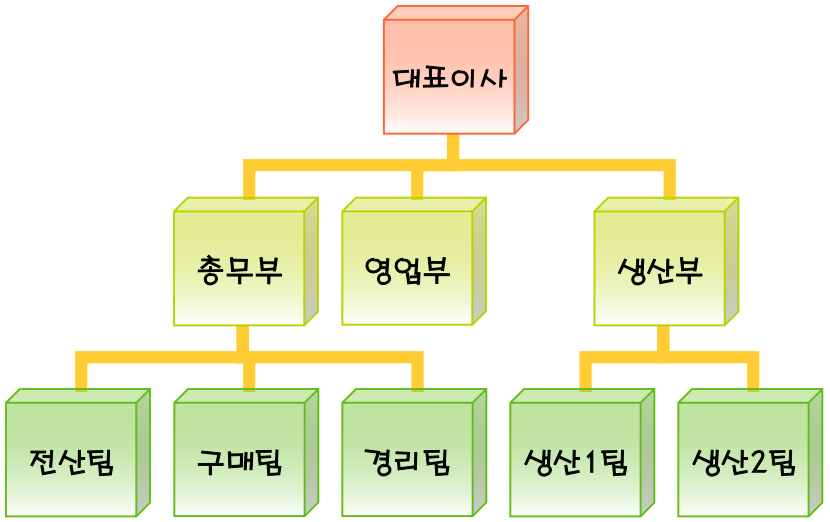
7장: 트리

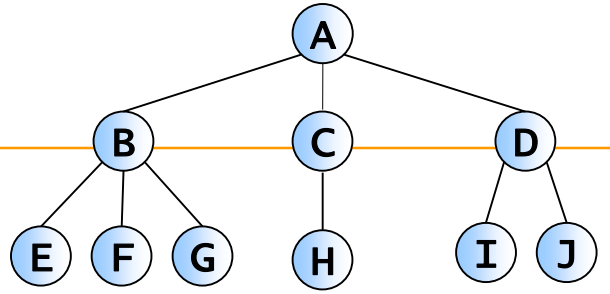
- 목차
  - 1. 트리의 개념
  - 2. 이진 트리 소개
  - 3. 이진 트리 표현
  - 4. 이진 트리 순회
  - 5. 이진 트리 연산
  - 6. 스레드 이진 트리
  - 7. 이진 탐색 트리
  - 8. 이진 탐색 트리의 응용

1. 트리의 개념

- 트리: 계층적인 구조를 나타내는 자료구조
- 트리는 부모-자식 관계의 노드들로 이루어진다.
- 응용분야:
  - 계층적인 조직 표현
  - 파일 시스템
  - 인공지능에서의 결정트리



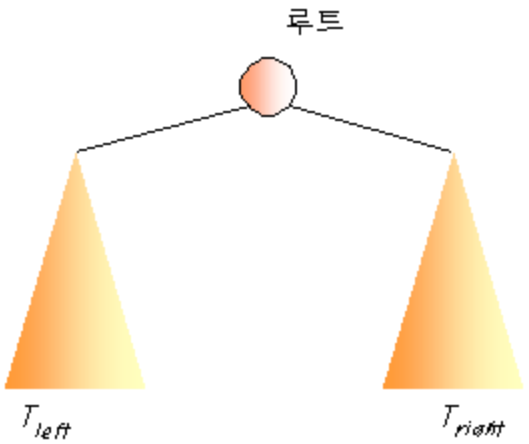
트리의 용어



- 노드(node): 트리의 구성요소
- 루트(root): 부모가 없는 노드(A)
- 서브트리(subtree): 하나의 노드와 그 노드들의 자손들로 이루어진 트리
- 단말노드(terminal node): 자식이 없는 노드(E,F,G,H,I,J)
- 비단말노드: 적어도 하나의 자식을 가지는 노드 (A,B,C,D)
- 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

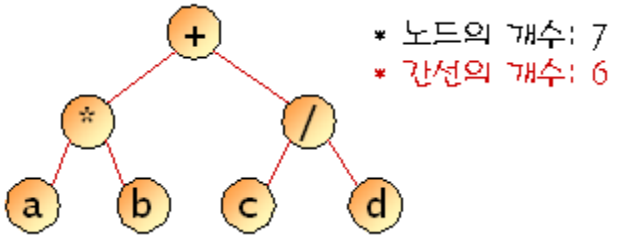
2. 이진 트리 소개

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

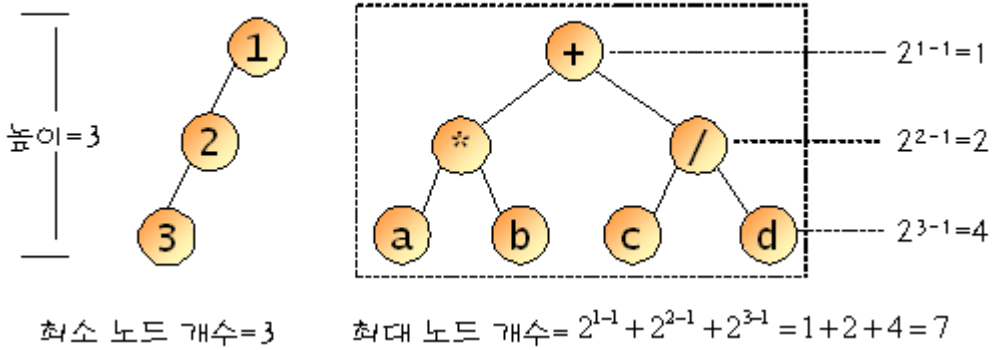


이진트리의 성질

- 노드의 개수가 n개이면 간선의 개수는 n-1

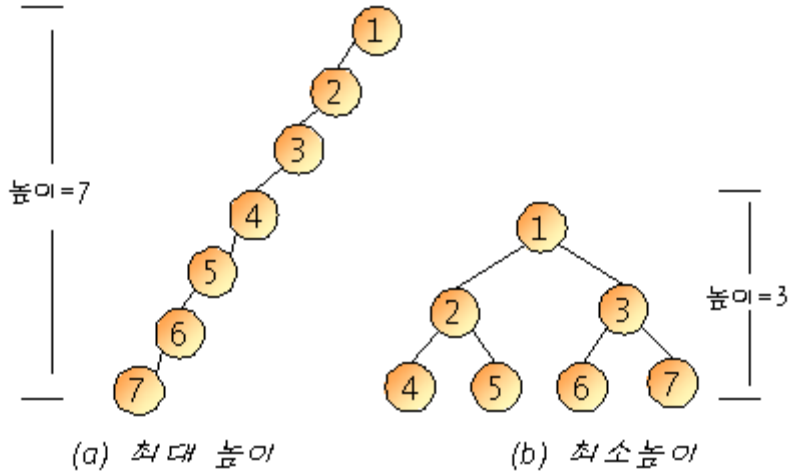


- 높이가 h인 이진트리의 경우, 최소 h개의 노드를 가지며 최대 2^h-1개의 노드를 가진다.



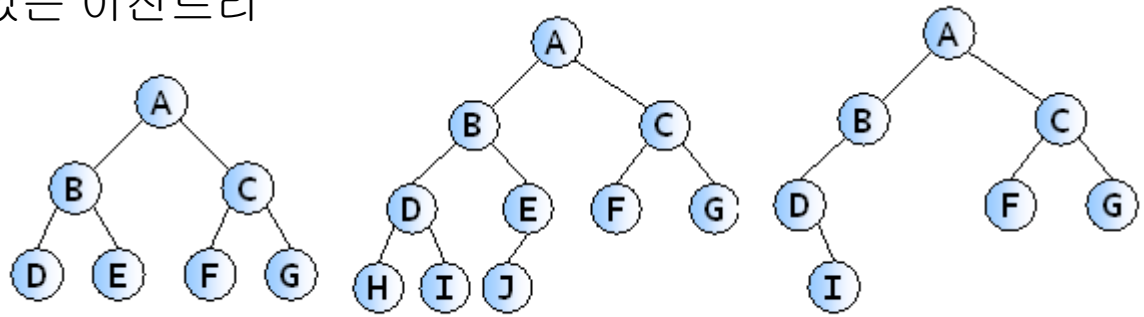
이진트리의 성질

- n개의 노드를 가지는 이진트리의 높이는 최대 n이거나 최소 $\lceil \log_2(n+1) \rceil$



이진트리의 분류

- 포화 이진 트리(full binary tree): 트리의 각 레벨에 노드가 꽉 차있는 이진트리
- 완전 이진 트리(complete binary tree): 높이가 h일 때 레벨 1부터 h까지는 노드가 모두 채워져 있고 마지막 레벨 h에서는 왼쪽부터 오른쪽으로 노드가 순서대로 채워져 있는 이진트리



(a) 포화이진트리 (b) 완전이진트리 (c) 기타이진트리

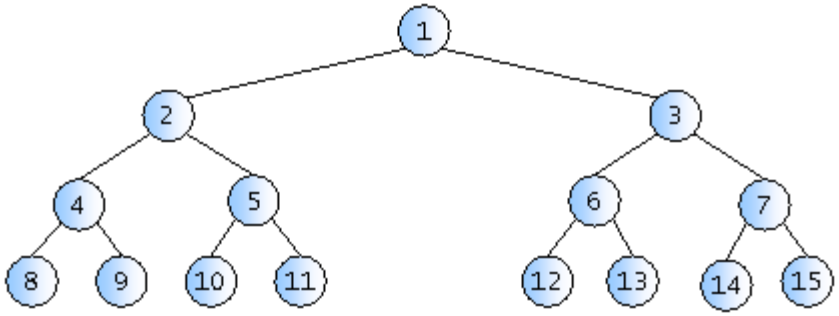
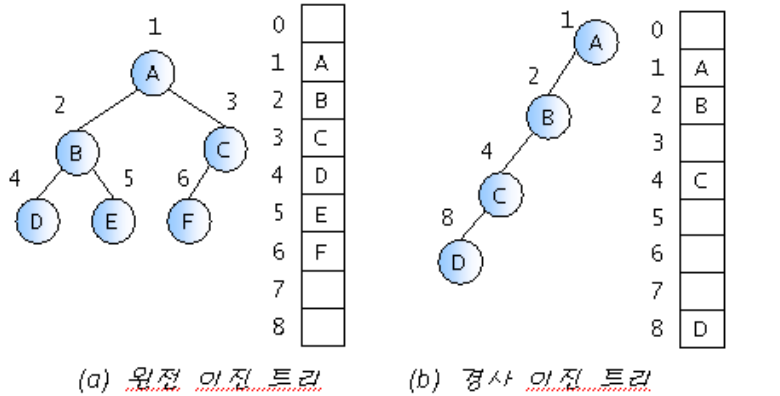
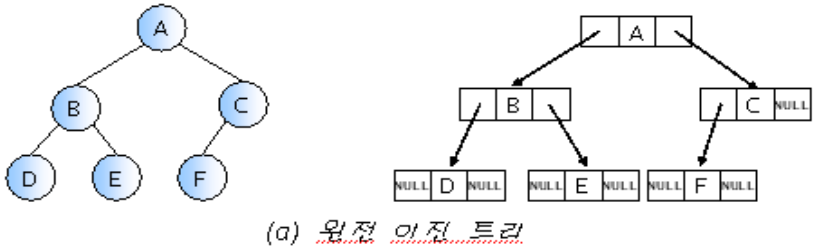


그림 7.15 포화이진트리에서의 노드의 번호

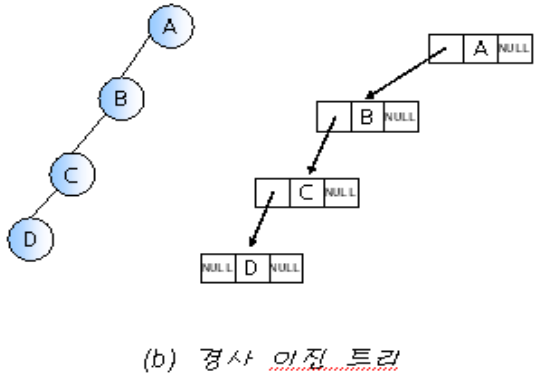
3. 이진 트리 표현

- 배열표현법:
  모든 이진트리를
  포화이진트리라고 가정하고
  각 노드에 번호를 붙여서 그
  번호를 배열의 인덱스로
  삼아 노드의 데이터를
  배열에 저장하는 방법



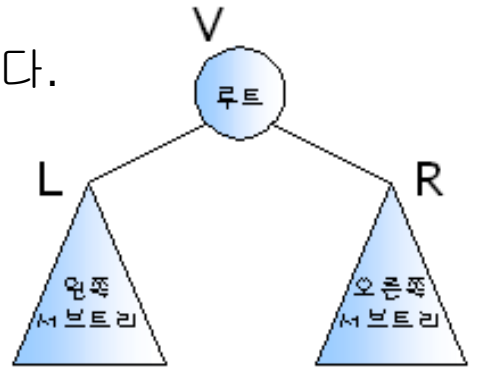


- 링크표현법:
  포인터를 이용하여
  부모노드가 자식노드를
  가리키게 하는 방법



4. 이진 트리 순회

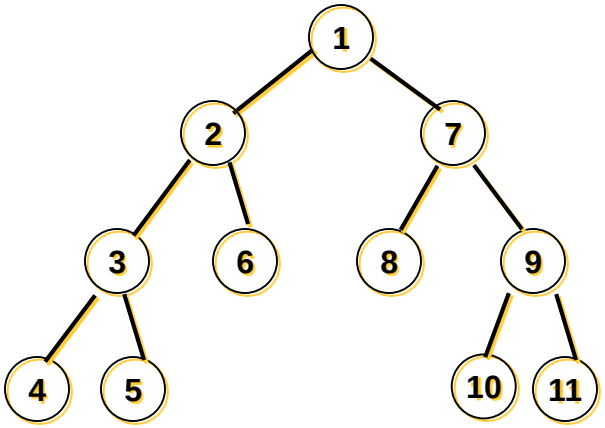
- 순회(traversal): 트리의 노드들을 체계적으로 방문하는 것
- 3가지의 기본적인 순회방법
  - 전위순회(preorder traversal) : VLR
    - 자손노드보다 루트노드를 먼저 방문한다.
  - 중위순회(inorder traversal) : LVR
    - 왼쪽 자손, 루트, 오른쪽 자손 순으로 방문한다.
  - 후위순회(postorder traversal) : LRV
    - 루트노드보다 자손을 먼저 방문한다.



전위순회

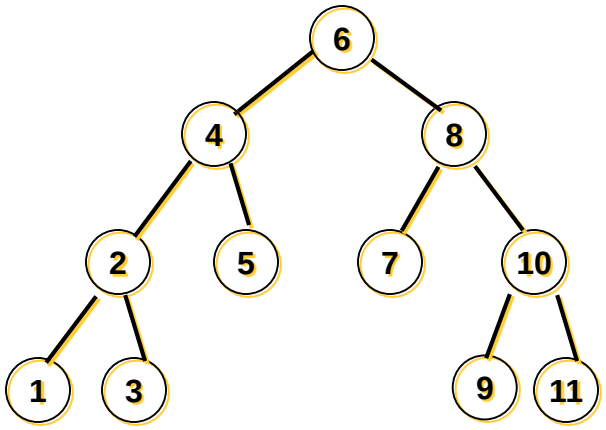
- 루트를 먼저 방문하는 순회방법
- 루트 -> 왼쪽 서브트리 -> 오른쪽 서브트리

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



중위순회

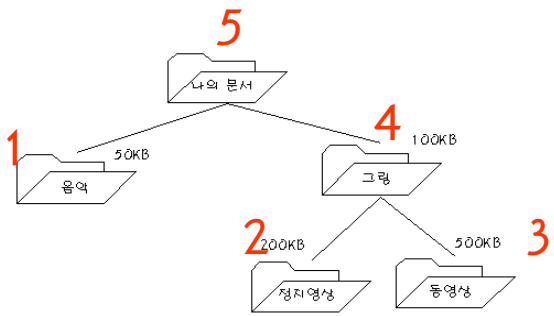
- 왼쪽 서브트리->루트->오른쪽 서브트리 순으로 방문



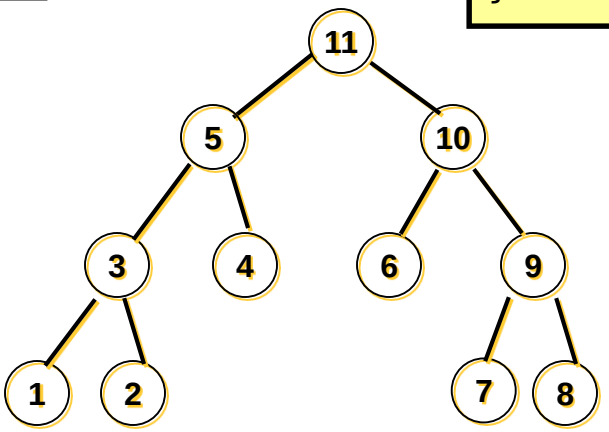

```
// 중위 순회
inorder( TreeNode *root ){
    if ( root ){
      inorder( root->left );// 왼쪽서브트리 순회
      printf("%d", root->data );  // 노드 방문
      inorder( root->right );// 오른쪽서브트리 순회
    }
}
```

후위순회

- 왼쪽 서브트리->오른쪽 서브트리 -> 루트 순으로 방문
- (예) 디렉토리 용량 계산

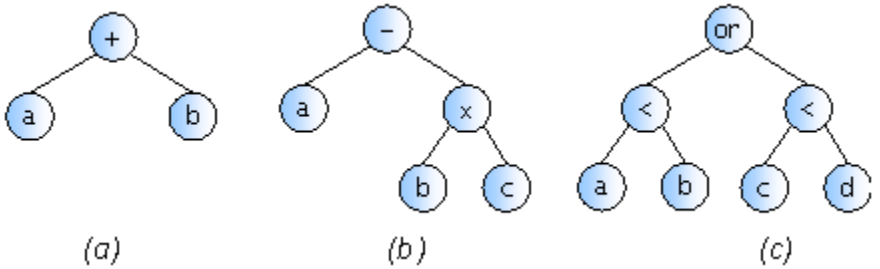



```
// 후위 순회
postorder( TreeNode *root ){
    if ( root ){
          postorder( root->left );// 왼쪽서브트리 순회
          postorder( root->right );// 오른쪽서브트리순회
          printf("%d", root->data );  // 노드 방문
    }
}
```



수식 트리

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



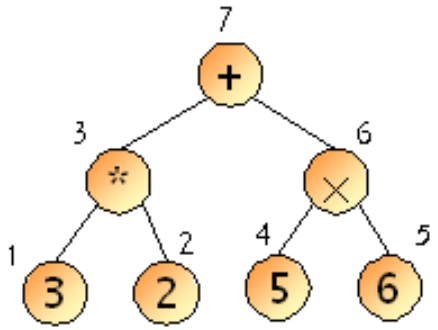
수식	a + b	a − (b × c)	(a < b) or (c < d)
전위순회	+ a b	− a × b c	or < a b < c d
중위순회	a + b	a − b × c	a < b or c < d
후위순회	a b +	a b c × −	a b < c d < or

수식트리계산

- 후위순회를 사용
  - 서브트리의 값을 순환호출로 계산
  - 비단말노드를 방문할때 양쪽서브트리의 값을 저장된 연산자를 이용하여 계산한다

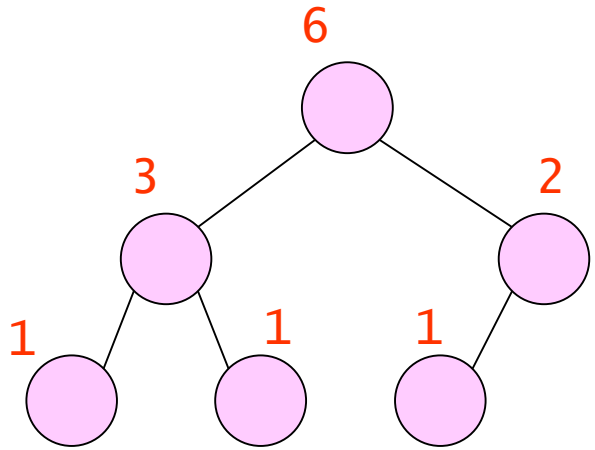
```
evaluate(exp)

if exp = NULL
  then return 0;
  else x←evaluate(exp->left);
       y←evaluate(exp->right);
       op←exp->data;
       return (x op y);
```



5. 이진 트리 연산

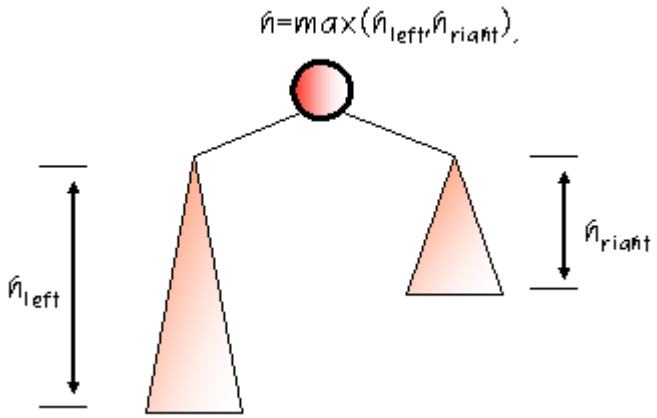
- 노드 개수
  - 탐색 트리안의 노드의 개수를 계산
  - 각각의 서브트리에 대하여 순환 호출한 다음, 반환되는 값에 1을 더하여 반환




```
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;
}
```

이진트리연산

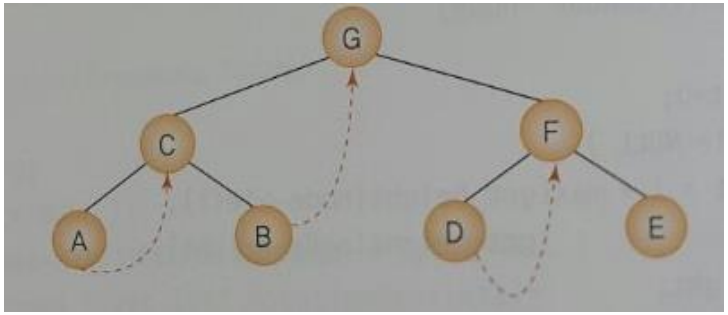
- 높이
  - 서브트리에 대하여 순환호출하고 서브 트리들의 반환값 중에서 최대값을 구하여 반환




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

6. 스레드 이진 트리

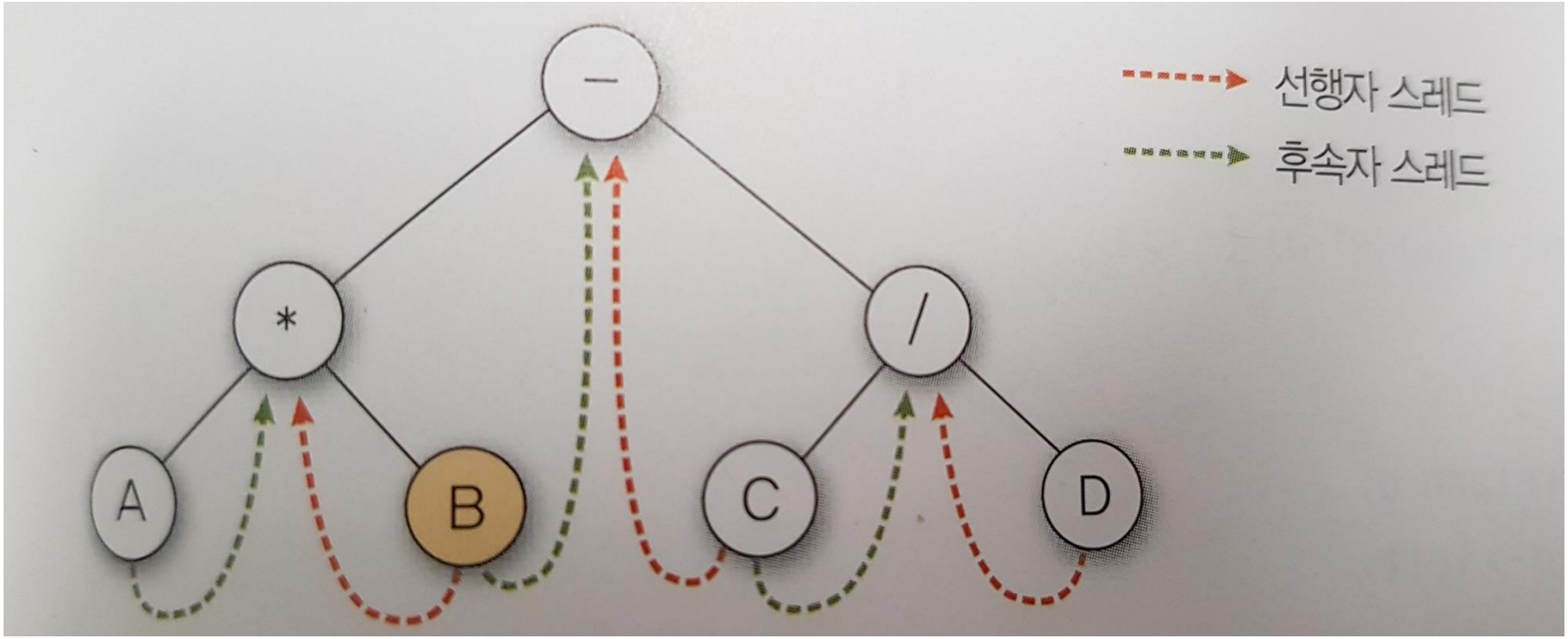
- 이진트리의 NULL 링크를 이용하여 순환호출 없이도 트리의 노드들을 순회
- 트리의 노드수: n, 노드당 2개의 링크: 2n, 이중에서 n-1은 NULL링크가 아니지만 n+1개의 링크는 NULL
- NULL 링크에 중위 순회시에 후속 노드인 중위 후속자(inorder successor)를 저장시켜 놓은 트리가 스레드 이진 트리(threaded binary tree)
- 단말노드와 비단말노드의 구별을 위하여 is_thread 필드 필요
- 문제점: 스레드를 설정하기 위해 삽입이나 삭제시 더 많은 일을 요구




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

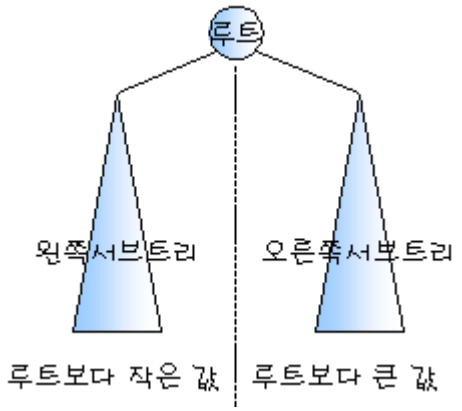
스레드 이진 트리

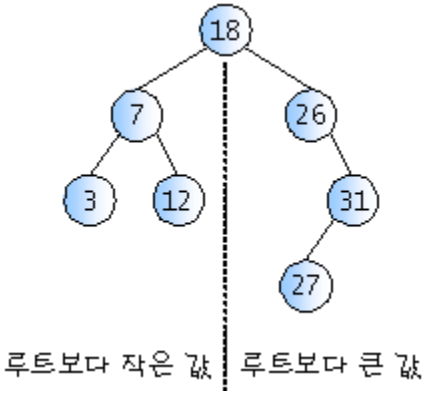
- 선행자(Predecessor): 현재 노드 직전에 처리한 노드
- 후행자(Successor): 현재 노드 직후에 처리할 노드



7. 이진 탐색 트리

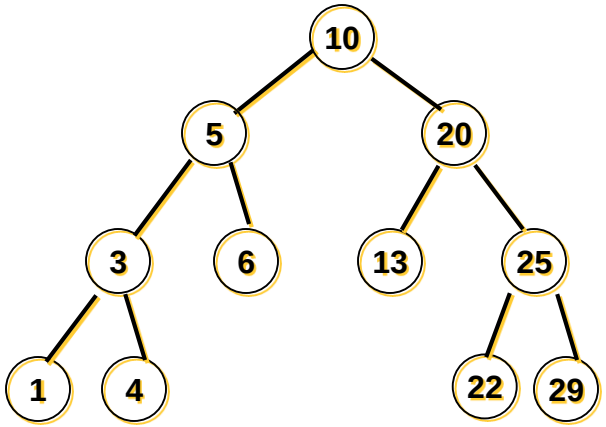
- 탐색작업을 효율적으로 하기 위한 자료구조
- key(왼쪽서브트리)<key(루트노드)<key(오른쪽서브트리)
- 이진탐색를 중위순회하면 오름차순으로 정렬된 값을 얻을 수 있다.





이진 탐색 트리의 예

- 아래처럼 이진 탐색 트리가 구성되면,



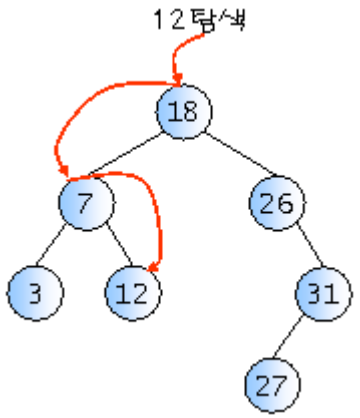
- 중위 순회로 이진 탐색 트리를 순회하면,
- 1 – 3 – 4 – 5 – 6 – 10 – 13 – 20 – 22 – 25 – 29 의 결과를 얻을수 있다

이진 탐색 트리에서의 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 키값보다 크면 탐색은 이 루트 노드의 오른쪽 자식을 기준으로 다시 시작한다.

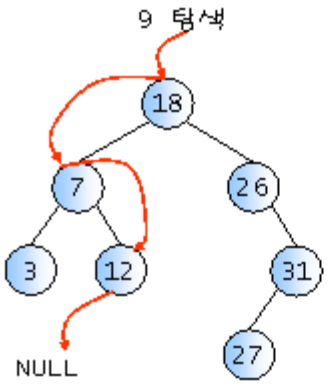
```
search(x, k)

if x=NULL
            then return NULL;
if k=x->key
            then return x;
else if k<x->key
     then return search(x->left, k);
     else return search(x->right, k);
```

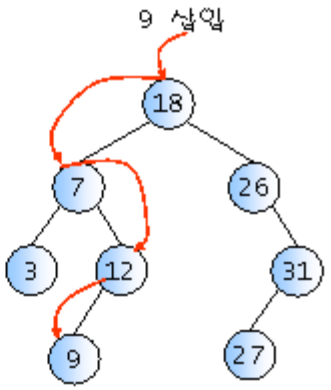


이진 탐색 트리에서의 삽입연산

- 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

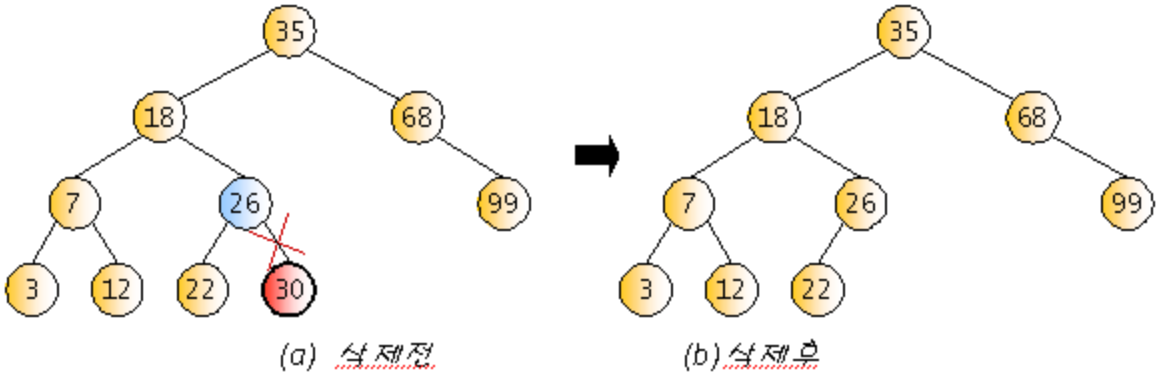
```
insert_node(T,z)

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 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.

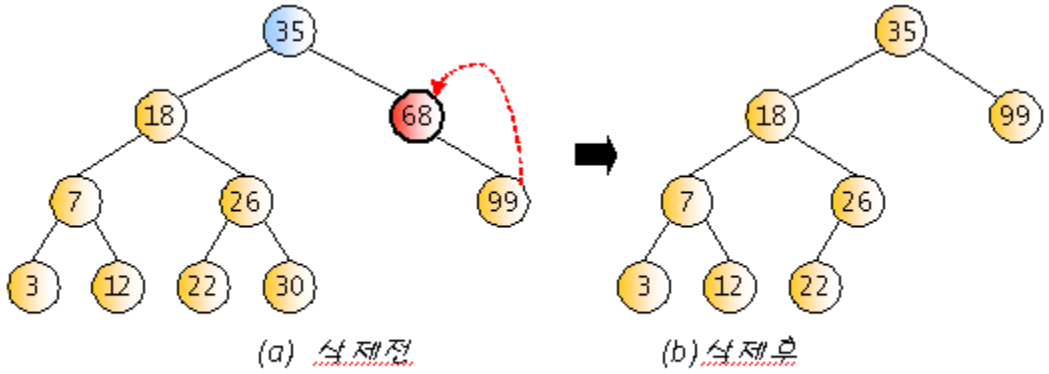


(a) 삭제전 (b) 삭제후

그림 7.42 이진탐색트리의 삭제연산: 삭제노드가 단말노드인 경우

이진 탐색 트리에서의 삭제연산

- CASE 2:삭제하려는 노드가 하나의 서브트리만 가지고 있는 경우 :삭제되는 노드가 왼쪽이나 오른쪽 서브 트리중 하나만 가지고 있는 경우에는 노드는 삭제하고 서브 트리는 부모 노드에 붙여준다.

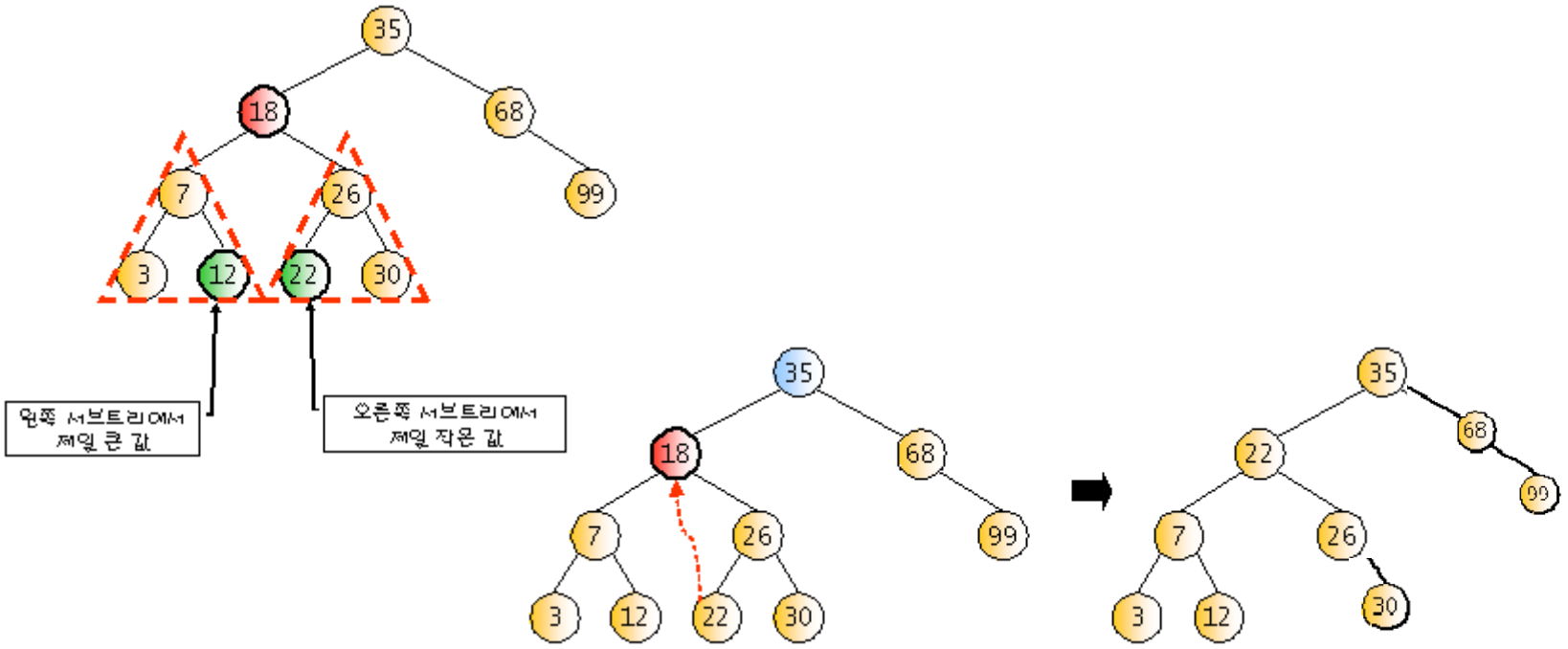


(a) 삭제전 (b)삭제후

그림 7.43 이진탐색트리의 삭제연산: 삭제노드가 하나의 서브트리를 가지고 있는 경우

이진 탐색 트리에서의 삭제연산

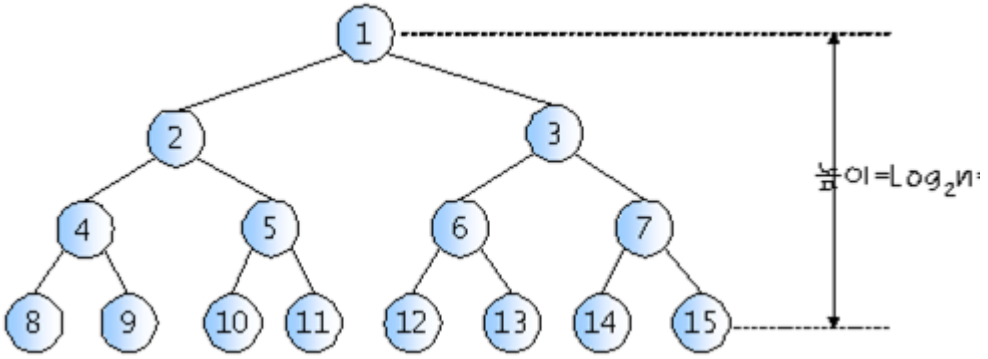
- CASE 3:삭제하려는 노드가 두개의 서브트리를 가지고 있는 경우: 삭제노드와 가장 비슷한 값을 가진 노드를 삭제노드 위치로 가져온다.



이진 탐색 트리의 성능

- 이진 탐색 트리에서의 탐색, 삽입, 삭제 연산의 시간 복잡도는 트리의 높이를 h라고 했을때 h에 비례한다

- 최선의 경우
  - 이진 트리가 균형적으로 생성되어 있는 경우
  - $h=\log_2 n$



- 최악의 경우
  - 한쪽으로 치우친 경사이진트리의 경우
  - $h=n$
  - 순차탐색과 시간복잡도가 같다.