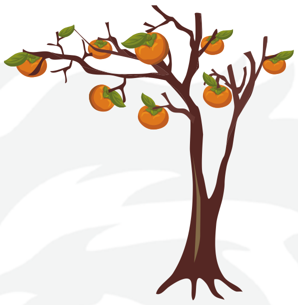
CSE2010 자료구조론

Week 7: Tree

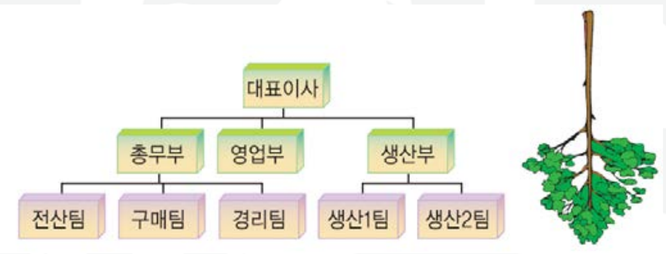
ICT융합학부 조용우

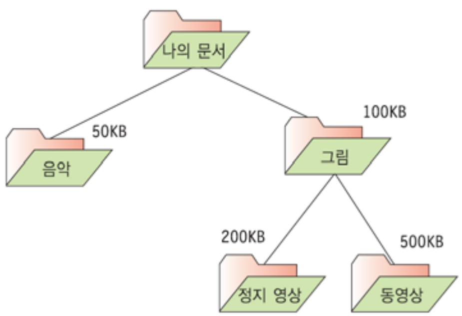
트리(tree)란?

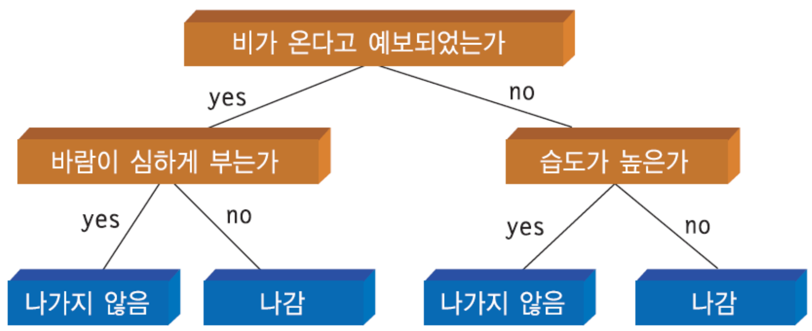
- 트리: 계층적인 구조를 나타내는 자료구조
  - cf) 리스트, 스택, 큐 등은 선형 구조
  - 부모-자식 관계의 노드들로 이루어짐



- 응용분야
  - 계층적인 조직 표현
  - 컴퓨터 디스크의 디렉토리 구조
  - 인공지능에서의 결정 트리(decision tree)



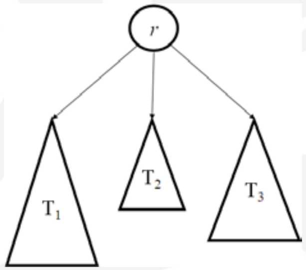




트리 정의

- 트리
  - 자료 사이의 계층적 관계를 구조화
  - 예) 가계도: 자신, 부모, 형제, 조부모, 선조, 후손

- 트리는 하나 이상의 노드로 구성된 유한 집합으로서,
  - 특별히 지정된 노드인 루트(root)가 있음
  - 루트를 제외한 나머지 노드들은 교차하지 않는 분리 집합 T_(1,) T_2,…,T_m (m≧1)으로 분할되며, T_(1,) T_2,…,T_m은 루트의 부분 트리(subtree)로서 각각이 트리임



트리 용어(1)

- 노드(node): 저장된 정보와 부분트리를 가리키는 링크들로 구성
- 차수(degree)
  - 노드의 차수(degree of a node) : 해당 노드의 부분트리의 개수
- 리프(leaf): 단말노드, 즉 차수가 0인 노드
- 내부노드(internal node): 비단말 노드
- 부모 노드(parent), 자식 노드(child), 조부모 노드(grandparent), 손자 노드(grandson)
- 형제 노드(brother or sibling): 부모가 같은 자식 노드들

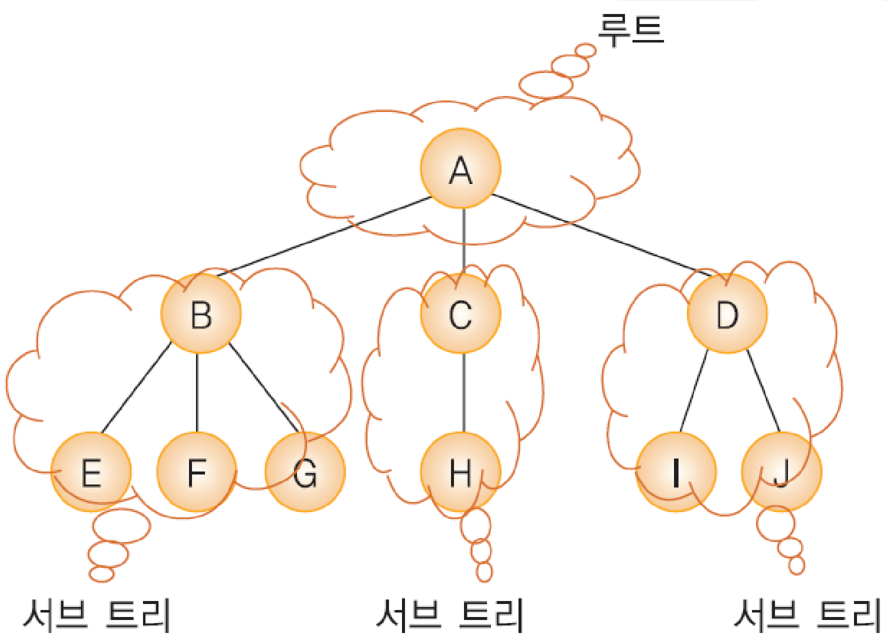
트리 용어(2)

- 선조(ancestor): 해당 노드부터 루트까지의 모든 노드들
- 후손(descendants): 해당 노드를 루트로 하는 부분트리의 모든 노드들
- 고유 선조(proper ancestor) : 해당 노드를 제외한 선조
- 고유 후손(proper descendants) : 해당 노드를 제외한 후손

- 레벨(level) : 루트의 레벨은 0, 나머지는 부모 노드 레벨 + 1
  - 또는, 루트의 레벨을 1로 정의하기도 함
- 트리의 높이(height), 깊이(depth): 해당 트리의 최대 레벨 값

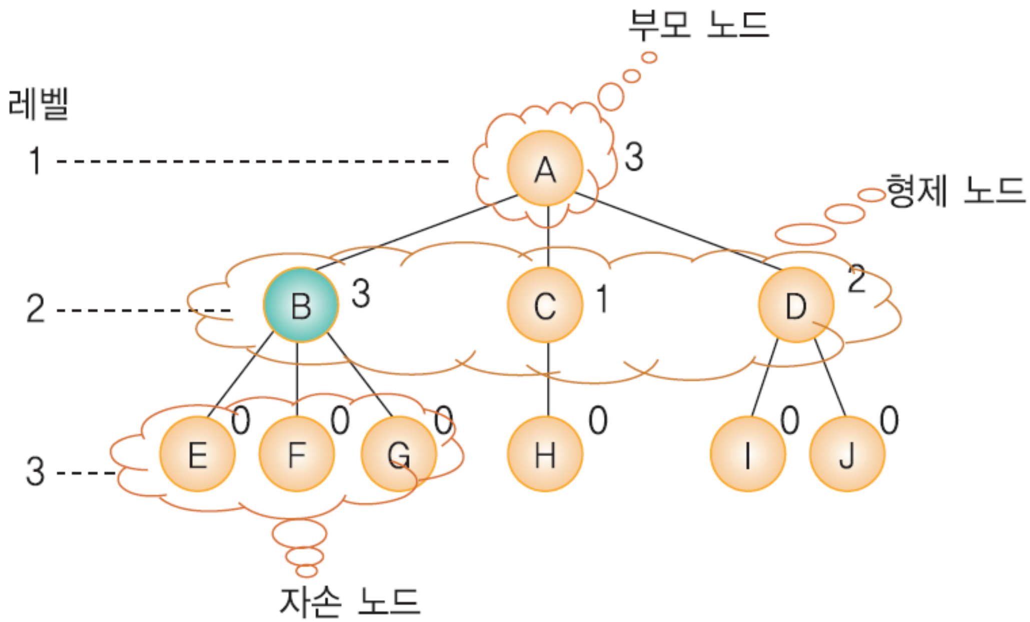
트리 용어 예(1)

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



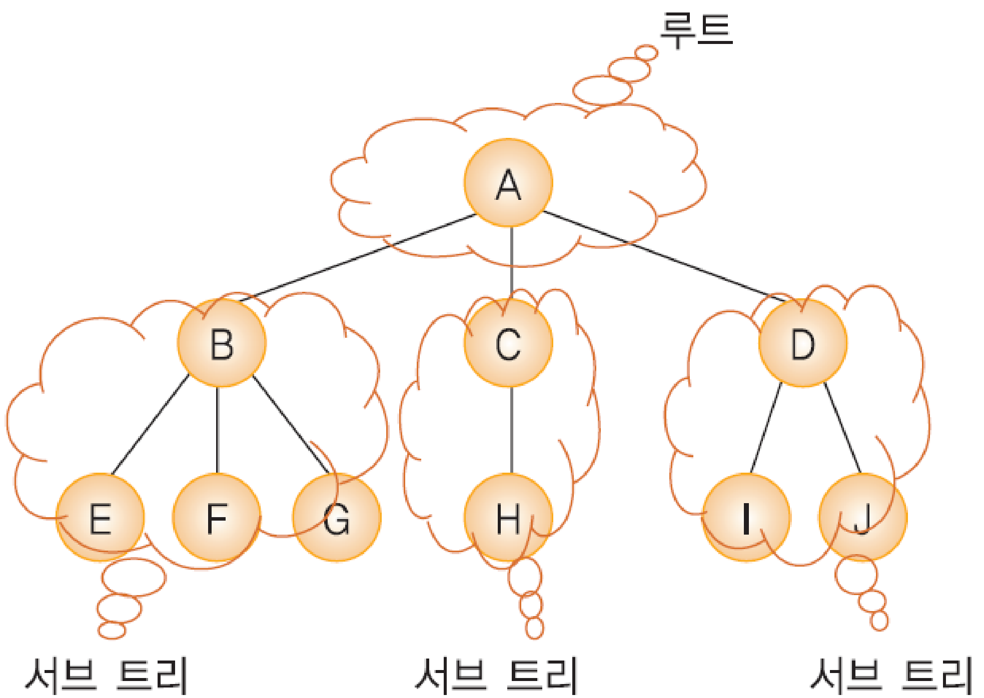
트리 용어 예(2)

- 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



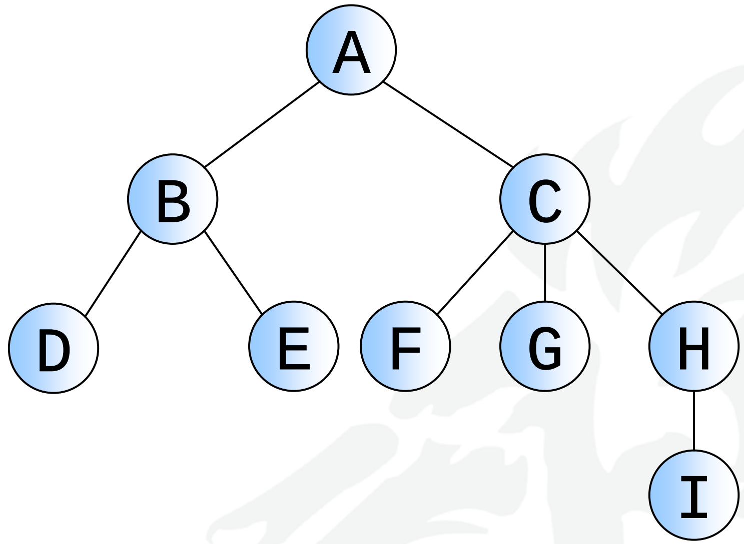
트리 용어 예(3)

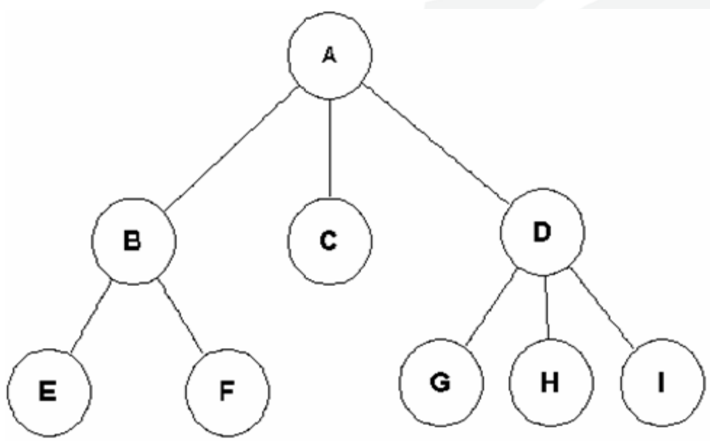
- 단말노드(terminal node): 자식이 없는 노드(A,B,C,D) = leaf
- 비단말노드: 적어도 하나의 자식을 가지는 노드(E,F,G,H,I,J)



트리 용어 연습

- A는 루트 노드
- B는 D와 E의 부모노드
- C는 B의 형제 노드
- D와 E는 B의 자식노드
- B의 차수는 2
- 트리의 높이는 4



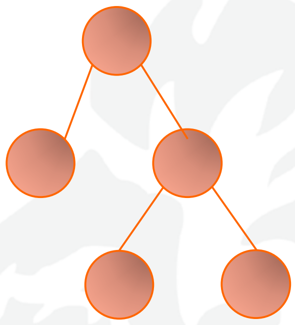


root	A
leaves	E F C G H I
height	2
level of root	0
level of node with F	2
nodes at level 1	3
parent of G, H and I	D
descendants of B	E F

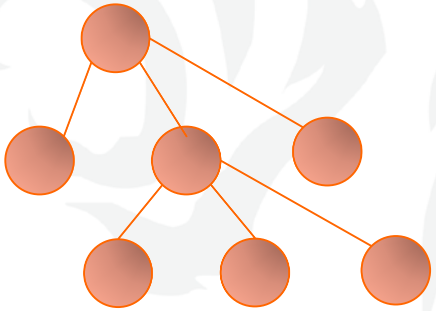
트리의 종류

트리

- 이진 트리

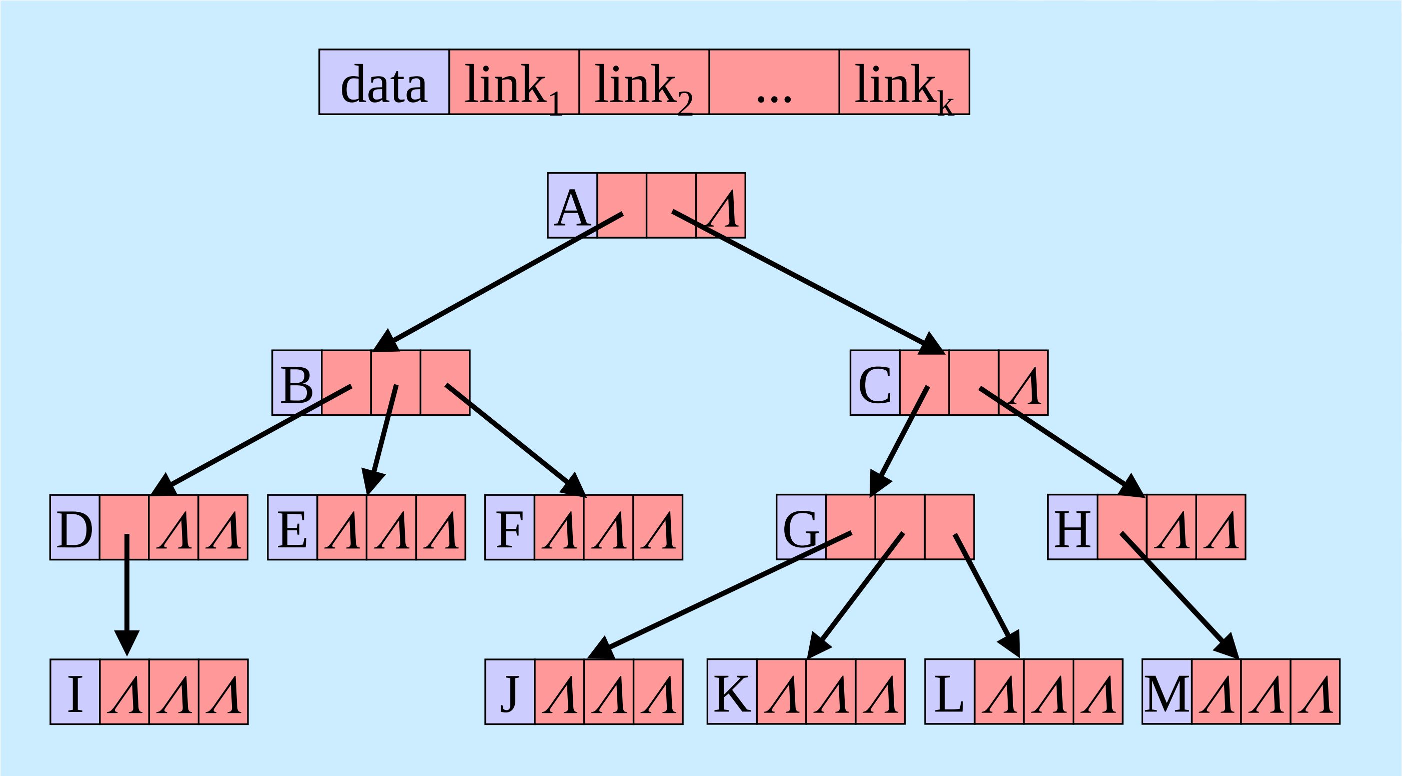


- 일반 트리



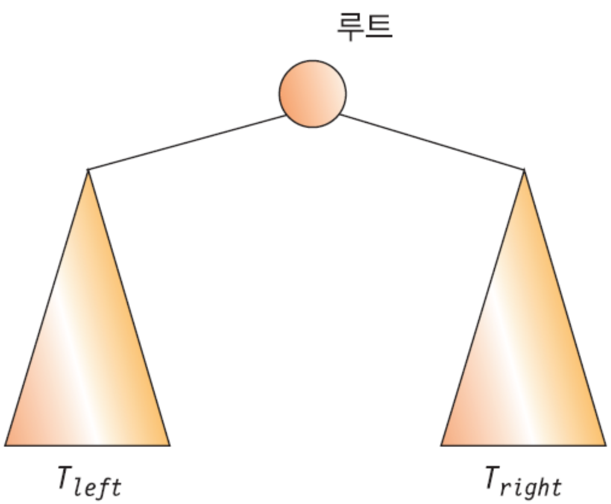
일반 트리의 표현

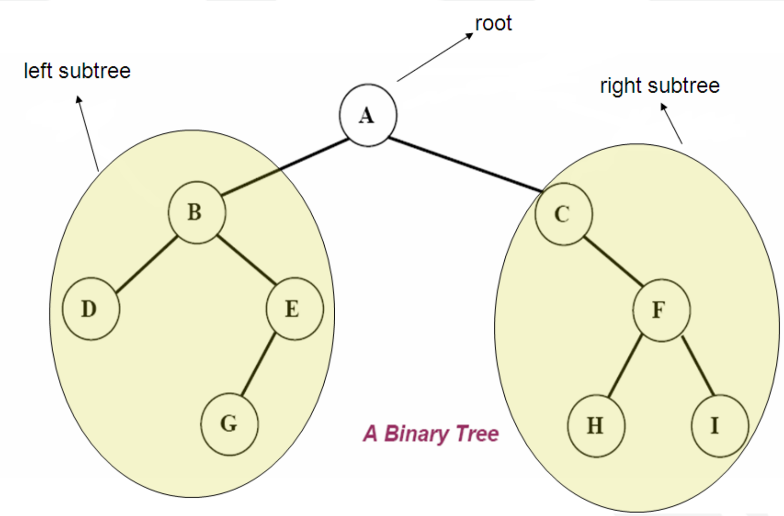
- 괄호: (A(B(D(I),E,F),C(G(J,K,L),H(M))))
- 컴퓨터상의 표현: 데이터와 최대 차수만큼의 링크 필드



이진 트리(Binary Tree)

- 이진 트리(binary tree)
  - 모든 노드가 2개의 서브트리를 가지고 있는 트리
  - 노드의 유한 집합으로 (1) 공집합이거나 (2) 루트와 왼쪽 부분트리 및 오른쪽 부분트리로 불리는 두개의 서로 분리된 이진 트리로 구성됨
- 이진 트리의 노드에는 최대 2개까지의 자식 노드가 존재
  - 모든 노드의 차수가 2 이하가 됨 -> 구현하기가 편리함

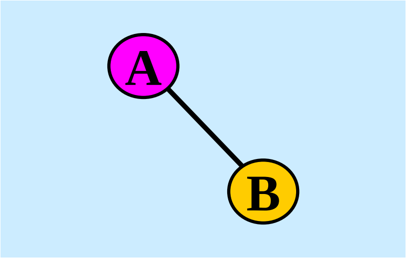




이진 트리의 서브 트리

- 이진 트리의 서브 트리는,
  - (1) 공집합이거나
  - (2) 루트와 왼쪽 서브 트리, 오른쪽 서브 트리로 구성된 노드들의 유한 집합으로 정의됨
- 이진 트리에는 서브 트리간의 순서가 존재
  - 예

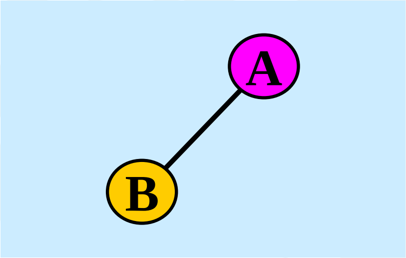
왼쪽 부분트리가 공집합



≠

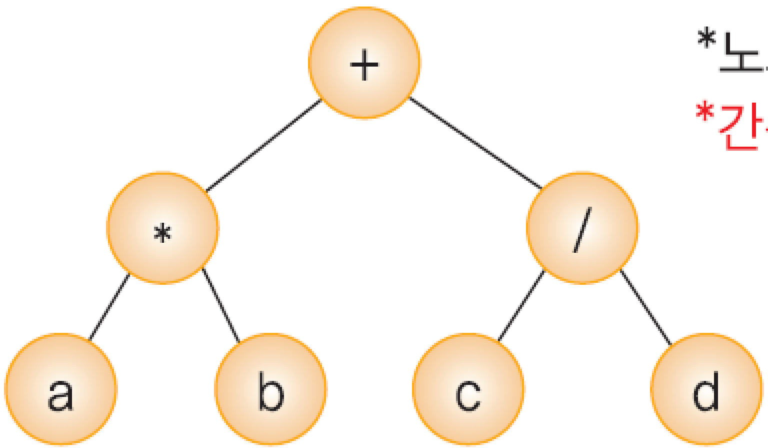
서로 다른 이진 트리

오른쪽 부분 트리가 공집합



이진 트리의 성질(1)

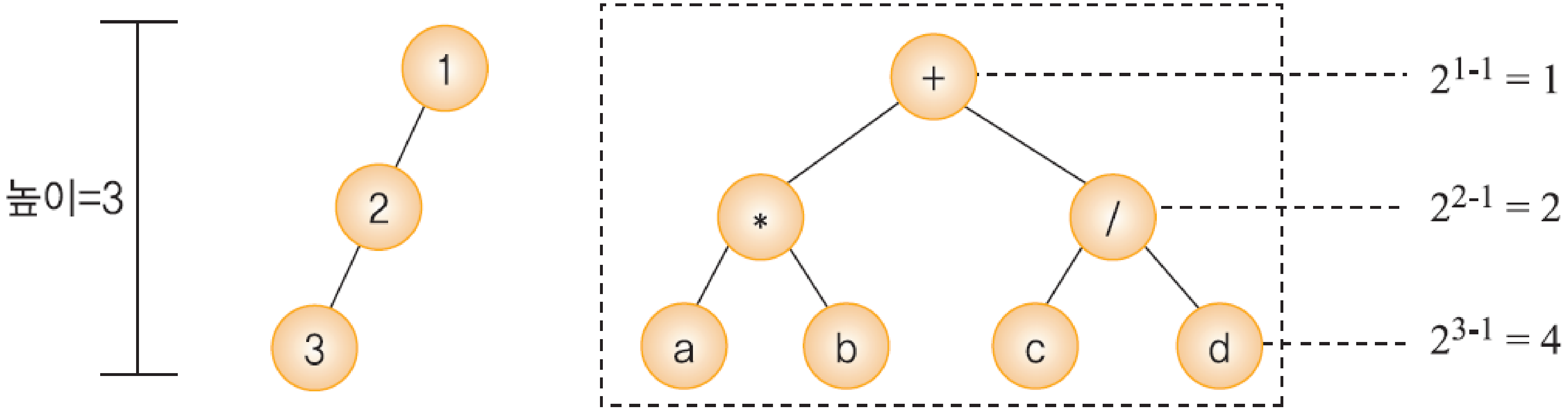
- 노드의 개수가 n개이면, 간선의 개수는 n-1



*노드의 개수: 7
*간선의 개수: 6

이진 트리의 성질(2)

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

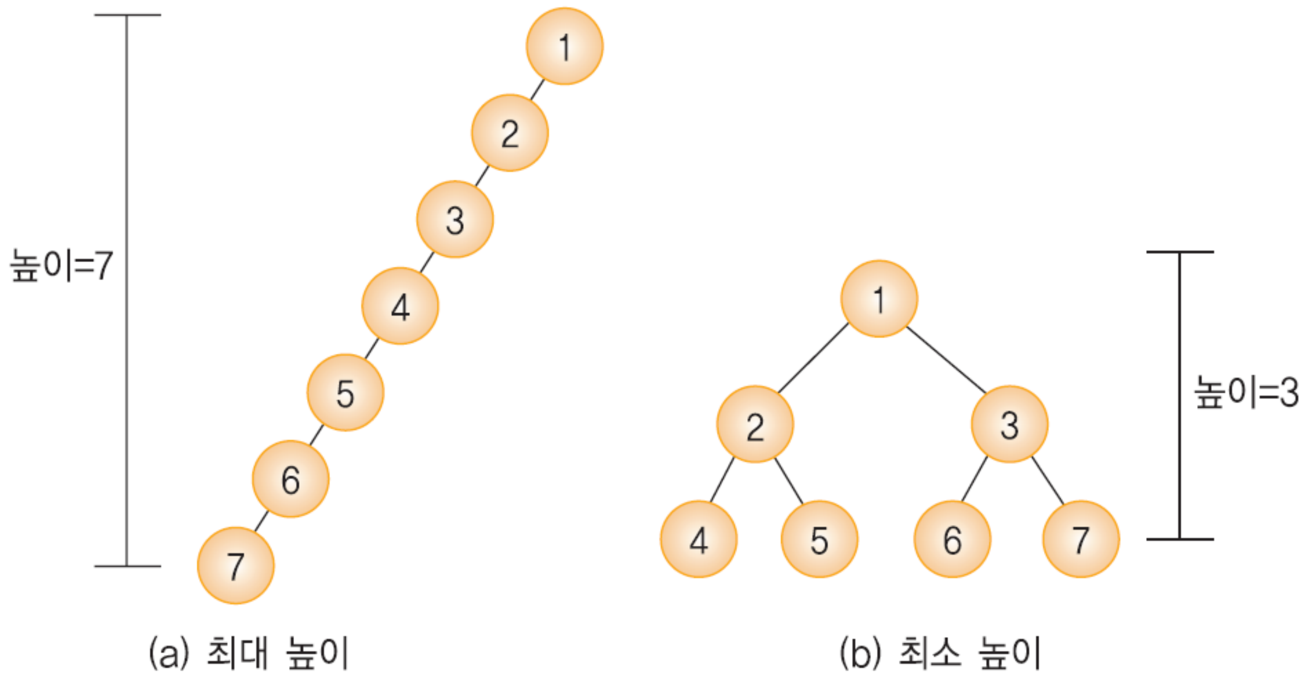


최소 노드 개수 = 3

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

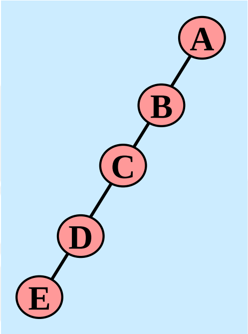
이진 트리의 성질(3)

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

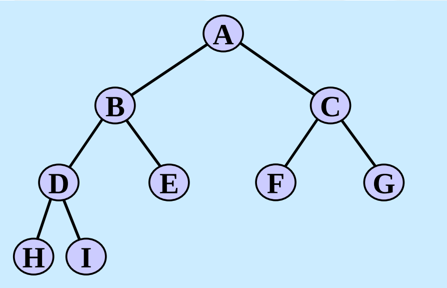


이진 트리의 종류

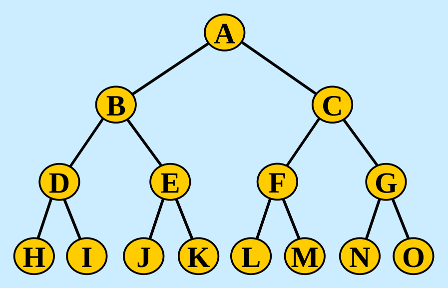
- 사향 이진 트리(skewed binary tree)
  - 모두 왼쪽 자식만 있거나(왼쪽 사향 이진 트리),
  - 모두 오른쪽 자식만 있는 트리(오른쪽 사향 이진 트리)



- 완전 이진 트리(complete binary tree)
  - 높이가 k일 때, 레벨 $k-1$ 까지는 노드가 완전히 차 있고, 레벨 k 에서는 노드가 왼쪽부터 차 있는 트리



- 포화 이진 트리(full binary tree)
  - 각 레벨에 노드가 모두 차 있는 이진 트리
  - 포화 이진 트리 $\Rightarrow$ 완전 이진 트리
  - 포화 이진 트리 $\nLeftarrow$ 완전 이진 트리

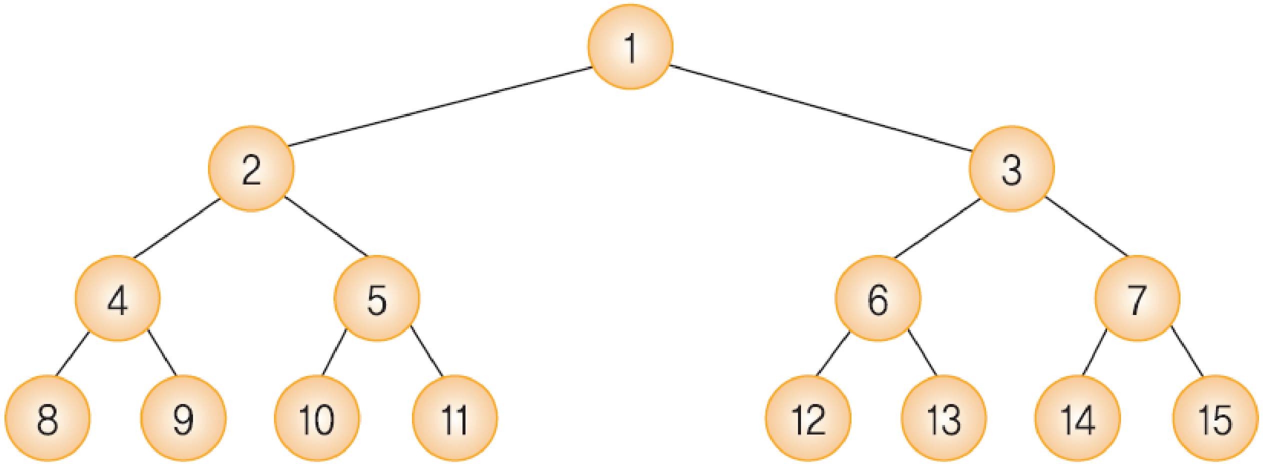


포화 이진 트리

- 트리의 각 레벨에 노드가 꽉 차있는 이진트리

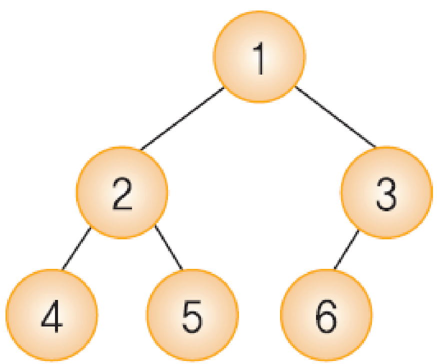
전체 노드 개수 : $2^{1-1} + 2^{2-1} + 2^{3-1} + \ldots + 2^{k-1} = \sum_{i=0}^{k-1} 2^i = 2^k - 1$

- 포화 이진 트리에는 다음과 같이 각 노드에 번호를 붙일 수 있음

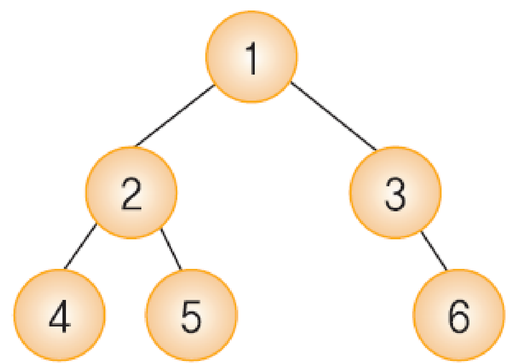


완전 이진 트리

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



(a) 완전 이진 트리



(b) 완전 이진 트리가 아님

이진 트리의 노드 수

- 레벨 i에 있는 최대 노드 수: 2^i
- 깊이가 k인 트리의 최대 노드 수: $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$
- 단말 노드 수를 n_0, 차수가 2인 노드 수를 n_2라 할 때, 항상 $n_0 = n_2 + 1$임

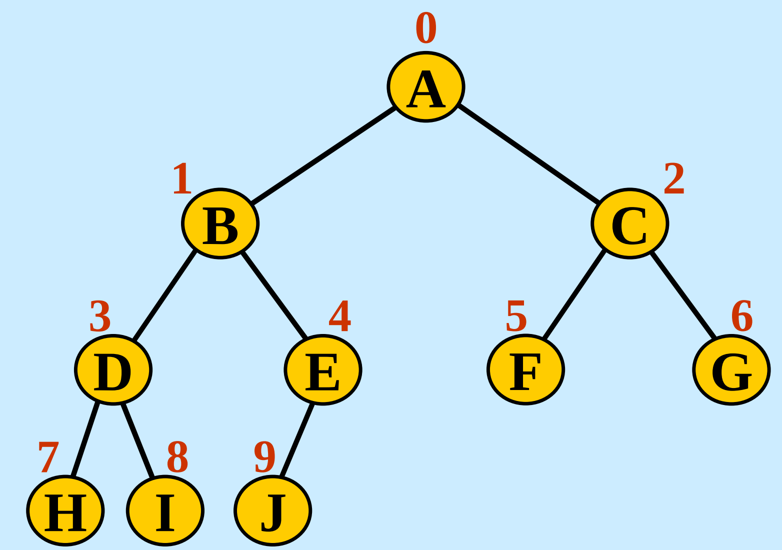
  (증명)

  전체 노드 수: n, 전체 링크 수: b

  - n_0: 자식이 없는 노드 수, n_1: 자식이 하나인 노드 수, n_2: 자식이 2개인 노드 수
  - $n=n_0+n_1+n_2$, $b+1=n$, $b=n_1+2 \cdot n_2$
    - $n_1 + 2 \cdot n_2 + 1 = n$
    - $n_1 + 2 \cdot n_2 + 1 = n_0+n_1+n_2$
    - $n_0 = n_2 + 1$

이진 트리의 표현

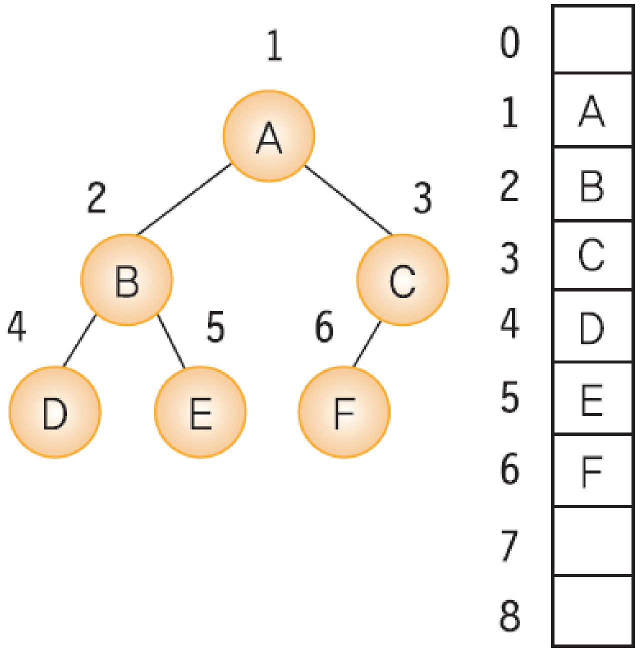
- 배열에 의한 방법
  - 각 노드에 0 ~ 의 번호를 부여
  - 배열의 노드번호 순서에 저장
- 연결 리스트에 의한 방법



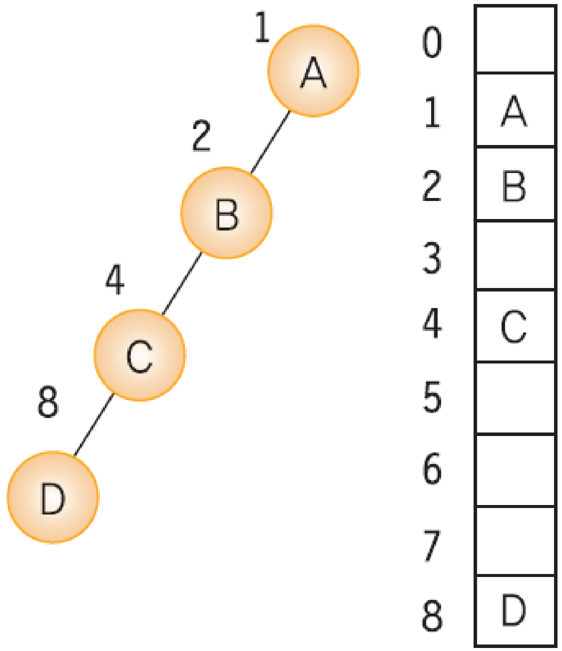
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
A	B	C	D	E	F	G	H	I	J

이진 트리의 표현: 배열 표현법

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



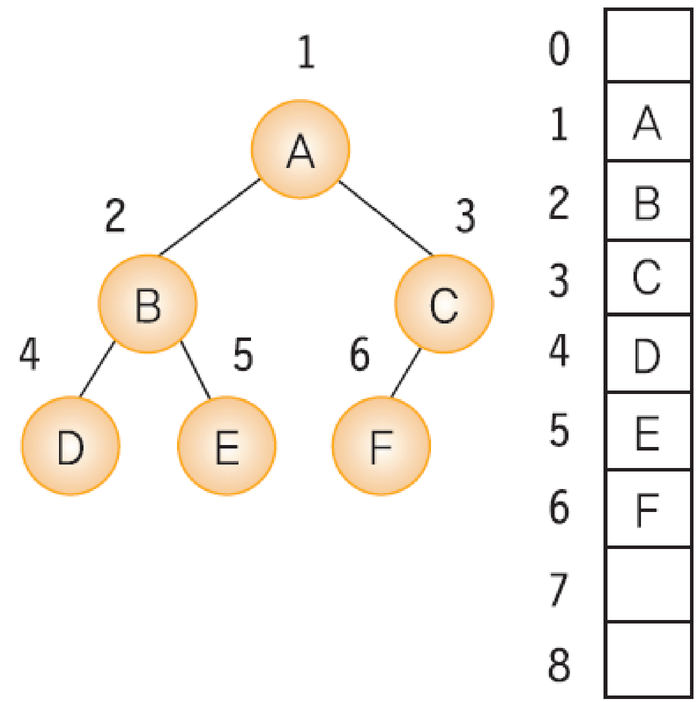
(a) 완전 이진 트리



(b) 경사 이진 트리

배열 표현법: 노드 위치 계산

- i 번째 노드의 부모 노드 위치
  - $\left\lfloor \frac{i-1}{2} \right\rfloor$, (단, $i \neq 0$)
- i 번째 노드의 왼쪽 자식 노드 위치
  - $2i+1$ (단, $2i+1 \le n-1$)
- i 번째 노드의 오른쪽 자식 노드 위치
  - $2(i+1)$ (단, $2(i+1) \le n-1$)

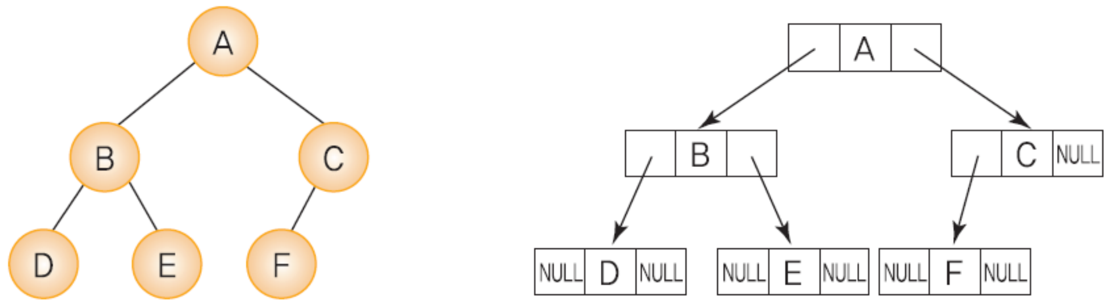


배열 표현법: 장단점

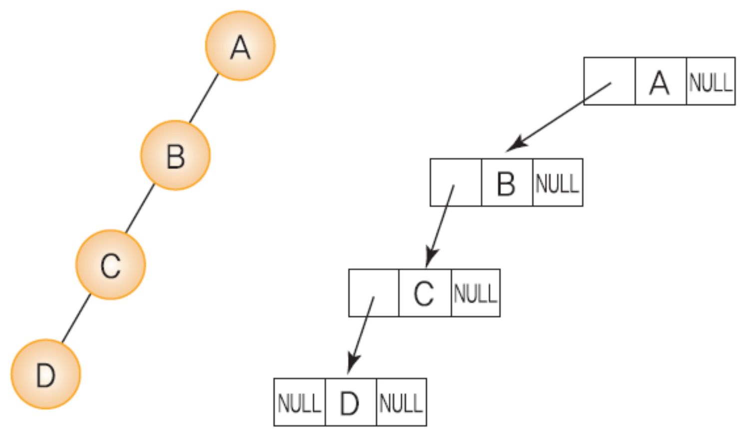
- 장점
  - 임의의 노드에 대해 부모, 자식 노드의 위치 계산이 쉬움
- 단점
  - 완전 이진 트리가 아니면 메모리 낭비가 큼
  - 예: 사향트리

이진 트리의 표현: 연결리스트 표현법

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



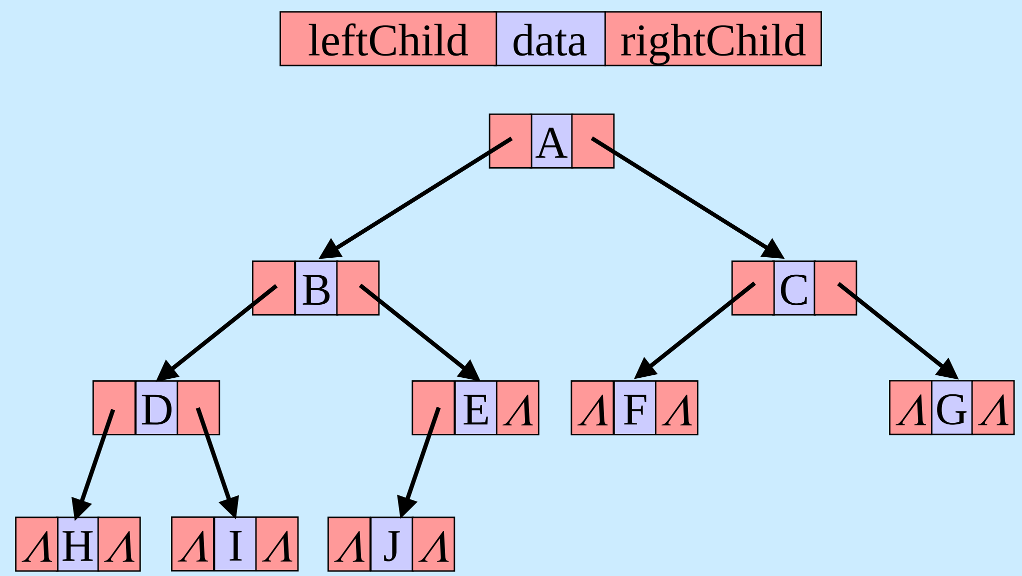
(a) 완전 이진 트리



(b) 경사 이진 트리

연결리스트 표현법에 의한 구현(1)

```
typedef struct nd *treePointer;
typedef struct nd {
    char data;
    treePointer  leftChild;
    treePointer  rightChild;
} node;
```




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

부모 링크 필드를 첨가하면 부모 노드 위치도 쉽게 알 수 있음

연결리스트 표현법에 의한 구현(2)

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

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

//      n1
//     /  |
//   n2   n3

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

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

연결리스트 표현법에 의한 구현(3)

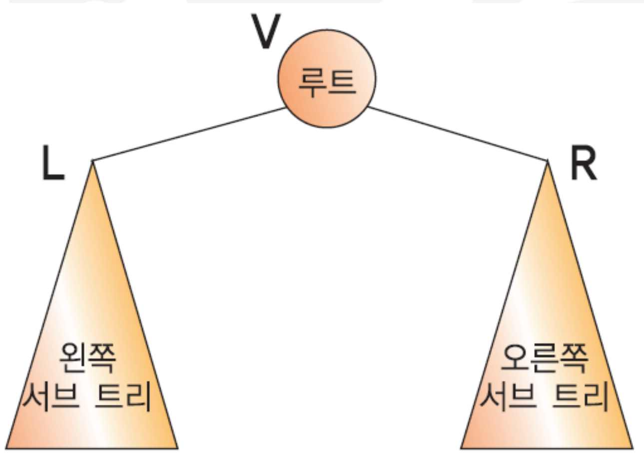
```
  n1->data = 10;     // 첫번째 노드를 설정한다.
  n1->left = n2;
  n1->right = n3;

  n2->data = 20;     // 두번째 노드를 설정한다.
  n2->right = NULL;

  n3->data = 30;     // 세번째 노드를 설정한다.
  n3->right = NULL;
}
```

이진 트리 순회(traversal)

- 트리 순회 : 트리의 모든 노드를 한번씩 방문하는 것
  - 트리의 구조를 분석하거나, 각 노드에 저장되어 있는 정보를 체계적으로 읽어올 때 사용

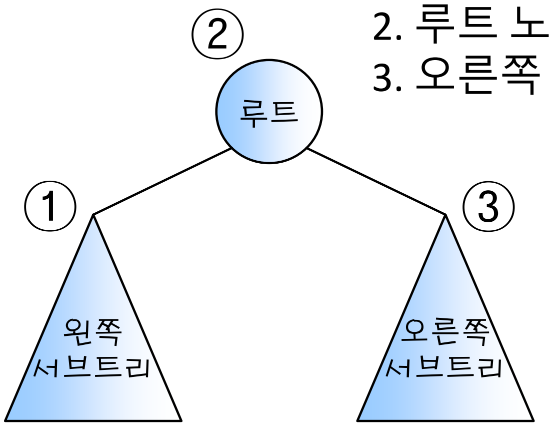
\>>순서대로 한번씩 방문<<

- 3가지의 기본적인 순회방법
  - 전위순회(preorder traversal): VLR
    - 자손 노드보다 루트 노드를 먼저 방문
  - 중위순회(inorder traversal): LVR
    - 왼쪽 자손, 루트, 오른쪽 자손 순으로 방문
  - 후위순회(postorder traversal): LRV
    - 루트 노드보다 자손을 먼저 방문

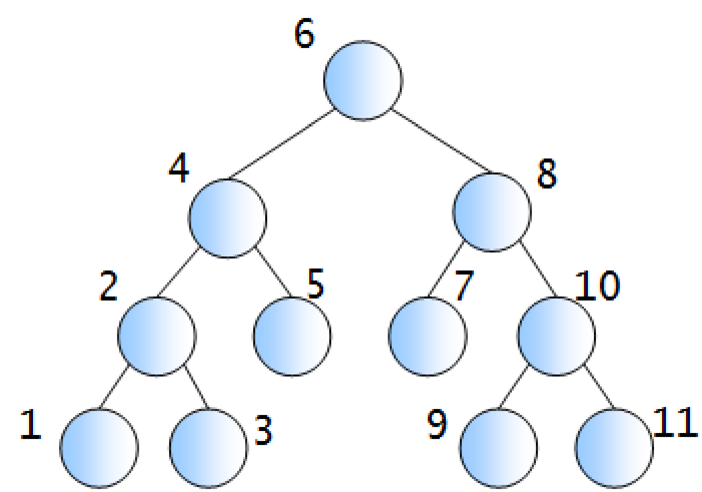


중위 순회

- 알고리즘
  - 먼저 트리의 왼쪽으로 NULL 노드를 만날 때까지 계속 내려간 다음, NULL 노드를 만나면 그 NULL 노드의 부모 노드를 방문하고 다시 오른쪽 부트리에 대해 같은 방법으로 수행함
  - 만약 오른쪽 부트리에 더 이상 방문할 노드가 없다면 트리의 바로 윗 레벨의 방문되지 않은 마지막 노드에 대해서 계속적으로 운행



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



중위 순회 알고리즘

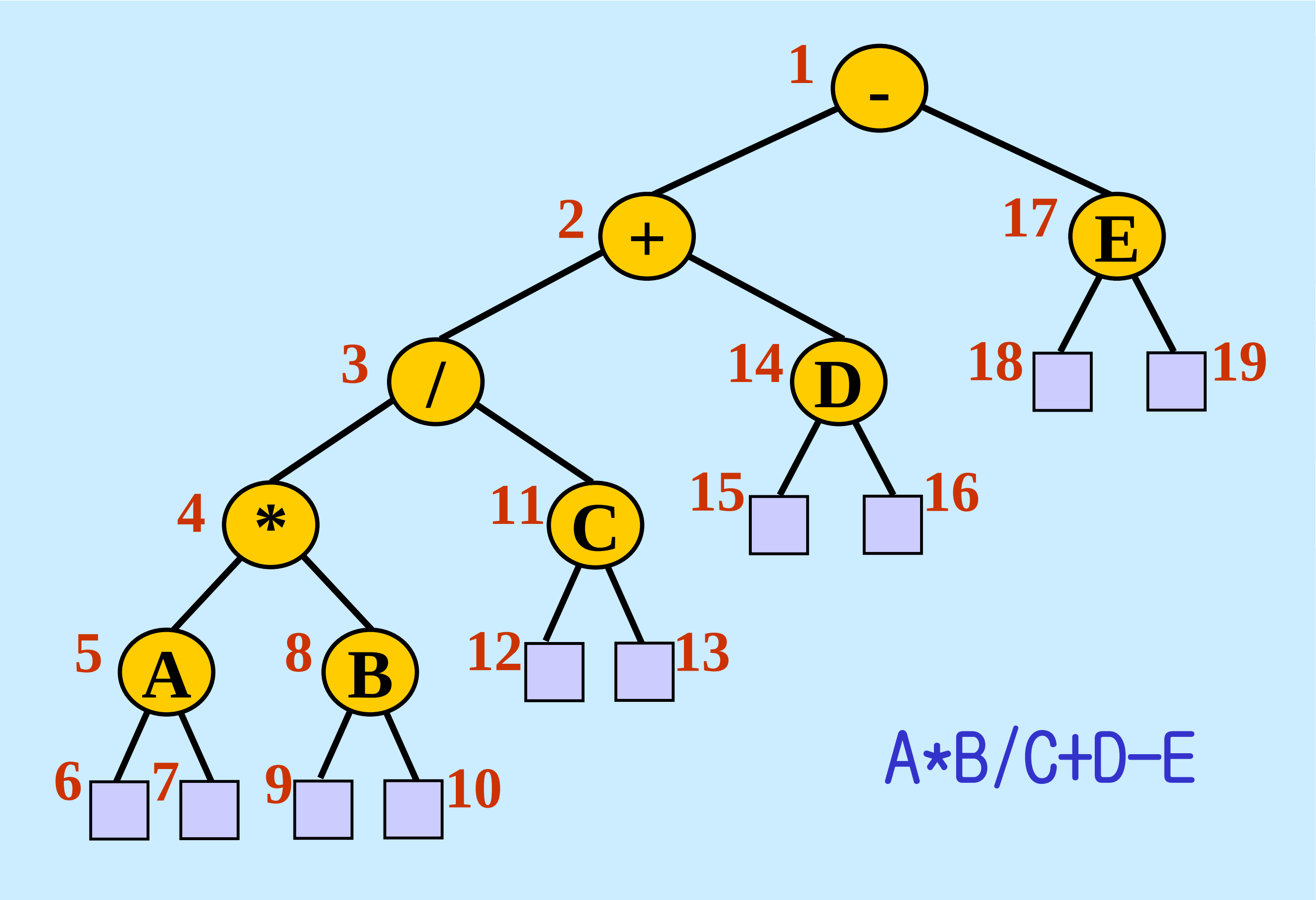
- 순환 호출 활용

```
inorder(x)

if x≠NULL
    then inorder(LEFT(x));
         print DATA(x);
         inorder(RIGHT(x));
```

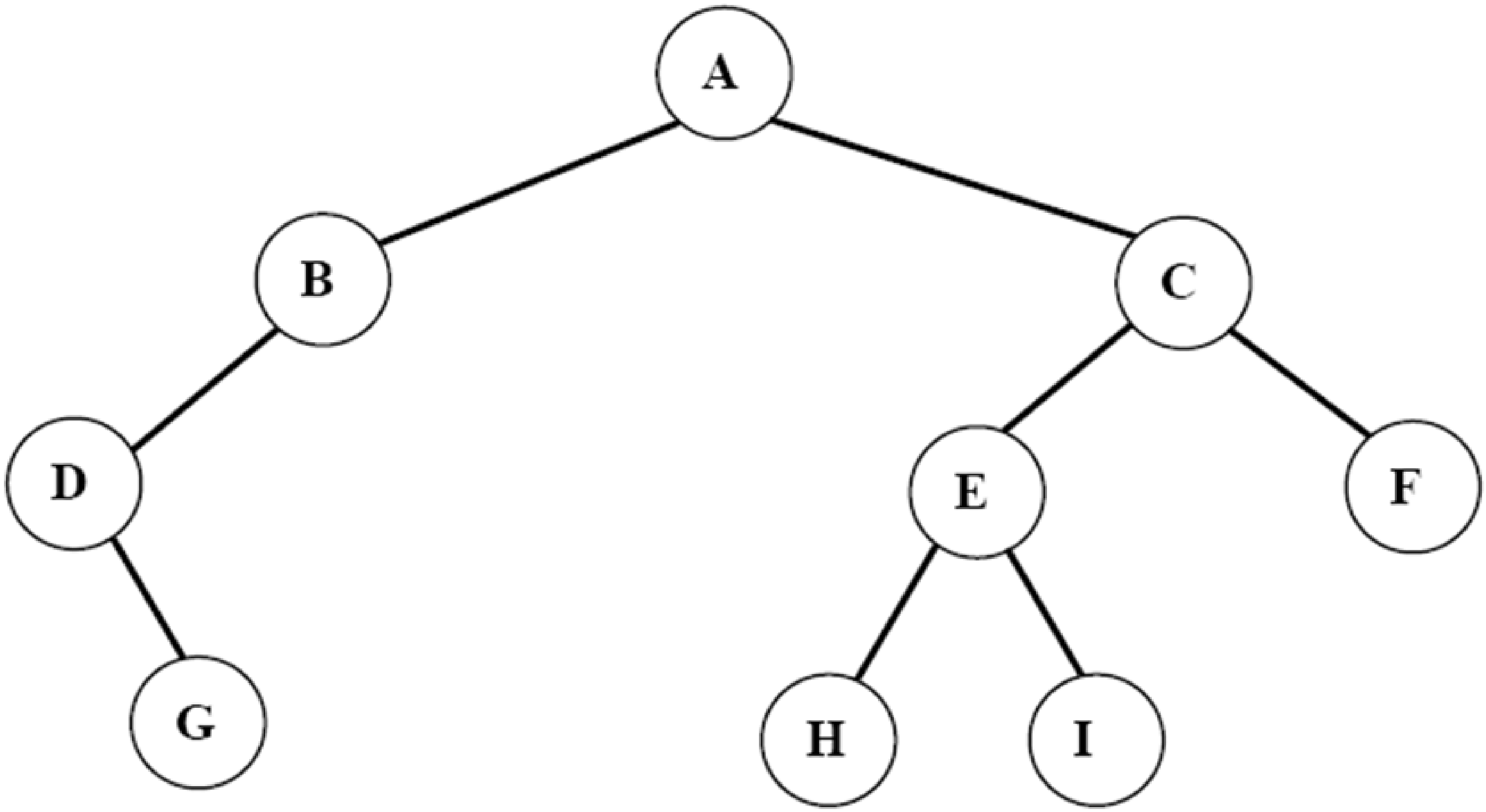
중위 순회 구현

```
void    inorder (treePointer ptr)
{
   if (ptr) {
       inorder(ptr->leftChild);//왼쪽 서브트리 방문
       printf("%c",ptr->data);// 현재 노드방문
       inorder(ptr->rightChild);//오른쪽 서브트리방문
    }
}
```

중위 순회 예(1)



중위 순회 예(2)

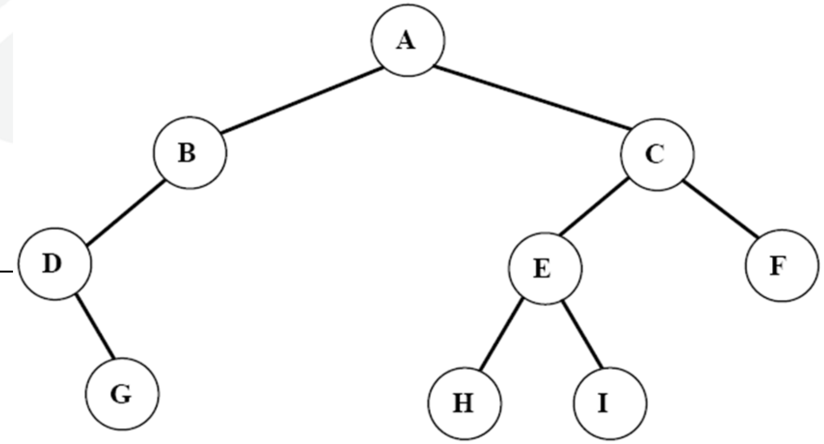


중위 순회 Complexity

- 재귀적 프로그램의 시간 복잡도: O(n)
  - 기본 연산: printf(); inorder();
    - printf()의 수행 횟수
      - 각 노드의 값은 한번 씩만 출력됨: n번
    - inorder()의 수행 횟수
      - 모든 노드의 수 + null 링크 방문 수
        - null 링크 방문 수 = 단말 노드의 수 * 2
          - (모든 노드의 수 > 단말 노드의 수)
      - inorder()의 수행 횟수 < n+2n
- 공간 복잡도: O(n)
  - 사향 트리일 때 최악의 경우 c*n번(모든 노드 수+ null링크 방문 수)의 재귀 호출이 가능함
  - 재귀 호출 때 마다 ptr이 스택에 쌓이므로 최악의 경우 필요한 스택의 크기는 c*n임

중위 순회 Complexity - 반복문

```
void  iterInorder (treePointer ptr)
{
   int                  top = -1;
   treePointer     stack[MAX_SIZE];
   for (;;) {
      for (; ptr; ptr=ptr->leftChild)
          push(&top, ptr);
      ptr = pop(&top);    /* 스택에서 팝, 공스택일 경우 NULL을 반환 */
      if (!ptr)
          break;
      printf("%c", ptr->data);
      ptr= ptr->rightChild;
   }
}
```

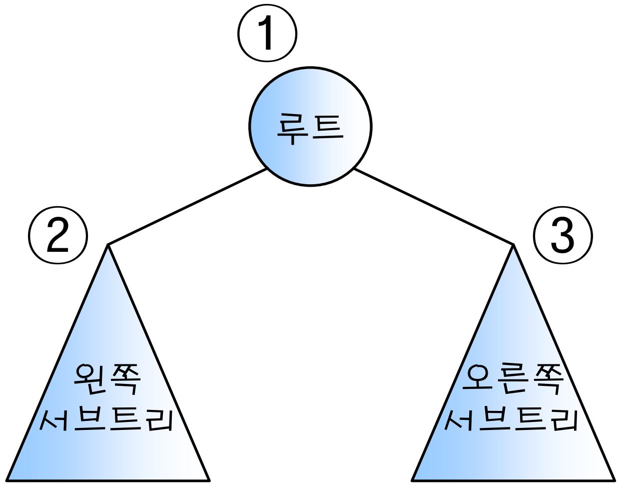


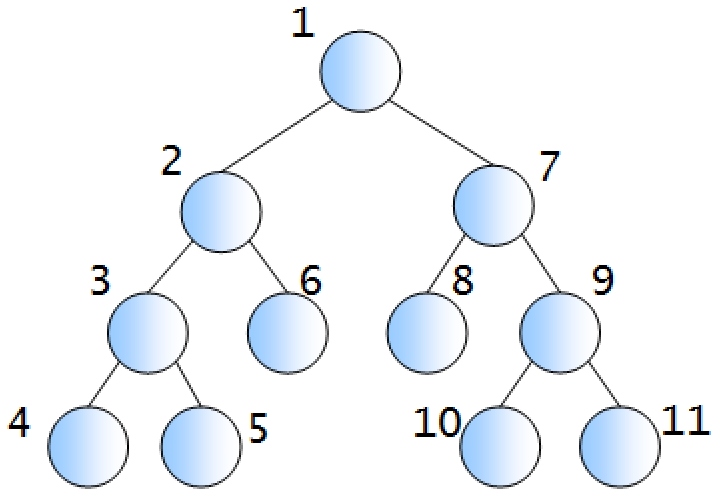
Inorder: *DGBAHEICF*

반복문을 이용한 중위 순회의 시간 복잡도:
O(n)

전위 순회

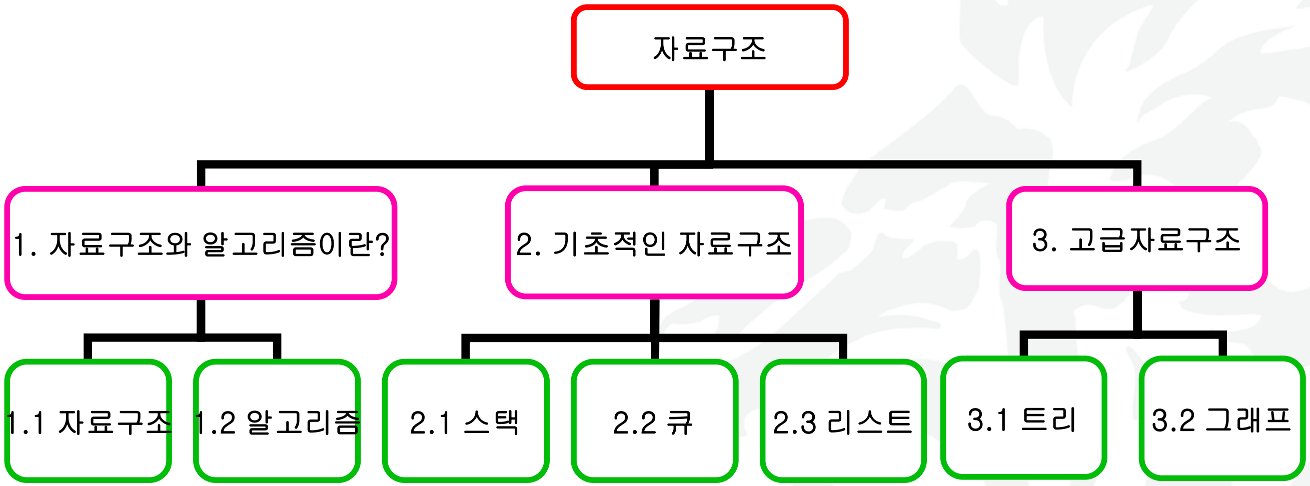
- 기본 알고리즘
  - 1. 루트 노드를 방문
  - 2. 왼쪽 서브트리를 방문
  - 3. 오른쪽 서브트리를 방문





전위 순회 응용

- 구조화 된 문서 출력



전위 순회 알고리즘

- 순환 호출 활용

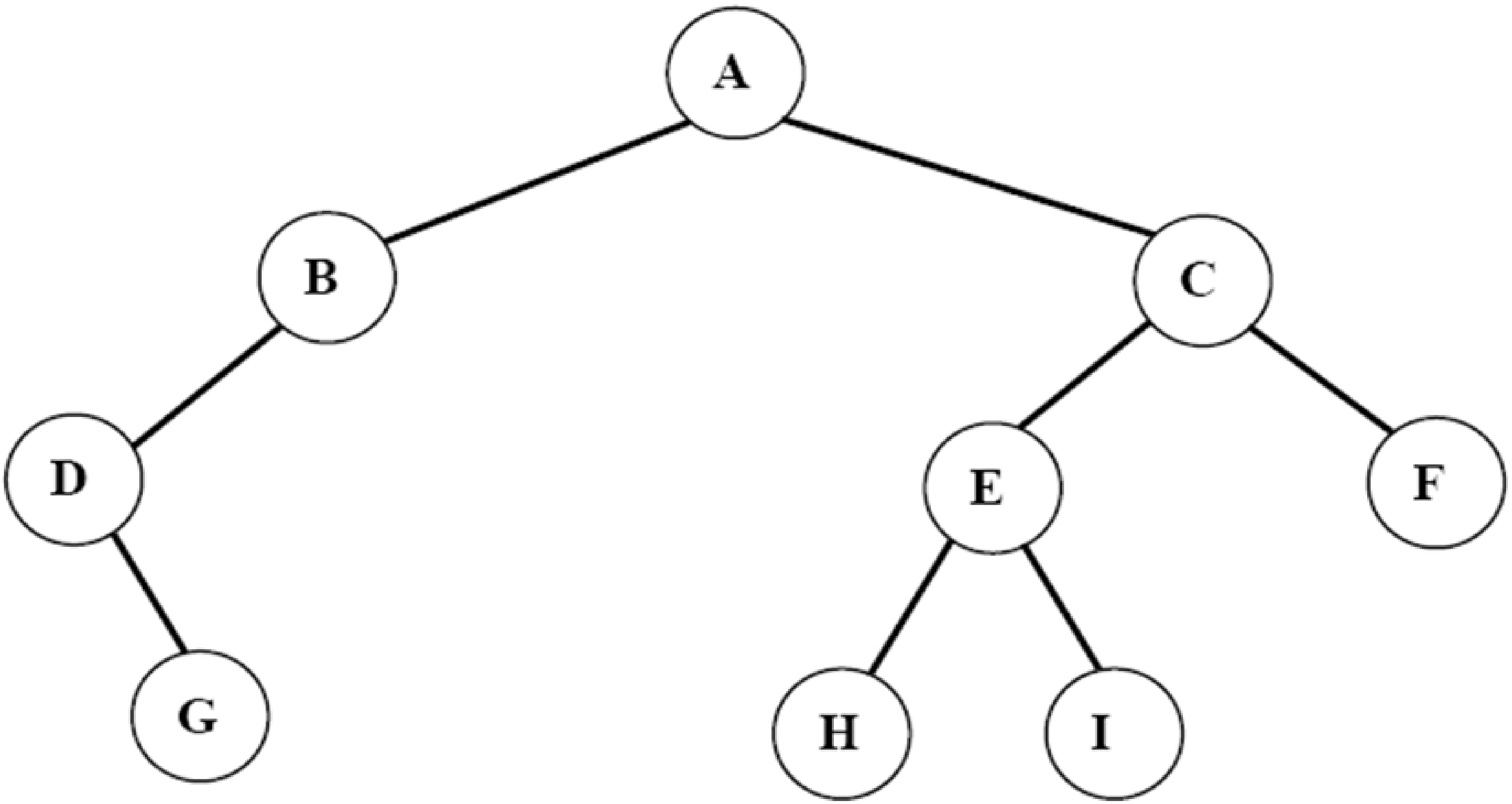
```
preorder(x)

if x≠NULL
    then print DATA(x);
        preorder(LEFT(x));
        preorder(RIGHT(x));
```

전위 순회 구현

- 순환 호출 활용

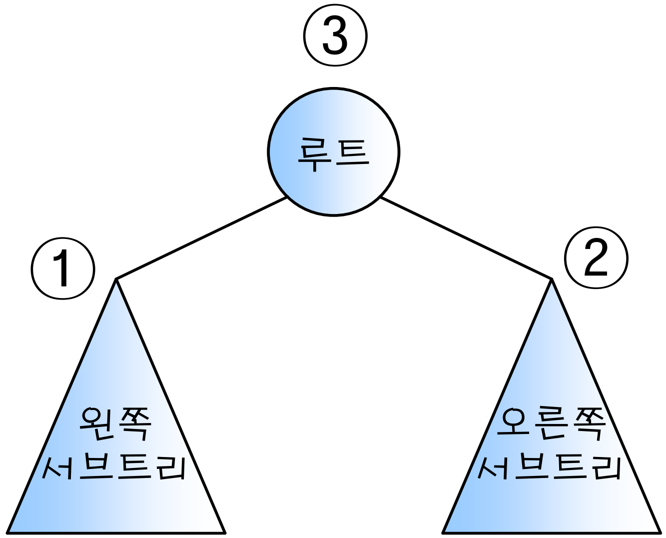
```
void  preorder (treePointer ptr)
{
    if (ptr) {
        printf("%c",ptr->data);
        preorder(ptr->leftChild);
        preorder(ptr->rightChild);
    }
}
```

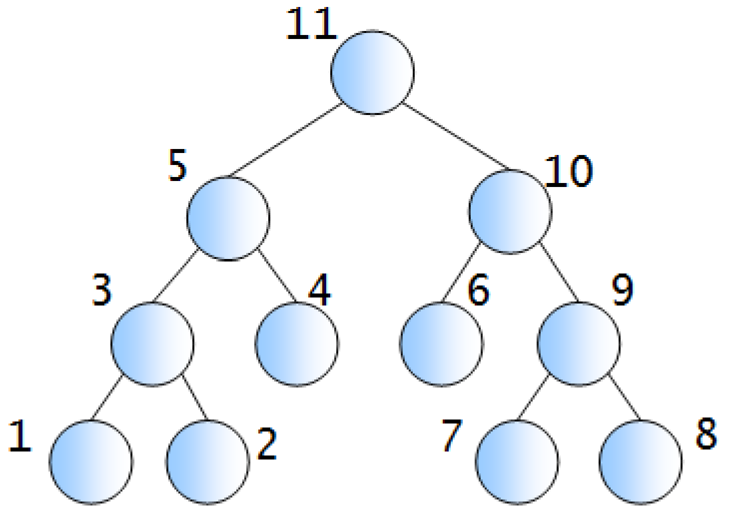
전위 순회 예



후위 순회

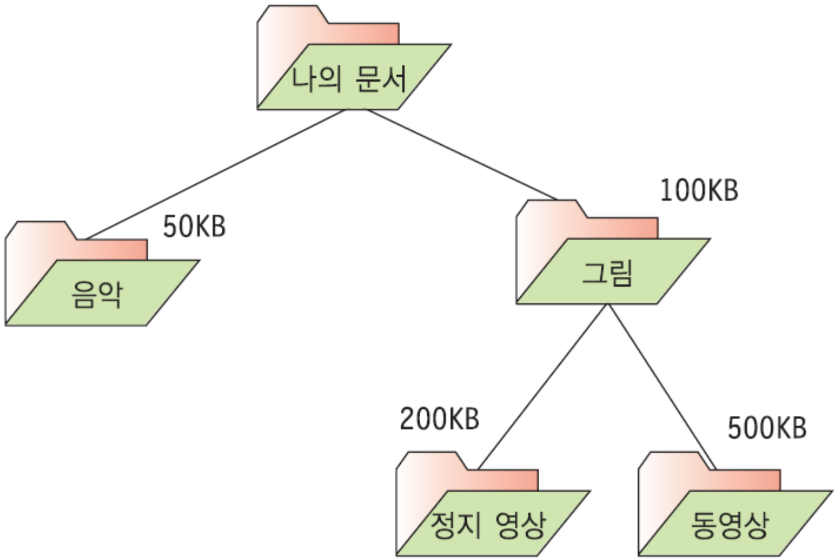
- 알고리즘
  - 1. 왼쪽 서브트리를 방문
  - 2. 오른쪽 서브트리를 방문
  - 3. 루트 노드를 방문





후위 순회 응용

- 디렉토리 용량 계산



후위 순회 알고리즘

- 순환 호출 활용

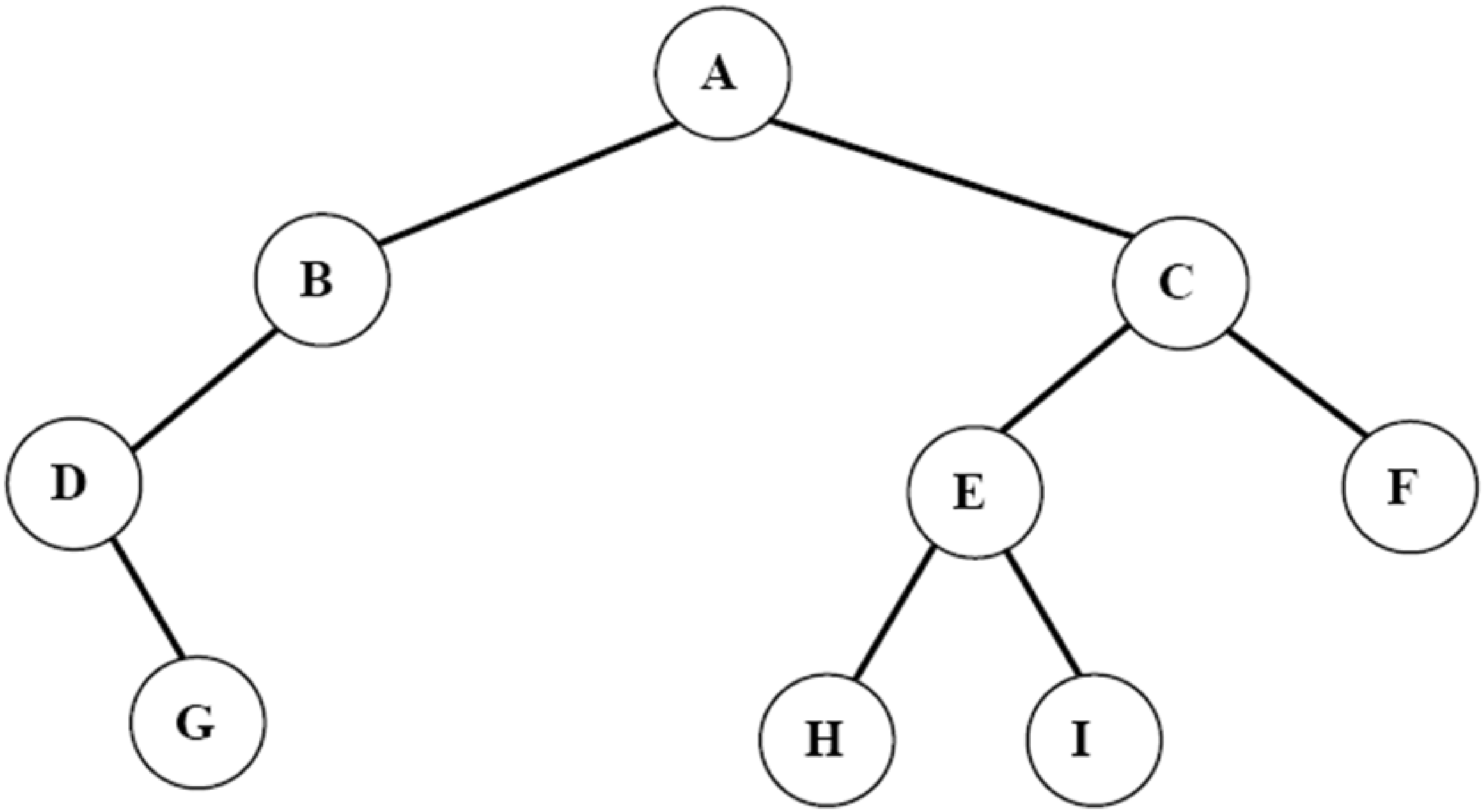
```
postorder(x)

if x≠NULL
    then postorder(LEFT(x));
         postorder(RIGHT(x));
         print DATA(x);
```

후위 순회 구현

- 순환 호출 활용

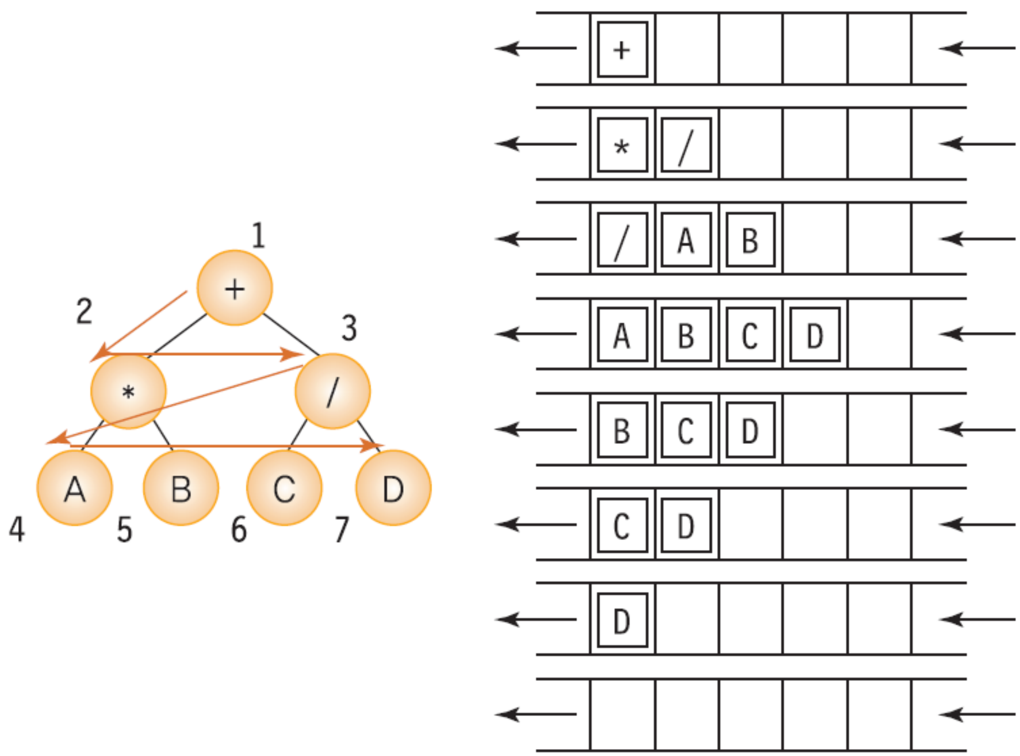
```
void   postorder (treePointer ptr)
{
    if (ptr) {
        postorder(ptr->leftChild);
        postorder(ptr->rightChild);
        printf("%c",ptr->data);
    }
}
```

후위 순회 예

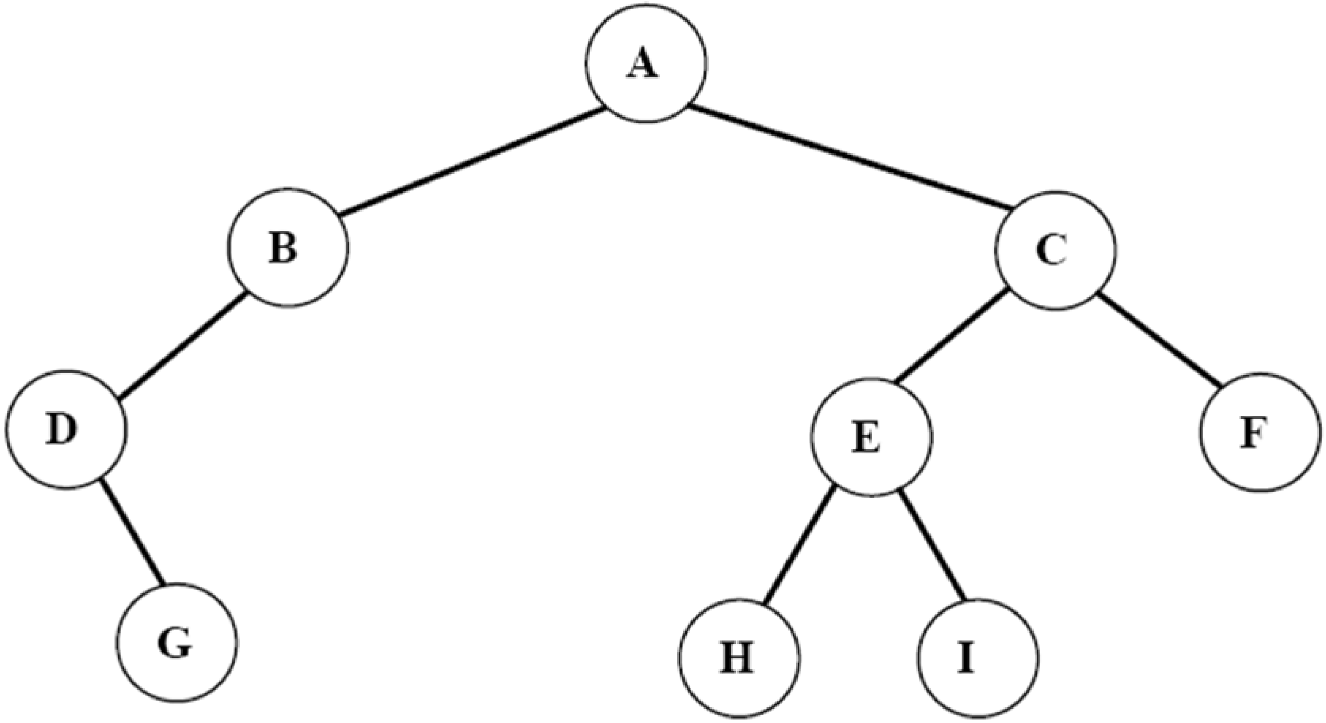


레벨 순회

- 레벨 순회(level order)
  - 각 노드를 레벨순으로 검사하는 순회 방법
  - 지금까지의 순회 방법이 스택을 사용했던 것에 비해 레벨 순회는 큐를 사용하는 순회 방법



레벨 순회 예



레벨 순회 알고리즘

```
level_order(root)


initialize queue;
enqueue(queue, root);
while is_empty(queue)≠TRUE do
        x← dequeue(queue);
        print x→data;
        if(x → left != NULL) then
                enqueue(queue, x→ left);
        if(x → right !=NULL) then
                enqueue(queue, x→ right);
```

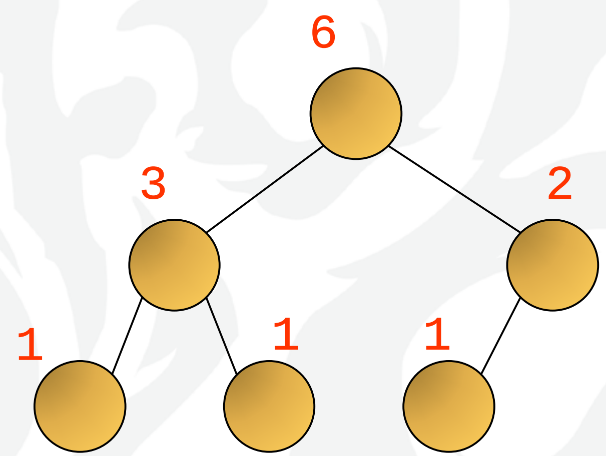
레벨 순회 구현

```
void  levelOrder (treePointer ptr) {
     int  front = rear = 0;
     treePointer queue[MAX_SIZE];
     if (!ptr)
          return;                                      // empty tree
     enqueue(front, &rear, ptr);
     for (;;) {
         ptr = dequeue(&front, rear);      // 큐가 비어 있으면 null 반환
         if (ptr) {
              printf("%c", ptr->data);
              if (ptr->leftChild)
                   enqueue(front, &rear, ptr->leftChild);
              if (ptr->rightChild)
                   enqueue(front, &rear, ptr->rightChild);
         }
         else
              break;
     }
}
```

이진트리연산: 노드개수

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

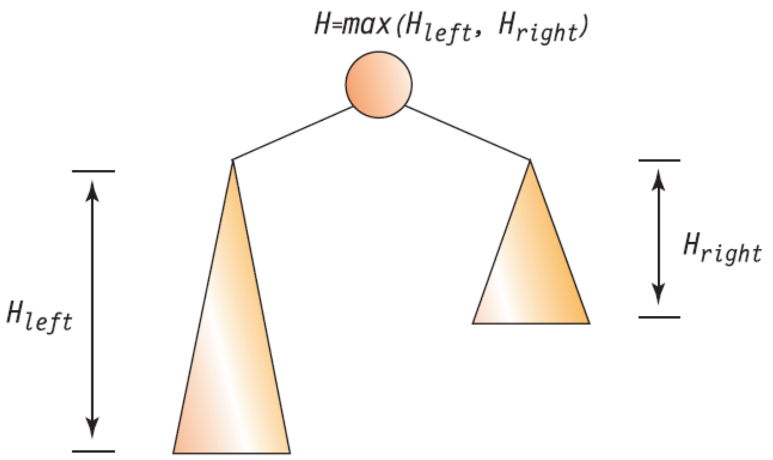
```
int get_node_count(TreeNode *node)
{
  int count=0;
  if( node != NULL )
    count = 1 + get_node_count(node->left)+
        get_node_count(node->right);
  return count;
}
```



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;
}
```



Week 7: Tree