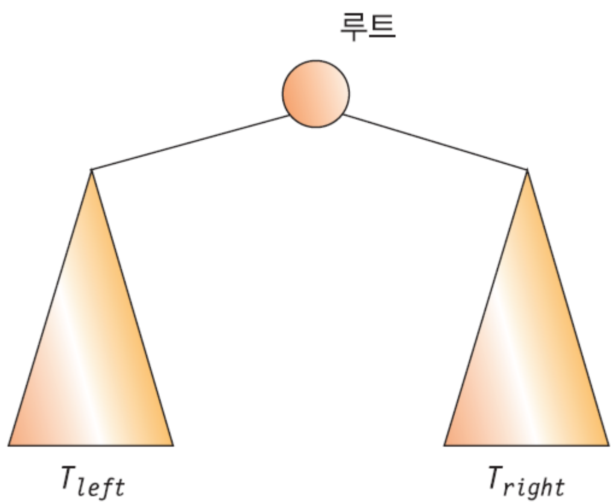
CSE2010 자료구조론

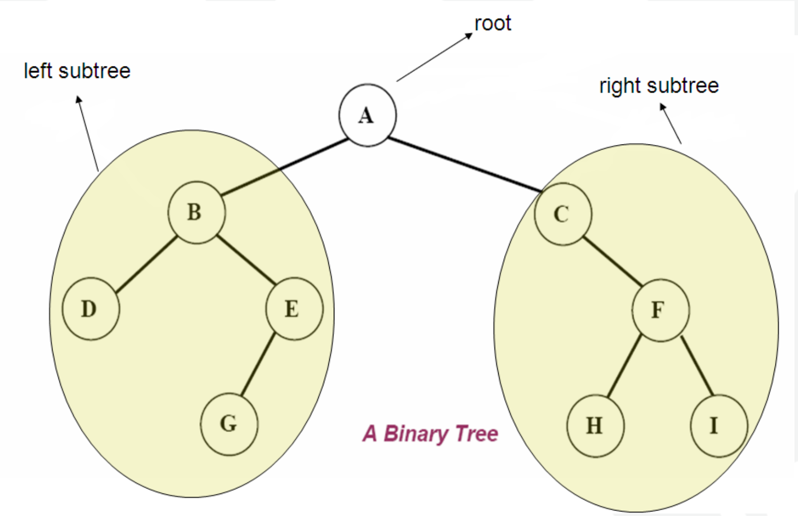
Week 6: Tree 2

ICT융합학부 한진영

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

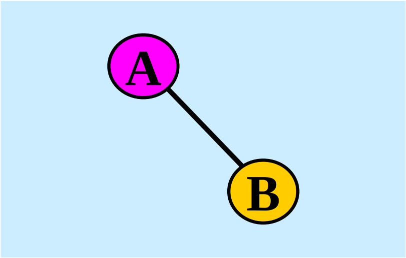




이진 트리의 서브 트리

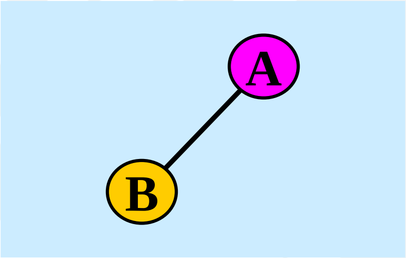
- 이진 트리의 서브 트리는,
  - (1) 공집합이거나
  - (2) 루트와 왼쪽 서브 트리, 오른쪽 서브 트리로 구성된 노드들의 유한 집합으로 정의됨
- 이진 트리에는 서브 트리간의 순서가 존재
  - 예

왼쪽 부분트리가 공집합 오른쪽 부분 트리가 공집합



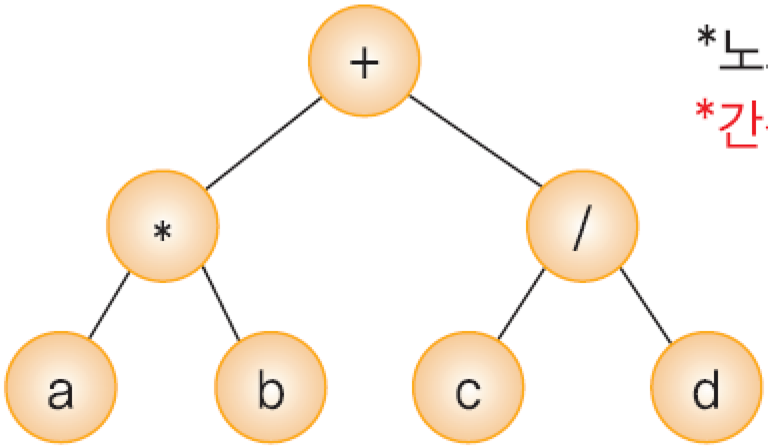
≠

서로 다른 이진 트리



이진 트리의 성질(1)

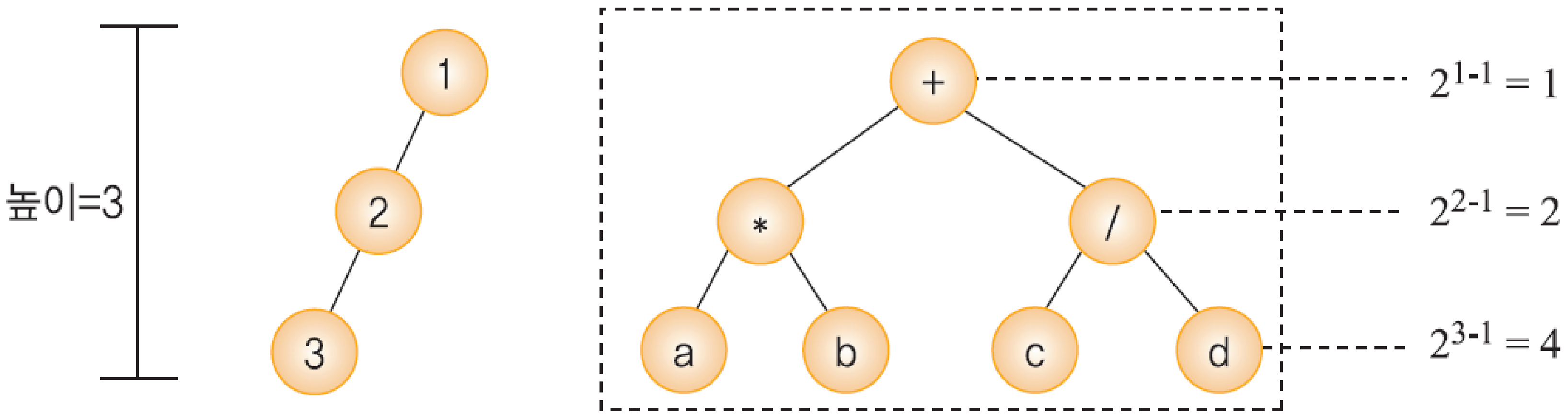
- 노드의 개수가 n개이면, 간선의 개수는 n-1



*노드의 개수: 7
*간선의 개수: 6

이진 트리의 성질(2)

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

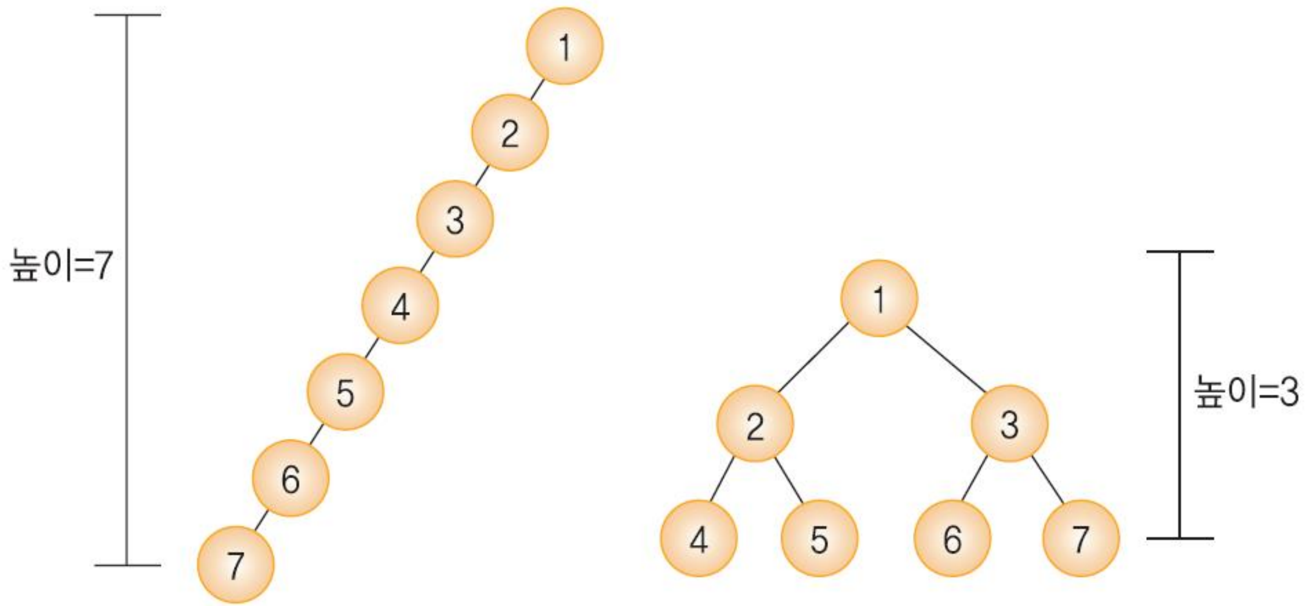


최소 노드 개수 = 3

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

이진 트리의 성질(3)

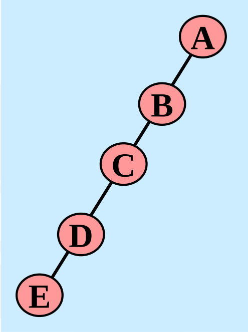
- n개의 노드를 가지는 이진트리의 높이
  - 최대 n
  - 최소 $\lceil \log_2(n+1) \rceil$



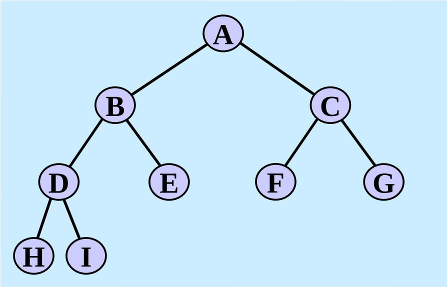
(a) 최대 높이 (b) 최소 높이

이진 트리의 종류

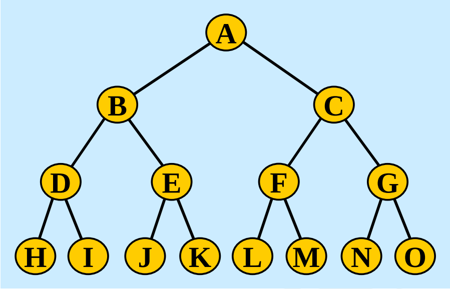
- 사향 이진 트리(skewed binary tree)
  - 모두 왼쪽 자식만 있거나(왼쪽 사향 이진 트리),
  - 모두 오른쪽 자식만 있는 트리(오른쪽 사향 이진 트리)



- 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

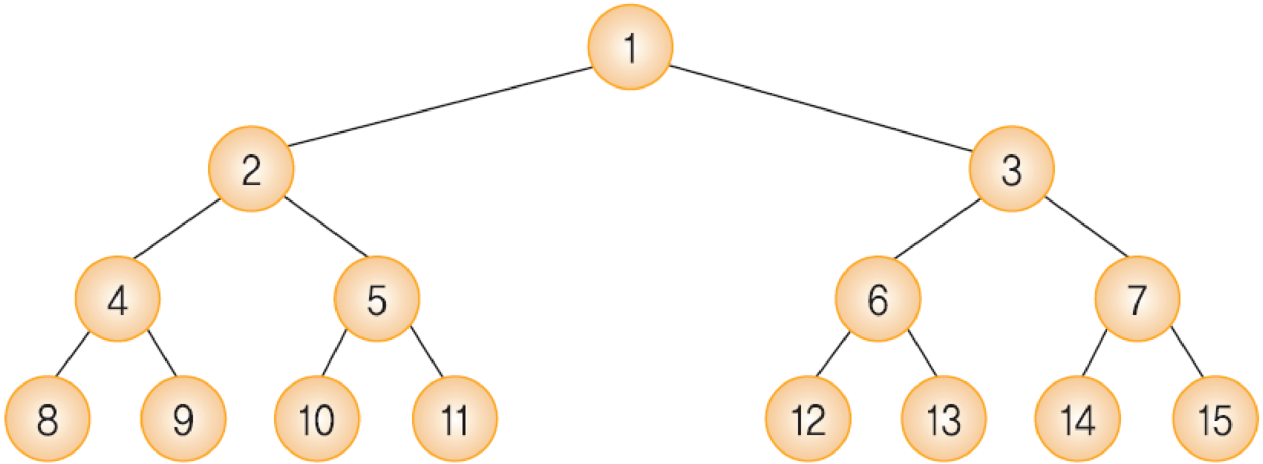


포화 이진 트리

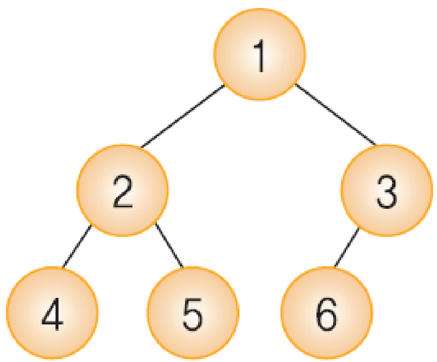
- 트리의 각 레벨에 노드가 꽉 차있는 이진트리

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

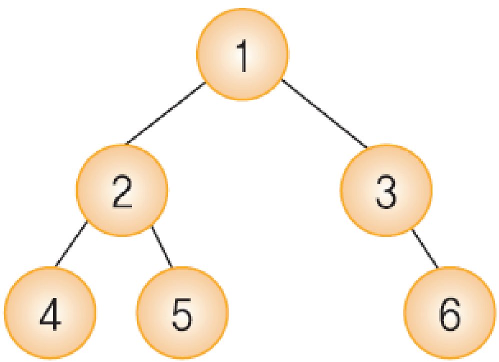
- 포화 이진 트리에는 다음과 같이 각 노드에 번호를 붙일 수 있음



- 완전 이진 트리(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

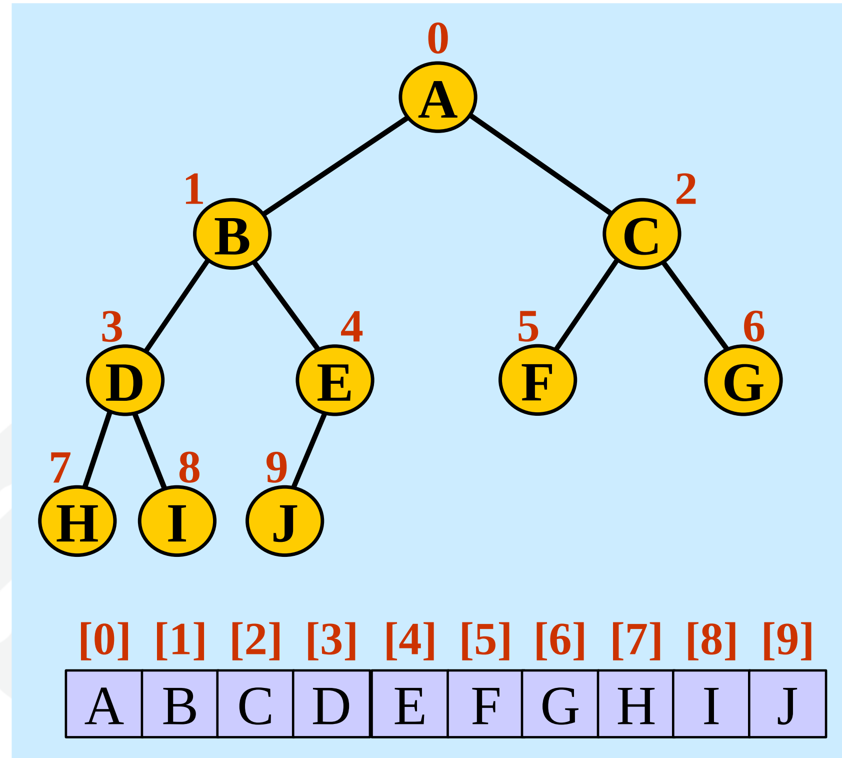
  (증명)

  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
  - $n = n_0 + n_1 + n_2$, $b + 1 = n$, $b = n_1 + 2 \cdot n_2$
    - $n_1 + 2 \cdot n_2 + 1 = n$
    - $n_1 + 2 \cdot n_2 + 1 = n_0 + n_1 + n_2$
    - $n_0 = n_2 + 1$

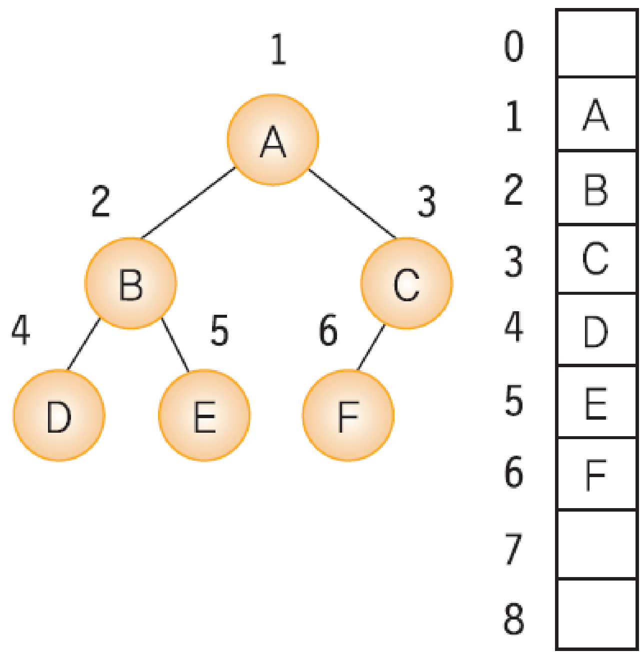
이진 트리의 표현

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

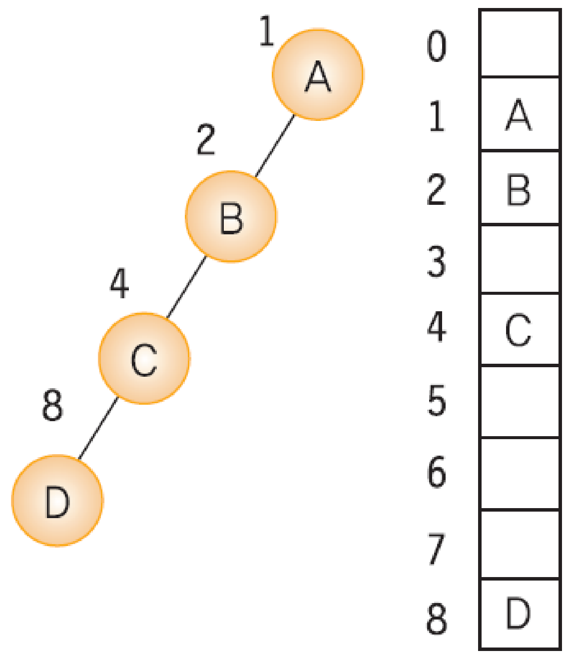


이진 트리의 표현: 배열 표현법

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



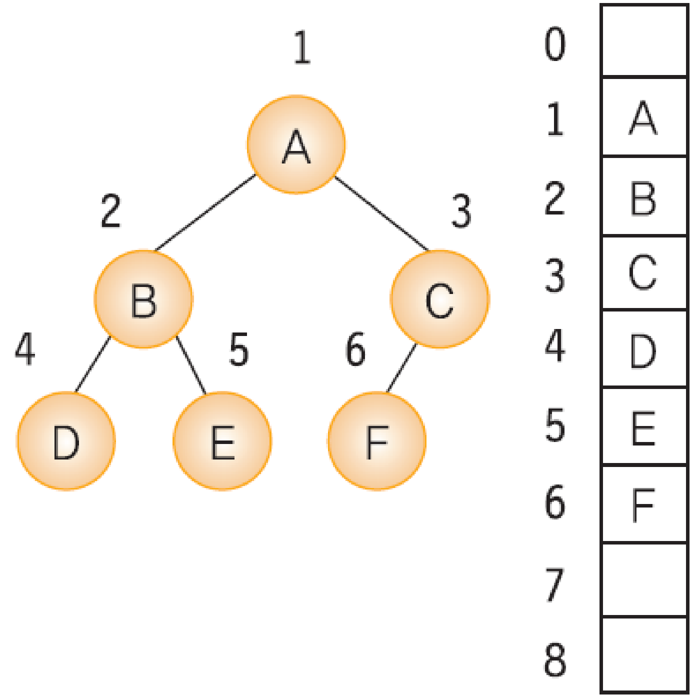
(a) 완전 이진 트리



(b) 경사 이진 트리

배열 표현법: 노드 위치 계산

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

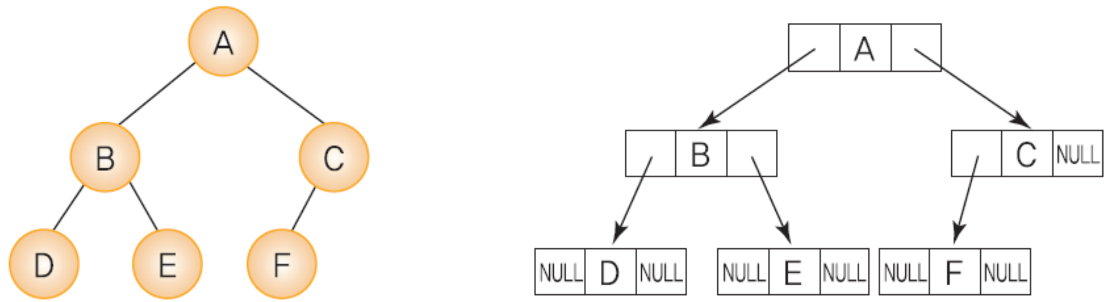


배열 표현법: 장단점

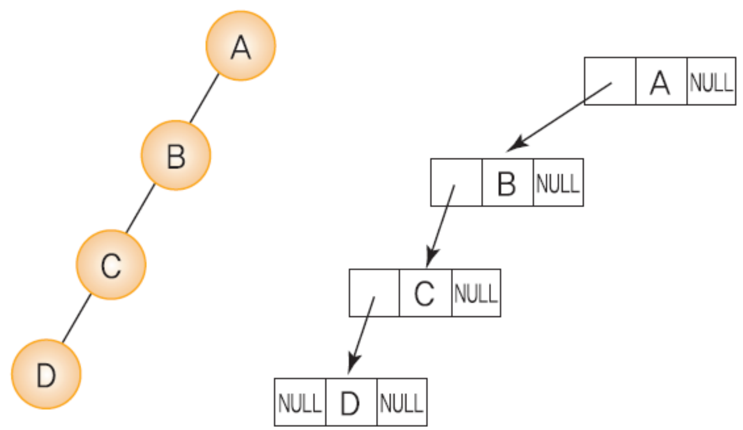
- 장점
  - 임의의 노드에 대해 부모, 자식 노드의 위치 계산이 쉬움
- 단점
  - 완전 이진 트리가 아니면 메모리 낭비가 큼
  - 예: 사향트리

이진 트리의 표현: 연결리스트 표현법

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

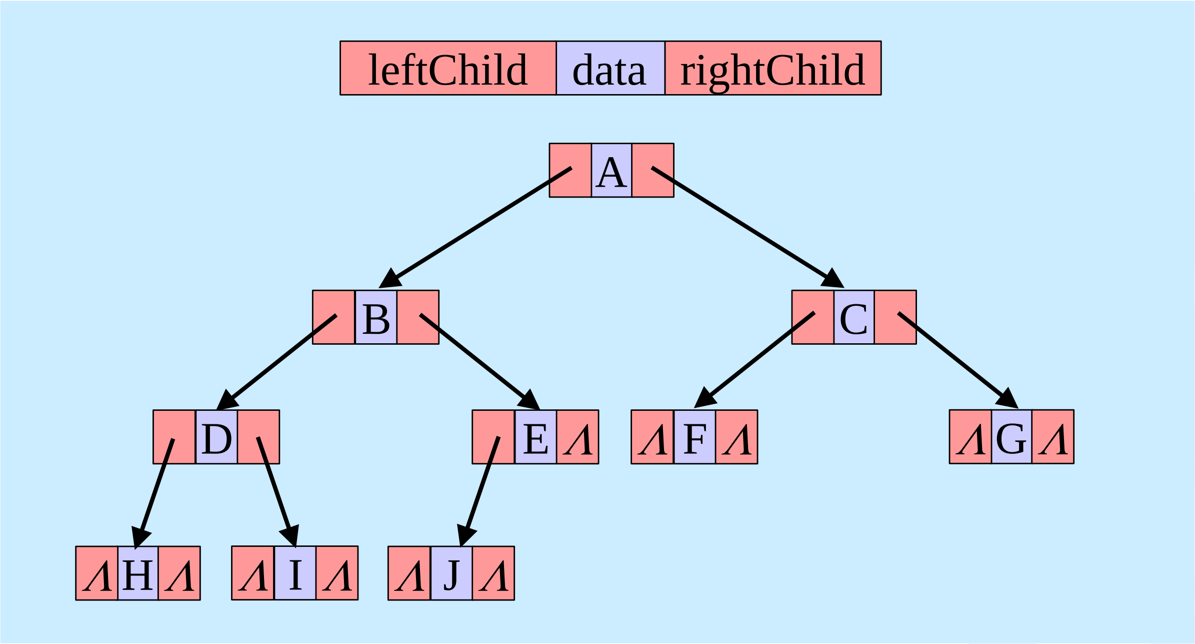


(a) 완전 이진 트리



(b) 경사 이진 트리

연결리스트 표현법에 의한 구현(1)




```
typedef struct 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;
}
```

Week 6: Tree 2