트리 (Tree)

강 지 훈

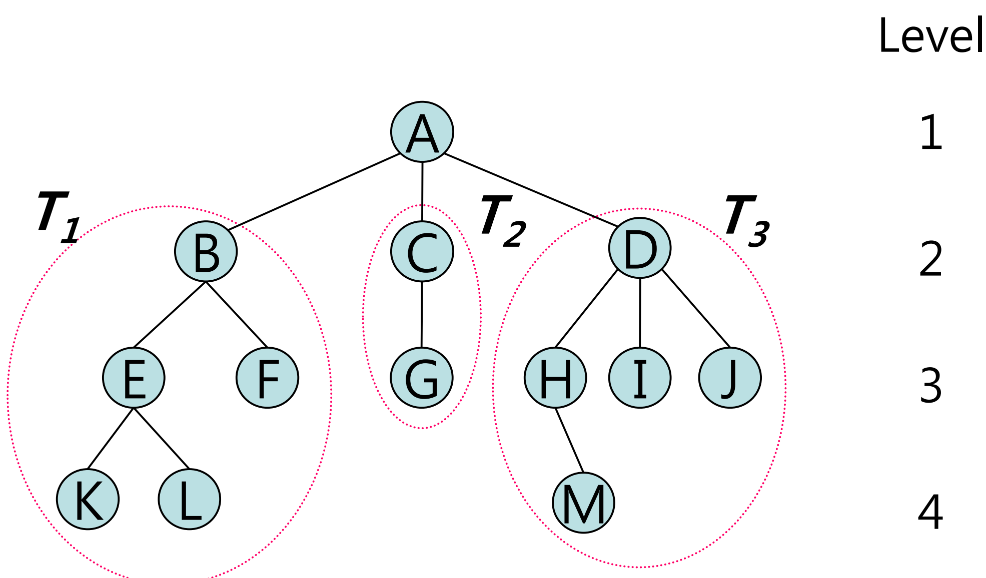
jhkang@cnu.ac.kr

충남대학교 컴퓨터공학과

트리 (Tree)

❑ 트리

- 트리(tree) 는 하나 이상의 노드(node)로 구성되는 유한집합으로, 다음의 조건을 만족해야 한다:
  1. 루트(root) 라 부르는 특별히 지정된 노드가 하나 있다.
  2. 나머지 노드들은 n 개 ($n \ge 0$)의 노드가 서로 중복되지 않는 집합 (disjoint sets) $T_1, \ldots, T_n$ 으로 나누어지며, 이 각각의 노드 집합은 트리이어야 한다.
     $T_1, \ldots, T_n$ 이들을 루트의 부트리(subtrees) 라 한다.
- 재귀적으로 정의: 부트리는 다시 트리로 정의되고 있다.



용어 [1]

- 노드(node): 정보 항목 (item) 과 다른 노드로의 가지 (branches)로 구성되는 트리의 구성 단위
- 노드의 디그리 (degree of a node): 그 노드의 서브트리의 개수
  - 예: degree(A) = 3, degree(M) = 0
- 트리의 디그리 (degree of a tree): 트리에 있는 노드들의 디그리 중에서 가장 큰 값
- 잎(leaf) (또는 끝(terminal)) : 디그리가 0인 노드
  - Example: K, L, F, G, M, I, J
- 노드 X의 자식 (child of a node X): X의 서브트리의 루트
- 부모(parent) : 자식의 역관계
  - 예: B 는 E 와 F의 부모.

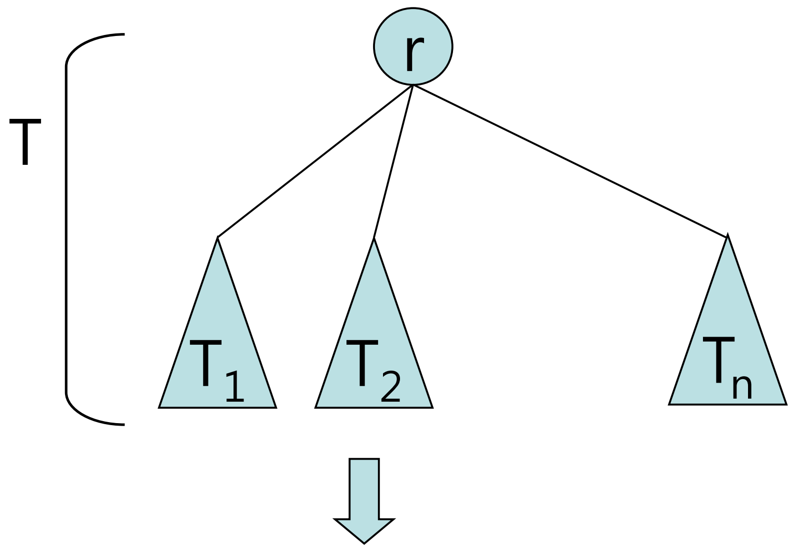
용어 [2]

- 형제(sibling): 부도가 동일한 노드
  - Example: H, I, J are siblings
- 노드 X의 조상(ancestor): 노드 X에서부터 루트까지의 경로를 따라 존재하는 노드
  - 예: M의 조상은 A, D, H 이다.
- 레벨(level):
  - 루트의 레벨은 1 이다.
  - 루트의 자식의 레벨은 2 이다.
  - 노드 X의 레벨이 k 이면, X의 자식의 레벨은 (k+1) 이다.
- 높이(height) (또는 깊이(depth)): 트리의 최대 레벨
  - 예:
    - M의 레벨 = 4 : (트리에서 최대값)
    - 트리의 높이 = 4

트리의 표현

❑ 리스트를 이용한 표현

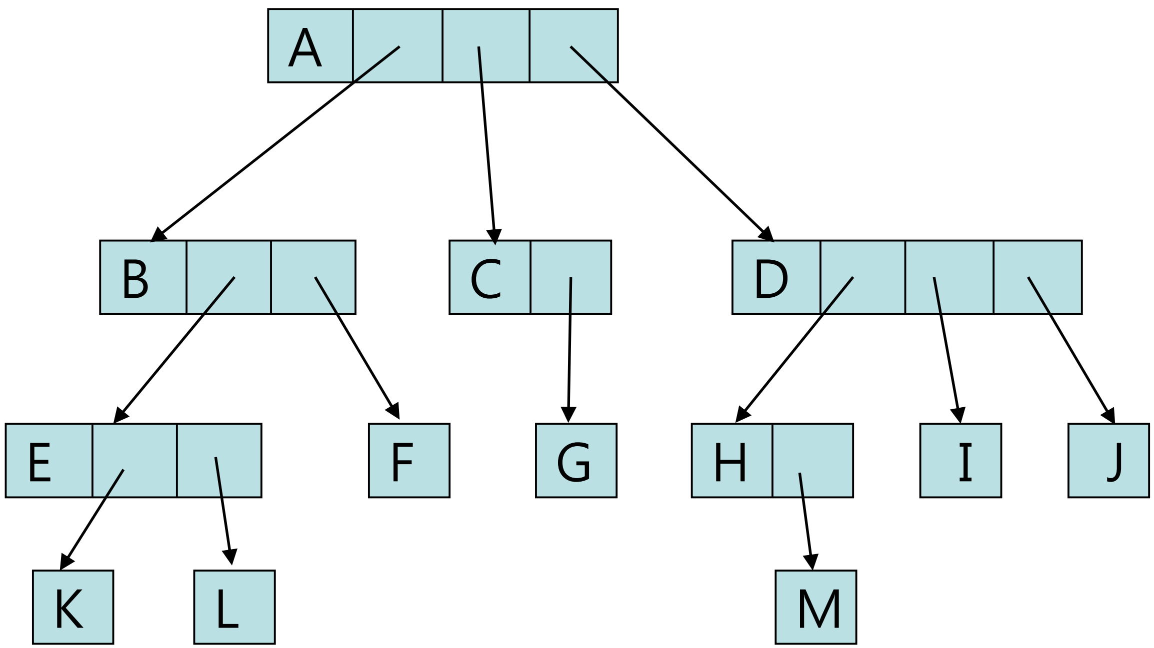
- 부트리(subtree)들을 리스트로 표현



$$(T) = (r\ (T_1, T_2, \ldots, T_n))$$

❑ 리스트를 이용한 표현

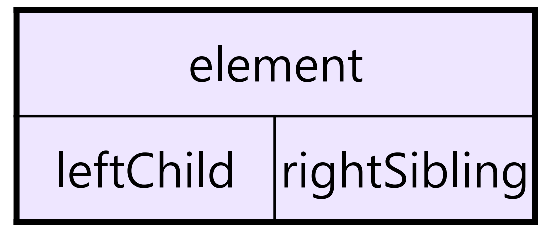
- 예: 가변길이 노드(variable length nodes)를 사용하여 리스트를 표현
  - (A (B (E (K, L), F), C (G), D (H (M), I, J)))

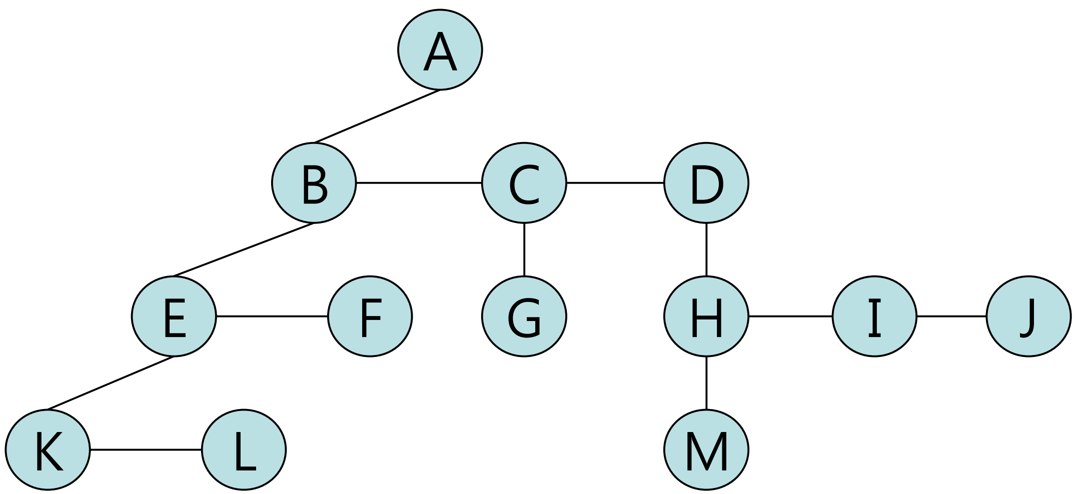


❑ 왼쪽 자식-오른쪽 형제 (Left Child-Right Sibling) 표현

- 고정길이 노드 사용
- 각 노드는 3 개의 속성이 필요

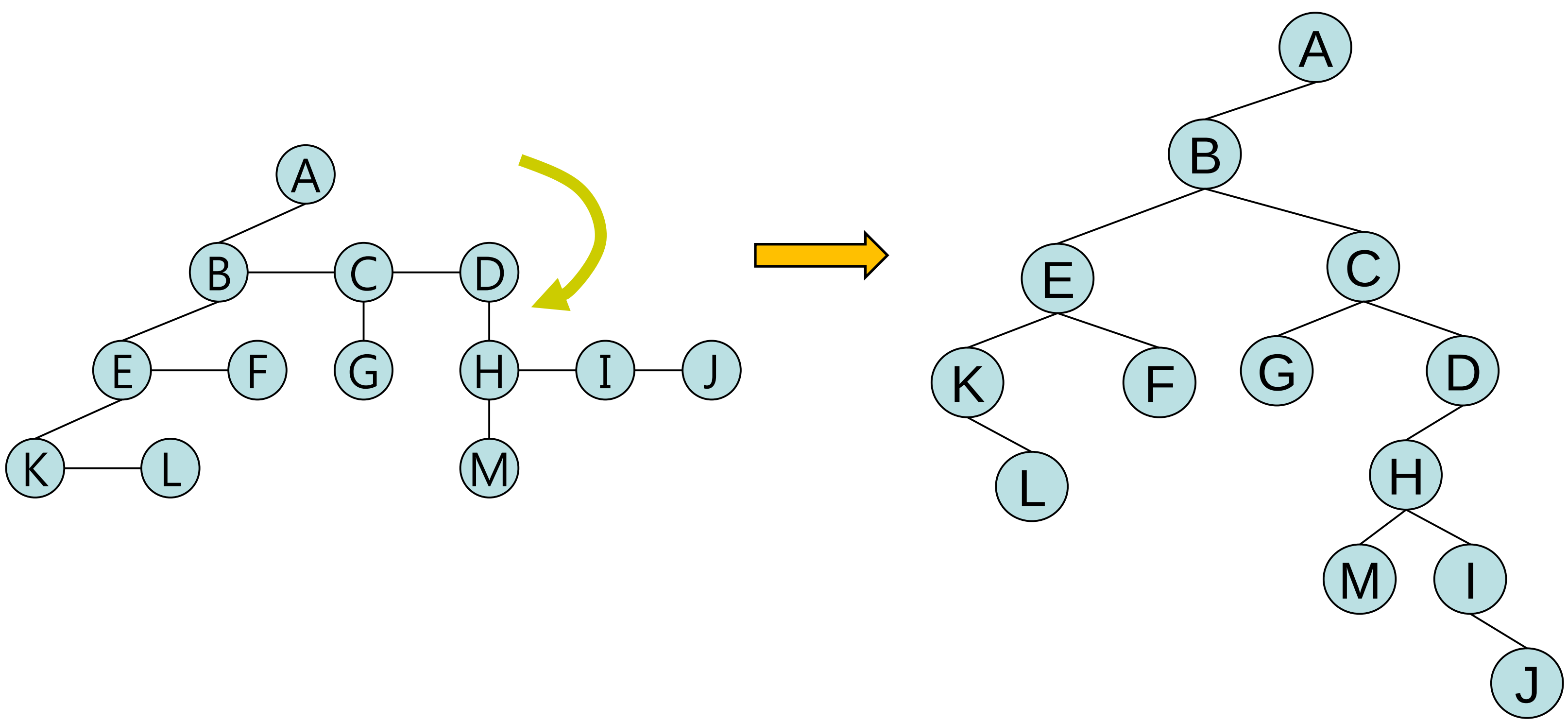
```
public  class TreeNode {
    private Element     element ;
    private Node        leftChild ;
    private Node        rightSibling ;
    ......
}
```





이진 트리로 표현

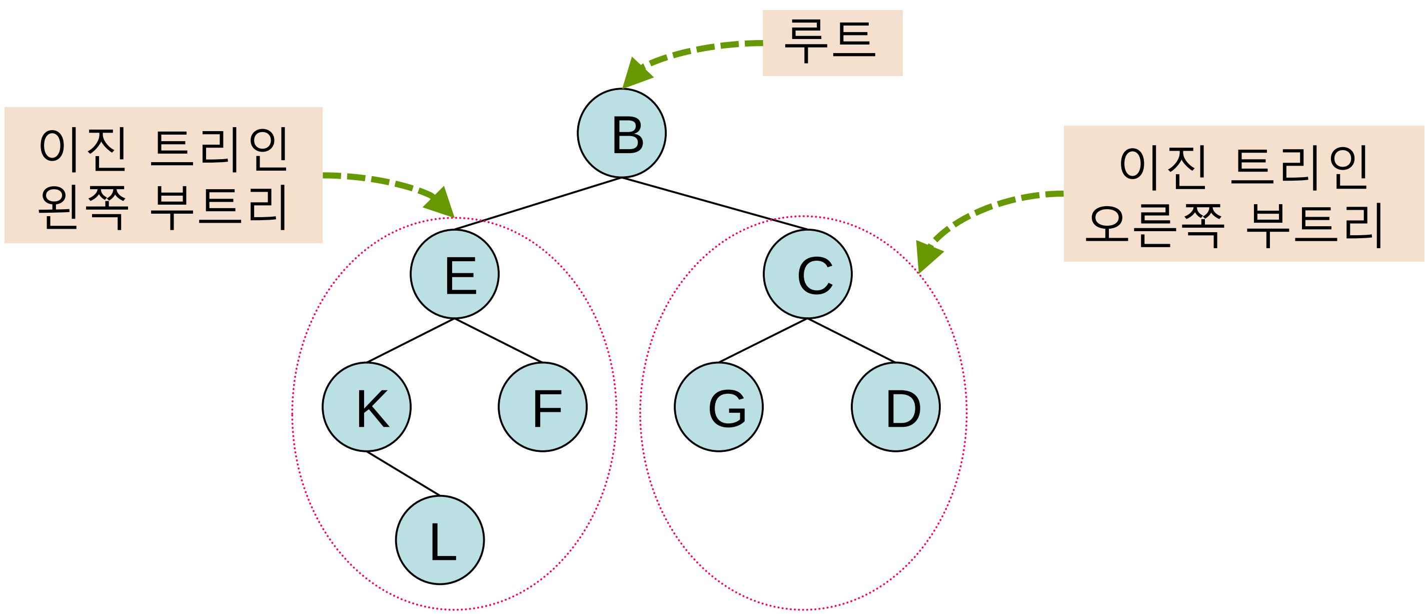
- Left child-right sibling 트리를 시계방향으로 45도 회전시킨다.
- 결국 이진트리(binary tree)가 된다.



이진 트리(Binary Tree)

이진트리 (Binary Trees)

- 이진트리 (binary tree) 는 노드의 유한집합으로 다음의 조건을 만족해야 한다:
  1) 비어 있거나, 또는
  2) 루트(root)와 두개의 서로 겹치지 않는 이진트리로 구성되며, 각각 왼쪽부트리(left subtree), 오른쪽부트리(right subtree)라 한다.

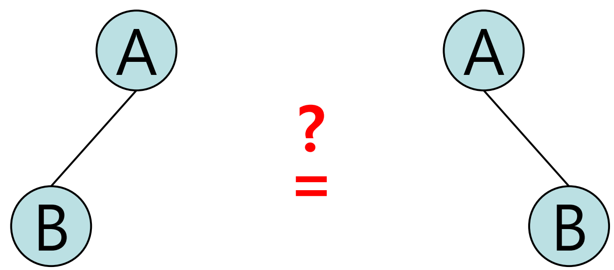


- 이진 트리에서도 트리의 용어를 그대로 사용

트리와 이진트리의 차이점?

서브트리의 순서

- 이진 트리에는 서브트리의 순서가 있다. 그러나 트리에는 순서가 없다.



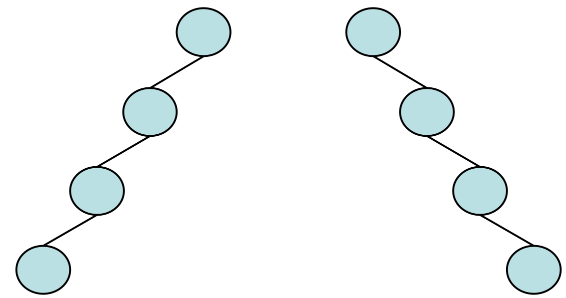
- 이들이 트리라면, 이 둘은 같다.
- 이들이 이진트리라면, 이 둘은 다르다.

노드의 최소 개수

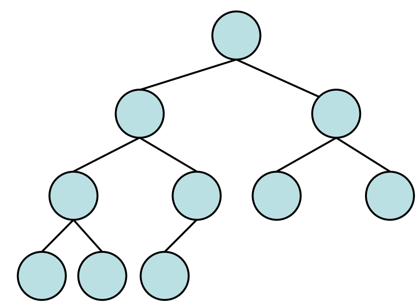
- 트리에는 적어도 하나의 노드가 존재한다.
- 이진트리에는 노드가 하나도 없을 수도 있다.

❑ 특수한 이진 트리

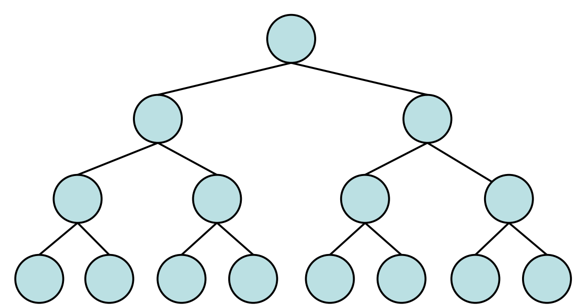
- 치우친 이진트리
  (Skewed binary tree
  / Degenerate binary tree)



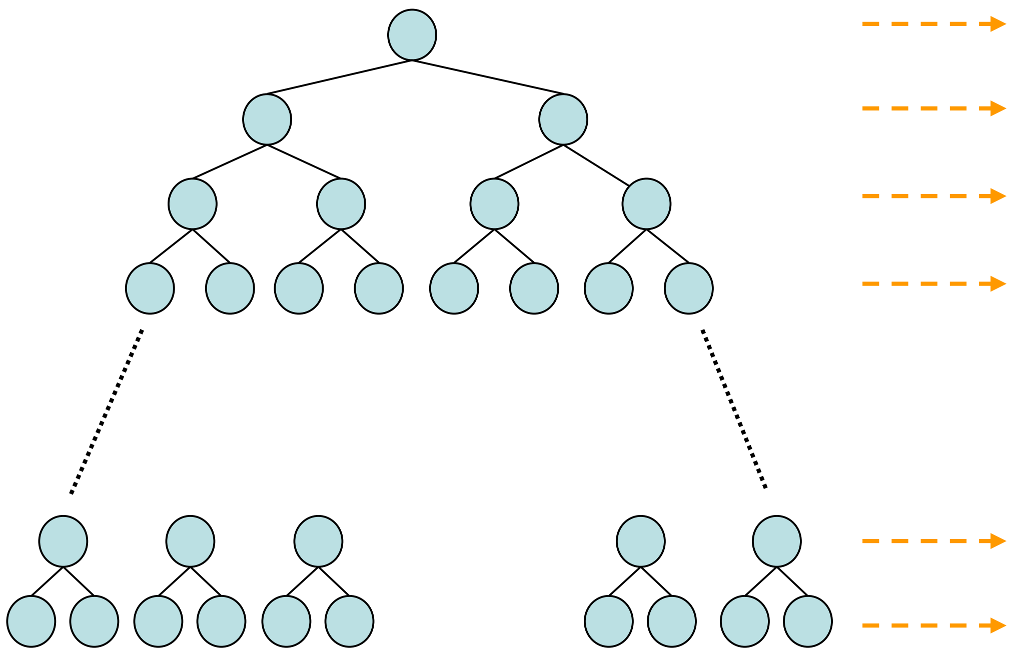
- 완전이진트리
  (Complete binary tree)



- 꽉찬 이진트리
  (Full binary tree)



레벨과 노드 수와의 관계



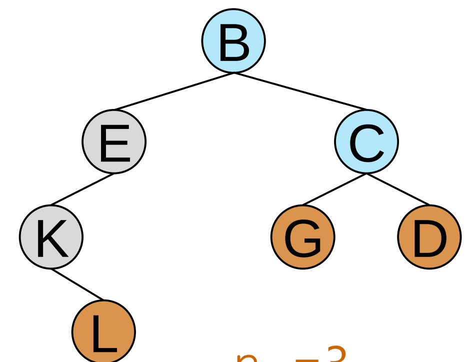
Level i	2^{i-1}	$2^i - 1$
1	2^0	2^1-1
2	2^1	2^2-1
3	2^2	2^3-1
4	2^3	2^4-1
k-1	2^{k-2}	$2^{k-1}-1$
k	2^{k-1}	2^k-1

- 레벨 i 에 있을 수 있는 노드의 최개 개수
  $\Rightarrow 2^{i-1}\ (i \geq 1)$

- 깊이가 k 인 이진트리가 가질 수 있는 노드의 최대 개수
  $\Rightarrow 2^0 + 2^1 + \cdots + 2^{k-1} = 2^k - 1\ (k \geq 1)$

❑ 잎 노드 수와 디그리 2인 노드 수와의 관계

- Relationship between number of leaf nodes and nodes of degree 2.
  - Let n_0 : # of leaf nodes, and
    n_2 : # of nodes of degree 2.
    Then, $n_0 = n_2 + 1$.



$n_0 = 3$
$n_1 = 2$
$n_2 = 2$

(Proof)

Let n: # of nodes in the tree, and
n_1: # of nodes of degree 1.

Then $n = n_0 + n_1 + n_2$. ......[1]

Let B: # of branches in the tree.

Then

$n = B + 1.$

$B = 1 \bullet n_1 + 2 \bullet n_2$.

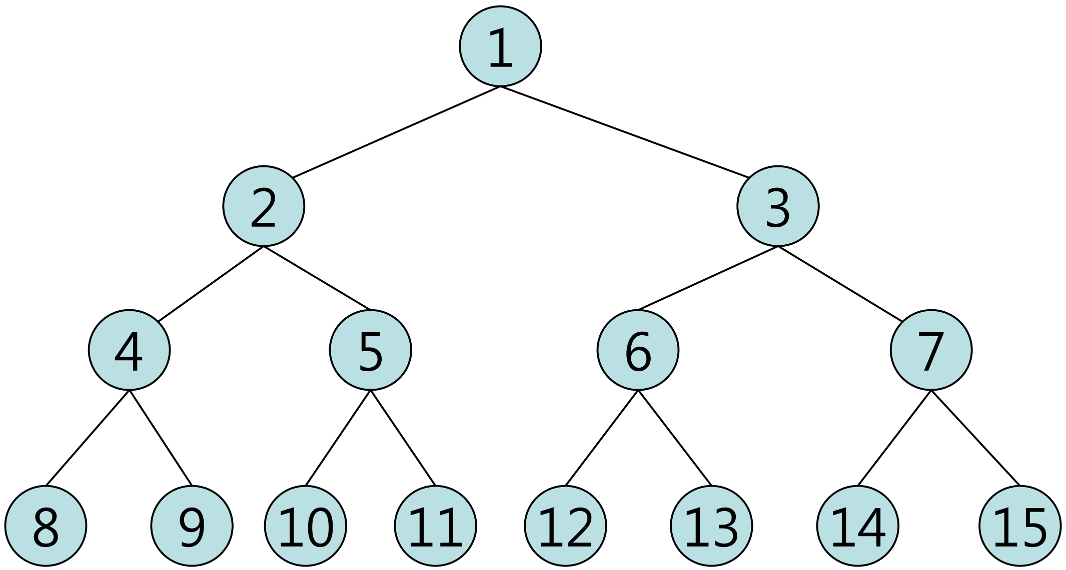
So, $n = 1 \bullet n_1 + 2 \bullet n_2 + 1$ ......[2]

Since [1]=[2], $n = n_0 + n_1 + n_2 = 1 \bullet n_1 + 2 \bullet n_2 + 1$.

Therefore, $n_0 = n_2 + 1$. ■

꽉찬 이진트리 (Full Binary Trees)

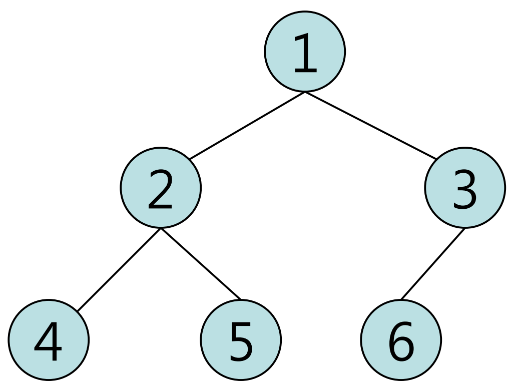
- A full binary tree of Depth k :
  A binary tree of depth k having 2^k -1 nodes, $(k \geq 0)$
  - Example: Full binary tree of depth 4.



완전 이진트리 (Complete binary trees)

- 노드가 n 개인 완전 이진트리:
  - (n개 이상의 노드를 가지고 있는) 꽉찬 이진트리에서 노드 번호가 1 번부터 n번까지의 노드를 가지고 있는 트리

- 예: 노드가 6 개인 완전 이진트리



- 꽉찬 이진트리는 완전 이진트리이다.

Class “BinaryTree”

Class “BinaryTree”의 공개함수

- BinaryTree 객체 사용법을 Java로 구체적으로 표현
  - // 공개함수
  - public BinaryTree() ;
  - public BinaryTree (Element aRootElement, BinaryTree<Element> aLeftTree, BinaryTree<Element> aRightTree) ;
  - public boolean isEmpty() ;
  - public int height() ;
  - public int size() ;
  - public Element rootElement() ;
  - public BinaryTree<Element> leftSubtree() ;
  - public BinaryTree<Element> rightSubtree() ;
  - public void setTree (Element givenRootElement, BinaryTree<Element> aLeftTree, BinaryTree<Element> aRightTree) ;

이진트리의 표현

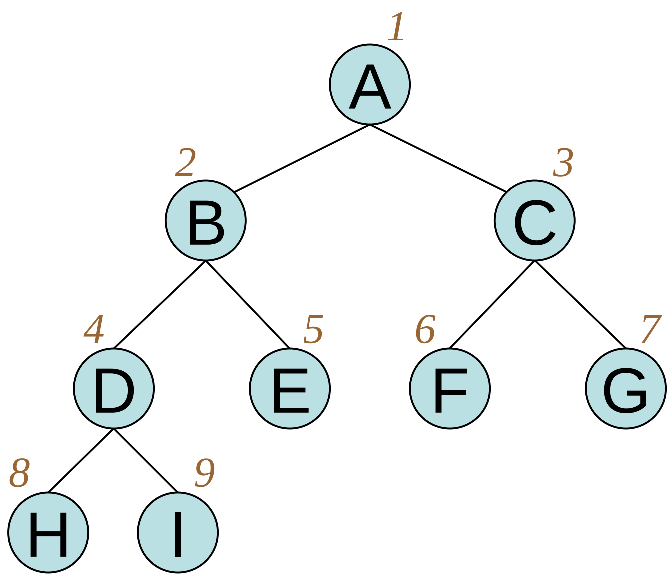
배열을 이용

연결 체인을 이용

배열을 이용한 표현

❑ 배열을 이용한 표현 [1]

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
	A	B	C	D	E	F	G	H	I



$\text{parent}(6) \rightarrow \lfloor 6/2 \rfloor = 3$

$\text{parent}(9) \rightarrow \lfloor 9/2 \rfloor = 4$

$\text{parent}(1) \rightarrow$ ***none***

$\text{lchild}(2) \rightarrow 2 \cdot 2 = 4$

$\text{lchild}(3) \rightarrow 2 \cdot 3 = 6$

$\text{lchild}(7) \rightarrow$ ***none***

$\text{rchild}(2) \rightarrow 2 \cdot 2 + 1 = 5$

$\text{rchild}(3) \rightarrow 2 \cdot 3 + 1 = 7$

$\text{rchild}(7) \rightarrow =$ ***none***

노드 번호와 배열 인덱스의 관계

- 노드가 n 개인 완전 이진트리를 배열을 이용하여 표현하면 다음의 관계가 성립한다:

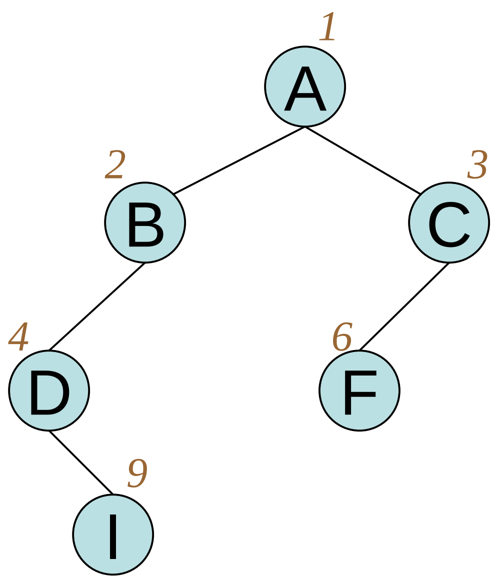
  배열 인덱스가 i ($1 \leq i \leq n$)인 노드 x 에 대해,

  1. $i \neq 1$ 이면, x 의 부모는 $\lfloor i/2 \rfloor$ 에 존재
     $i = 1$ 이면, x 는 루트 노드이므로, 부모가 존재하지 않음
  2. $2 \cdot i \leq n$ 이라면, x의 왼쪽 자식은 $2 \cdot i$ 에 존재
     $2 \cdot i > n$ 이라면, x는 왼쪽 자식이 없음
  3. $2 \cdot i + 1 \leq n$ 이라면, x의 오른쪽 자식은 $2 \cdot i + 1$ 에 존재
     $2 \cdot i + 1 > n$ 이라면, x는 오른쪽 자식이 없음

- 노드가 n 개인 완전 이진트리의 깊이: $\lfloor \log_2 n \rfloor + 1$

일반 이진트리를 배열로 표현 [1]

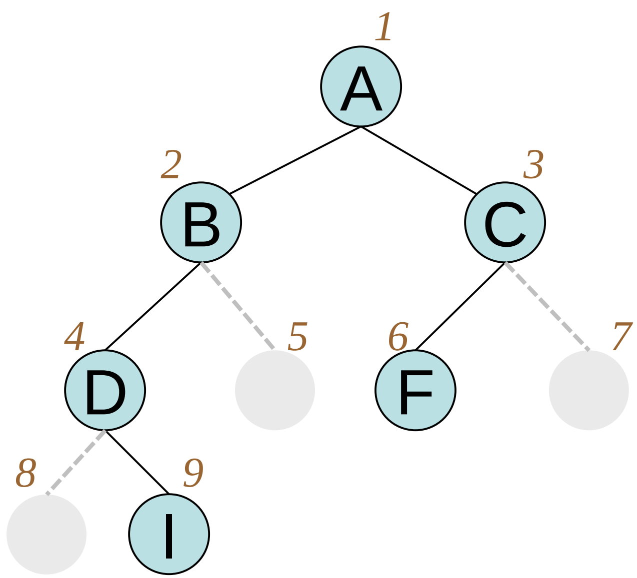
- 일반 이진트리에서의 노드 번호



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
	A	B	C	D		F			I

❑ 일반 이진트리를 배열로 표현 [2]

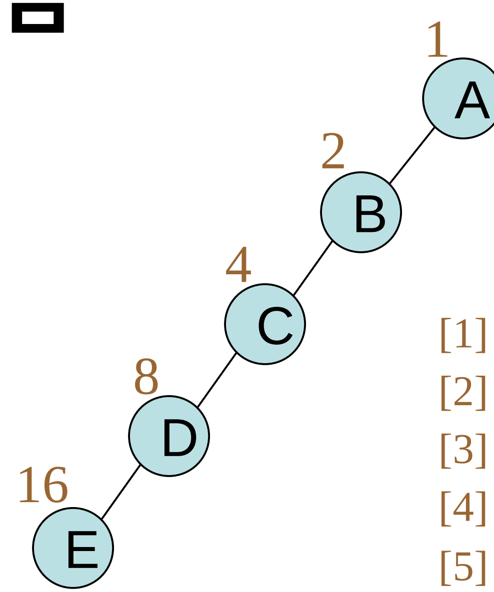
- 일반 이진트리에서의 노드 번호



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
	A	B	C	D		F			I

배열을 이용할 경우의 장단점

- 어떠한 이진트리도 배열을 이용하여 표현 가능
  - 메모리 낭비가 많을 수 있다.
- 완전 이진트리에서는:
  - 메모리 낭비 공간이 없음
- 트리에서 삽입과 삭제가 자주 발생할 경우
  - 위치를 바꾸어야 할 노드의 수가 많을 수 있다 ➡ 비효율적
  - 연결 체인을 이용한 표현을 사용하는 것이 더 좋음



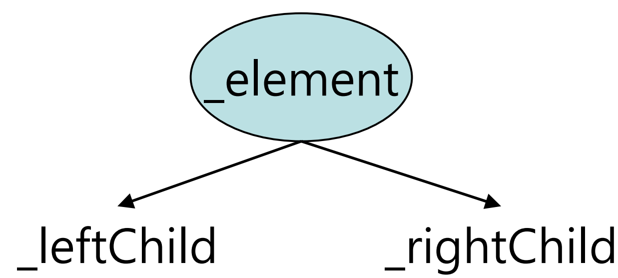
[1]	A
[2]	B
[3]	
[4]	C
[5]	
[6]	
[7]	
[8]	D
[9]	
[10]	
[11]	
[12]	
[13]	
[14]	
[15]	
[16]	E

연결 체인을 이용한 표현

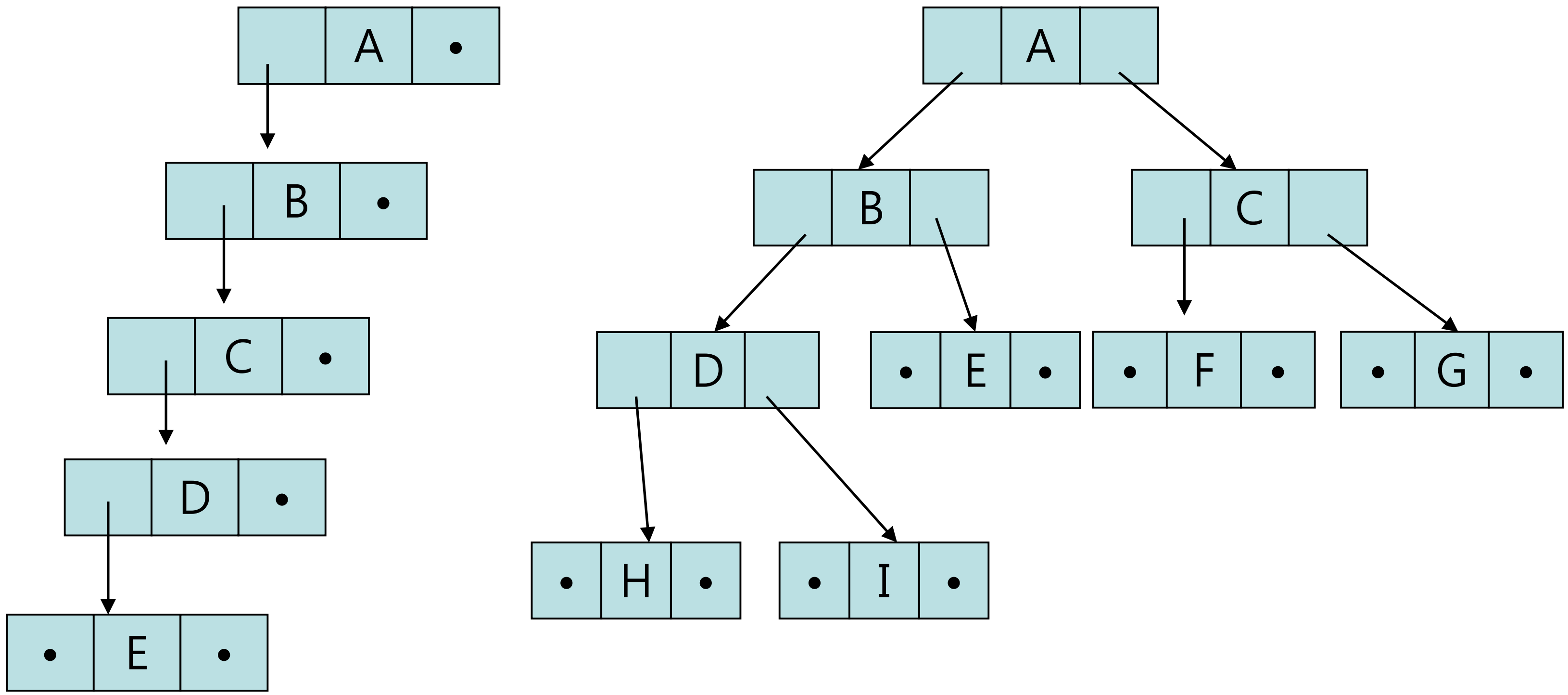
Class “BinaryNode”

연결체인을 사용한 노드의 구현 구조

```
public class BinaryNode<Element>
{
    // 비공개 멤버 변수
    private    Element                    _element   ;
    private    BinaryNode<Element>    _leftChild ;
    private    BinaryNode<Element>    _rightChild ;
```



예제



Class "BinaryNode"의 공개함수

- BinaryNode 객체 사용법
  - public BinaryNode() ;
  - public BinaryNode(Element anElement,
    BinaryNode<Element> aLeftChild,
    BinaryNode<Element> aRightChild) ;

  - public boolean hasLeftChild() ;
  - public boolean hasRightChild() ;
  - public boolean isLeaf() ;
  - public int height() ;
  - public int numberOfNodes() ;

  - public Element element() ;
  - public void setElement(Element anElement) ;
  - public BinaryNode<Element> leftChild() ;
  - public void setLeftChild(BinaryNode<Element> aLeftChild) ;
  - public BinaryNode<Element> rightChild() ;
  - public void setRightChild(BinaryNode<Element> aRightChild) ;

❑ Class "BinaryNode"의 공개함수

- BinaryNode 객체 사용법을 Java로 구체적으로 표현
  - public BinaryNode() ;
  - public BinaryNode(Element anElement,
    BinaryNode<Element> aLeftChild,
    BinaryNode<Element> aRightChild) ;

  - public boolean hasLeftChild() ;
  - public boolean hasRightChild() ;
  - public boolean isLeaf() ;
  - public int height() ;
  - public int numberOfNodes() ;

  - public Element element() ;
  - public void setElement(Element anElement) ;
  - public BinaryNode<Element> leftChild() ;
  - public void setLeftChild(BinaryNode<Element> aLeftChild) ;
  - public BinaryNode<Element> rightChild() ;
  - public void setRightChild(BinaryNode<Element> aRightChild) ;

Class “BinaryNode”의 구현: 인스턴스 변수

```
public class BinaryNode<Element>
{
    // 비공개 멤버 변수
    private    Element                     _element ;
    private    BinaryNode<Element>    _leftChild ;
    private    BinaryNode<Element>    _rightChild ;
```

Class “BinaryNode”의 구현: 생성자

```
public class BinaryNode<Element>
{
    // 비공개 멤버 변수
    ......

    // 생성자
    public BinaryNode ( )
    {
        this._element = null ;
        this._leftChild = null ;
        this._rightChild = null ;
    }

    public BinaryNode( Element                  anElement,
                       BinaryNode<Element>      aLeftChild,
                       BinaryNode<Element>      aRightChild)
    {
        this._element = anElement ;
        this._leftChild = aLeftChild ;
        this._rightChild = aRightChild ;
    }
```

BinaryNode : 상태 알아보기

```
public class BinaryNode<Element>
{
    // 비공개 멤버 변수
    ……

    // 왼쪽 자식을 가지고 있는지 확인
    public boolean  hasLeftChild()
    {
        return (this._leftChild != null) ;
    }

    // 왼쪽 자식을 가지고 있는지 확인
    public boolean  hasRightChild()
    {
        return (this._rightChild != null) ;
    }

    // Leaf인지 확인
    public boolean  isLeaf()
    {
        return (this._leftChild == null) && (this._rightChild == null) ;
    }
```

BinaryNode : 상태 알아보기

```
public class BinaryNode<Element>
{
    // 비공개 멤버 변수
    ......

    // 트리의 높이 얻기 (Recursively)
    public int      height()
    {
        int leftHeight = 0 ;
        if (this.hasLeftChild()) {
            leftHeight = this._leftChild.height() ;
        }
        int rightHeight = 0 ;
        if (this.hasRightChild) {
             rightHeight = this._rightChild.height() ;
        }
        if (leftHeight > rightHeight)
            return (leftHeight+1) ;
        else
            return (rightHeight+1) ;
    }
```

BinaryNode : 상태 알아보기

```
public class BinaryNode<Element>
{
    // 비공개 멤버 변수
    ……

    // 트리의 노드 개수 얻기
    public int      numberOfNodes()
    {
        int numberOfLeftNodes = 0 ;
        if (this.hasLeftChild()) {
            numberOfLeftNodes = this._leftChild.numberOfNodes() ;
        }
        int numberOfRightNodes = 0 ;
        if (this.hasRightChild) {
            numberOfRightNodes = this._rightChild.numberOfNodes() ;
        }
        return (1+numberOfLeftNodes+numberOfRightNodes) ;
    }
```

BinaryNode : 상태 알아보기

```
public class BinaryNode<Element>
{
    // 비공개 멤버 변수
    ……

    // 원소 얻어내기: getter for element
    public Element element()
    {
        return this._element ;
    }

    // 원소 설정하기: setter for element
    public void  setElement(Element anElement)
    {
        this._element = anElement ;
    }
```

BinaryNode : 상태 알아보기

```
public class BinaryNode<Element>
{
    // 비공개 멤버 변수
    ……

    // left의 자식 Node를 얻어냄: getter for leftChild
    public BinaryNode<Element>  leftChild()
    {
        return  this._leftChild ;
    }

    // left의 자식을 설정함: setter for leftChild
    public void setLeftChild(BinaryNode<Element> aLeftChild)
    {
        this._leftChild = aLeftChild ;
    }
```

❑ BinaryNode : 상태 알아보기

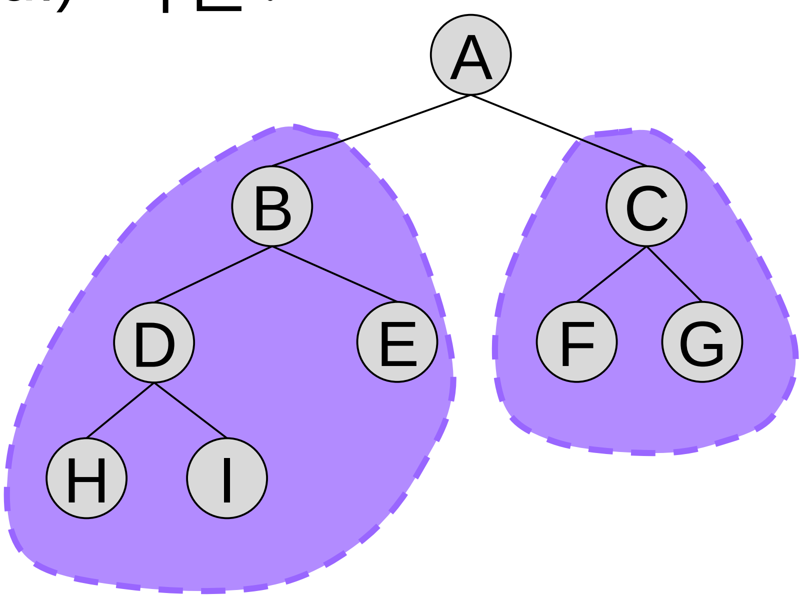
```
public class BinaryNode<Element>
{
    // 비공개 멤버 변수
    ……

    // right의 자식 Node를 얻어냄: getter for rightChild
    public BinaryNode<Element>  rightChild()
    {
        return  this._rightChild ;
    }

    // right의 자식을 설정함: setter for rightChild
    public void setRightChild(BinaryNode<Element> aRightChild)
    {
        this._rightChild = aRightChild ;
    }
```

이진 트리 탐색
(Binary Tree Traversals)

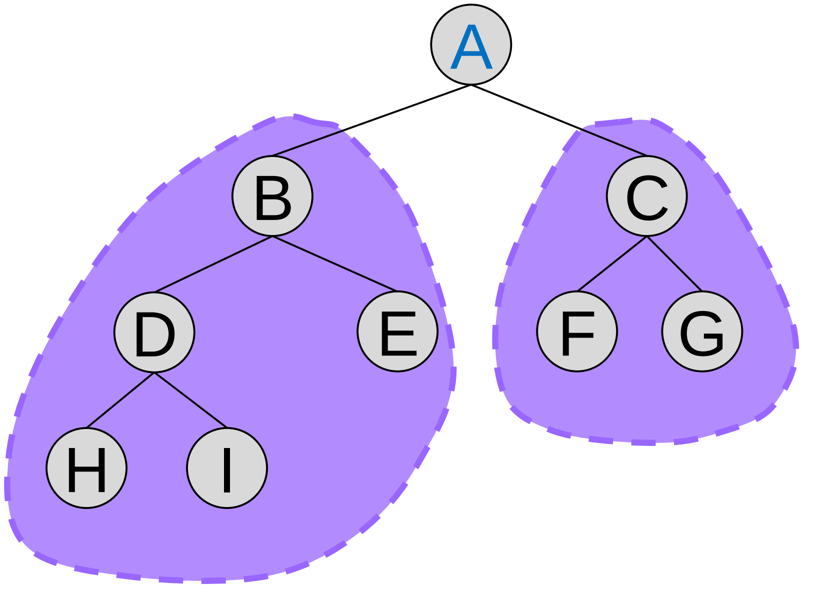
이진트리 탐색

- 체계적으로 모든 노드를 방문
  - 중위 탐색 (Inorder traversal)
  - 전위 탐색 (Preorder traversal)
  - 후위 탐색 (Postorder traversal)

- 중위탐색 (Inorder Traversal) 이란?
  - 왼쪽 부트리를 중위 탐색하여 모든 노드를 방문
  - 루트를 방문
  - 오른쪽 부트리를 중위 탐색하여 모든 노드를 방문

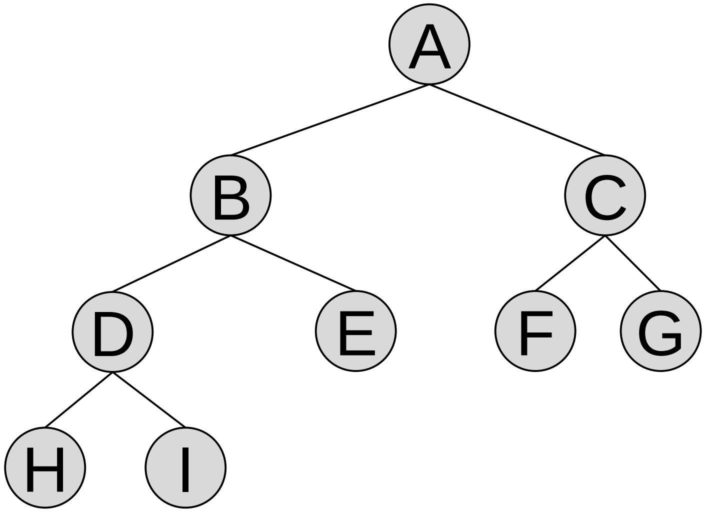


❑ 이진트리 탐색 순서에서 루트의 위치는?

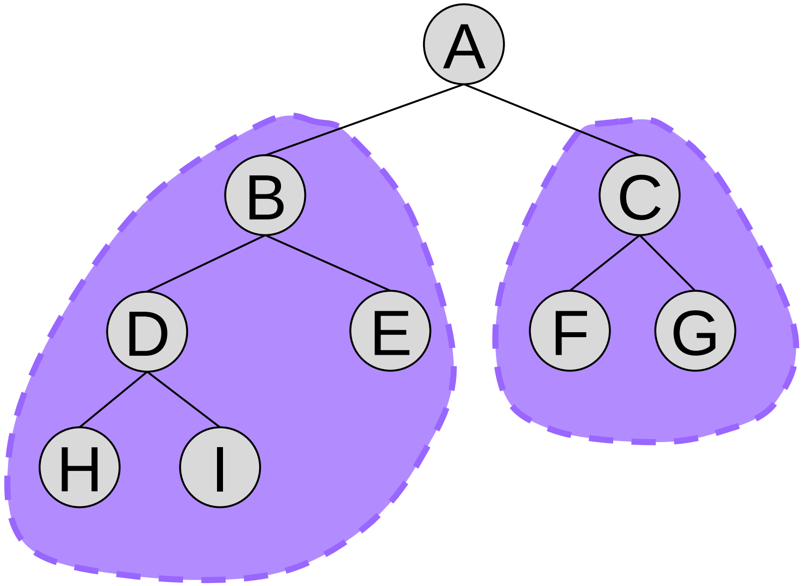


- Inorder : (((H)–D–(I))–B–(E))–A–((F)–C–(G))
  H – D – I – B – E – A – F – C – G
- Preorder : A – B – D – H – I – E – C – F – G
- Postorder : H – I – D – E – B – F – G – C – A

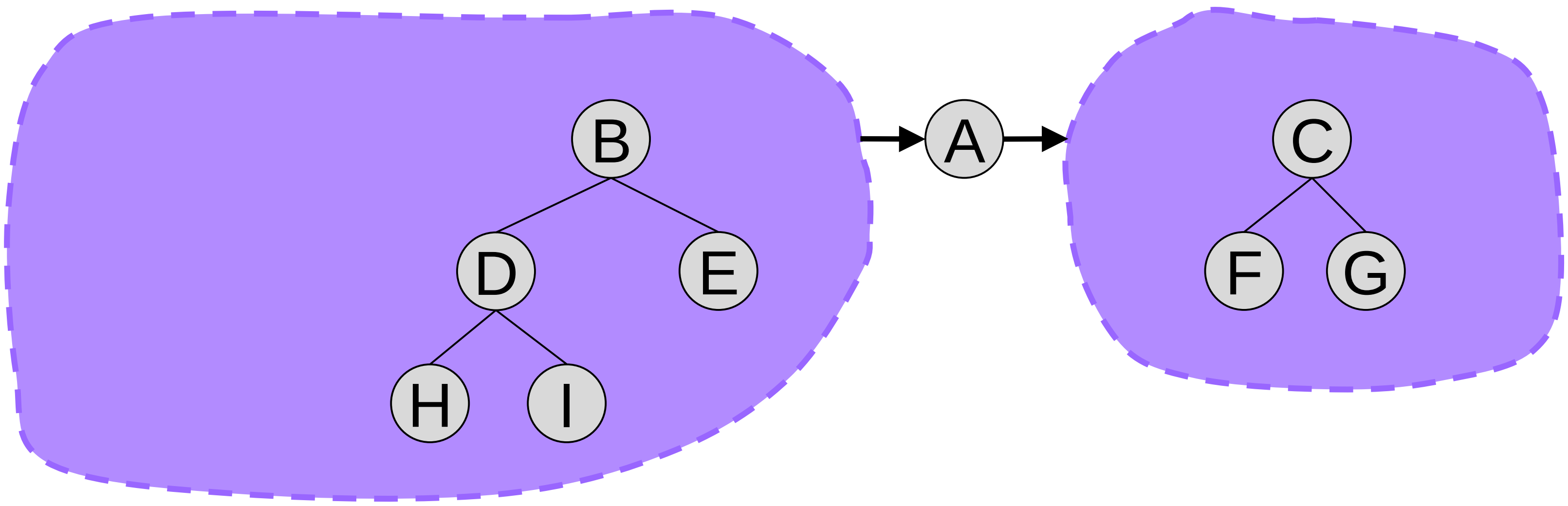
❑ 예제: 중위 탐색 (Inorder) [0]



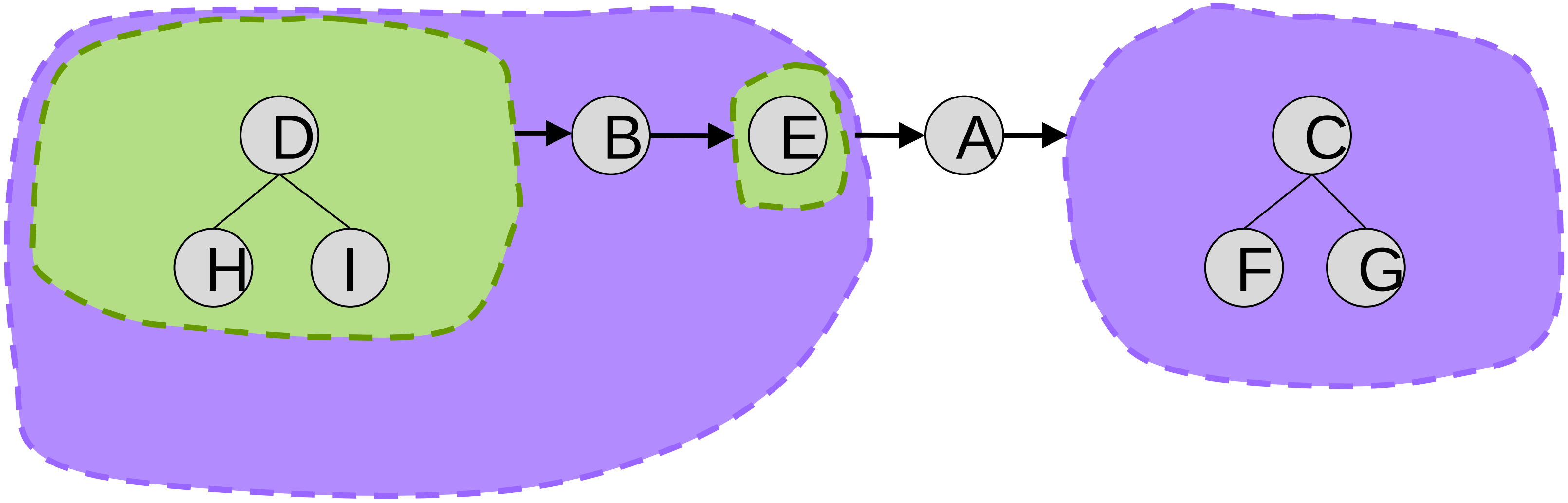
❑ 예제: 중위 탐색 (Inorder) [1]



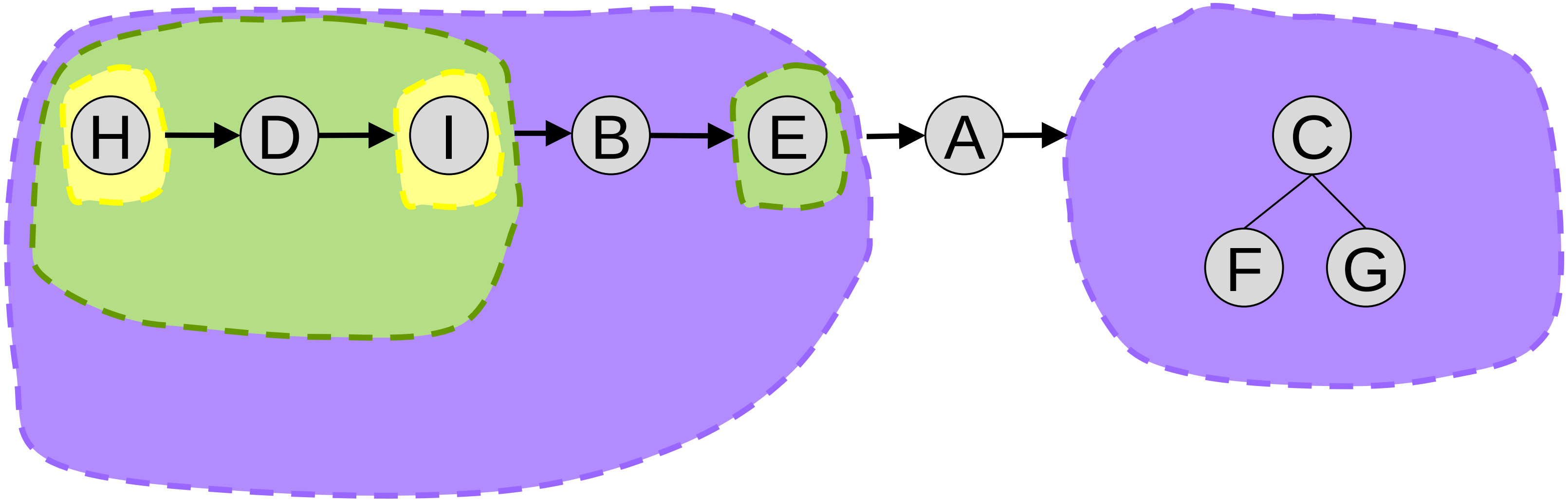
❑ 예제: 중위 탐색 (Inorder) [2]



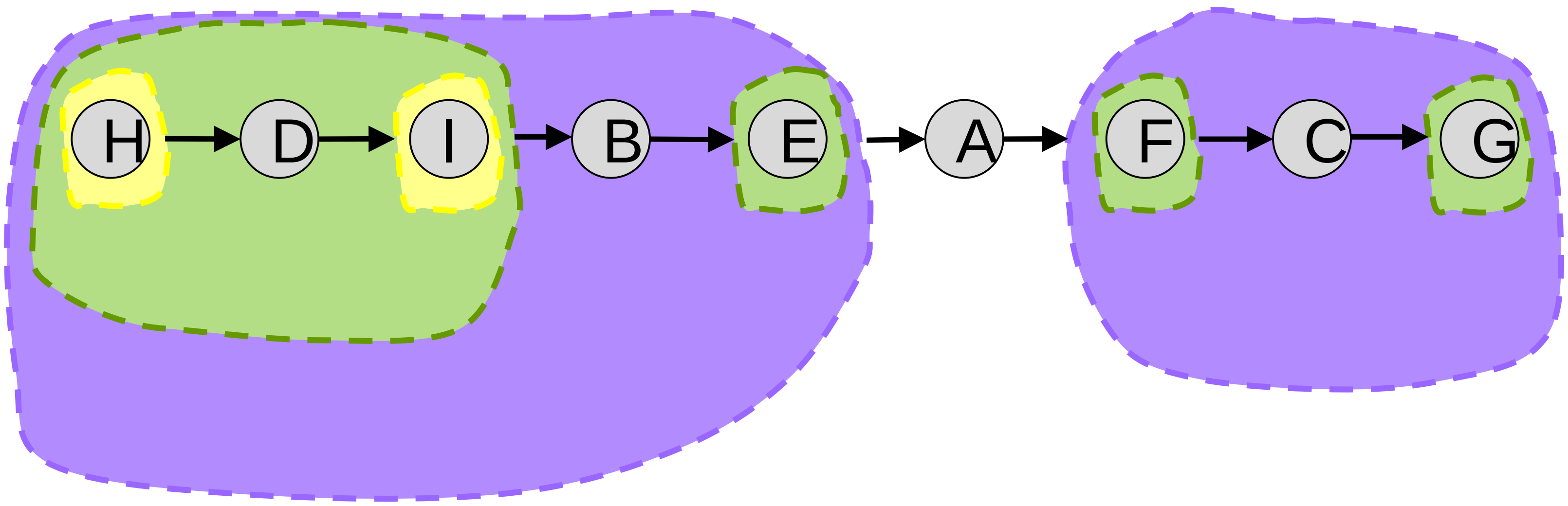
❑ 예제: 중위 탐색 (Inorder) [3]



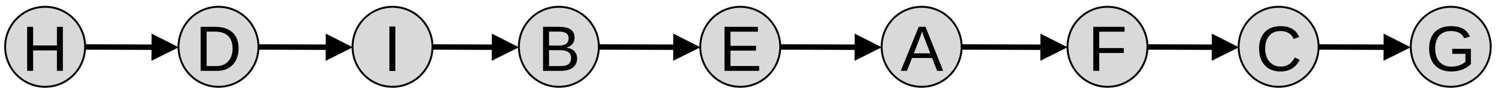
❑ 예제: 중위 탐색 (Inorder) [4]



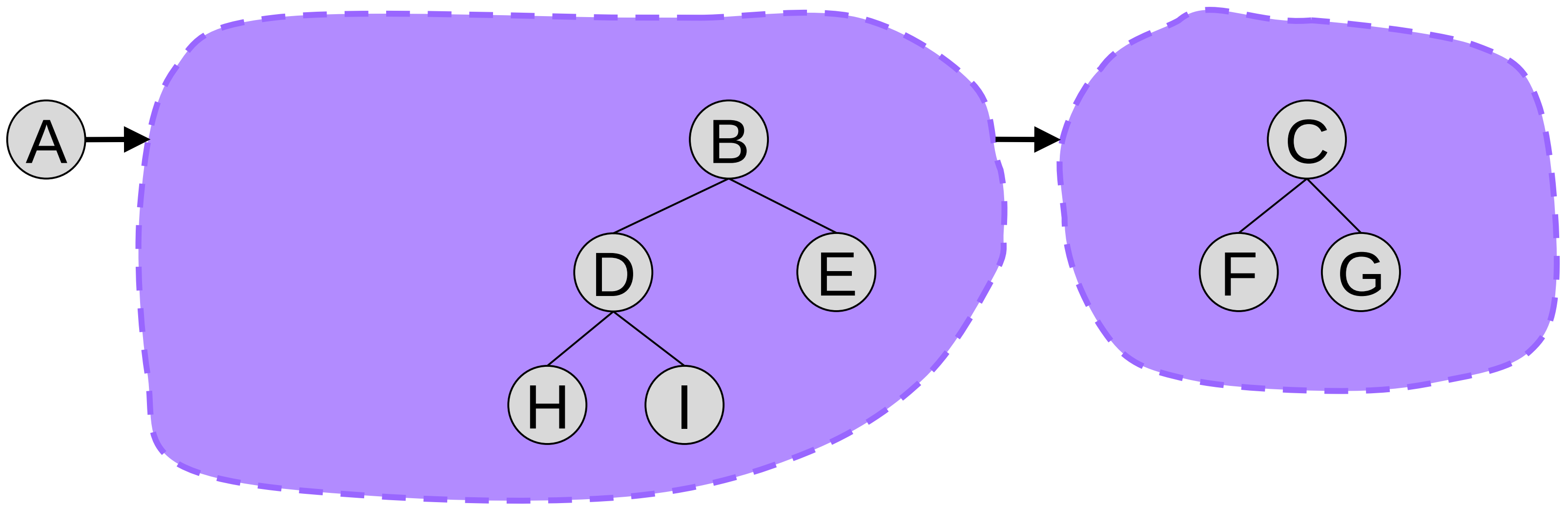
예제: 중위 탐색 (Inorder) [5]



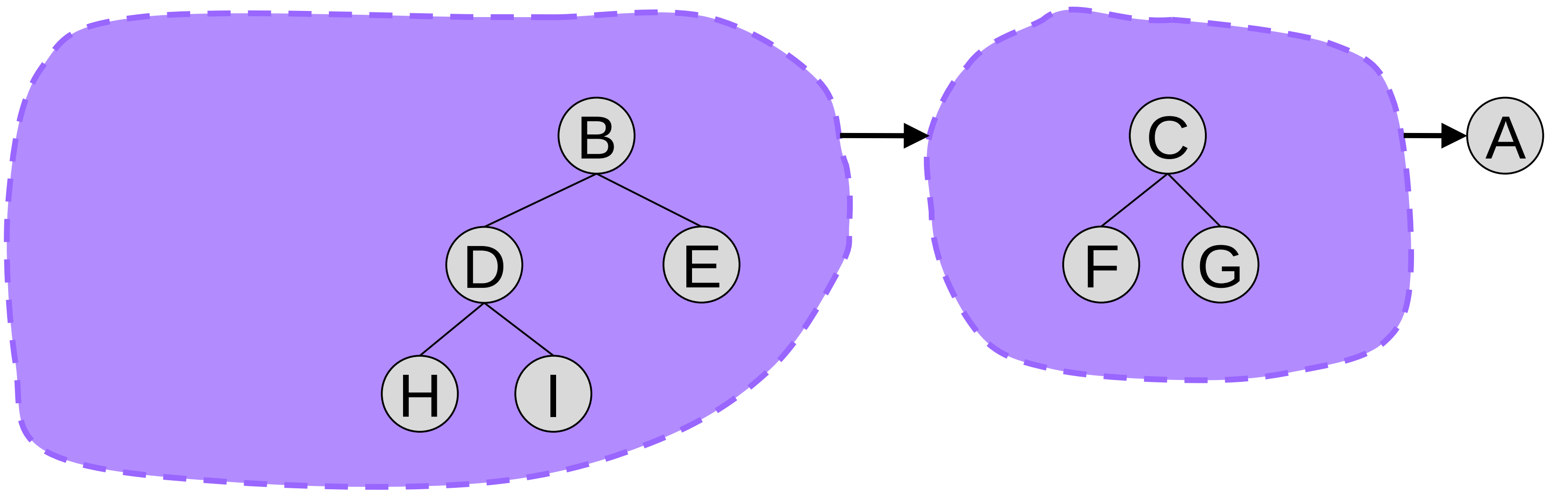
❑ 예제: 중위 탐색 (Inorder) [6]



예제: 전위 탐색 (Preorder)



❑ 예제: 후위 탐색 (Postorder)



탐색 알고리즘

중위 탐색 (Inorder traversal)

```
private  void  inOrderRecursively (BinaryNode aRoot)
{
    if ( aRoot != NULL ) {
        this.inOrderRecursively (aRoot.leftChild()) ;
        this.visit (aRoot.element()) ;
        this.inOrderRecursively (aRoot.rightChild()) ;
    }
}
```

- 탐색은 재귀적으로(recursively) 실행된다.
- 그러므로, 코드에 보이지는 않지만 스택이 사용되고 있다.

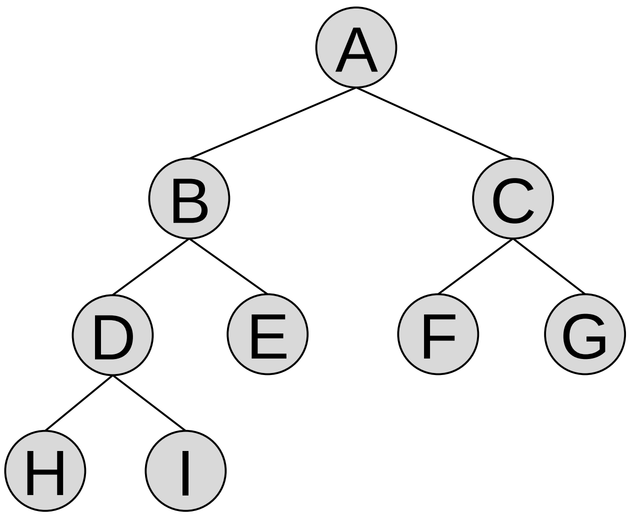
공개 함수 inOrder() 는?

```
public  void  inOrder ()
{
    this.inOrderRecursively (this._root) ;
}
```

사용자가 visit() 기능 정의하는 방법은?

❑ 레벨 순으로 탐색

```
Initially, addToQueue(root);
While (queue is not empty) {
    deleteFromQueue(X);
    visit(X);
    addToQueue all children of X;
}
```



Queue	Visit
→ A →	A
→ C → B →	B
→ E → D → C →	C
→ G → F → E → D →	D
→ I → H → G → F → E →	E
→ I → H → G → F →	F
→ I → H → G →	G
→ I → H →	H
→ I →	I
→ →	(End of Traversal)

멤버 함수 levelOrder ()

```
public void  levelOrder()
{
    Queue<BinaryNode>  nodeQueue= new Queue() ;
    BinaryNode  currentNode ;
    nodeQueue.add(this._root) ;
    while (! nodeQueue.isEmpty()) {
        currentNode = nodeQueue.remove() ;
        this.visit(currentNode.element());
        if ( currentNode.hasLeftChild() )
            nodeQueue.add(currentNode.leftChild()) ;
        if ( currentNode.hasRightChild() )
            nodeQueue.add(currentNode.rightChild()) ;
    }
}
```

Class “BinaryTree”

Class “BinaryTree”의 공개함수

BinaryTree 객체 사용법

```
// 공개함수
public          BinaryTree() ;
public          BinaryTree (    Element                   aRootElement,
                                BinaryTree<Element> aLeftTree,
                                BinaryTree<Element> aRightTree) ;

public boolean     isEmpty() ;
public int         height() ;
public int         size() ;

public Element     rootElement() ;
public BinaryTree<Element>  leftSubtree() ;
public BinaryTree<Element>  rightSubtree() ;
public void      setTree (    Element   aRootElement,
                              BinaryTree<Element>  aLeftTree,
                              BinaryTree<Element>  aRightTree) ;

public void        inOrder() ;
public void        preOrder() ;
public void        postOrder() ;
public void        levelOrder() ;
```

Class “Binary Tree”: 인스턴스 변수

```
public class BinaryTree<T>
{
    // 비공개 멤버 (인스턴스) 변수
    private   BinaryNode<T>   _root ;
```

Class “BinaryTree”의 구현: 생성자

```
public class BinaryTree<Element>
{
    // 생성자
    public  BinaryTree ( )
    {
        _root = null ;
    }

    public  BinaryTree ( boolean                shared ;
                         Element                aRootElement,
                         BinaryTree<Element>    aLeftTree,
                         BinaryTree<Element>    aRightTree)
    {
        if (shared) {
            this.setTreeByShare (aRootElement, aLeftTree, aRightTree) ;
        }
        else {
            this.setTreeByCopy (aRootElement, aLeftTree, aRightTree) ;
        }
    }
```

BinaryTree : 상태 알아보기

```
public class BinaryTree<Element>
{
    // 비공개 멤버 변수
    ......

    // 트리가 비어있는지 확인
    public boolean   isEmpty()
    {
        return (this._root == null) ;
    }

    // 트리의 높이를 돌려준다
    public int     height()
    {
        if ( this._root == null) {
            return  0 ;
        }
        else {
            return ( this._root.height() ) ;
        }
    }

    // 트리의 노드 개수를 돌려준다
    public int  numberOfNodes()
    {
        if ( this._root == null) {
            return  0 ;
        }
        else {
            return ( this._root.numberOfNodes) ) ;
        }
    }
```

BinaryTree: setTreeByShare()

```
public class BinaryTree<Element>
{
    ......

    public void    setTreeByShare(  Element                  aRootElement,
                                    BinaryTree<Element>      aLeftSubtree,
                                    BinaryTree<Element>      aRightSubtree)
    {
        this._root = new BinaryNode<Element>
            (aRootElement, aLeftSubtree._root, aRightSubtree._root) ;

    }
```

BinaryTree: setTreeByCopy()

```
public class BinaryTree<Element>
{
    ......

    public void    setTreeByCopy ( Element                    aRootElement,
                                   BinaryTree<Element>    aLeftSubtree,
                                   BinaryTree<Element>    aRightSubtree )
    {
        BinaryNode<Element> rootOfCopiedLeftSubtree =
            copyBinaryTreeNodes(aLeftSubtree.root()) ;
        BinaryNode<Element> rootOfCopiedRightSubtree =
            copyBinaryTreeNodes(aRightSubtree.root()) ;
        this._root = new BinaryNode<Element>
           (aRootElement, rootOfCopiedLeftSubtree, rootOfCopiedRightSubtree) ;

    }
```

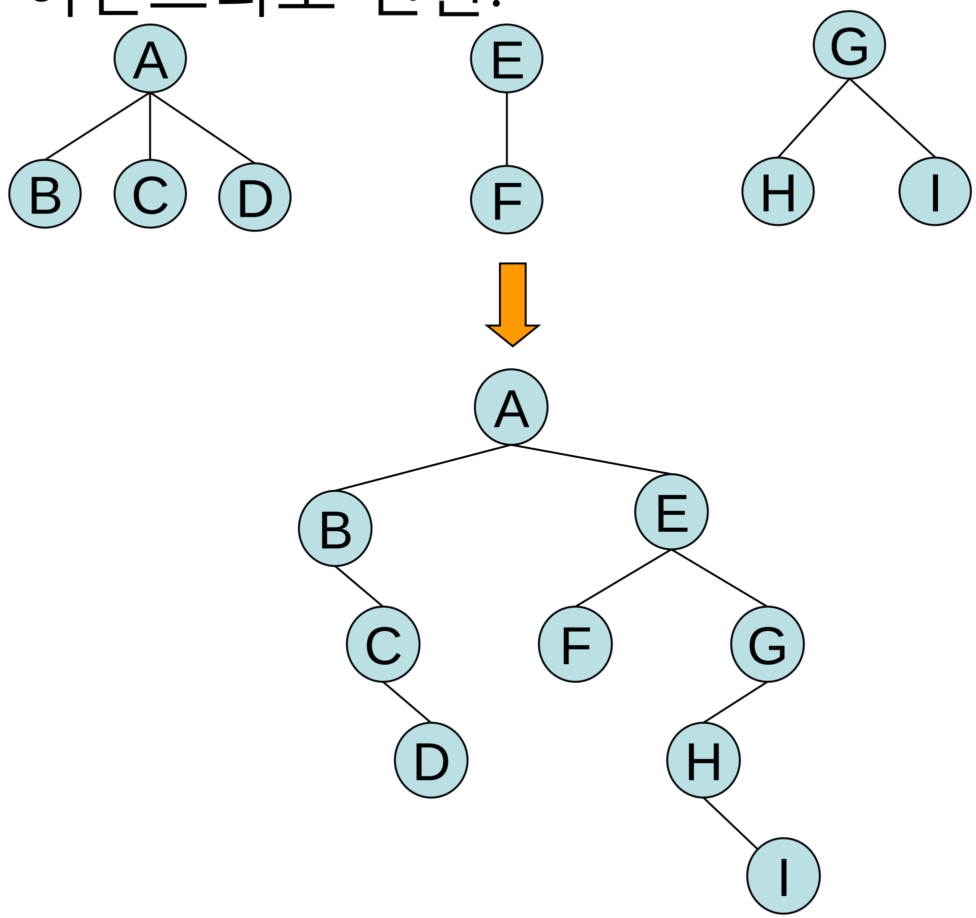
BinaryTree : 트리의 복사

```
public class BinaryTree<Element>
{
    // 비공개 멤버 변수
    ......
    private BinaryNode<Element>
                copyBinaryTreeNodes (BinaryNode<Element> aRoot)
    {
        if (aRoot == null) {
            return null ;
        }
        else {
            BinaryNode<Element>
                copiedLeftChild = copyBinaryTreeNodes(aRoot.leftChild()) ;
            BinaryNode<Element>
                copiedRightChild = copyBinaryTreeNodes(aRoot.rightChild()) ;
            BinaryNode<Element> copiedRoot =
                new BinaryNode<Element>
                    (aRoot.element().copy(), copiedLeftChild, copiedRightChild) ;
            return copiedRoot ;
        }
    }
```

Traversals for Forests

숲 (Forests)

- 숲 (forest): 0 개 이상의 서로 원소가 겹치지 않는 트리들의 집합.
- 숲을 이진트리로 변환:



숲의 전위 탐색

- F = { T1, T2, …, Tn }
- Tree Preorder (F)

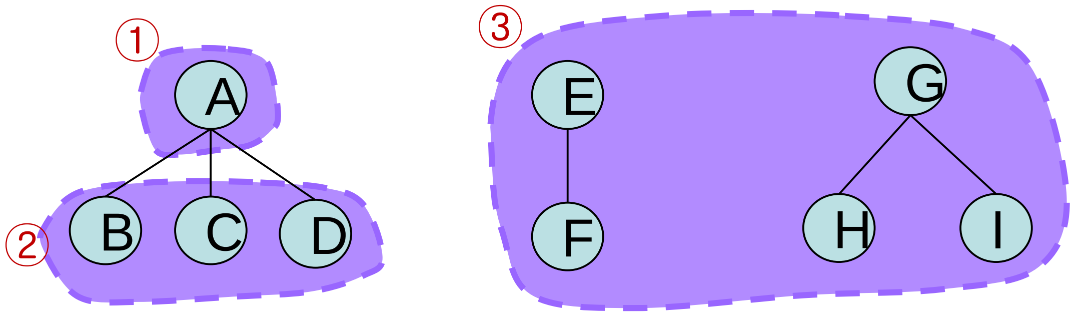
  If (F 가 공집합이 아니라면) {

  T1의 루트를 방문 ;

  T1의 의 부트리들을 Tree Preorder 로 탐색 ;

  F의 나머지 트리들(T2 , … , Tn) 을 Tree Preorder 로 탐색 ;

  }



- 예:
  - 숲: A – B – C – D – E – F – G – H – I
  - 이진 트리: A – B – C – D – E – F – G – H – I

숲의 중위 탐색

- F = { T1, T2, …, Tn }
- Tree Inorder (F)

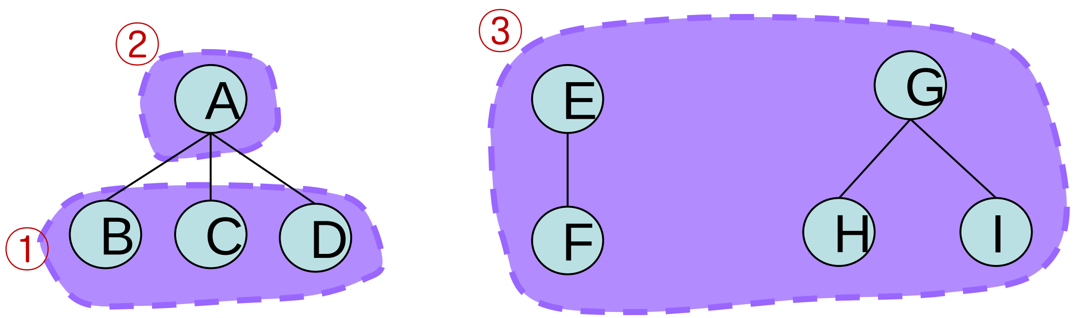
  If (F 가 공집합이 아니라면) {

  T1의 의 부트리들을 Tree Inorder 로 탐색 ;

  T1의 루트를 방문 ;

  F의 나머지 트리들(T2 , … , Tn) 을 Tree Inorder 로 탐색 ;

  }

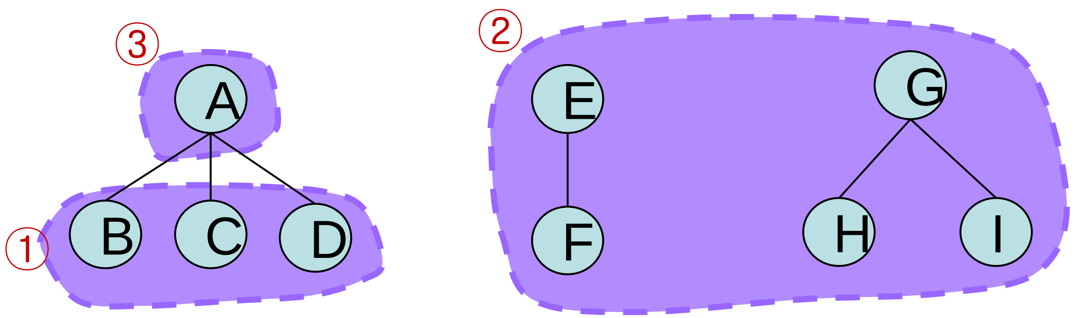


- 예:
  - Forest : B – C – D – A – F – E – H – I – G
  - Binary Tree : B – C – D – A – F – E – H – I – G

Postorder Traverse of Forests

- F = { T1, T2, ..., Tn }
- Postorder (F)

If (F 가 공집합이 아니라면) {
    T1의 의 부트리들을 Tree Postorder 로 탐색 ;
    F의 나머지 트리들(T2 , ... , Tn) 을 Tree Postorder 로 탐색 ;
    T1의 루트를 방문 ;
}



- 예:
  - Forest : D – C – B – F – I – H – G – E – A
  - Binary Tree : D – C – B – F – I – H – G – E – A

"Tree" [끝]