

7장 트리



순서

7.1 트리의 기본 개념

7.2 이진 트리

7.3 이진 트리의 표현

7.4 이진 트리 순회

7.5 이진 트리의 기타 주요 연산

7.6 스레드 이진 트리

7.7 일반 트리를 이진 트리로의 표현

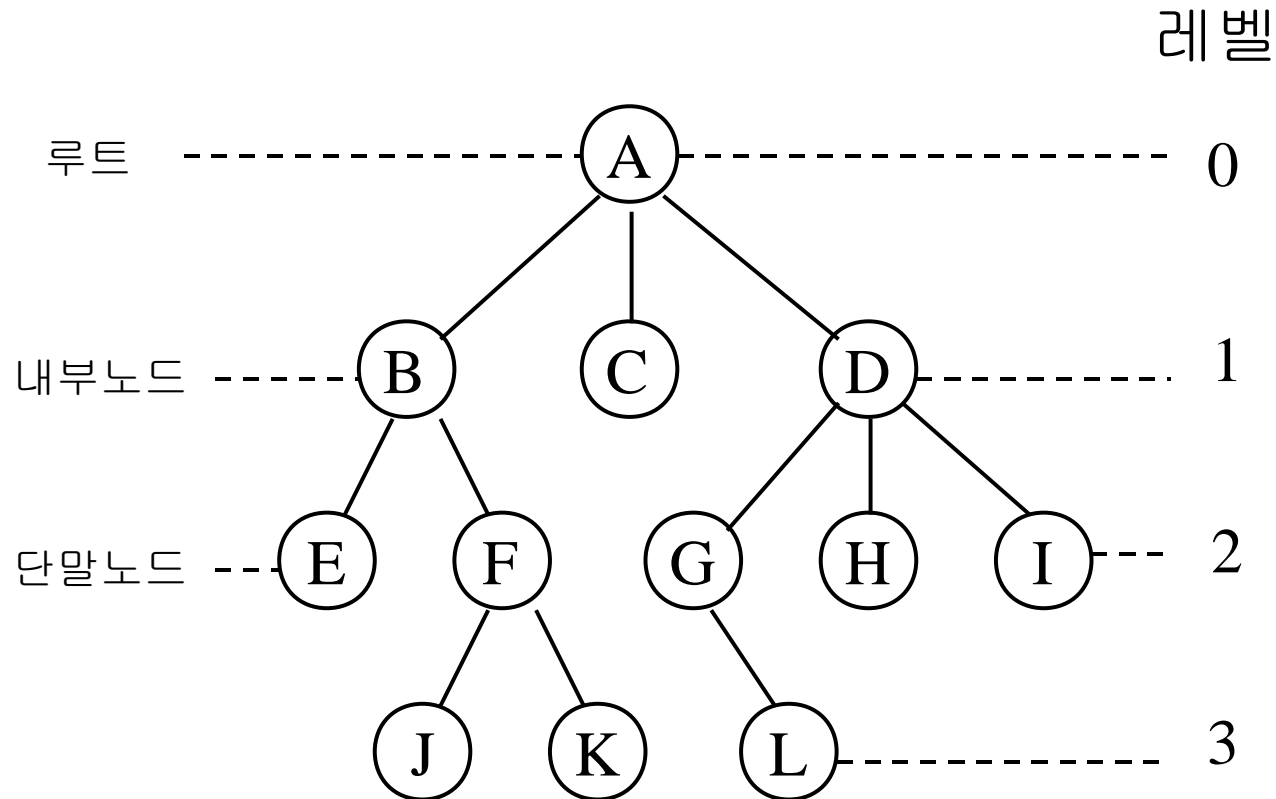
트리의 기본 개념 (1)

◆ 트리(tree)

- 계층형 자료구조 (hierarchical data structure)
 - ◆ 나무 형태로 노드들을 간선으로 연결
 - ◆ 하나의 루트 노드와 $n \geq 0$ 개의 분리집합 T_1, T_2, \dots, T_n 으로 분할
 - $T_i, 1 \leq i \leq n$ 는 트리로서 루트의 서브 트리
- 트리의 순환적 정의
 - ◆ 하나의 노드는 그 자체로 트리이면서 해당 트리의 루트가 됨
 - ◆ 만일 n 이 노드이고 T_1, T_2, \dots, T_k 가 n_1, n_2, \dots, n_k 를 루트로 갖는 트리라고 할 때
 - n 을 부모로 n_1, n_2, \dots, n_k 를 연결 – 새로운 트리 생성
 - n 은 루트(root)
 - T_1, T_2, \dots, T_k 는 루트 n 의 서브트리(subtree)
 - 노드 n_1, n_2, \dots, n_k 는 노드 n 의 자식들(children)

트리의 기본 개념 (2)

◆ 트리의 예: 트리 T



트리의 기본 개념 (3)

- ◆ **노드(node)** : 데이터와 링크를 통합적으로 표현
- ◆ **노드의 차수(degree)**
 - 한 노드가 가지고 있는 서브 트리의 수
 - ◆ A의 차수: 3, B의 차수: 2, C의 차수: 0
 - 리프(leaf), 단말 또는 터미널(terminal) 노드
 - ◆ 차수가 0인 노드
 - 비단말(nonterminal) 노드
 - ◆ 차수가 1 이상인 노드
- ◆ **노드의 parent/children 구조**
 - 자식(children) : 노드 x의 서브 트리 루트들
 - 부모(parent) : 노드 x
 - ◆ 노드 D의 자식들: G, H, I
 - ◆ D의 부모: A

트리의 기본 개념 (4)

◆ 형제(siblings)

- 한 부모의 자식들
 - ◆ 노드 G, H, I는 형제들

◆ 트리의 차수

- 그 트리에 있는 노드의 최대 차수
 - ◆ 트리 T의 차수 : 3

트리의 기본 개념 (5)

◆ 노드의 레벨(level)

- 루트 : 0
- 한 노드가 레벨 l 에 속하면, 그 자식들은 레벨 $l+1$ 에 속함
 - ◆ level(v)
 - if ($v = \text{root}$) then return 0;
 - else return (1 + level(v 의 부모));
 - end level()

◆ 트리의 높이(height) 또는 깊이(depth)

- 그 트리의 최대 레벨
 - ◆ 트리 T 의 높이 : 3

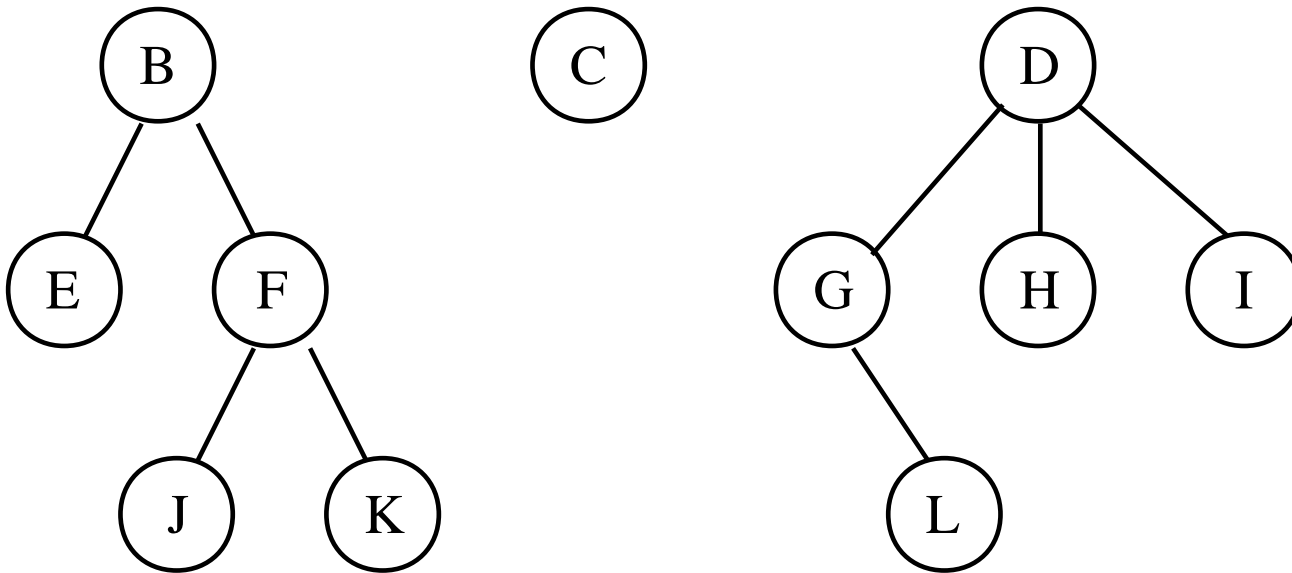
◆ 노드의 레벨순서(level order)

- 트리의 노드들에 레벨별로 위에서 아래로,
 - ◆ 같은 레벨 안에서는 왼쪽에서 오른쪽으로 차례로 순서를 매긴 것

트리의 기본 개념 (6)

◆ 포리스트(forest)

- $n \geq 0$ 개의 분리된 트리의 집합
 - ◆ 트리 T에서 루트 A를 제거한 결과로 얻은 포리스트



트리의 기본 개념 (7)

◆ 트리의 기술 방법

- 리스트(list) 표현

- ◆ 각 서브 트리를 또 다시 리스트로 표현

- (A(B(E, F(J, K)), C, D(G(L), H, I)))

- ◆ 메모리 표현 (연결 리스트 표현)

- 노드 A:

데이터	링크 1	링크 2	링크 3
-----	------	------	------

- 노드 B:

데이터	링크 1	링크 2
-----	------	------

- ◆ 효율적인 알고리즘 작성을 위해서는 노드 구조가 일정한 것이 좋음

이진 트리 (1) – Binary Tree

◆ 특징

- 컴퓨터 응용에서 가장 많이 사용하는 아주 중요한 트리 구조
- 모든 노드가 정확하게 두 서브트리를 가지고 있는 트리
 - ◆ 서브트리는 공백이 될 수 있음
 - ◆ 리프 노드 (leaf node) : 두 공백 서브트리를 가지고 있는 노드
- 왼쪽 서브 트리과 오른쪽 서브 트리를 분명하게 구별
- 이진트리 자체가 노드가 없는 공백이 될 수 있음

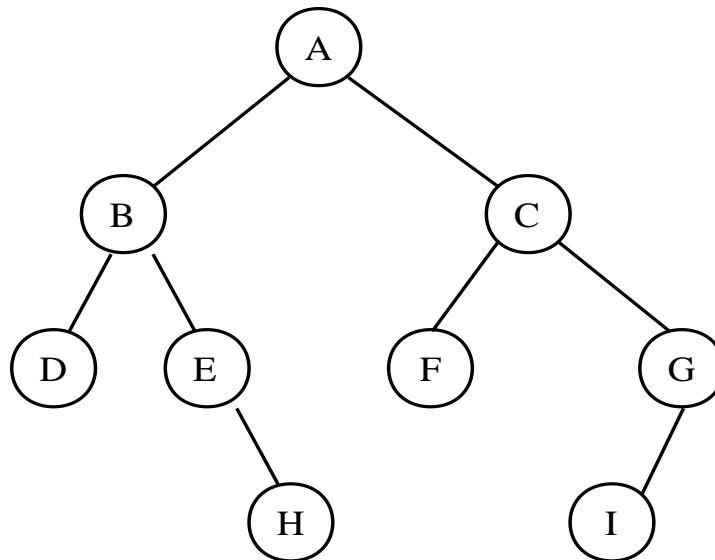
◆ 정의 : 이진 트리(binary tree: BT)

- 노드의 유한집합
- 공백이거나 루트와 두 개의 분리된 이진 트리인 왼쪽 서브 트리와 오른쪽 서브 트리로 구성

이진 트리 (2) – Binary Tree

◆ 이진 트리 예

- 리프노드 (leaf node) : D, H, F, I
- 노드 E와 G : 공백이 아닌 서브트리를 하나씩 가지고 있음
- 노드 E : 공백 왼쪽 서브트리와 오른쪽 서브트리 H를 가지고 있음
- 노드 G : 왼쪽 서브트리 I와 공백 오른쪽 서브트리를 가지고 있음



이진 트리 (3) – Binary Tree

ADT BinaryTree

데이타 : 노드의 유한집합으로 공백이거나 루트 노드, 왼쪽 서브 트리, 오른쪽 서브 트리로 구성

연산 :

```
BT, BT1, BT2 ∈ BinaryTree; item ∈ Element;
```

```
createBT() ::= create an empty binary tree;
```

```
isEmpty(BT) ::= if BT = empty then return true
                else return false;
```

```
makeBT(BT1, item, BT2) ::= return a binary tree whose root contains item,  
                           left subtree is BT1, and right subtree is BT2;
```

```
leftSubtree(BT) ::= if isEmpty(BT) then return null
                    else return left subtree of BT;
```

```
data(BT) ::= if isEmpty(BT) then return null
           else return the item in the root node of BT;
```

```
rightSubtree(BT) ::= if isEmpty(BT) then return null
                    else return right subtree of BT;
```

End BinaryTree

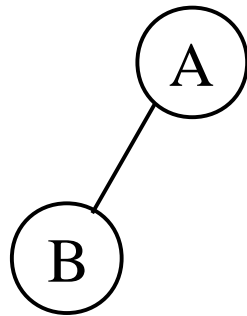


three adj.java

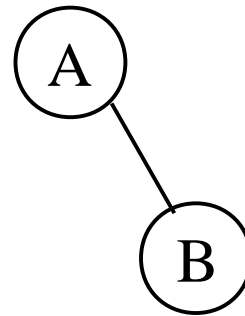
이진 트리 (4) – Binary Tree

◆ 이진 트리와 일반 트리의 차이점

- 이진 트리에는 공백 이진 트리가 있지만, 일반 트리에는 공백 트리가 없음
- 이진 트리에서는 서브 트리의 순서를 구분
 - ◆ 서로 다른 이진 트리 예



(a)

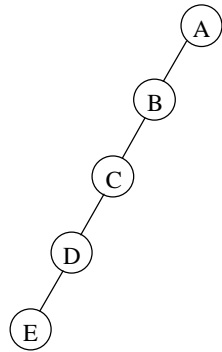


(b)

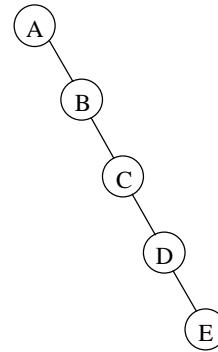
이진 트리 (5) – Binary Tree

◆ 편향 이진 트리(skewed binary tree)

- 왼쪽으로 편향
- 오른쪽으로 편향



(a)



(b)

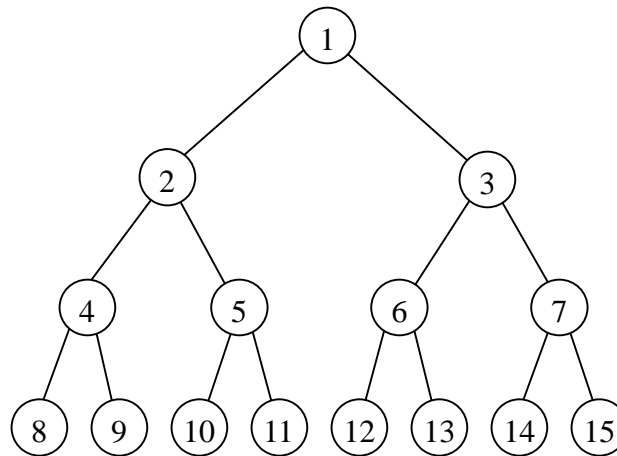
◆ 이진 트리의 주요 성질

- 레벨 $i (i \geq 0)$ 의 최대 노드 수: 2^i
- 높이가 $h (h \geq 0)$ 인 이진 트리의 최대 노드수 : $2^{h+1}-1$

이진 트리 (6) – Binary Tree

◆ 포화 이진 트리(full binary tree)

- 높이가 h 이고 노드수가 $2^{h+1}-1$ 인 이진 트리
- 높이가 3인 포화 이진 트리의 예



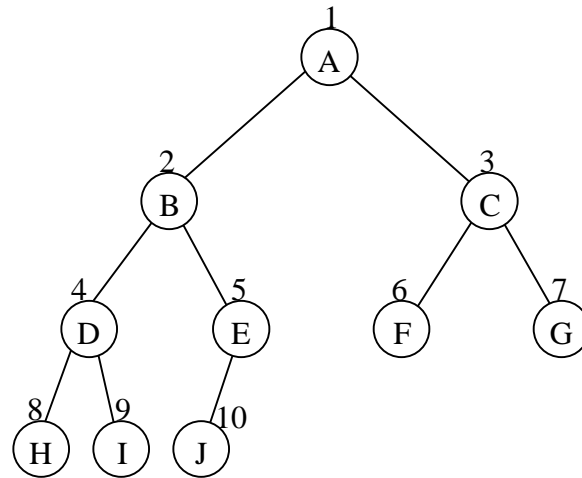
높이가 3인 포화 이진트리

- 포화 이진 트리 번호(full binary tree number) : 루트 노드를 1번으로 하고 레벨 별로 왼쪽에서 오른쪽으로 차례로 노드 위치에 번호를 $2^{h+1}-1$ 까지 유일하게 부여한 것

이진 트리 (7) – Binary Tree

◆ 완전 이진 트리 (complete binary tree)

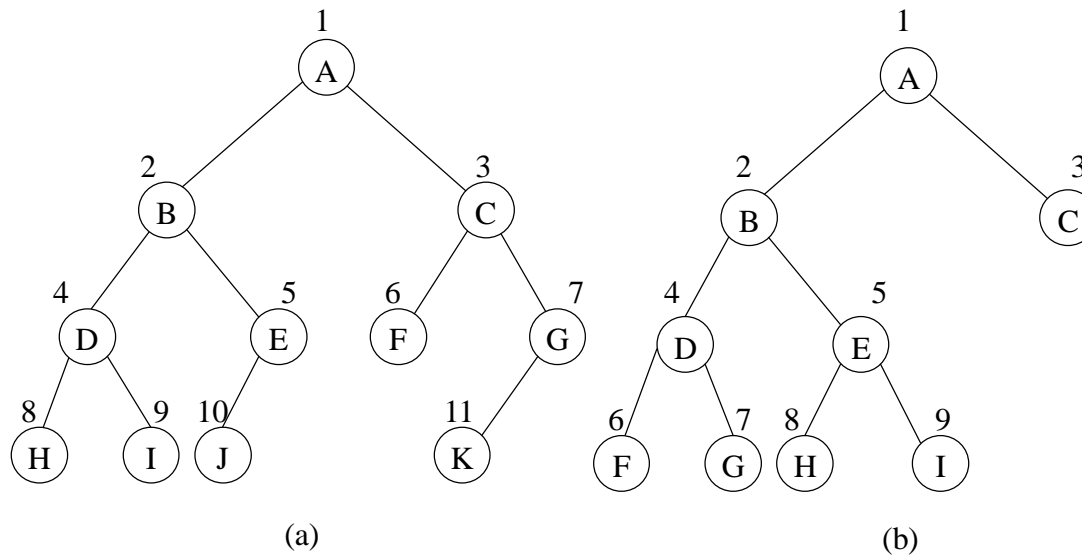
- 높이가 h 이고 노드수가 n 인 이진 트리에서 노드의 레벨순서 번호들의 각 위치가 높이가 h 인 포화 이진 트리 번호 1에서 n 까지 모두 일치하는 트리



완전 이진 트리

이진 트리 (8) – Binary Tree

- 완전 이진 트리가 아닌 이진 트리의 예

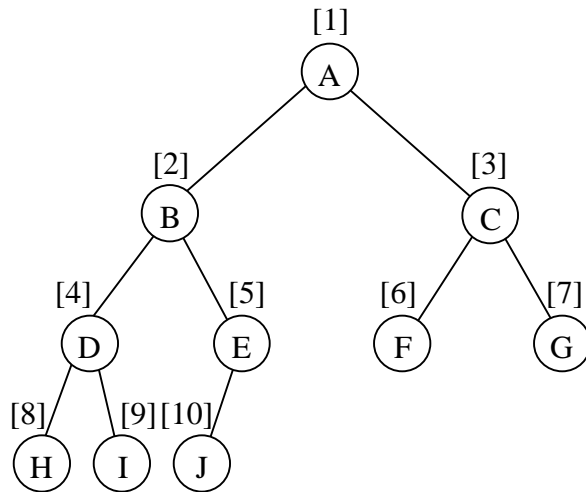


완전 이진 트리가 아닌 이진 트리

이진 트리의 표현 (1)

◆ 일차원 배열 표현

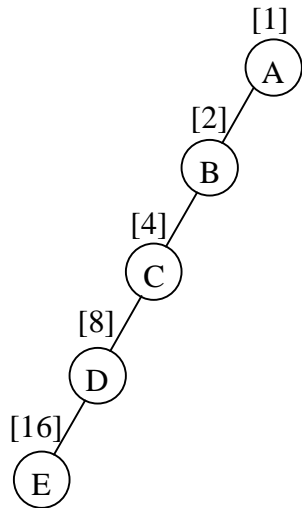
- 포화 이진 트리 번호를 배열의 인덱스로 사용
- 이진 트리의 순차표현
 - ◆ 인덱스 0: 실제로 사용하지 않음
 - ◆ 인덱스 1: 항상 루트 노드
- 완전 이진 트리(T)의 순차 표현 예



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

이진 트리의 표현 (2)

- 편향 이진 트리(S)의 순차 표현 예



S	
[0]	-
[1]	A
[2]	B
[3]	-
[4]	C
[5]	-
[6]	-
[7]	-
[8]	D
⋮	-
⋮	...
⋮	-
[16]	E

- 완전 이진 트리의 1차원 배열 표현에서의 인덱스 관계

목표 노드	인덱스 값	조 건
노드 i의 부모	$\lfloor i/2 \rfloor$	$i > 1$
노드 i의 왼쪽 자식	$2*i$	$2*i \leq n$
노드 i의 오른쪽 자식	$2*i + 1$	$(2*i + 1) \leq n$
루트 노드	1	$0 < n$

이진 트리의 표현 (3)

- 배열 T에 대한 인덱스 관계 적용 예
 - ◆ 노드 I의 부모: 배열 T에서 $I=T[9]$, 그 부모는 $T[\lfloor 9/2 \rfloor]=T[4]$ 에 위치
 - ◆ 노드 C의 오른쪽 자식: $C=T[3]$, 그 오른쪽 자식은 $T[3*2+1]=T[7]$ 에 위치
 - ◆ 노드 C의 왼쪽 자식: $T[3*2]=T[6]$ 에 위치
- 순차 표현의 장단점
 - ◆ 장점
 - 어떤 이진 트리에도 사용 가능
 - 완전 이진 트리: 최적
 - ◆ 단점
 - 편향 이진 트리: 배열 공간을 절반도 사용하지 못할 수 있음
→ 높이가 k인 편향 이진 트리: $2^{k+1}-1$ 개의 공간이 요구될 때 k+1만 실제 사용
 - 트리의 중간에 노드의 삭제나 삽입시: 많은 다른 노드들의 이동 불가피

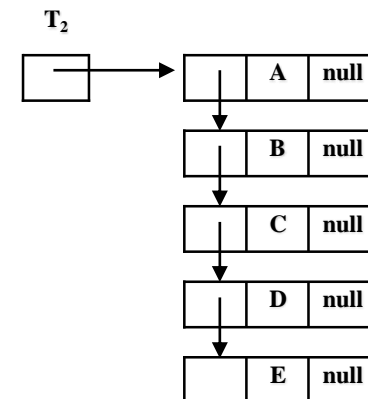
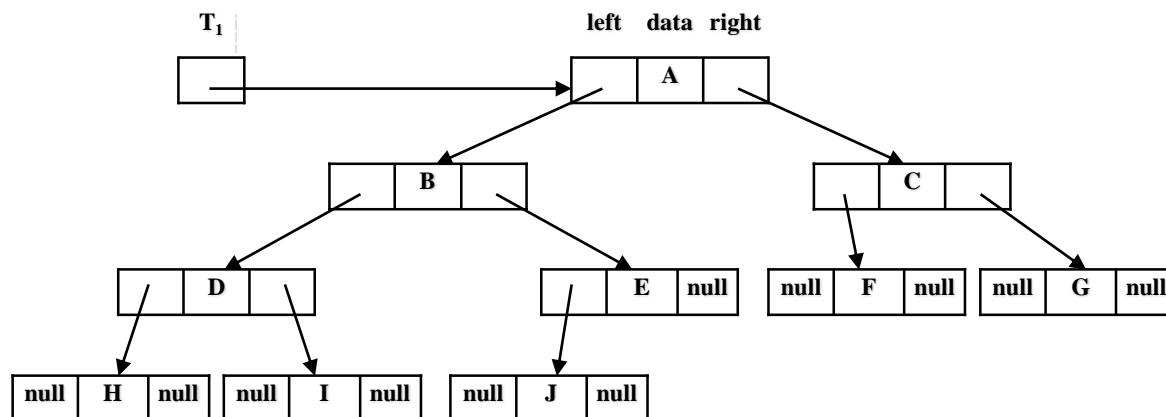
이진 트리의 표현 (4)

◆ 연결 리스트 표현

- 각 노드를 3개의 필드 left, data, right로 구성

left	data	right
------	------	-------

- ◆ left와 right : 각각 왼쪽 서브트리와 오른쪽 서브트리를 가리키는 링크
- ◆ 필요시 부모를 가리키는 parent 필드 추가
- 완전 이진 트리와 편향 이진 트리의 연결 표현



이진 트리의 표현 (5)

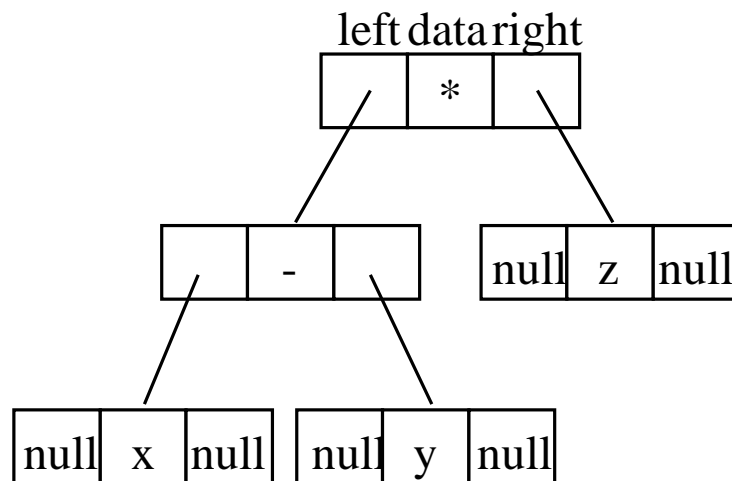
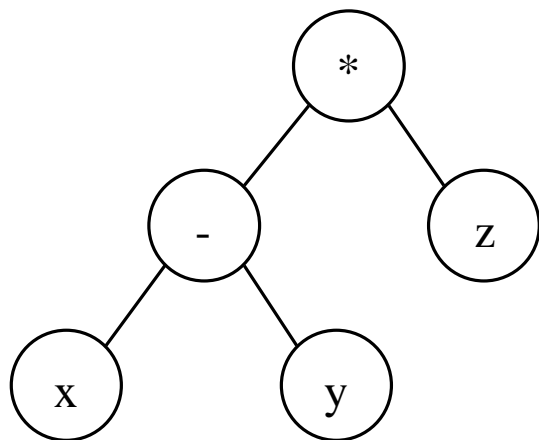
- 이진 트리 노드의 선언을 위한 Java 클래스 정의

```
class TreeNode{  
    Object data;  
    TreeNode left;  
    TreeNode right;  
}
```

이진 트리의 표현 (6)

◆ 수식 이진 트리(expression binary tree)

- 수식을 이진 트리로 표현
- 수식 $((x - y) * z)$ 를 표현하는 수식 이진 트리의 연결 리스트 표현 예



이진 트리 순회 (1)

◆ 순회의 개요

- 이진 트리에서 각 노드를 차례로 방문
- n 개 노드시 노드들의 가능한 순회 순서: $n!$
- 한 노드에서 취할 수 있는 조치
 - ◆ 왼쪽 서브 트리로의 이동 (L)
 - ◆ 현재의 노드 방문 (D)
 - ◆ 오른쪽 서브 트리로의 이동 (R)
- 한 노드에서 취할 수 있는 순회 방법
 - ◆ LDR, LRD, DLR, RDL, RLD, DRL 등 6가지
 - ◆ 왼편을 항상 먼저 순회한다고 가정시 : LDR, LRD, DLR 3가지
→ 이것을 중위(inorder), 후위(postorder), 그리고 전위(preorder) 순회라 함 : 데이터 필드의 위치기준
 - ◆ 산술식 표현 이진 트리의 중위, 후위, 전위 순회 결과
→ 각각 중위 표기식, 후위 표기식, 전위 표기식이 됨

이진 트리 순회 (2)

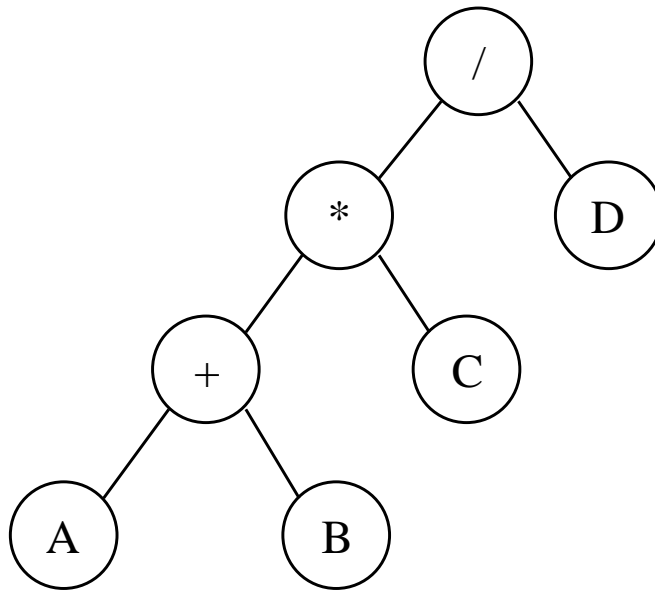
◆ 중위 순회 (inorder traversal)

- 중위 순회 방법의 순환식 기술
 - i) 왼편 서브 트리(left subtree)를 중위 순회한다.
 - ii) 루트 노드(root node)를 방문한다.
 - iii) 오른편 서브 트리(right subtree)를 중위 순회한다.
- 순환 함수 표현

```
inorder(T)
  if (T ≠ null) then {
    inorder(T.left);
    visit T.data;
                                inorder(T.right);
                                }
end inorder()
```

이진 트리 순회 (3)

- 수식 이진 트리의 예



- 수식 이진 트리의 중위 순회 결과: $A+B*C/D$

이진 트리 순회 (4)

◆ 후위 순회 (postorder traversal)

- 후위 순회 방법의 순환식 기술
 - i) 왼쪽 서브 트리(left subtree)를 후위 순회한다.
 - ii) 오른쪽 서브 트리(right subtree)를 후위 순회한다.
 - iii) 루트 노드(root node)를 방문한다.
- 순환 함수 표현

```
postorder(T)
if (T ≠ null) then {
    postorder(T.left);
    postorder(T.right);
    visit T.data;
}
end postorder()
```

- 수식 이진 트리의 후위 순회 결과: $AB+C*D/$

이진 트리 순회 (5)

◆ 전위 순회 (preorder traversal)

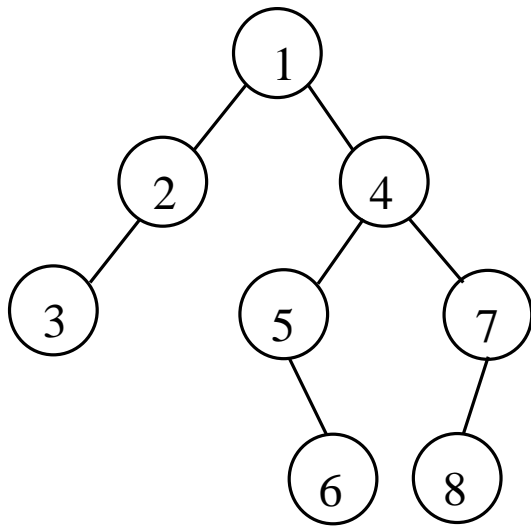
- 전위 순회 방법의 순환식 기술
 - i) 루트 노드(root node)를 방문한다.
 - ii) 왼편 서브 트리(left subtree)를 전위 순회한다.
 - iii) 오른편 서브 트리(right subtree)를 전위 순회한다.
- 순환 함수 표현

```
preorder(T)
if (T ≠ null) then {
    visit T.data;
    preorder(T.left);
    preorder(T.right);
}
end preorder()
```

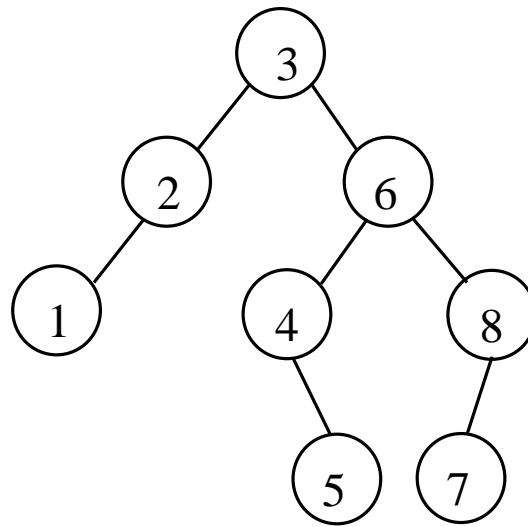
- 수식 이진 트리의 전위 순회 결과: /*+ABCD

이진 트리 순회 (6)

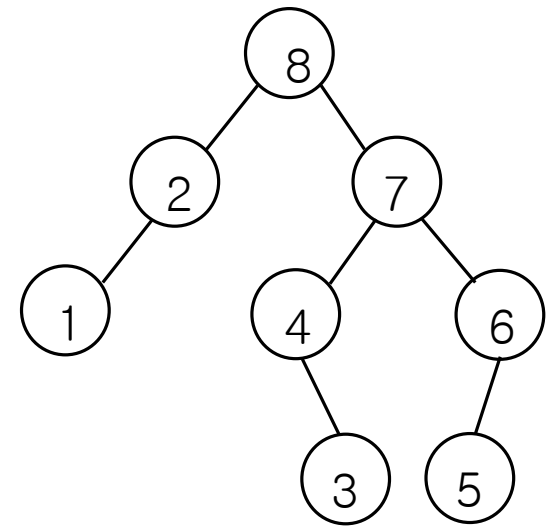
◆ 전위, 중위, 후위 순회 방법 경로의 종합적인 표현



(a) 전위 순회



(b) 중위 순회



(c) 후위 순회

이진 트리 순회 (7)

◆ 이진 트리 순회에 대한 비순환 알고리즘

- 순환 호출 해결을 위해 스택을 사용
- 비순환 중위 이진 트리 순회 알고리즘

```
iterInorder(T)
initialize stack; //스택을 초기화
p ← T;
if (p ≠ null) then push(stack, p); //루트를 삽입
while (not(isEmpty(stack))) do {
    if (p ≠ null) then {
        p ← peek(stack);
        p ← p.left; //왼쪽 서브트리 순회
        while (p ≠ null) do{
            push(stack, p);
            p ← p.left;
        }
    }
    p ← pop(stack);
    visit p.data;
    p ← p.right; //오른쪽 서브트리 순회
    if (p ≠ null) then push(stack, p);
}
end iterInorder()
```

이진 트리 순회 (8)

◆ 레벨 순서 이진트리순회(level order binary tree traversal)

- 이진 트리를 노드의 레벨 순서로 순회
- 큐를 사용
- 알고리즘 개요
 - ◆ 먼저 루트노드를 큐에 삽입하는 것으로 시작
 - ◆ 순회는 큐에 있는 노드를 삭제하며 널이 아니면 그 노드를 방문하고
 - ◆ 그의 자식들을 순서대로 큐에 삽입
 - ◆ 이 과정을 큐가 공백이 될 때까지 반복
- 레벨 순서 이진트리 순회 알고리즘

```
levelorder(T)
  initialize queue; // 큐를 초기화
  enqueue(queue,T);
  while (not (isEmpty(queue))) do {
    p←dequeue(queue);
    if (p≠null) then {
      visit p.data;
      enqueue(queue, p.left);
      enqueue(queue, p.right);
    }
  }
end levelorder()
```

이진 트리의 기타 주요 연산 (1)

◆ 주어진 이진 트리의 복사

```
copy(T)
  S ← null;           //공백 이진 트리 초기화.
  if (T ≠ null) then {
    L ← copy(T.left);  // 왼쪽 서브 트리 복사
    R ← copy(T.right); // 오른쪽 서브 트리 복사
    S ← getnode();     // 노드는 data, left, right 필드로 구성
    S.left ← L;        // s의 필드에 복사
    S.right ← R;
    S.data ← T.data;
  }
  return S;
end copy()
```


이진 트리의 기타 주요 연산 (2)

◆ 주어진 두 개의 이진 트리의 동등성 결정

```
equal(S, T)
  ans ← false;
  case {
    S = null and T = null :
      ans ← true;
    S ≠ null and T ≠ null :
      if (S.data = T.data) then {
        ans ← equal(S.left, T.left);
        if ans then
          ans ← equal(S.right, T.right);
      }
  }
  return ans;
end equal()
```



tree_adj_1.java

스레드 이진 트리 (1)

◆ 연결 리스트 표현 이진 트리의 문제점

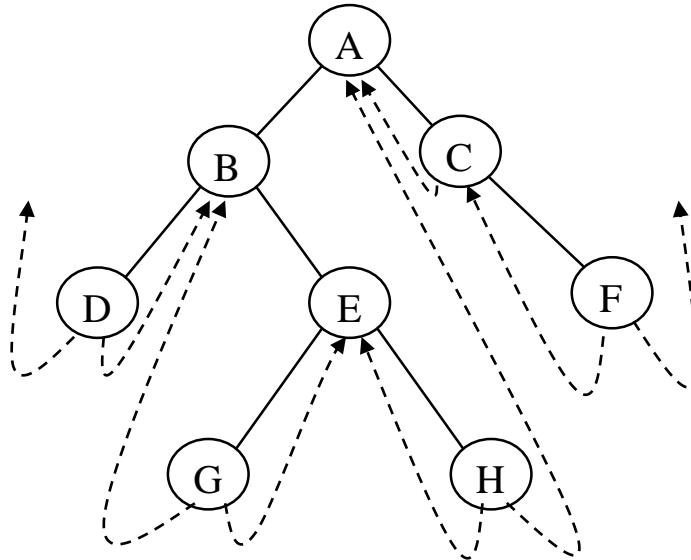
- 실제로 사용하는 링크수보다 사용하지 않는 널(null)링크가 더 많음
 - ◆ n 개의 노드를 가진 이진 트리의 총 링크수: $2n$ 개
 - ◆ 실제 사용되는 링크수: $n-1$ 개
 - ◆ 널 링크수: $n+1$ 개

◆ 스레드 이진 트리(threaded binary tree)

- 널 링크들을 낭비하지 않고 스레드(thread)를 저장해 활용
- 스레드 : 트리의 어떤 다른 노드에 대한 포인터
 - ◆ 트리를 순회하는 정보로 활용
- 스레드 이진 트리 생성 방법
 - ◆ 노드 p 의 right가 널 : 중위 순회에서 중위 후속자에 대한 포인터 저장
 - ◆ 노드 p 의 left가 널 : 중위 순회에서 중위 선행자에 대한 포인터 저장

스레드 이진 트리 (2)

- 점선으로 스레드를 표현한 스레드 이진 트리(TBT)의 예



- ◆ TBT는 9개의 널링크를 스레드로 대체 사용
- ◆ 노드 H : left 필드에 중위 선행자 E에 대한 스레드, right 필드에 중위 후속자 A에 대한 스레드 가짐

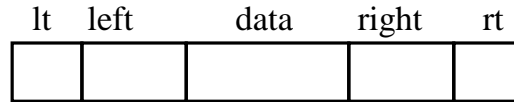
스레드 이진 트리 (3)

- ◆ 스레드와 보통의 널 포인터 구별: 두개의 불리언 필드
lt(왼쪽 스레드)와 rt(오른쪽 스레드) 사용

$$\begin{aligned} p.lt &= \begin{cases} \text{true} : P.left \text{는 중위 선행자에 대한 스레드} \\ \text{false} : P.left \text{는 왼쪽 자식을 가리키는 포인터} \end{cases} \\ p.rt &= \begin{cases} \text{true} : P.right \text{는 중위 후속자에 대한 스레드} \\ \text{false} : P.right \text{는 오른쪽 자식을 가리키는 포인터} \end{cases} \end{aligned}$$

스레드 이진 트리 (4)

- 노드 구조

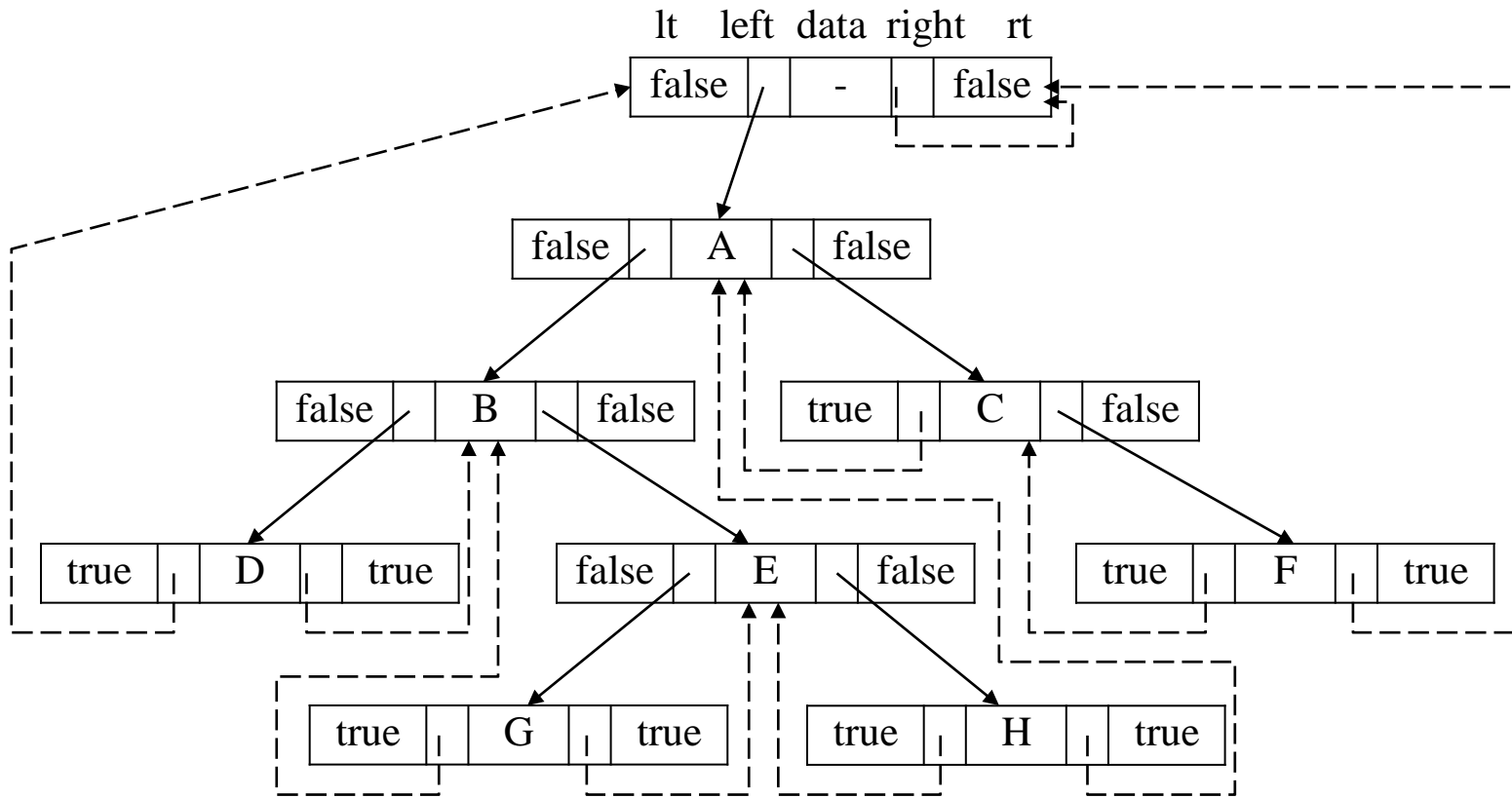


- 헤더 노드

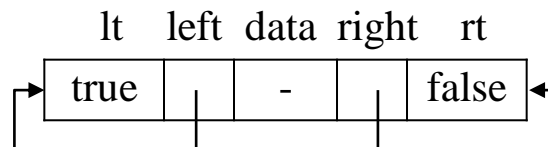
- ◆ D.lt와 F.rt의 스레드가 매달려 있는 경우 해결
- ◆ 다른 노드 구조와 동일
- ◆ data필드는 다른 용도로 사용
- ◆ 스레드 이진 트리 TBT를 헤더 노드의 왼쪽 서브 트리으로 표현

스레드 이진 트리 (5)

- 헤더노드를 가진 TBT의 예



- 이진 트리가 공백인 경우의 공백 스레드 이진 트리에 대한 헤더 노드



스레드 이진 트리 (6)

◆ 스레드 이진 트리에 대한 중위 순회 알고리즘

- 알고리즘 개요
 - ◆ 임의의 노드 p 에서 $p.rt = t$ 이면,
 - p 의 중위 후속자는 스레드 정의에 의하여 $p.right$ 가 됨
 - ◆ $p.rt = false$ 이면,
 - p 의 오른쪽 자식의 왼쪽 링크만을 계속 따라 내려가다가 $lt = true$ 인 노드를 만나면 바로 이 노드가 p 의 중위 후속자가 됨
- 임의의 노드 p 에 대한 중위 후속자 탐색 알고리즘

```
inorderSuccessor(p)
```

```
    // 중위 스레드 이진 트리에서  $p$ 의 중위 후속자를 반환
```

```
     $q \leftarrow p.right;$ 
```

```
    if ( $p.rt = false$ ) then
```

```
        while ( $q.lt = false$ ) do
```

```
             $q \leftarrow q.left;$ 
```

```
        return  $q$ ;
```

```
    end inorderSuccessor()
```

스레드 이진 트리 (7)

- 중위 스레드 이진 트리의 순회 알고리즘
 - ◆ 알고리즘 `inorderSuccessor`를 반복적으로 계속 호출

```
inorderThread(p)
// 중위 스레드 이진트리를 중위 순회
  q ← inorderSuccessor(p);
  while (q ≠ p) do {
    visit q.data;
    q ← inorderSuccessor(q);
  }
end inorderThread()
```

- 노드 `p`의 중위 선행자를 찾아내는 알고리즘
 - ◆ 중위 후속자를 찾아내는 알고리즘에서 `right`와 `left`의 기능을 서로 바꿔줌

```
inorderPredecessor(p)
// 중위 스레드 이진트리에서 중위 선행자를 반환
  q ← p.left;
  if (p.lt = false) then
    while (q.rt = false) do
      q ← q.right;
    return q;
  end inorderPredecessor()
```


스레드 이진 트리 (8)

◆ 스레드 이진 트리에 대한 전위 순회 알고리즘

● 개요

- ◆ 먼저 전위 후속자를 반환 알고리즘 preorderSuccessor를 작성
- ◆ 전위 순회하는 알고리즘 preorderThread를 작성

● 전위 후속자 반환 알고리즘

```
preorderSuccessor(p)
// 중위 스레드 이진트리에서 p의 전위 후속자를 반환
if (p.lt = false) then return p.left;
else {
    q ← p;
    while (q.rt = true) do
        q ← q.right;
    return q.right;
}
end preorderSuccessor()
```

스레드 이진 트리 (9)

- 전위 순회 알고리즘

```
preorderThread(p)
// 중위 스레드 이진트리를 전위 순회
q ← preorderSuccessor(p);
while (q ≠ p) do {
    visit q.data
    q ← preorderSuccessor(q);
}
end preorderThread()
```

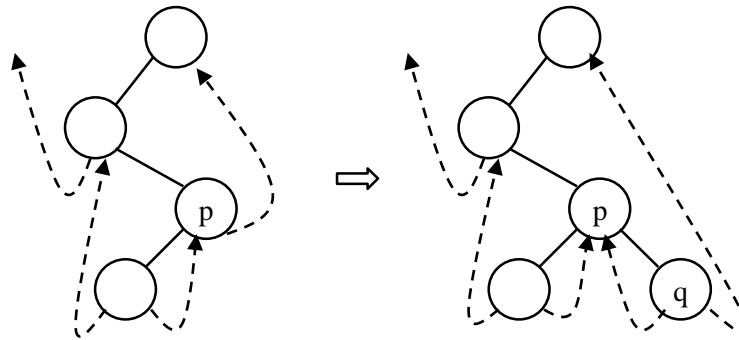
스레드 이진 트리 (10)

◆ 중위 스레드 이진트리에 대한 노드의 삽입과 삭제

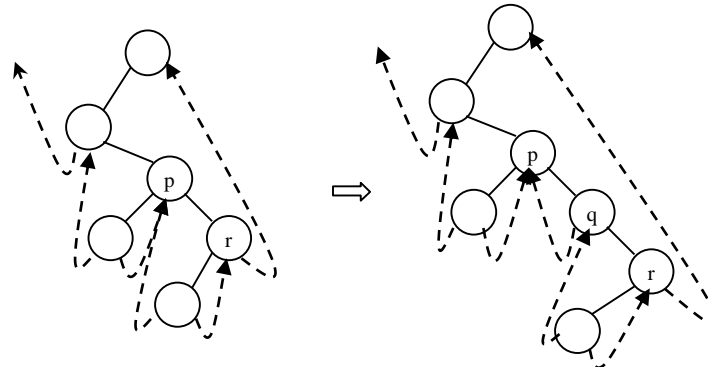
- 개요
 - ◆ 노드 p 의 오른쪽 자식으로 노드 q 를 삽입하는 경우만 설명
- 노드 p 의 오른쪽 서브 트리가 공백시 (그림 (a))
 - ◆ 노드 p 의 `right`와 `rt`를 그대로 노드 q 에 복사
 - ◆ q 의 `left`는 스레드로 p 를 가리키게 하면서 `lt = true`로 만들어 주기만 하면 됨
- 노드 p 가 오른쪽 서브 트리 r 을 가지고 있을 때 (그림 (b))
 - ◆ 이 r 을 노드 q 의 오른쪽 서브 트리로 만듦
 - ◆ 그리고 q 의 `left`는 스레드로 p 를 가리키게 함
 - ◆ 다음에는 q 의 중위 후속자(q 를 삽입하기 전에 p 의 중위 후속자)의 `left`가 스레드로 q 를 가리키게 함

스레드 이진 트리 (11)

- ◆ 스레드 이진 트리에서 **p**의 오른쪽 서브 트리로 **q**를 삽입하는 예



(a) **p**의 오른쪽 서브트리가 공백인 경우



(b) **p**의 오른쪽 서브트리가 공백이 아닌 경우

스레드 이진 트리 (12)

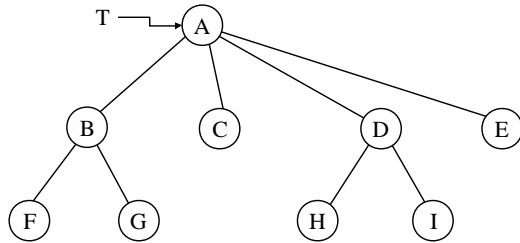
- 오른쪽 노드 삽입 알고리즘

```
threadInsertRight(p, q)
  // 중위 스레드 이진트리에서 노드 p의 오른쪽 서브 트리로 노드 q를
  삽입
  q.right ← p.right;
  q.rt ← p.rt;
  q.left ← p;
  q.lt ← true;
  p.right ← q;
  p.rt ← false;
  if (q.rt = f) then {
    r ← inorderSuccessor(q);
    r.left ← q;
  }
end threadInsertRight()
```

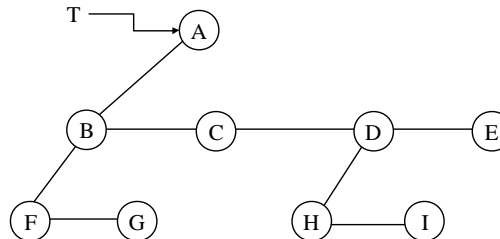
일반 트리를 이진 트리로의 표현 (1)

◆ 일반 트리의 이진 트리로의 변환 방법

- 한 노드의 모든 자식들을 첫 번째 자식과 나머지 다음 형제 관계로 만듦
 - ◆ 모든 노드는 기껏해야 하나의 첫째 자식과 하나의 다음 형제를 가지게 함
 - ◆ 트리(T)의 예



- 노드 B: 첫째 자식은 F, 다음 형제는 C
- 대응 이진 트리: 부모노드와 첫째 자식노드, 그리고 다음 형제노드들을 연결



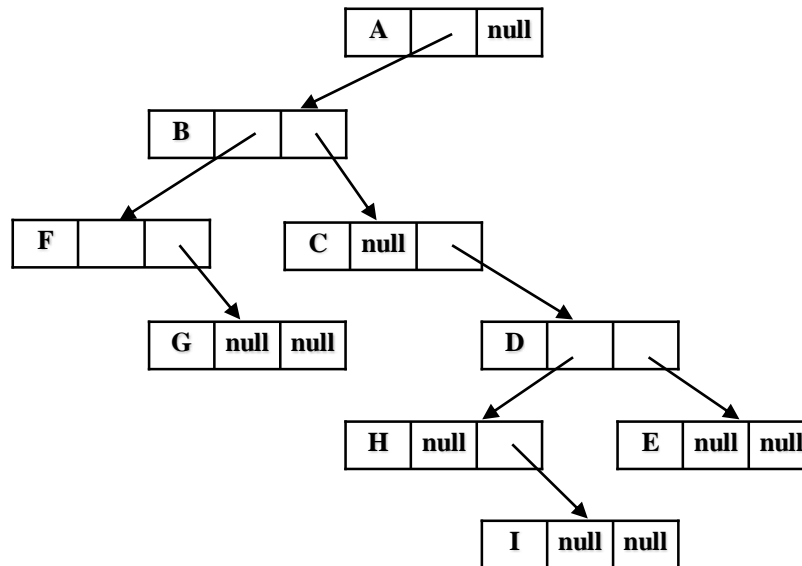
일반 트리를 이진 트리로의 표현 (2)

- 이진 트리 표현

- ◆ 노드구조: data와 두 개의 필드 child와 sibling으로 구성

data	child	sibling
------	-------	---------

- ◆ 오른쪽 서브 트리를 왼쪽으로 45도 회전
- ◆ 널링크는 null로 표시
- ◆ 변환된 이진 트리의 루트 노드는 sibling 필드가 항상 공백
- ◆ 변환된 이진 트리의 연결 리스트 표현



일반 트리를 이진 트리로의 표현 (3)

◆ 트리에 대한 순회 알고리즘

- 트리전위(tree preorder) 순회

- ① 루트를 방문
- ② 첫 번째 서브트리를 트리 전위로 순회
- ③ 나머지 서브트리들을 트리 전위로 순회

- 트리중위(tree inorder)순회

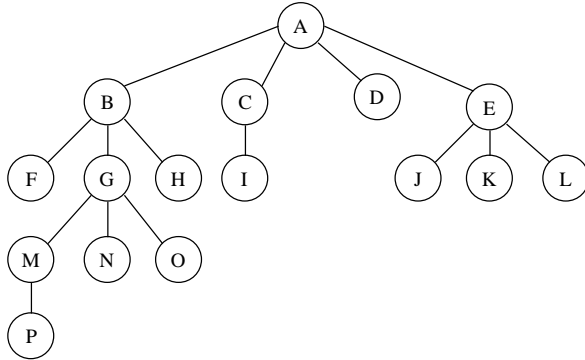
- ① 첫 번째 서브트리를 트리 중위로 순회
- ② 루트를 방문
- ③ 나머지 서브트리들을 트리 중위로 순회

- 트리후위(tree postorder)순회

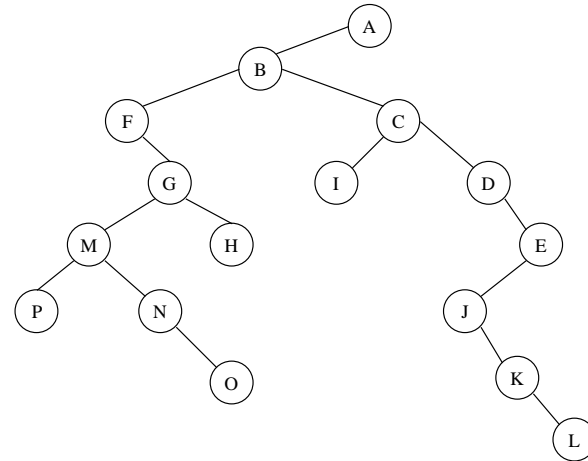
- ① 첫 번째 서브트리를 트리후위로 순회
- ② 나머지 서브트리를 트리 후위로 순회
- ③ 루트를 방문

일반 트리를 이진 트리로의 표현 (4)

- 트리와 변환된 이진 트리의 예



(a) 트리



(b) 변환된 이진 트리

- 트리 순회 방법의 적용 결과
 - ♦ 트리전위순회: A, B, F, G, M, P, N, O, H, C, I, D, E, J, K, L
 - ♦ 트리중위순회: F, B, P, M, G, N, O, H, A, I, C, D, J, E, K, L
 - ♦ 트리후위순회: F, P, M, N, O, G, H, B, I, C, D, J, K, L, E, A
- 변환된 이진트리의 순회 결과와의 비교
 - ♦ 트리전위순회 결과 = 변환된 이진트리의 전위순회 결과
 - ♦ 트리중위순회 결과 \neq 변환된 이진트리의 어떤 순회 결과
 - ♦ 트리후위결과 = 변환된 이진트리의 중위순회 결과

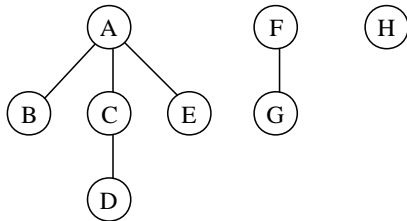
포리스트를 이진 트리로의 표현 (1)

◆ 포리스트의 이진트리로의 변환

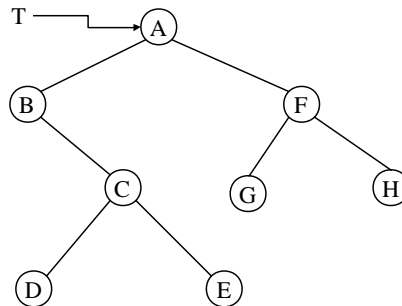
- 개요

- ◆ 각 트리들을 이진트리로 변환
- ◆ 이들의 루트노드들을 첫 번째 루트노드의 형제로 취급하여 Sibling 필드로 연결

- 포리스트 F의 예



- 포리스트 F의 이진트리 T로의 변환 예



포리스트를 이진 트리로의 표현 (2)

◆ 포리스트에 대한 순회 알고리즘

- 포리스트 전위(forest preorder) 순회

- ① F의 첫 번째 트리의 루트를 방문
- ② 첫 번째 트리의 서브 트리들을 포리스트 전위로 순회
- ③ F의 나머지 트리들을 포리스트 전위로 순회

- 포리스트 중위(forest inorder) 순회

- ① 첫 번째 트리의 서브 트리들을 포리스트 중위로 순회
- ② 첫 번째 트리의 루트를 방문
- ③ 나머지 트리들을 포리스트 중위로 순회

- 포리스트 후위(forest postorder) 순회

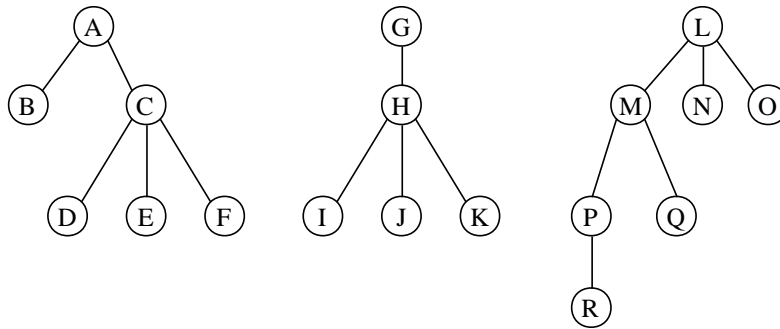
- ① 첫 번째 트리의 서브 트리들을 포리스트 후위로 순회
- ② 나머지 트리들을 포리스트 후위로 순회
- ③ 첫 번째 트리의 루트를 방문

- 변환된 이진트리의 순회 결과와의 비교

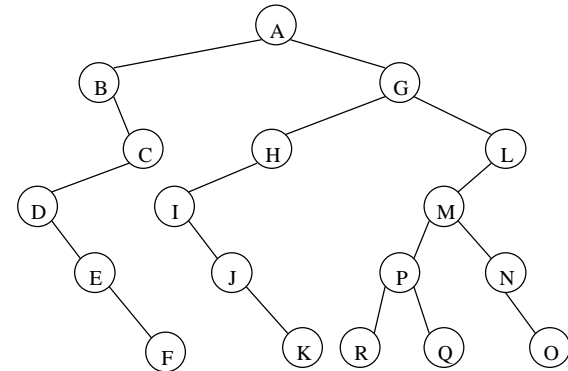
- ◆ 포리스트 전위순회 결과 = 변환된 이진트리의 전위순회 결과
- ◆ 포리스트 중위순회 결과 = 변환된 이진트리의 중위순회 결과
- ◆ 포리스트 후위순회 결과 \neq 변환된 이진트리의 후위순회 결과

포리스트를 이진 트리로의 표현 (3)

- 3개의 트리로 구성된 포리스트와 이를 변환한 이진트리 예



(a) 포리스트



(b) 변환 이진 트리

- 순회 결과의 비교
 - 포리스트 전위순회 결과 = 변환된 이진트리의 전위순회 결과
= (A, B, C, D, E, F, G, H, I, J, K, L, M, P, R, Q, N, O)
 - 포리스트 중위순회 결과 = 변환된 이진트리의 중위순회 결과
= (B, D, E, F, C, A, I, J, K, H, G, R, P, Q, M, N, O, L)
 - 포리스트의 후위순회 결과
= (B, D, E, F, C, I, J, K, H, G, R, P, Q, M, N, O, L, A)
 - 변환된 이진트리의 후위순회 결과
= (F, E, D, C, B, K, J, I, H, R, Q, P, O, N, M, L, G, A)

포리스트를 이진 트리로의 표현 (4)

◆ 포리스트의 레벨 순서순회

- 알고리즘 개요

- ◆ 포리스트에 있는 각 트리의 루트 노드에서부터 시작하여 레벨별로 노드들을 순회
- ◆ 같은 레벨 안에서는 왼편에서 오른편으로 순회

- 포리스트의 레벨 순서 순회와 변환된 이진트리의 레벨순서 순회는 그 결과가 반드시 일치하지 않음
 - ◆ 위 그림 (a)의 포리스트 레벨순서순회 결과:

A, G, L, B, C, H, M, N, O, D, E, F, I, J, K, P, Q, R