

자료구조(Data Structures)

7장. 트리(tree)

담당 교수 : 조 미경

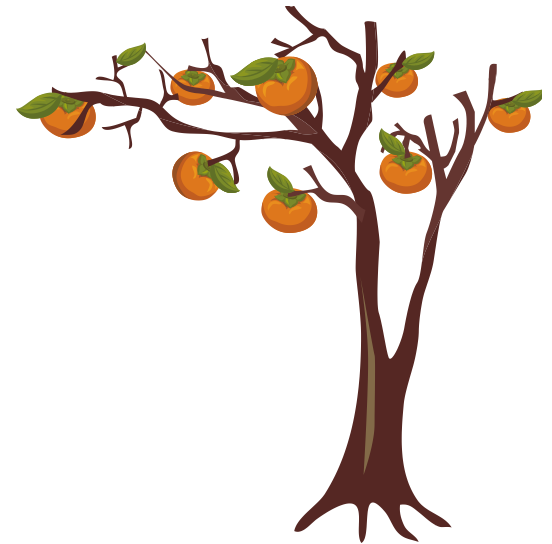
이번 장에서 학습할 내용



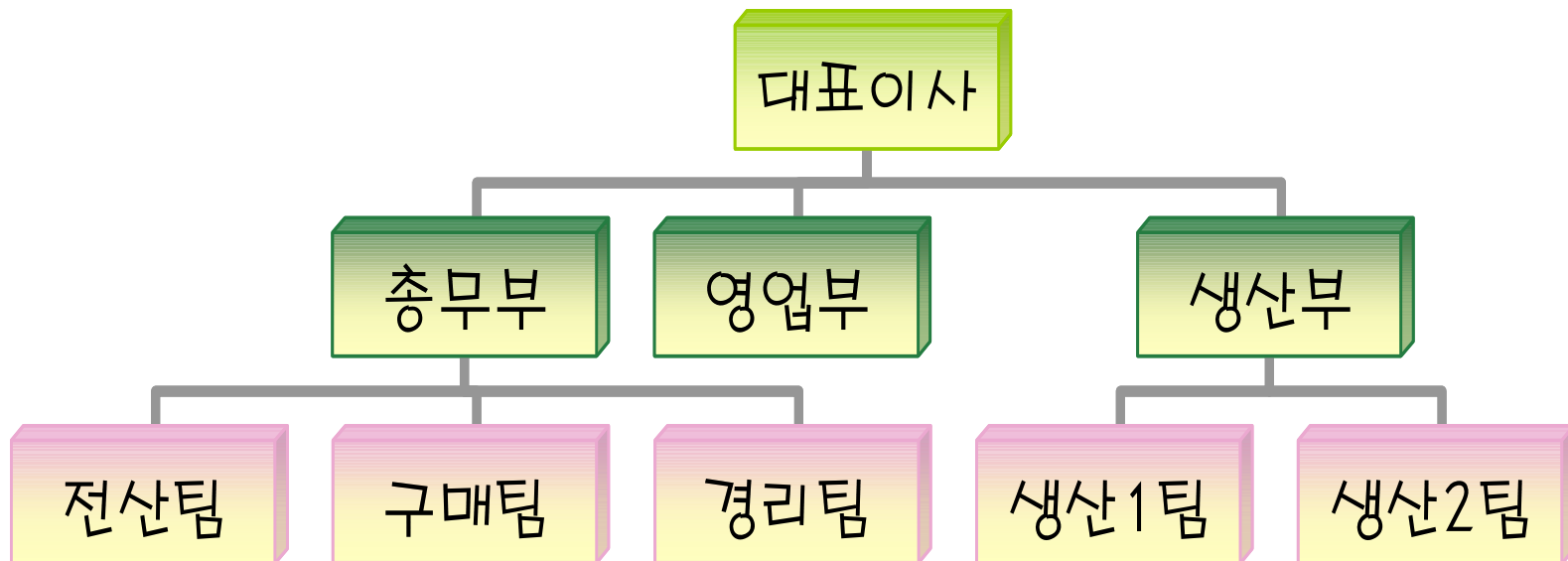
- * 트리란?
- * 이진 트리란?
- * 이진 트리 구현(배열, 연결리스트)
- * 이진 트리 순회(전위, 중위, 후위, 레벨)
- * 스레드 이진 트리
- * 이진 탐색 트리
- * 이진 탐색 트리의 응용: 영어사전

트리(TREE)

- 트리: 계층적인 구조를 나타내는 자료구조
 - ▶ 리스트, 스택, 큐 등은 선형 구조
- 트리는 부모-자식 관계의 노드들로 이루어진다.
- 응용분야:
 - ▶ 계층적인 조직 표현
 - ▶ 컴퓨터 디스크의 디렉토리 구조
 - ▶ 인공지능에서의 결정트리 (decision tree)



회사의 조직



파일 디렉토리 구조

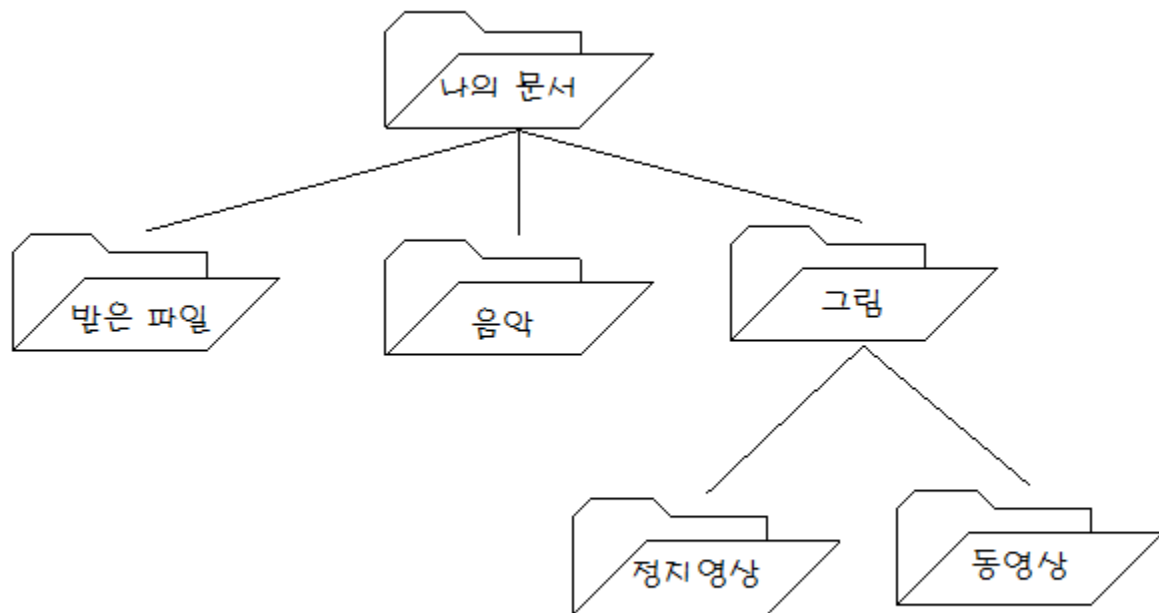


그림 7.2 트리의 예: 컴퓨터 디렉토리

결정 트리

- (예) 골프에 대한 결정 트리

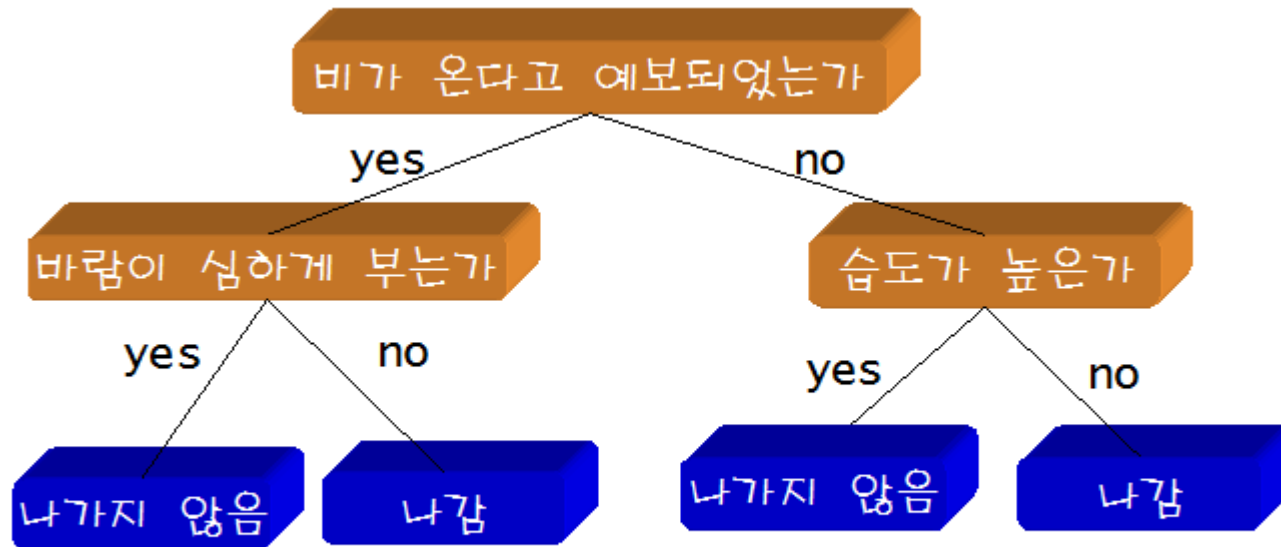
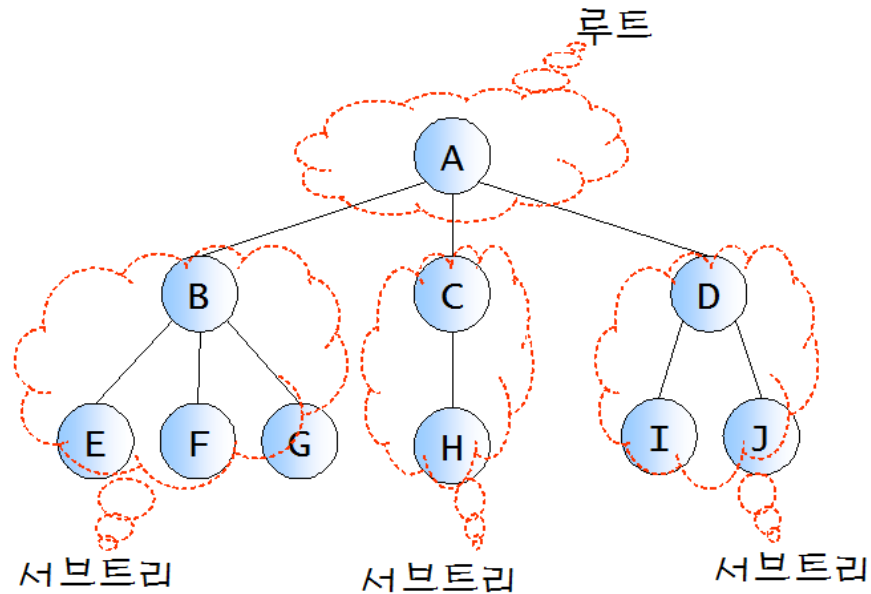


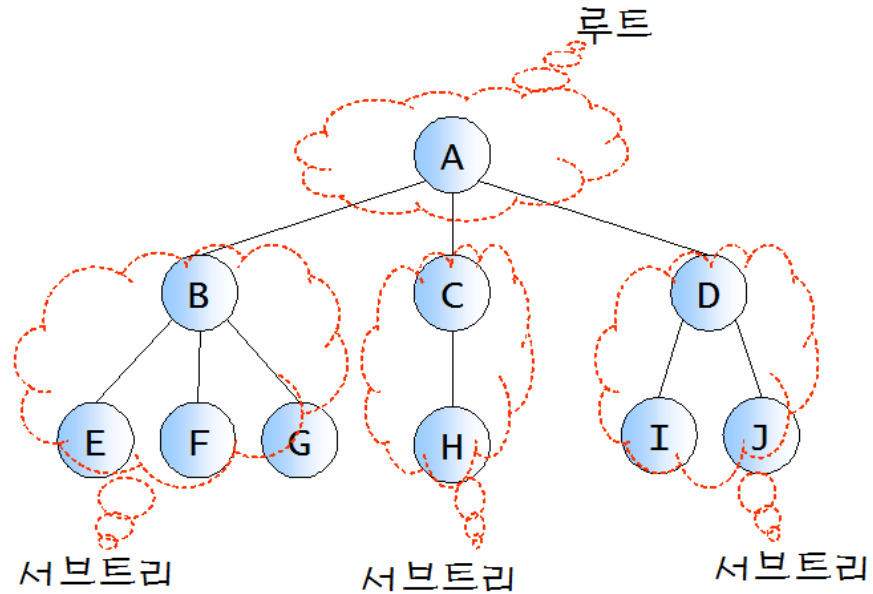
그림 7.3 트리의 예: 결정 트리

트리의 용어



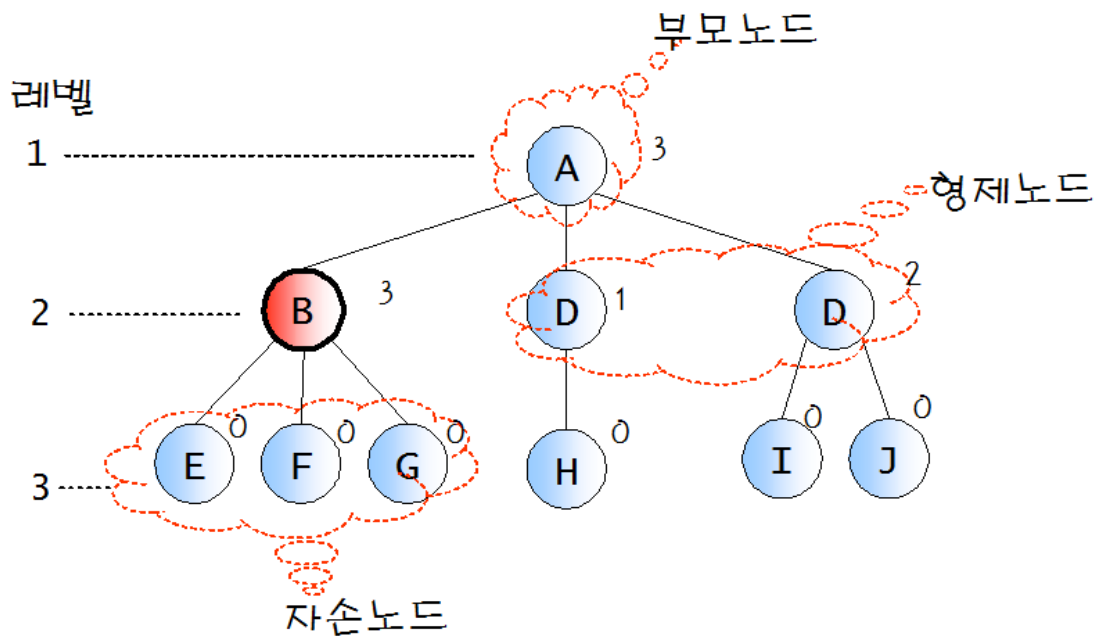
- **노드(node):** 트리의 구성요소
- **루트(root):** 부모가 없는 노드(A)
- **서브트리(subtree):** 하나의 노드와 그 노드들의 자손들로 이루어진 트리

트리의 용어



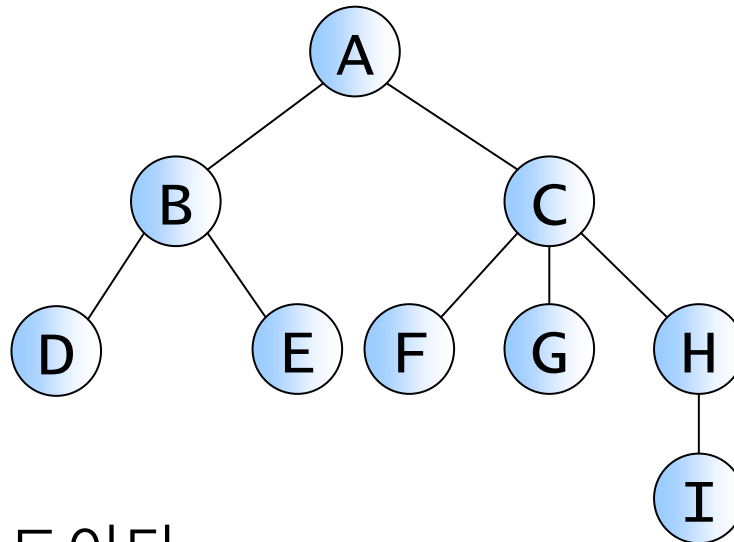
- 단말노드(terminal node): 자식이 없는 노드(A,B,C,D)
- 비단말노드: 적어도 하나의 자식을 가지는 노드(E,F,G,H,I,J)

트리의 용어



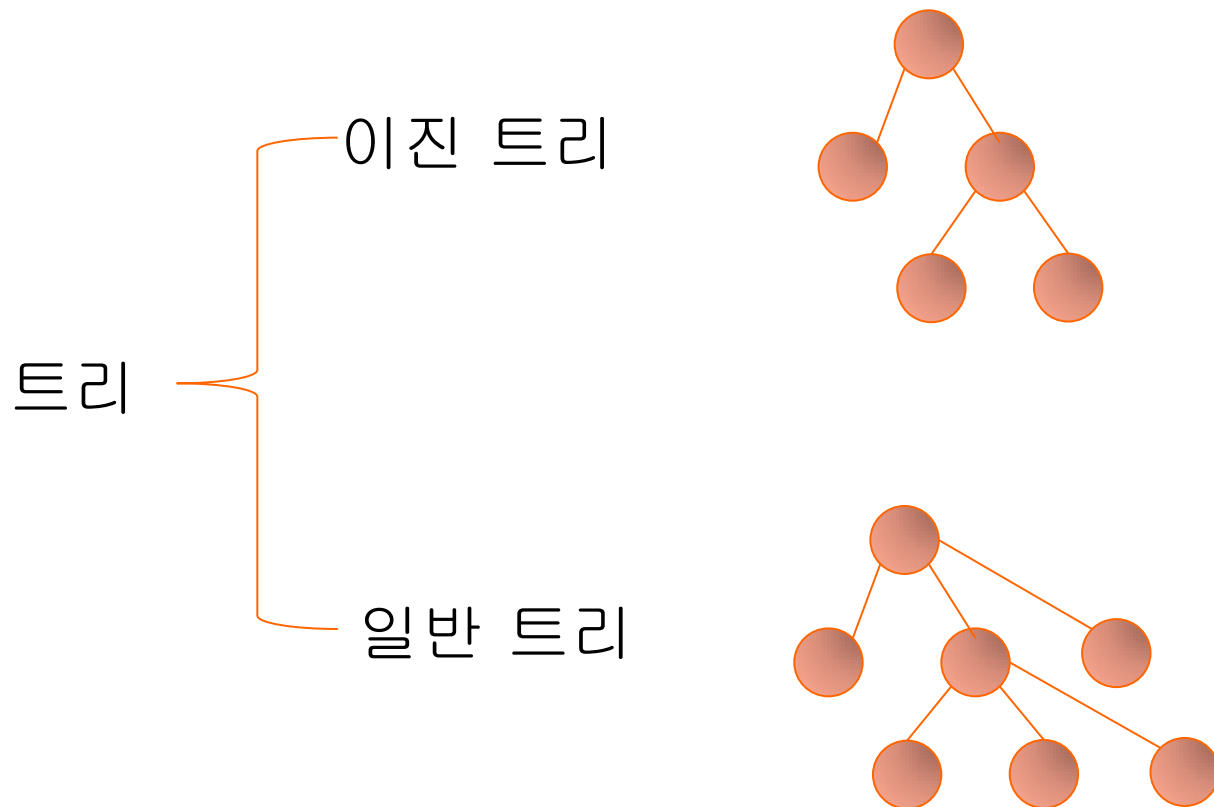
- 자식, 부모, 형제, 조상, 자손 노드: 인간과 동일
- 레벨(level): 트리의 각층의 번호
- 높이(height): 트리의 최대 레벨(3)
- 차수(degree): 노드가 가지고 있는 자식 노드의 개수

예제



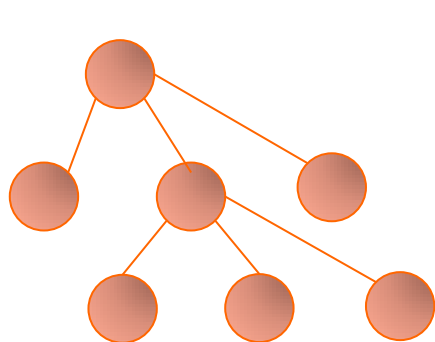
- A는 루트 노드이다.
- B는 D와 E의 부모노드이다.
- C는 B의 형제 노드이다.
- D와 E는 B의 자식노드이다.
- B의 차수는 2이다.
- 위의 트리의 높이는 4이다.

트리의 종류

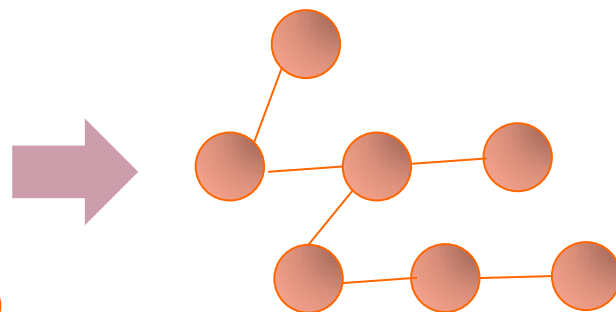


일반 트리를 이진 트리로 변환

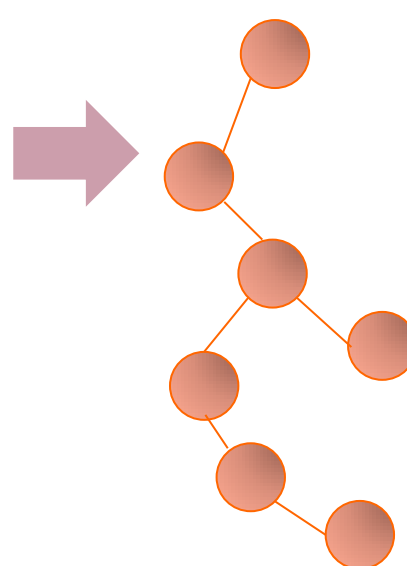
- 일반 트리를 이진 트리로 만드는 방법
 - ▶ 왼쪽노드는 자식 노드로, 오른쪽 노드들은 형제 노드로 배치(Left-Child, Right-Sibling)
- 모든 트리를 이진 트리 형태로 재구성 가능



Left-Child, Right-Sibling

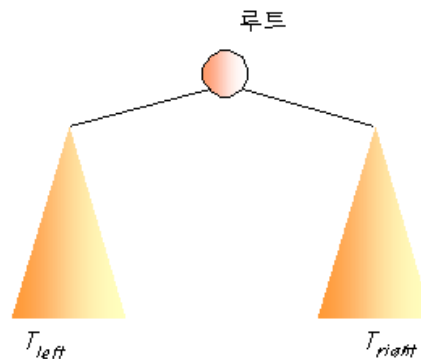


45도 회전



이진트리 (binary tree)

- **이진 트리(binary tree)** : 모든 노드가 2개의 서브 트리를 가지고 있는 트리
 - ▶ 서브트리는 공집합일수 있다.
- 이진트리의 노드에는 최대 2개까지의 자식 노드가 존재
- 모든 노드의 차수가 2 이하가 된다-> 구현하기가 편리함
- 이진 트리에는 서브 트리간의 순서가 존재



이진 트리 검증

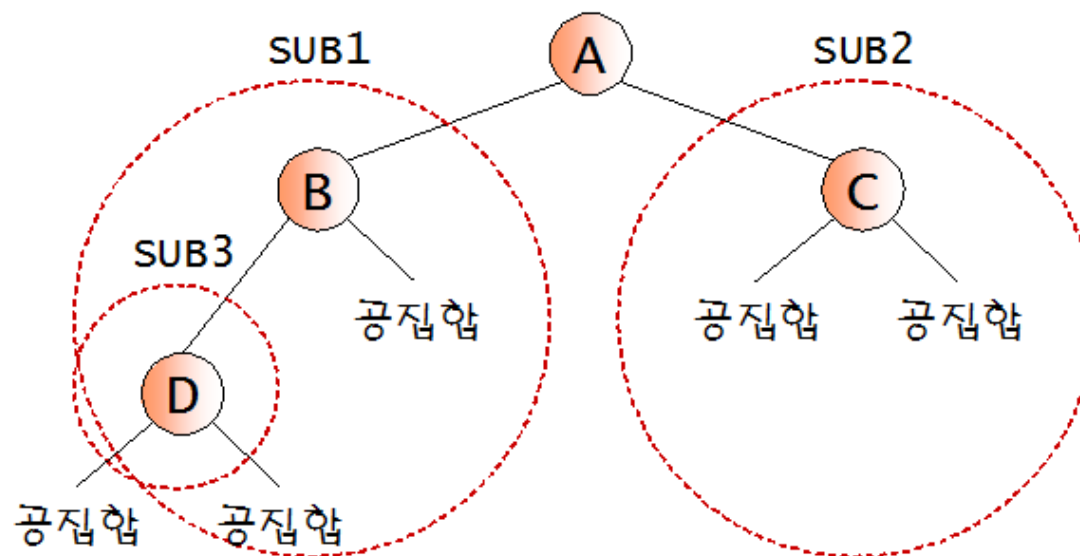


그림 7.8 이진트리 검증

- 이진 트리는 공집합이거나
- 루트와 왼쪽 서브 트리, 오른쪽 서브 트리로 구성된 노드들의 유한 집합으로 정의된다. 이진트리의 서브 트리들은 모두 이진 트리여야 한다.

이진트리의 성질

- 노드의 개수가 n 개이면 간선의 개수는 $n-1$

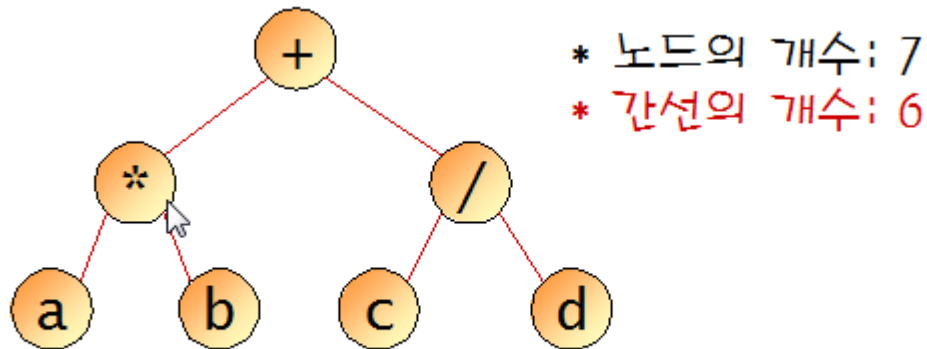
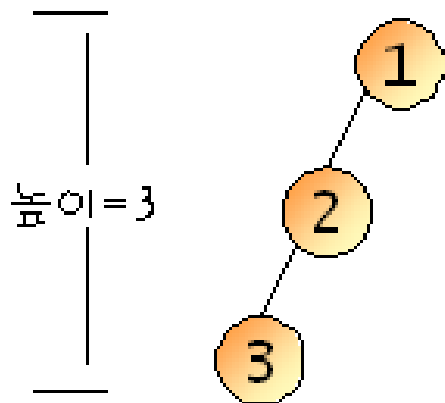


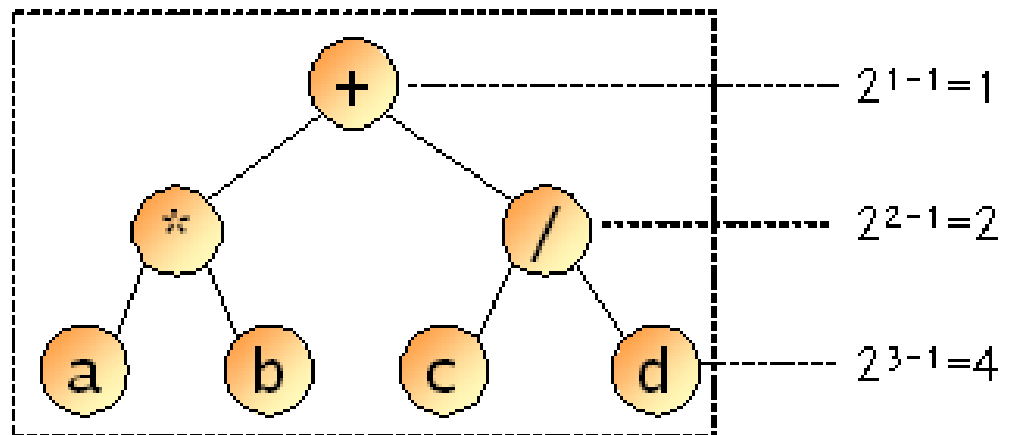
그림 7.10 노드의 개수와 간선의 개수와의 관계

이진트리의 성질

- 높이가 h 인 이진트리의 경우, 최소 h 개의 노드를 가지며 최대 $2^h - 1$ 개의 노드를 가진다.



최소 노드 개수 = 3

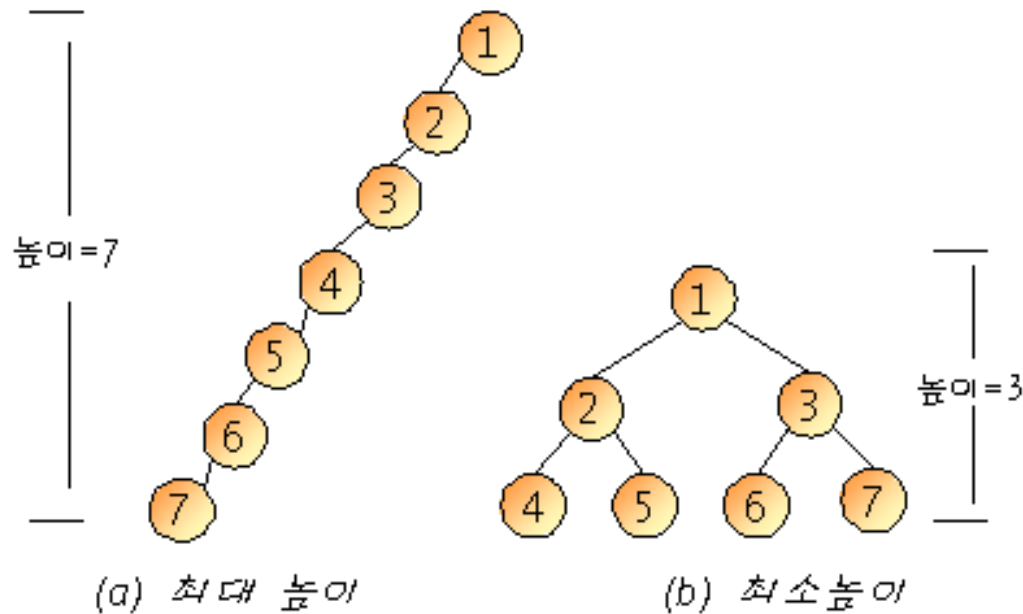


최대 노드 개수 = $2^{1-1} + 2^{2-1} + 2^{3-1} = 1 + 2 + 4 = 7$

이진트리의 성질

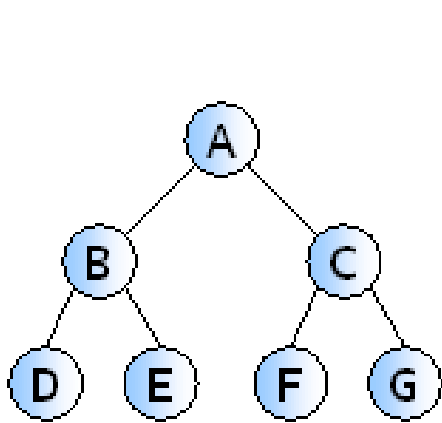
- n 개의 노드를 가지는 이진트리의 높이

- ▶ 최대 n
- ▶ 최소 $\lceil \log_2(n+1) \rceil$

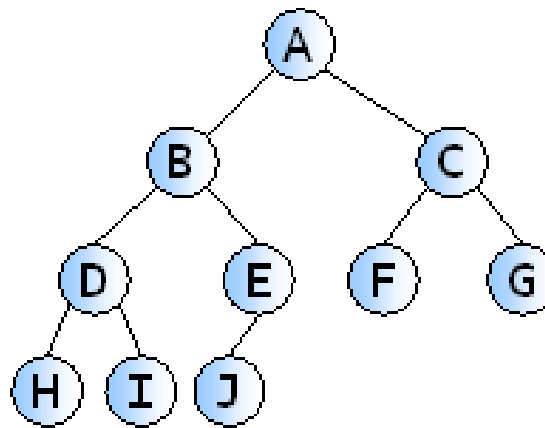


이진트리의 분류

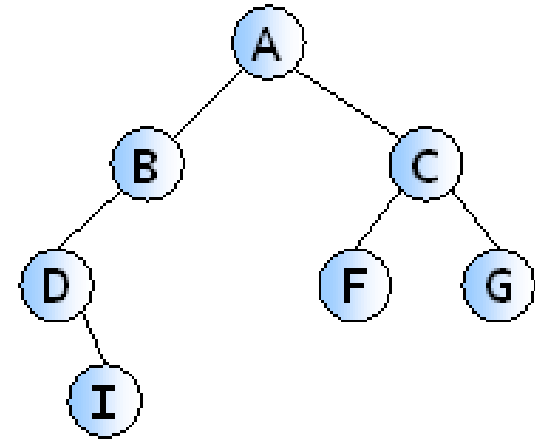
- 포화 이진 트리(full binary tree)
- 완전 이진 트리(complete binary tree)
- 기타 이진 트리



(a) 포화 이진 트리



(b) 완전 이진 트리



(c) 기타 이진 트리

포화 이진 트리

- 용어 그대로 트리의 각 레벨에 노드가 꽉 차있는 이진트리를 의미한다.

$$\text{전체 노드 개수} : 2^{1-1} + 2^{2-1} + 2^{3-1} + \dots + 2^{k-1} = \sum_{i=0}^{k-1} 2^i = 2^k - 1$$

- 포화 이진 트리에는 다음과 같이 각 노드에 번호를 붙일 수 있다.

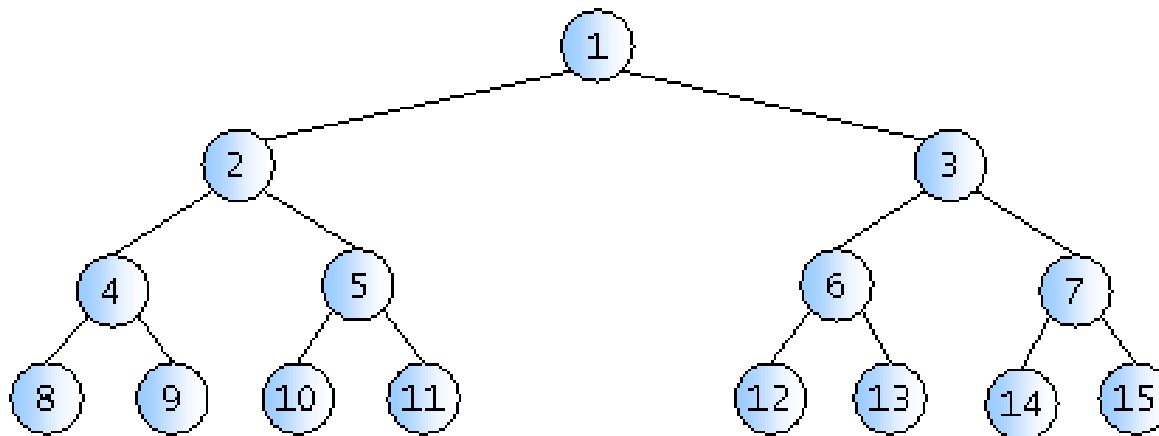
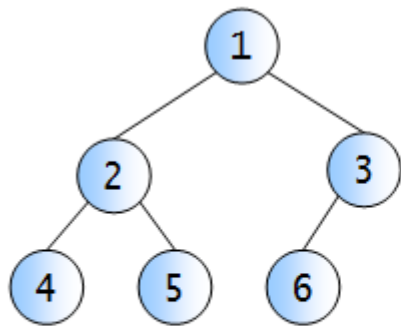


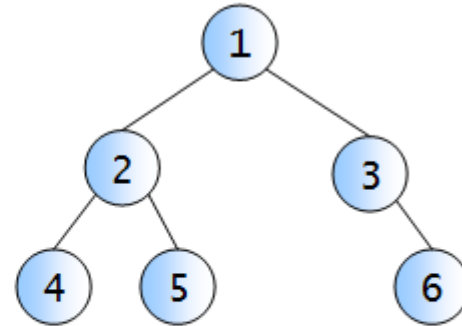
그림 7.15 포화이진트리에서의 노드의 번호

완전 이진 트리

- **완전 이진 트리(complete binary tree):** 레벨 1부터 $k-1$ 까지는 노드가 모두 채워져 있고 마지막 레벨 k 에서는 왼쪽부터 오른쪽으로 노드가 순서대로 채워져 있는 이진트리
- 포화 이진 트리와 노드 번호가 일치



(a) 완전이진트리



(b) 완전이진트리가 아님

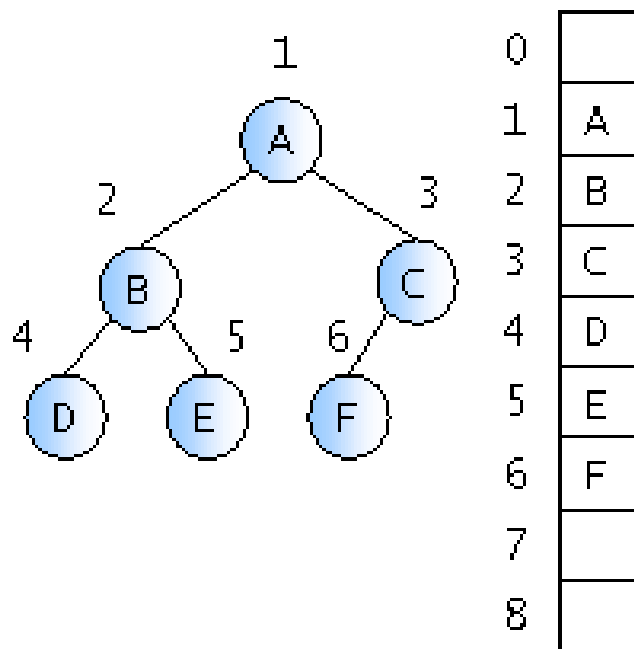
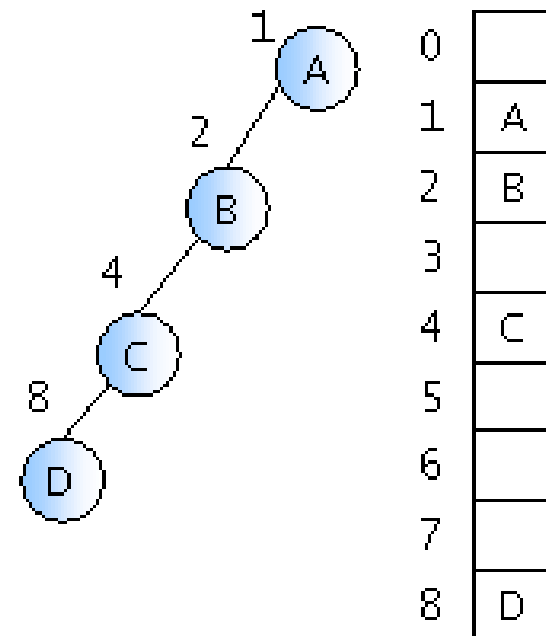
그림 7.16 완전 이진트리의 예

이진 트리의 표현

- 배열을 이용하는 방법
- 포인터를 이용하는 방법

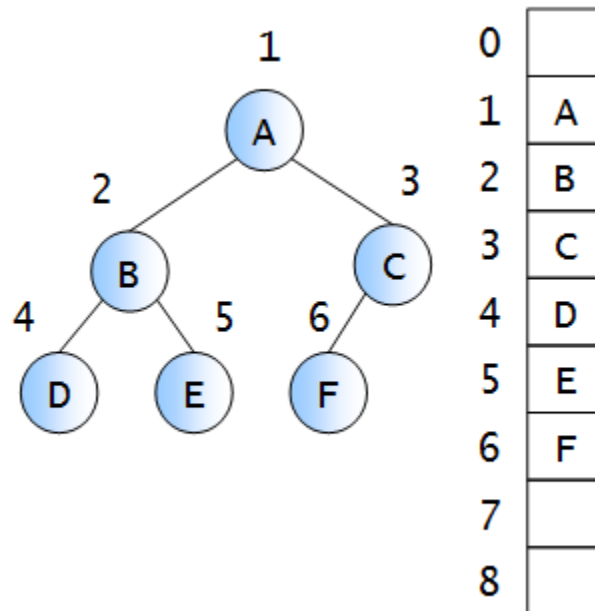
배열 표현법

- 배열표현법: 모든 이진 트리를 포화 이진 트리라고 가정하고 각 노드에 번호를 붙여서 그 번호를 배열의 인덱스로 삼아 노드의 데이터를 배열에 저장하는 방법

(a) 완전 이진 트리(b) 경사 이진 트리

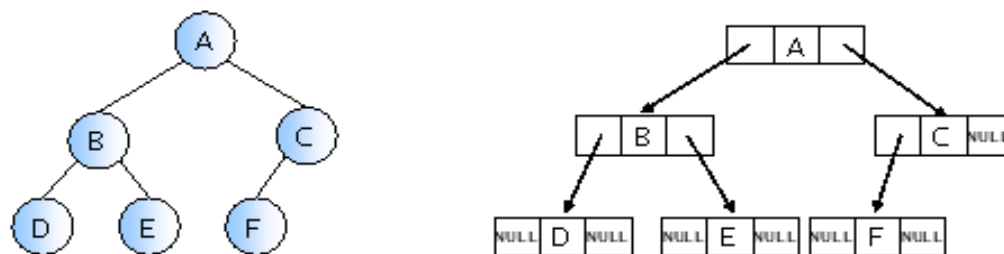
부모와 자식 인덱스 관계

- 노드 i 의 부모 노드 인덱스 = $i/2$
- 노드 i 의 왼쪽 자식 노드 인덱스 = $2*i$
- 노드 i 의 오른쪽 자식 노드 인덱스 = $2*i+1$

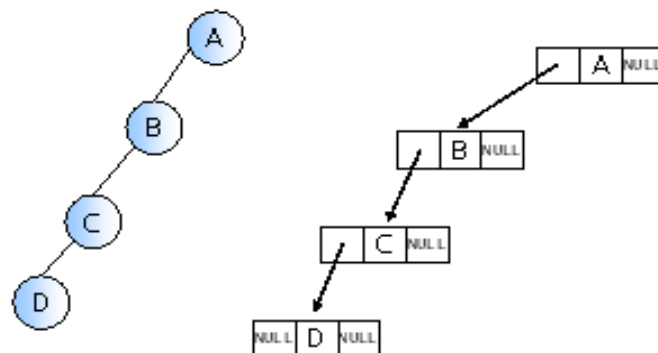


링크 표현법

- 링크 표현법: 포인터를 이용하여 부모 노드가 자식 노드를 가리키게 하는 방법



(a) 완전 이진 트리



(b) 경사 이진 트리

링크의 구현

- 노드는 구조체로 표현
- 링크는 포인터로 표현

```
typedef struct TreeNode {  
    int data;  
    struct TreeNode *left, *right;  
} TreeNode;
```

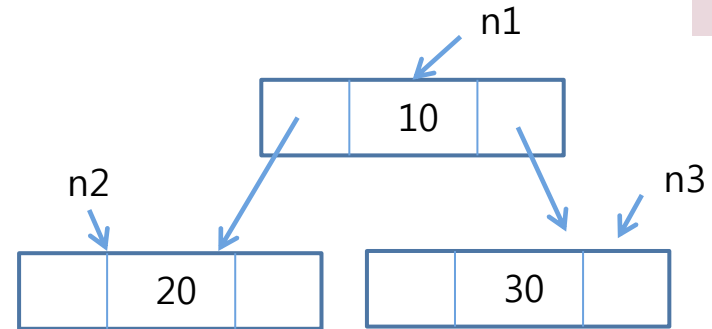
링크 표현법 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

typedef struct TreeNode {
    int data;
    struct TreeNode *left, *right;
} TreeNode;

void main()
{
    TreeNode *n1, *n2, *n3;

    n1= (TreeNode *)malloc(sizeof(TreeNode));
    n2= (TreeNode *)malloc(sizeof(TreeNode));
    n3= (TreeNode *)malloc(sizeof(TreeNode));
```



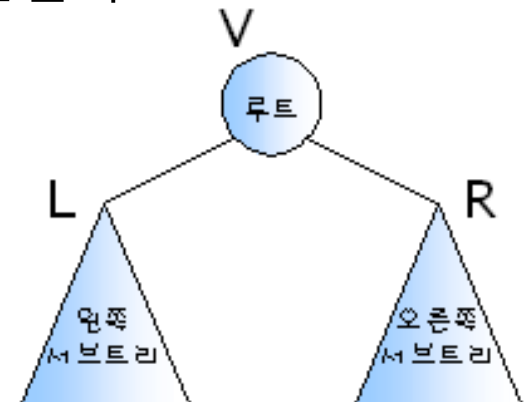
```
n1->data = 10;
n1->left = n2;
n1->right = n3;

n2->data = 20;
n2->left= n2->right = NULL;

n3->data = 30;
n3->left = n3->right = NULL;
}
```

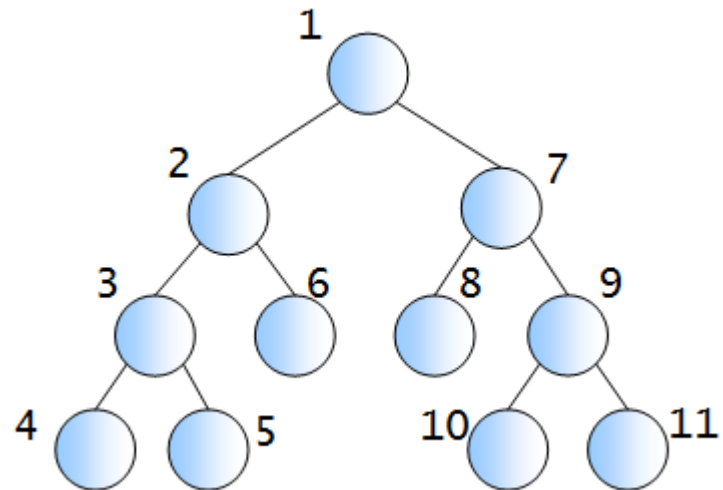
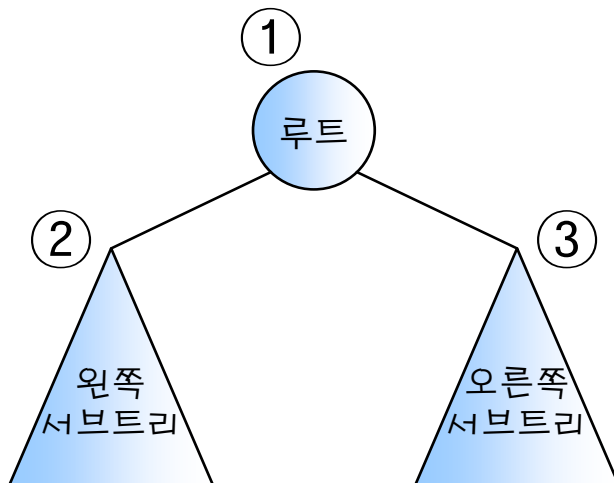
이진 트리의 순회

- 순회(traversal): 트리의 노드들을 체계적으로 방문하는 것
- 3가지의 기본적인 순회방법
 - ▶ 전위순회(preorder traversal) : VLR
 - ▶ 자손 노드보다 루트 노드를 먼저 방문한다.
 - ▶ 중위순회(inorder traversal) : LVR
 - ▶ 왼쪽 자손, 루트, 오른쪽 자손 순으로 방문한다.
 - ▶ 후위순회(postorder traversal) : LRV
 - ▶ 루트 노드보다 자손을 먼저 방문한다.



전위 순회

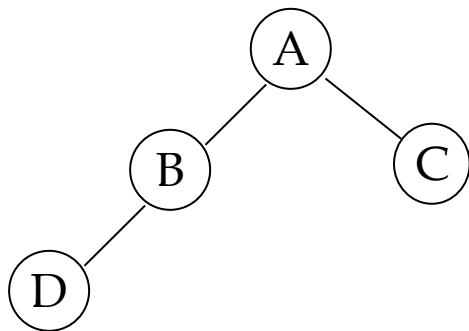
1. 루트 노드를 방문한다
2. 왼쪽 서브트리를 방문한다
3. 오른쪽 서브트리를 방문한다



전위순회 프로그램

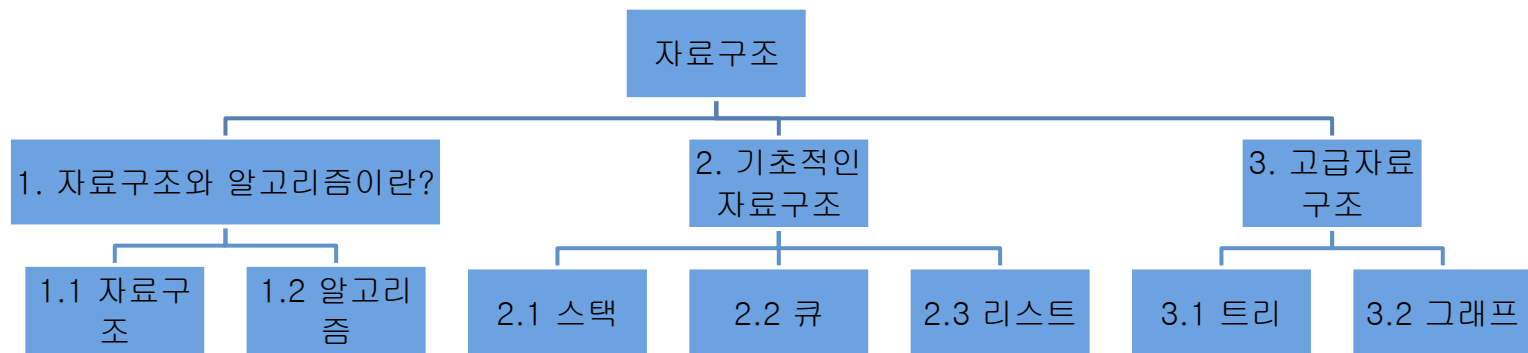
- 순환 호출을 이용한다.

```
preorder(x)  
if  $x \neq \text{NULL}$   
    then    print DATA(x);  
            preorder(LEFT(x));  
            preorder(RIGHT(x));
```



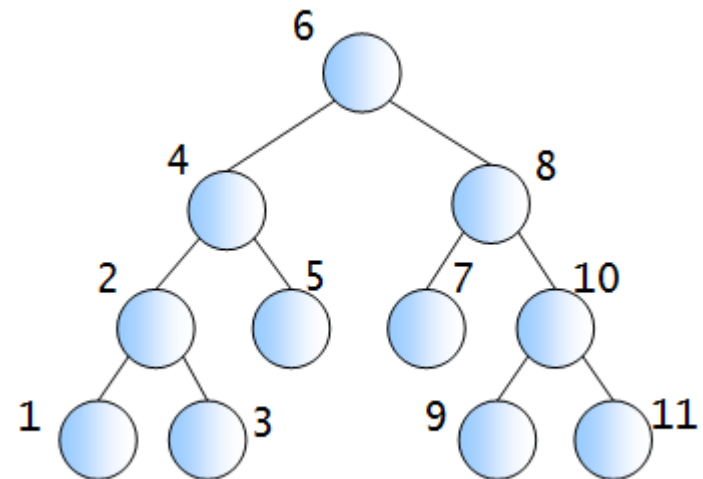
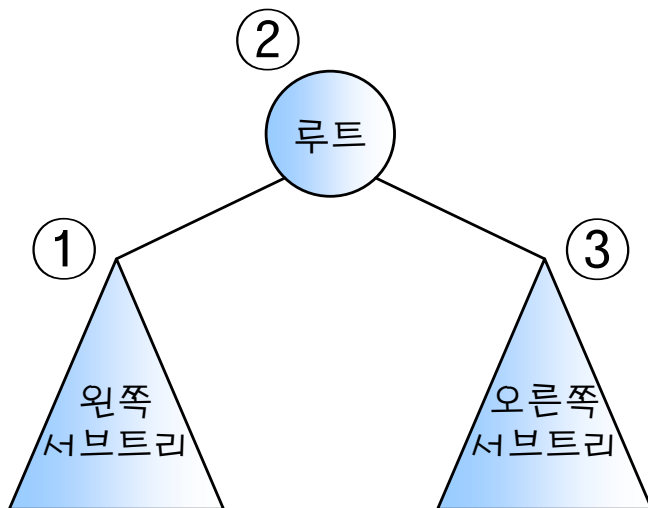
전위 순회 응용

- (예) 구조화된 문서출력



중위 순회

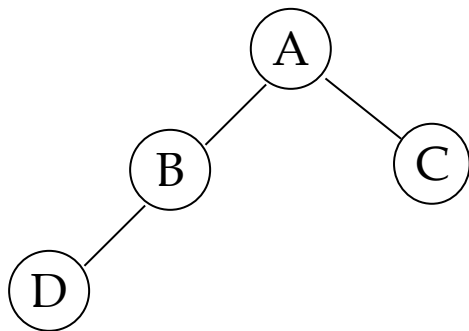
1. 왼쪽 서브 트리를 방문한다
2. 루트 노드를 방문한다
3. 오른쪽 서브 트리를 방문한다



중위순회 알고리즘

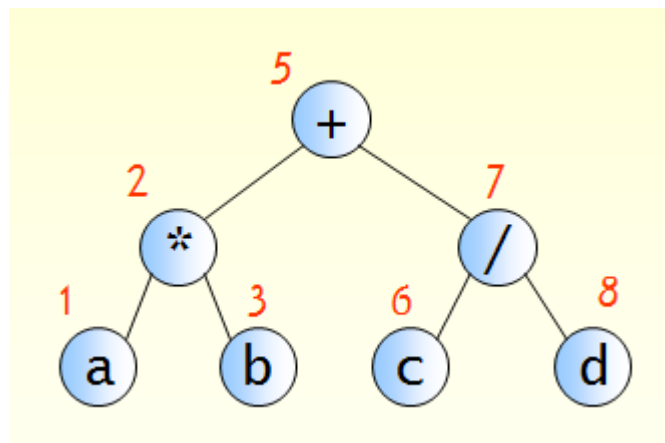
- 순환 호출을 이용한다.

```
inorder(x)  
if  $x \neq \text{NULL}$   
    then     $\text{inorder}(\text{LEFT}(x));$   
            $\text{print DATA}(x);$   
            $\text{inorder}(\text{RIGHT}(x));$ 
```



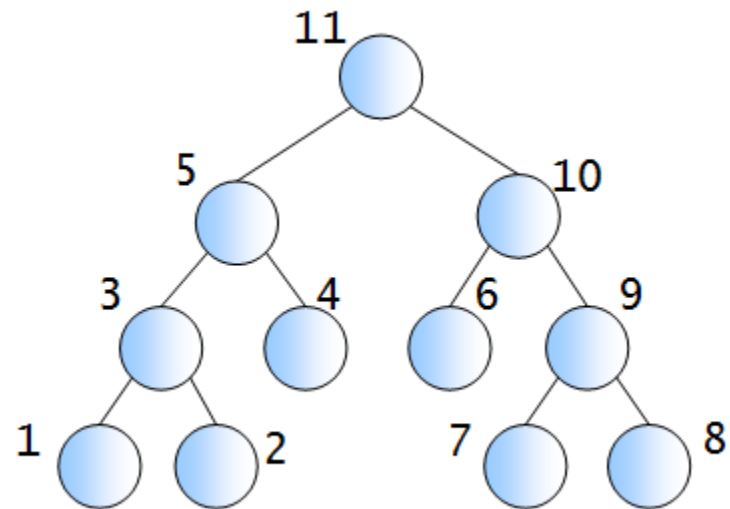
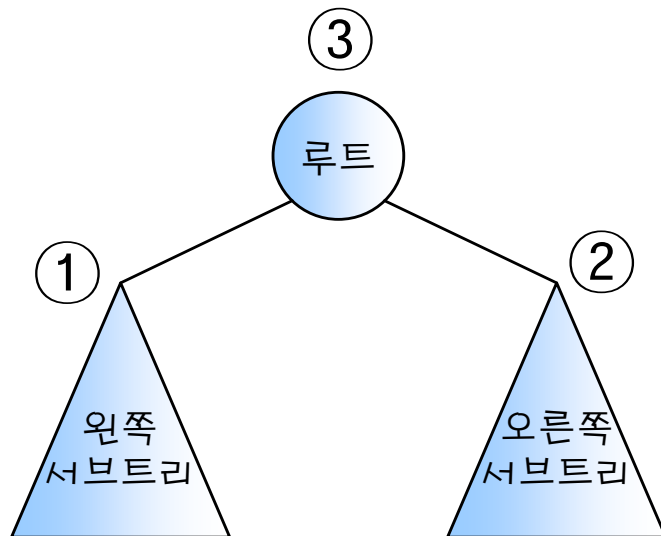
중위 순회 응용

- (예) 수식 트리



후위 순회

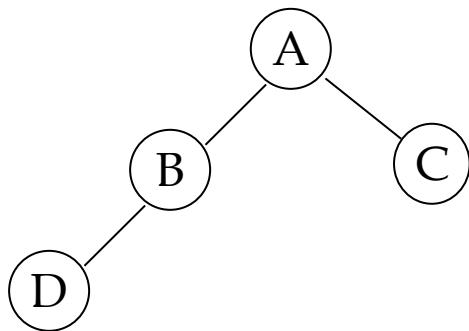
1. 왼쪽 서브트리를 방문한다
2. 오른쪽 서브트리를 방문한다
3. 루트 노드를 방문한다



후위순회 알고리즘

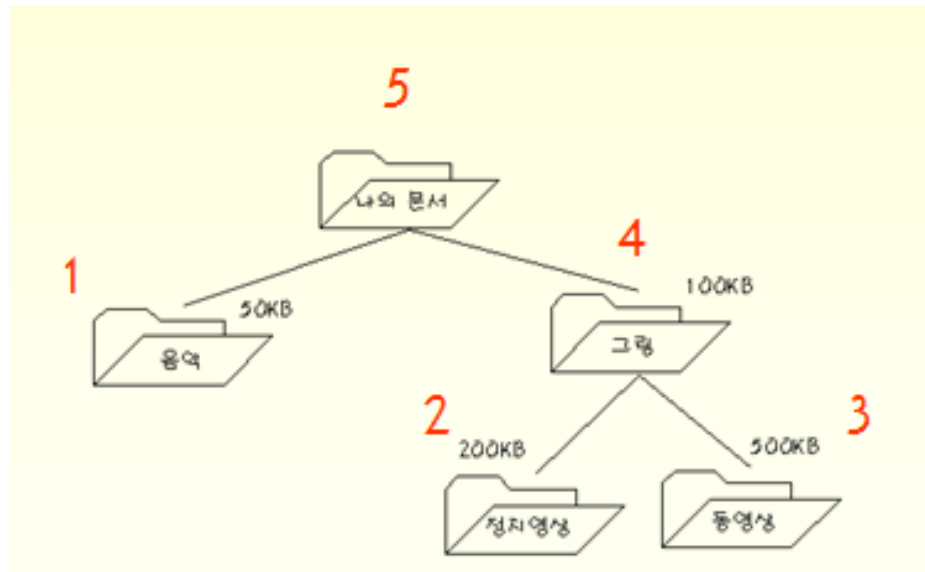
- 순환 호출을 이용한다.

```
postorder(x)  
if  $x \neq \text{NULL}$   
    then    postorder(LEFT(x));  
           postorder(RIGHT(x));  
           print DATA(x);
```



후위 순회 응용

- (예) 디렉토리 용량 계산



순회 프로그램

```
typedef struct TreeNode {
    int data;
    struct TreeNode *left, *right;
} TreeNode;

//           15
//      4       20
//    1       16       25
TreeNode n1={1, NULL, NULL};
TreeNode n2={4, &n1, NULL};
TreeNode n3={16, NULL, NULL};
TreeNode n4={25, NULL, NULL};
TreeNode n5={20, &n3, &n4};
TreeNode n6={15, &n2, &n5};
TreeNode *root= &n6;
```

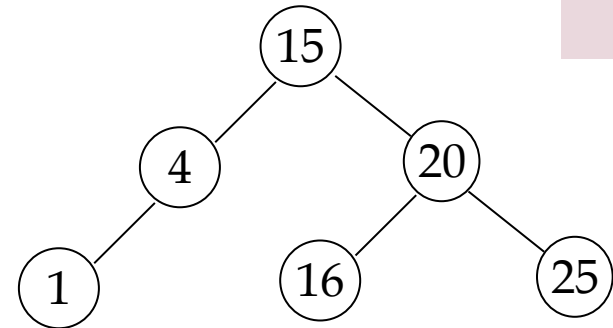
```
// 중위 순회
void inorder( TreeNode *root ){
    if ( root ){
        inorder( root->left ); // 왼쪽서브트리 순회
        printf("%3d", root->data ); // 노드 방문
        inorder( root->right ); // 오른쪽서브트리 순회
    }
}

// 전위 순회
void preorder( TreeNode *root ){
    if ( root ){
        printf("%3d", root->data ); //노드 방문
        preorder( root->left ); //왼쪽서브트리 순회
        preorder( root->right ); //오른쪽서브트리 순회
    }
}
```

순회 프로그램

```
// 후위 순회
void postorder( TreeNode *root ){
    if ( root ){
        postorder( root->left ); // 왼쪽서브트리 순회
        postorder( root->right ); // 오른쪽서브트리 순회
        printf("%3d", root->data ); // 노드 방문
    }
}

void main()
{
    printf("##중위순회 결과: "); inorder(root);
    printf("\n##전위순회 결과: "); preorder(root);
    printf("\n##후위순회 결과: "); postorder(root);
}
```



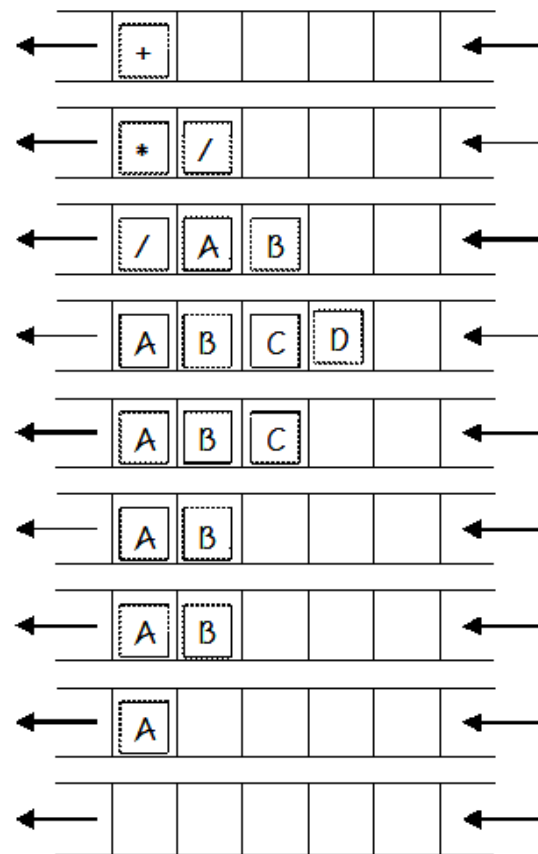
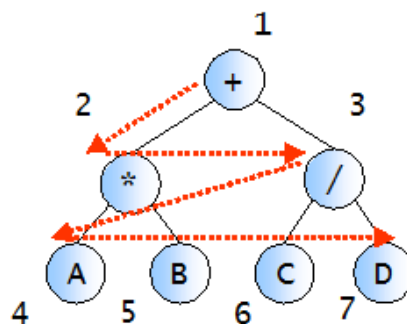
```

C:\Windows\system32\cmd.exe
##중위순회 결과: 1 4 15 16 20 25
##전위순회 결과: 15 4 1 20 16 25
##후위순회 결과: 1 4 16 25 20 15
  
```

레벨 순회

- **레벨 순회(level order)**는 각 노드를 레벨 순으로 검사하는 순회 방법

- 지금까지의 순회법이 스택을 사용했던 것에 비해 레벨 순회는 큐를 사용하는 순회법이다.

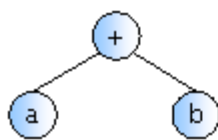


레벨 순회 알고리즘

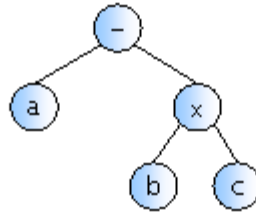
- **level_order(root)**
 1. initialize queue;
 2. enqueue(queue, root);
 3. while is_empty(queue)≠TRUE do
 4. $x \leftarrow \text{dequeue}(\text{queue});$
 5. if($x \neq \text{NULL}$) then
 6. print DATA(x);
 7. enqueue(queue, LEFT(x));
 8. enqueue(queue, RIGHT(x));

수식 트리

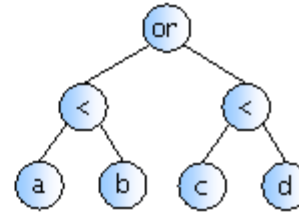
- 수식트리: 산술식을 트리형태로 표현한 것
 - ▶ 비단말노드: 연산자(operator)
 - ▶ 단말노드: 피연산자(operand)
- 예)



(a)



(b)

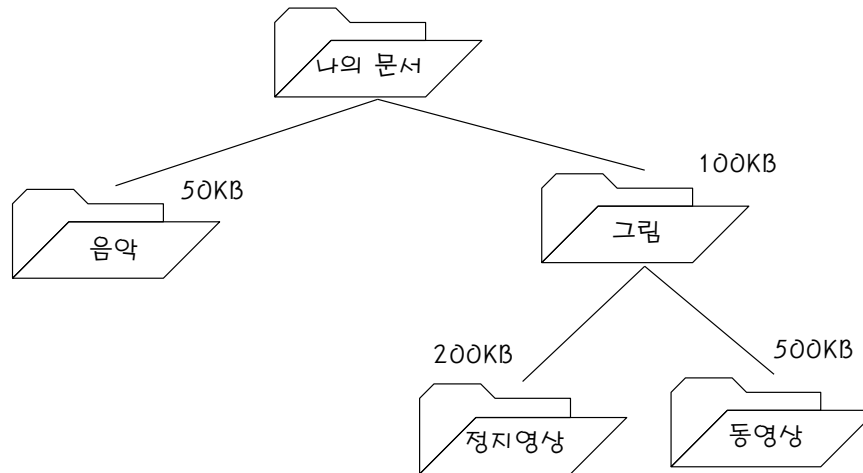


(c)

수식	$a + b$	$a - (b \times c)$	$(a < b) \text{ or } (c < d)$
전위순회	$+ a b$	$- a \times b c$	$\text{or} < a b < c d$
중위순회	$a + b$	$a - b \times c$	$a < b \text{ or } c < d$
후위순회	$a b +$	$a b c \times -$	$a b < c d < \text{or}$

디렉토리 용량 계산

- 디렉토리의 용량을 계산하는데 후위 트리 순회 사용



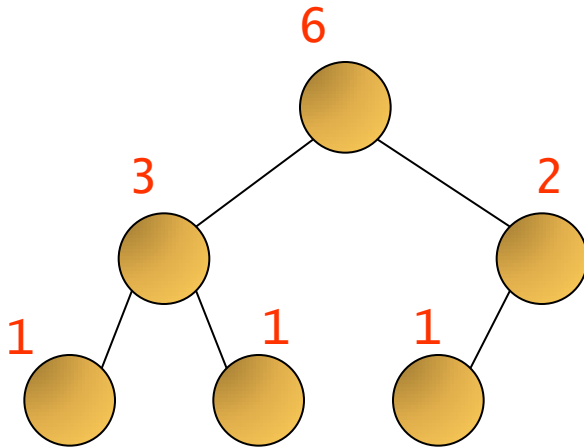
디렉토리 용량 계산 프로그램

```
int calc_dir_size(TreeNode *root)
{
    int left_dir, right_dir;
    if ( root ){
        left_size = calc_size( root->left );
        right_size = calc_size(root->right );
        return (root->data+left_size+right_size);
    }
}

void main()
{
    TreeNode n4={500, NULL, NULL};
    TreeNode n5={200, NULL, NULL};
    TreeNode n3={100, &n4, &n5};
    TreeNode n2={50, NULL, NULL};
    TreeNode n1={0, &n2, &n3};
    printf("디렉토리의 크기=%d\n",calc_dir_size(&n1));
}
```

이진 트리 연산: 노드 개수

- 탐색 트리안의 노드의 개수를 계산
- 각각의 서브트리에 대하여 순환 호출한 다음, 반환되는 값에 1을 더하여 반환

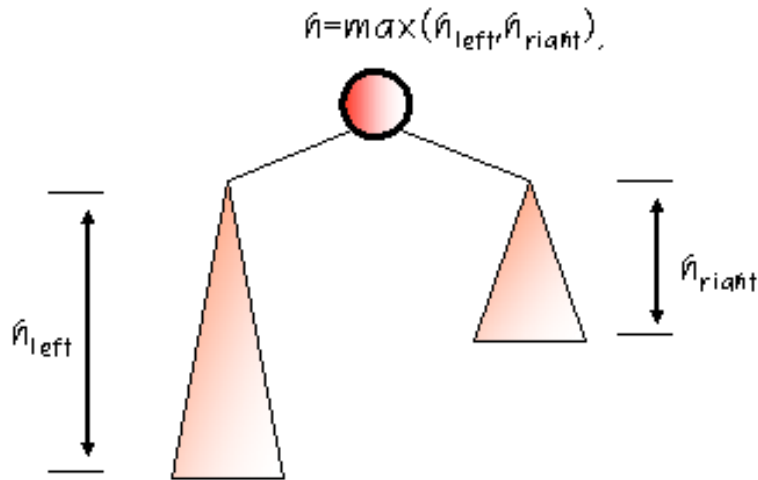


```
int get_node_count(TreeNode *node)
{
    int count=0;
    if( node != NULL )
        count = 1 + get_node_count(node->left)+
                get_node_count(node->right);
    return count;
}
```

이진 트리 연산: 높이

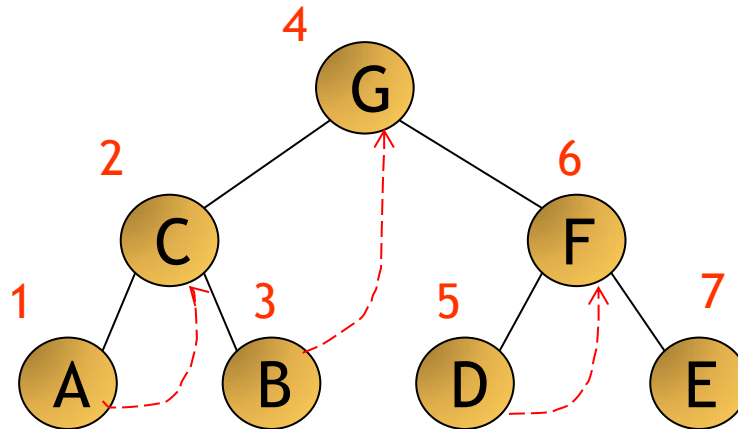
- 서브트리에 대하여 순환 호출하고 서브 트리들의 반환값 중에서 최대값을 구하여 반환

```
int get_height(TreeNode *node)
{
    int height=0;
    if( node != NULL )
        height = 1 + max(get_height(node->left),
                        get_height(node->right));
    return height;
}
```



스레드 이진 트리

- 이진트리의 NULL 링크를 이용하여 순환호출 없이도 트리의 노드들을 순회
- NULL 링크에 중위 순회시에 후속 노드인 **중위 후속자 (inorder successor)**를 저장시켜 놓은 트리가 **스레드 이진 트리 (threaded binary tree)**



스레드 이진 트리의 구현

- 단말노드와 비단말노드의 구별을 위하여 is_thread 필드 필요

```
typedef struct TreeNode {  
    int data;  
    struct TreeNode *left, *right;  
    int is_thread; //만약 오른쪽 링크가 스레드이면 TRUE  
} TreeNode;
```

스레드 이진 트리의 구현

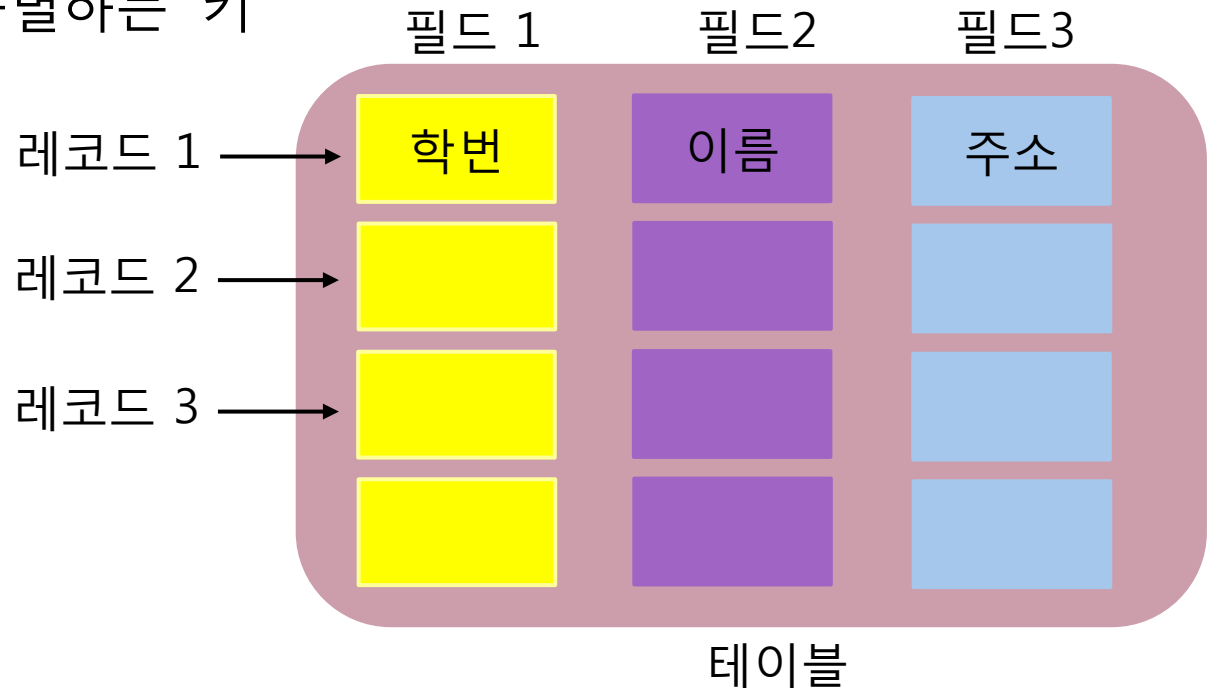
● 스레드 버전 중위 순회 함수 작성

```
void thread_inorder(TreeNode *t){
    TreeNode *q;
    q=t;
    while (q->left) q = q->left; //가장 왼쪽 노드로 간다.
    do{
        printf("%c ", q->data); //데이터 출력
        q = find_successor(q); //후속자 함수 호출
    } while(q); // NULL이 아니면
}
```

```
TreeNode *find_successor(TreeNode *p) {
    TreeNode *q = p->right; // q는 p의 오른쪽 포인터
    // 만약 오른쪽 포인터가 NULL이거나 스레드이면 오른쪽 포인터를 반환
    if( q==NULL || p->is_thread == TRUE)
        return q;
    // 만약 오른쪽 자식이면 다시 가장 왼쪽 노드로 이동
    while( q->left != NULL ) q = q->left;
    return q;
}
```


이진탐색트리

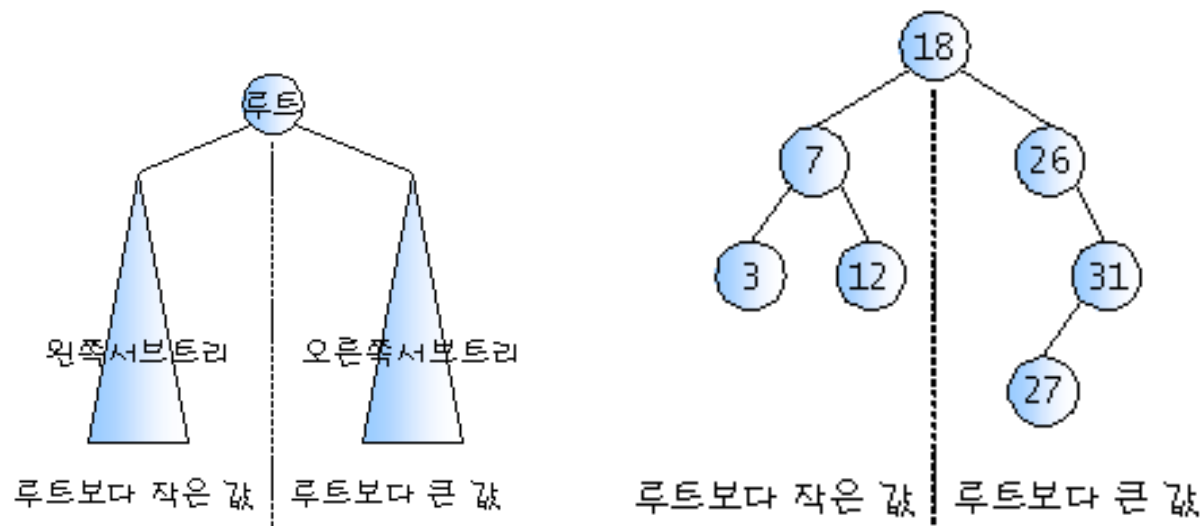
- 탐색작업을 효율적으로 하기 위한 자료구조
- 탐색은 레코드집합에서 특정한 레코드를 찾아내는 작업
- 주요키(primary key)
 - ▶ 각 레코드를 구별하는 키



이진탐색트리

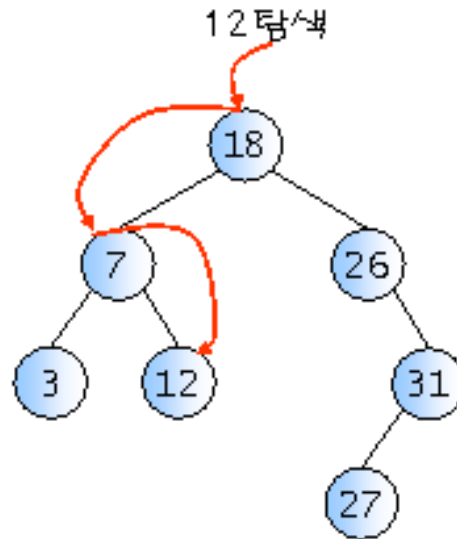
● 이진 탐색 트리의 정의

- ▶ 모든 노드의 키는 유일하다.
- ▶ 왼쪽 서브 트리의 키들은 루트의 키보다 작다.
- ▶ 오른쪽 서브 트리의 키들은 루트의 키보다 크다.
- ▶ 왼쪽과 오른쪽 서브 트리도 이진 탐색 트리이다.



이진탐색트리에서의 탐색연산

- 비교한 결과가 같으면 탐색이 성공적으로 끝난다.
- 비교한 결과가, 주어진 키 값이 루트 노드의 키값보다 작으면 탐색은 이 루트 노드의 왼쪽 자식을 기준으로 다시 시작한다.
- 비교한 결과가, 주어진 키 값이 루트 노드의 키 값보다 크면 탐색은 이 루트 노드의 오른쪽 자식을 기준으로 다시 시작한다.

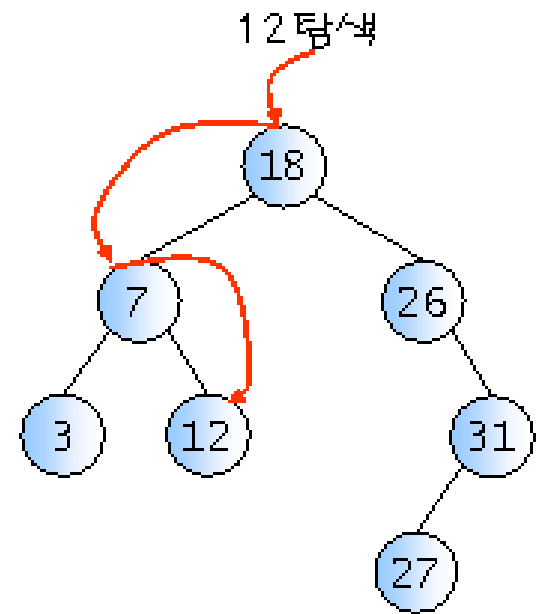


탐색을 구현하는 방법

- 순환적 방법
- 반복적 방법

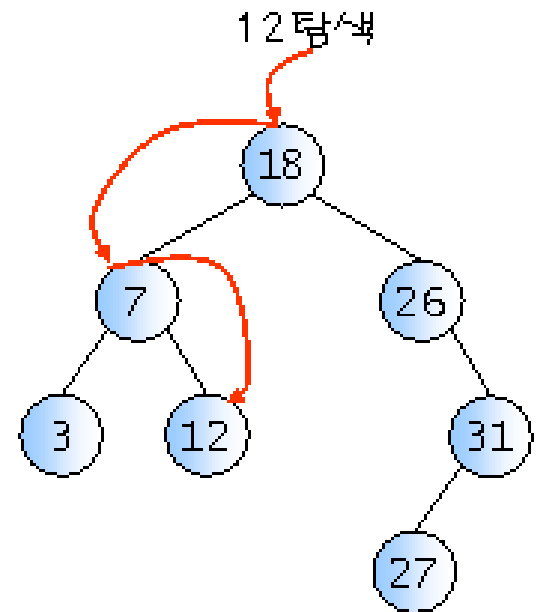
순환적인 방법

```
//순환적인 탐색 함수
TreeNode *search(TreeNode *node, int key)
{
    if ( node == NULL ) return NULL;
    if ( key == node->key ) return node;      (1)
    else if ( key < node->key )
        return search(node->left, key);      (2)
    else
        return search(node->right, key);     (3)
}
```



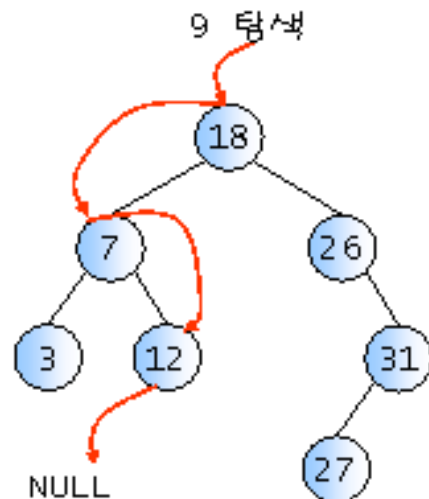
반복적인 방법

```
// 반복적인 탐색 함수
TreeNode *search(TreeNode *node, int key)
{
    while(node != NULL){
        if( key == node->key ) return node;
        else if( key < node->key )
            node = node->left;
        else
            node = node->right;
    }
    return NULL; // 탐색에 실패했을 경우 NULL 반환
}
```

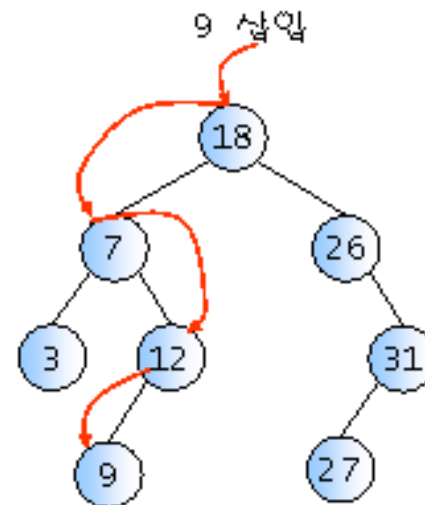


이진 탐색 트리에서의 삽입연산

- 이진 탐색 트리에 원소를 삽입하기 위해서는 먼저 탐색을 수행하는 것이 필요
- 탐색에 실패한 위치가 바로 새로운 노드를 삽입하는 위치



(a) 탐색을 먼저 수행



(b) 탐색이 실패한 위치에 9를 삽입

이진 탐색 트리에서의 삽입연산

```
insert_node(T,z)
```

```
p←NULL;
```

```
t←root;
```

```
while t≠NULL do
```

```
  p←t;
```

```
  if z->key < p->key
```

```
    then t←p->left;
```

```
    else t←p->right;
```

```
if p=NULL
```

```
  then root←z;
```

```
// 트리가 비어있음
```

```
else if z->key < p->key
```

```
  then p->left←z
```

```
  else p->right←z
```


이진 탐색 트리에서의 삽입연산

```
// key를 이진 탐색 트리 root에 삽입한다.
// key가 이미 root안에 있으면 삽입되지 않는다.
void insert_node(TreeNode **root, int key)
{
    TreeNode *p, *t; // p는 부모노드, t는 현재노드
    TreeNode *n;     // n은 새로운 노드
    t = *root;
    p = NULL;
    // 탐색을 먼저 수행
    while (t != NULL){
        if( key == t->key ) return;
        p = t;
        if( key < t->key ) t = t->left;
        else t = t->right;
    }
```

```
// key가 트리 안에 없으므로 삽입 가능
n = (TreeNode *) malloc(sizeof(TreeNode));
if( n == NULL ) return;
n->key = key; // 데이터 복사
n->left = n->right = NULL;
// 부모 노드와 링크 연결
if( p != NULL )
    if( key < p->key )
        p->left = n;
    else p->right = n;
else *root = n;
}
```

이진 탐색 트리에서의 삭제연산

- 3가지의 경우
 1. 삭제하려는 노드가 단말 노드일 경우
 2. 삭제하려는 노드가 하나의 왼쪽이나 오른쪽 서브 트리중 하나만 가지고 있는 경우
 3. 삭제하려는 노드가 두개의 서브 트리 모두 가지고 있는 경우
- CASE 1: 삭제하려는 노드가 단말 노드일 경우: 단말노드의 부모노드를 찾아서 연결을 끊으면 된다.

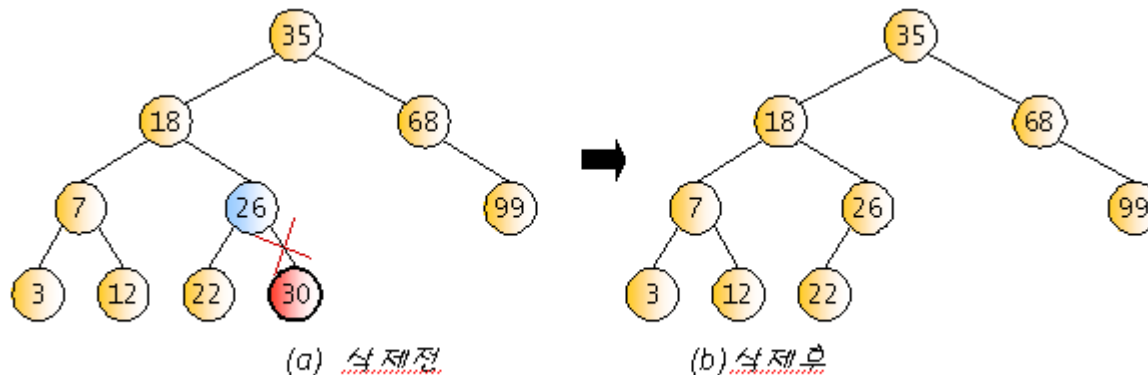


그림 7.42 이진 탐색 트리의 삭제연산: 삭제노드가 단말노드인 경우

이진 탐색 트리에서의 삭제연산

- CASE 2: 삭제하려는 노드가 하나의 서브트리만 갖고 있는 경우 : 삭제되는 노드가 왼쪽이나 오른쪽 서브 트리중 하나만 갖고 있을 때, 그 노드는 삭제하고 서브 트리는 부모 노드에 붙여준다.

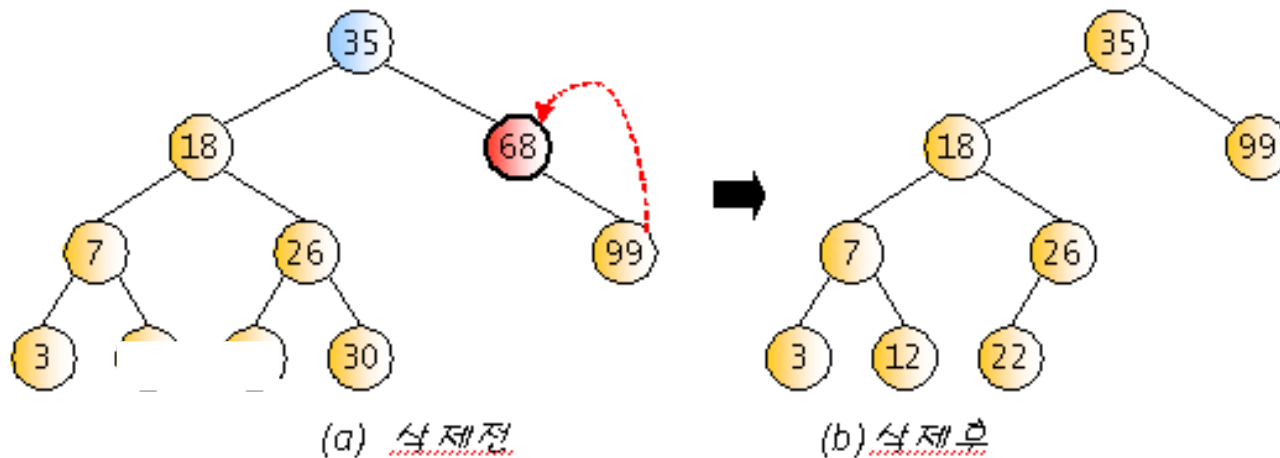
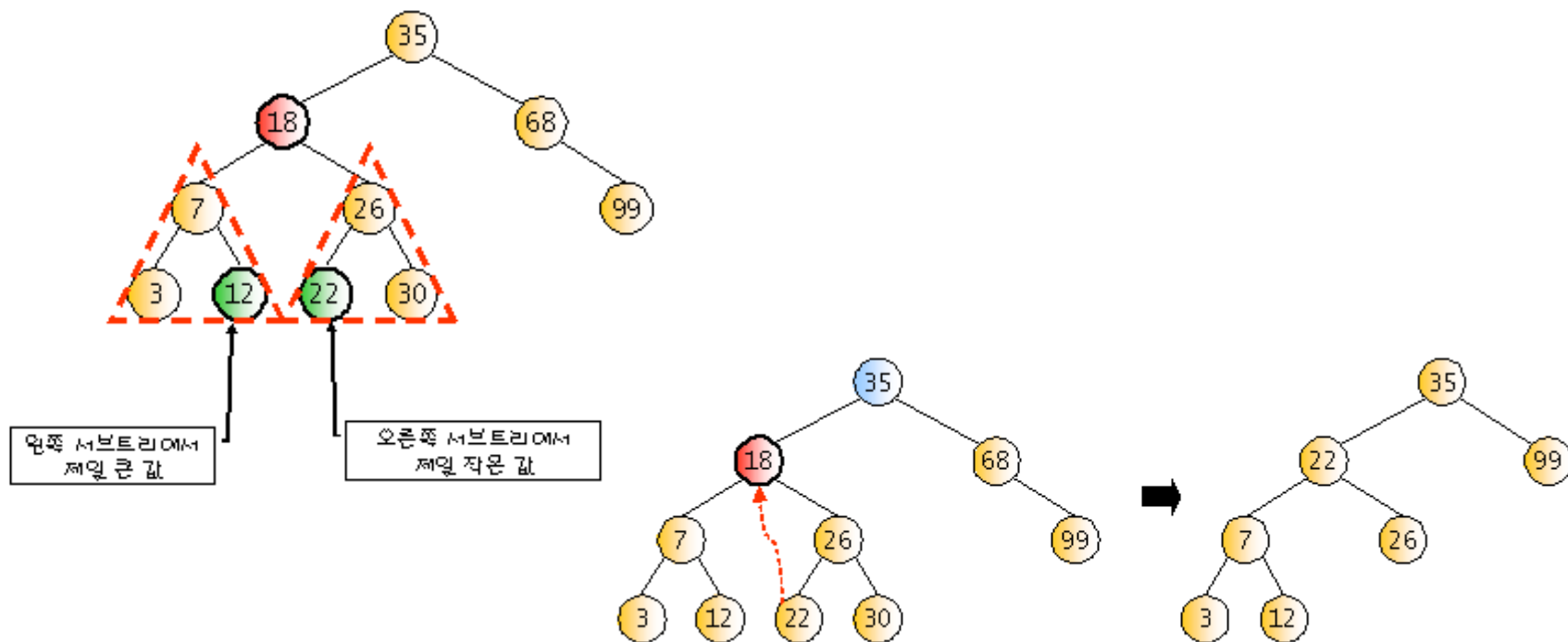


그림 7.43 이진탐색트리의 삭제연산: 삭제노드가 하나의 서브트리를 가지고 있는 경우

이진 탐색 트리에서의 삭제연산

- CASE 3: 삭제하려는 노드가 두개의 서브트리를 갖고 있는 경우: 삭제 노드와 가장 비슷한 값을 가진 노드를 삭제노드 위치로 가져온다.



이진 탐색 트리에서의 삭제연산

```
// 삭제 함수
void delete_node(TreeNode **root, int key)
{
    TreeNode *p, *t, *child, *succ, *succ_p;
    // key를 갖는 노드 t를 탐색, p는 t의 부모노드
    p = NULL;
    t = *root;
    // key를 갖는 노드 t를 탐색한다.
    while( t != NULL && t->key != key ){
        p = t;
        t = ( key < t->key ) ? t->left : t->right;
    }
    // 탐색이 종료된 시점에 t가 NULL이면 트리안에 key가 없음
    if( t == NULL ) { // 탐색트리에 없는 키
        printf("key is not in the tree");
        return;
    }
}
```

이진 탐색 트리에서의 삭제연산

```
// 첫번째 경우: 단말노드인 경우
if( (t->left==NULL) && (t->right==NULL) ){
    if( p != NULL ){
        // 부모노드의 자식필드를 NULL로 만든다.
        if( p->left == t )
            p->left = NULL;
        else p->right = NULL;
    }
    else // 만약 부모노드가 NULL이면 삭제되는 노드가 루트
        *root = NULL;
}
```

```
// 두번째 경우: 하나의 자식만 가지는 경우
else if((t->left==NULL)||(t->right==NULL)){
    child = (t->left != NULL) ? t->left : t->right;
    if( p != NULL ){
        if( p->left == t ) // 부모를 자식과 연결
            p->left = child;
        else p->right = child;
    }
    else //만약 부모노드가 NULL이면 삭제되는 노드가 루트
        *root = child;
}
```

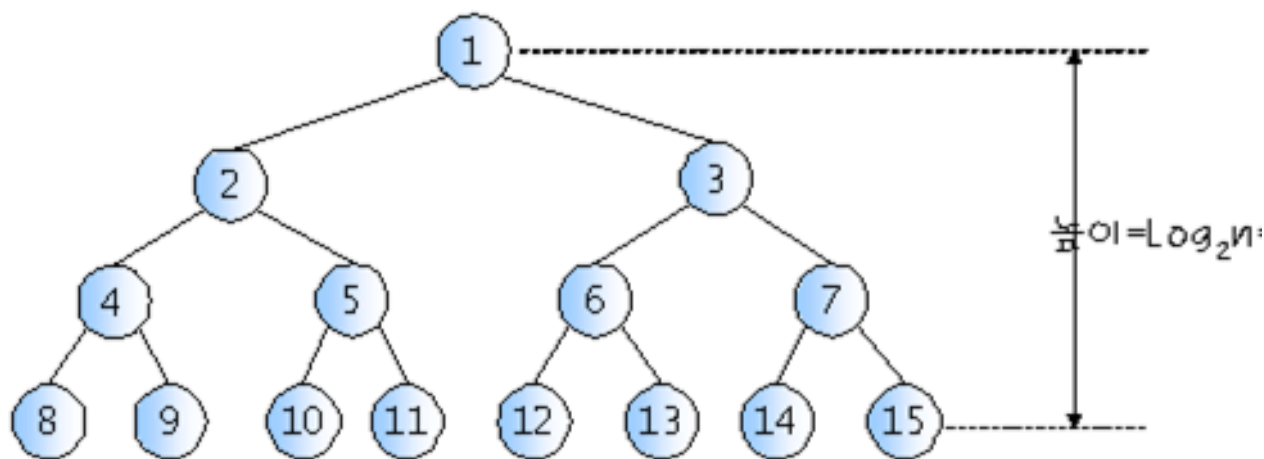
이진 탐색 트리에서의 삭제연산

```
// 세번째 경우: 두개의 자식을 가지는 경우
else{

    // 오른쪽 서브트리에서 후계자를 찾는다.
    succ_p = t;
    succ = t->right;
    // 후계자를 찾아서 계속 왼쪽으로 이동한다.
    while(succ->left != NULL){
        succ_p = succ;
        succ = succ->left;
    }
    // 후속자의 부모와 자식을 연결
    if( succ_p->left == succ )
        succ_p->left = succ->right;
    else
        succ_p->right = succ->right;
    t->key = succ->key; // 후속자가 가진 키값을 현재 노드에 복사
    t = succ; // 원래의 후속자 삭제
}
free(t);
}
```

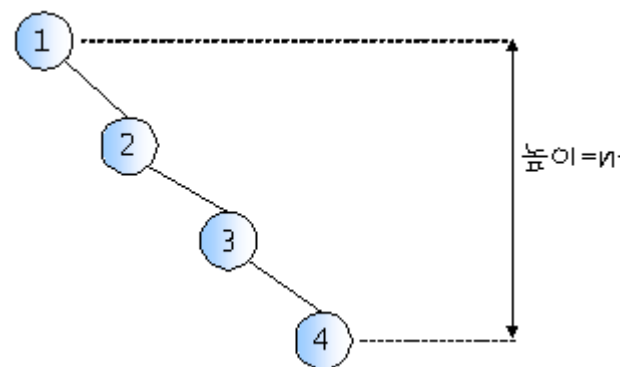
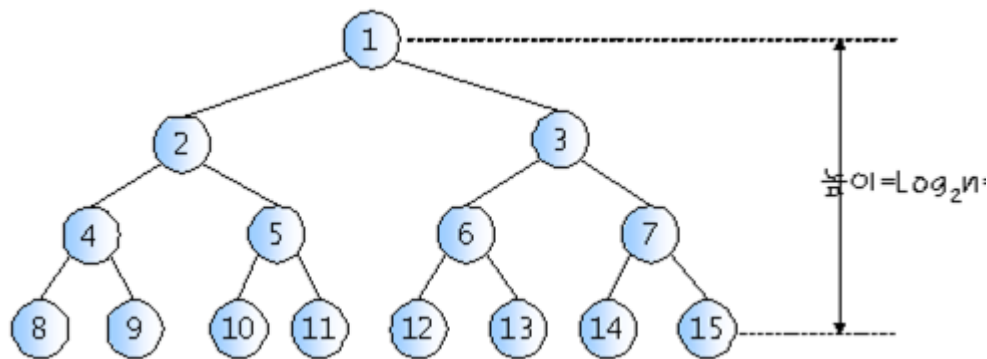
이진 탐색 트리의 성능분석

- 이진 탐색 트리에서의 탐색, 삽입, 삭제 연산의 시간 복잡도는 트리의 높이를 h 라고 했을때 h 에 비례한다



이진 탐색 트리의 성능분석

- 최선의 경우
 - ▶ 이진 트리가 균형적으로 생성되어 있는 경우
 - ▶ $h = \log_2 n$
- 최악의 경우
 - ▶ 한쪽으로 치우친 경사이진트리의 경우
 - ▶ $h = n$
 - ▶ 순차탐색과 시간복잡도가 같다.



이진 탐색 트리 응용: 영어사전구현

- 이진 탐색 트리를 이용하여 다음과 같은 메뉴의 기능을 제공하는 영어 사전을 구현해 보자.
- 이진 탐색 트리에 대한 연산
 - ▶ 입력
 - ▶ 삭제
 - ▶ 탐색
 - ▶ 사전 전체 출력(중위순회)
- 레코드 구조

```
typedef struct {  
    char word[MAX_WORD_SIZE]; // 키필드  
    char meaning[MAX_MEANING_SIZE];  
} element;  
// 노드의 구조  
typedef struct TreeNode {  
    element key;  
    struct TreeNode *left, *right;  
} TreeNode;
```

이진 탐색 트리 응용: 영어사전구현

● 실행 결과

i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료

i
단어:suspect
의미:~일 것 같다고 생각한다.

i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료

i
단어:ascertain
의미:확인하다

i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료

i
단어:exception
의미:예외

i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료

p
((ascertain(exception))suspect)

i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료

d
단어:exception

i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료

p
((ascertain)suspect)

i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료

s
단어:suspect
의미:~일 것 같다고 생각한다.

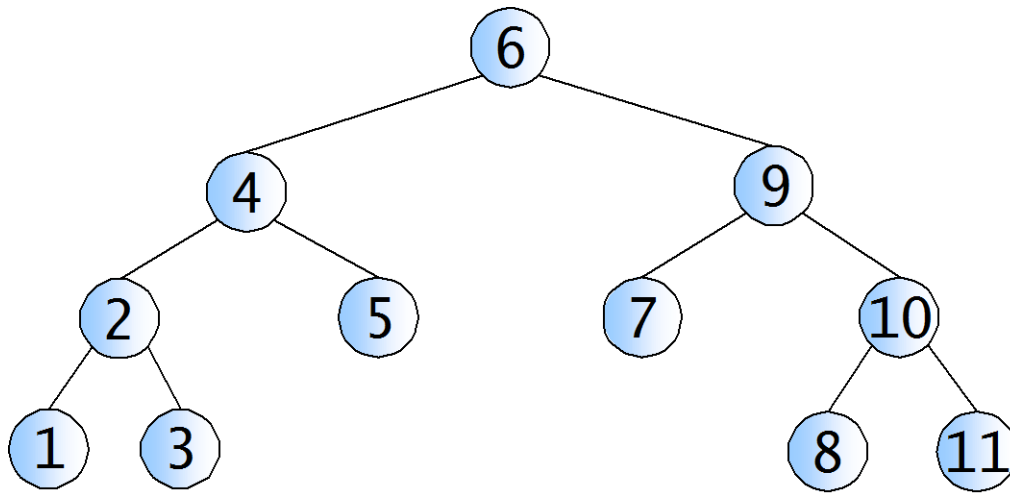
i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료

s
단어:exception

i: 입력
d: 삭제
s: 탐색
p: 출력
q: 종료

연습문제 1

- 다음의 이진 트리에 대해 다음 질문에 답하십시오.
 - ▶ 전위, 후위, 중위, 레벨 순회한 결과는?



연습문제 2

- 다음의 전위 순회와 중위 순회 결과를 생성할 수 있는 이진 트리를 그려보시오.
 - ▶ 전위 순회: A, B, D, E, C, F, G, H
 - ▶ 중위 순회: E, D, B, A, G, F, H, C

연습문제 3

- 다음 순서로 자료가 입력되었다고 가정하여 이진 탐색 트리를 생성하시오.

5,3,7,2,4,6,9,1,8,10