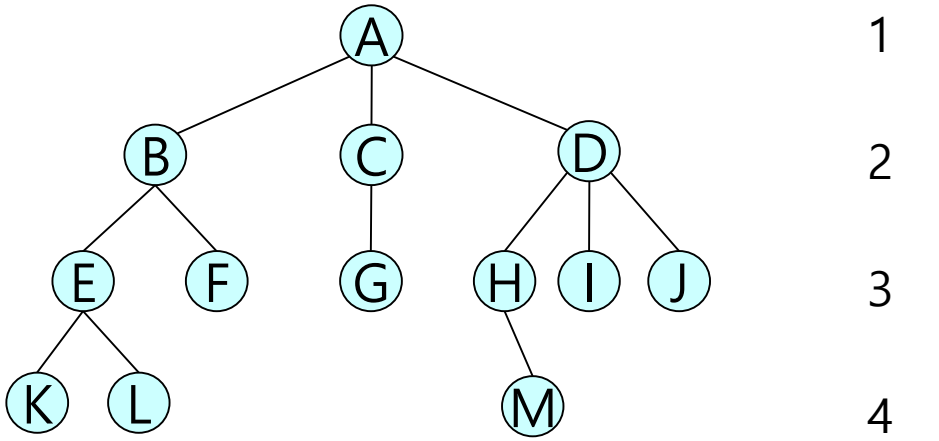
자료구조: 2022년 1학기 [강의]

트리 (Tree)

강지훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터융합학부

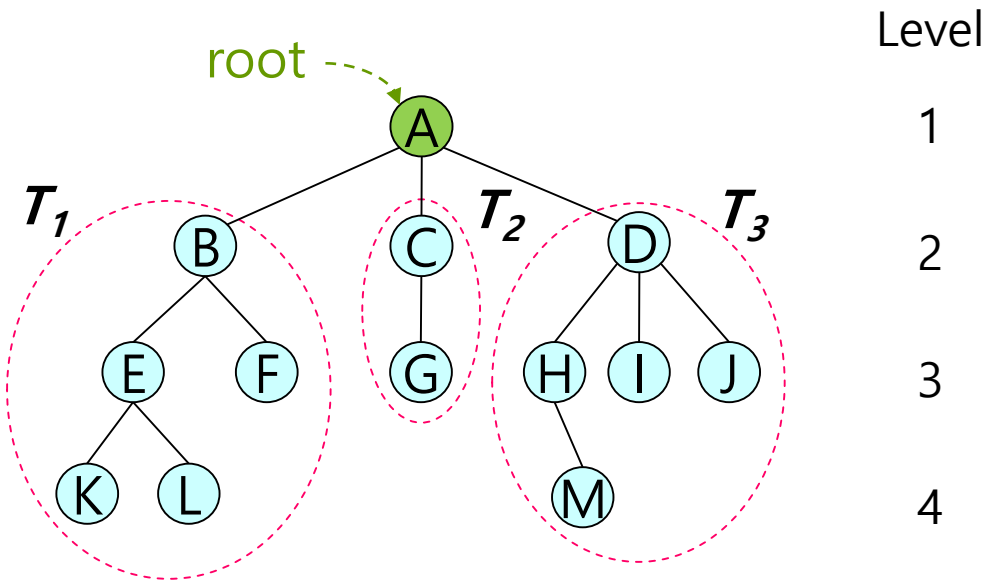
트리 (Tree)

트리



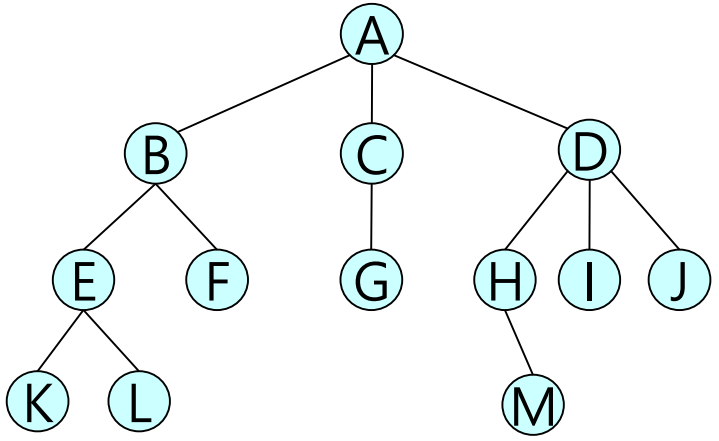
트리 (Tree)

- 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 ($n \geq 0$) 의 노드가 서로 중복되지 않는 집합 (disjoint sets) $T_1, \ldots, T_n$ 으로 나누어지며, 이 각각의 노드 집합은 트리 이어야 한다. $T_1, \ldots, T_n$ 이들을 루트의 부트리(subtrees) 라 한다.
- 재귀적으로 정의: 부트리는 다시 트리로 정의되고 있다.



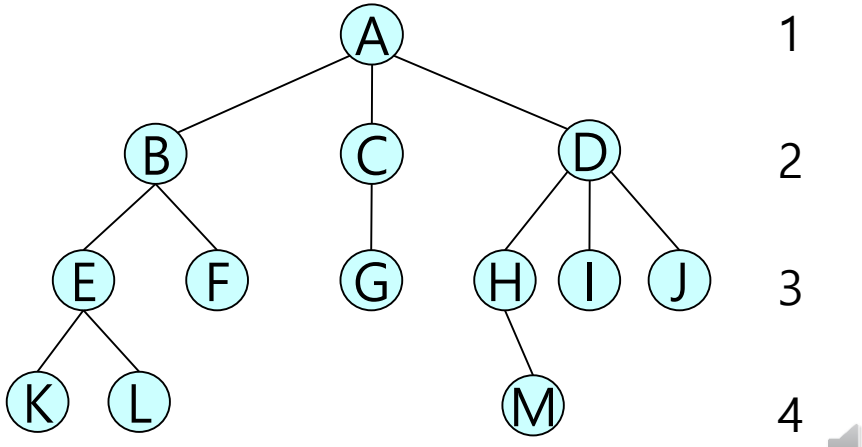
용어 [1]

- 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 0 인 노드
  - Example: K, L, F, G, M, I, J
  - non-leaf (non-terminal): leaf 가 아닌 노드
- 노드 X 의 자식 (child of a node X): X 의 서브트리의 루트
- 부모 (parent): 자식의 역관계
  - 예: B 는 E 와 F 의 부모.



용어 [2]

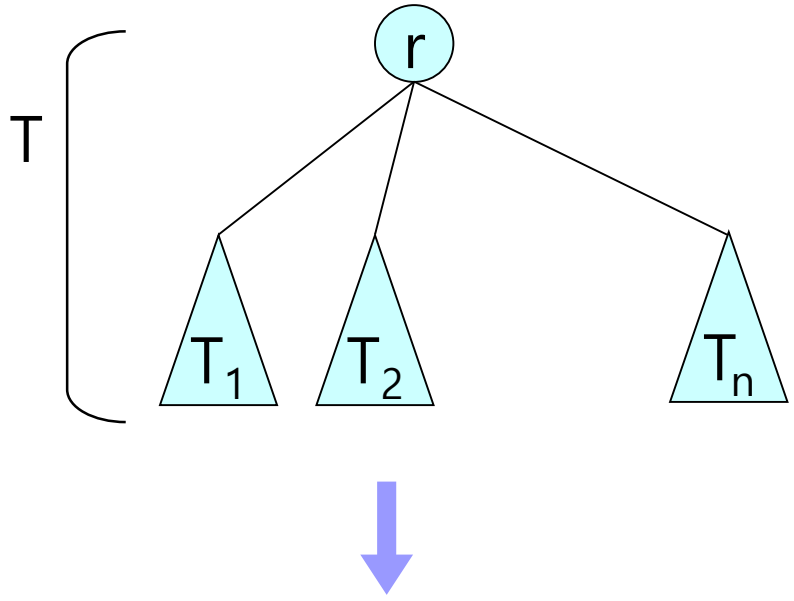
- 형제 (sibling): 부모가 동일한 노드
  - Example: H, I, J are siblings
- 노드 X 의 조상 (ancestor): 노드 X 에서부터 루트까지의 경로를 따라 존재하는 노드
  - 예: M 의 조상은 A, D, H 이다.
  - 자손 (후손, descendent): (예) 노드 B 의 자손은 E, F, K, M 이다.
- 레벨 (level):
  - 루트의 레벨은 1 이다.
  - 루트의 자식의 레벨은 2 이다.
  - 노드 X 의 레벨이 k 이면, X 의 자식의 레벨은 (k+1) 이다.
- 높이 (height) (또는 깊이 (depth)): 트리의 최대 레벨
  - 예:
    - M의 레벨 = 4 : (트리에서 최대값)
    - 트리의 높이 = 4



트리의 표현

리스트를 이용한 표현

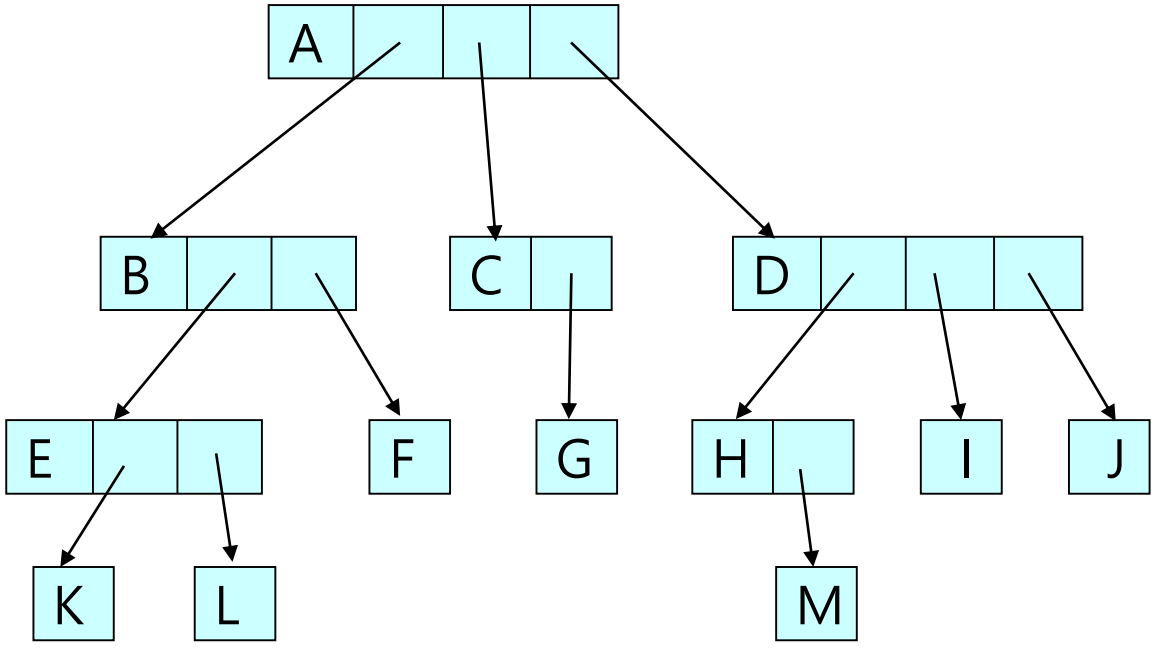
- 부트리(subtree) 들을 리스트로 표현



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

리스트를 이용한 표현

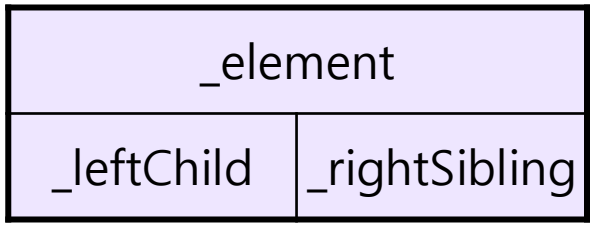
- 예: 가변길이 노드 (variable length nodes)를 사용하여 리스트를 표현
  - (A (B (E (K, L), F), C (G), D (H (M), I, J)))

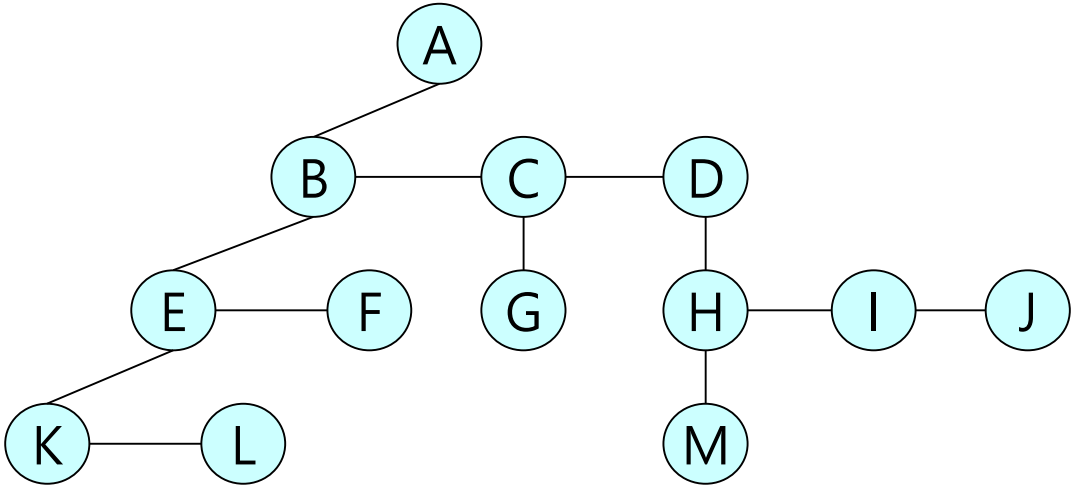


왼쪽 자식-오른쪽 형제 (Left Child - Right Sibling) 표현

- 고정 길이 (fixed length) 노드 사용
- 각 노드는 3 개의 속성:

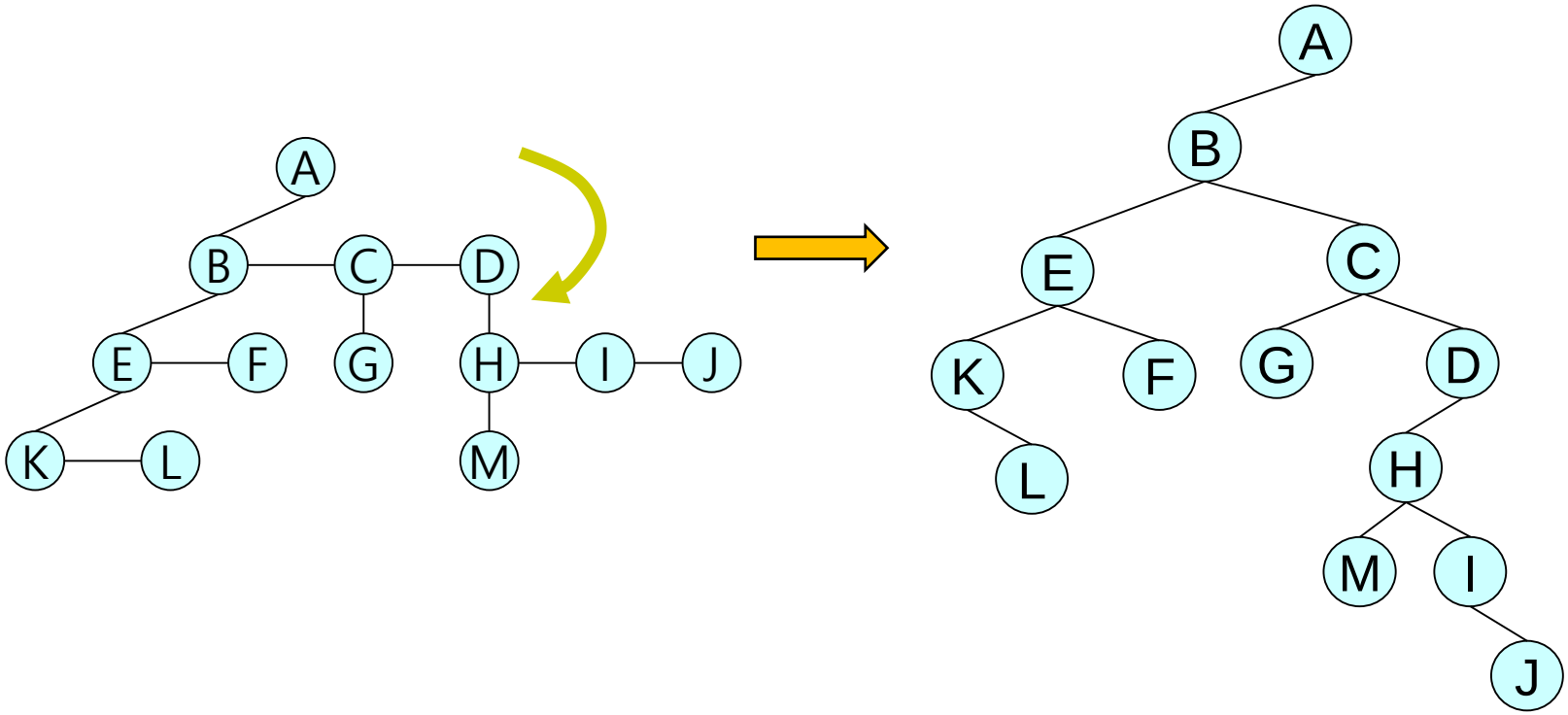
```
public class  TreeNode<T> {
    private T                 _element ;
    private TreeNode<T>       _leftChild ;
    private TreeNode<T>       _rightSibling ;
    ......
}
```





이진 트리로 표현

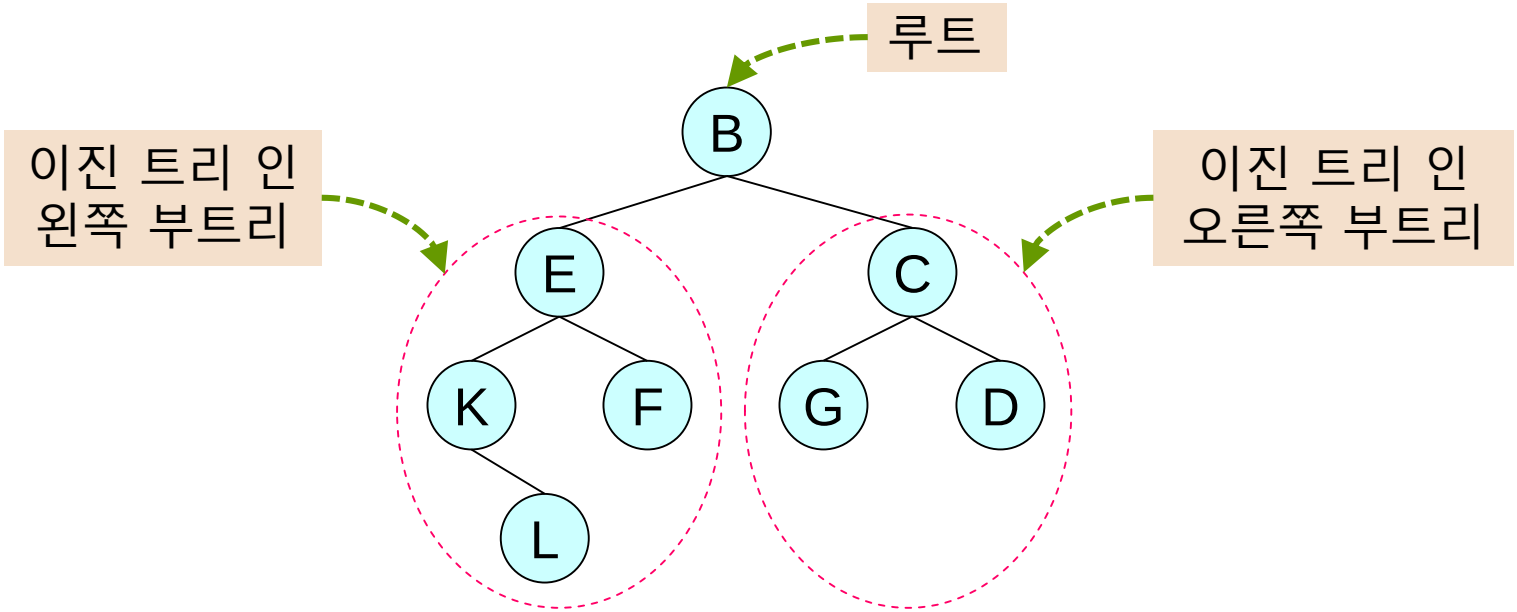
- Left child - right sibling 트리를 시계 방향으로 45 도 회전시킨다.
- 결국 이진트리 (binary tree) 가 된다.



이진 트리 (Binary Tree)

이진트리 (Binary Trees)

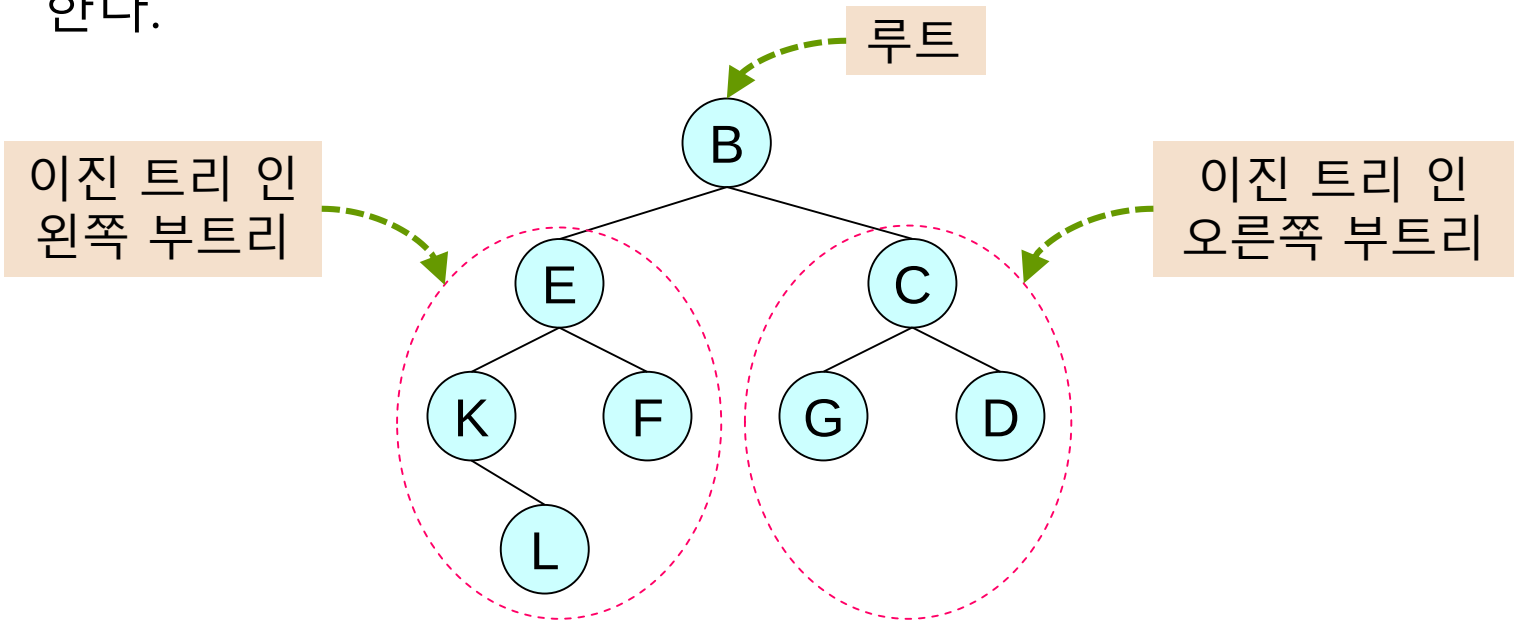
- 이진트리 (binary tree) 는 노드의 유한집합으로 다음의 조건을 만족해야 한다:
  1) 비어 있거나, 또는
  2) 루트(root) 와 두개의 서로 겹치지 않는 이진 트리로 구성되며, 각각 왼쪽부트리 left subtree), 오른쪽부트리 (right subtree) 라 한다.



- 이진 트리에서도 트리의 용어를 그대로 사용

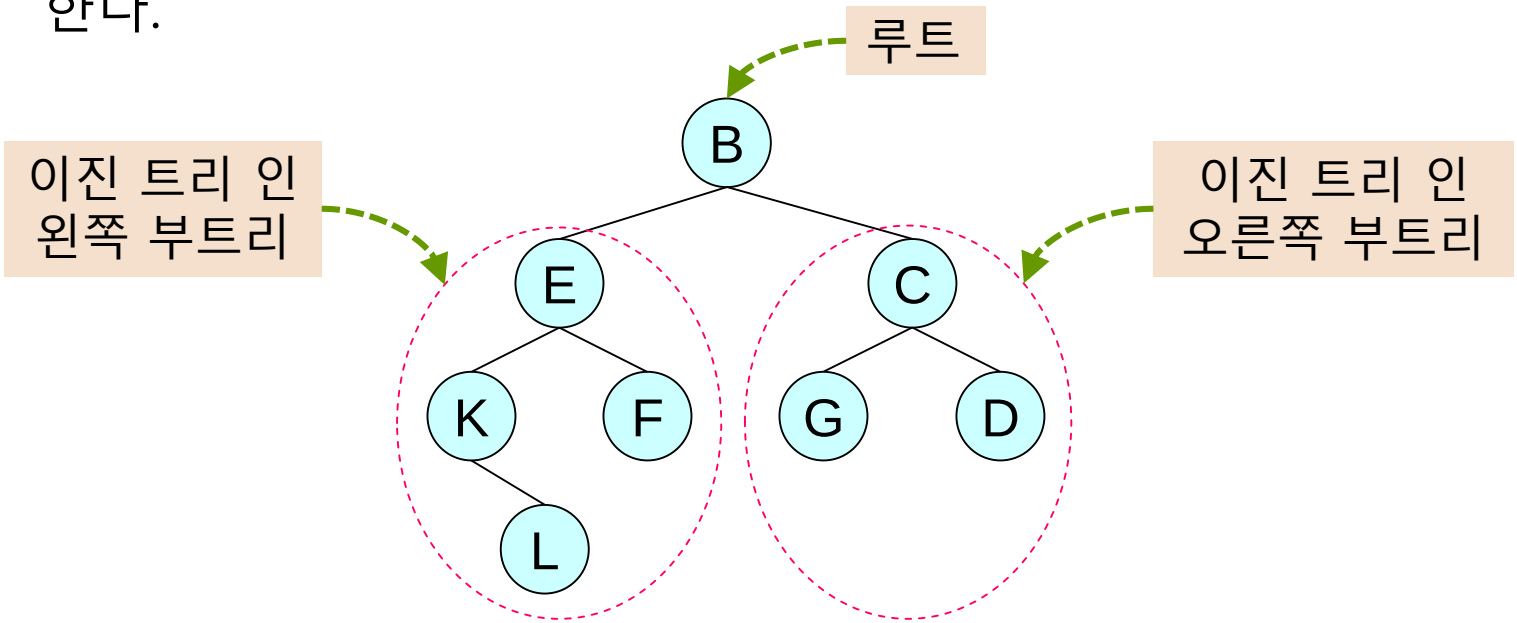
이진트리 (Binary Trees)

- 이진트리 (binary tree) 는 노드의 유한집합으로 다음의 조건을 만족해야 한다:
  1) 비어 있거나, 또는
  2) 루트(root) 와 두 개의 서로 겹치지 않는 이진 트리로 구성되며, 각각 왼쪽 부트리 left subtree), 오른쪽 부트리 (right subtree) 라 한다.



이진트리 (Binary Tree)

- 이진트리 (binary tree) 는 노드의 유한집합으로 다음의 조건을 만족해야 한다:
  1) 비어 있거나, 또는
  2) 루트(root) 와 두 개의 서로 겹치지 않는 이진 트리로 구성되며, 각각 왼쪽 부트리 left subtree), 오른쪽 부트리 (right subtree) 라 한다.

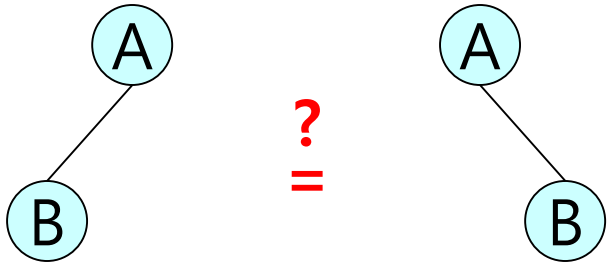


- 이진 트리에서도 트리의 용어를 그대로 사용

트리와 이진트리의 차이점?

서브트리의 순서

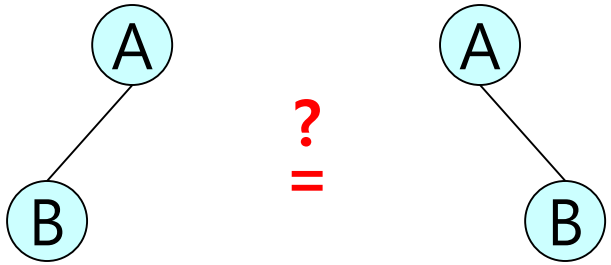
- 이진 트리에는 서브트리의 위치가 있다.
  그러나 트리의 서브트리에는 순서도 위치도 없다.



- 이들이 이진트리 라면, 이 둘은 다르다.
- 이들이 트리 라면, 이 둘은 같다.
  - 응용에 따라 트리에서도 필요하다면, 서브트리에 순서를 부여할 수 있다.

트리와 이진트리의 차이점?

- 서브트리의 순서
  - 이진 트리에는 서브트리의 위치가 있다.
    그러나 트리의 서브트리에는 순서도 위치도 없다.

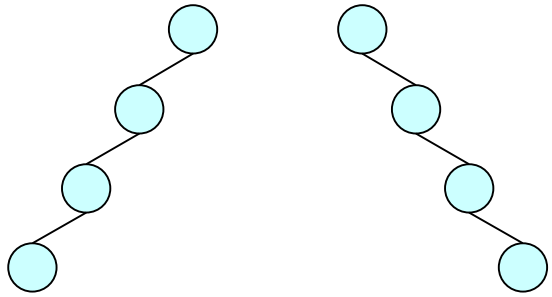


  - 이들이 이진트리 라면, 이 둘은 다르다.
  - 이들이 트리 라면, 이 둘은 같다.
    - 응용에 따라 트리에서도 필요하다면, 서브트리에 순서를 부여할 수 있다.

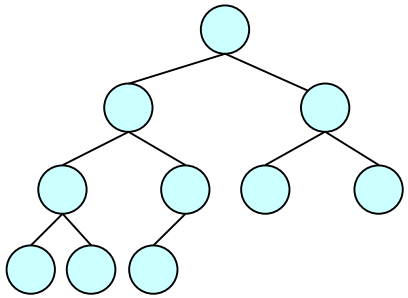
- 노드의 최소 개수 (이론적 관점)
  - 이진트리에는 노드가 하나도 없을 수도 있다: Empty Binary Tree
  - 트리에는 적어도 하나의 노드가 존재한다.
    - Empty tree -> Forest (0 개 이상의 트리의 집합)

특수한 이진 트리

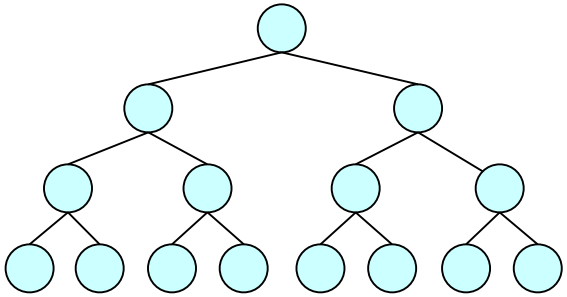
- 치우친 이진트리
  (Skewed binary tree
  / Degenerate binary tree)



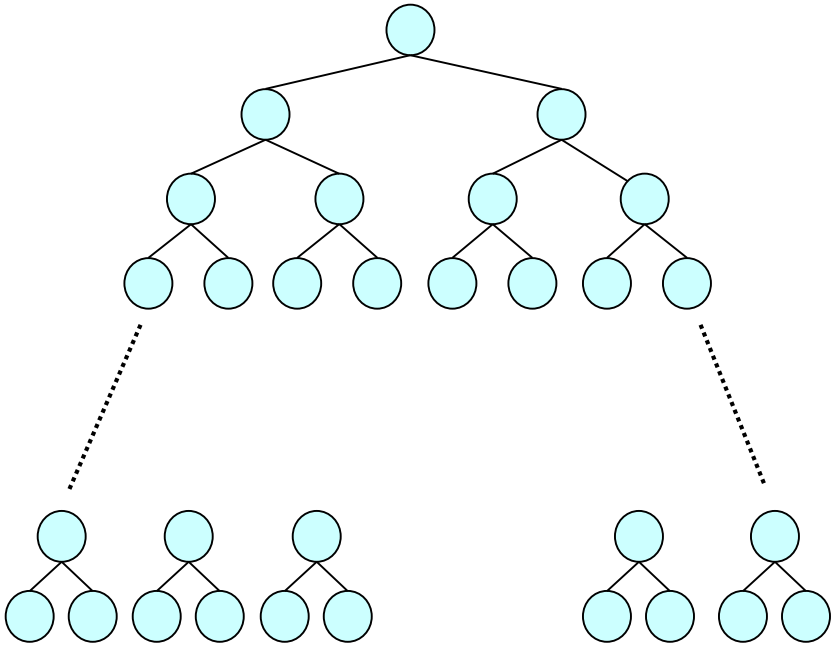
- 완전이진트리
  (Complete binary tree)



- 꽉찬 이진트리
  (Full binary tree)



레벨과 노드 수와의 관계

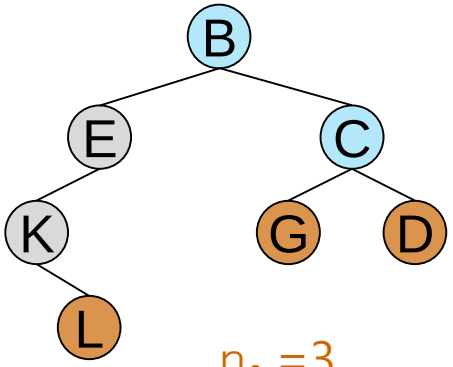


Level i	2^{i-1}	$2^i - 1$
1	2^0	$2^1 - 1$
2	2^1	$2^2 - 1$
3	2^2	$2^3 - 1$
4	2^3	$2^4 - 1$
$k-1$	2^{k-2}	$2^{k-1} - 1$
k	2^{k-1}	$2^k - 1$

- 레벨 i 에 있을 수 있는 노드의 최대 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 2 인 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, $n_0 = n_2 + 1$.



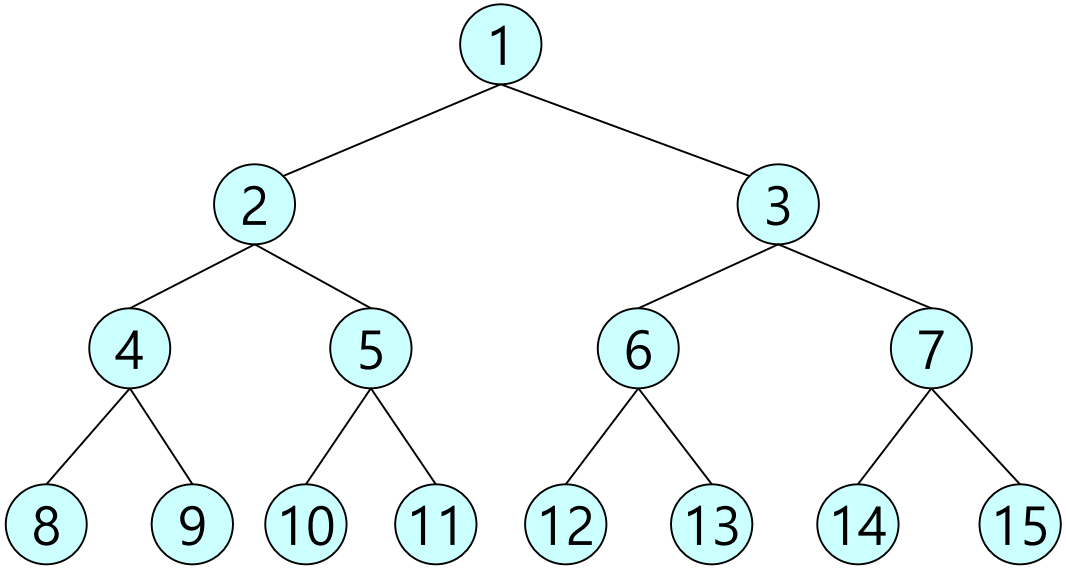
n_0 =3
n_1=2
n_2=2

(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
$n = B + 1$.
$B = 1 \cdot n_1 + 2 \cdot n_2$.
So, $n = 1 \cdot n_1 + 2 \cdot n_2 + 1$ .....[2]
Since [1]=[2], $n = n_0 + n_1 + n_2 = 1 \cdot n_1 + 2 \cdot n_2 + 1$.
Therefore, $n_0 = n_2 + 1$. ■

꽉찬 이진트리 (Full Binary Tree)

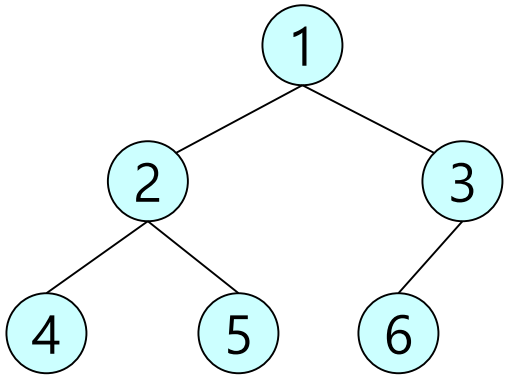
- A full binary tree of Depth k :
  A binary tree of depth k having 2^k -1 nodes, ($k \geq 1$)
  - Example: Full binary tree of depth 4.

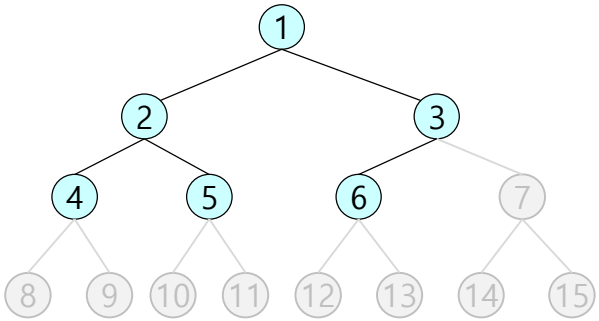


완전 이진트리 (Complete Binary Tree)

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

- 예: 노드가 6 개인 완전 이진트리





- 꽉찬 이진트리는 완전 이진트리이다.

이진트리의 표현

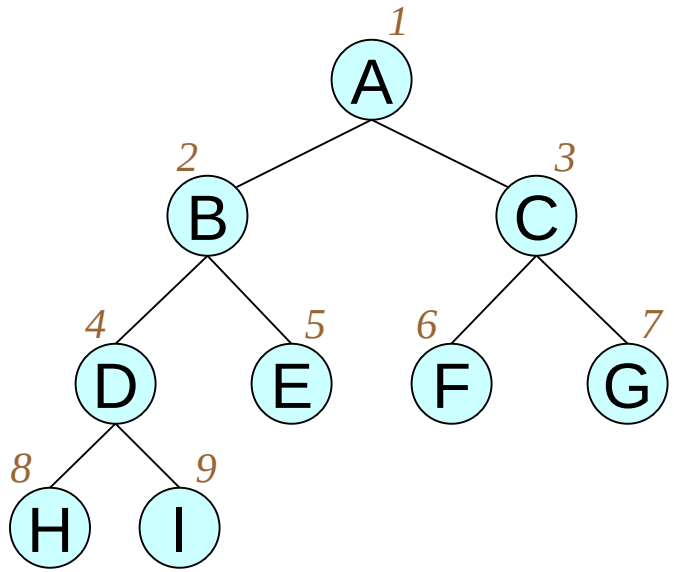
배열을 이용

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



parent(6) → $\lfloor 6/2 \rfloor = 3$
parent(9) → $\lfloor 9/2 \rfloor = 4$
parent(1) → ***none***

lchild(2) → $2 \cdot 2 = 4$
lchild(3) → $2 \cdot 3 = 6$
lchild(7) → ***none***

rchild(2) → $2 \cdot 2 + 1 = 5$
rchild(3) → $2 \cdot 3 + 1 = 7$
rchild(7) → ***none***

노드 번호와 배열 인덱스의 관계

- 노드의 개수가 n 인 완전 이진트리를 배열을 이용하여 표현하면 다음의 관계가 성립한다:

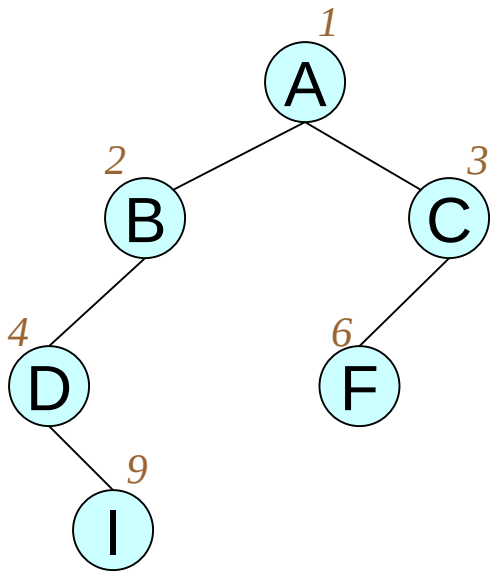
  배열 인덱스가 i $(1 \le i \le n)$ 인 노드 x 에 대해,

  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. $2 \cdot i \le n$ 이라면, x 의 왼쪽 자식은 $2 \cdot i$ 에 존재
     $2 \cdot i > n$ 이라면, x 는 왼쪽 자식이 없음
  3. $2 \cdot i + 1 \le n$ 이라면, x 의 오른쪽 자식은 $2 \cdot i + 1$ 에 존재
     $2 \cdot i + 1 > n$ 이라면, x 는 오른쪽 자식이 없음

- 노드의 개수가 n 인 완전 이진트리의 깊이: $\lfloor \log_2 n \rfloor + 1$
  - $O(\log_2 n)$

일반 이진트리를 배열로 표현 [1]

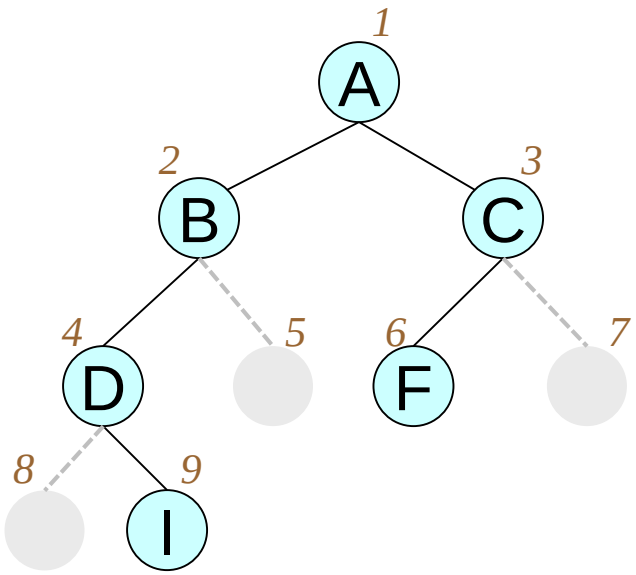
- 일반 이진트리에서의 노드 번호



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

일반 이진트리를 배열로 표현 [2]

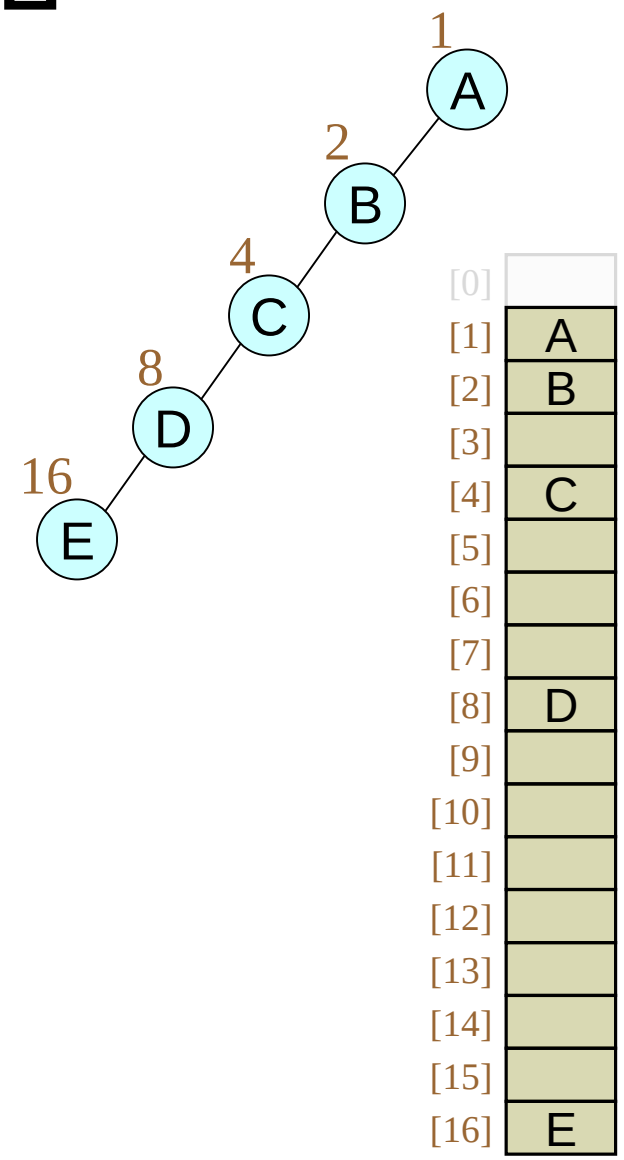
- 일반 이진트리에서의 노드 번호



[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



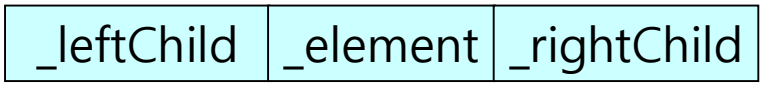
연결 체인을 이용한 표현

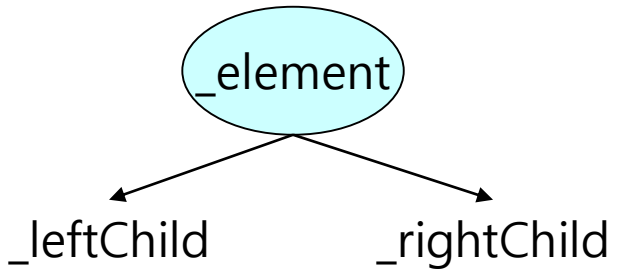
Class “BinaryNode”
Class “BinaryTree”

Class “BinaryNode<T>”

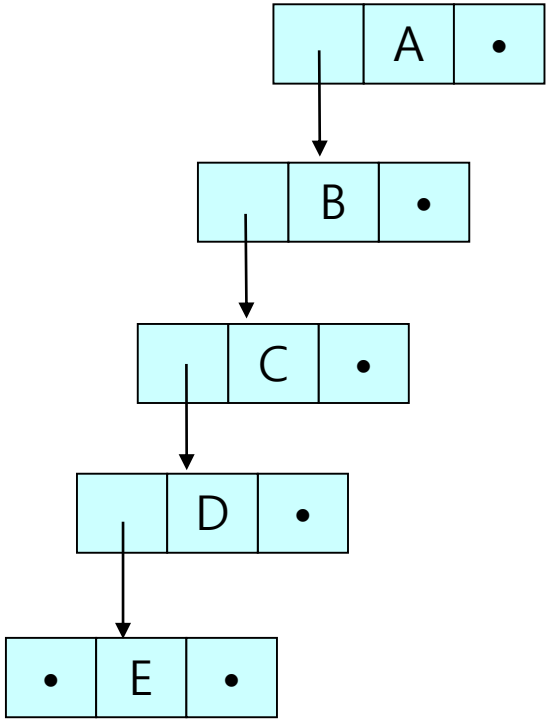
연결체인을 사용한 노드의 구현 구조

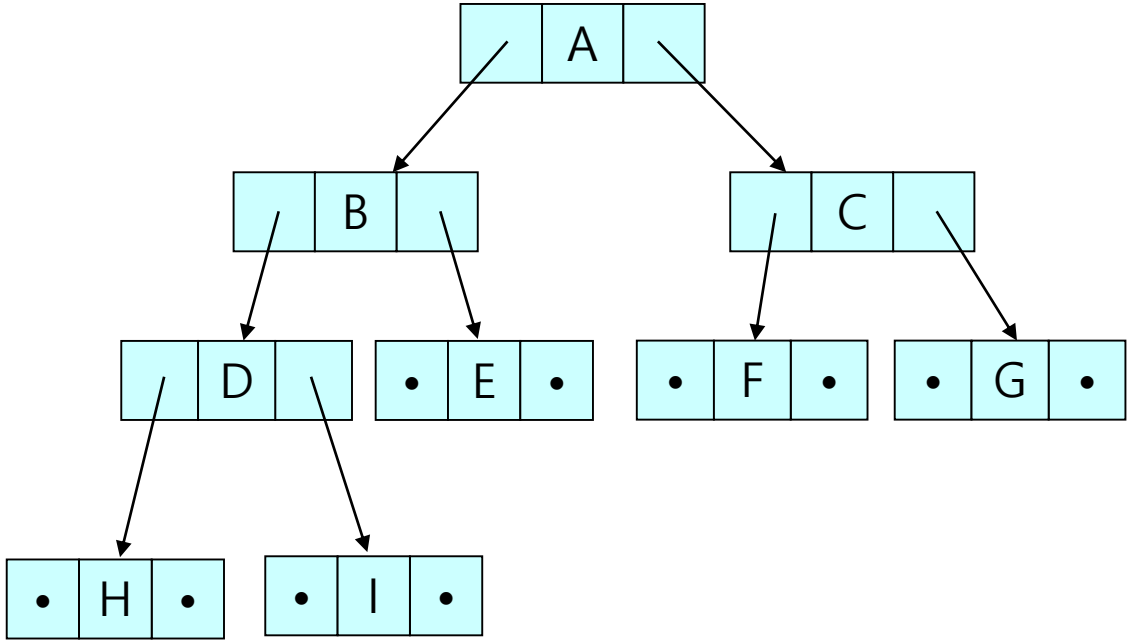
```
public class  BinaryNode<T>
{
    // 비공개 멤버 변수
    private   T                  _element   ;
    private   BinaryNode<T>      _leftChild ;
    private   BinaryNode<T>      _rightChild ;
```





예제





Class "BinaryNode<T>" 의 공개함수 [1]

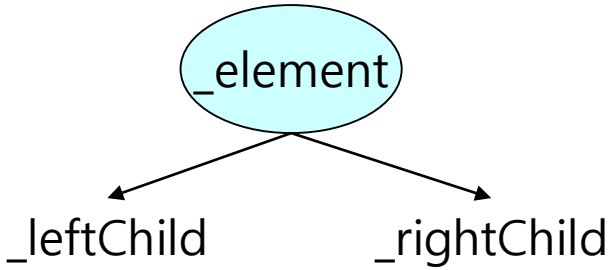
- BinaryNode 객체 사용법
  - public BinaryNode () ;
  - public BinaryNode (T givenElement,
    BinaryNode<T> givenLeftChild,
    BinaryNode<T> givenRightChild) ;
  - public boolean hasLeftChild() ;
  - public boolean hasRightChild() ;
  - public boolean isLeaf() ;
  - public T element() ;
  - public void setElement (T newElement) ;
  - public BinaryNode<T> leftChild() ;
  - public void setLeftChild (BinaryNode<T> newLeftChild) ;
  - public BinaryNode<T> rightChild() ;
  - public void setRightChild (BinaryNode<T> newRightChild) ;

Class "BinaryNode<T>" 의 공개함수 [2]

- BinaryNode 객체 사용법을 Java로 구체적으로 표현
  - public BinaryNode () ;
  - public BinaryNode (T anElement,
    BinaryNode<T> aLeftChild,
    BinaryNode<T> aRightChild) ;
  - public boolean hasLeftChild() ;
  - public boolean hasRightChild() ;
  - public boolean isLeaf() ;
  - public T element() ;
  - public void setElement (T newElement) ;
  - public BinaryNode<T> leftChild() ;
  - public void setLeftChild (BinaryNode<T> newLeftChild) ;
  - public BinaryNode<T> rightChild() ;
  - public void setRightChild (BinaryNode<T> newRightChild) ;

BinaryNode<T> : Getter/Setter 의 구현

```
public class  BinaryNode<T>
{
    // 비공개 인스턴스 변수
    private T                   _element ;
    private BinaryNode<T>       _leftChild ;
    private BinaryNode<T>       _rightChild ;

    // Getter/Setter
    public T  element() {
        return  this._element ;
    }
    public void  setElement (T newElement) {
        this._element = newElement ;
    }
    public BinaryNode<T>  leftChild() {
        return  this._leftChild ;
    }
    public void  setLeftChild (BinaryNode<T> newLeftChild) {
        this._leftChild = newLeftChild ;
    }
    public BinaryNode<T>  rightChild() {
        return  this._rightChild ;
    }
    public void  setRightChild (BinaryNode<T> newRightChild) {
        this._rightChild = newRightChild ;
    }
```



BinaryNode<T>: 생성자

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

    // 생성자
    public  BinaryNode ( )
    {
        this.setElement (null) ;
        this.setLeftChild (null) ;
        this.setRightChild (null) ;
    }

    public BinaryNode ( T                  givenElement,
                        BinaryNode<T>  givenLeftChild,
                        BinaryNode<T>  givenRightChild)
    {
        this.setElement (givenElement) ;
        this.setLeftChild (givenLeftChild) ;
        this.setRightChild (givenRightChild) ;
    }
```

BinaryNode<T> : 상태 알아보기

```
public class  BinaryNode<T>
{
    // 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) ;
    }

    // Leaf 인지 확인
    public boolean  isLeaf()
    {
        return  (this.leftChild() == null) && (this.rightChild() == null) ;
    }
```

Class “BinaryTree<T>”

BinaryTree<T>: 공개함수

- BinaryTree 객체 사용법

```
// 생성자
public      BinaryTree () ;
public      BinaryTree ( boolean          shared,
                         T                givenRootElement,
                         BinaryTree<T>    givenLeftTree,
                         BinaryTree<T>    givenRightTree ) ;

// 상태 알아보기
public boolean      isEmpty() ;
public boolean      isFull() ;
public int          size() ;
public int          height() ;

// 내용 알아보기
public T            rootElement() ;

// 탐색 (나중에 다루기로 함)
public void         inOrder() ;
public void         preOrder() ;
public void         postOrder() ;
public void         levelOrder() ;

// 내용 바꾸기: 삽입, 삭제 (나중에 다루기로 함)
```

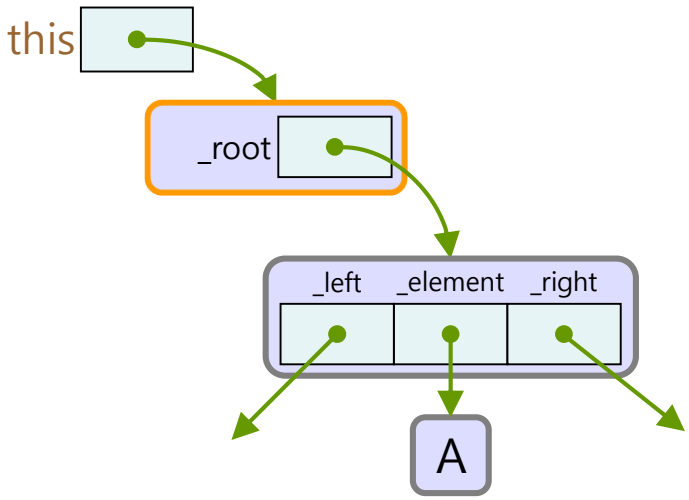
Binary Node를 이용하여 구현하는 Class “BinaryTree<T>”

BinaryTree<T>: 초기 형태

```
public class  BinaryTree<T>
{
        public  BinaryTree ( )
        {
                // 수정해야 함
        }
        public  BinaryTree ( boolean          shared,
                             T                givenRootElement,
                             BinaryTree<T>    givenLeftTree,
                             BinaryTree<T>    givenRightTree )
        {
                // 수정해야 함
        }

        public boolean  isEmpty () {
                return  true ; // 수정해야 함
        }
        public boolean  isFull () {
                return  true ; // 수정해야 함
        }
        public int    size () {
                return  0 ; // 수정해야 함
        }
        public int  height () {
                return  0 ; // 수정해야 함
        }

        public T  rootElement() {
                return  null ;
        }

        public void  inOrder() {
                return ; // 수정해야 함
        }
        public void  preOrder() {
                return ; // 수정해야 함
        }
        public void  postOrder() {
                return ; // 수정해야 함
        }
        public void  levelOrder() {
                return ; // 수정해야 함
        }

} // End of Class "BinaryTree<T>"
```

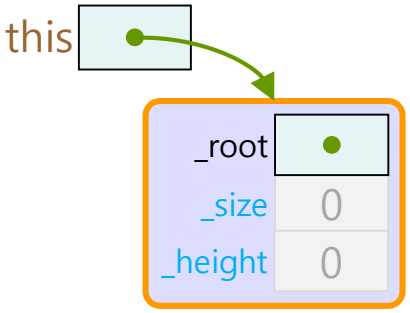
BinaryTree<T>: 인스턴스 변수

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



BinaryTree<T>: 더 필요한 인스턴스 변수는?

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

    // private int     _size ;
    // private int     _height ;
```



BinaryTree<T>: 인스턴스 변수, Getter/Setter

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


    // Getter/Setter
    private BinaryNode<T>  root() {
        return  this._root ;
    }
    private void  setRoot (BinaryNode<T> newRoot) {
        this._root = newRoot ;
    }
```

BinaryTree<T>: 생성자 [1]

```
public class  BinaryTree<T>
{
    // 생성자: empty tree 를 생성
    public  BinaryTree ( )
    {
        // 수정해야 함
    }

    // 생성자: root 원소가 주어져서, 생성되는 트리가 empty 가 아닌 경우
    public  BinaryTree ( boolean            shared ;
                         T                  givenRootElement,
                         BinaryTree<T>      givenLeftTree,
                         BinaryTree<T>      givenRightTree)
    {
        // 수정해야 함




    }
```

BinaryTree<T>: 생성자 [2]

```
public class  BinaryTree<T>
{
    // 생성자: empty tree 를 생성
    public  BinaryTree ( )
    {
        this.setRoot (null) ;
    }

    // 생성자: root 원소가 주어져서, 생성되는 트리가 empty 가 아닌 경우
    public  BinaryTree ( boolean            shared ;
                         T                  givenRootElement,
                         BinaryTree<T>      givenLeftTree,
                         BinaryTree<T>      givenRightTree)
    {
        // 수정해야 함




    }
```

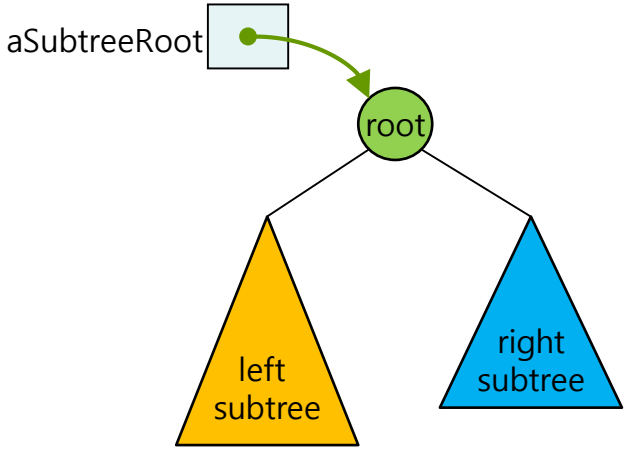
BinaryTree<T>: 생성자 [3]

```
public class  BinaryTree<T>
{
    // 생성자: empty tree 를 생성
    public  BinaryTree ( )
    {
        this.setRoot (null) ;
    }

    // 생성자: root 원소가 주어져서, 생성되는 트리가 empty 가 아닌 경우
    public  BinaryTree ( boolean          shared ;
                         T                givenRootElement,
                         BinaryTree<T>    givenLeftTree,
                         BinaryTree<T>    givenRightTree)
    {
        if ( shared ) {
            this.setTreeByShare (givenRootElement, givenLeftTree, givenRightTree) ;
        }
        else {
            this.setTreeByCopy (givenRootElement, givenLeftTree, givenRightTree) ;
        }
    }
```

BinaryTree<T>: isEmpty(), isFull()

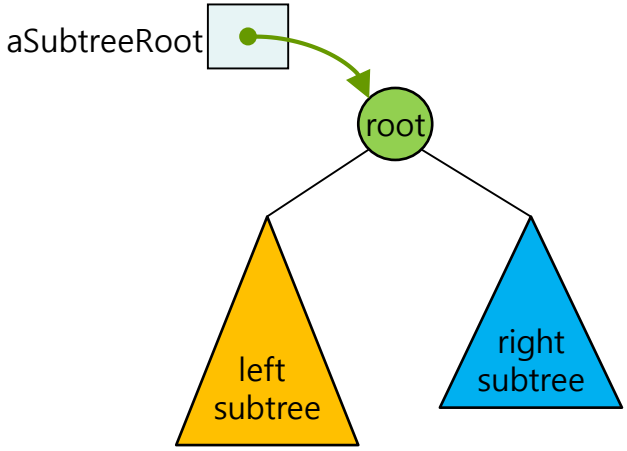
```
public class  BinaryTree<T>
{
    // 비공개 인스턴스 변수
    ……

    // Getter/Setter
    ……

    // 생성자
    ……

    // 트리가 비어있는지 확인
    public boolean  isEmpty ()
    {
        return  (this.root() == null) ;
    }
    // 트리에 새로운 원소를 저장할 공간이 있는지 확인
    public boolean  isFull ()
    {
        return  false  ;
    }
```

BinaryTree<T>: size()




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

      // 트리의 원소의 개수를 돌려준다
      public int  size()
      {
            return  this.sizeOfSubtree (this.root()) ;
      }

      private int  sizeOfSubtree (BinaryNode<T> aSubtreeRoot)
      {
            int  size = 0 ;
            if ( aSubtreeRoot != null ) {
                  int  leftSubtreeSize   = this.sizeOfTree (aSubtreeRoot.leftChild()) ;
                  int  rightSubtreeSize = this.sizeOfTree (aSubtreeRoot.rightChild()) ;
                  size = 1 + (leftSubtreeSize + rightSubtreeSize) ;
            }
            return  size ;
      }
```

BinaryTree<T>: height()



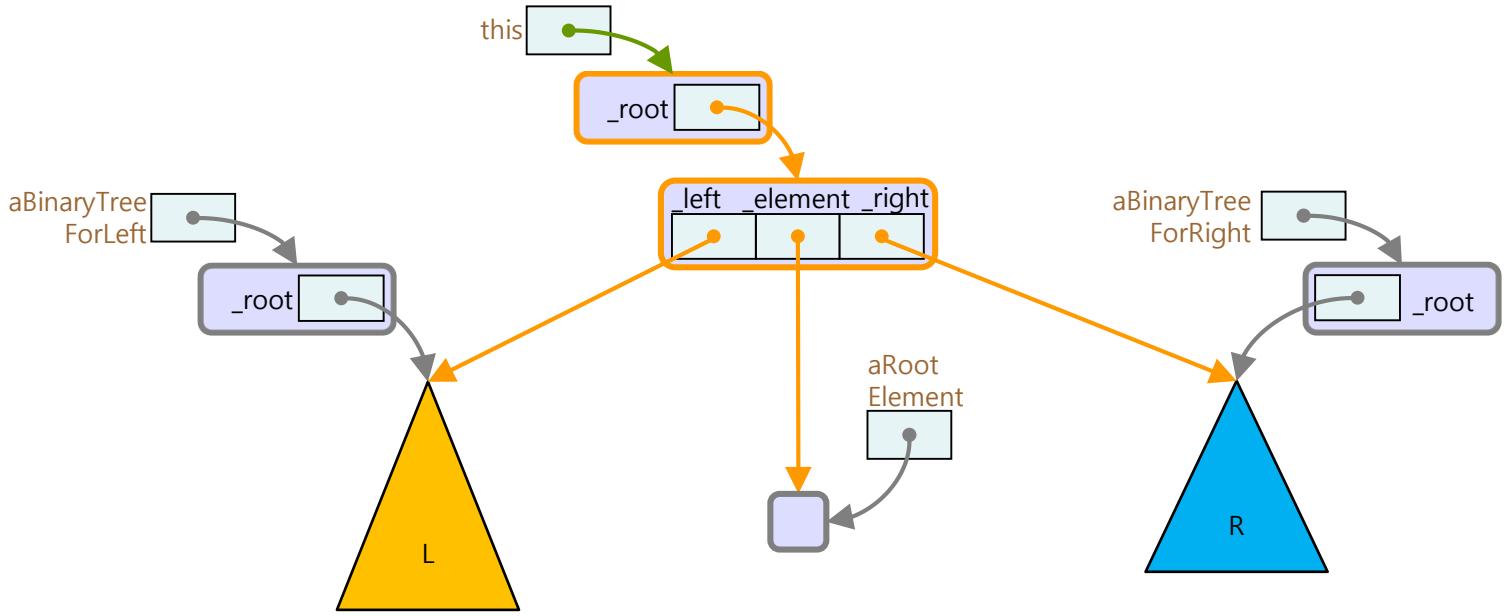

```
public class  BinaryTree<T>
{
      ......

      // 트리의 높이를 돌려준다
      public int  height()
      {
            return  this.heightOfSubtree (this.root()) ;
      }

      private int  heightOfSubtree (BinaryNode<T> aSubtreeRoot)
      {
            int  height = 0 ;
            if ( aSubtreeRoot != null ) {
                  int  leftSubtreeHeight   = this.heightOfSubtree(aSubtreeRoot.leftChild()) ;
                  int  rightSubtreeHeight = this.heightOfSubtree(aSubtreeRoot.rightChild()) ;
                  int  maxSubtreeHeight =
                        (leftSubtreeHeight >= rightSubtreeHeight) ? leftSubtreeHeight : rightSubtreeHeight ;
                  height = 1 + maxSubtreeHeight ;
            }
            return  height ;
      }
```

BinaryTree<T>: rootElement()

```
public class  BinaryTree<T>
{
      ……

      // Root element 를 얻는다
      public T  rootElement () {
            if ( this.root() == null ) {
                  return  null ;
            }
            else {
                  return  this.root().element() ;
            }
      }
```

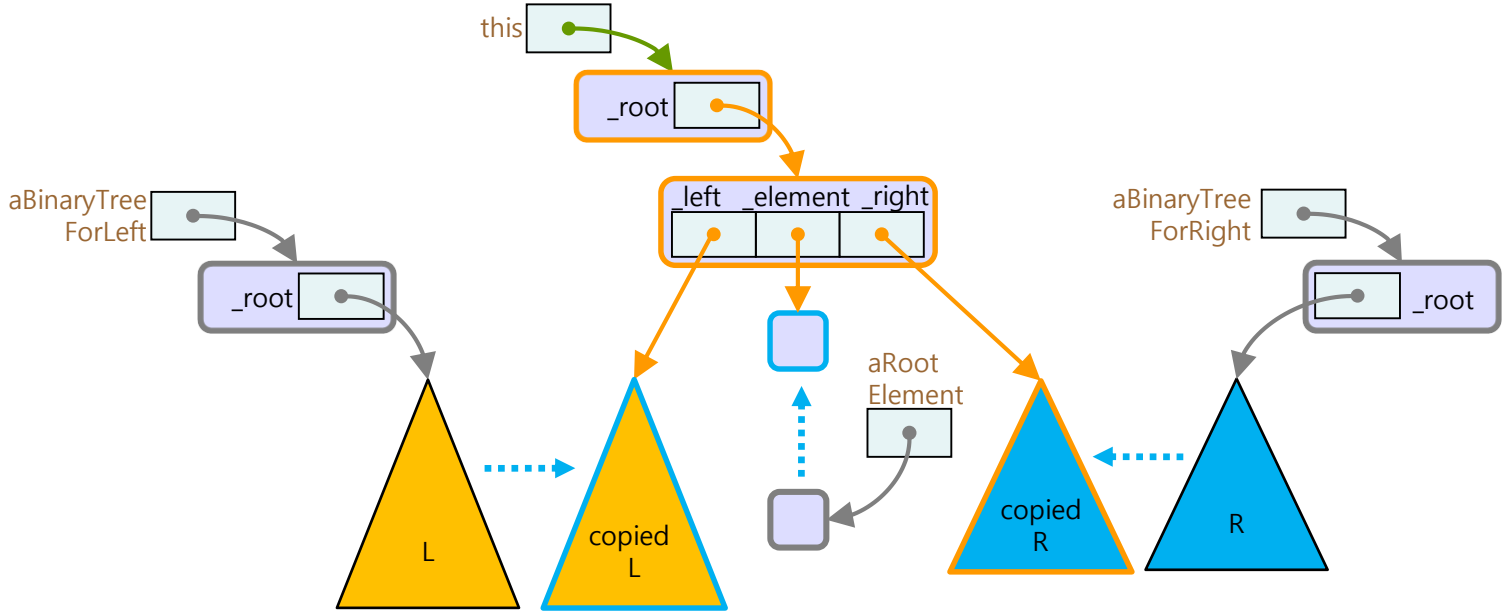
BinaryTree<T>: setTreeByShare()

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

      private void  setTreeByShare ( T                aRootElement,
                                     BinaryTree<T>    aBinaryTreeForLeft,
                                     BinaryTree<T>    aBinaryTreeForRight )
      {
            this.setRoot ( new BinaryNode<T>
                  (aRootElement, aBinaryTreeForLeft.root(), aBinaryTreeForRight.root()) ) ;

      }
```



BinaryTree<T>: setTreeByCopy()

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

      private void  setTreeByCopy ( T                aRootElement,
                                    BinaryTree<T>    aBinaryTreeForLeft,
                                    BinaryTree<T>    aBinaryTreeForRight )
      {
            T  copiedRootElement = aRootElement.copy() ;
            BinaryNode<T> copiedRootOfLeftTree =
                  this.copyBinaryTreeNodes (aBinaryTreeForLeft.root()) ;
            BinaryNode<T> copiedRootOfRightTree =
                  this.copyBinaryTreeNodes (aBinaryTreeForRight.root()) ;
            this.setRoot ( new BinaryNode<T>
               (copiedRootElement, copiedRootOfLeftTree, copiedRootOfRightTree) ) ;
      }
```



BinaryTree<T>: 부트리 노드들의 재귀적 복사

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

      private BinaryNode<T>  copyBinaryTreeNodes (BinaryNode<T> aRoot)
      {
            if ( aRoot == null ) {
                  return  null ;
            }
            else {
                  T  copiedRootElement = aRoot.element.copy() ;
                  BinaryNode<T>  copiedLeftSubtree =
                        this.copyBinaryTreeNodes (aRoot.leftChild()) ;
                  BinaryNode<T>  copiedRightSubtree =
                        this.copyBinaryTreeNodes (aRoot.rightChild()) ;
                  return  new BinaryNode<T>
                        (copiedRootElement, copiedLeftSubtree, copiedRightSubtree) ;
            }
      }
```

BinaryTree<T>: 이진트리의 복사

```
public class  BinaryTree<T>
{
    // 비공개 인스턴스 변수
    ......

    @Override
    public  BinaryTree<T>  copy ()
    {
        BinaryTree<T>  copiedTree = new BinaryTree() ;
        if ( ! this.isEmpty() ) {
            BinaryNode<T> copiedLeftSubtree =
                this.copyBinaryTreeNodes (this.root().leftChild()) ;
            BinaryNode<T> copiedRightSubtree =
                this.copyBinaryTreeNodes (this.root().rightChild()) ;
            copiedTree.setRoot ( new BinaryNode<T>
                (this.root().element().copy(), copiedLeftSubtree, copiedRightSubtree) ) ;
        }
        return  copiedTree ;
    }
```

BinaryTree<T>: 탐색 (Traversal) 함수

```
public class  BinaryTree<T>
{
    ……

    public void  inOrder () {
        return ; // 수정해야 함
    }

    public void  preOrder () {
        return ; // 수정해야 함
    }

    public void  postOrder () {
        return ; // 수정해야 함
    }

    public void  levelOrder () {
        return ; // 수정해야 함
    }
```

End of

"Tree"