
D.COM
ALGORITHM
STUDY
2020 - 02 VER.
6강) 트리

6강.
트-리

오늘의 목표

자료구조 중 하나인
트리의 **용어**와 **종류** 그리고
구현하는 방법에 대해 알아봅시다!

#그래프와 유사합니다!

가볍게,
읽을거리

;

링크

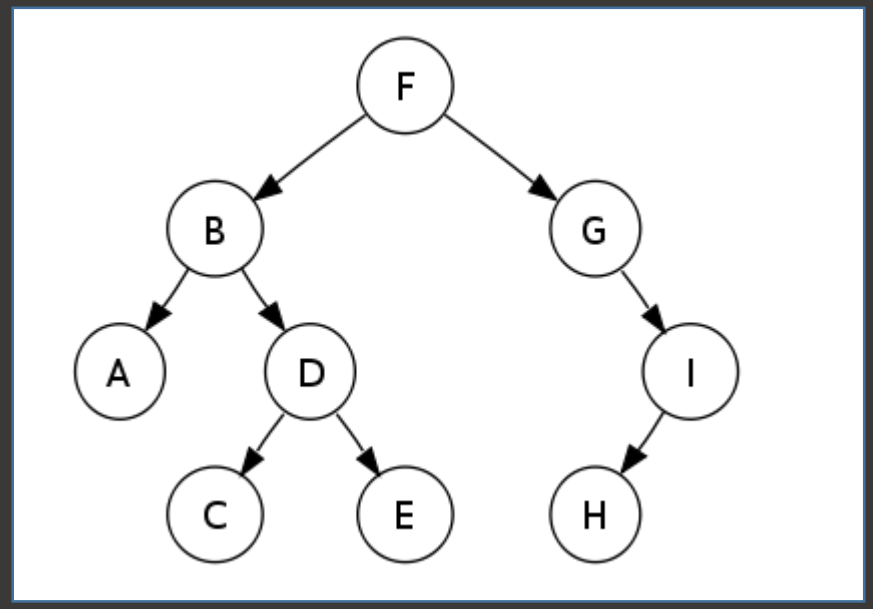
더 나은 개발자가 되는 8가지 방법[번역]

"저는 프로그래머 되기"

트리

Tree

트리



그래프의 일종으로
사이클이 없는 연결 그래프입니다.

다음 슬라이드에서 좀 더 자세히 알아봅시다!

사이클이 없는 연결 그래프입니다.

사이클이 없는 연결 그래프입니다.

모든 정점에서 다른 모든 정점으로 가는 경로가 유일합니다!
정점의 개수가 V라면, 간선의 개수는 V-1개라는 특징을 이용하면 쉽게 사이클이 없는지 판단할 수 있습니다.

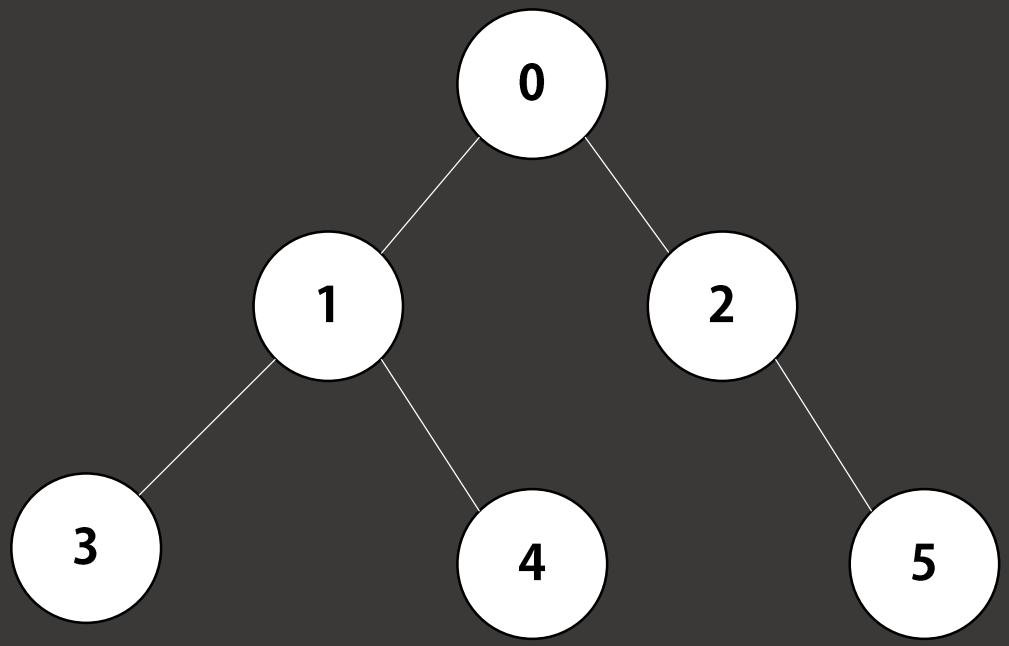
사이클이 없는 연결 그래프입니다.

모든 정점에서 다른 모든 정점으로 가는 경로가 존재합니다!
즉, 연결 요소가 1개입니다.

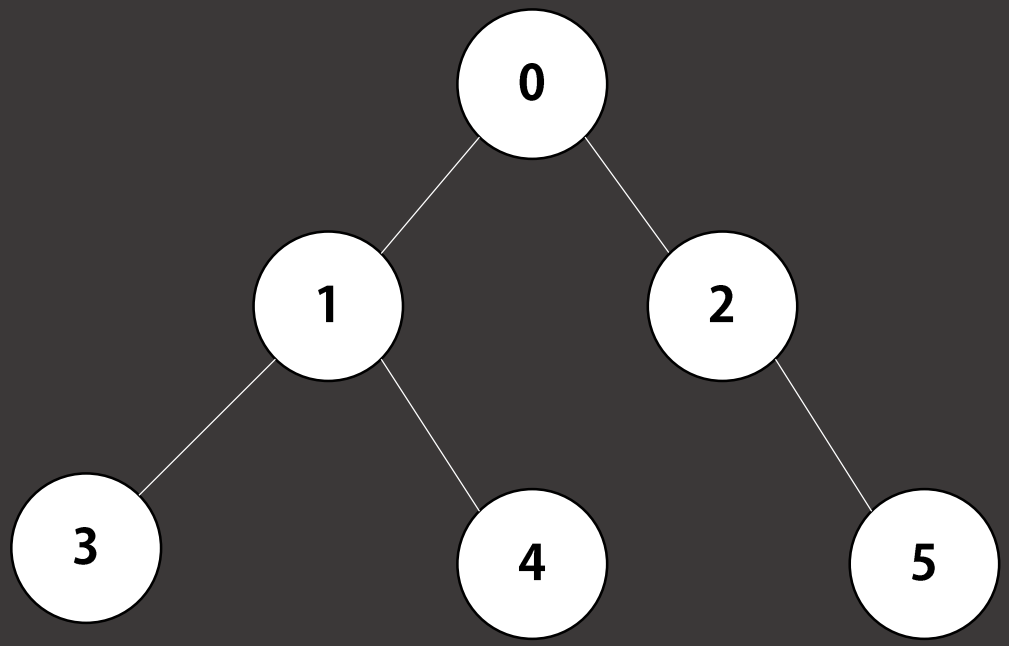
다시 한번 트리는,

사이클이 없는 연결 그래프입니다.

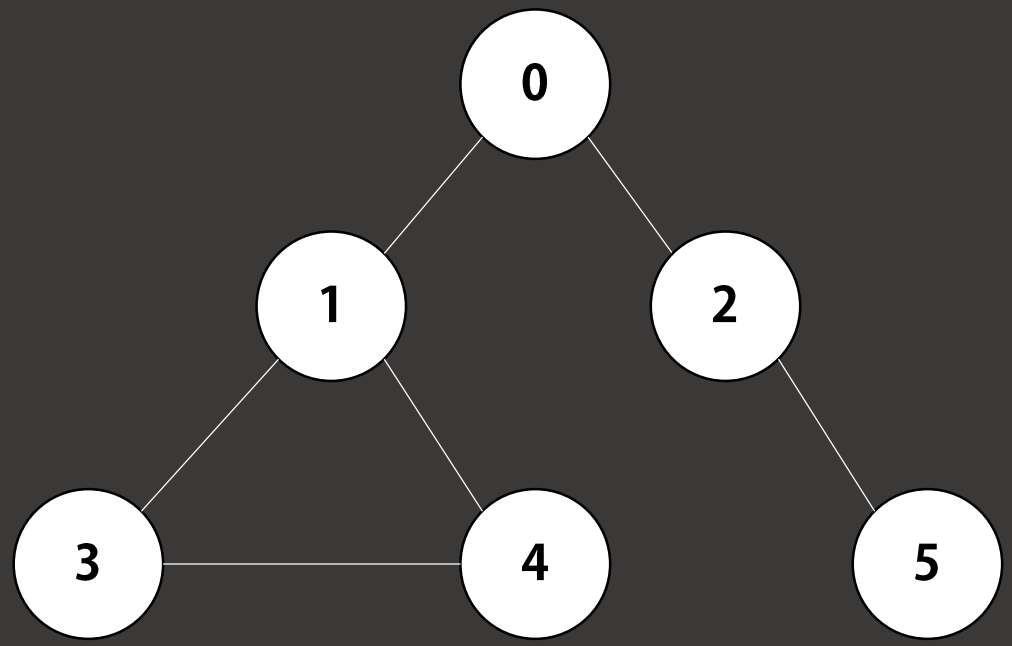
다음 자료구조가 트리 구조인지
판단해봅시다!



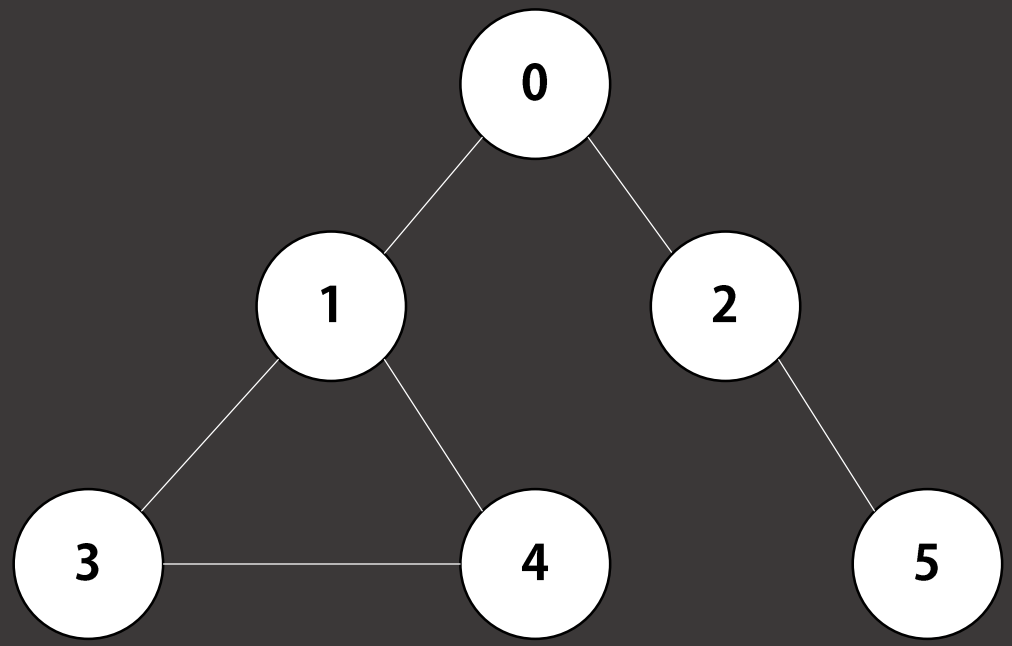
위 자료구조는 **트리**일까요?



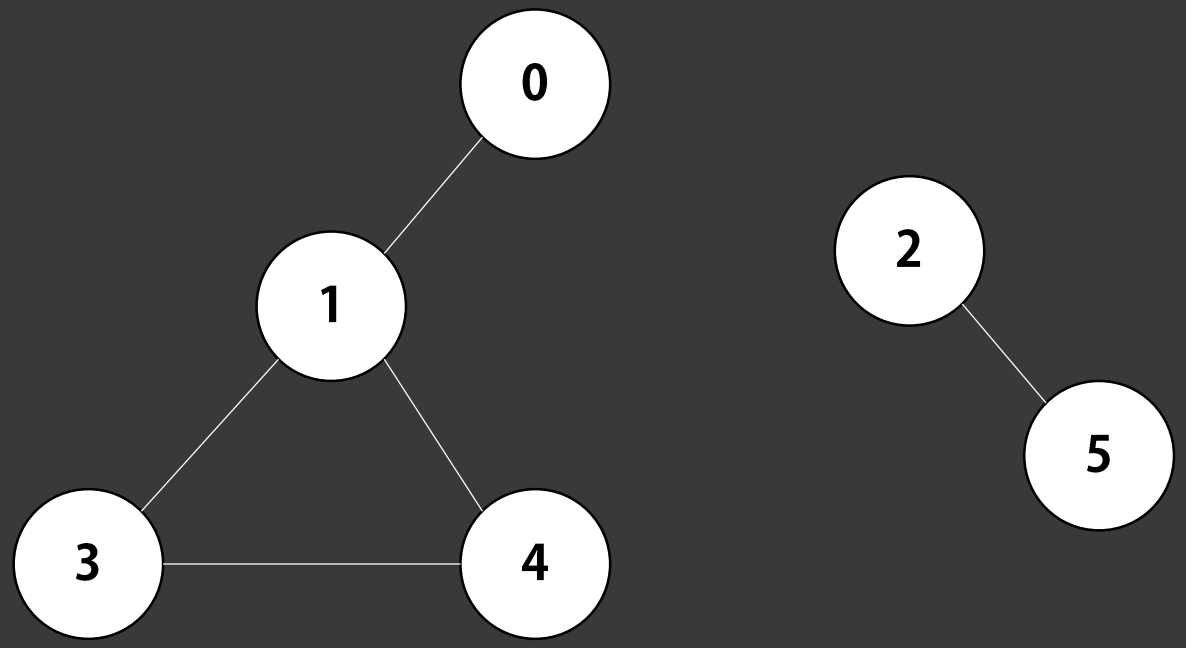
정점의 개수가 6개, 간선의 개수가 5개로 사이클이 없습니다!
또한 연결 그래프이므로 트리가 맞습니다!



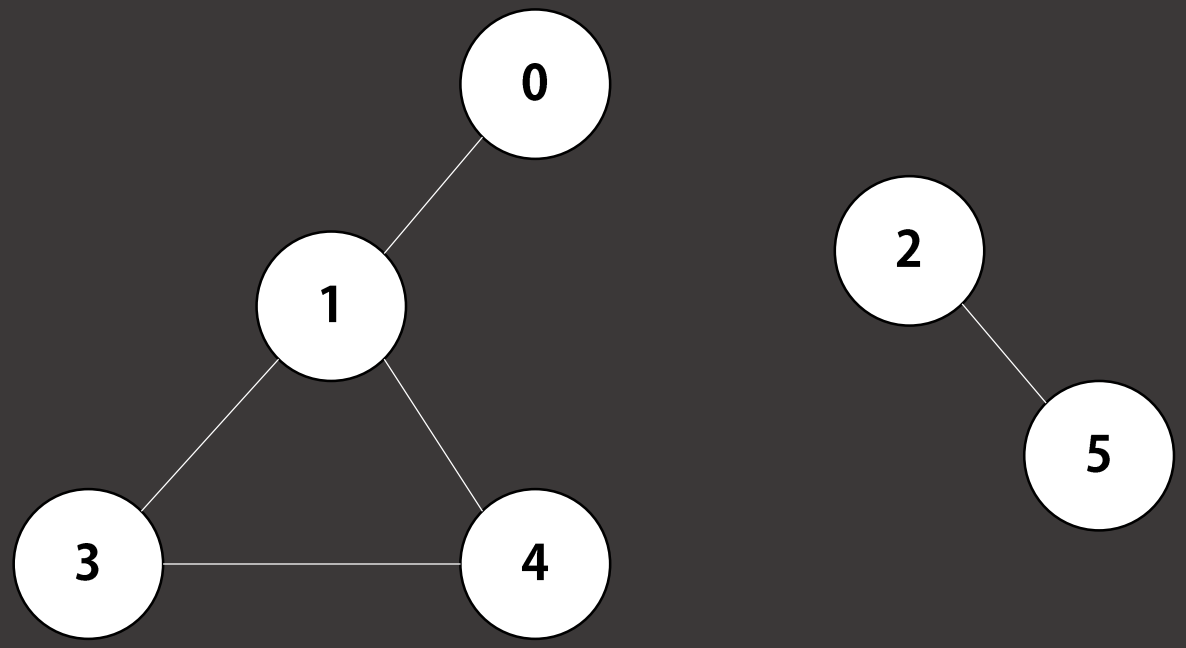
위 자료구조는 **트리**일까요?



연결 그래프이지만,
정점의 개수가 6개 간선의 개수가 6개이므로 사이클이 존재합니다.
따라서 트리가 아닙니다.



위 자료구조는 **트리**일까요?

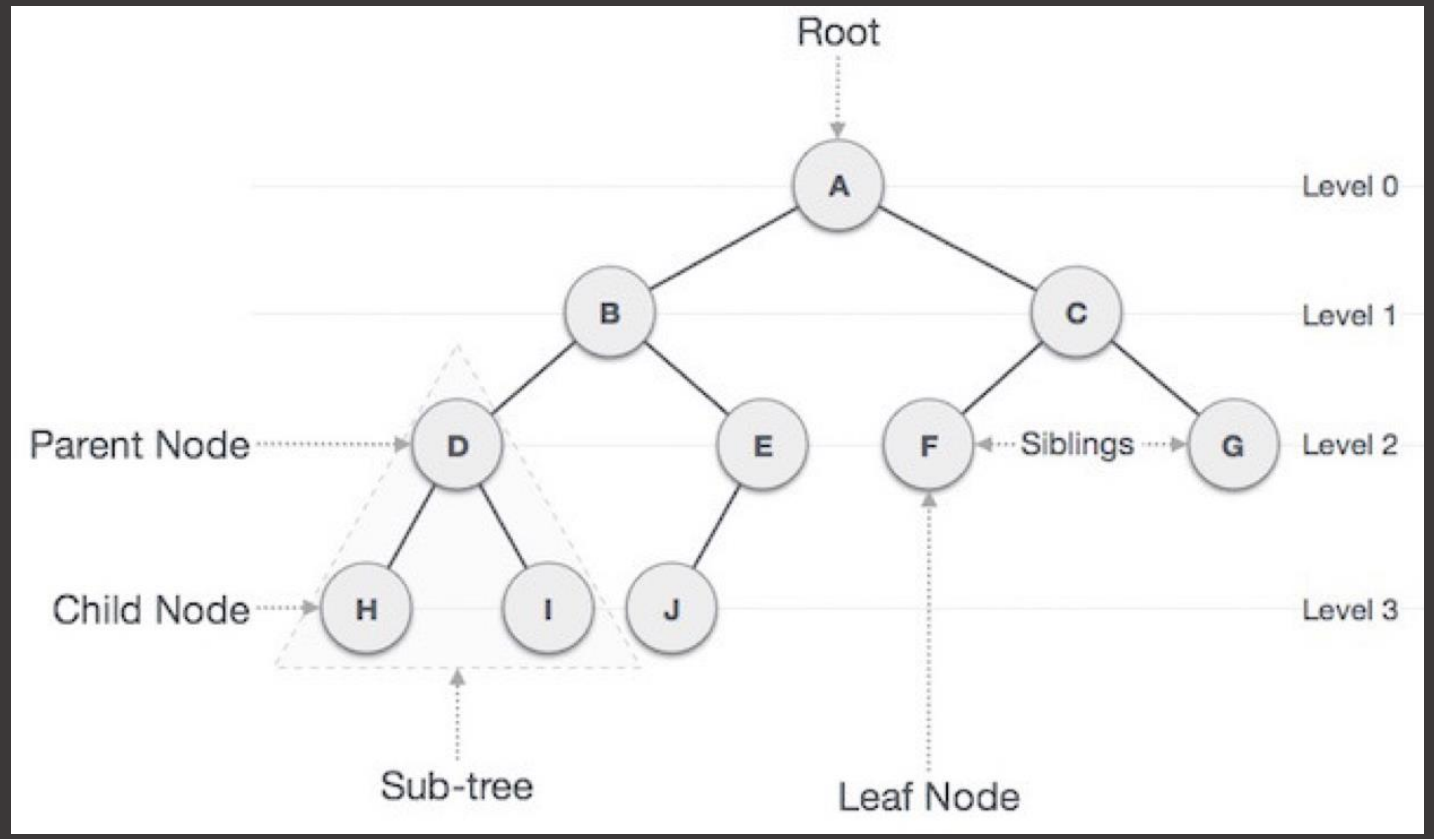


우선 연결 그래프가 아닙니다.
또한 정점의 개수가 6개, 간선의 개수가 5개 이지만
연결 그래프가 아니기 때문에 사이클이 존재합니다.
따라서 트리가 아닙니다.

이번에는 트리에 쓰이는

용어를 알아봅시다.

루트 정점
단말 정점
내부 정점
부모
자식
형제
깊이
높이
조상
자손
.
.



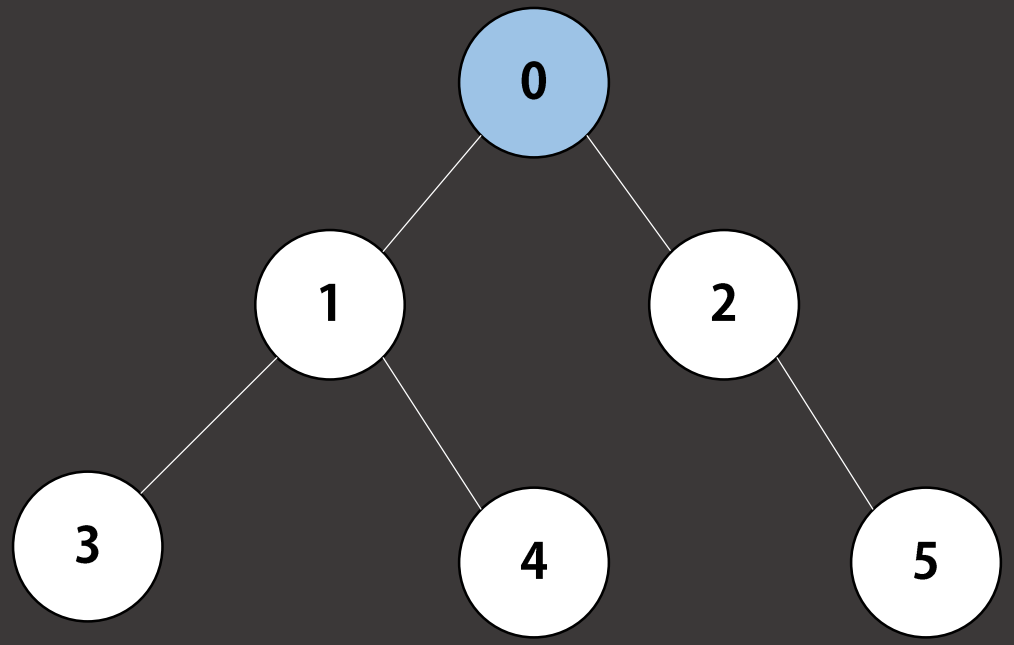
용어가 있습니다!

루트 정점 (Root node)

부모가 없는 정점을 의미합니다.
트리는 단 하나의 루트 정점을 가지거나
가지지 않을 수 있습니다.

예를 들어 오른쪽 트리는
0이라는 루트 정점을 가진다고
'지정'할 수 있습니다.

#즉, 트리 만든 사람 마음대로 입니다!
#0 이외의 다른 정점들도 루트 정점이 될 수 있습니다!

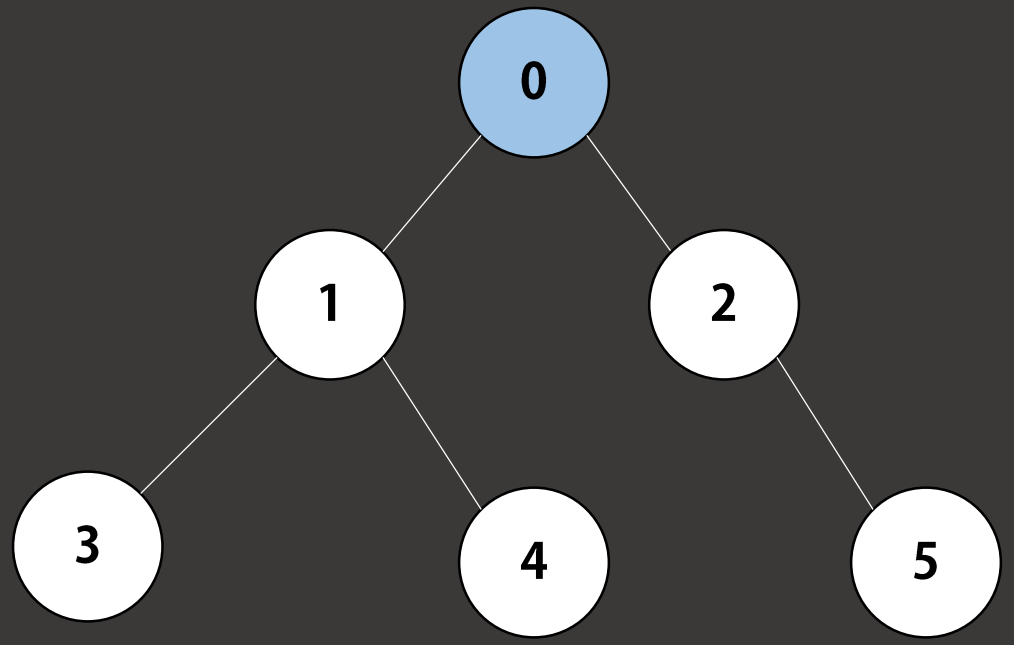


부모 (Parent)

만약 루트 정점을 정했다면
트리의 '방향'을 정할 수 있습니다.

어떤 정점의 위 방향에 있는 바로 인접한 정점들을
부모 관계에 있다고 합니다.

만약 0을 루트 정점이라 했을 때,
예를 들어 1의 부모는 0이고,
5의 부모는 2입니다.

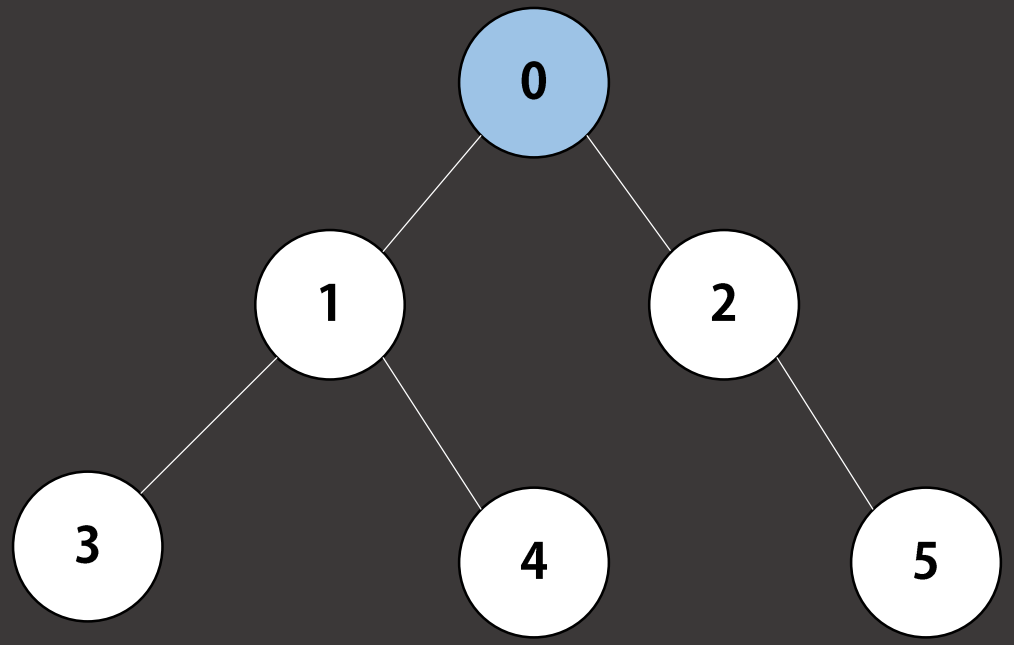


자식 (Children)

만약 루트 정점을 정했다면
트리의 '방향'을 정할 수 있습니다.

어떤 정점의 아래 방향에 있는 바로 인접한 정점들을
자식 관계에 있다고 합니다.

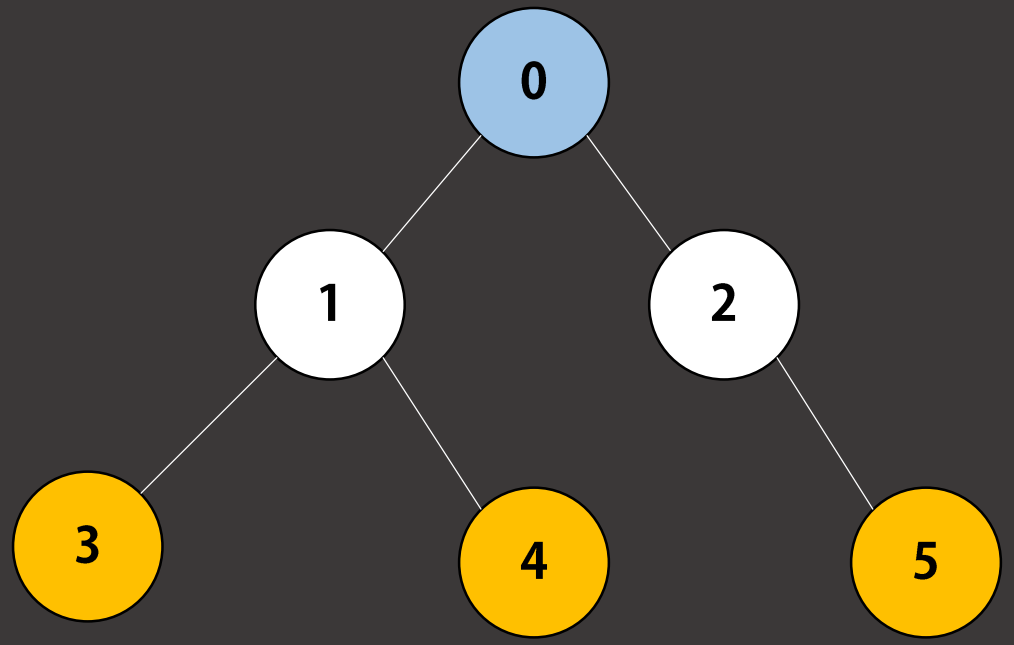
만약 0을 루트 정점이라 했을 때,
예를 들어 1의 자식은 3과 4이고,
2의 자식은 5입니다.
그리고 0의 자식은 1과 2입니다.



단말 정점 (Leaf node)

자식이 없는 정점을 의미합니다.
말단 정점, 잎 정점이라고도 합니다.

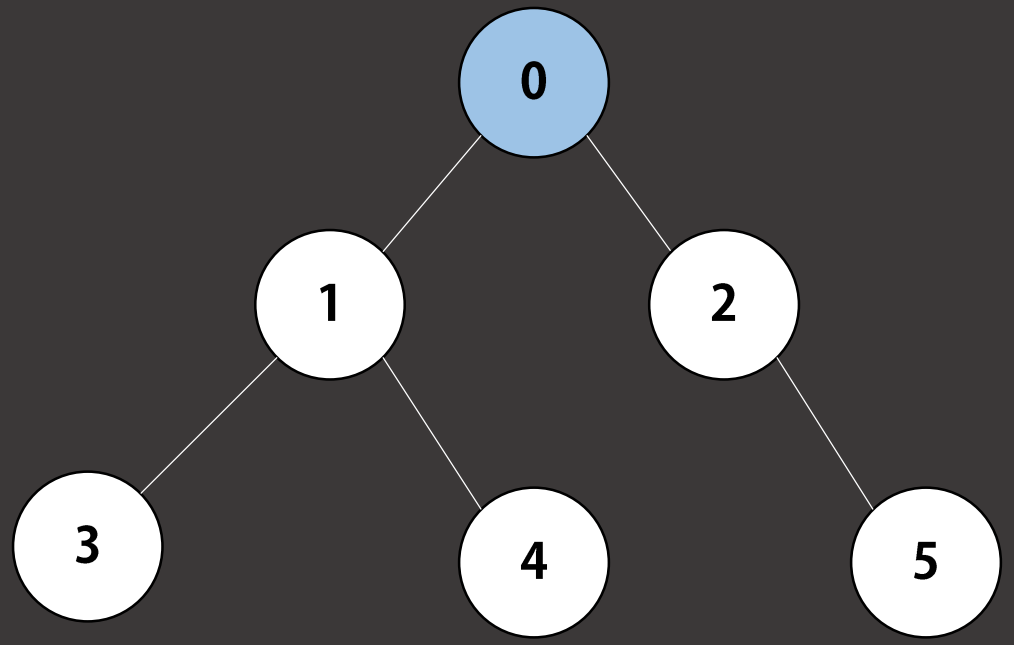
만약 0을 루트 정점이라 했을 때,
3, 4, 5가 단말 정점입니다.



형제 (Sibling)

같은 부모를 가지는 정점들을 의미합니다.

예를 들어
1과 2는 형제이고,
3과 4는 형제입니다.

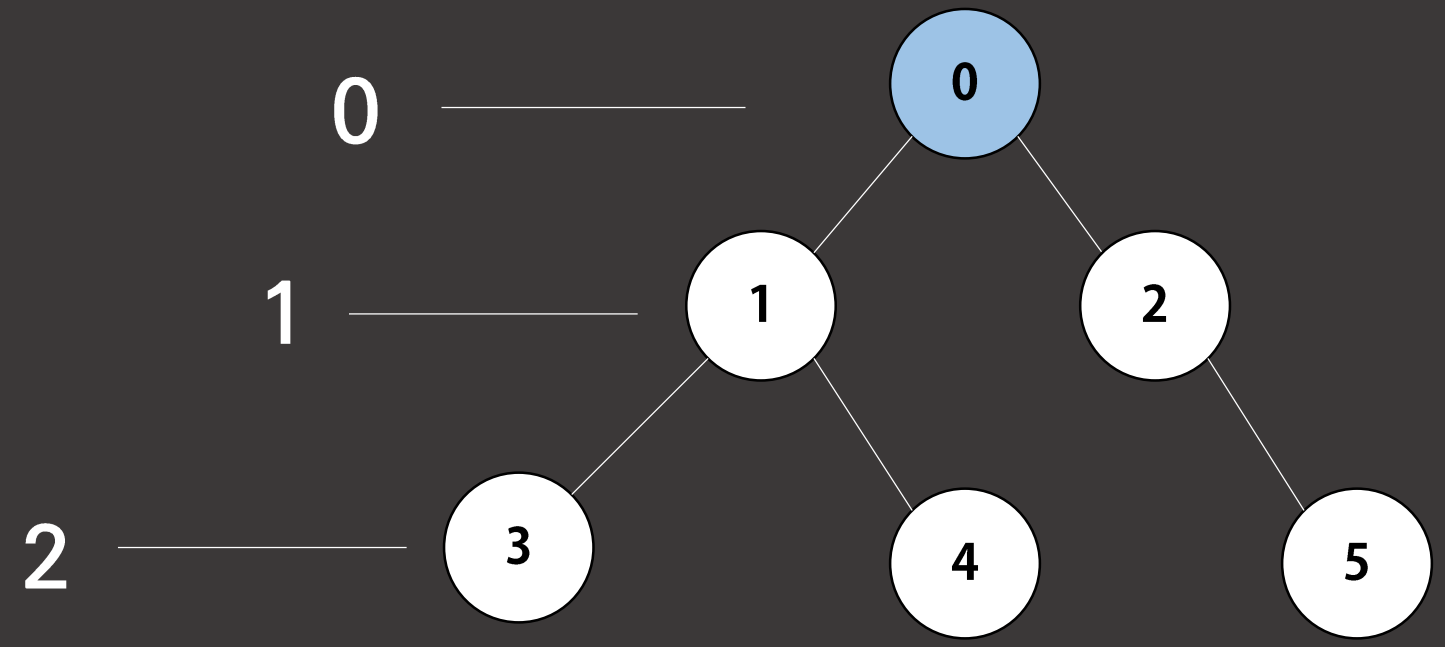


깊이 (Depth)

루트 정점으로부터의 거리를 의미합니다.

3번 정점의 깊이는 2이고,
1번 정점의 깊이는 1입니다.

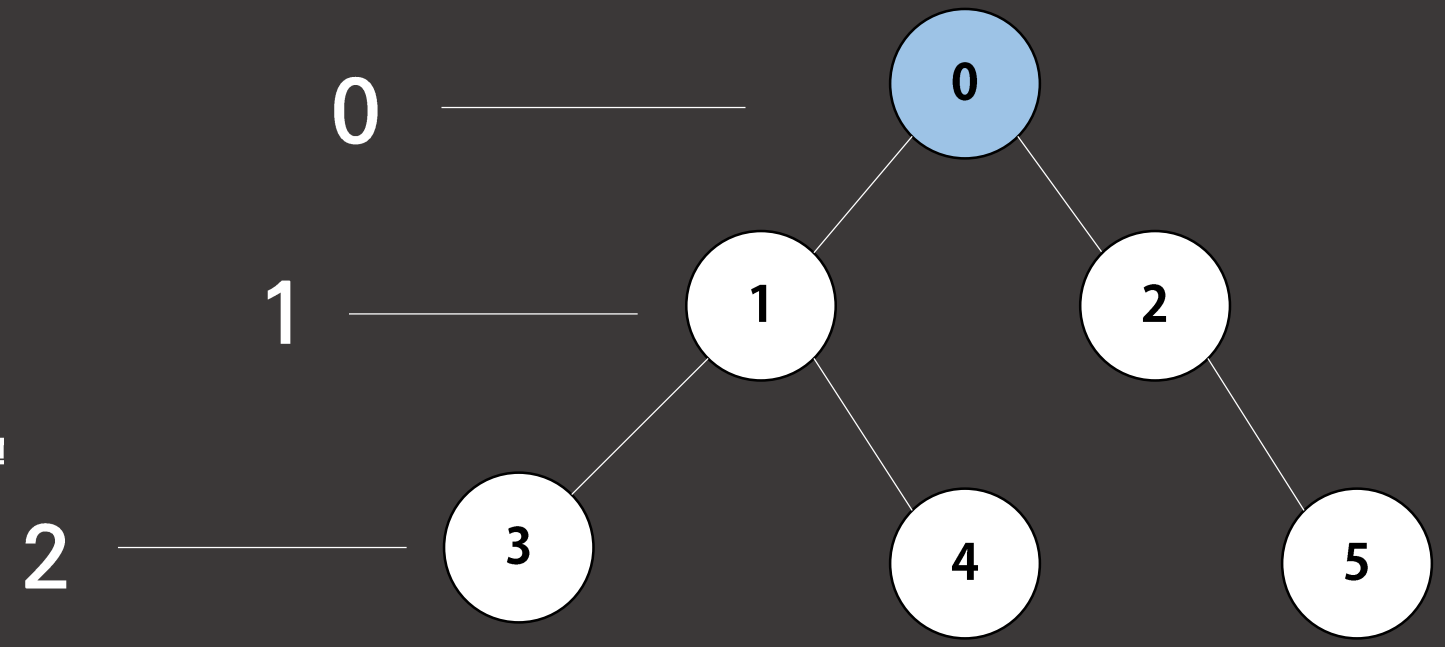
루트 정점의 깊이를 1로 놓는 경우도 있습니다!



높이 (Height)

깊이 중 가장 큰 값을 의미합니다.
오른쪽 그래프의 높이는 2입니다.

깊이와 마찬가지로 높이가 1부터 시작하는 경우도 있습니다!

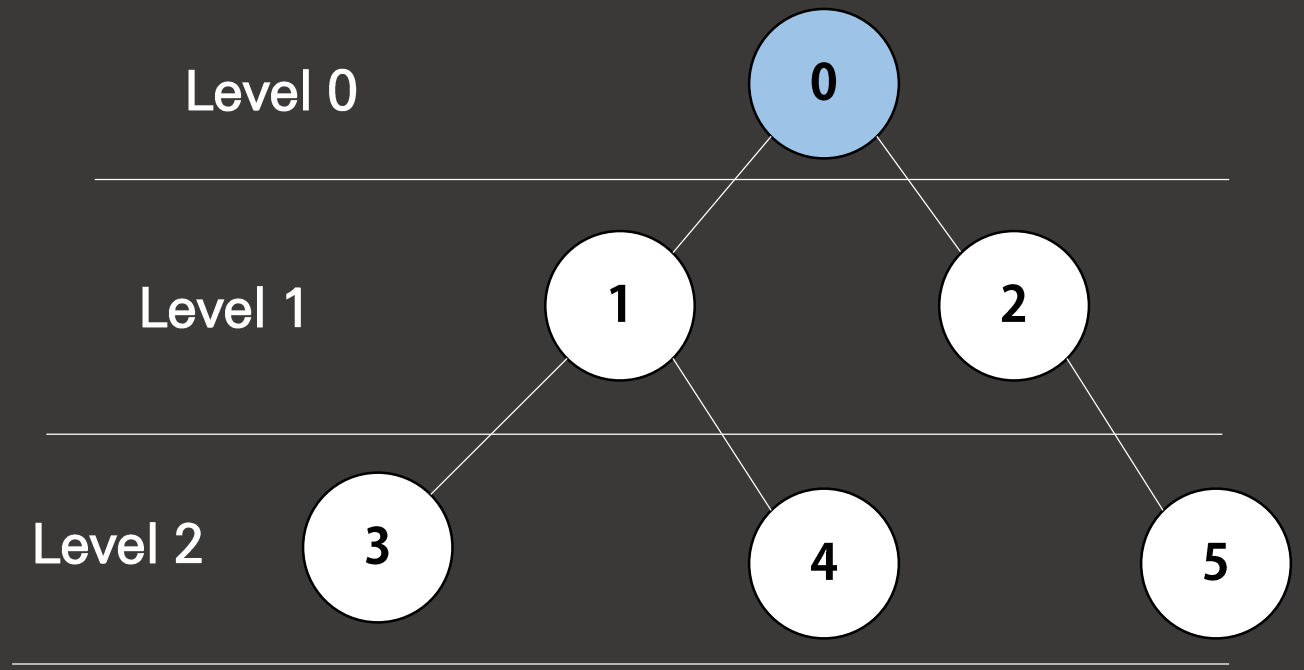


레벨 (Level)

트리의 특정 깊이를 가지는
정점의 집합입니다!

예를 들어 레벨 2 정점은
3, 4, 5입니다.

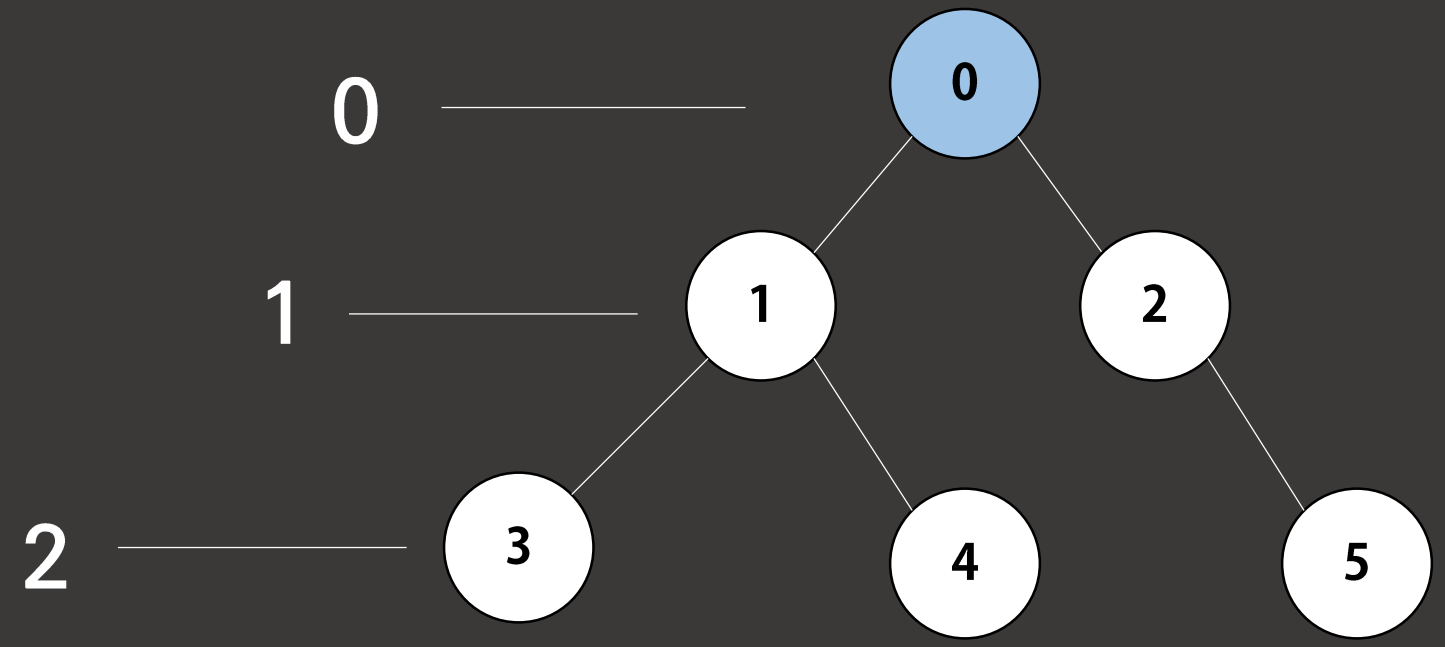
#Level 1부터 시작하는 경우도 있습니다.



조상 (Ancestor)

특정 정점에서 부모 정점들의 총 집합입니다.

예를 들어 3번 정점의 조상은
0, 1입니다.



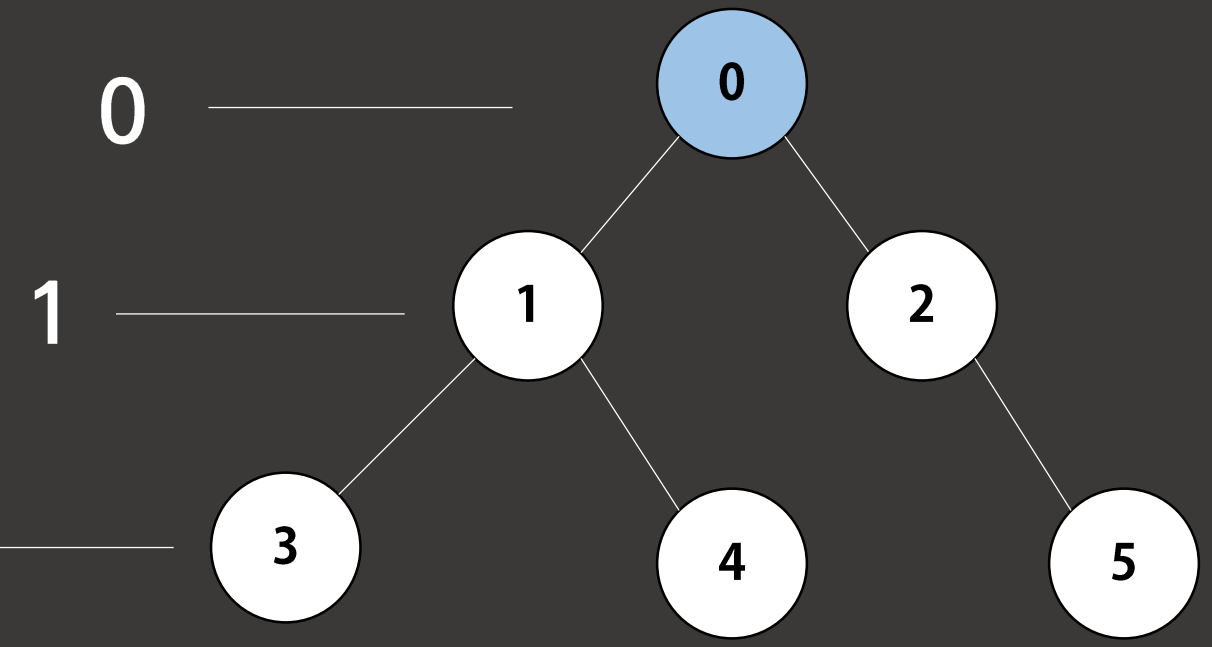
자손 (Descendent)

특정 정점에서 자식 정점들의 총 집합입니다.

예를 들어 0번 정점의 자손은
1, 2, 3, 4, 5입니다.

1번 정점의 자손은 3, 4 입니다.

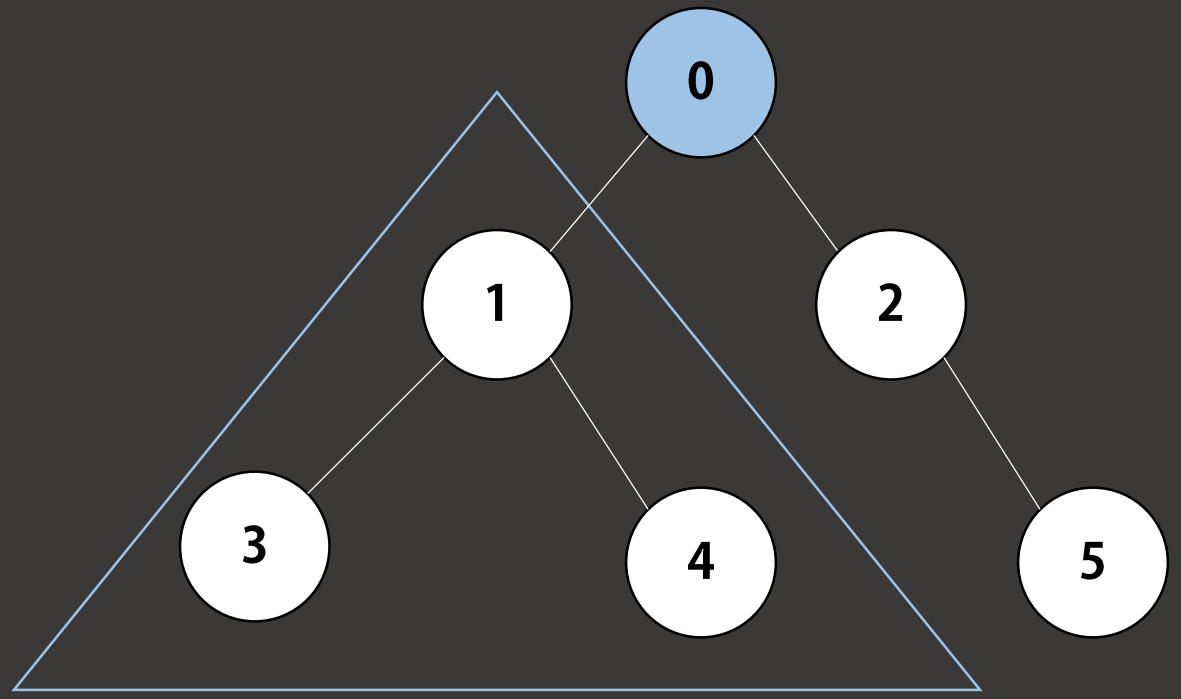
<u>#부모와 조상, 자식과 자손의 차이점을 알아 두세요!</u>



부 트리 (Sub tree)

간단하게, 트리 안에 존재하는 트리를
서브 트리라고 합니다.

예를 들어 다음 파란색 삼각형 안에 있는 트리를
서브 트리라고 할 수 있습니다.



트리 중에서도 자주 사용되는

이진 트리에 대하여 알아봅시다!

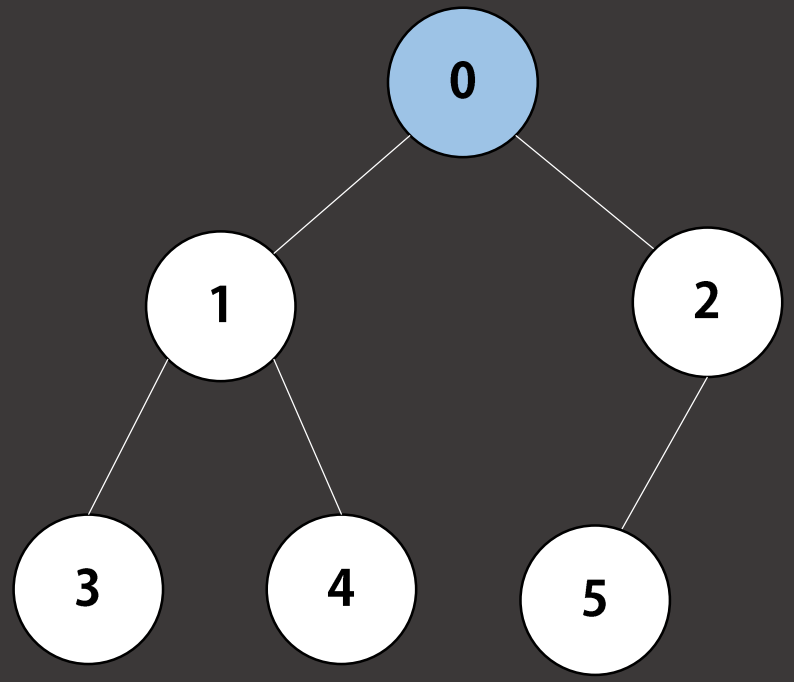
이진 트리 (binary tree)

간단하게, 각 정점이 최대 2개의 자식을 가질 수 있는 트리입니다!

이진 트리에도,
완전 이진 트리, 포화 이진 트리, 정 이진 트리 등 다양한 종류가 있습니다.

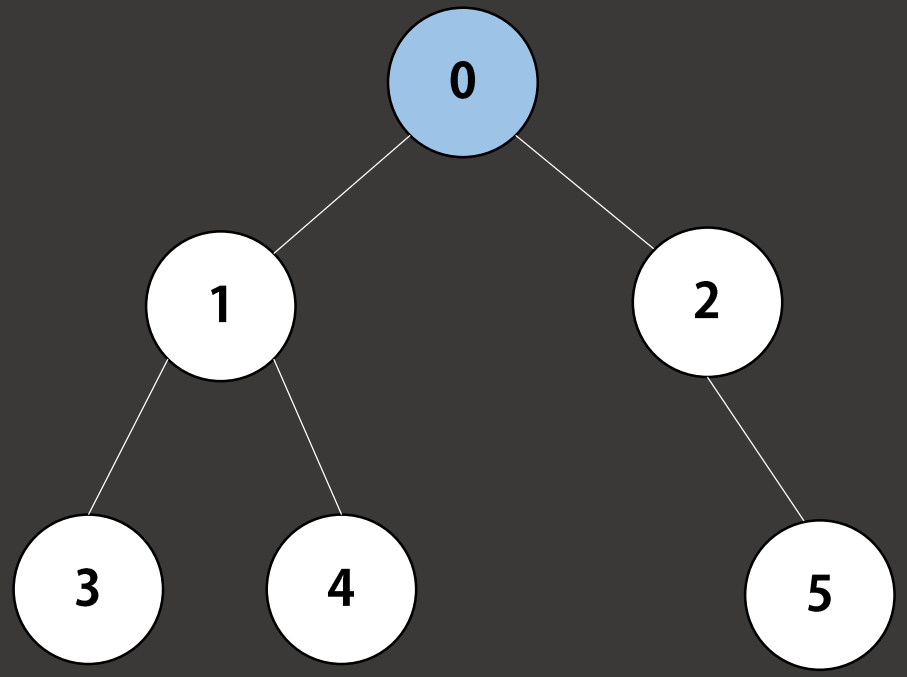
완전 이진 트리 (complete binary tree)

마지막 레벨을 제외하고
모든 정점의 자식 정점이 2개인 트리입니다.
이때, 마지막 레벨의 정점은
왼쪽부터 채워져야 합니다!



완전 이진 트리 (complete binary tree)

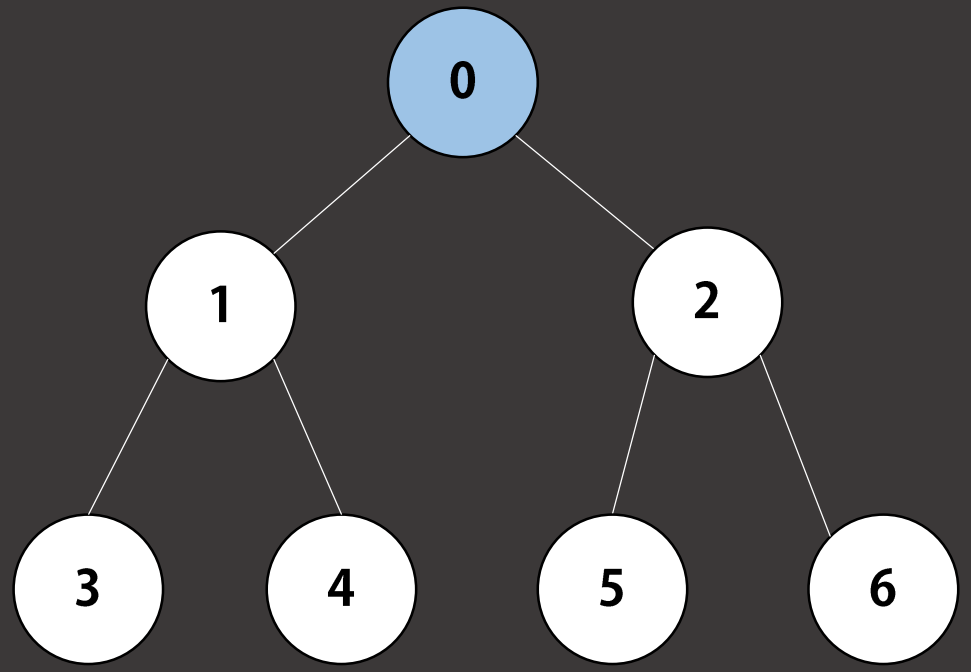
즉, 다음 트리는
완전 이진 트리가 아닙니다!



포화 이진 트리 (perfect binary tree)

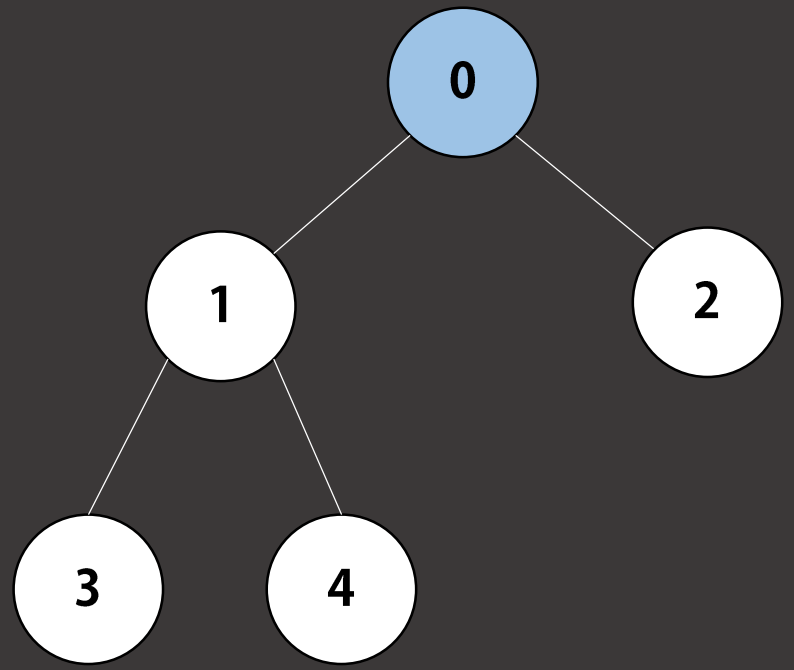
마지막 레벨을 제외하고
모든 정점의 자식 정점이 2개인 트리입니다.
단, 이때 모든 단말 정점의 레벨이 같습니다!

#포화 이진 트리는 완전 이진 트리에 포함됩니다!



정 이진 트리 (Full binary tree)

마지막 레벨을 제외하고
모든 정점의 자식 정점이 2개인 트리입니다.
단, 1개의 자식을 가지는 정점이 없습니다!



그렇다면 트리는 어떻게
구현할 수 있을까요?

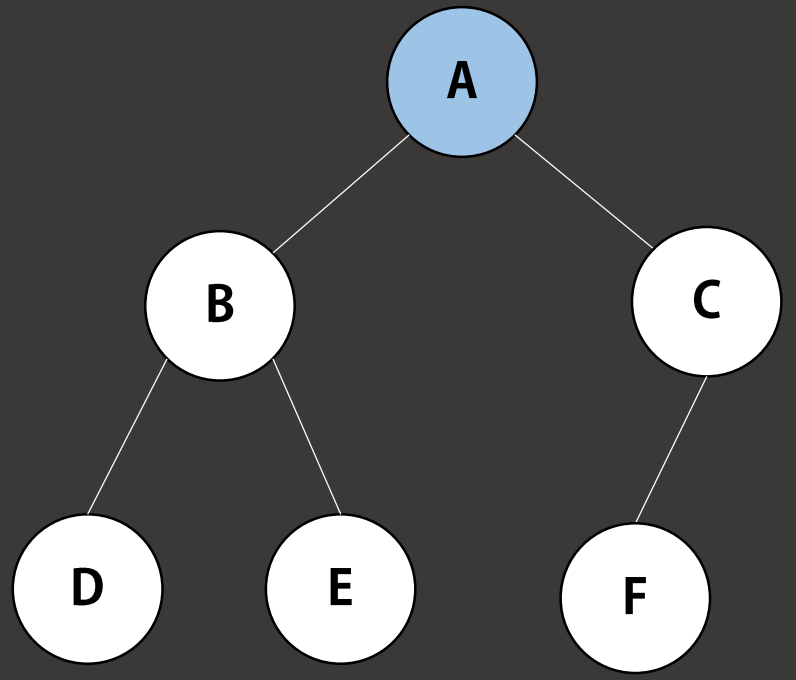
트리의 구현

트리는 그래프의 일종이기 때문에
그래프와 같은 방식으로 구현할 수 있습니다.
인접 행렬, 인접 리스트 등등

트리의 구현

하지만, 트리만의 방법도 있습니다!

트리의 구현

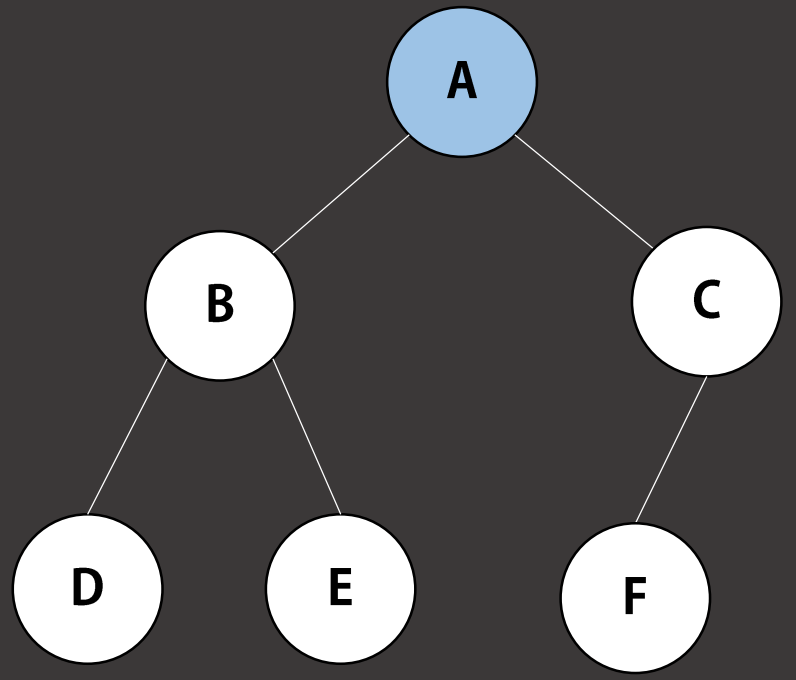


첫 번째로,

각 정점의 부모를 배열에 저장하는 방법이 있습니다.
이때, 루트 정점에는 임의의 값을 사용하면 됩니다

i	0	1	2	3	4	5
T[i]	0	A	A	B	B	F

트리의 구현



두 번째로,

완전 이진 트리의 경우 배열로도
표현할 수 있습니다.
이때 부모 정점이 i라면
왼쪽 자식은 2*i+1, 오른쪽 자식은
2*i+2입니다.
왜냐하면 완전 이진 트리의 자식은
왼쪽부터 채워 지기 때문입니다.

i	0	1	2	3	4	5
T[i]	A	B	C	D	E	F

트리의 구현

마지막으로,

이진 트리의 경우
구조체 또는 클래스를 이용하여
직관적으로 표현하는 방법도 있습니다.
코드가 직관적이라는 장점이 있습니다.

```
struct Node
{
    int left;
    int right;
};

Node T[50];
```

가볍게,
읽을거리

;

링크

구조체란 무엇인가요?

"알아 두면 매우 유용합니다!"
"구조체란 하나 이상의 변수를 그룹 지어서 새로운 자료형을 정의하는 것"

이제 이진 트리를 어떻게
순회하는지 알아봅시다.

그전에,
순회란 무엇일까요?

순회란,

각각의 정점을 정확히 한번만,
체계적인 방법으로 방문하는 과정을 말합니다.

#앞서 배운 DFS/BFS도 그래프 순회의 일종입니다.

트리의 순회

대표적으로 세가지 방법이 있습니다.

Pre-order 전위 순회

In-order 중위 순회

Post-order 후위 순회

트리의 순회

어렵지 않습니다!
어떤 정점을 먼저 순회할지 정하는 것입니다.

Pre-order 전위 순회
In-order 중위 순회
Post-order 후위 순회

트리의 순회

현재 정점을 언제 방문하는지 주목해주세요!

Pre-order 전위 순회

현재 정점 -> 왼쪽 가지 -> 오른쪽 가지

In-order 중위 순회

왼쪽 가지 -> 현재 정점 -> 오른쪽 가지

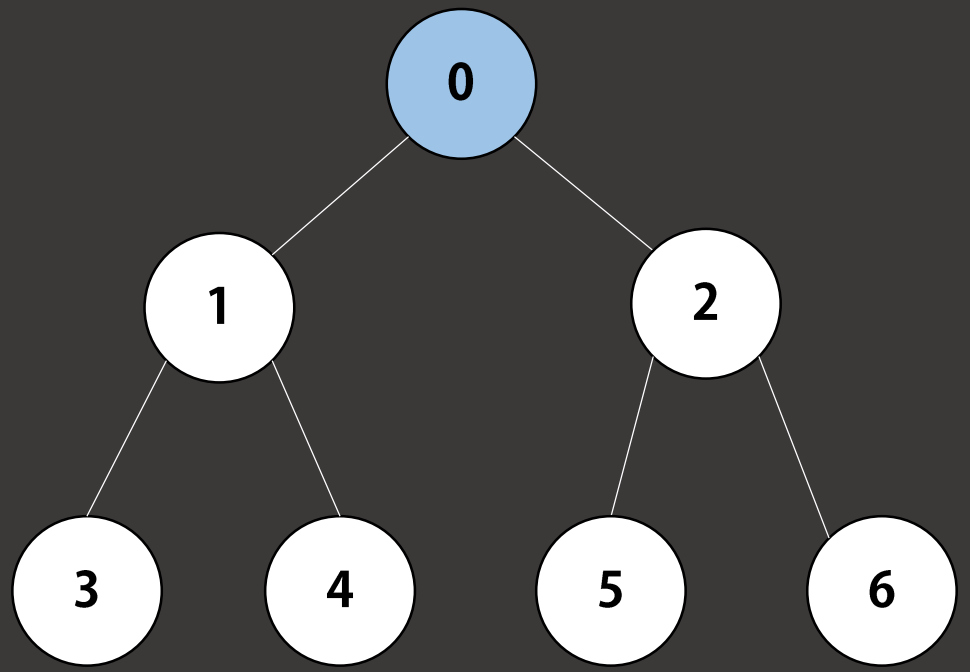
Post-order 후위 순회

왼쪽 가지 -> 오른쪽 가지 -> 현재 정점

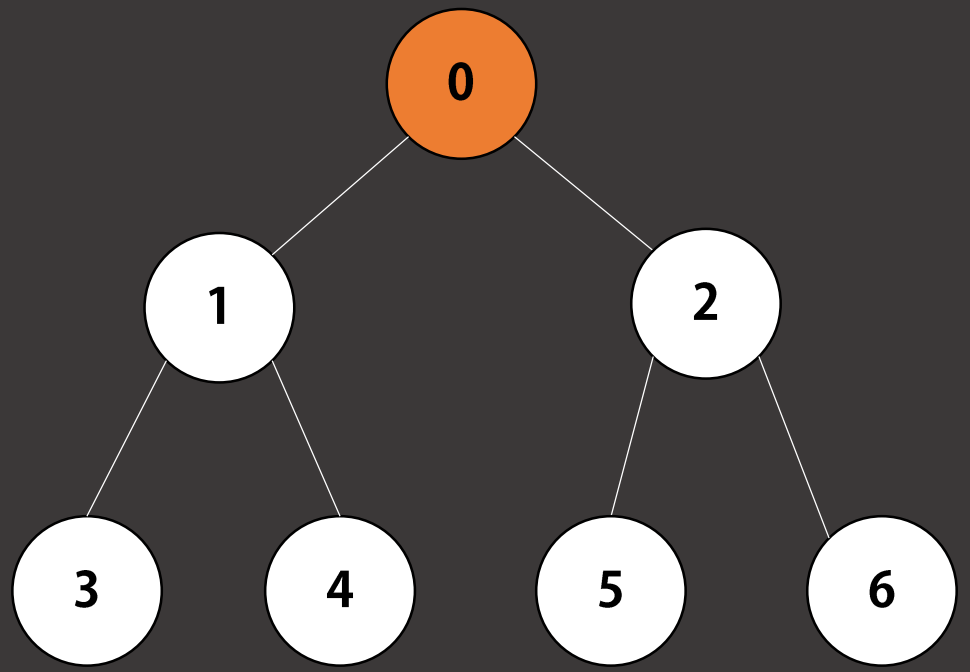
트리의 순회

Pre-order 전위 순회를 예시로
좀 더 자세히 알아봅시다!

#이것만 이해해도 다른 순회는 식은 죽 먹기입니다!

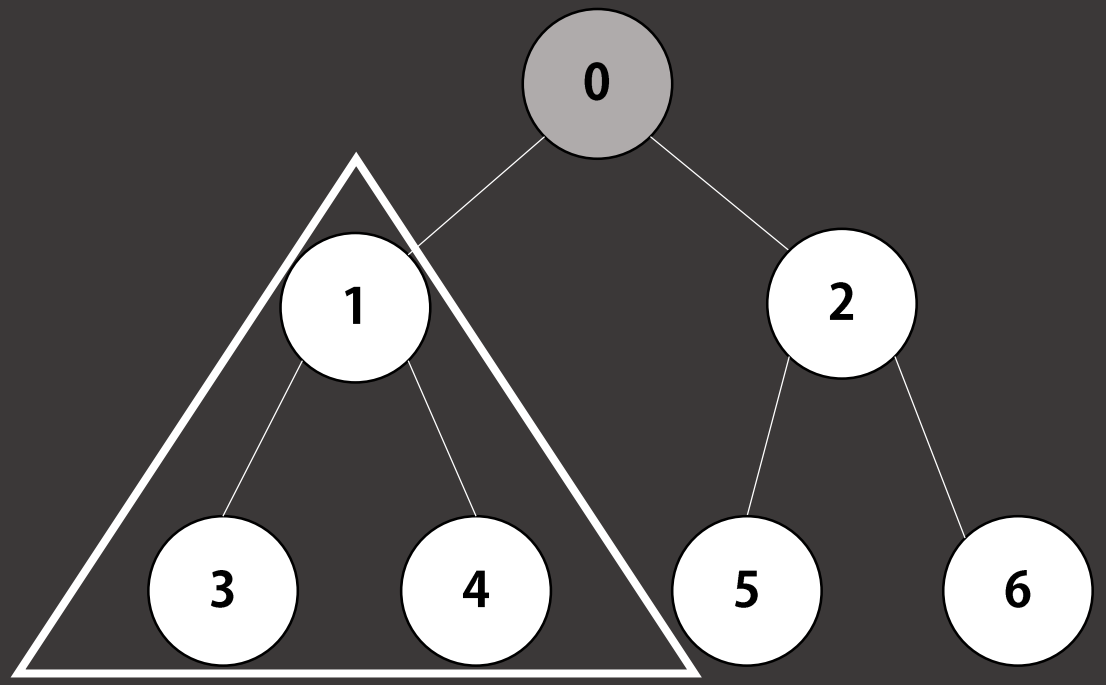


다음 트리를 Pre-order로 순회해봅시다!
Pre-order는 현재 정점 -> 왼쪽 가지 -> 오른쪽 가지 순이라는 것을 기억해주세요!
이때, 방문한 정점을 순서대로 기록하겠습니다.



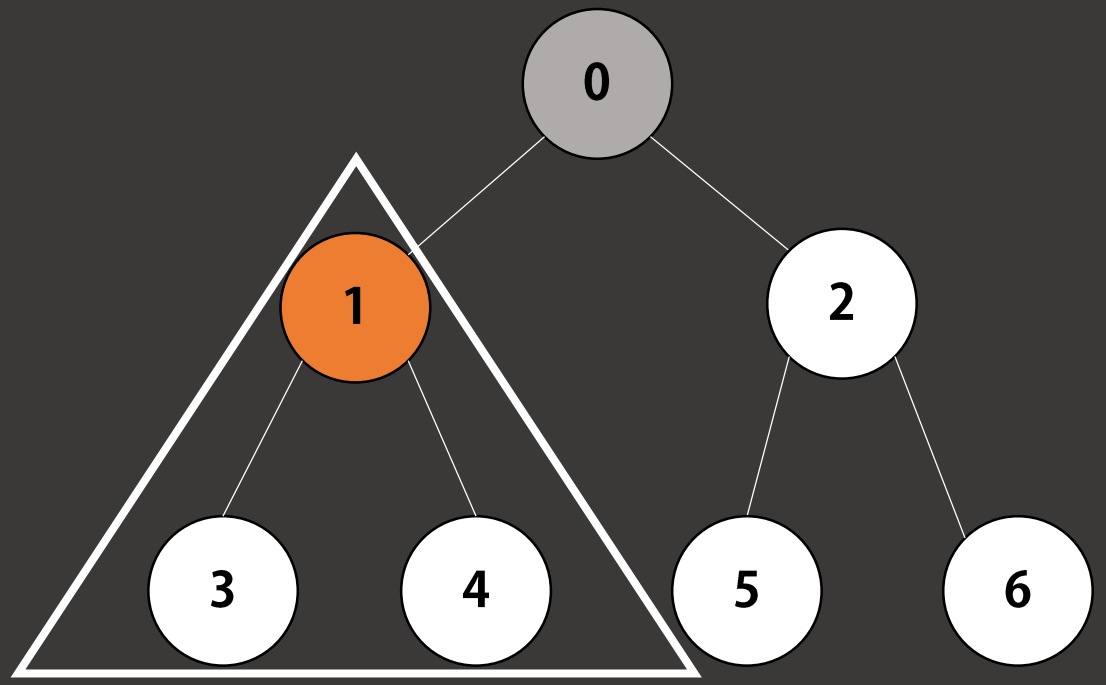
방문한 정점
0

가장 먼저,
루트 정점인 0부터 순회를 시작하겠습니다.
현재 정점인 0을 방문합니다.



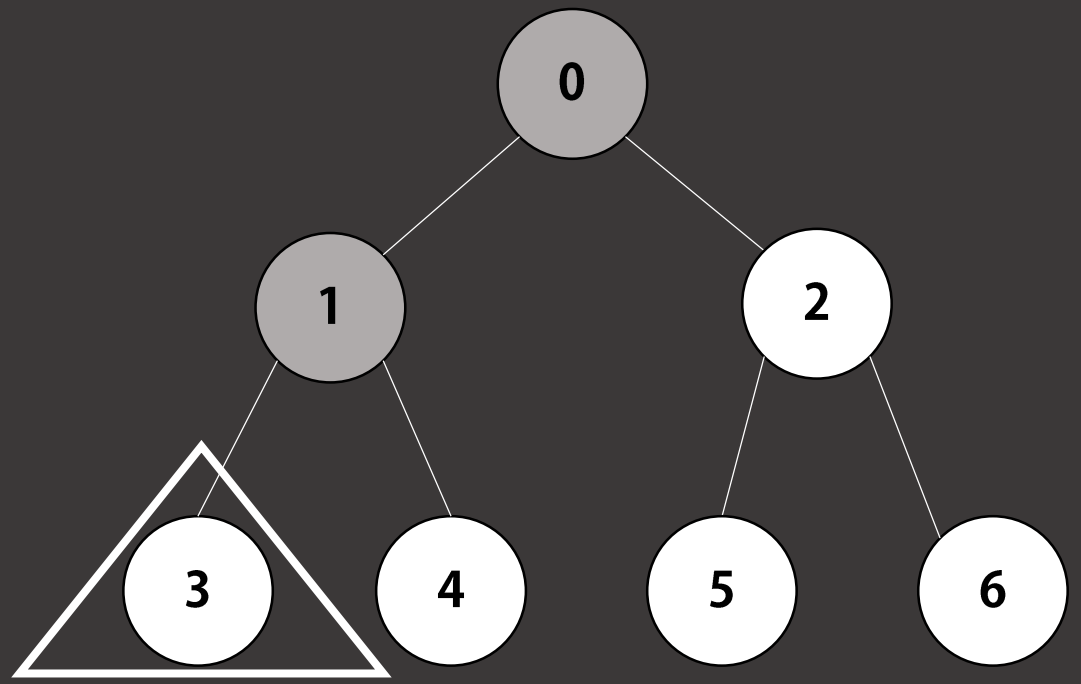
방문한 정점
0

그리고 왼쪽 가지를 순회하겠습니다



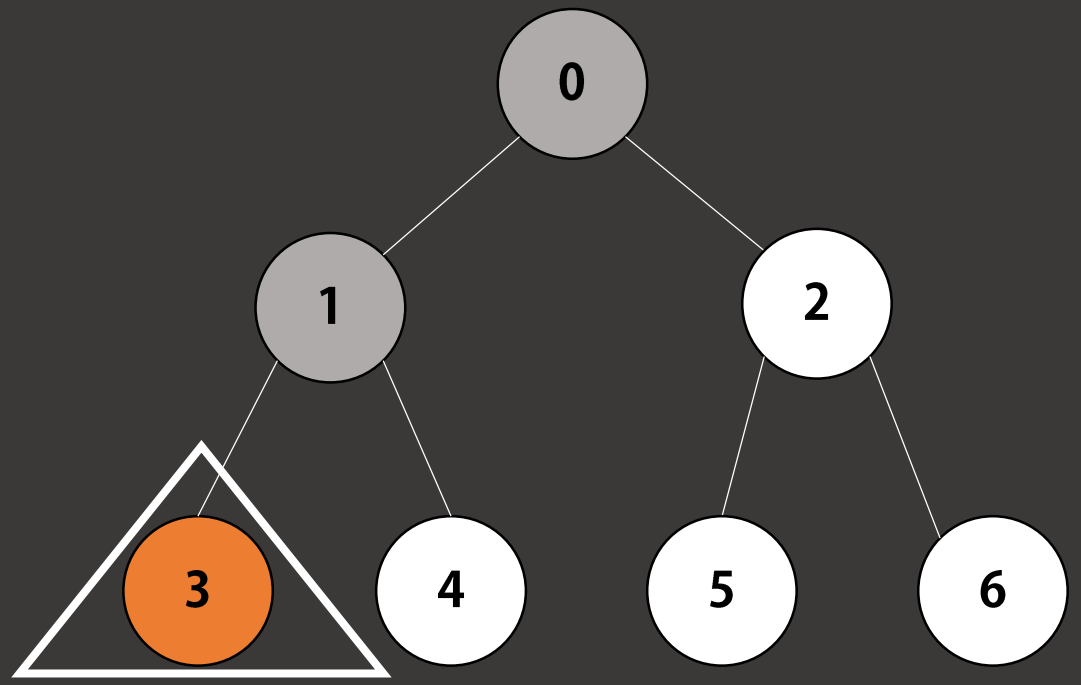
방문한 정점
0 1

우선 왼쪽 가지의 현재 정점인 1을 방문합니다.



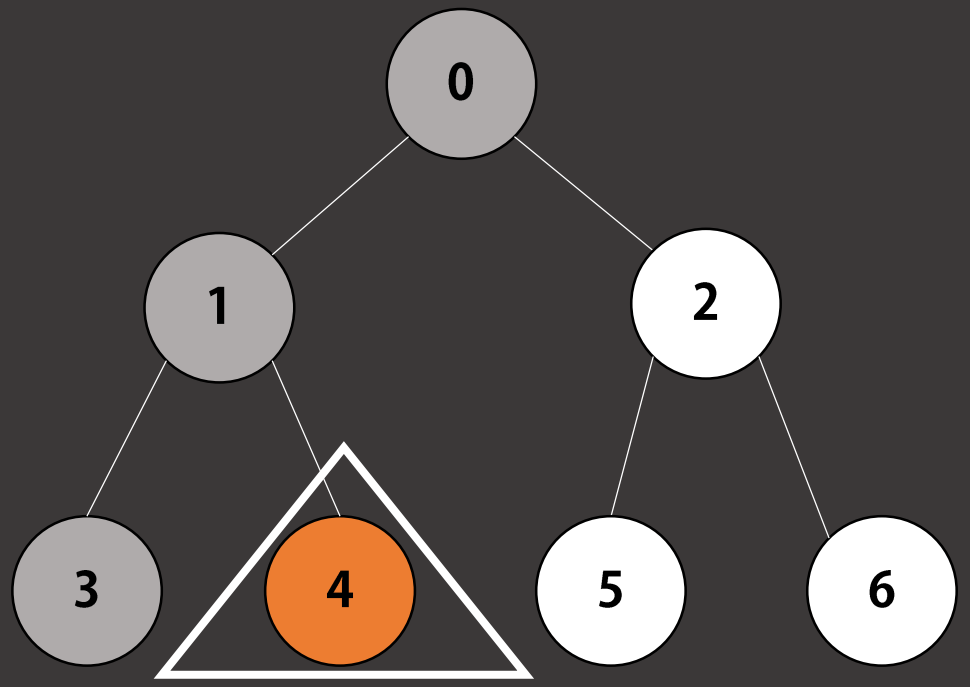
방문한 정점
0 1

그리고 현재 정점 1의 왼쪽 가지를 순회하겠습니다.



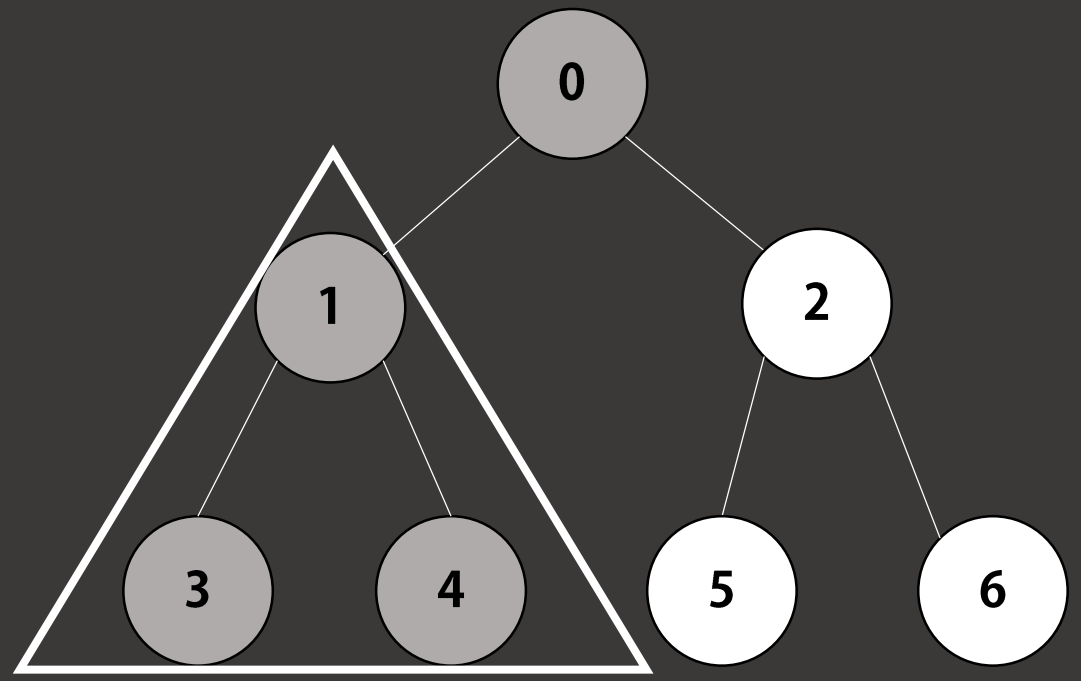
방문한 정점
0 1 3

왼쪽 가지의 현재 정점인 3을 방문합니다.
현재 정점 3에는 더 이상 자식 정점이 없으므로 정점 3의 순회를 종료합니다.



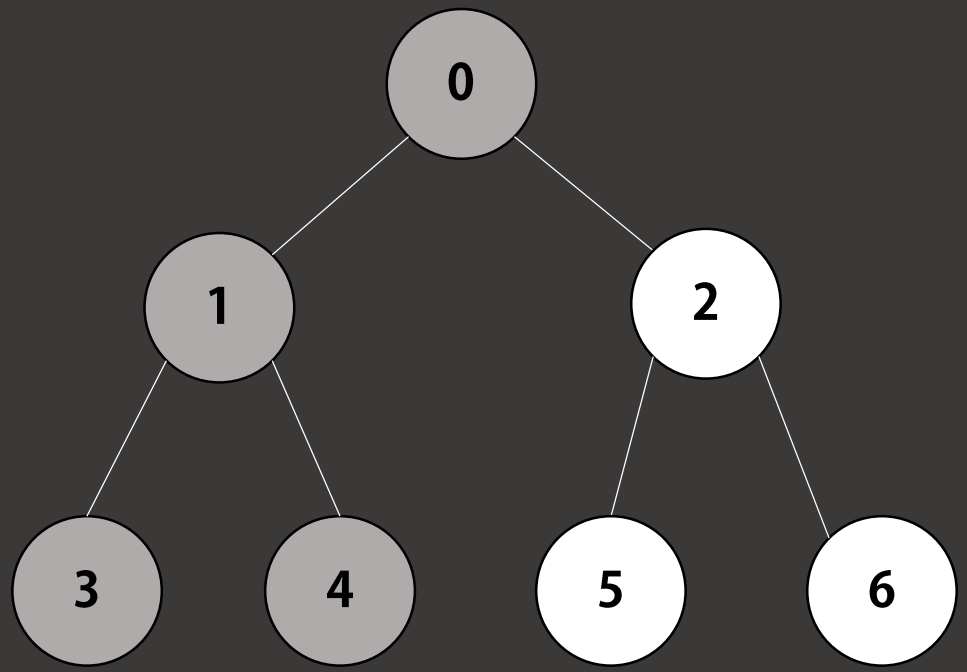
방문한 정점
0 1 3 4

그리고 이번에는 정점 1의 오른쪽 가지를 방문합니다.
마찬가지로 오른쪽 가지의 현재 정점인 4를 방문합니다.
그리고 더 이상 4의 자식 정점이 없으므로 순회를 종료합니다.

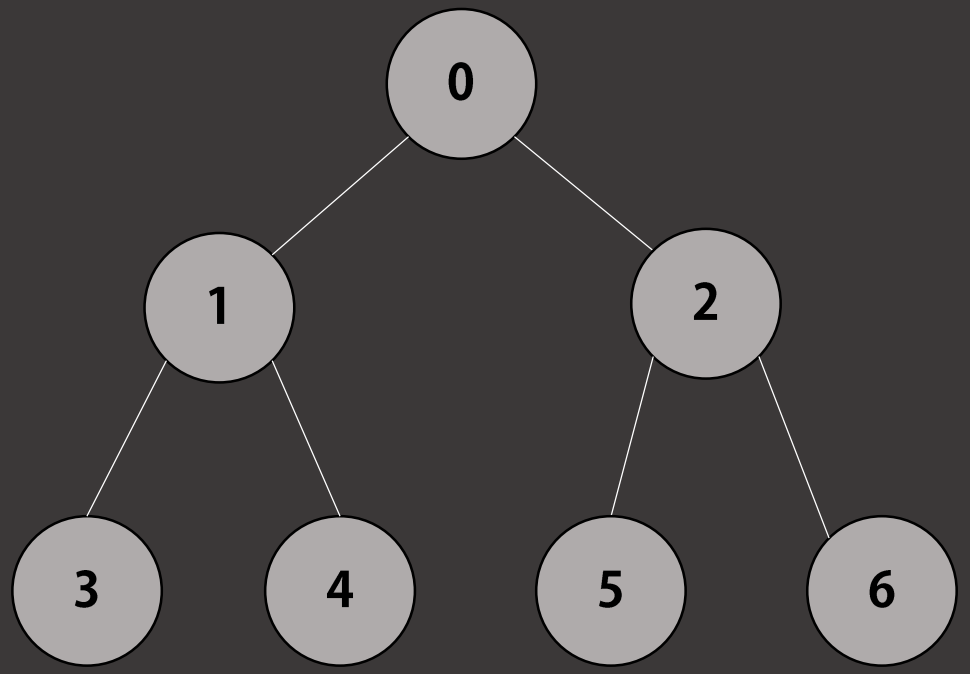


방문한 정점
0 1 3 4

정점 1의 현재 정점, 왼쪽 가지, 오른쪽 가지 순회를 마쳤습니다.



다시 말해, 정점 0의 왼쪽 가지의 순회를 마쳤습니다.



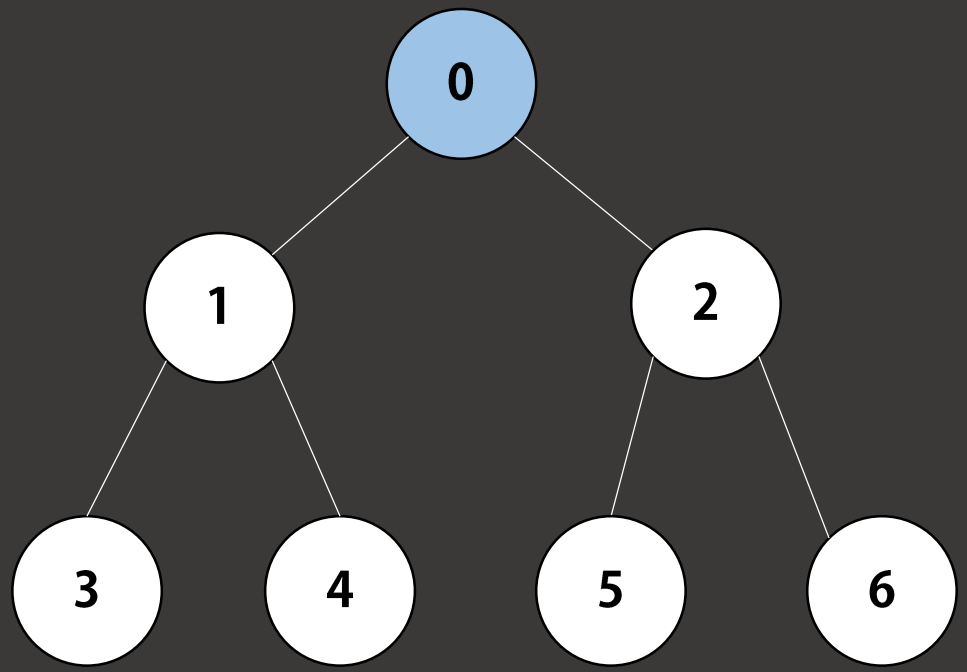
방문한 정점
0 1 3 4 2 5 6

같은 방법으로 정점 0의 오른쪽 가지의 순회를 마치면,
트리의 순회가 완료됩니다!

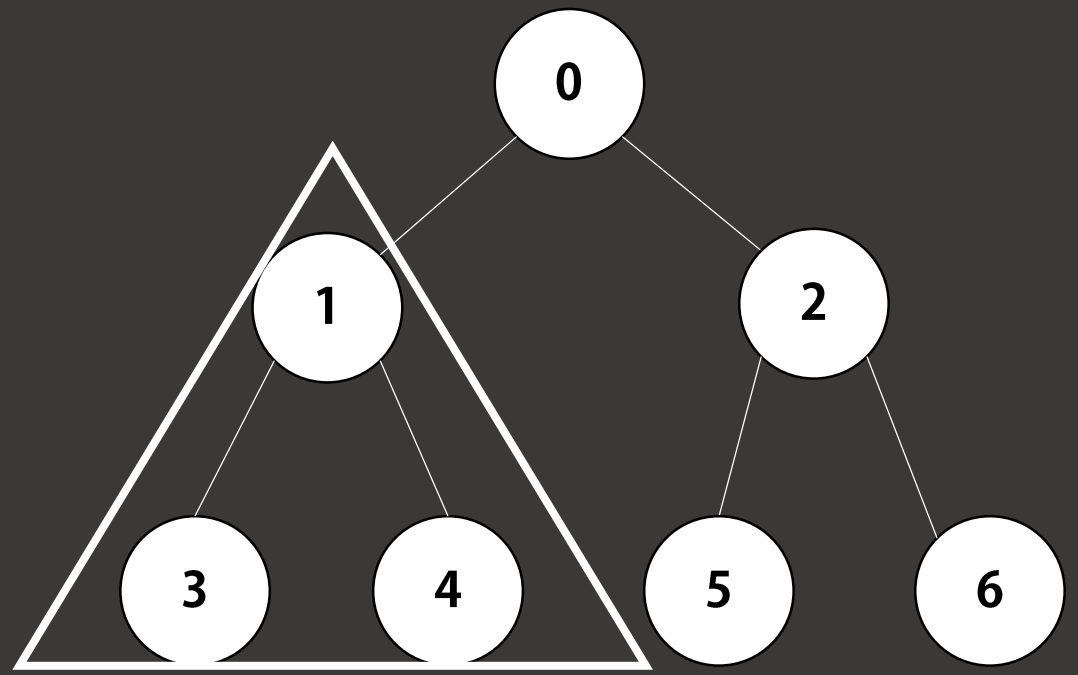
#오른쪽 가지의 순회에 대한 자세한 설명은 생략하겠습니다!

트리의 순회

Post-order 후위 순회를 예시로
한번만 더 자세히 알아봅시다!

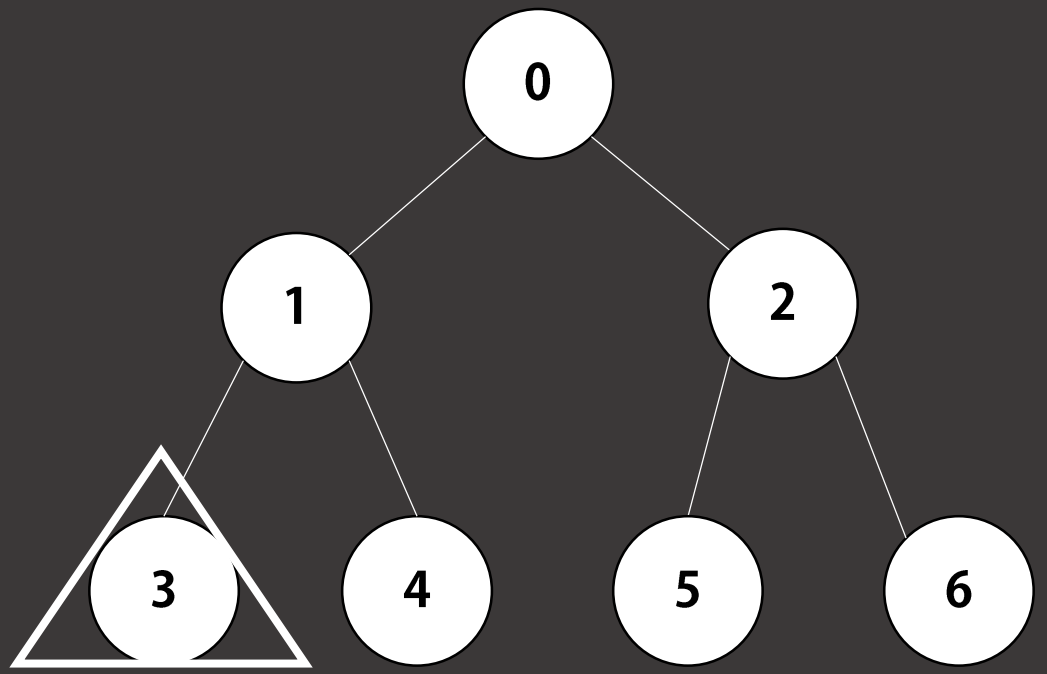


다음 트리를 Post-order로 순회해봅시다!
Post-order은 왼쪽 가지 -> 오른쪽 가지 -> 현재 정점 순이라는 것을 기억해주세요!



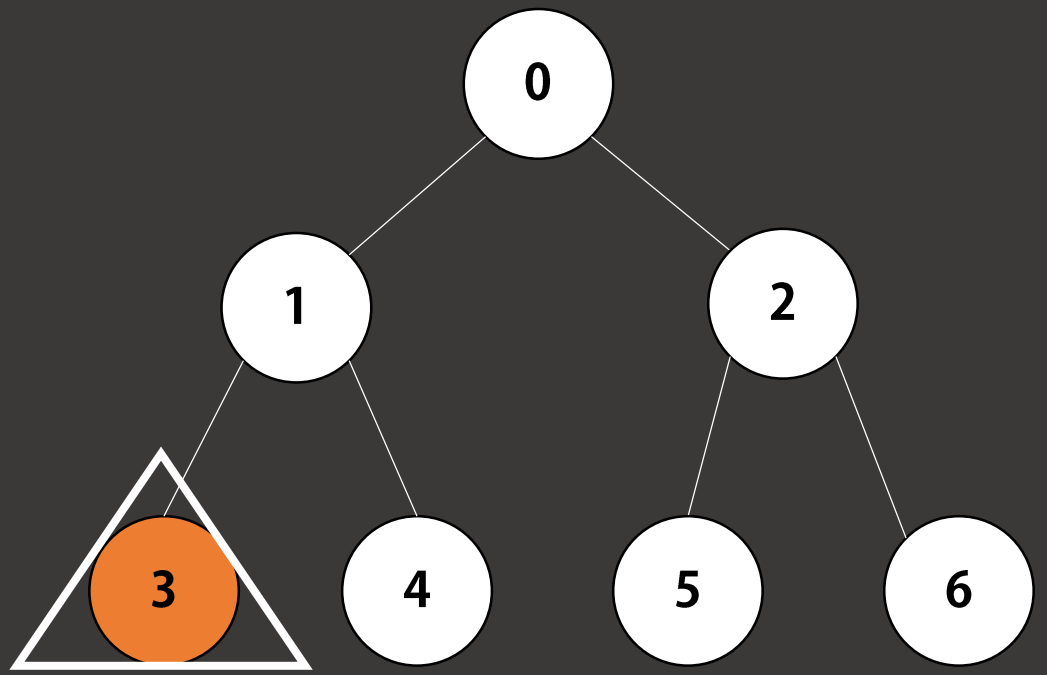
방문한 정점

루트 정점인 0부터 순회를 시작하겠습니다!
Pre-order 방식과 다르게 현재 정점인 0을 일단 방문하지 않고,
우선 왼쪽 가지를 먼저 순회합니다.



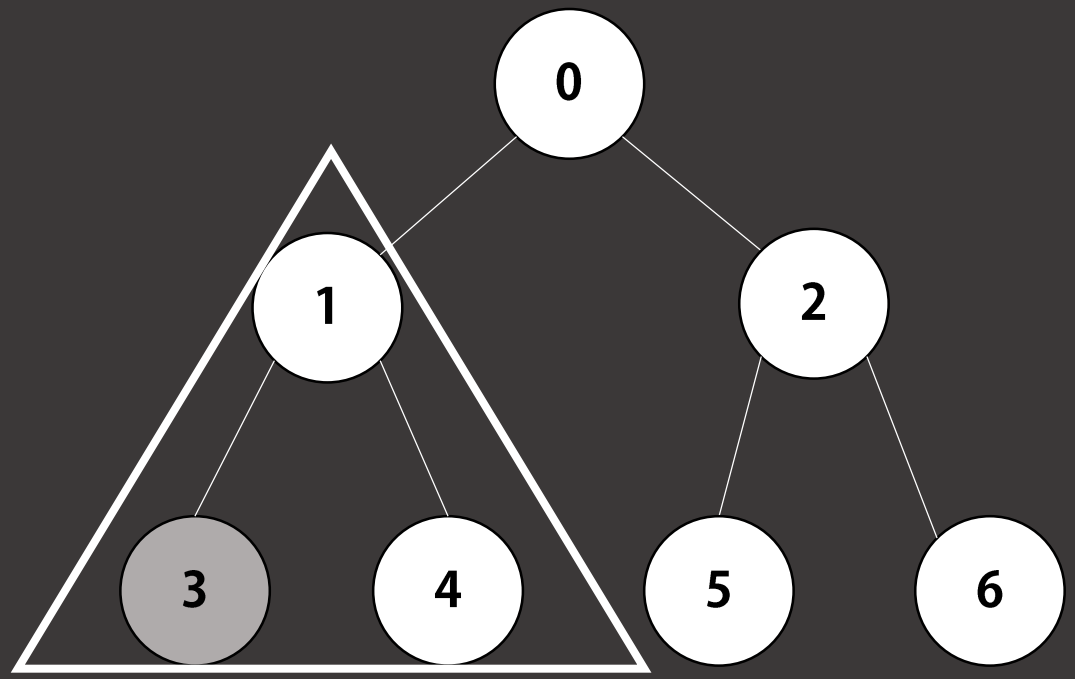
방문한 정점

현재 정점인 1을 방문하지 않고,
다시 왼쪽 가지를 순회합니다.



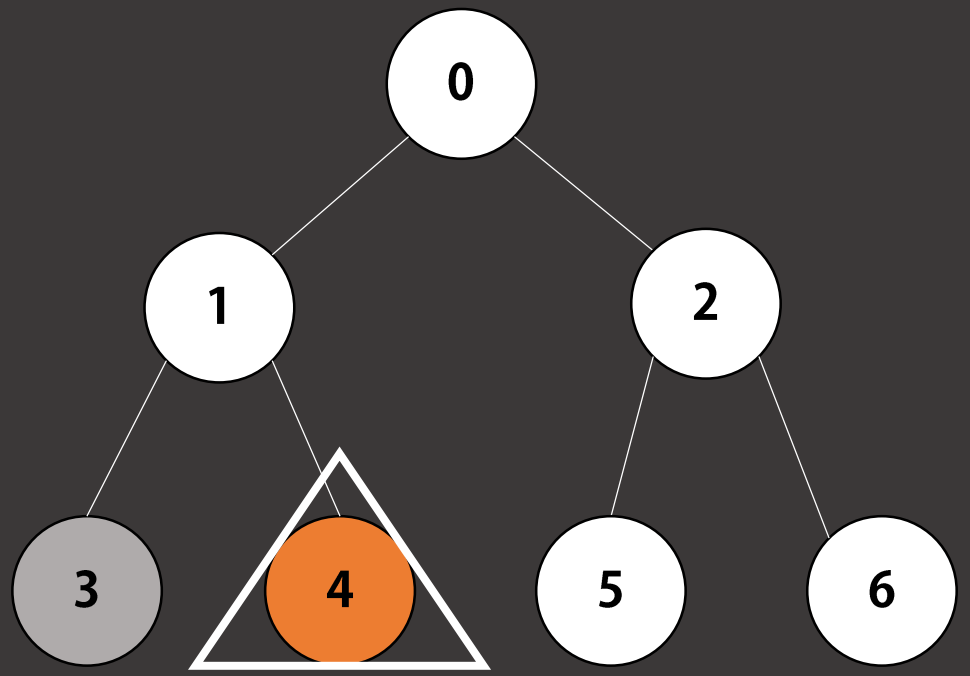
방문한 정점
3

정점 3에 더 이상 왼쪽 가지가 없습니다.
오른쪽 가지 또한 없습니다.
마지막으로 현재 정점 3을 방문합니다.



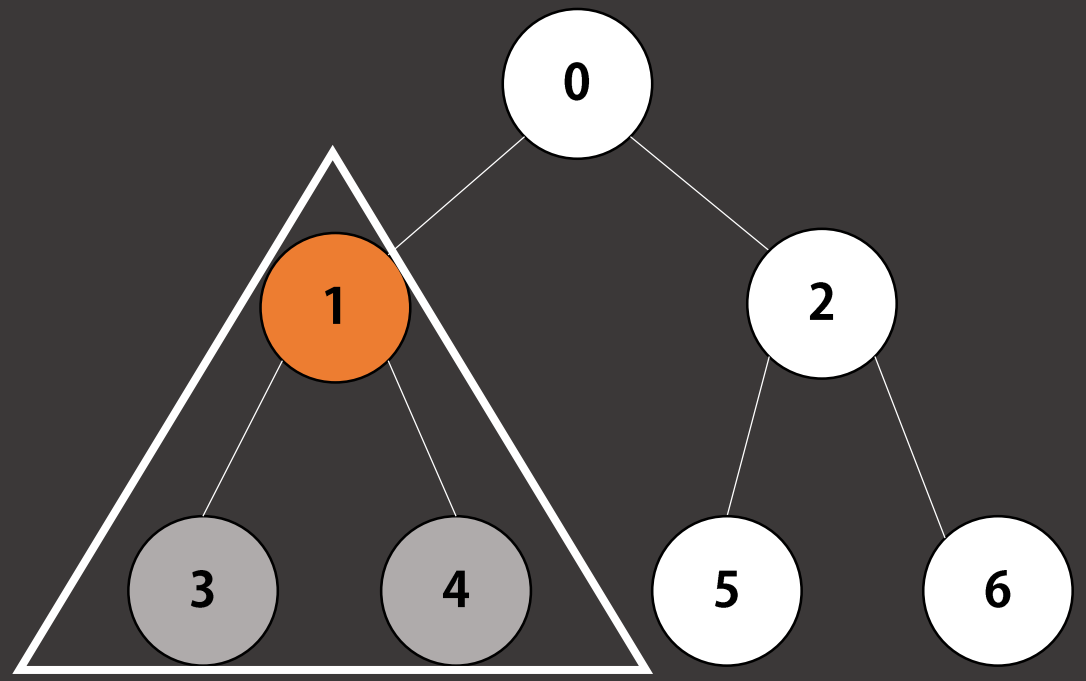
방문한 정점
3

이제 1의 오른쪽 가지를 방문할 차례입니다.



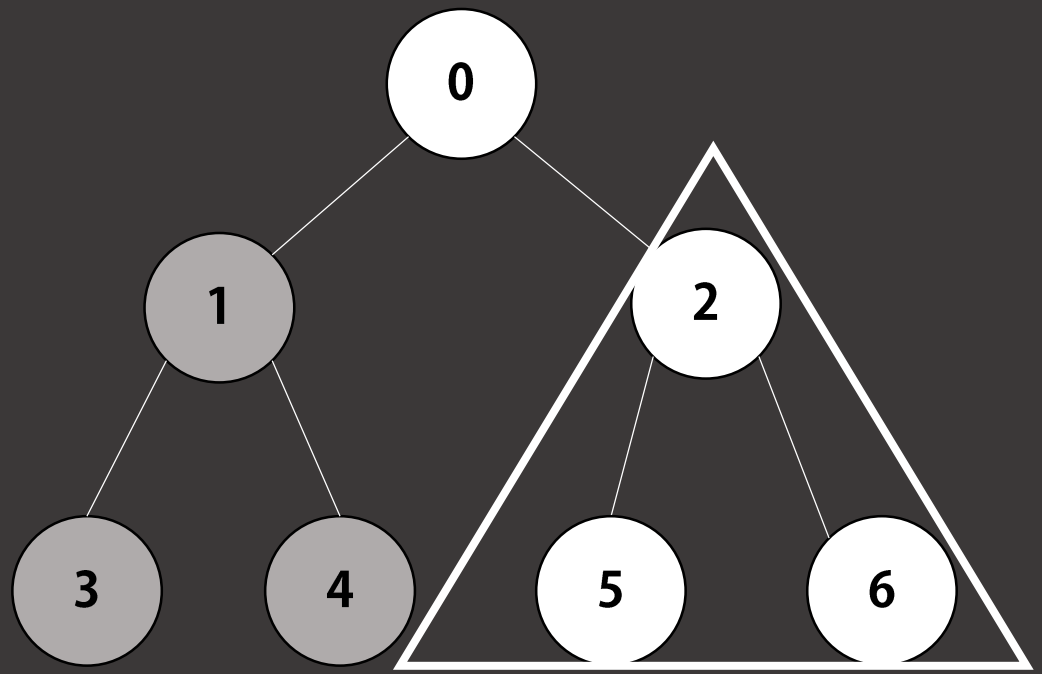
방문한 정점
3 4

정점 4의 왼쪽 가지가 없습니다.
오른쪽 가지 또한 없기 때문에
마지막으로 현재 정점인 4를 방문합니다.



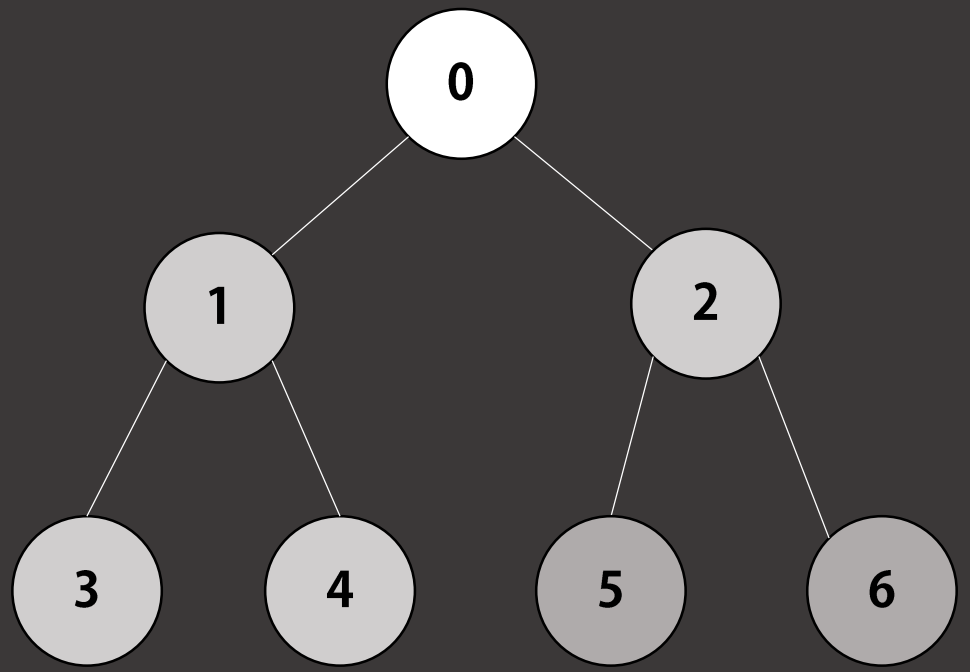
방문한 정점
3 4 1

정점 1의 오른쪽 가지의 순회가 끝났습니다.
또한 왼쪽 가지의 순회가 끝났습니다.
이제 현재 정점인 1를 방문합니다.



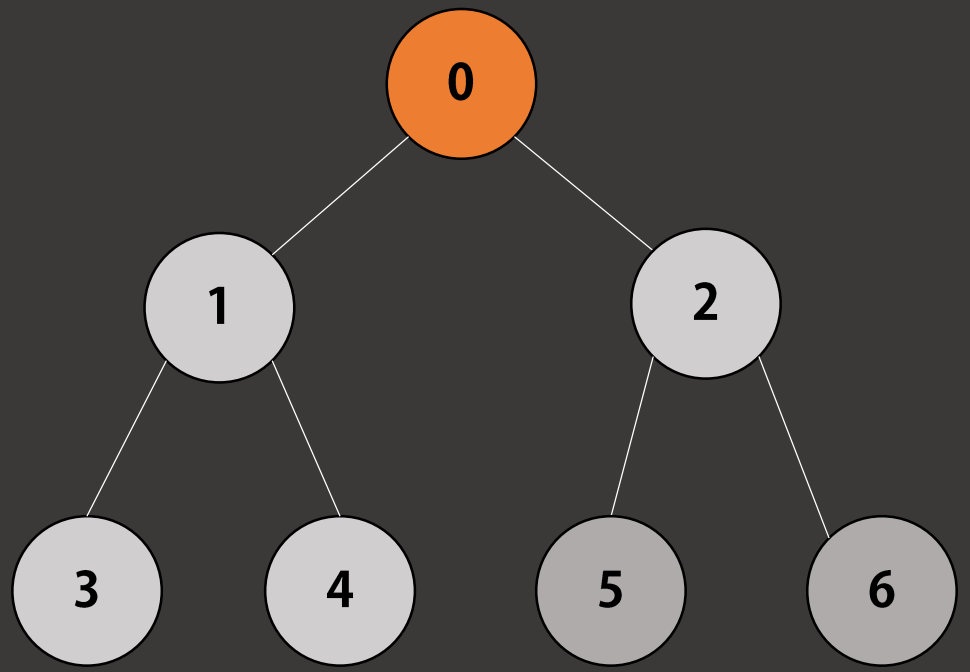
방문한 정점
3 4 1

정점 0의 왼쪽 가지의 순회가 끝났기 때문에
이제 정점 0의 오른쪽 가지의 순회를 할 차례입니다.



방문한 정점
3 4 1 5 6 2

정점 0의 오른쪽 가지를 순회하는 자세한 설명은 생략하겠습니다.
지금까지 순회한 결과는 다음과 같습니다.



방문한 정점
3 4 1 5 6 2 0

정점 0의 왼쪽 가지와 오른쪽 가지를 방문했으므로,
마지막으로 현재 정점인 0을 방문합니다.
이로써 트리의 순회가 완료됩니다.

트리의 순회

어떻게 구현할 수 있을까요?

재귀를
이용합니다!

```cpp
void postorder(int node)
{
    if (node == -1)
    {
        return;
    }
    postorder(T[node].left);
    postorder(T[node].right);
    cout << char(node + 'A');
}
```

```cpp
void preorder(int node)
{
    if (node == -1)
    {
        return;
    }
    cout << char(node + 'A');
    preorder(T[node].left);
    preorder(T[node].right);
}
```

```cpp
void inorder(int node)
{
    if (node == -1)
    {
        return;
    }
    inorder(T[node].left);
    cout << char(node + 'A');
    inorder(T[node].right);
}
```

세가지 구현 방법의 코드입니다.
크게 다르지 않습니다!
종료 조건과, 방문 순서에 주목해주세요!
방문한 값을 cout으로 출력하는 대신 배열에 넣어도 무방합니다!

연습 문제

1991번 트리 순회

https://www.acmicpc.net/problem/1991

앞서 배운 트리의 순회를 이용하면 됩니다!

이진 트리 구현 꿀팁

```
cin >> root >> left >> right;
T[root - 'A'].left = left == '.' ? -1 : left - 'A';
T[root - 'A'].right = right == '.' ? -1 : right - 'A';
```

1. 트리를 저장할 때 int형으로 받으면 메모리를 절약할 수 있습니다.
2. 자식 정점이 없을 때 -1을 저장합니다.

연습 문제

2250번 트리의 높이와 너비

https://www.acmicpc.net/problem/2250

앞서 배운 트리의 순회를 이용하면 됩니다!
현재 정점의 열 번호와 레벨을 어떻게 알 수 있을까요?

트리의 탐색

에 대해 알아봅시다.

당연하게도 트리는 그래프의 일종이므로,
DFS/BFS 알고리즘을 이용할 수 있습니다.

연습 문제

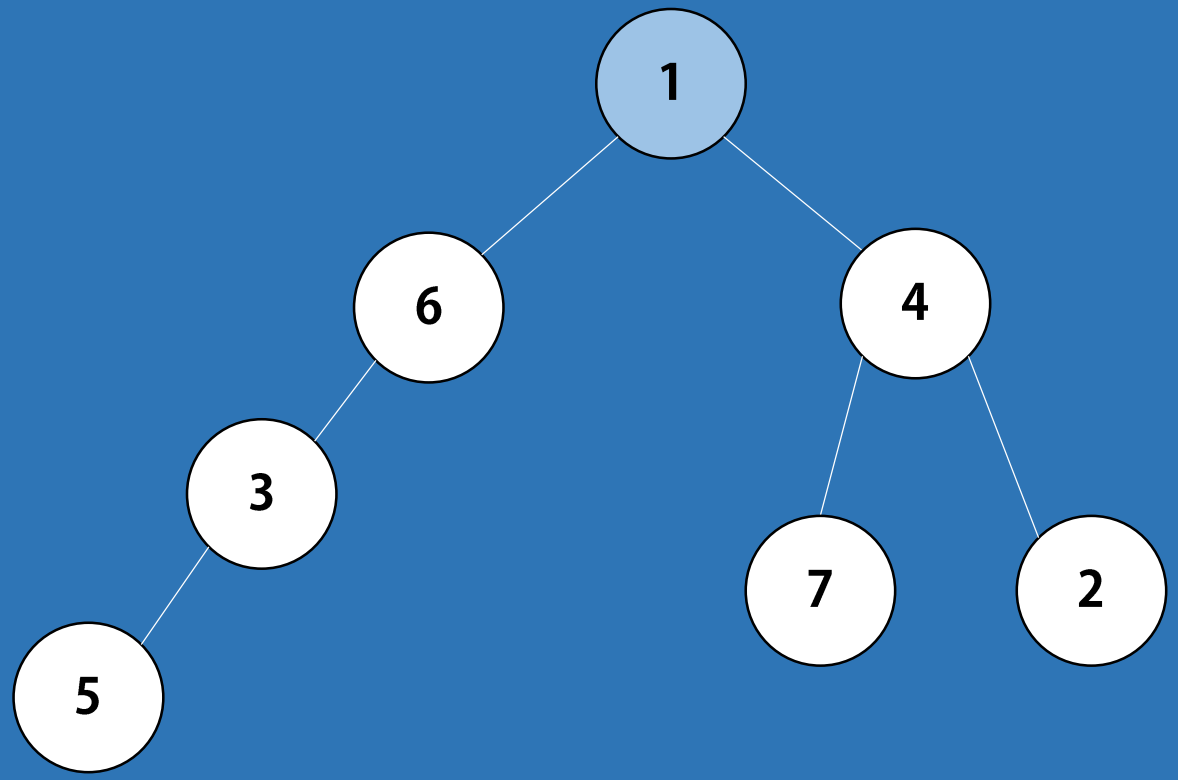
11725번 트리의 부모 찾기

https://www.acmicpc.net/problem/2250

탐색으로 풀 수 있습니다!
구체적으로 어떻게 탐색으로 풀 수 있을까요?
다음 슬라이드에서 알아봅시다.

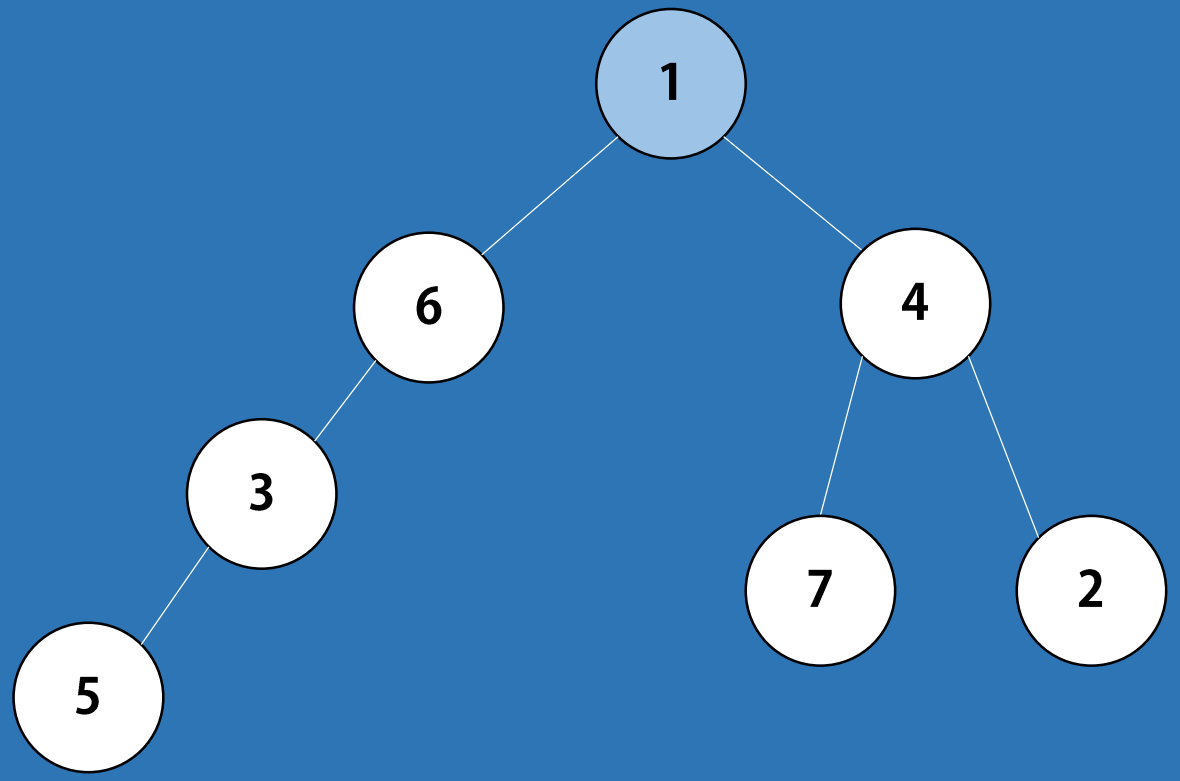
11725번 풀이

너무 어렵게 생각할 필요 없습니다.
예제 입력 1을 그림으로 표현하면 다음과 같습니다.



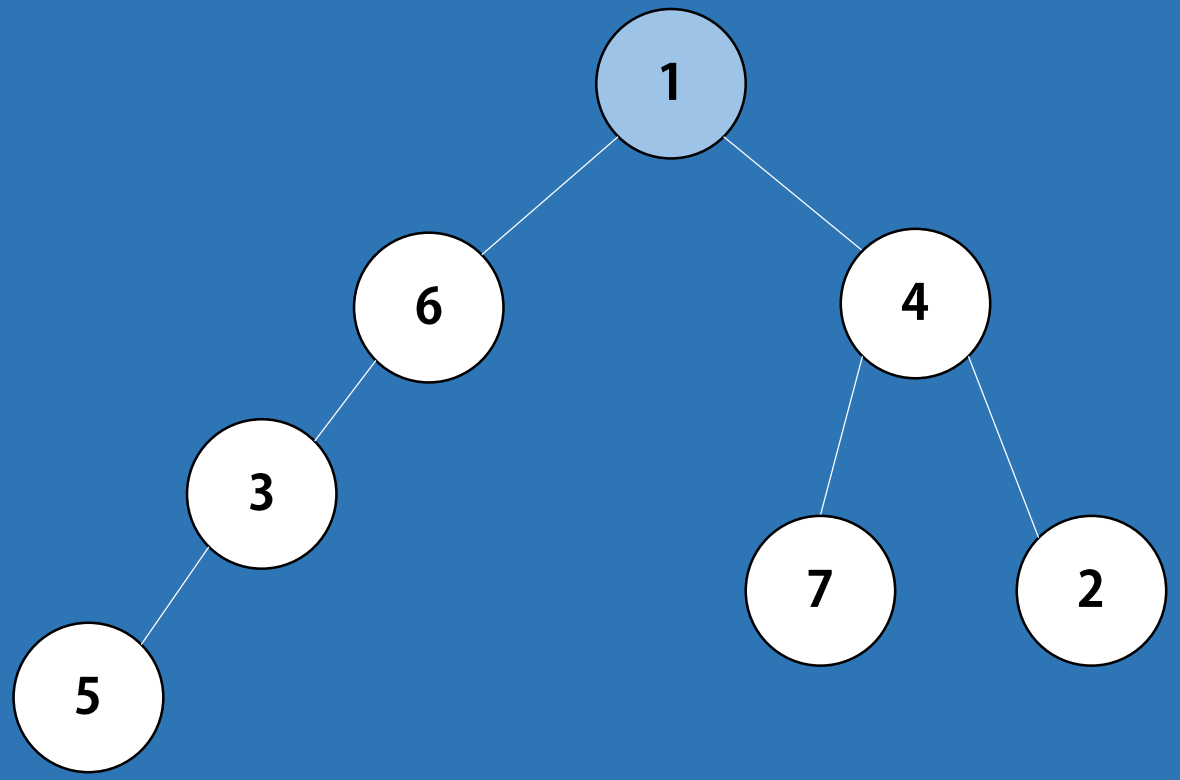
11725번 풀이

그래프와 동일하게,
입력을 인접 행렬 혹은 인접 리스트로
받아 주시면 됩니다!



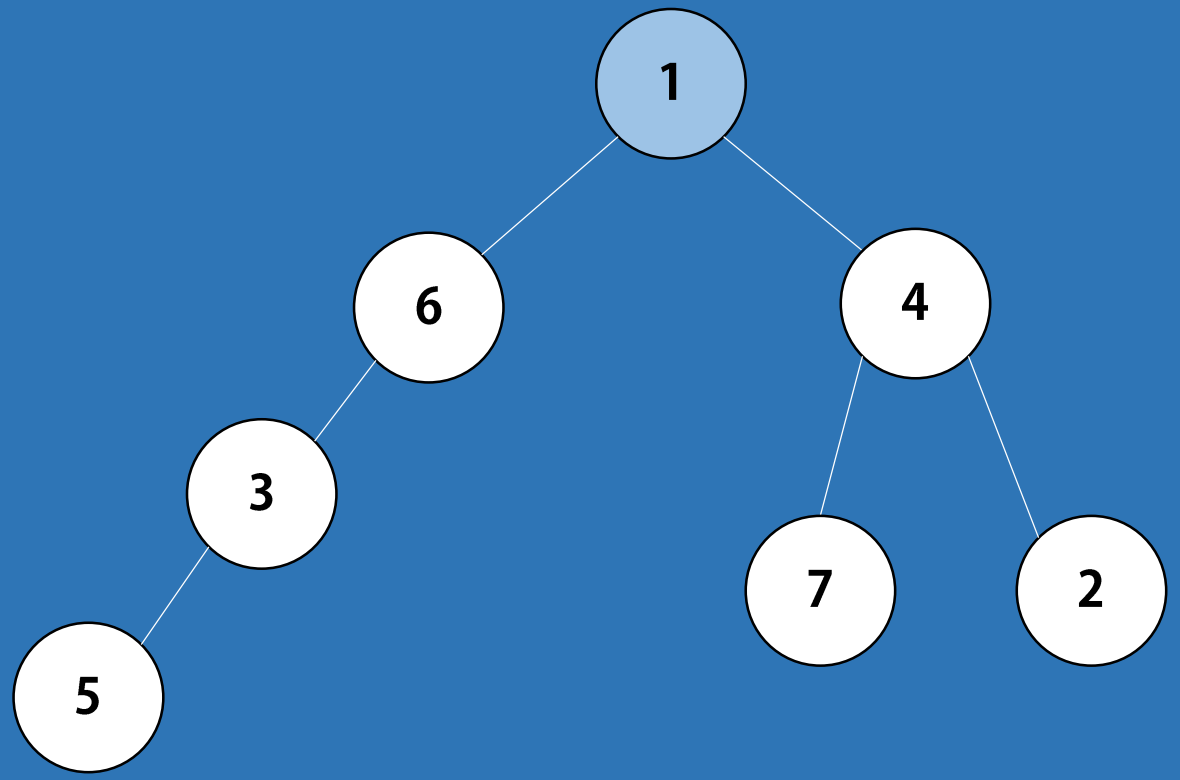
11725번 풀이

그리고 루트 정점인 1부터
DFS 또는 BFS 탐색을 진행하면 됩니다.



11725번 풀이

탐색하는 과정에서,
어떤 정점에서 다음으로 방문할
정점들은 자식 관계입니다!



연습 문제

1167번 트리의 지름

https://www.acmicpc.net/problem/1167

다음 슬라이드에서
트리의 지름을 어떻게 구할 수 있을지
알아봅시다!

1167번 풀이

결론적으로, 트리의 지름은 다음 방법으로 구할 수 있습니다.

1. 한 정점 x에서 다른 모든 정점까지의 거리를 구합니다. 이때 가장 먼 거리의 정점을 y라고 합니다.
2. 정점 y에서 다른 모든 정점까지의 거리를 구합니다. 이때 가장 먼 거리의 정점을 z라고 합니다.
3. 트리의 지름은 두 정점 y와 z 사이의 거리라고 할 수 있습니다.

즉, 탐색 두번으로 트리의 지름을 구할 수 있습니다!

읽을거리

;

링크

왜 트리의 지름을 다음과 같이 구할 수 있을까요?
트리의 지름 증명입니다.

정리

이번 시간에는

트리에 대해 알아보았습니다!

1. 트리의 정의
2. 트리와 관련된 용어
3. 트리의 구현
4. 트리의 탐색

트리는 탐색이나 정렬 등 다양한 알고리즘을 구현하는데
매우 유용하게 사용됩니다!

문제들을 풀어보며, 개념을 탄탄하게 정리해 봅시다!

The End

수고하셨습니다!

다음 시간에는

정렬에 대해 알아봅시다!