



부산대학교

PUSAN NATIONAL UNIVERSITY

Tree Terminology

From Links to Hierarchies

Vocabulary, Binary Trees, and Python

DS3600082-140

자료구조

2026학년도 2학기

오용경

데이터사이언스전공

yongkyung.oh@pusan.ac.kr

Trees Around Us

한 시작점에서 갈라지는 계층 구조의 공통 모양을 찾는다

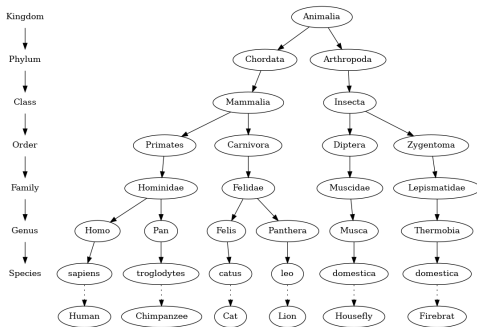
정의

트리(tree)

노드들이 하나의 시작점에서 여러 단계로 갈라지는 계층 구조다. 시작점 아래에서 여러 갈래로 나뉘지만 서로 다른 갈래가 다시 합쳐지지는 않는다. 연결리스트와 달리 한 노드 아래에 여러 다음 노드를 둘 수 있다.

분류 체계도 트리다

넓은 분류에서 시작해 아래로 갈수록 더 구체적인 분류로 나뉜다.



Animalia에서 시작해 분류 단계를 따라가면 각 동물에 도달한다.

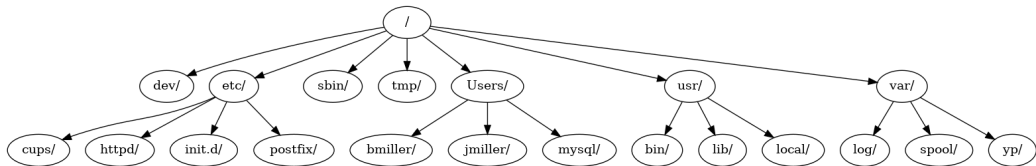
계층 구조의 공통점

모양	뜻
시작점이 하나	같은 시작점에서 모든 노드까지 따라갈 수 있음
아래로 갈라짐	한 노드가 여러 하위 노드를 가질 수 있음
다시 합쳐지지 않음	시작점에서 각 노드까지 이어지는 순서가 하나

Source: Miller and Ranum (2013), Vocabulary and Definitions.

파일 시스템도 같은 모양이다

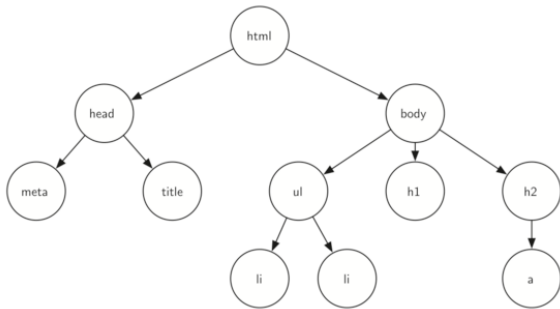
디렉터리 안에 파일과 하위 디렉터리가 반복해서 들어간다.



/에서 시작해 디렉터리 이름을 따라가면 원하는 위치에 도달한다.

HTML 문서도 트리로 읽을 수 있다

태그 안에 다른 태그가 들어가는 중첩 관계가 트리 구조를 만든다.



html 아래에 head와 body가 있고, body 아래에서 다시 ul과 제목 태그가 갈라진다.

디렉터리는 같은 함수를 다시 부른다

```
def walk(folder):  
    for item in list_items(folder):  
        if is_file(item): visit(item)  
        else: walk(item)
```

walk처럼 재귀(recursion)를 사용하는 코드는 하위 디렉터리에도 같은 절차를 적용한다.

연결리스트와 트리의 차이

구조	한 노드의 다음 연결	전체 모양
SLList	최대 1개	한 줄
자식이 최대 둘인 트리	최대 2개	두 갈래씩 분기
일반 트리	여러 개 가능	여러 갈래의 계층

Adapted from Miller and Ranum (2013), Vocabulary and Definitions.

트리인지 확인하는 기준

확인할 것	트리의 조건
시작 노드 수	하나
시작점 이외 노드의 위쪽 연결 수	하나
시작점에서 한 노드로 가는 순서의 수	하나
갈라진 갈래가 다시 합쳐짐	없음

Source: Miller and Ranum (2013), Vocabulary and Definitions.

Checkpoint

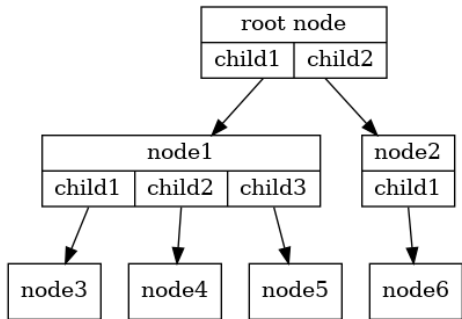
Trees Around Us

- 연결리스트와 트리는 한 노드 아래의 연결 수가 어떻게 다른가?
- 파일 시스템에서 하위 디렉터리에 같은 함수를 다시 적용할 수 있는 이유는 무엇인가?
- 트리에서 시작점부터 각 노드까지 이어지는 순서는 몇 개인가?

Tree Vocabulary

트리의 위치와 관계를 나타내는 용어를 한 그림에 적용한다

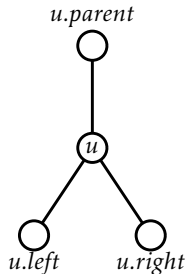
한 그림에서 관계를 읽는다



루트 아래에서 자식 노드가 갈라지고, node3부터 node6까지는 자식 노드가 없다.

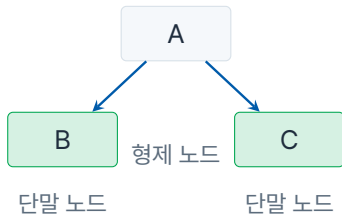
루트·부모 노드·자식 노드

용어	뜻
루트(root)	부모 노드가 없는 유일한 노드
부모 노드(parent)	한 노드의 바로 위에 연결된 노드
자식 노드(child)	한 노드의 바로 아래에 연결된 노드



노드 u 를 기준으로 위의 노드가 부모 노드이고, 아래 두 노드가 자식 노드다.

형제 노드와 단말 노드



형제 노드(sibling) B와 C는 같은 부모 노드에서 나오며 단말 노드(leaf)는 자식 노드가 없다.

정의

부분 트리(subtree)

한 노드와 그 아래의 모든 자손으로 이루어진 트리다. 트리 안의 부분 트리도 다시 트리어므로 같은 정의와 알고리즘을 적용할 수 있다.

깊이

정의

깊이(depth)

루트에서 해당 노드까지의 연결 수다. 루트의 깊이는 0이고, 자식 노드의 깊이는 부모 노드의 깊이보다 1 크다.

$$\text{depth}(u) = \text{depth}(\text{parent}(u)) + 1$$

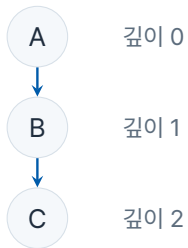
정의

높이(height)

해당 노드에서 가장 먼 단말 노드까지의 연결 수다. 단말 노드의 높이는 0이고, 트리의 높이는 루트의 높이이다.

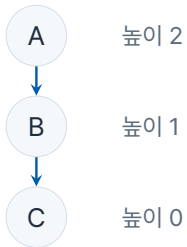
$$\text{height}(u) = 1 + \max_{v \in u \text{의 자식 노드}} \text{height}(v)$$

깊이는 루트에서 아래로 센다



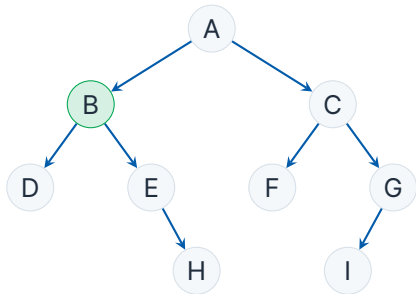
같은 트리에서도 깊이는 어느 노드를 묻는지에 따라 달라진다.

높이는 단말 노드에서 위로 센다



루트에서 가장 먼 단말 노드까지 연결이 두 개이므로 이 트리의 높이는 2다.

한 트리에서 깊이와 높이를 함께 구한다



노드	깊이	높이
A	0	3
B	1	2
D	2	0
E	2	1
H	3	0

B의 높이는 짧은 D 쪽이 아니라 긴 E 쪽을 따라 $1 + \max(0, 1) = 2$ 다.

Checkpoint

Tree Vocabulary

- 루트, 부모 노드, 자식 노드, 단말 노드는 각각 어떤 노드를 뜻하는가?
- 깊이와 높이는 각각 어느 방향의 거리를 재는가?
- 부분 트리에도 같은 트리 알고리즘을 다시 적용할 수 있는 이유는 무엇인가?

Binary Trees

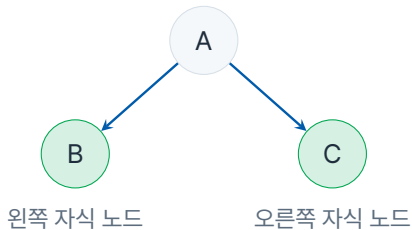
두 자식 노드의 위치가 구분되는 이진트리를 재귀적으로 정의한다

정의

이진트리(binary tree)

각 노드가 최대 두 자식 노드를 가지는 트리다. 두 자식 노드는 왼쪽 자식 노드와 오른쪽 자식 노드로 구분되며 한쪽만 있을 수도 있다.

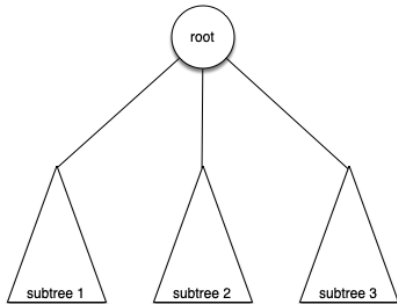
왼쪽과 오른쪽은 서로 다른 자리다



Caution B와 C를 바꾸면 노드 수는 같아도 서로 다른 이진트리가 된다.

트리는 자신 안에 다시 트리를 가진다

루트 아래의 각 갈래를 하나의 부분 트리로 보면 같은 구조가 반복된다.



각 삼각형은 그 안에 다시 루트와 부분 트리를 가질 수 있다.

이진트리의 재귀 정의

경우	이진트리
빈 경우	노드가 없는 빈 트리
빈 경우가 아님	루트와 왼쪽 부분 트리, 오른쪽 부분 트리

빈 트리를 포함해야 단말 노드의 왼쪽·오른쪽 부분 트리도 빈 트리로 표현할 수 있다.

높이도 부분 트리의 결과를 합친다

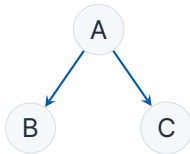
빈 부분 트리의 높이를 -1 로 두면 단말 노드의 높이가 0 이 된다.

$$\text{height}(u) = \begin{cases} -1, & u \text{가 빈 트리이면} \\ 1 + \max\{\text{height}(u.\text{left}), \text{height}(u.\text{right})\}, & \text{그 밖에는} \end{cases}$$

같은 노드 수라도 높이는 다르다



높이 2



높이 1

노드 수만으로 높이를 정할 수 없고 트리의 모양을 함께 보아야 한다.

이진트리는 아직 BST가 아니다

이진트리에서 정한 것	아직 정하지 않은 것
자식 노드 수는 최대 두 개	키의 크기 순서
왼쪽과 오른쪽 자리 구분	탐색할 때 어느 쪽으로 갈지
재귀적인 부분 트리 구조	균형과 높이 제한

Caution 왼쪽 값이 작고 오른쪽 값이 크다는 조건은 모든 이진트리의 성질이 아니다.

Checkpoint

Binary Trees

- 왼쪽 자식 노드와 오른쪽 자식 노드를 바꾸면 왜 다른 이진트리가 되는가?
- 빈 트리를 재귀 정의에 포함해야 하는 이유는 무엇인가?
- 노드 수가 같은 두 이진트리의 높이가 달라질 수 있는 이유는 무엇인가?

Trees in Python

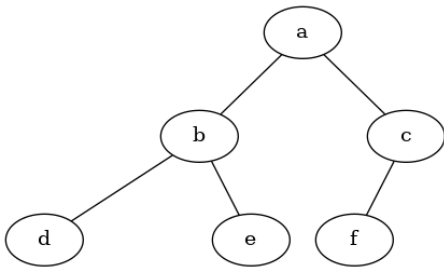
중첩 리스트와 노드 참조로 같은 이진트리를 표현한다

표현 방법은 두 가지다

표현	한 부분 트리의 모양
중첩 리스트	<code>[root, left, right]</code>
노드와 참조	<code>key, left_child, right_child</code>

메모리 모양은 달라도 루트와 두 부분 트리라는 재귀 구조는 같다.

작은 트리를 중첩 리스트로 바꾸면



이 트리는 `[a, [b, [d, [], []], [e, [], []]], [c, [f, [], []], []]]`로 나타낼 수 있다.

세 인덱스의 역할은 고정된다

인덱스	내용	예
0	루트 값	tree[0]
1	왼쪽 부분 트리	tree[1]
2	오른쪽 부분 트리	tree[2]

Source: Miller and Ranum (2013), List of Lists Representation.

단말 노드도 같은 리스트 모양을 쓴다

노드	중첩 리스트	뜻
단말 노드 A 빈 트리	["A", [], []] []	두 부분 트리가 비어 있음 루트가 없음

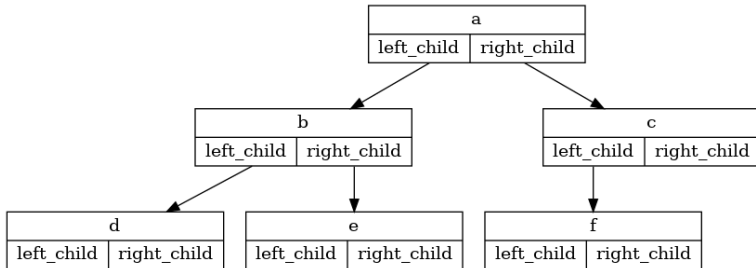
특별한 단말 노드 형식을 새로 만들지 않고 빈 부분 트리 두 개로 표현한다.

왼쪽에 새 노드를 끼워 넣으면

기존 왼쪽 부분 트리를 버리지 않고 새 노드의 왼쪽 부분 트리로 내린다.

단계	<code>insert_left(root, x)</code>
1	기존 <code>root[1]</code> 을 꺼낸다
2	<code>[x, old_left, []]</code> 을 만든다
3	새 리스트를 <code>root[1]</code> 에 넣는다

노드와 참조로 표현하면



각 노드는 값과 왼쪽·오른쪽 자식 노드를 가리키는 참조를 가진다.

BinaryTree 노드의 세 필드

```
1 class BinaryTree:
2     def __init__(self, root_obj):
3         self.key = root_obj
4         self.left_child = None
5         self.right_child = None
```

key는 현재 노드의 값이고, left_child와 right_child는 자식 노드를 가리키며 자식 노드가 없으면 None이다. 앞의 노드 관계 그림과 높이 식은 두 참조를 u.left, u.right로 적는다.

참조를 따라 부분 트리에 접근한다

표현	접근 결과
<code>tree.key</code>	루트 값
<code>tree.left_child</code>	왼쪽 부분 트리의 루트
<code>tree.right_child</code>	오른쪽 부분 트리의 루트
<code>tree.left_child.key</code>	왼쪽 자식 노드가 있을 때 그 값

Checkpoint

Trees in Python

- 중첩 리스트에서 인덱스 0, 1, 2는 각각 무엇을 저장하는가?
- 기존 왼쪽 부분 트리가 있을 때 새 왼쪽 자식 노드를 넣으면 기존 부분 트리는 어디로 가는가?
- 노드와 참조 표현에서 자식 노드가 없는 자리는 무엇으로 나타내는가?

What Makes a Tree Recursive?

Next Lecture: Tree Traversals

- 전위·중위·후위 순회
- 레벨 순회와 큐
- parse tree에서의 순회