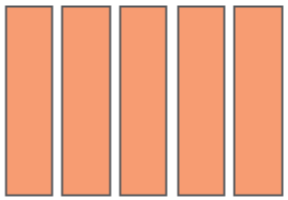
8장 트리

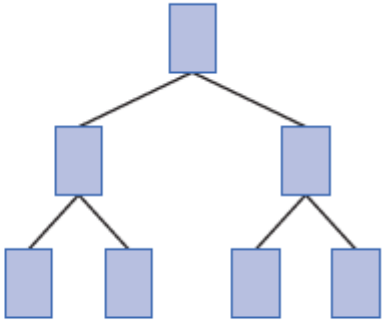
트리 개념, 이진 트리 표현, 순회 알고리즘

8.1 트리(TREE)의 개념

- 리스트, 스택, 큐 등은 선형 구조
- 트리: 계층적인 구조를 나타내는 자료구조



선형 자료구조

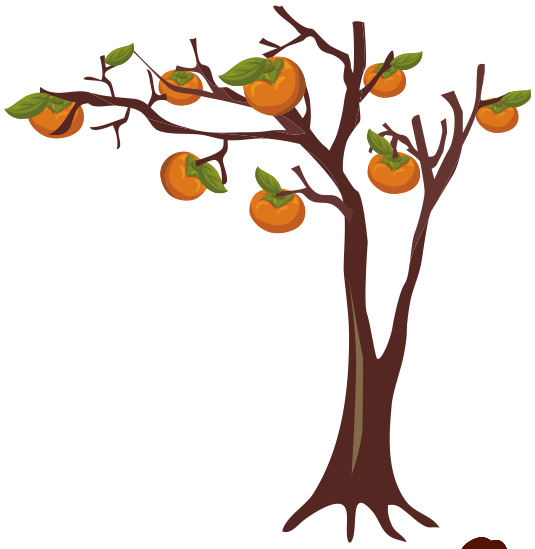


비선형 자료구조

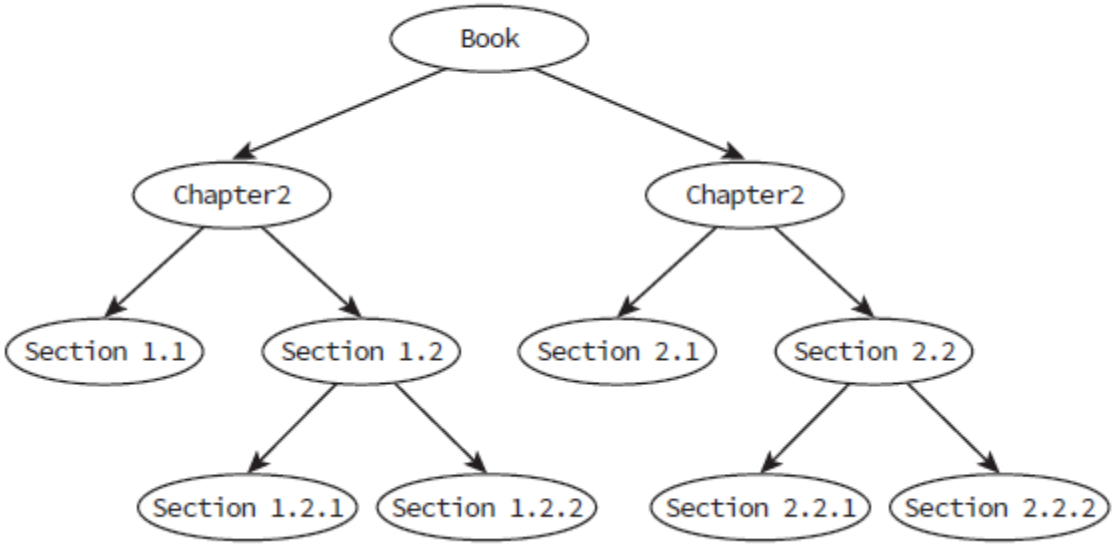
트리(TREE)

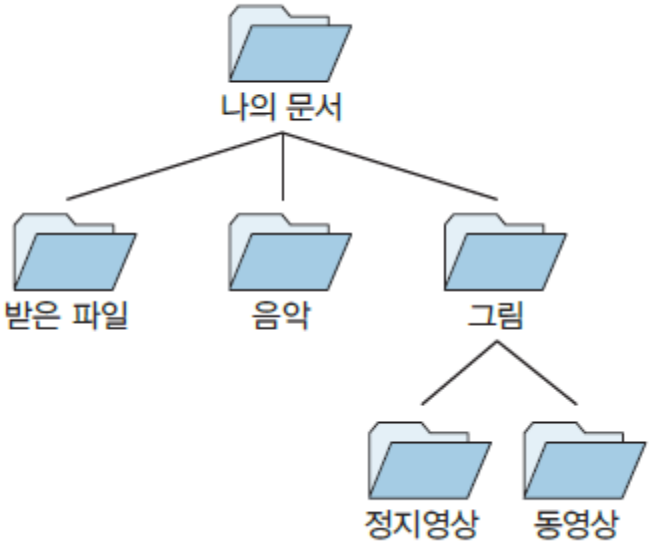
- 트리는 부모-자식 관계의 노드들로 이루어진다.

- 응용분야:
  - 계층적인 조직 표현
  - 컴퓨터 디스크의 디렉토리 구조
  - 인공지능에서의 결정트리 (decision tree)



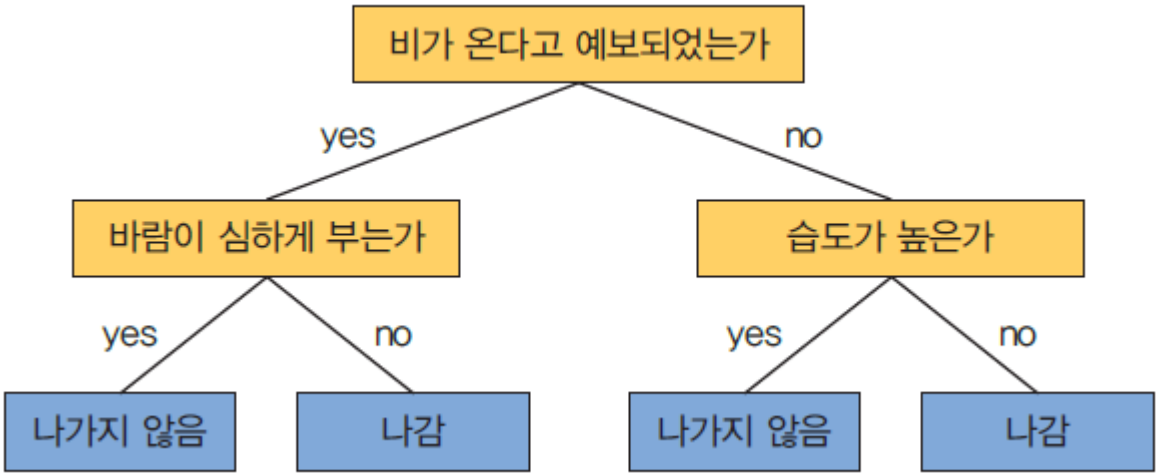
트리의 예


Book
Chapter2
Chapter2
Section 1.1
Section 1.2
Section 2.1
Section 2.2
Section 1.2.1
Section 1.2.2
Section 2.2.1
Section 2.2.2

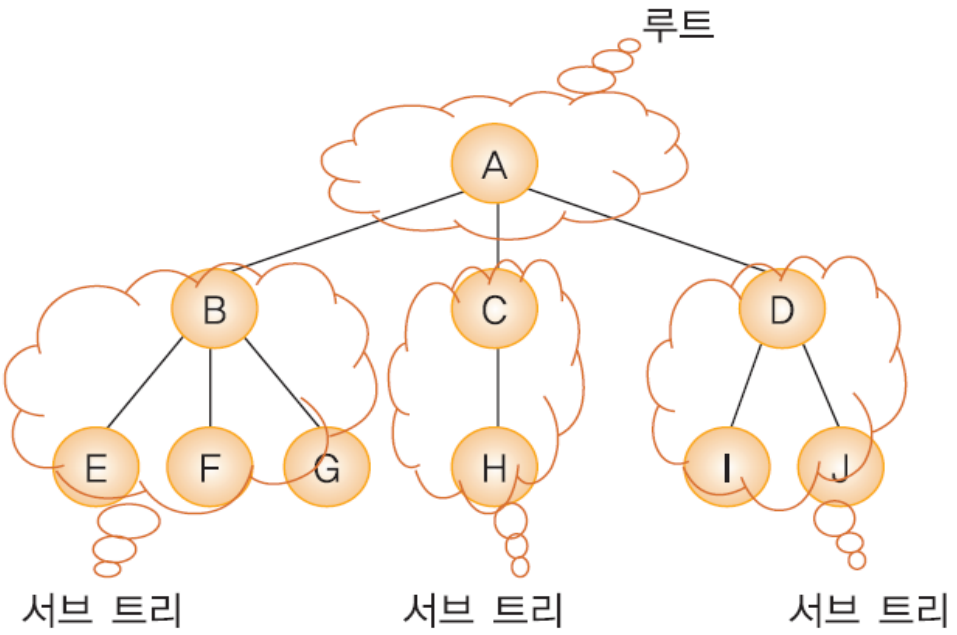

나의 문서
받은 파일
음악
그림
정지영상
동영상

트리의 예: 결정 트리 (인공지능 응용)

- (예) 골프에 대한 결정 트리

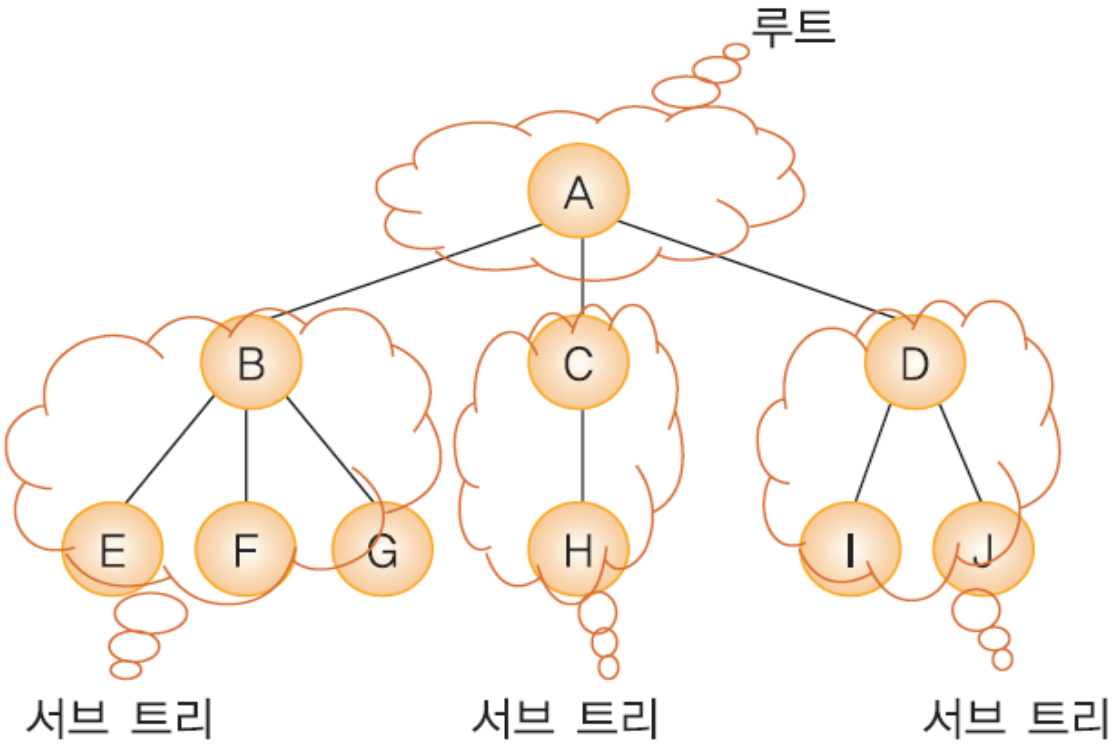


트리의 용어 (1)



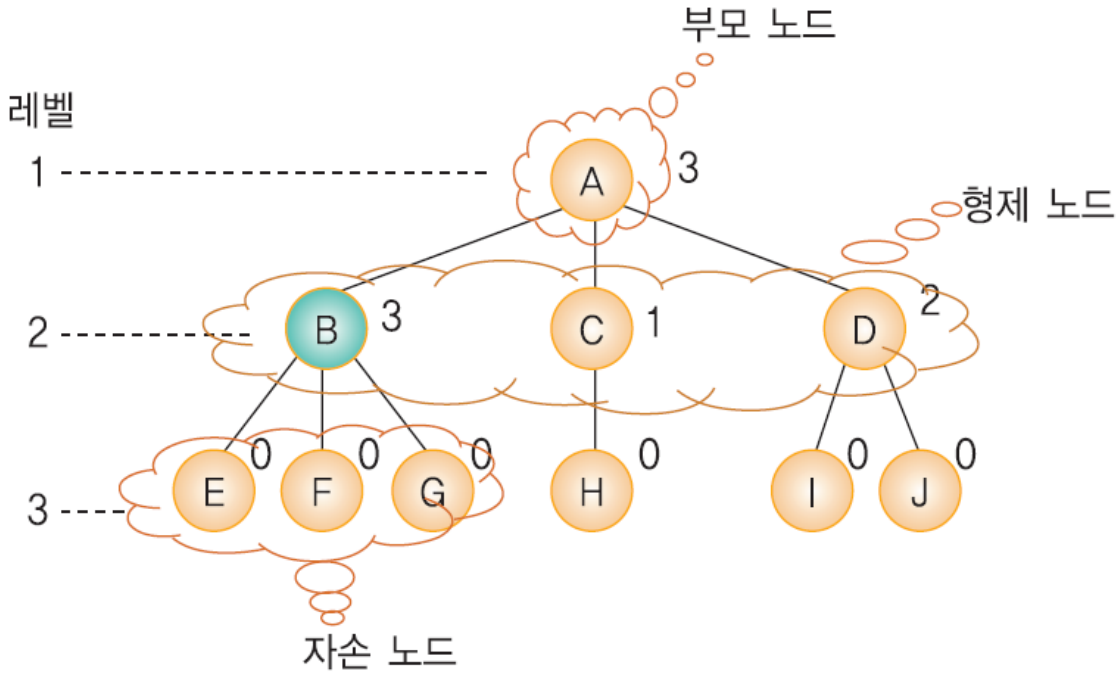
- 노드(node): 트리의 구성요소
- 루트(root): 부모가 없는 노드(A)
- 서브트리(subtree): 하나의 노드와 그 노드들의 자손들로 이루어진 트리

트리의 용어 (2)



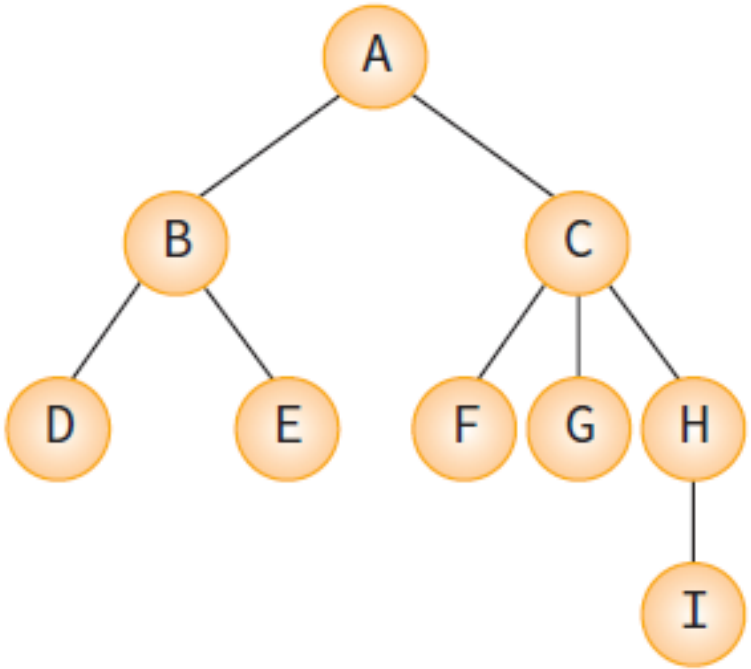
- 비단말노드(non-terminal node): 적어도 하나의 자식을 가지는 노드(A,B,C,D)
- 단말노드(terminal node): 자식이 없는 노드 (E,F,G,H,I,J)

트리의 용어 (3)



- 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

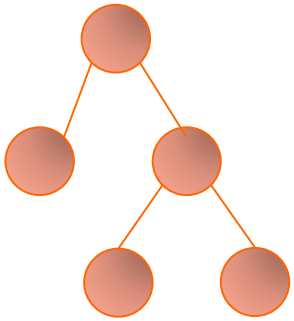


- A는 루트 노드이다.
- B는 D와 E의 부모노드이다.
- C는 B의 형제(sibling) 노드이다.
- D와 E는 B의 자식노드이다.
- B의 차수는 2이다.
- 위의 트리의 높이는 4이다.

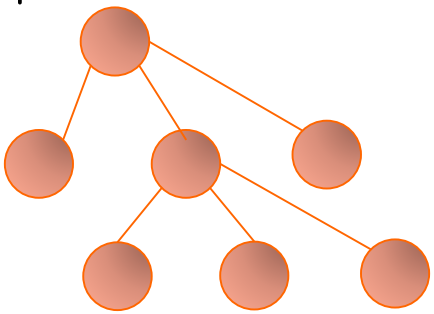
트리의 종류

트리

- 이진 트리: 모든 노드의 차수가 2이하



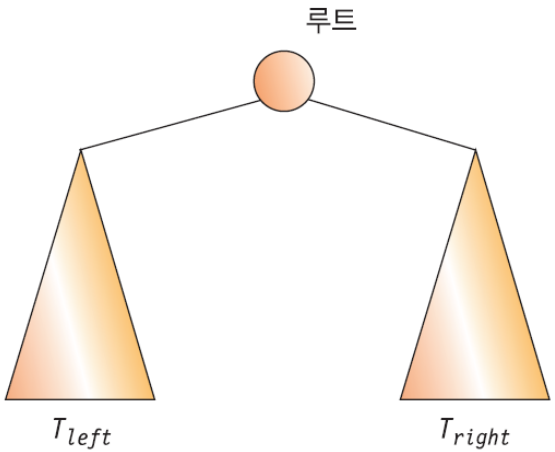
- 일반 트리



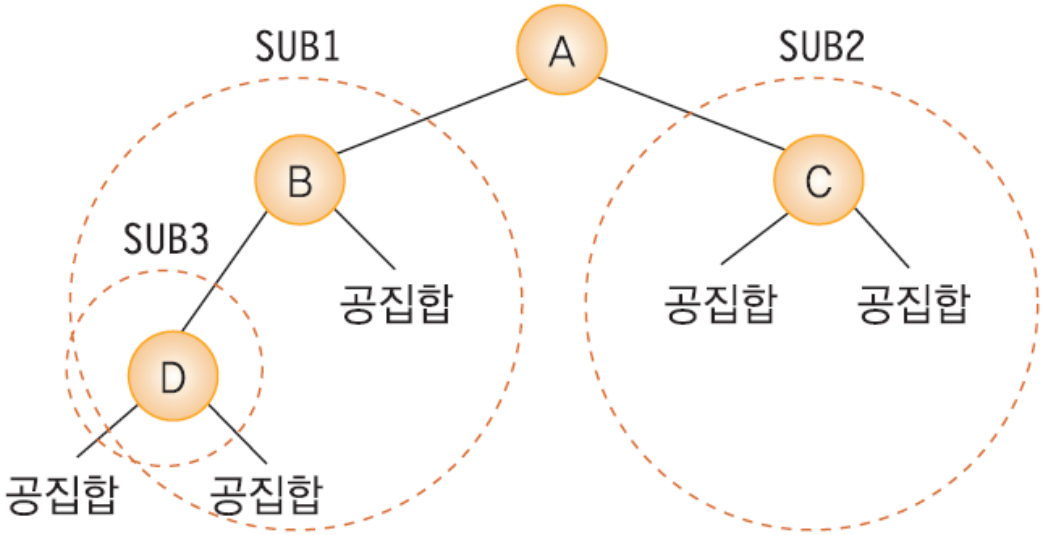
8.2 이진 트리 (binary tree) 소개

- **이진 트리(binary tree)** : 모든 노드가 2개의 서브 트리를 가지고 있는 트리
  - 단, 서브트리는 공집합일 수 있다.

- 모든 노드의 차수가 2 이하가 된다-> 구현하기가 편리함
- 이진 트리에는 서브 트리간의 순서가 존재



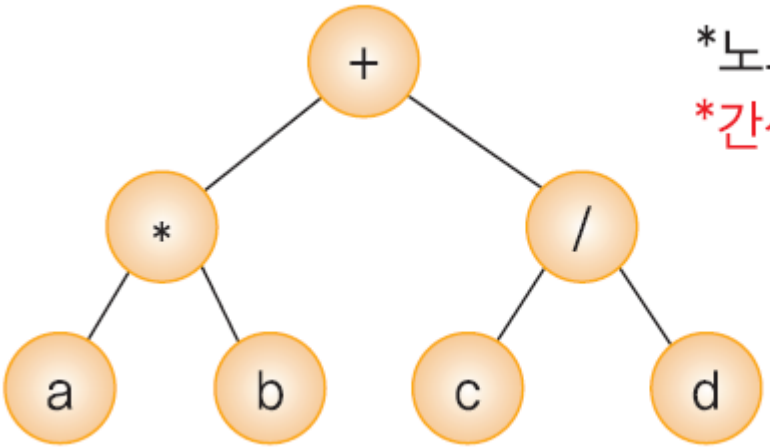
이진 트리 개념



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

이진 트리의 성질 (1)

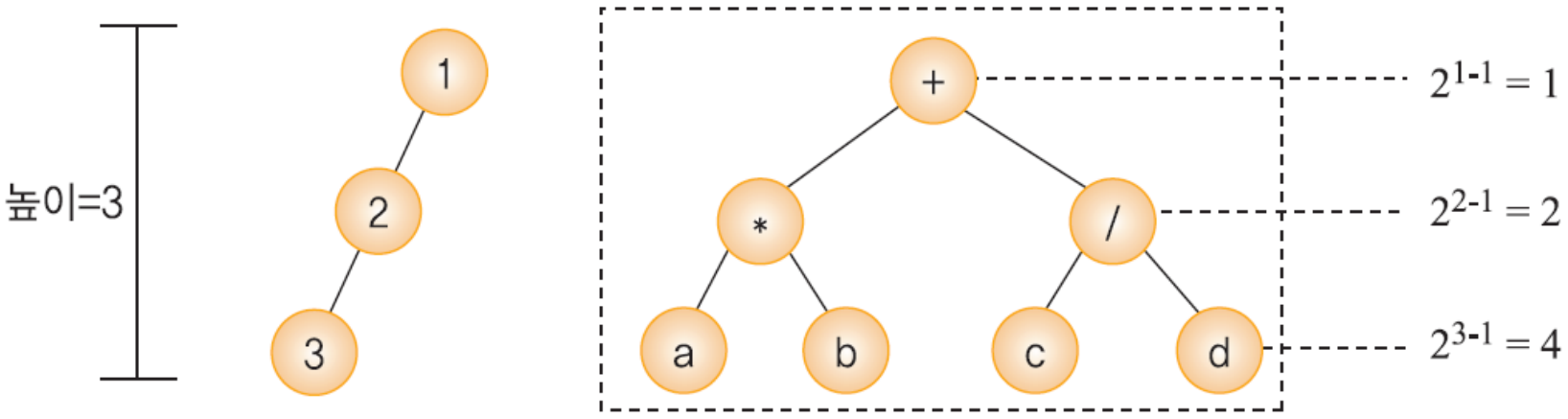
- 노드의 개수가 n개이면 간선(edge)의 개수는 n-1



*노드의 개수: 7
*간선의 개수: 6

이진 트리의 성질 (2)

- 높이가 h인 이진 트리의 경우, 최소 h개의 노드를 가지며 최대 2^h-1개의 노드를 가진다.
  - 레벨 i에서의 최대 노드 수: 2^{i-1}
  - 최대 노드 수: $\sum_{i=1}^{h} 2^{i-1} = 2^h - 1$

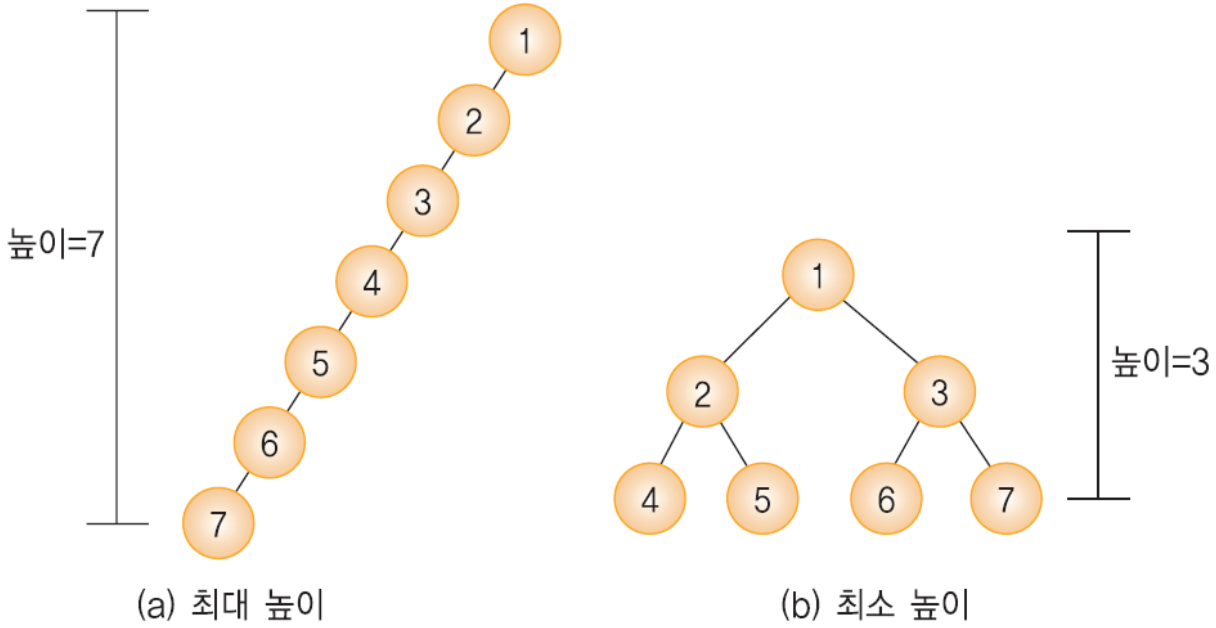


최소 노드 개수 = 3

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

이진 트리의 성질 (3)

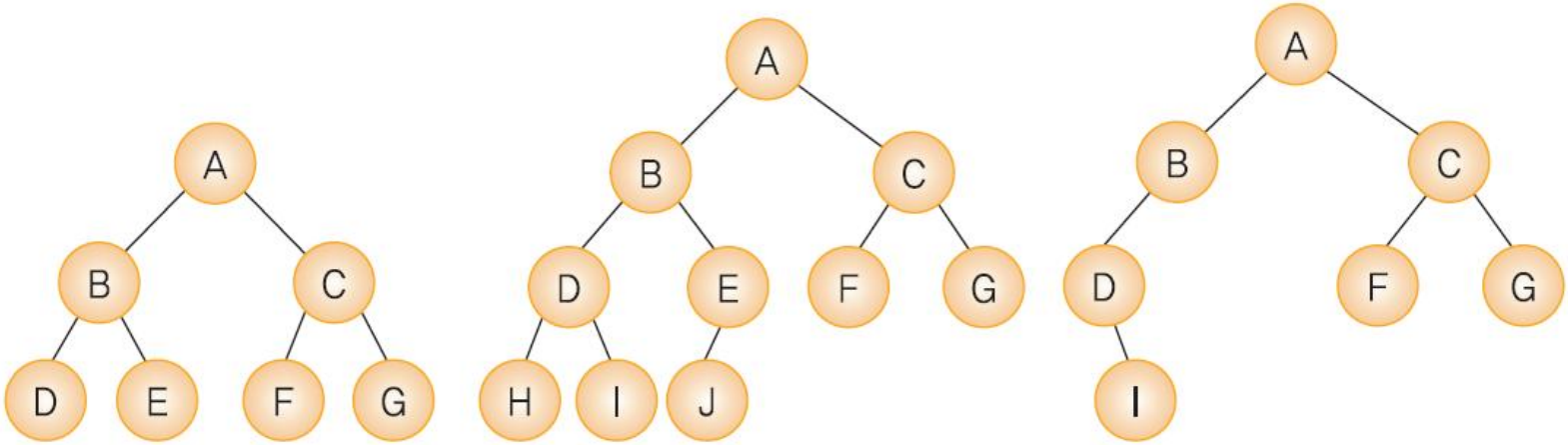
- n개의 노드를 가지는 이진트리의 높이
  - 최대: n
  - 최소: $\lceil \log_2(n+1) \rceil$
    - $n \leq 2^h - 1 \Rightarrow h \geq 1og_2(n+1)$, 이때 h는 정수



(a) 최대 높이 (b) 최소 높이

이진 트리의 분류

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



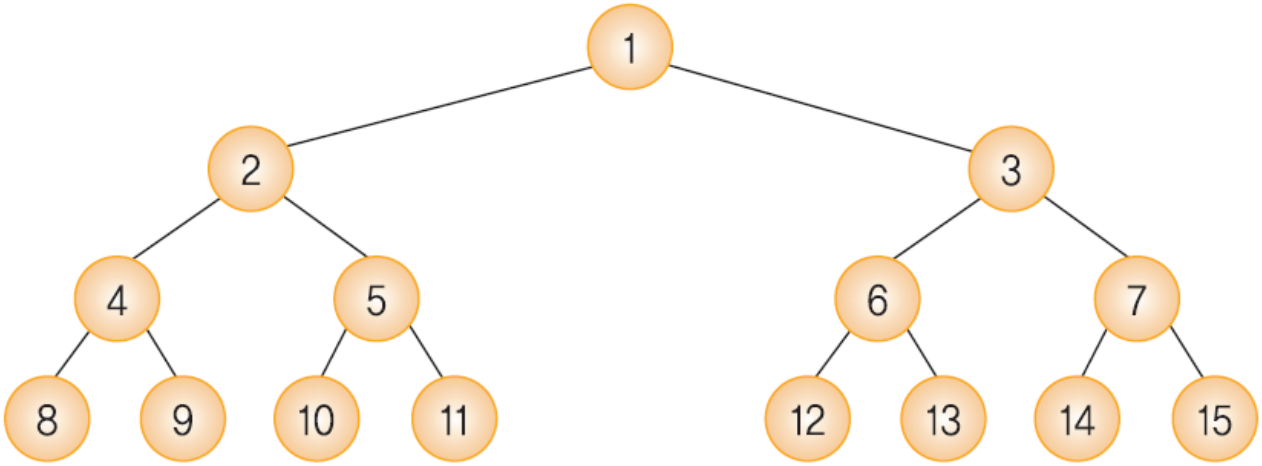
(a) 포화 이진 트리 (b) 완전 이진 트리 (c) 기타 이진 트리

포화 이진 트리

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

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

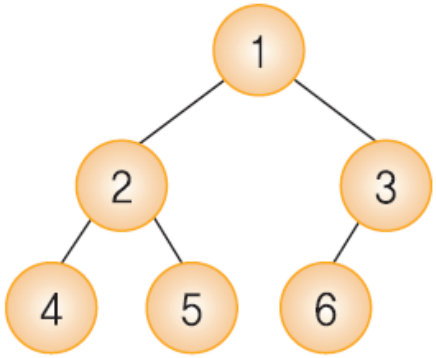
- 포화 이진 트리에는 다음과 같이 각 노드에 번호를 붙일 수 있다.



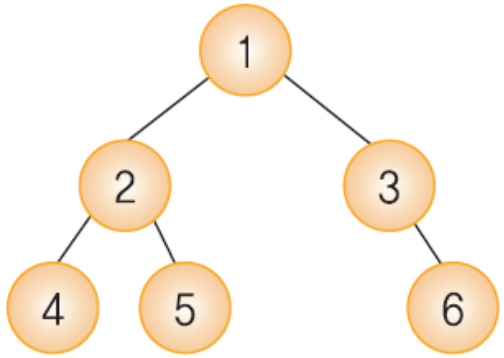
완전 이진 트리

- **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



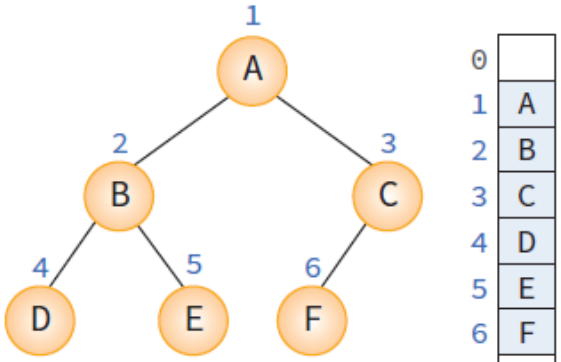
(b) 완전 이진 트리가 아님

8.3 이진 트리의 표현

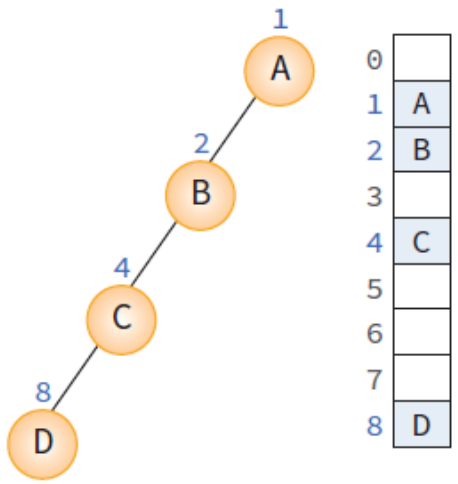
1) 배열을 이용하는 방법
2) 링크(포인터)를 이용하는 방법

배열 표현법

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



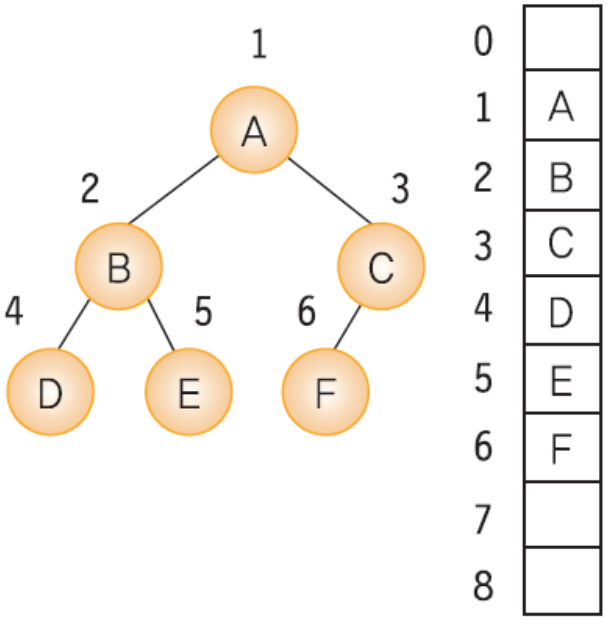
(a) 완전 이진트리



(b) 경사 이진트리

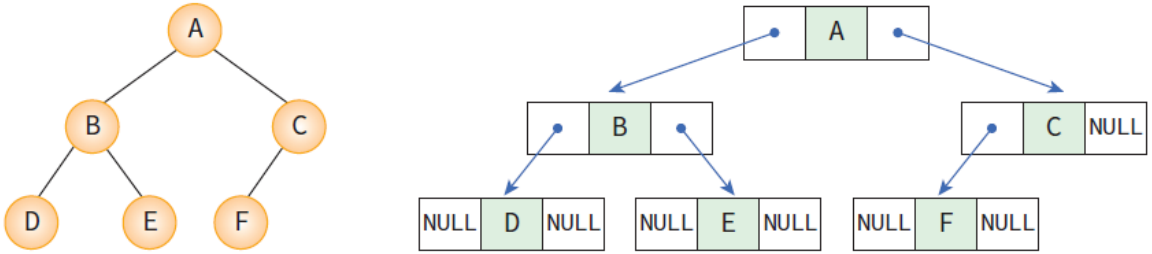
부모와 자식 인덱스 관계

- 노드 i의 부모 노드 인덱스 = i/2
- 노드 i의 왼쪽 자식 노드 인덱스 = 2i
- 노드 i의 오른쪽 자식 노드 인덱스 = 2i+1

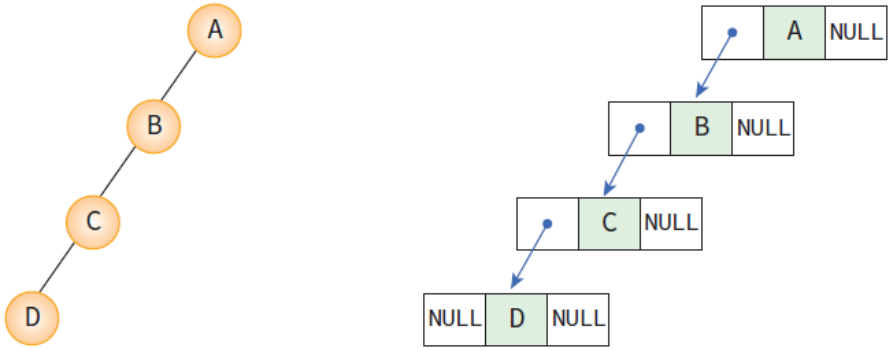


링크 표현법

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



(a) 완전 이진트리



(b) 경사 이진트리

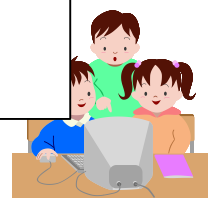
링크로 이진트리 구현

- 노드는 구조체로 표현
- 링크는 포인터로 표현

```
5  typedef struct TreeNode {
6      int data;
7      struct TreeNode *left, *right;
8  } TreeNode;
```

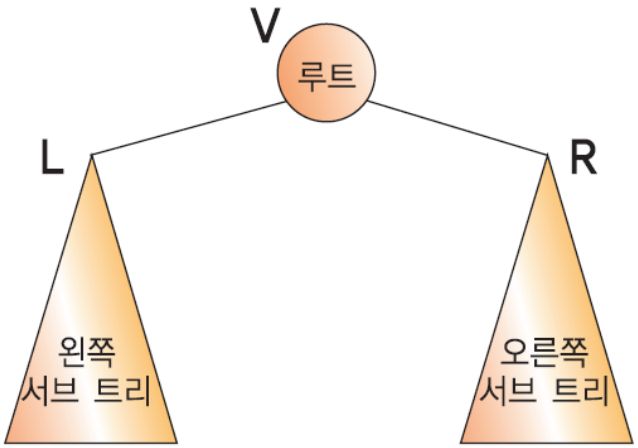
링크 표현법 이진트리 프로그램: tree1.c


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

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

//          n1
//         / |
//        n2  n3
int main(void)
{
    TreeNode *n1, *n2, *n3;
    n1 = (TreeNode *)malloc(sizeof(TreeNode));
    n2 = (TreeNode *)malloc(sizeof(TreeNode));
    n3 = (TreeNode *)malloc(sizeof(TreeNode));
    n1->data = 10;      // 첫 번째 노드를 설정한다.
    n1->left = n2;
    n1->right = n3;
    n2->data = 20;      // 두 번째 노드를 설정한다.
    n2->left = NULL;
    n2->right = NULL;
    n3->data = 30;      // 세 번째 노드를 설정한다.
    n3->left = NULL;
    n3->right = NULL;
    free(n1); free(n2); free(n3);
    return 0;
}
```



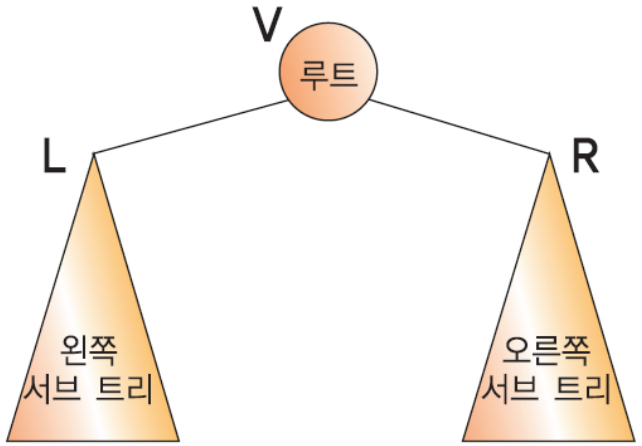
8.4 이진 트리의 순회

- 순회(traversal): 트리의 노드들을 체계적으로 방문하는 것



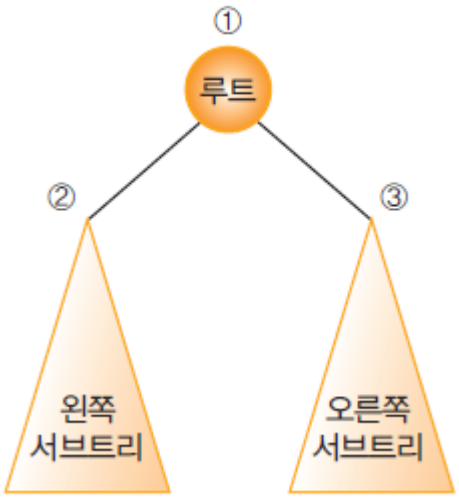
이진 트리의 순회 3가지 방법

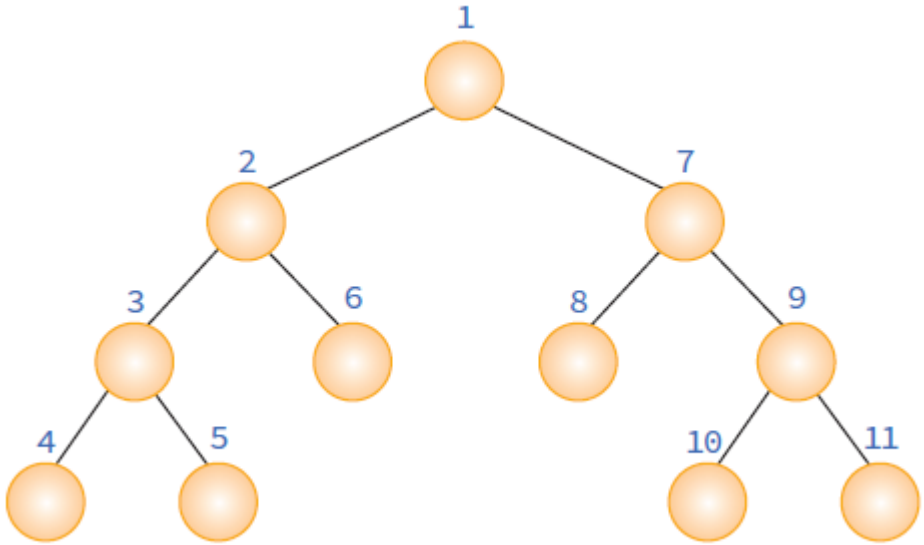
- 전위순회(preorder traversal) : VLR
  - 자손노드보다 루트노드를 먼저 방문한다.
- 중위순회(inorder traversal) : LVR
  - 왼쪽 자손, 루트, 오른쪽 자손 순으로 방문한다.
- 후위순회(postorder traversal) : LRV
  - 루트노드보다 자손을 먼저 방문한다.



전위 순회

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





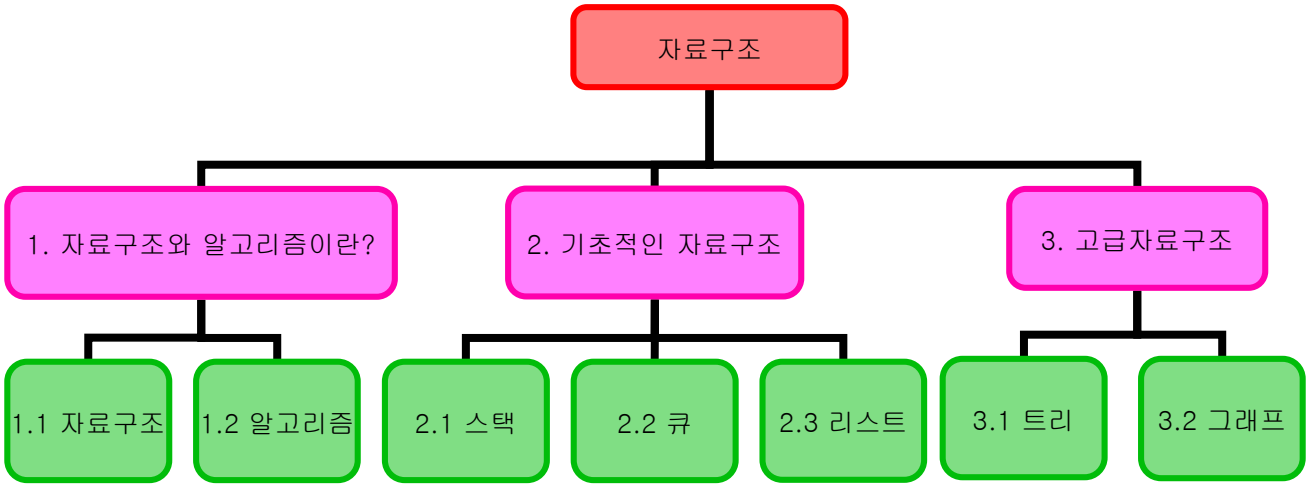
전위 순회 알고리즘

- 순환(재귀) 호출을 이용한다.

```
preorder(x)

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

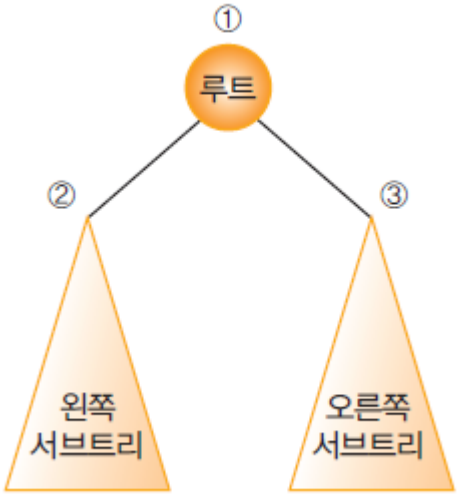
전위 순회 응용

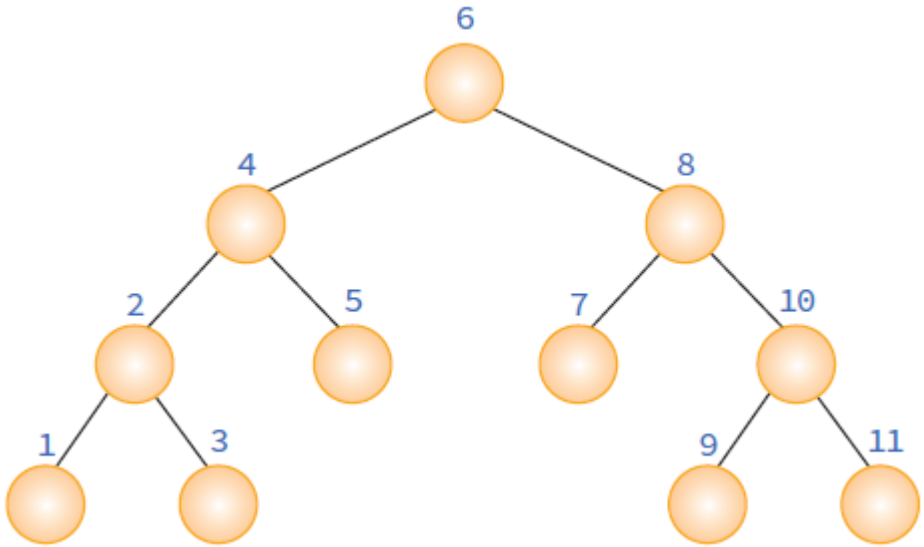
- (예) 구조화된 문서출력



중위 순회

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





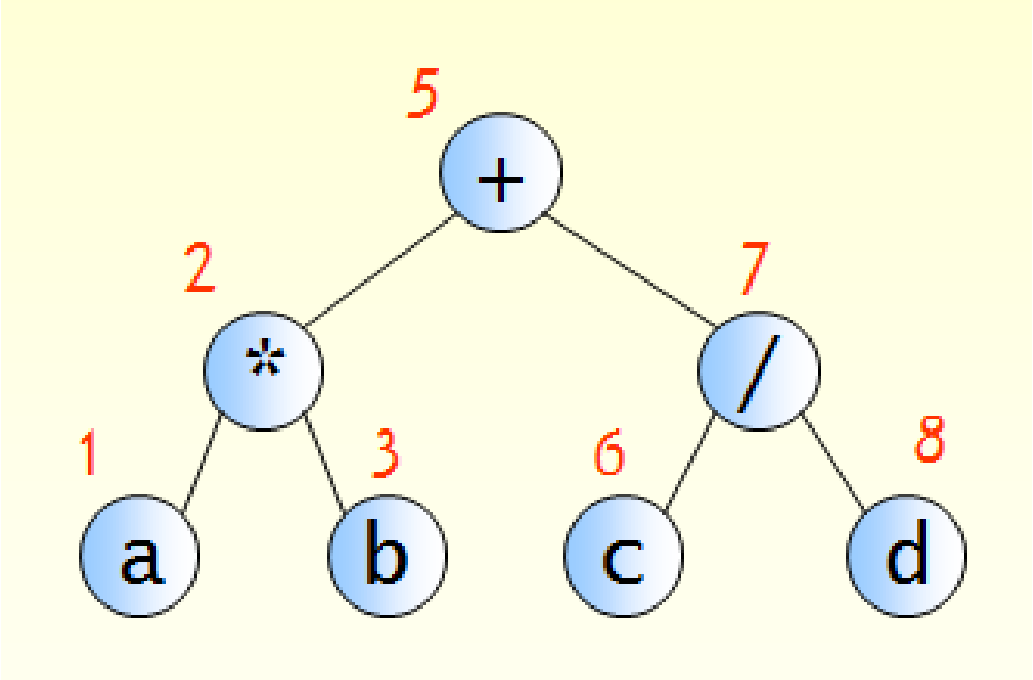
중위 순회 알고리즘

- 순환 호출을 이용한다.

```
inorder(x)

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

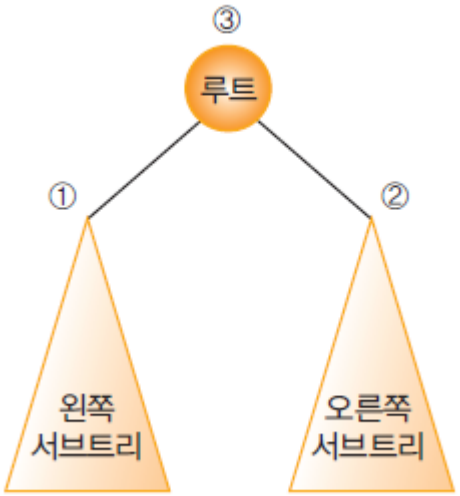
중위 순회 응용

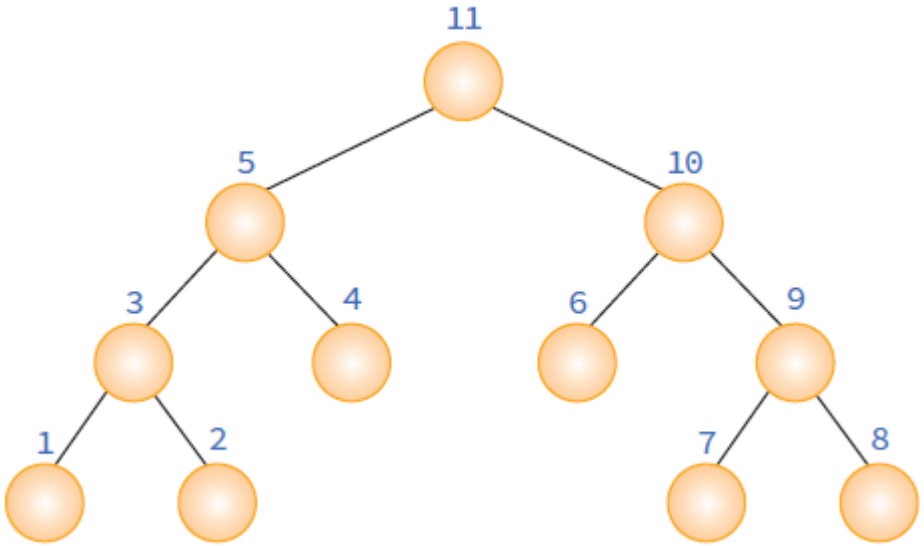
- (예) 수식 트리



후위 순회

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





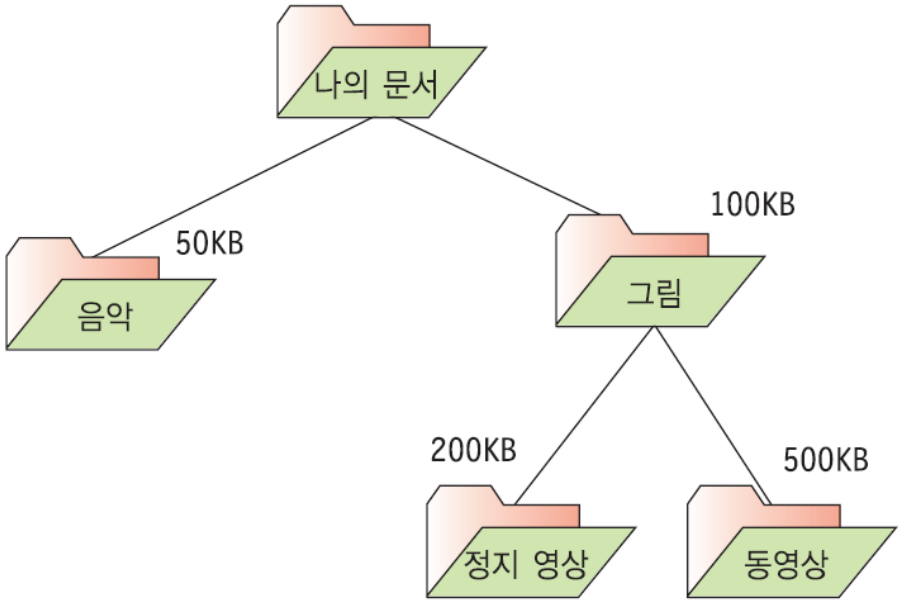
후위 순회 알고리즘

- 순환 호출을 이용한다.

```
postorder(x)

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

후위 순회 응용

- (예) 디렉토리 용량 계산



순회 프로그램: tree2.c (1/3)


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

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;
```

순회 프로그램: tree2.c (2/3)


```c
// 중위 순회
void inorder(TreeNode *root) {
    if (root != NULL) {
        inorder(root->left);// 왼쪽 서브트리 순회
        printf("[%d] ", root->data);  // 노드 방문
        inorder(root->right);// 오른쪽 서브트리 순회
    }
}

// 전위 순회
void preorder(TreeNode *root) {
    if (root != NULL) {
        printf("[%d] ", root->data);  // 노드 방문
        preorder(root->left);// 왼쪽 서브트리 순회
        preorder(root->right);// 오른쪽 서브트리 순회
    }
}

// 후위 순회
void postorder(TreeNode *root) {
    if (root != NULL) {
        postorder(root->left);// 왼쪽 서브트리 순회
        postorder(root->right);// 오른쪽 서브트리 순회
        printf("[%d] ", root->data);  // 노드 방문
    }
}
```

순회 프로그램: tree2.c (3/3)


```
48  int main(void)
49  {
50      printf("중 위  순 회 =");
51      inorder(root);
52      printf("\n");
53
54      printf("전 위  순 회 =");
55      preorder(root);
56      printf("\n");
57
58      printf("후 위  순 회 =");
59      postorder(root);
60      printf("\n");
61      return 0;
62  }
```

```
중위 순회=[1] [4] [15] [16] [20] [25]
전위 순회=[15] [4] [1] [20] [16] [25]
후위 순회=[1] [4] [16] [25] [20] [15]

--------------------------------
Process exited after 0.7944 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

8.5 반복적인 순회 (구현)

- 지금까지 순환(재귀)적인 순회를 구현하였음
- 순환을 사용하지 않고, 스택을 이용하여 일반적인 반복 방식의 순회도 구현이 가능함

반복적인 중위순회 프로그램: tree3.c (1/2)


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

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

#define SIZE 100
int top = -1;
TreeNode *stack[SIZE];

void push(TreeNode *p)
{
    if (top < SIZE - 1)
        stack[++top] = p;
}

TreeNode *pop()
{
    TreeNode *p = NULL;
    if (top >= 0)
        p = stack[top--];
    return p;
}
```

반복적인 중위순회 프로그램: tree3.c (2/2)

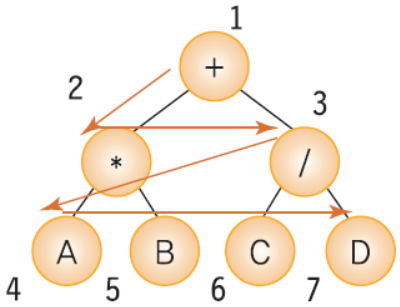

```c
void inorder_iter(TreeNode *root)
{
    while (1) {
        for (; root; root = root->left)
            push(root);
        root = pop();
        if (!root) break;
        printf("[%d] ", root->data);
        root = root->right;
    }
}

//         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;

int main(void)
{
    printf("중 위  순 회 =");
    inorder_iter(root);
    printf("\n");
    return 0;
}
```

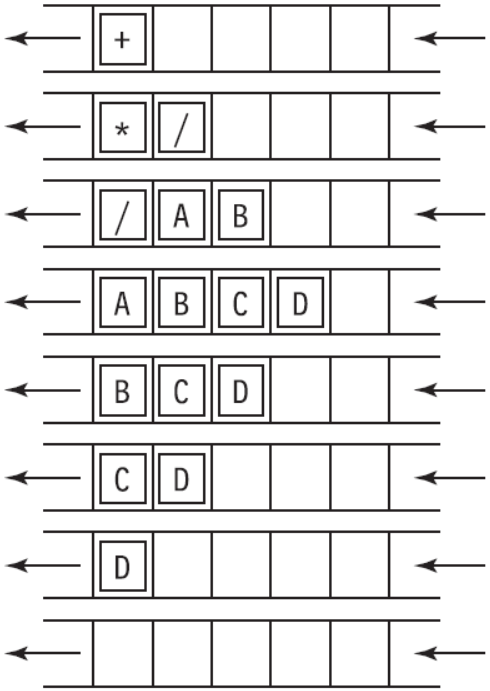
```
중위 순회=[1] [4] [15] [16] [20] [25]

--------------------------------
Process exited after 0.05907 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

8.6 레벨 순회

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

- 지금까지의 순회 법이 스택을 사용했던 것에 비해 레벨 순회는 큐를 사용하는 순회 법이다.





레벨 순회 알고리즘

- **level_order(root):**

```
1.  initialize queue;
2.  enqueue(queue, root);
3.  while is_empty(queue) ≠ TRUE do
4.        x← dequeue(queue);
5.        if (x ≠ NULL) then
6.              print DATA(x);
7.        enqueue(queue, LEFT(x));
8.        enqueue(queue, RIGHT(x));
```

레벨 순회 프로그램: tree4.c (1/4)


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

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

// ================ 원형 큐 코드 시작 ================
#define MAX_QUEUE_SIZE 100
typedef TreeNode * element;
typedef struct { // 큐 타입
    element  data[MAX_QUEUE_SIZE];
    int  front, rear;
} QueueType;

// 오류 함수:
void error(const char *message)
{
    fprintf(stderr, "%s\n", message);
    exit(1);
}
```

레벨 순회 프로그램: tree4.c (2/4)


```
25  // 공백 상태 검출 함수
26  void init_queue(QueueType *q)
27  {
28      q->front = q->rear = 0;
29  }
30
31  // 공백 상태 검출 함수
32  int is_empty(QueueType *q)
33  {
34      return (q->front == q->rear);
35  }
36
37  // 포화 상태 검출 함수
38  int is_full(QueueType *q)
39  {
40      return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
41  }
42
43  // 삽입 함수
44  void enqueue(QueueType *q, element item)
45  {
46      if (is_full(q))
47          error("큐가 포화상태입니다");
48      q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
49      q->data[q->rear] = item;
50  }
```

레벨 순회 프로그램: tree4.c (3/4)


```
52  // 삭제 함수
53  element dequeue(QueueType *q)
54  {
55      if (is_empty(q))
56          error("큐가 공백상태입니다");
57      q->front = (q->front + 1) % MAX_QUEUE_SIZE;
58      return q->data[q->front];
59  }
60
61  void level_order(TreeNode *ptr)
62  {
63      QueueType q;
64
65      init_queue(&q);  // 큐 초기화
66
67      if (ptr == NULL) return;
68      enqueue(&q, ptr);
69      while (!is_empty(&q)) {
70          ptr = dequeue(&q);
71          printf(" [%d] ", ptr->data);
72          if (ptr->left)
73              enqueue(&q, ptr->left);
74          if (ptr->right)
75              enqueue(&q, ptr->right);
76      }
77  }
```

레벨 순회 프로그램: tree4.c (4/4)


```
79  //         15
80  //     4          20
81  //   1         16     25
82  TreeNode n1 = { 1,  NULL, NULL };
83  TreeNode n2 = { 4,  &n1,  NULL };
84  TreeNode n3 = { 16, NULL, NULL };
85  TreeNode n4 = { 25, NULL, NULL };
86  TreeNode n5 = { 20, &n3,  &n4 };
87  TreeNode n6 = { 15, &n2,  &n5 };
88  TreeNode *root = &n6;
89
90  int main(void)
91  {
92      printf("레 벨  순 회 =");
93      level_order(root);
94      printf("\n");
95      return 0;
96  }
```

```
레벨 순회= [15]  [4]  [20]  [1]  [16]  [25]

--------------------------------
Process exited after 0.04772 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```