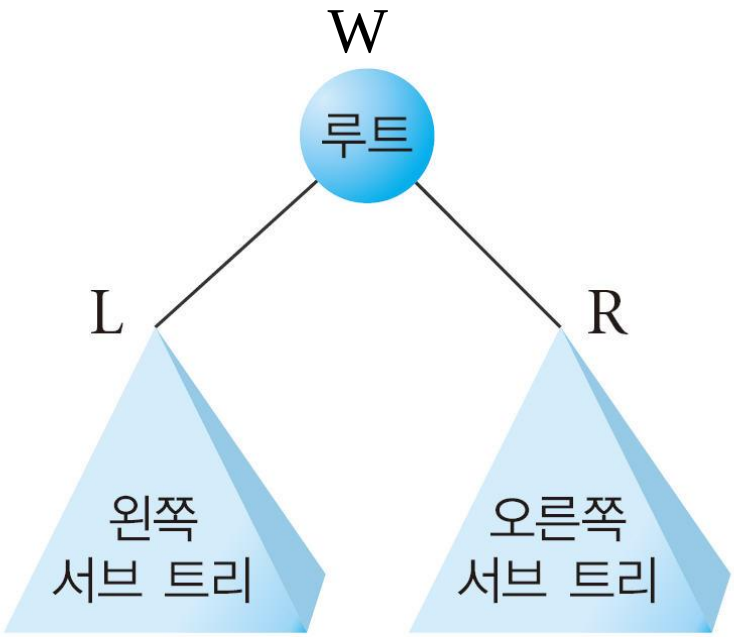
Part 4 – Tree Traversal

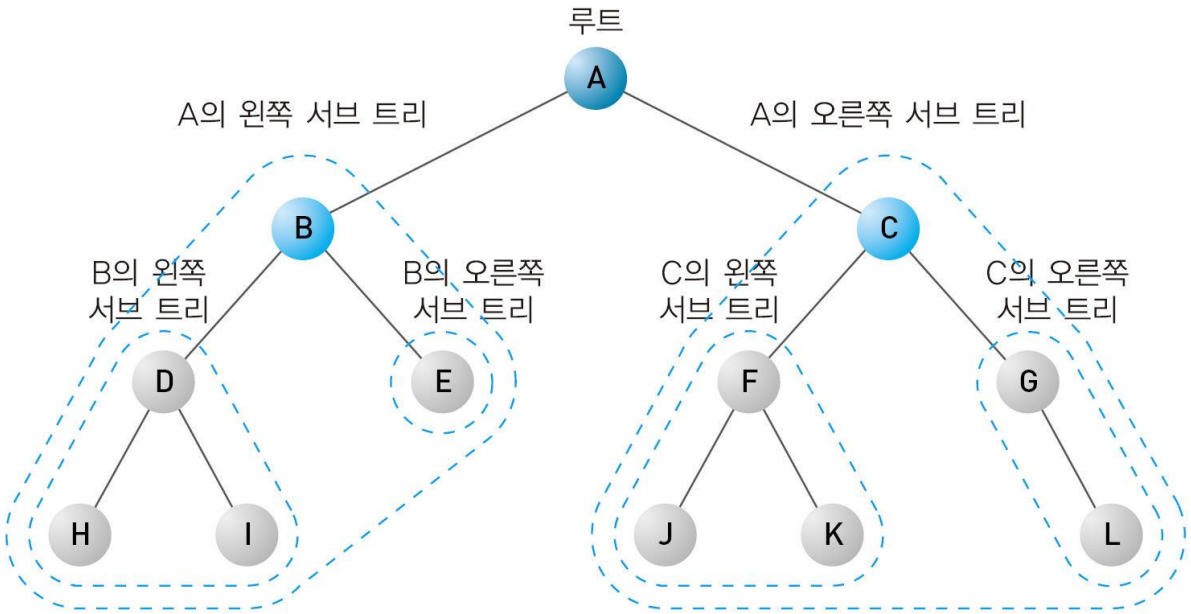
Binary Tree Traversal Methods



- 이진 트리 순회(traversal)
  - 이진 트리에 속하는 모든 노드를 한 번씩 방문하여 노드가 가지고 있는 데이터를 목적에 맞게 처리하는 것
- 방문하는 순회 방법으로는 기본적으로
  - Preorder(전위순회, WLR)
  - Inorder(중위순회, LWR)
  - Postorder(후위순회, LRW)
  - Level order(LRLR..)

❶ 현재 노드를 방문하여 작업 W
❷ 현재 노드의 왼쪽 서브 트리로 이동하는 작업 L
❸ 현재 노드의 오른쪽 서브 트리로 이동하는 작업 R

- 이진트리의 구조
  - 전체 트리의 구조와 서브 트리의 구조가 동일
  - 따라서 전체 트리 순회에 사용된 알고리즘을 서브 트리 순회에 똑같이 적용할 수 있으므로 재귀 함수로 구현할 수 있다

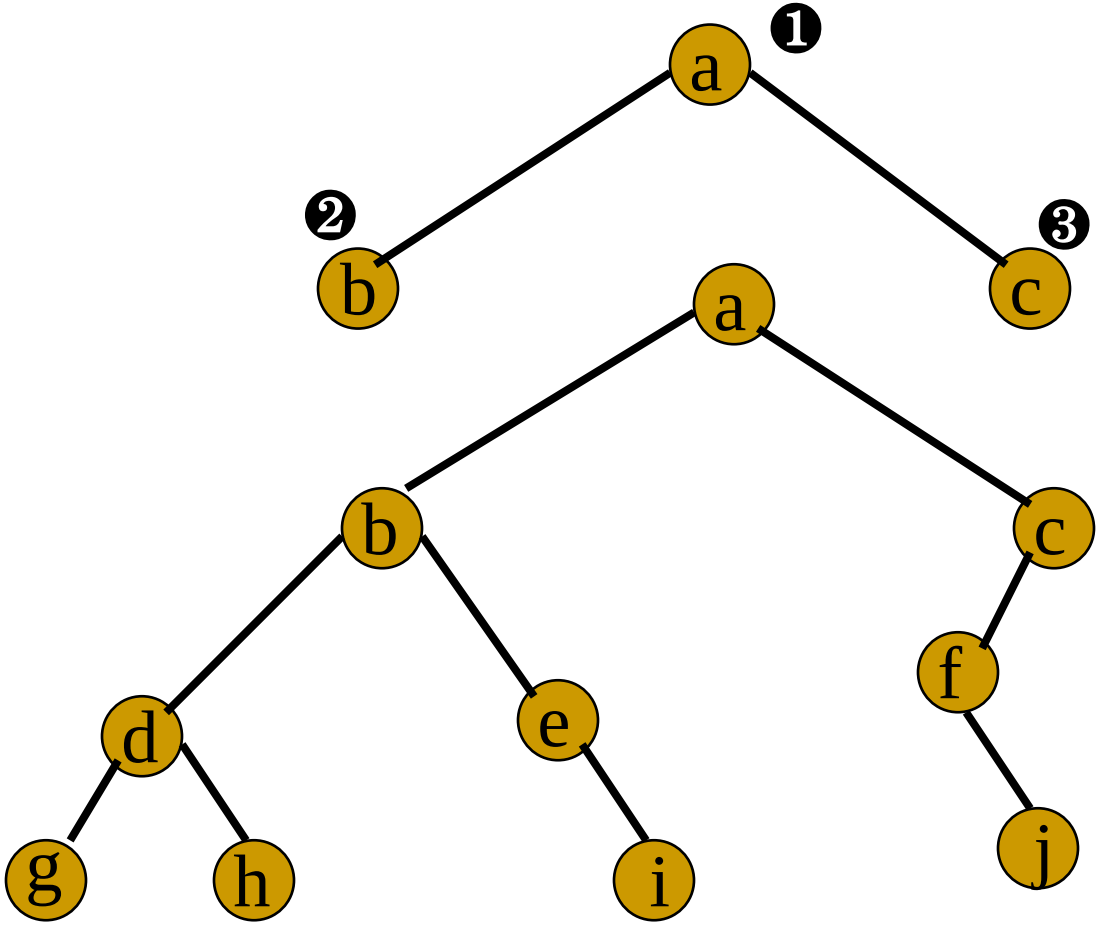


Preorder Traversal(전위순회)

❶ 현재 노드 n을 방문하여 처리한다 : W
❷ 현재 노드 n의 왼쪽 서브 트리로 이동한다 : L
❸ 현재 노드 n의 오른쪽 서브 트리로 이동한다 : R

```
void preOrder(treePointer *ptr)
{
   if (ptr != NULL)
   {
      visit(ptr);    // ❶ 예를 들면 printf("%c", ptr->data)
      preOrder(ptr->leftChild); // ❷
      preOrder(ptr->rightChild);  // ❸
   }
}
```

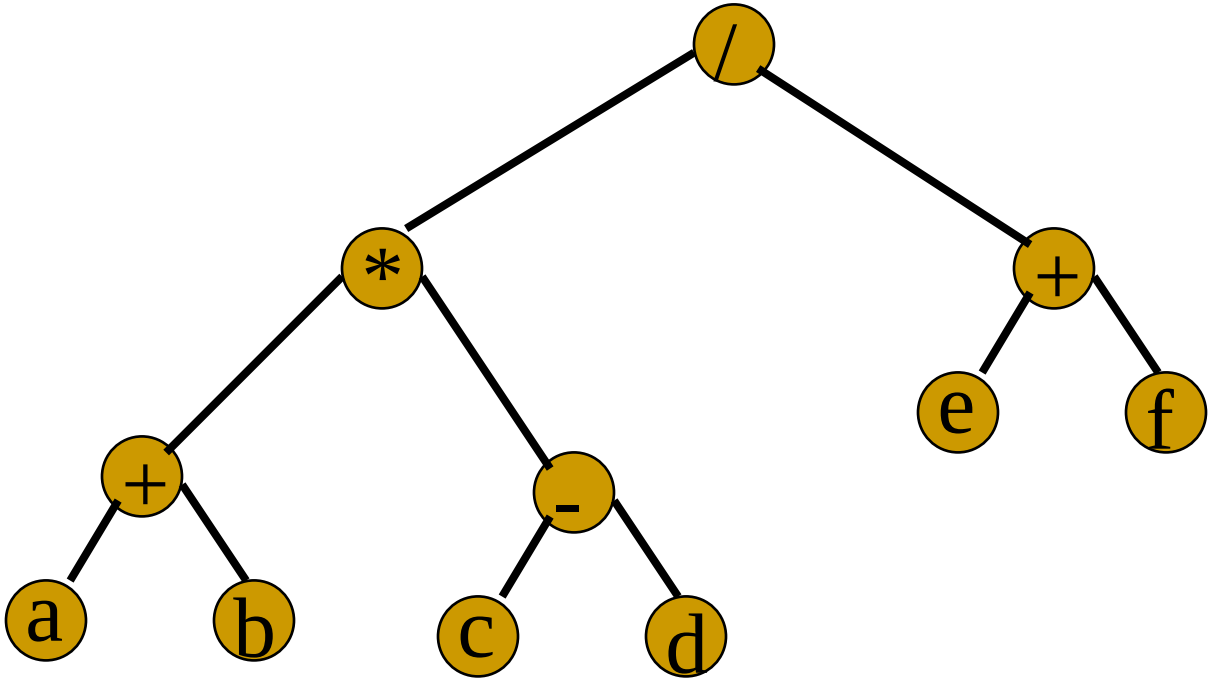
preorder Example (Visit = print)



a b c

a b d g h e i c f j

preorder Of Expression Tree



/ * +a b- c d+e f

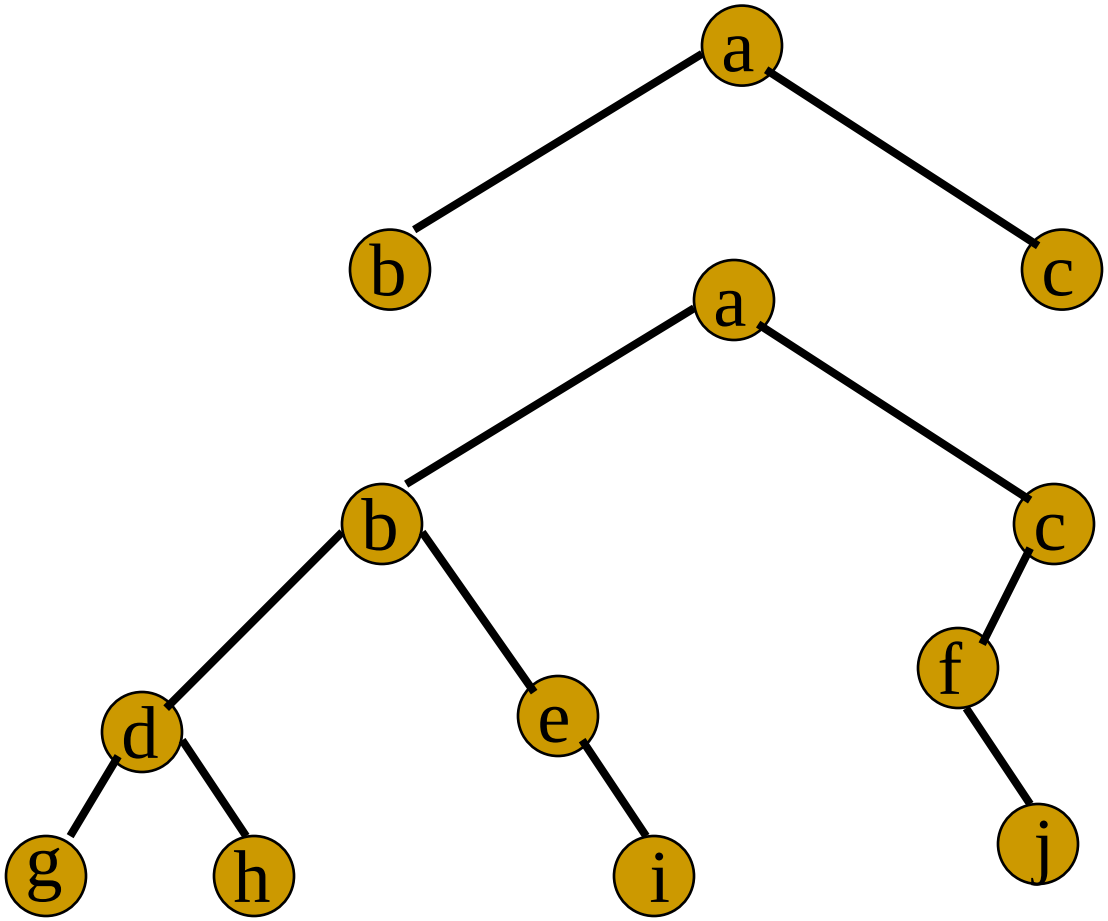
prefix form of expression!

Inorder Traversal(중위순회)

❶ 현재 노드 n의 왼쪽 서브 트리로 이동한다 : L
❷ 현재 노드 n을 방문하여 처리한다 : W
❸ 현재 노드 n의 오른쪽 서브 트리로 이동한다 : R

```
void inOrder(treePointer *ptr)
{
   if (ptr != NULL)
   {
      inOrder(ptr->leftChild);
      visit(ptr);    //  예를 들면 printf("%c", ptr->data)
      inOrder(ptr->rightChild);
   }
}
```

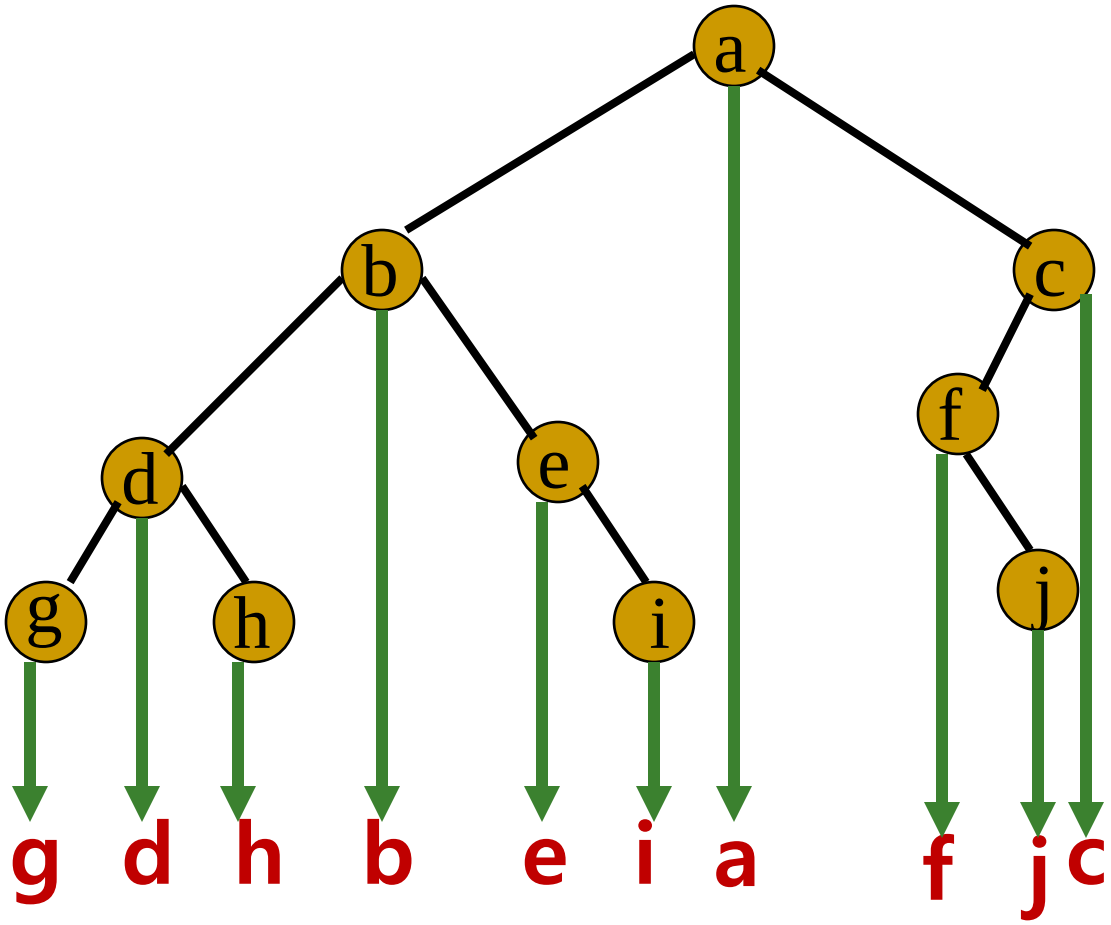
inorder Example (Visit = print)



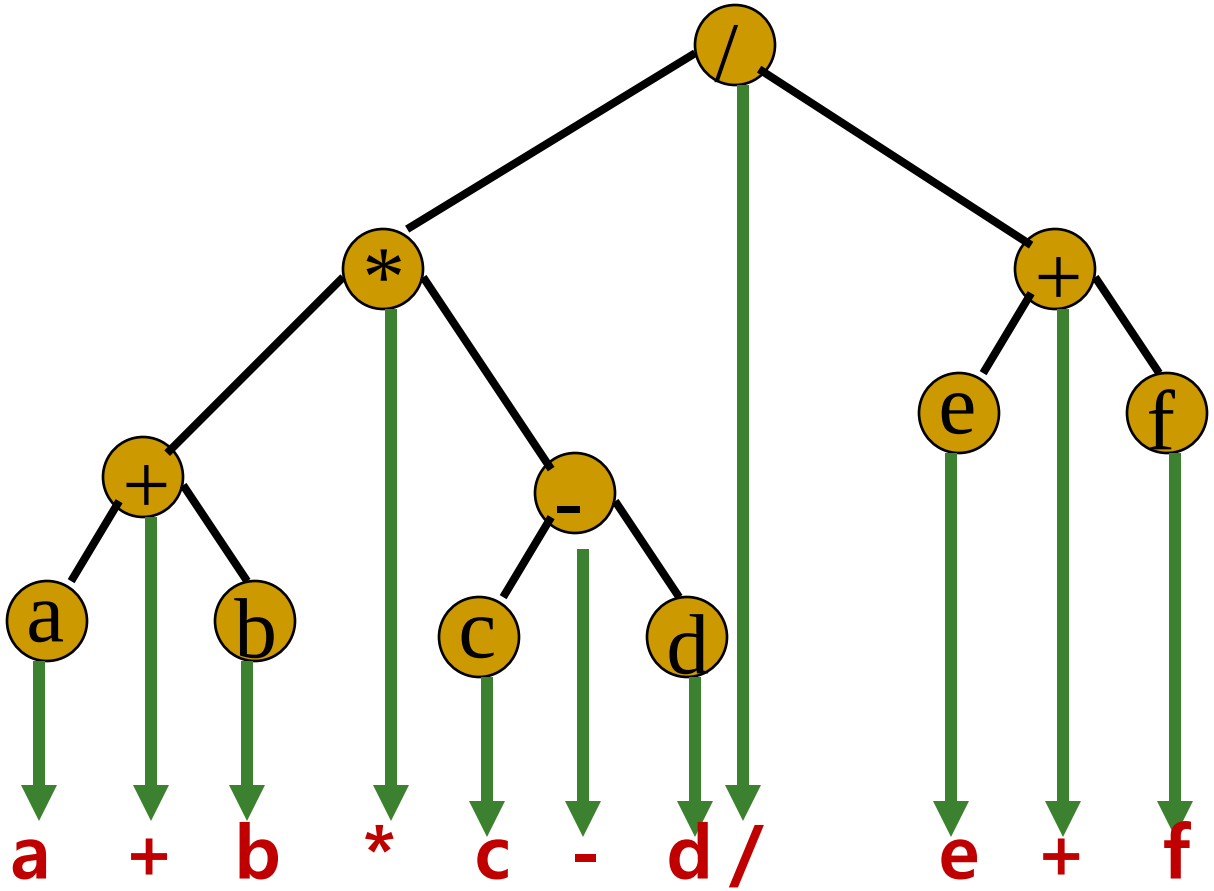
b a c

g d h b e i a f j c

inorder By Projection (Squishing)



inorder Of Expression Tree



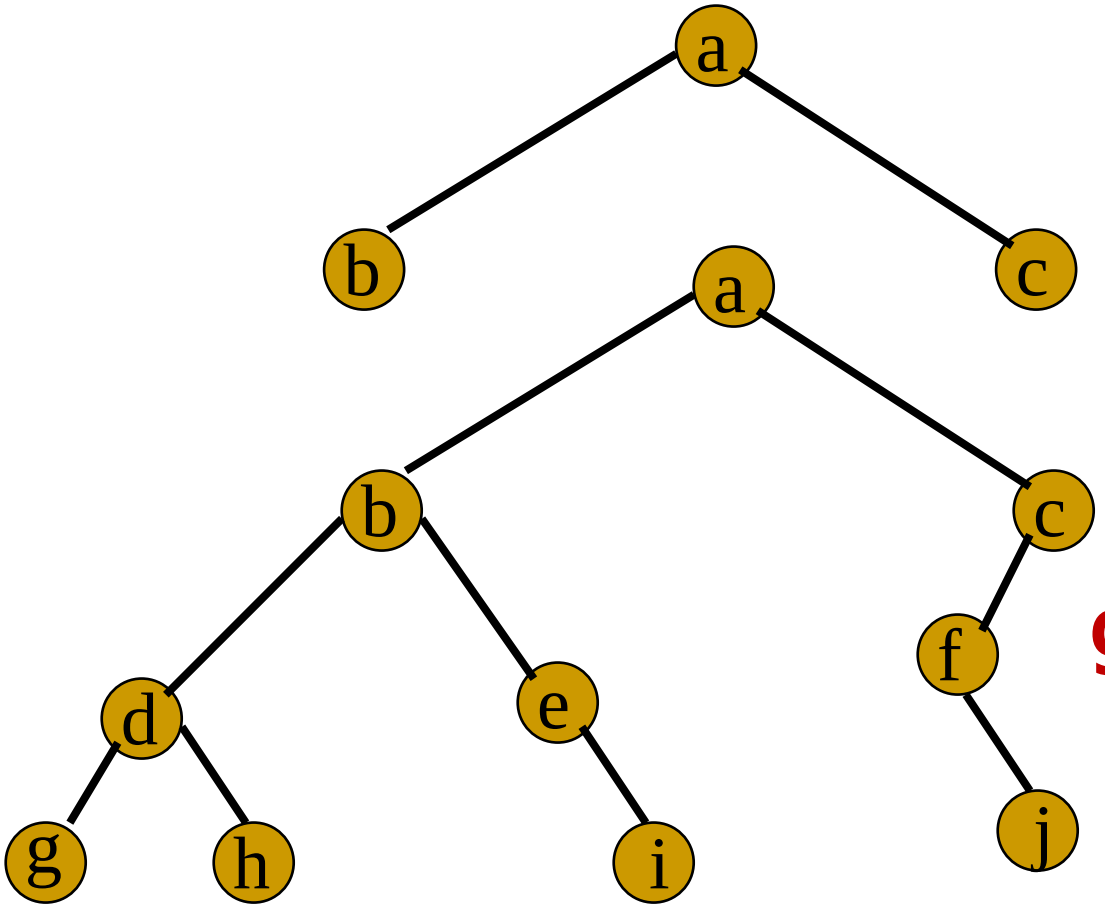
infix form of expression (sans parentheses)!

Postorder Traversal(후위순회)

❶ 현재 노드 n의 왼쪽 서브 트리로 이동한다 : L
❷ 현재 노드 n의 오른쪽 서브 트리로 이동한다 : R
❸ 현재 노드 n을 방문하여 처리한다 : W

```
void postOrder(treePointer *ptr)
{
    if (ptr != NULL)
    {
        postOrder(ptr->leftChild);
        postOrder(ptr->rightChild);
        visit(ptr);
    }
}
```

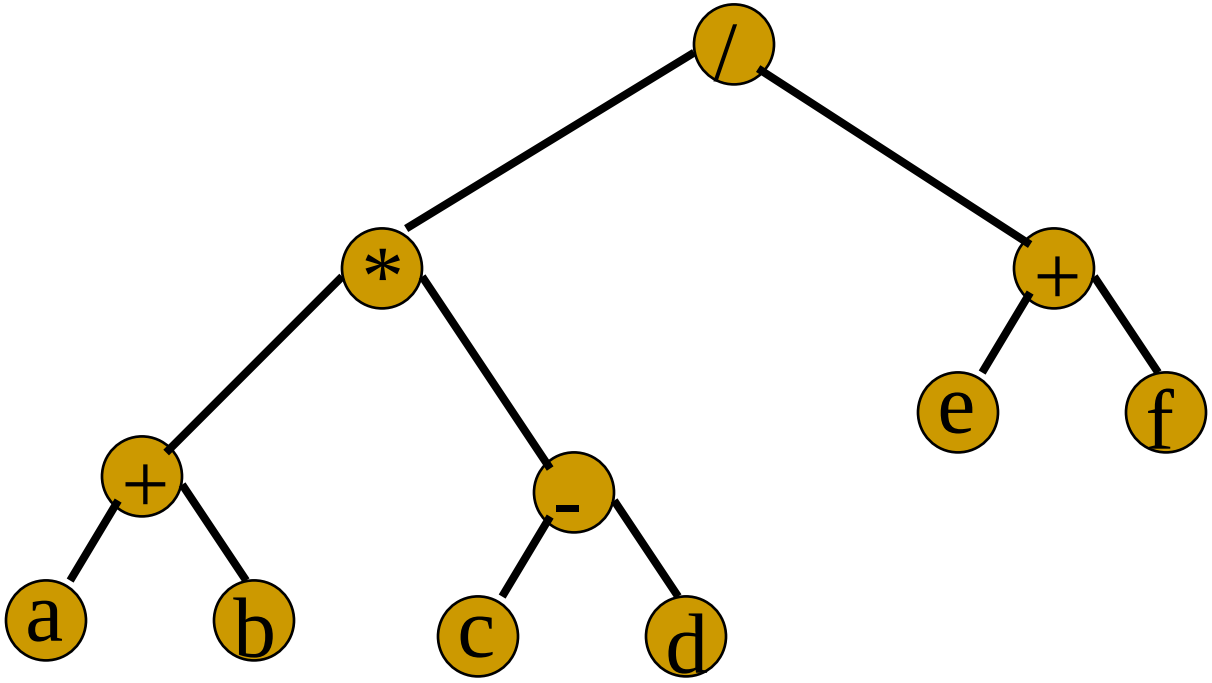
postorder Example (Visit = print)



b c a

g h d i e b j f c a

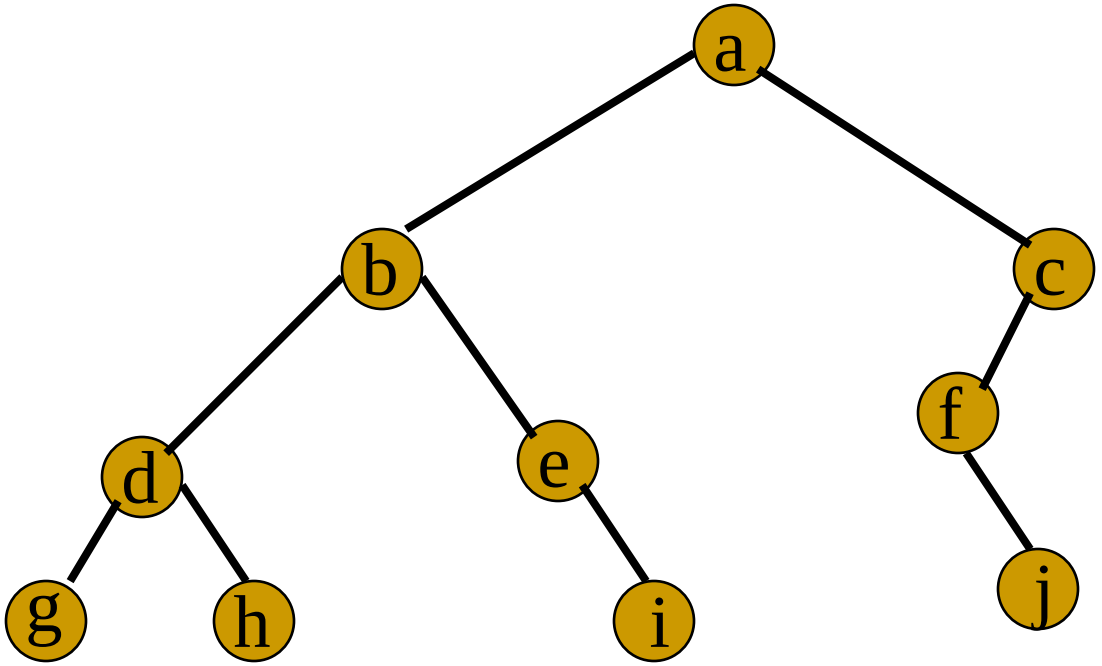
postorder Of Expression Tree



a b + c d - * e f + /

postfix form of expression!

Traversal Applications



- Make a clone.
- Determine height.
- Determine number of nodes.

디렉토리 용량 계산

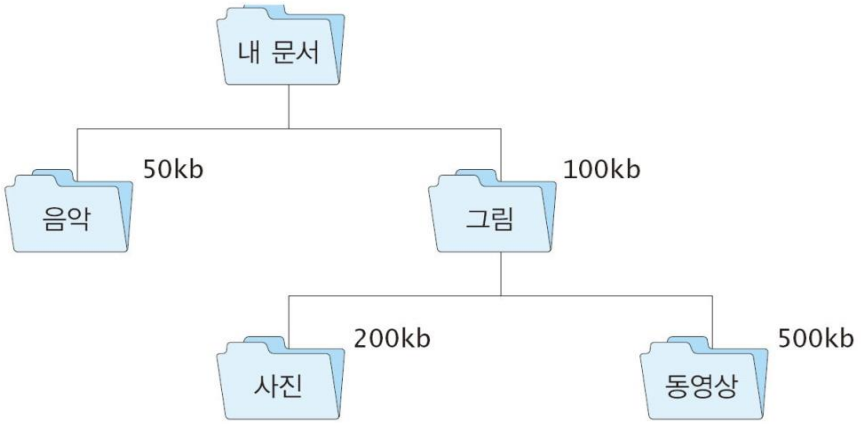


그림 폴더의 전체 용량

= 그림 폴더의 용량(100KB) + 하위 폴더의 용량{사진 폴더(200KB)
+ 동영상 폴더(500KB)}

= 800KB

나의 문서 전체 용량

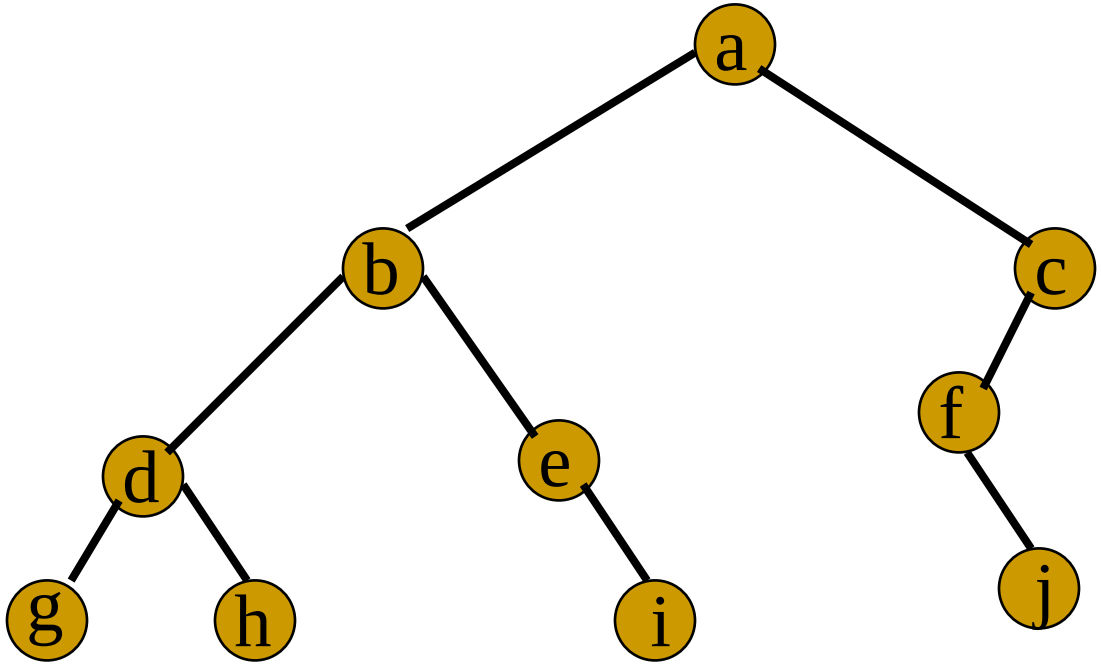
= 나의 문서의 용량(0M) + 하위 폴더의 용량{음악 폴더(50KB)
+ 그림 폴더의 전체 용량(800KB)}

= 850KB

Level Order

```
ptr = &root_node
while (ptr != NULL)
{
    - visit(ptr);
    - add(leftChildren, FIFO); add(rightChildren, FIFO);
    - if(isEempty(FIFO)), ptr = NULL;
      else remove a node from the FIFO, and ptr = &node;
}
```

Level-Order Example (Visit = print)



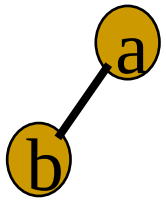
a b c d e f g h i j

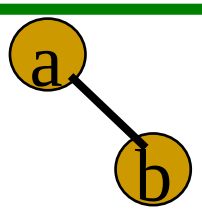
순회로부터의 Binary Tree 의 구성

- Binary tree를 구성하는 모든 노드는 서로 구별된다고 가정한다
- 어떤 traversal sequence가 주어지면 그 sequence를 만족하는 단 하나의 binary tree를 구성할 수 있는가?
  - Traversal sequence가 2개 이상의 노드로 구성될 경우, 그에 대응하는 binary tree 는 unique하지 않다, 즉 2개 이상의 tree가 존재할 수 있다

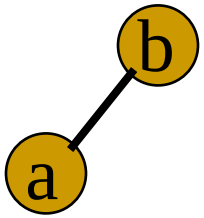
Some Examples

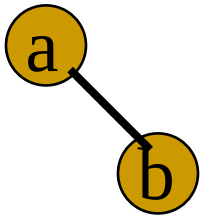
preorder = ab



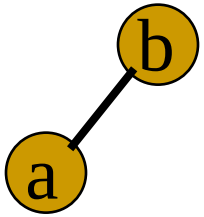


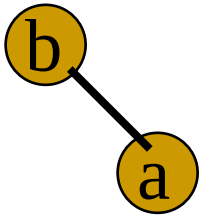
inorder = ab



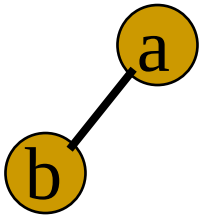


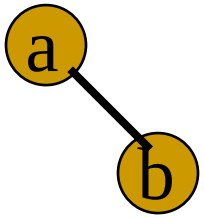
postorder = ab





level order = ab





Binary Tree Construction

- 그렇다면 2가지 형태의 traversal sequence가 주어진다면 unique한 binary tree의 구성이 가능한가?
  - 이는 그 2개의 sequence가 어떻게 주어지는 가에 따라 달라질 수 있다

Given Preorder and Postorder

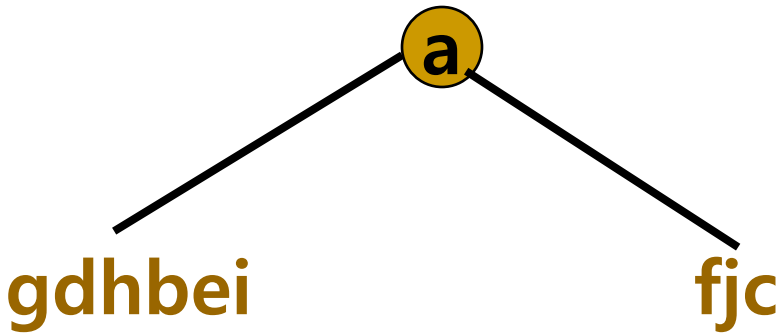
preorder = ab

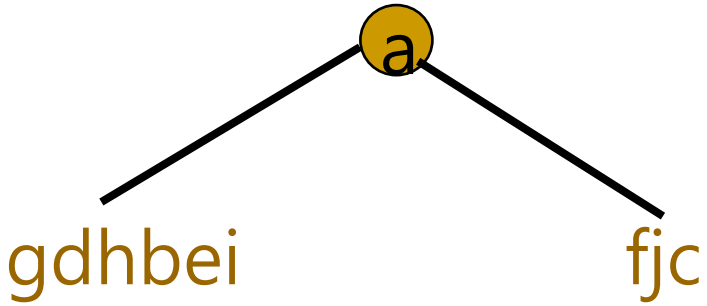
postorder = ba

- **preorder and postorder** do not uniquely define a binary tree.
- Nor do **preorder and level order** (same example).
- Nor do **postorder and level order** (same example).

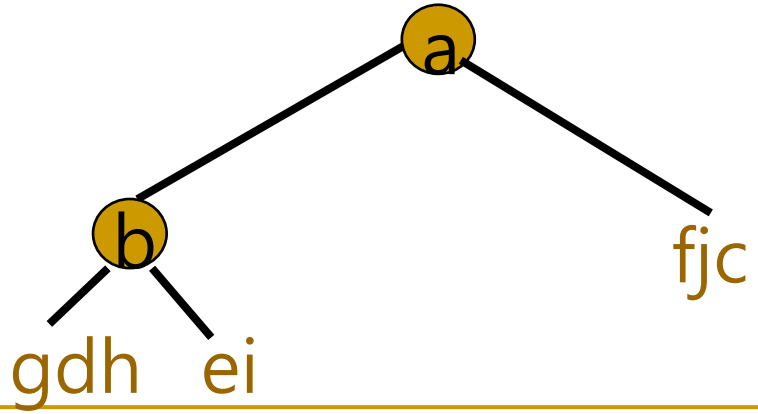
Inorder and Preorder

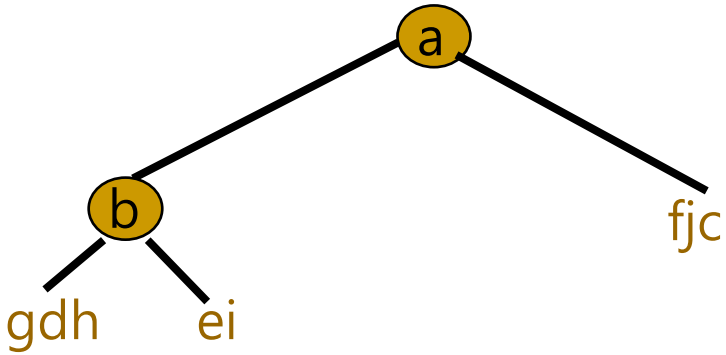
- inorder = g d h b e i **a** f j c
- preorder = **a** b d g h e i c f j
- preorder sequence를 기반으로 inorder sequence를 왼쪽부터 오른쪽으로 스캔 하면서 left subtree와 right subtree를 구성한다
- **a** 가 root 노드가 되고; **gdhbei** 는 left subtree의 inorder sequence가 되며; **fjc** 는 right subtree의 inorder sequence가 될 것이다



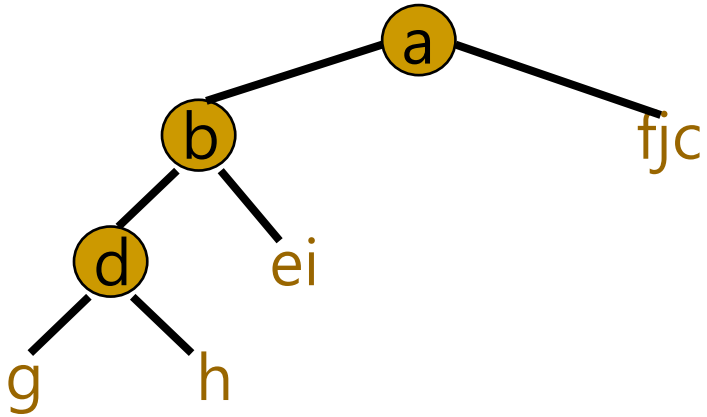


- preorder = a b d g h e i c f j inorder = g d h b e i **a** f j c
- b is the next root; gdh are in the left subtree; ei are in the right subtree.





- preorder = a b d g h e i c f j inorder = g d h b e i **a** f j c
- d is the next root; g is in the left subtree; h is in the right subtree.



Inorder and Postorder

- postorder sequence를 기반으로 inorder sequence를 오른쪽부터 왼쪽으로 스캔 하면서 left subtree와 right subtree를 구성한다
- inorder = g d h b e i **a** f j c
- postorder = g h d i e b j f c **a**
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.

Inorder and Level Order

- levelorder sequence를 기반으로 inorder sequence를 왼쪽부터 오른쪽으로 스캔 하면서 left subtree와 right subtree를 구성한다
- inorder = g d h b e i a f j c
- level order = a b c d e f g h i j
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.