



부산대학교
PUSAN NATIONAL UNIVERSITY

Tree Traversals

When Do We Visit the Root?

Recursive Orders, a Stack, and a Queue

DS3600082-140

자료구조

2026학년도 2학기

오용경

데이터사이언스전공

yongkyung.oh@pusan.ac.kr

Why Order Matters

어느 자식 노드를 먼저 방문할지 정해야 하는 이유를 설명한다

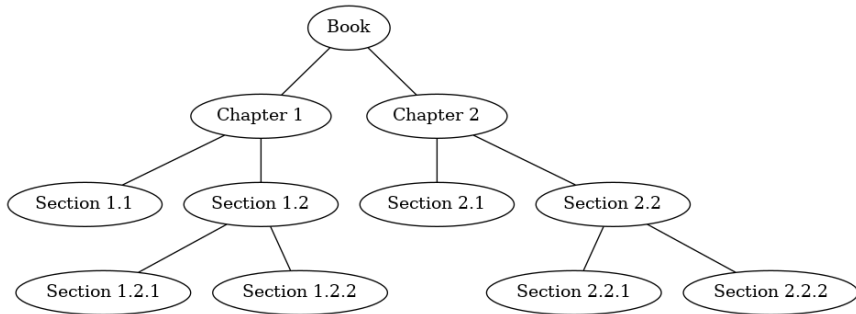
정의

트리 순회

트리의 모든 노드를 정해진 순서로 한 번씩 방문하는 절차다. 한 노드에서 왼쪽과 오른쪽 가운데 어느 쪽을 먼저 볼지, 현재 노드를 언제 처리할지 정해야 한다. 연결리스트는 다음 노드가 하나뿐이라 순서가 정해져 있지만 트리는 갈라지는 곳마다 선택해야 한다.

책의 구조도 트리다

책, 장, 절, 소절을 노드로 두면 문서의 계층을 트리로 표현할 수 있다.



책을 앞에서부터 읽을 때는 상위 제목을 먼저 보고 그 아래 내용을 차례로 읽는다.

책을 읽는 순서는 루트부터 시작한다

순서	방문 노드
1	Book
2	Chapter 1
3	Section 1.1
4	Section 1.2와 그 소절
5	Chapter 2와 그 아래

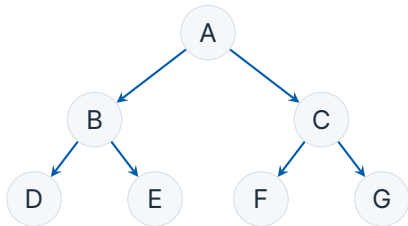
상위 제목을 먼저 처리하고 각 부분 트리로 내려가는 순서가 전위 순회다.

세 순서는 루트를 처리하는 때가 다르다

순회	루트를 처리하는 때
전위 순회	두 부분 트리보다 먼저
중위 순회	왼쪽과 오른쪽 부분 트리 사이
후위 순회	두 부분 트리보다 나중

Source: Miller and Ranum (2013), Tree Traversals.

같은 트리도 출력 순서가 달라진다



전위 순회는 A로 시작하고 중위 순회는 D로 시작하며 후위 순회는 A로 끝난다.

Checkpoint

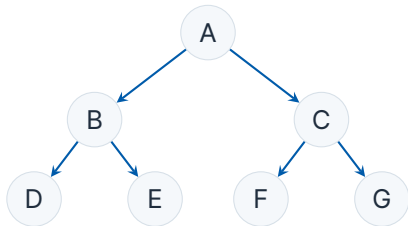
Why Order Matters

- 연결리스트보다 트리에서 방문 순서를 먼저 정해야 하는 이유는 무엇인가?
- 전위 순회, 중위 순회, 후위 순회의 이름은 루트를 언제 처리하는지와 어떻게 연결되는가?
- 자식 노드의 결과를 모아 부모 노드를 계산하려면 루트를 언제 처리해야 하는가?

Three Recursive Orders

같은 이진트리에서 전위 순회, 중위 순회, 후위 순회 출력을 계산한다

세 순서에 사용할 트리



세 순회 모두 왼쪽 자식 노드를 먼저 처리하며 루트를 처리하는 위치만 다르다.

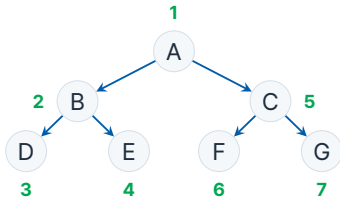
정의

전위 순회(preorder traversal)

루트를 처리한 뒤 왼쪽과 오른쪽 부분 트리에 같은 순서를 적용한다. 순서는 루트 → 왼쪽 → 오른쪽이다.

전위 순회 과정

A를 먼저 처리한 뒤 B 부분 트리를 끝내고 C 부분 트리로 이동한다.



각 부분 트리에서도 그 부분 트리의 루트를 가장 먼저 처리한다.

Adapted from Miller and Ranum (2013), Preorder.

전위 순회 함수

```
def preorder(tree):  
    if tree is not None:  
        visit(tree.key)  
        preorder(tree.left_child)  
        preorder(tree.right_child)
```

Source: Miller and Ranum (2013), Preorder.

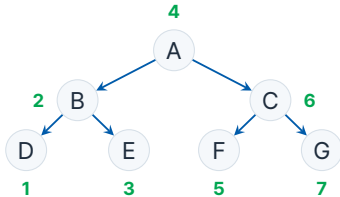
정의

중위 순회(inorder traversal)

왼쪽 부분 트리를 처리한 뒤 루트와 오른쪽 부분 트리 순서로 진행한다. 순서는 왼쪽 → 루트 → 오른쪽이다.

중위 순회 과정

가장 왼쪽 D까지 내려간 뒤 B와 E를 처리하고 A로 돌아온다.



중위 순회 함수

```
def inorder(tree):  
    if tree is not None:  
        inorder(tree.left_child)  
        visit(tree.key)  
        inorder(tree.right_child)
```

Source: Miller and Ranum (2013), Inorder.

후위 순회

정의

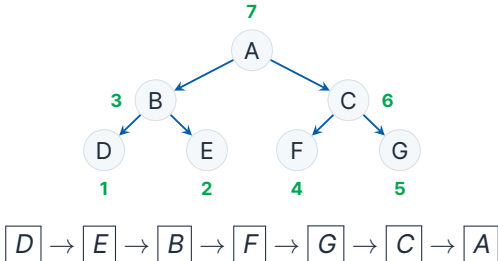
후위 순회(postorder traversal)

왼쪽 부분 트리와 오른쪽 부분 트리를 처리한 뒤 루트를 처리한다. 순서는 왼쪽 → 오른쪽 → 루트다.

Source: Miller and Ranum (2013), Postorder.

후위 순회 과정과 함수

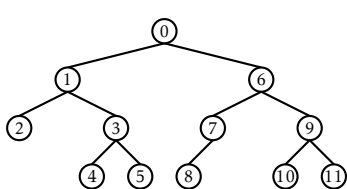
D와 E를 처리한 뒤 B를 처리하고, 마지막에 전체 루트 A를 처리한다.



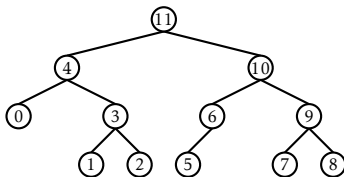
코드에서는 `visit(tree.key)`를 두 재귀 호출 뒤에 둔다.

Source: Miller and Ranum (2013), Postorder.

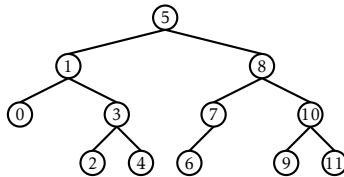
한 트리에 세 가지 번호를 매기면



(가)



(나)



(다)

같은 12노드 트리에 방문 순서를 0부터 적었다. (가), (나), (다)는 각각 어느 순회의 번호인가?

루트의 번호가 0이면 전위 순회, 11이면 후위 순회, 왼쪽 부분 트리의 노드 수와 같으면 중위 순회다.

Checkpoint

Three Recursive Orders

- 예제 트리의 전위 순회, 중위 순회, 후위 순회 출력을 각각 적어 보라.
- 세 재귀 함수에서 실제로 위치가 달라지는 문장은 무엇인가?
- 루트의 번호만 보고 어느 순회의 번호인지 알 수 있는 이유는 무엇인가?

Traversal with a Stack

재귀가 숨긴 방문 대기 상태를 스택으로 직접 관리한다

재귀는 호출 스택을 사용한다

왼쪽 부분 트리를 처리하는 동안 각 호출의 반환 위치와 남은 일을 보관한다.

호출	기다리는 일
preorder(A)	B가 끝난 뒤 C 처리
preorder(B)	D가 끝난 뒤 E 처리
preorder(D)	빈 부분 트리 두 개 확인

명시적 스택으로 전위 순회를 만든다

단계	동작
시작	루트를 스택에 push
반복	pop한 노드를 방문
다음	오른쪽 자식 노드를 먼저, 왼쪽 자식 노드를 나중에 push
종료	스택이 비면 끝

Caution LIFO이므로 왼쪽 자식 노드를 먼저 방문하려면 오른쪽 자식 노드를 먼저 push해야 한다.

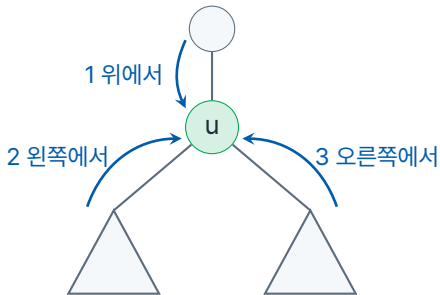
스택 상태를 따라가 본다

pop한 노드	방문 뒤 push	스택 맨 위에서 아래
A	C, B	B, C
B	E, D	D, E, C
D	없음	E, C
E	없음	C
C	G, F	F, G
F	없음	G
G	없음	비어 있음

명시적 전위 순회 코드

```
stack = [root] if root is not None else []  
while stack:  
    u = stack.pop()  
    visit(u.key)  
    if u.right_child:  
        stack.append(u.right_child)  
    if u.left_child:  
        stack.append(u.left_child)
```

각 노드를 일정한 횟수만 본다



n 개 노드의 순회 시간 = $O(n)$

한 노드에는 위에서, 왼쪽에서, 오른쪽에서 최대 세 번 도착한다. 첫째는 전위, 둘째는 중위, 셋째는 후위 순회가 처리하는 시점이다.

출력 순서가 달라도 모든 노드를 한 번씩 처리하는 세 순회의 시간은 $O(n)$ 이다.

추가 공간은 높이에 달려 있다

트리 모양	높이	재귀 호출 깊이
한쪽으로 긴 트리	$n - 1$	$O(n)$
높이가 낮은 트리	$O(\log n)$	$O(\log n)$

Caution 순회 시간은 둘 다 $O(n)$ 이지만 동시에 저장할 대기 상태 수는 다르다. 명시적 스택은 많아야 높이+1개를 담는다.

Checkpoint

Traversal with a Stack

- 스택으로 전위 순회를 할 때 오른쪽 자식 노드를 먼저 push하는 이유는 무엇인가?
- 명시적 스택은 재귀 호출이 기다리던 어떤 일을 대신 저장하는가?
- 순회 시간과 추가 공간은 각각 노드 수와 높이에 어떻게 의존하는가?

Level Order

큐로 얇은 노드부터 왼쪽에서 오른쪽으로 방문한다

레벨 순회

정의

레벨 순회(level order traversal)

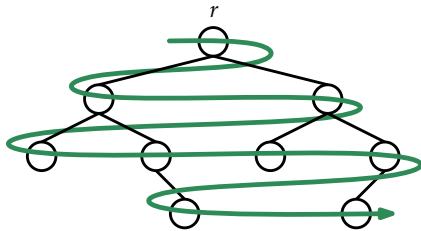
루트에서 가까운 노드를 먼저 방문하는 트리 순회다. 같은 깊이에서는 부모 노드를 방문할 때 큐에 넣은 자식 노드의 순서를 따른다.

Source: Morin (2013), Section 6.1.2, Breadth-First Traversal.

큐 하나로 깊이 순서를 만든다

단계	동작
시작	루트를 큐에 넣음
반복	큐에서 꺼낸 노드를 방문
다음	왼쪽과 오른쪽 자식 노드를 차례로 넣음
종료	큐가 비면 끝

초록 선은 레벨 순회의 방문 순서다. 루트 r 에서 시작해 층마다 왼쪽에서 오른쪽으로 지나가며 빈 자리는 건너뛴다.



큐 상태를 따라가 본다

꺼낸 노드	넣은 노드	큐 앞쪽에서 뒤쪽
A	B, C	B, C
B	D, E	C, D, E
C	F, G	D, E, F, G
D	없음	E, F, G
E	없음	F, G
F	없음	G
G	없음	비어 있음

먼저 넣은 얇은 노드가 더 깊은 자식 노드보다 먼저 나온다.

자식 노드를 넣는 순서가 좌우 순서를 정한다

큐에 넣는 순서	같은 깊이의 방문
왼쪽 뒤 오른쪽	왼쪽에서 오른쪽
오른쪽 뒤 왼쪽	오른쪽에서 왼쪽
임의 순서	그 순서를 그대로 보존

deque로 레벨 순회를 구현한다

```
queue = deque([root] if root is not None else [])
while queue:
    u = queue.popleft()
    visit(u.key)
    if u.left_child:
        queue.append(u.left_child)
    if u.right_child:
        queue.append(u.right_child)
```

큐에는 한 층의 노드가 한꺼번에 들어가므로 추가 공간은 가장 넓은 층의 크기에 달려 있다.

깊이별로 묶으면

깊이	방문 노드
0	A
1	B, C
2	D, E, F, G

레벨 순회 출력은 A, B, C, D, E, F, G이며 방문하는 노드의 깊이는 줄어들지 않는다.

Checkpoint

Level Order

- 레벨 순회에서 루트 다음에 방문하는 노드들은 어떤 깊이에 있는가?
- 같은 깊이를 왼쪽부터 방문하려면 자식 노드를 어떤 순서로 큐에 넣어야 하는가?
- 스택과 큐가 각각 만드는 방문 순서는 어떻게 다른가?

Parse Trees

parse tree에서 후위 순회로 값을 계산하고 중위 순회로 식을 출력하며 작성 과정을 스택으로 추적한다

parse tree

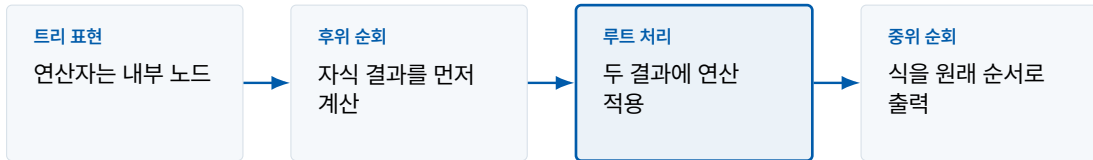
정의

parse tree

parse tree는 연산자를 내부 노드에, 수를 단말 노드에 두어 식의 계산 순서를 나타내는 이진트리다. 한 연산자의 왼쪽과 오른쪽 부분 트리가 두 피연산자를 나타낸다.

Source: Miller and Ranum (2013), Parse Trees.

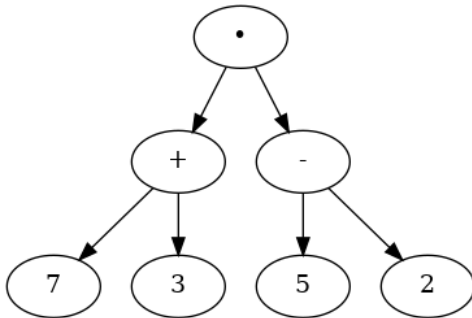
순회 순서가 계산 규칙이 된다



같은 parse tree에서 후위 순회로는 값을 계산하고 중위 순회로는 식을 출력한다.

식의 계층을 트리로 보면

$((7 + 3) \times (5 - 2))$ 에서 곱셈은 두 부분 트리의 계산 결과를 사용한다.



왼쪽 부분 트리는 10, 오른쪽 부분 트리는 3이 되고 루트의 곱셈 결과는 30이다.

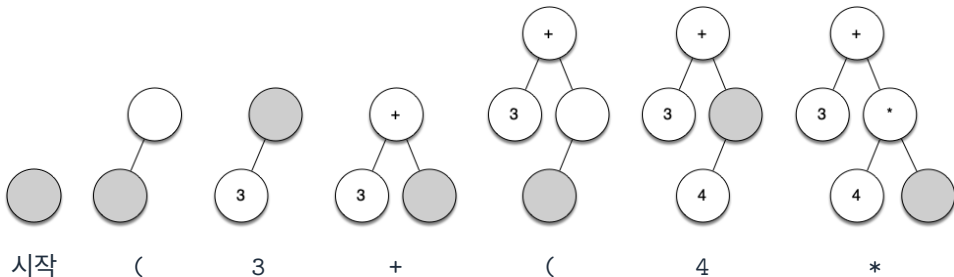
식을 읽으며 parse tree를 만든다

읽은 기호	동작
(	왼쪽 자식 노드를 만들고 내려감
수	현재 노드에 저장하고 부모 노드로 돌아감
연산자	현재 노드에 저장하고 오른쪽 자식 노드를 만들어 내려감
)	부모 노드로 돌아감

Source: Miller and Ranum (2013), Building a Parse Tree.

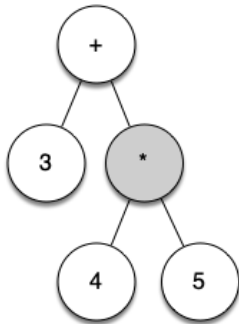
기호마다 parse tree가 자라는 과정

회색 노드는 다음 기호로 채울 현재 위치다.



(3 + (4 × 5))를 한 기호씩 읽은 상태다. (3+까지 읽으면 +가 루트이고 3이 왼쪽 자식 노드에 놓인다.

식을 모두 읽은 뒤의 parse tree



$(3 + (4 \times 5))$ 에서 +의 오른쪽 부분 트리가 곱셈 전체를 나타낸다.

스택으로 부모 노드에 돌아간다

자식 노드로 내려가기 전에 현재 노드를 push하고 식 하나가 끝나면 pop한다.

이동	스택 연산
새 자식 노드로 내려감	현재 노드를 push
수 또는 괄호가 끝남	부모 노드를 pop

값 계산에는 후위 순회를 쓴다

$((7 + 3) \times (5 - 2))$ 의 **parse tree**를 후위 순회로 출력하면 스택으로 계산하던 후위식 $7\ 3\ +\ 5\ 2\ -\ *$ 가 나온다.

노드 종류	처리
단말 노드의 수	그대로 반환
연산자 노드	왼쪽과 오른쪽 결과에 연산 적용

두 자식 노드의 결과가 있어야 부모 노드의 값을 계산하므로 루트를 마지막으로 처리한다.

식으로 출력할 때는 중위 순회를 쓴다

왼쪽 식, 현재 연산자, 오른쪽 식 순서로 출력하면 연산 순서가 드러나는 식이 나온다.

```
print_exp(u) = "(" + print_exp(u.left_child) + u.key + print_exp(u.right_child) + ")"
```

빈 트리는 빈 문자열을 돌려주므로 수에도 괄호가 붙어 $((7 + 3) \times (5 - 2))$ 의 트리는 $(((7 + 3)) * ((5) - (2)))$ 로 출력된다.

Caution 괄호를 함께 출력해야 연산 우선순위를 잃지 않는다.

Checkpoint

Parse Trees

- parse tree에서 연산자와 수는 각각 어떤 노드에 놓이는가?
- 식을 만들 때 스택에 저장하는 것은 무엇인가?
- 값 계산에는 후위 순회, 식 출력에는 중위 순회가 알맞은 이유는 무엇인가?

Which Visit Order Matches the Work?

Next Lecture: Binary Search Trees

- 키의 순서를 유지하는 구조
- ordering invariant
- 탐색 경로와 높이