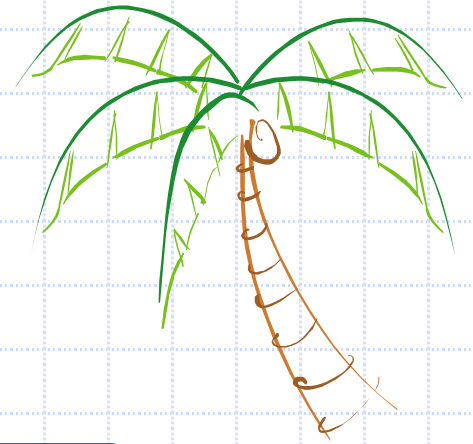
트리



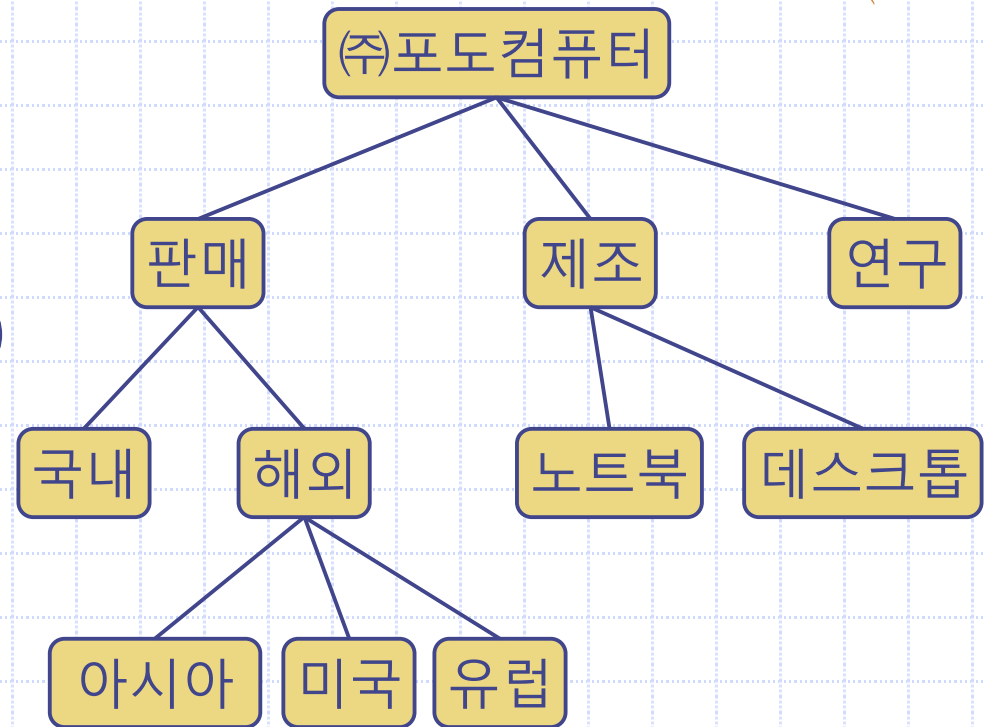
Outline

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

트리 ADT

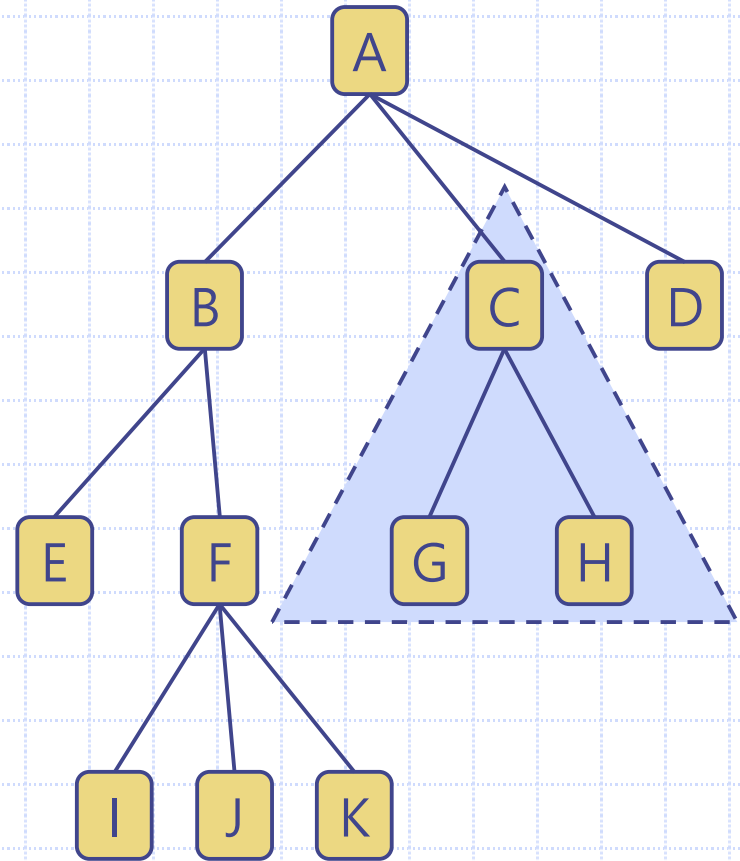


- ◆ **트리** ADT는 계층적으로 저장된 데이터원소들을 모델링
- ◆ 맨위의 원소를 제외하고, 각 트리 원소는 **부모**(parent) 원소와 0개 이상의 **자식**(children) 원소들을 가진다
- ◆ **전제**: 트리는 비어 있지 않다 – 알고리즘 단순화



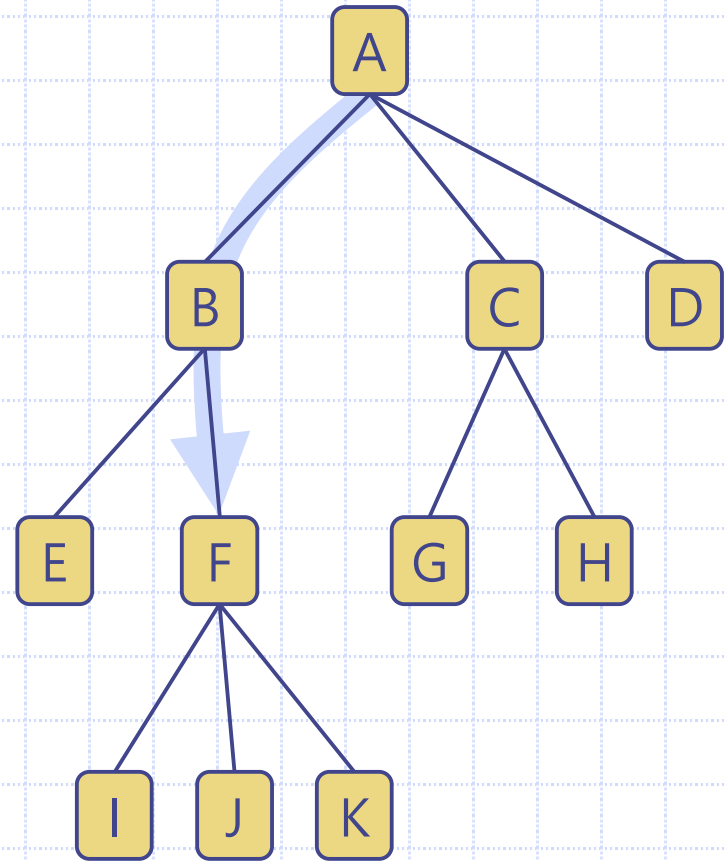
트리 용어

- ◆ 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.)

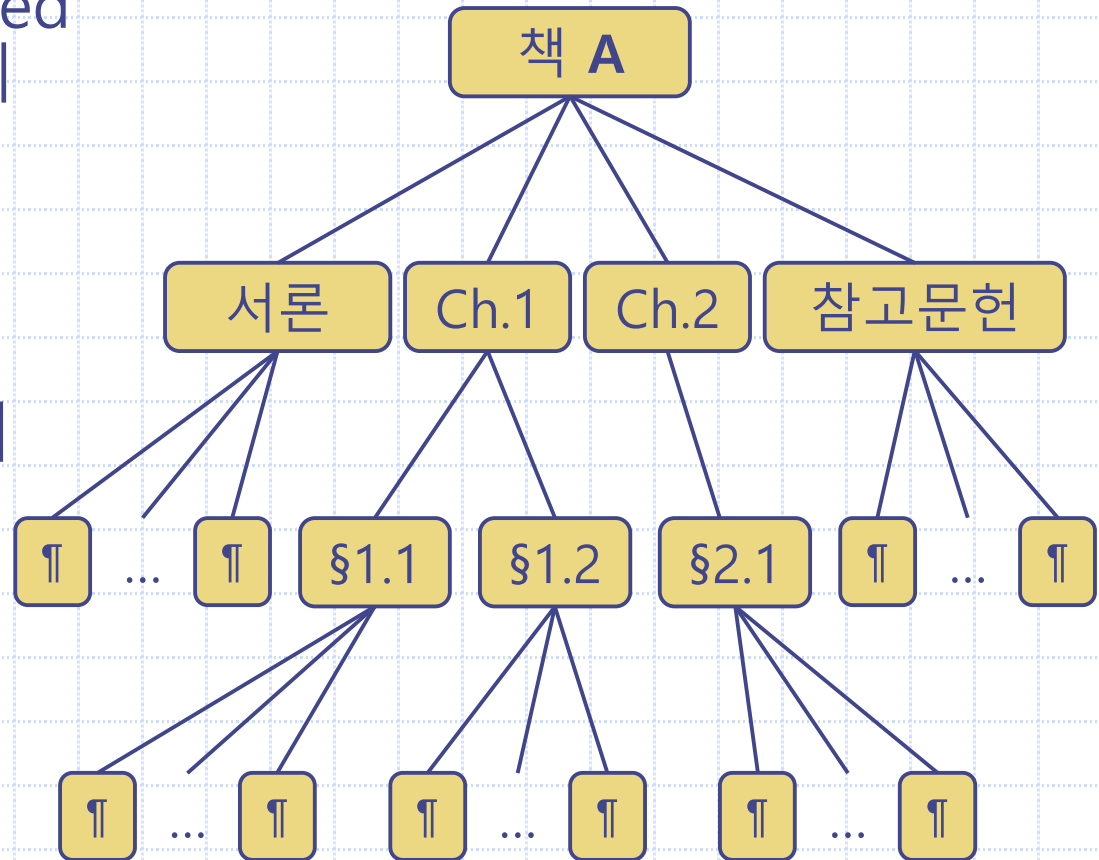
- ◆ **경로(path)**: 조상 또는 자손을 따라 이어진 노드 시퀀스(예: ABF)
- ◆ **경로길이(path length)**: 경로내 간선(edge)의 수
- ◆ **노드의 깊이(depth)**: 루트로부터 노드에 이르는 유일한 경로의 길이
- ◆ **노드의 높이(height)**: 노드로부터 외부노드에 이르는 가장 긴 경로의 길이
- ◆ **트리의 높이(height of a tree)**: 루트의 높이



순서트리

◆ 순서트리(ordered tree): 각 노드의 자식들에 대해 선형 순서가 정의되어 있는 트리

◆ 예: 구조적 문서



트리 ADT 메소드

◆ 일반 메소드

- boolean isEmpty()*
- integer size()

◆ 접근 메소드

- node root()
- node parent(v)
- node children(v)
- element element(v)

◆ 질의 메소드

- boolean isInternal(v)
- boolean isExternal(v)
- boolean isRoot(v)

◆ 갱신 메소드

- swapElements(v, w)
- element setElement(v, e)

◆ 예외

- invalidNodeException(): 불법 노드 접근 시 발령

◆ 트리 ADT를 구현하는 데이터구조에 따라 추가적인 갱신 메소드(삽입, 삭제 등) 정의 가능

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

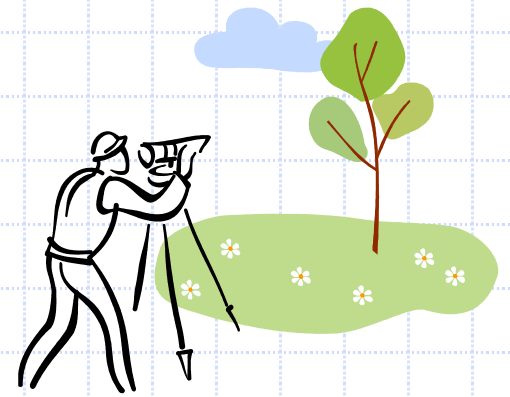


깊이

- ◆ 노드 v 의 **깊이**(depth)의 재귀적 정의
 - 만약 v 가 루트면, v 의 깊이는 0
 - 그렇지 않으면, v 의 깊이는 v 의 부모의 깊이 더하기 1
- ◆ 실행시간: $O(n)$ – 단, n 은 트리 내 총 노드 수
- ◆ **depth**(v)의 트리는 **일반**(generic) **트리** – 즉, 트리의 상세 구현과 독립

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

높이



- ◆ 노드 v 의 **높이**(height)의 재귀적 정의
 - 만약 v 가 외부노드면, v 의 높이는 0
 - 그렇지 않으면, v 의 높이는 v 의 자식들 중 최대 높이 더하기 1
- ◆ 실행시간: $O(n)$ – 단, n 은 트리 내 총 노드 수
- ◆ 루트가 r 인 **트리의 높이**(height of a tree)는 **height**(r)을 호출하여 계산

Alg *height*(v)

```
1. if (isExternal( $v$ ))
    return 0
else
     $h \leftarrow 0$ 
    for each  $w \in \text{children}(v)$ 
         $h \leftarrow \max(h, \text{height}(w))$ 
    return  $1 + h$ 
```

선위순회

- ◆ 순회(traversal)란 트리의 노드들을 체계적인 방식으로 방문하는 것을 말한다
- ◆ 선위순회(preorder traversal): 노드를 그의 자손들보다 앞서 방문
- ◆ 실행시간: $O(n)$ - 단, n 은 트리 내 총 노드 수
- ◆ 응용
 - 구조적 문서를 인쇄
 - 계층적 파일시스템의 모든 폴더들을 나열

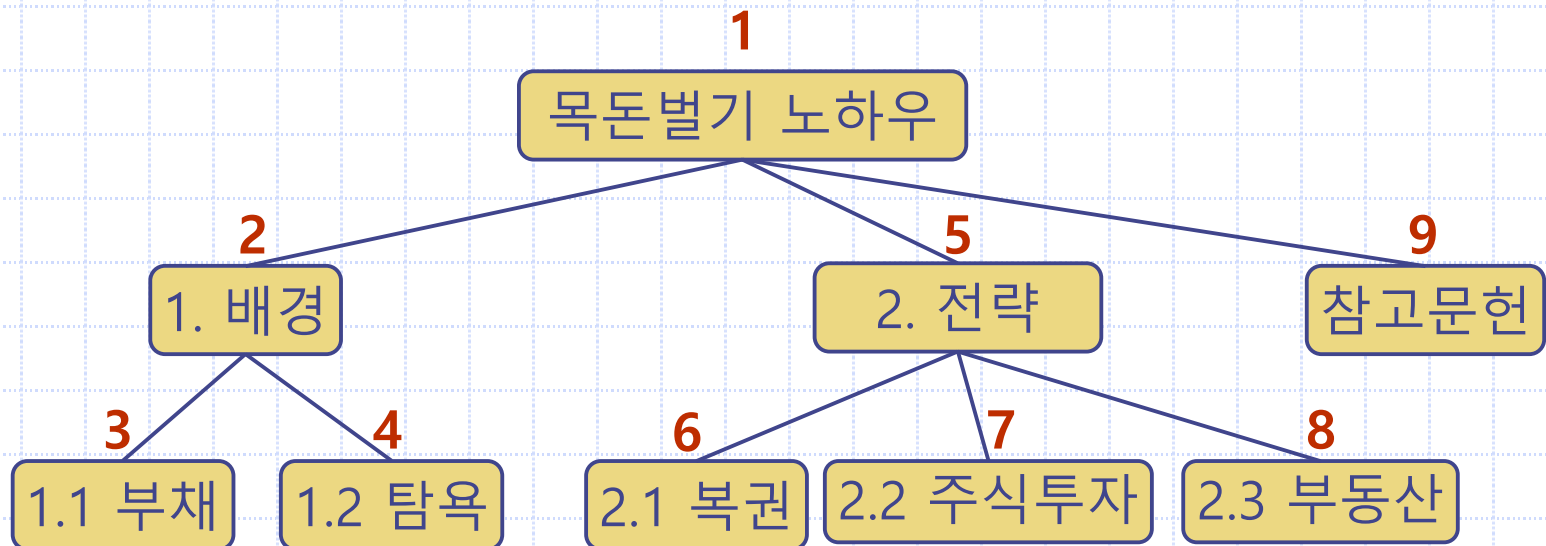
Alg *preOrder*(v)

1. *visit*(v)
2. for each $w \in \text{children}(v)$
 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* \leftarrow 1 to *d*
 write(*Tab*)
2. *write*(*element*(*v*))
3. for each *w* \in *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

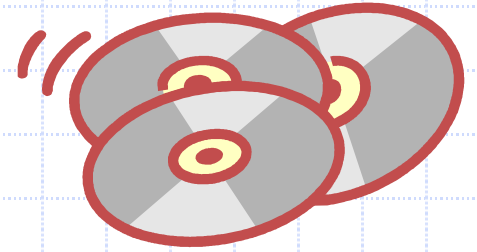
2.3 부동산

참고문헌

후위순회

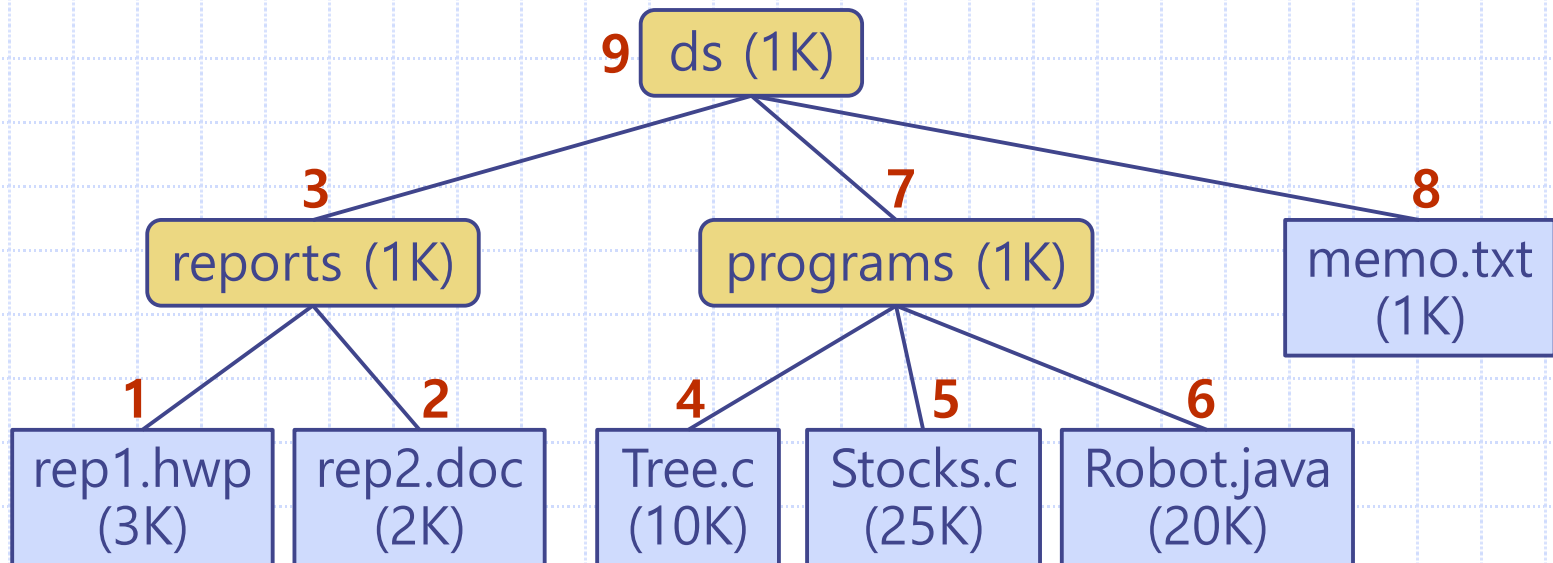
- ◆ 후위순회(postorder traversal): 노드를 그의 자손들보다 나중에 방문
- ◆ 실행시간: $O(n)$ – 단, n 은 트리 내 총 노드 수
- ◆ 응용
 - 계층적 파일시스템에서 폴더의 디스크 사용량 계산

```
Alg postOrder( $v$ )  
1. for each  $w \in \text{children}(v)$   
   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* \leftarrow 0
2. **for each** *w* \in *children*(*v*)
 sum \leftarrow *sum* + *diskUsage*(*w*)
3. **return** *sum* + *space*(*v*)

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

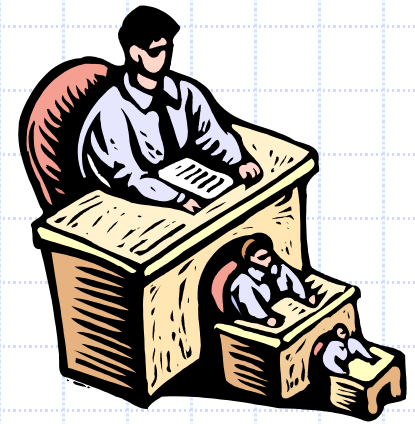
레벨순회

- ◆ **레벨(level) d 는** 트리의 같은 깊이 d 에 존재하는 모든 노드들의 집합을 나타낸다
 - 레벨 0에는 한 개의 노드, 루트만이 존재
- ◆ **레벨순회(levelorder traversal):** 큐를 이용하여 깊이 d 의 모든 노드들을 깊이 $d + 1$ 의 노드들에 앞서 방문
- ◆ **실행시간: $O(n)$**
- ◆ **응용**
 - 관료적 계층구조 인쇄

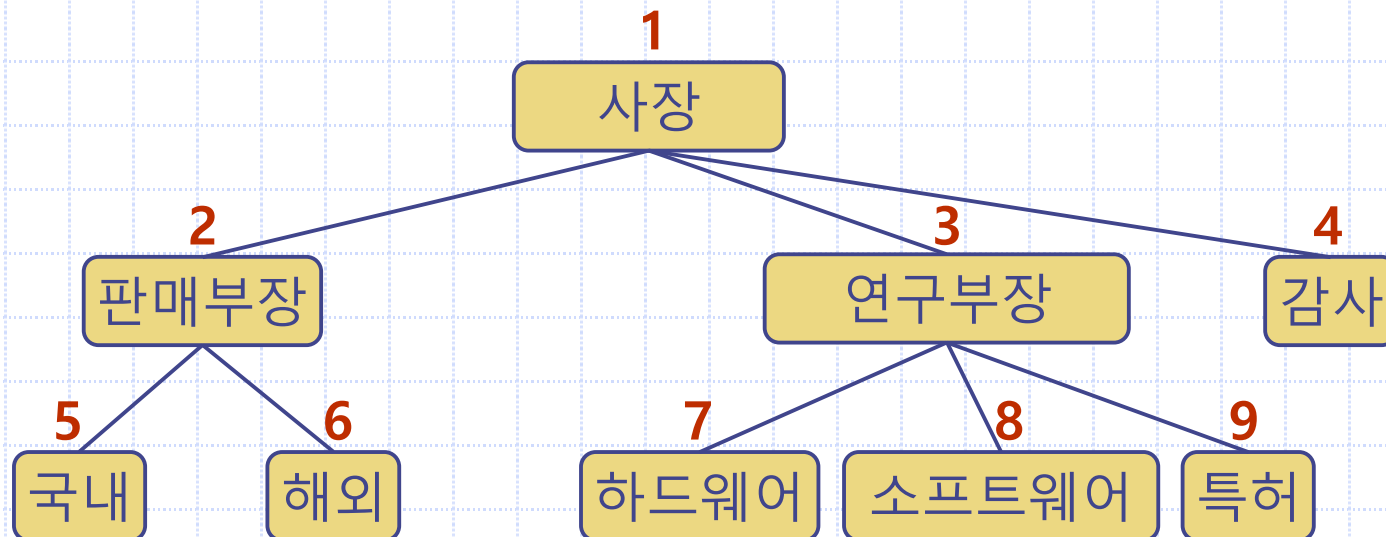
Alg **levelOrder**(v)

1. $Q \leftarrow \text{empty queue}$
2. $Q.enqueue(v)$
3. **while** ($!Q.isEmpty()$)
 - $v \leftarrow Q.dequeue()$
 - $visit(v)$
 - for each** $w \in children(v)$
 - $Q.enqueue(w)$
4. **return**

예: 계층구조 인쇄



- ◆ 회사의 계층구조를 상위 레벨에서 하위 레벨 순서로 인쇄하고자 한다



예: 계층구조 인쇄 (conti.)

Alg *printHierarchy(v)*

1. $Q \leftarrow \text{empty queue}$
2. $Q.\text{enqueue}(v)$
3. while ($!Q.\text{isEmpty}()$)
 - $v \leftarrow Q.\text{dequeue}()$
 - $\text{write}(\text{depth}(v), \text{element}(v))$
 - for each $w \in \text{children}(v)$
 - $Q.\text{enqueue}(w)$
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 모델링

◆ 전제: 적정이진트리로 구현

- 트리의 각 내부노드가 두 개의 자식을 가짐 - 왼쪽(left) 및 오른쪽(right) 자식
- 좌우 자식노드 가운데 하나가 비어 있는 경우라도 적정이진트리로 구현 가능

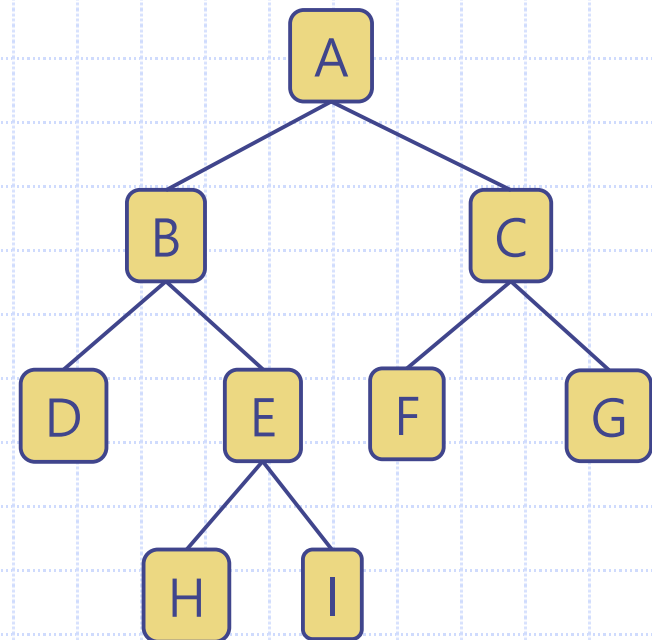
◆ 이진트리의 재귀적 정의

- 루트가 자식의 순서쌍을 가지며, 자식이 내부노드인 경우 이진트리다

◆ 전제: 이진트리는 비어있지 않다

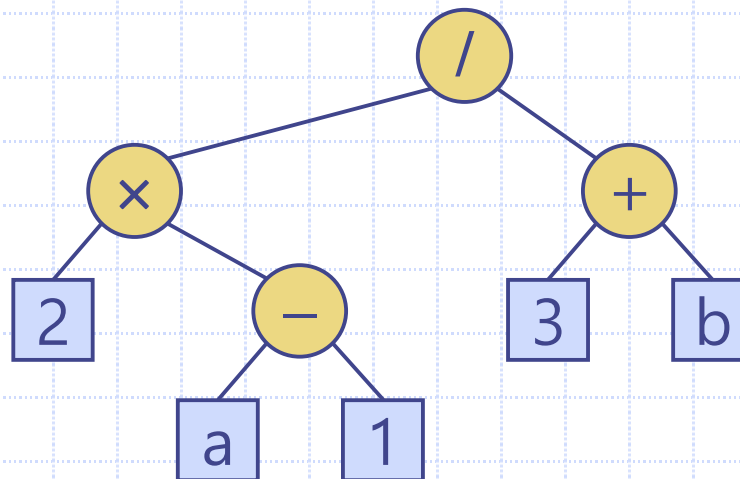
◆ 응용

- 수식 표현
- 의사결정 과정
- 검색



예: 수식 표현

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



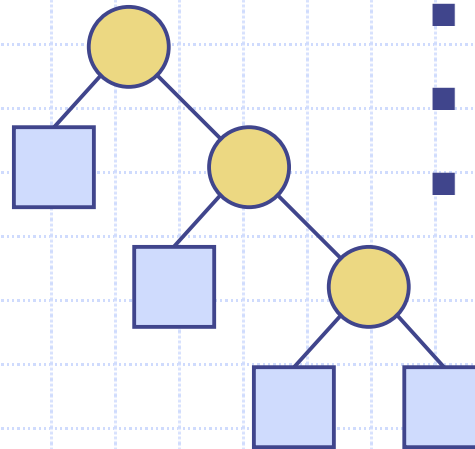
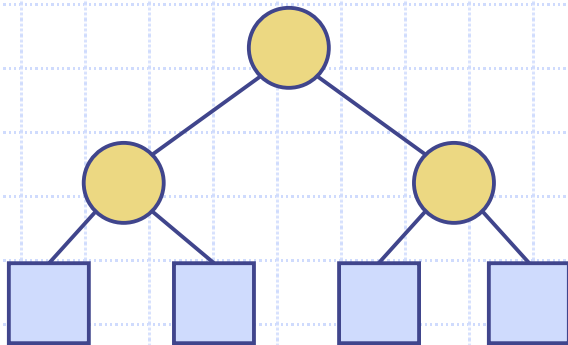
이진트리의 성질

◆ 표기

n 노드 수
 e 외부노드 수
 i 내부노드 수
 h 트리의 높이

◆ 성질

- $e = i + 1$
- $n = i + e = 2e - 1$
- $h \leq i$
- $h \leq (n - 1)/2$
- $e \leq 2^h$
- $h \geq \log_2 e$
- $h \geq \log_2(n + 1) - 1$



이진트리 ADT 메소드

◆ 이진트리 ADT는 트리 ADT의 확장이다 - 즉, 트리 ADT의 모든 메소드를 상속

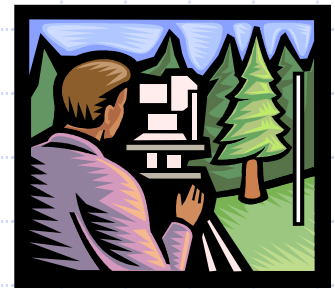
◆ 추가적인 메소드

- node leftChild(v)
- node rightChild(v)
- node sibling(v)

◆ 전제: 적정이진트리

- 만약 이진trie가 부적정하다면 작업시 오류 발생 가능

◆ 이진트리 ADT를 구현하는 데이터구조에 따라 추가적인 갱신 메소드(삽입, 삭제 등) 정의 가능



깊이와 높이

- ◆ 노드 v 의 **깊이**(depth)의 재귀적 정의
 - 만약 v 가 루트면, v 의 깊이는 0
 - 그렇지 않으면, v 의 깊이는 v 의 부모의 깊이 더하기 1
- ◆ 노드 v 의 **높이**(height)의 재귀적 정의
 - 만약 v 가 외부노드면, v 의 높이는 0
 - 그렇지 않으면, v 의 높이는 v 의 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  
     $h \leftarrow$   
        max(height(leftChild( $v$ )),  
            height(rightChild( $v$ )))  
    return 1 +  $h$ 
```


이진트리 순회

- ◆ 이진트리의 순회는 트리 순회의
특화(specialization)
- ◆ 선위순회(preorder traversal): 노드를 그의
왼쪽 및 오른쪽
부트리보다 앞서 방문
- ◆ 후위순회(postorder traversal): 노드를 그의
왼쪽 및 오른쪽
부트리보다 나중에 방문

Alg *binaryPreOrder(v)*

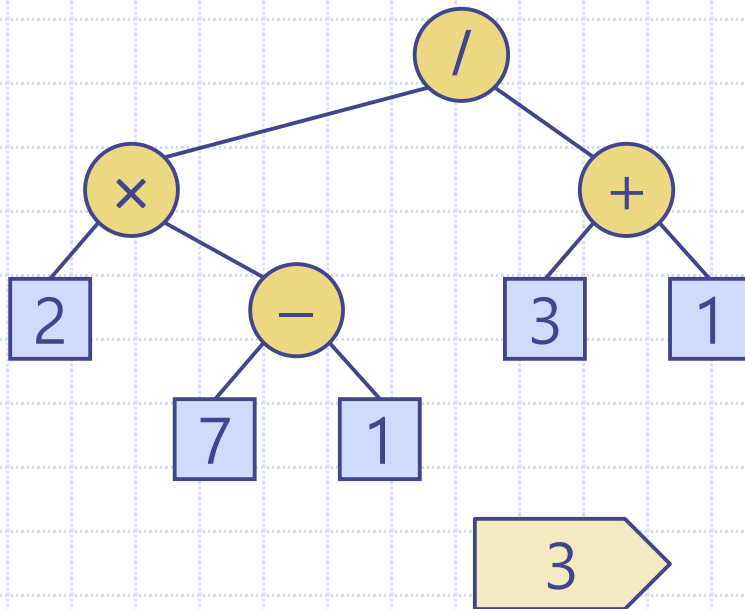
1. *visit(v)*
2. **if** (*isInternal(v)*)
 binaryPreOrder(leftChild(v))
 binaryPreOrder(rightChild(v))

Alg *binaryPostOrder(v)*

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

예: 수식 평가

- ◆ 수식트리에 저장된 수식을 평가하고자 한다



- ◆ 후위순회의 특화 – 단,
 - 부트리의 값을 반환하는 재귀적 메소드
 - 내부노드 방문 시에 부트리의 값들을 결합

Alg *evalExpr(v)*

1. if (*isExternal(v)*)

 return *element(v)*

else

$x \leftarrow evalExpr(leftChild(v))$

$y \leftarrow evalExpr(rightChild(v))$

$\diamond \leftarrow element(v)$

 return $x \diamond y$

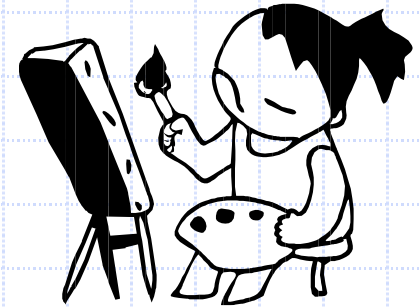
중위순회

- ◆ 중위순회(inorder traversal): 노드를 그의 왼쪽 부트리보다는 나중, 오른쪽 부트리보다는 앞서 방문
- ◆ 실행시간: $O(n)$ – 단, n 은 이진트리 내 총 노드 수
- ◆ 응용
 - 이진트리 그리기
 - 수식 인쇄

Alg *inOrder*(v)

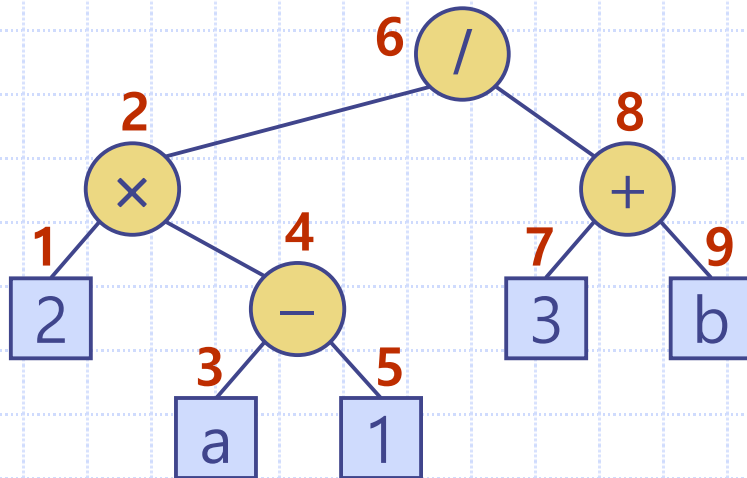
1. if (*isInternal*(v))
 inOrder(*leftChild*(v))
2. *visit*(v)
3. if (*isInternal*(v))
 inOrder(*rightChild*(v))

예: 이진트리 그리기



- ◆ 이진트리의 노드들을 xy 평면에 그리고자 한다
- ◆ 중위순회의 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. *rDBT*(v , 0)

Alg *rDBT*(v , $rank$)

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

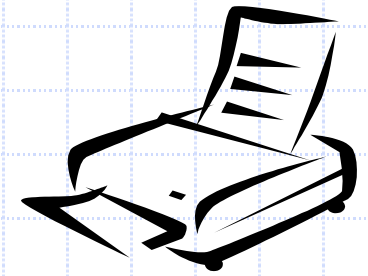
rDBT(*leftChild*(v), $rank$)

2. $rank \leftarrow rank + 1$

3. *drawNodeXY*(v , $rank$, *depth*(v))

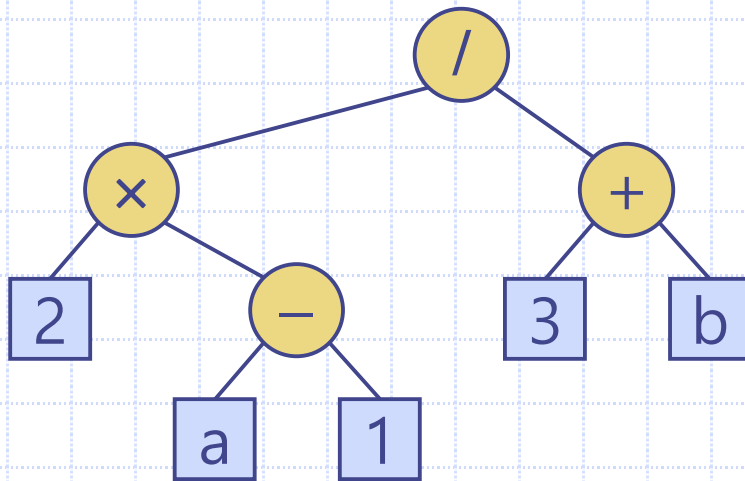
4. **if** (*isInternal*(v))

rDBT(*rightChild*(v), $rank$)



예: 수식 인쇄

- ◆ 수식트리로부터 괄호쳐진 수식을 인쇄하고자 한다



$((2 \times (a - 1)) / (3 + b))$

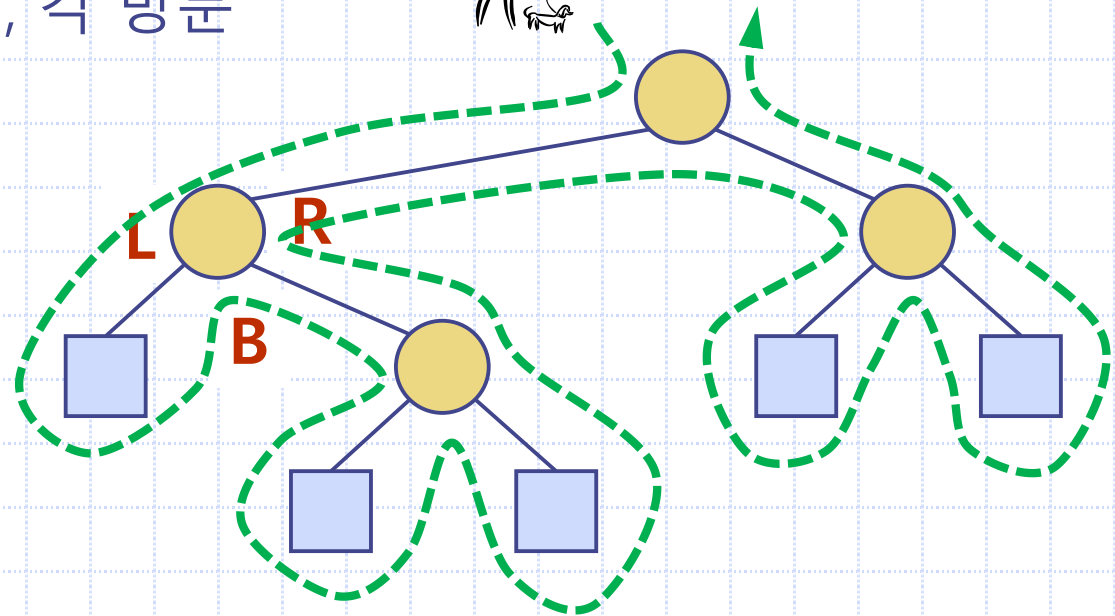
- ◆ 중위순회의 특화 - 단,
 - 노드를 방문시에 인쇄
 - 왼쪽 부트리를 순회하기 전에 '('를 인쇄
 - 오른쪽 부트리를 순회하고 나서 ')'를 인쇄

Alg **printExpr(v)**

1. if (*isInternal*(v))
 write('(')
 printExpr(leftChild(v))
2. write(element(v))
3. if (*isInternal*(v))
 printExpr(rightChild(v))
 write(')')

A black and white line drawing of a person walking a dog. The person is shown from the side, wearing a long-sleeved shirt and pants, and is holding a leash that is attached to a small dog. The dog is walking towards the left. The background is a light blue grid pattern.

- 

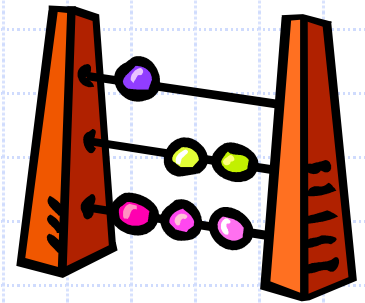


오일러 투어 순회 (conti.)

- ◆ 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 k 를 0으로 초기화한 후 오일러 투어를 시작
- ◆ 노드를 왼쪽에서 방문할 때마다 k 를 하나씩 증가
- ◆ 루트가 v 인 부트리의 크기는, v 를 왼쪽에서 방문했을 때의 k 값과 오른쪽에서 방문했을 때의 k 값의 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)

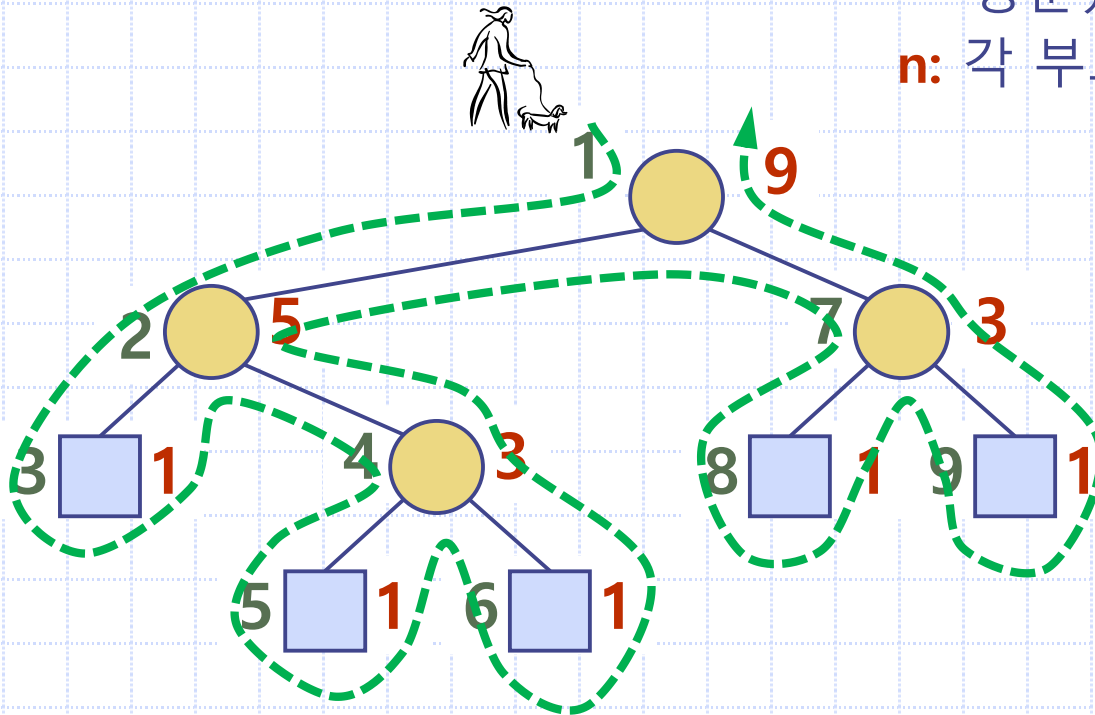
1. return

Alg *visitRight*(v)

1. $v.size \leftarrow k - v.kleft + 1$

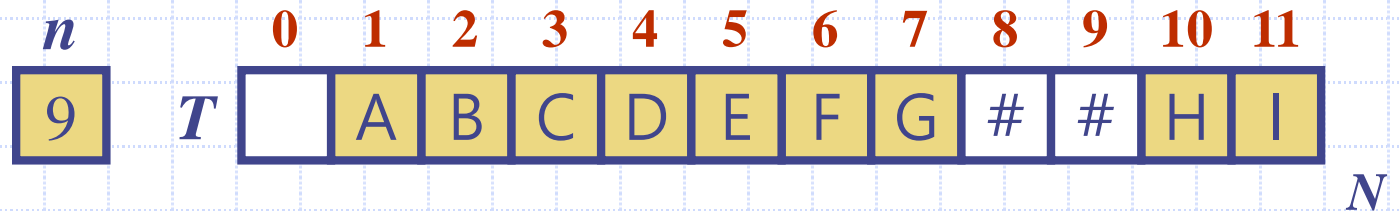
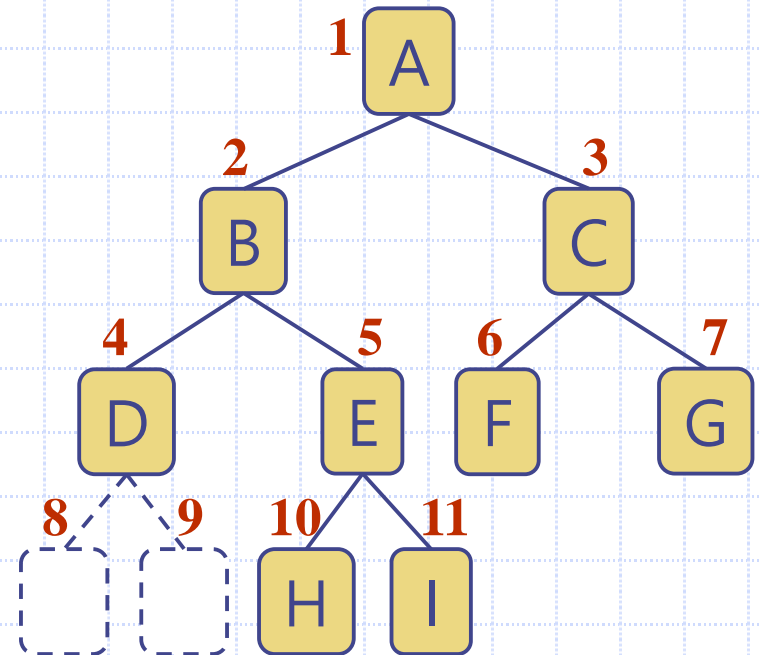
작동 원리

k: 각 노드를 왼쪽에서
방문했을 때의 카운터 값
n: 각 부트리의 크기



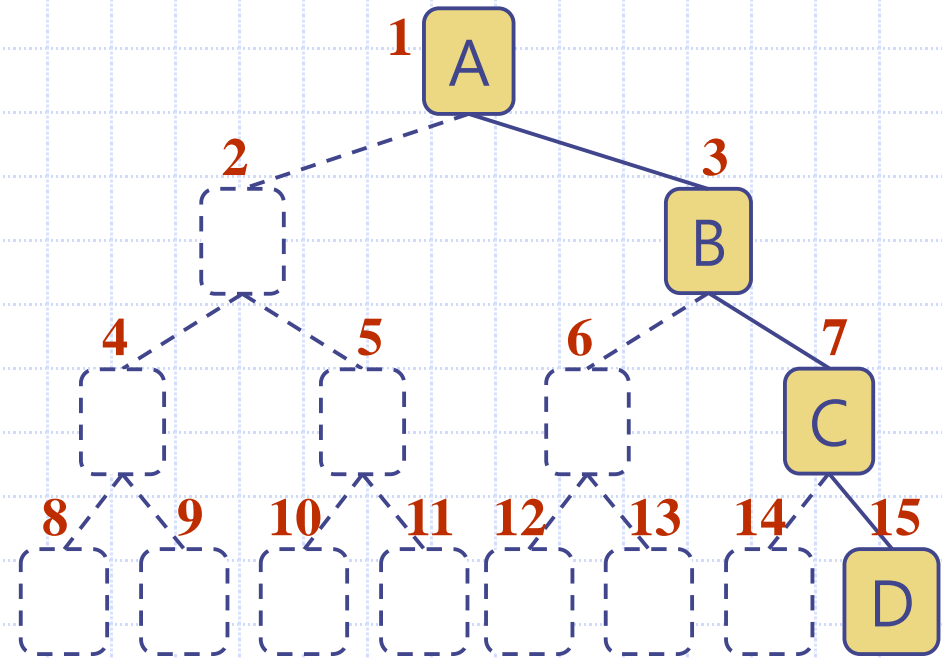
배열에 기초한 이진트리

- ◆ 1D 배열을 이용하여 이진트리 표현 가능
- ◆ 랭크 i 의 노드에 대해:
 - 왼쪽 자식의 위치는 순위 $2i$
 - 오른쪽 자식의 위치는 순위 $2i + 1$
 - 부모의 위치는 순위 $\lfloor i/2 \rfloor$
- ◆ 노드 간의 링크 저장 불필요
- ◆ 순위 0 셀은 미사용
- ◆ 비어 있는 셀은 특별값을 저장
 - 널마커(예: '#'), 또는
 - 널포인터(포인터배열인 경우)



최선과 최악의 경우

- ◆ MAX 를 노드 순위 중 최대값이라 하면, 배열크기 $N = MAX$
- ◆ 최선의 경우, $N = n$
(완전이진트리, complete binary tree)
- ◆ 최악의 경우, $N = 2^n - 1$
(단, 부적정이진트리 경우임)



n		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
4	T		A	#	B	#	#	#	C	#	#	#	#	#	#	#	D	N

배열에 기초한 이진트리 메소드

Alg *element*(v)

1. return $T[v]$

Alg *root*()

1. return 1

Alg *isRoot*(v)

1. return $v = 1$

Alg *parent*(v)

1. return $\lfloor v/2 \rfloor$

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 $(2v < N) \ \& \ (T[2v] \neq \text{Null})$

Alg *isExternal*(v)

1. return $(2v \geq N) \ \parallel \ (T[2v] = \text{Null})$

Alg *setElement*(v, e)

1. $T[v] \leftarrow e$
2. return e

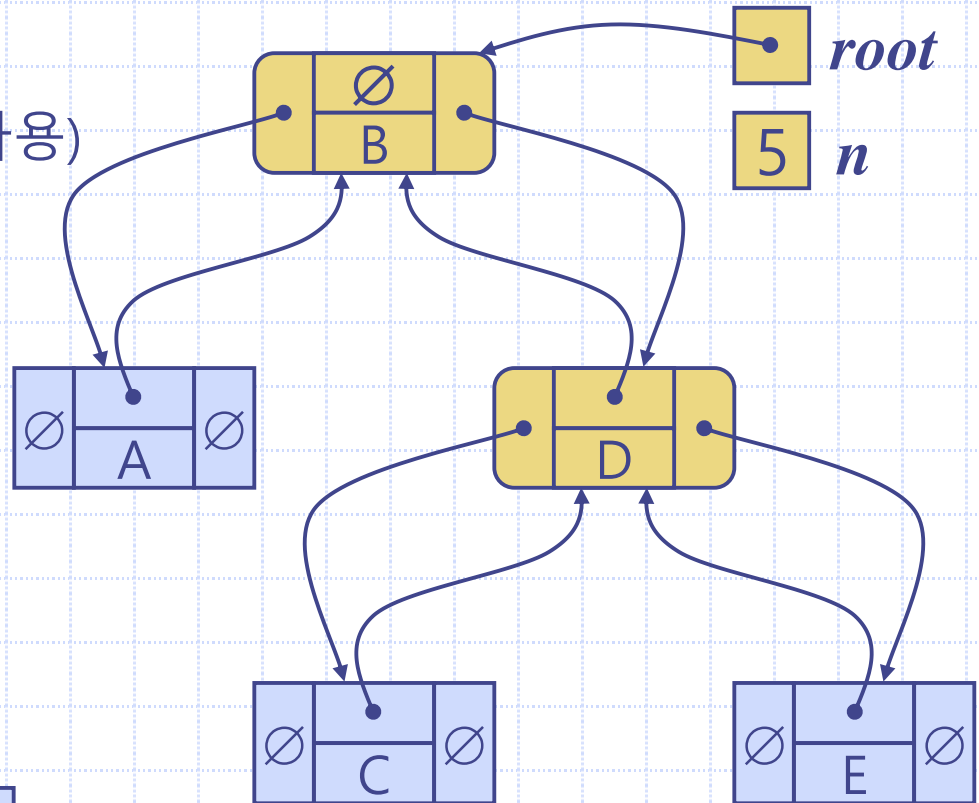
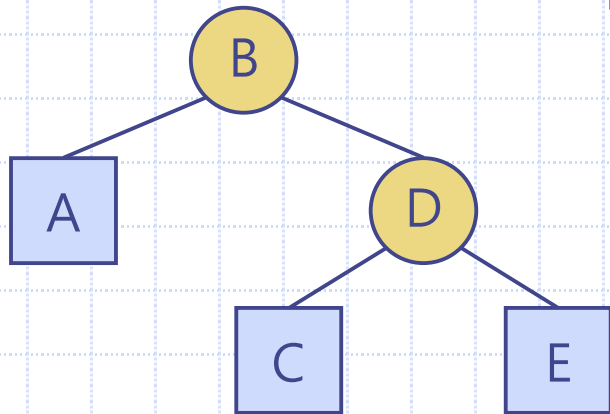
Alg *swapElements*(v, w)

1. $tmp \leftarrow T[v]$
2. $T[v] \leftarrow T[w]$
3. $T[w] \leftarrow tmp$
4. return

연결이진트리

◆ 노드 저장내용

- 원소
- 부모노드 (필요 시 사용)
- 왼쪽 자식노드
- 오른쪽 자식노드



연결이진트리 메소드

Alg *element*(*v*)

1. return *v.elem*

Alg *parent*(*v*)

1. return *v.parent*

Alg *isInternal*(*v*)

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

Alg *root*()

1. return *root*

Alg *leftChild*(*v*)

1. return *v.left*

Alg *isExternal*(*v*)

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

Alg *isRoot*(*v*)

1. return *v* = *root*

Alg *rightChild*(*v*)

1. return *v.right*

Alg *setElement*(*v*, *e*)

1. *v.elem* $\leftarrow e$

2. return *e*

Alg *sibling*(*v*)

1. *p* $\leftarrow v.parent$

2. if (*p.left* = *v*)

 return *p.right*

 else

 return *p.left*

Alg *swapElements*(*v*, *w*)

1. *tmp* $\leftarrow v.elem$

2. *v.elem* $\leftarrow w.elem$

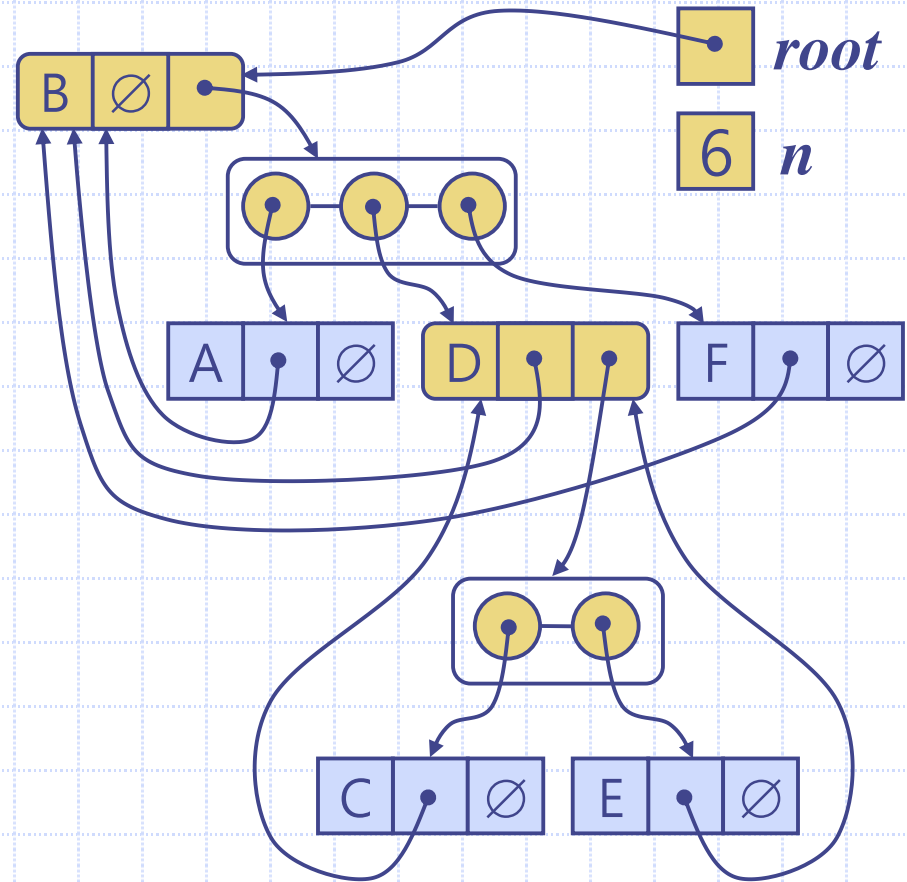
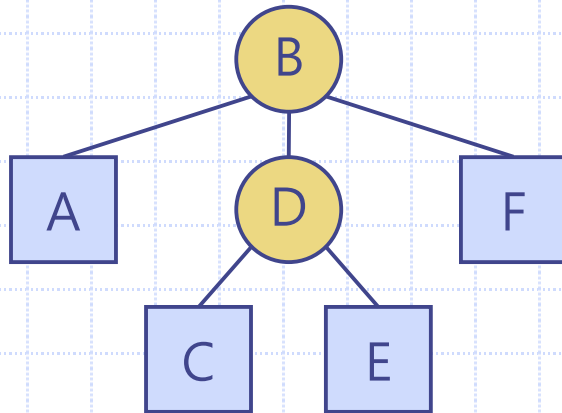
3. *w.elem* $\leftarrow tmp$

4. return

연결트리 (Ver.1)

◆ 노드 저장내용

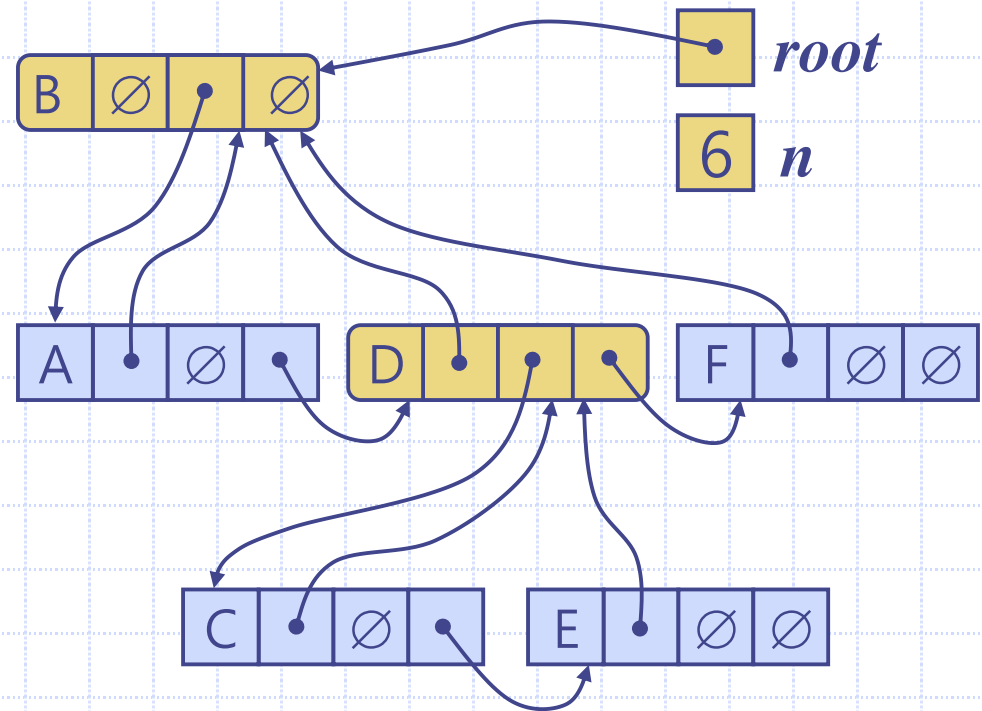
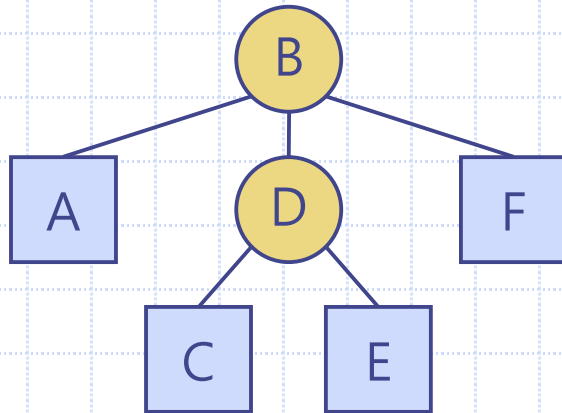
- 원소
- 부모노드
- 자식노드들의 리스트



연결트리 (Ver.2)

◆ 노드 저장내용

- 원소
- 부모노드
- 첫째 자식노드
- 바로 아래 동생노드



연결트리 메소드 (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 *setElement*(*v*, *e*)

1. *v.elem* $\leftarrow e$

2. return *e*

Alg *swapElements*(*v*, *w*)

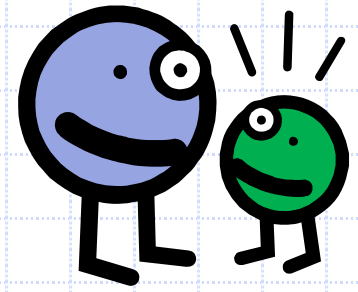
1. *tmp* $\leftarrow v.elem$

2. *v.elem* $\leftarrow w.elem$

3. *w.elem* $\leftarrow tmp$

4. return

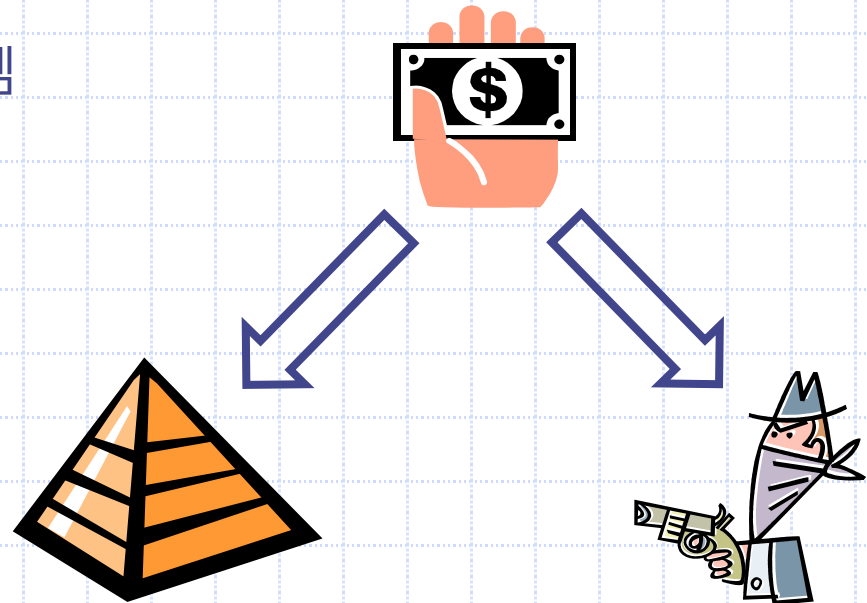
성능



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
setElement, swapElements	1	1	1

응용문제

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



응용문제: 계승자

- ◆ 이진트리 T 의 노드 v 에 대한 아래의 일반 알고리즘들을 작성하라
 - **preOrderSucc**(v): 선위순회 계승자(즉, T 를 선위순회할 경우 노드 v 직후에 방문되는 노드)를 반환
 - **inOrderSucc**(v): 중위순회 계승자(즉, T 를 중위순회할 경우 노드 v 직후에 방문되는 노드)를 반환
 - **postOrderSucc**(v): 후위순회 계승자(즉, T 를 후위순회할 경우 노드 v 직후에 방문되는 노드)를 반환
- ◆ 주의: 계승자 노드가 존재하지 않을 경우 **invalidNodeException**을 발령해야 함
- ◆ 힌트: 이진트리 ADT의 기본 메소드 사용 가능

해결

Alg *preOrderSucc(v)*

input node v

output node

1. **if** (*isInternal*(v))
 return *leftChild*(v)
2. $p \leftarrow \text{parent}(v)$
3. **while** (*leftChild*(p) $\neq v$)
 if (*isRoot*(p))
 invalidNodeException()
 $v \leftarrow p$
 $p \leftarrow \text{parent}(p)$
4. **return** *rightChild*(p)

Alg *inOrderSucc(v)*

input node v

output node

1. **if** (*isInternal*(v))
 $v \leftarrow \text{rightChild}(v)$
 while (*isInternal*(v))
 $v \leftarrow \text{leftChild}(v)$
 return v
2. $p \leftarrow \text{parent}(v)$
3. **while** (*leftChild*(p) $\neq v$)
 if (*isRoot*(p))
 invalidNodeException()
 $v \leftarrow p$
 $p \leftarrow \text{parent}(p)$
4. **return** p

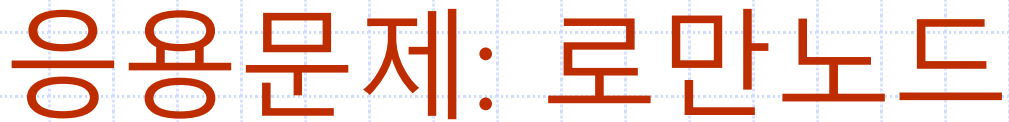
해결 (conti.)

Alg *postOrderSucc*(v)

input node v

output node

1. **if** (*isRoot*(v))
 invalidNodeException()
2. $p \leftarrow \text{parent}(v)$
3. **if** (*rightChild*(p) = v)
 return p
4. $v \leftarrow \text{rightChild}(p)$
5. **while** (*!isExternal*(v))
 $v \leftarrow \text{leftChild}(v)$
6. **return** v



- 진트리다
의 노드 v
브트리의 크기
한다

해결

- ◆ **romanSize(v)**는 노드 v 와 v 의 모든 후손이 로마이면 루트를 v 로 하는 부트리의 크기를, 그렇지 않으면 0을 반환
- ◆ 어느 노드도 한 번 이상 방문하지 않으므로 $O(n)$ 시간에 실행

Alg *romanSize(v)*

1. **if** (*isExternal(v)*) {roman}
 return 1
2. $l \leftarrow \text{romanSize}(\text{leftChild}(v))$
3. **if** ($l = 0$)
 return 0
4. $r \leftarrow \text{romanSize}(\text{rightChild}(v))$
5. **if** ($(r > 0) \ \& \ (|l - r| \leq 5)$)
 return $l + r + 1$ {roman}
 else
 return 0

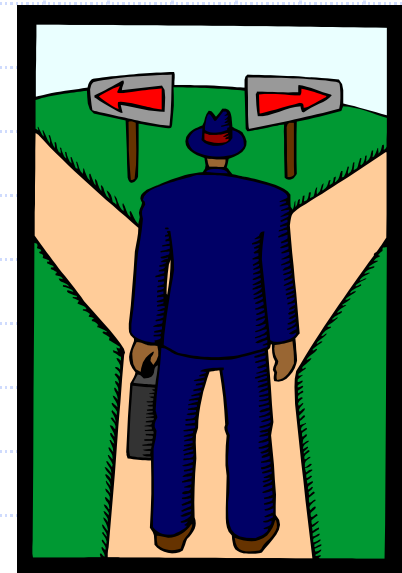
응용문제: 양자택일식 문답시스템

- ◆ **대답에 따라 다양한 결정 가운데 하나를 제공하는 양자택일식 문답시스템을 구축하고자 한다**

- 예/아니오 응답을 요하는 질문들
- 결정들

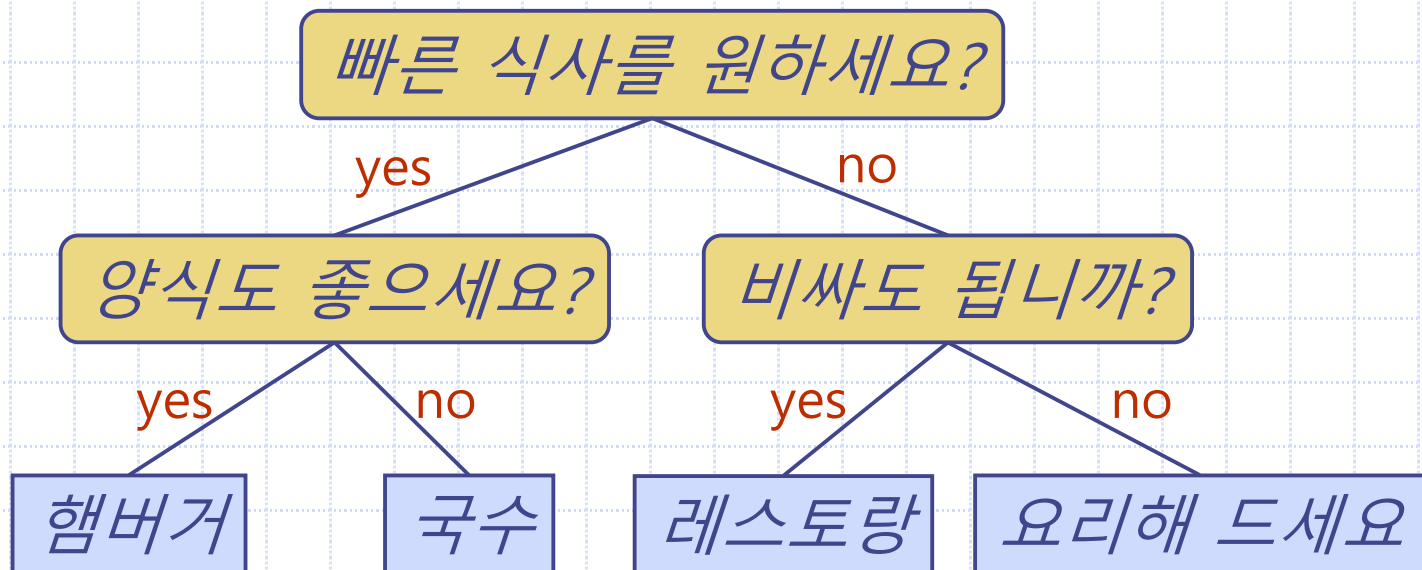
- ◆ **예**

- 스무고개
- 해결사(troubleshooters)
- 분류기(classifiers)
- 추천기(recommenders)



해결: 결정트리

- ◆ 결정트리(decision tree): 의사결정 과정과 연관된 이진트리
 - 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) {root}
3. **return**

Alg *makeExternalNode(i)*

1. *write*(“Enter decision:”)
2. $T[i] \leftarrow \text{read}()$
3. **if** ($2i < N$)
 $T[2i], T[2i + 1] \leftarrow \text{Null}$
4. **return**

Alg *makeInternalNode(i)*

1. *write*(“Enter question:”)
2. $T[i] \leftarrow \text{read}()$
3. *write*(“Question if yes to”, $T[i]$, “?”)
4. **if** ($\text{read}() = \text{“yes”}$)
 makeInternalNode($2i$)
 else
 makeExternalNode($2i$)
5. *write*(“Question if no to”, $T[i]$, “?”)
6. **if** ($\text{read}() = \text{“yes”}$)
 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. $v \leftarrow \text{getnode}()$
2. *write*("Enter *decision*:")
3. $v.\text{elem} \leftarrow \text{read}()$
4. $v.\text{left}, v.\text{right} \leftarrow \emptyset$
5. **return** v

Alg *makeInternalNode()*

1. $v \leftarrow \text{getnode}()$
2. *write*("Enter *question*:")
3. $v.\text{elem} \leftarrow \text{read}()$
4. *write*("Question if *yes to*", $v.\text{elem}$, "?")
5. **if** ($\text{read}() = \text{"yes"}$)
 $v.\text{left} \leftarrow \text{makeInternalNode}()$
 else
 $v.\text{left} \leftarrow \text{makeExternalNode}()$
6. *write*("Question if *no to*", $v.\text{elem}$, "?")
7. **if** ($\text{read}() = \text{"yes"}$)
 $v.\text{right} \leftarrow \text{makeInternalNode}()$
 else
 $v.\text{right} \leftarrow \text{makeExternalNode}()$
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