

/* 데이터 구조 및 알고리즘 */

신현규 강사, 화/목 20:00

트리, 재귀호출 (1)

좋은 코드란 무엇이고, 왜 좋은가?



/* elice */

주차별 커리큘럼

1주차 과정 소개, 배열, 연결리스트, 클래스

2주차 스택, 큐, 해싱

3주차 시간복잡도

4주차 트리, 트리순회, 재귀호출

5주차 힙

6주차 그래프 소개, DFS

7주차 그래프 심화, BFS

8주차 강의 요약, 알고리즘 과정 소개

컴퓨터를 이용한 문제 해결 과정

1. 문제를 정확히 이해한다
2. 문제를 해결하는 알고리즘을 개발한다
3. 알고리즘이 문제를 해결한다는 것을 증명한다
4. 알고리즘이 제한시간 내에 동작한다는 것을 보인다
5. 알고리즘을 코드로 작성한다
6. 제출 후 만점을 받고 매우 기뻐한다

요약 : 시간복잡도

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
def findNumber(myList, target) :  
    for v in myList :  
        if v == target :  
            return True  
  
    return False
```

요약 : 시간복잡도

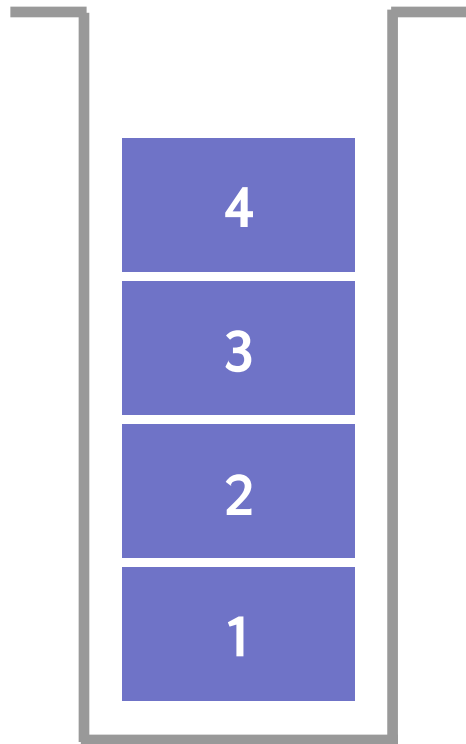
알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
def findNumber(myList, target) :  
    for v in myList :  
        if v == target :  
            return True  
  
    return False
```

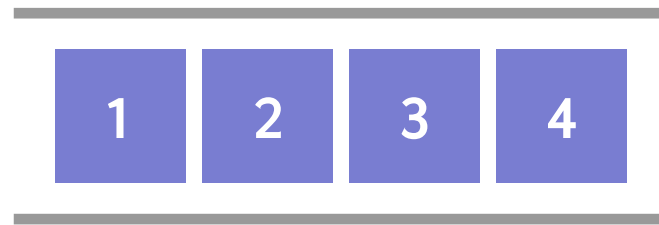
O(n)

대표적인 자료구조



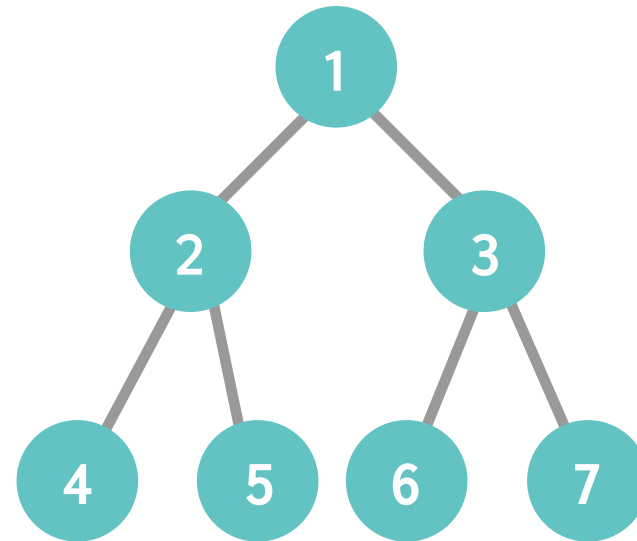
스택 (Stack)

Last In First Out

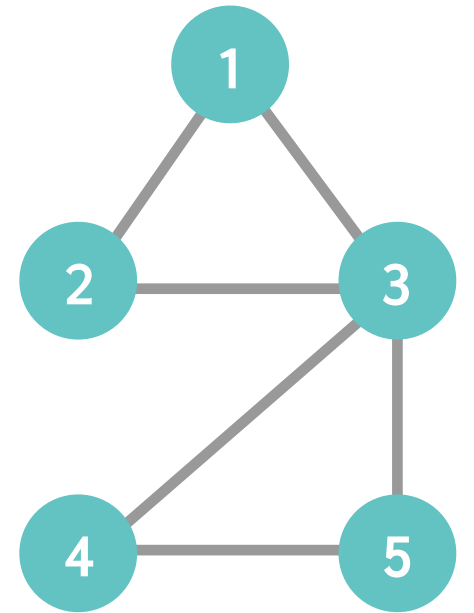


큐 (Queue)

First In First Out



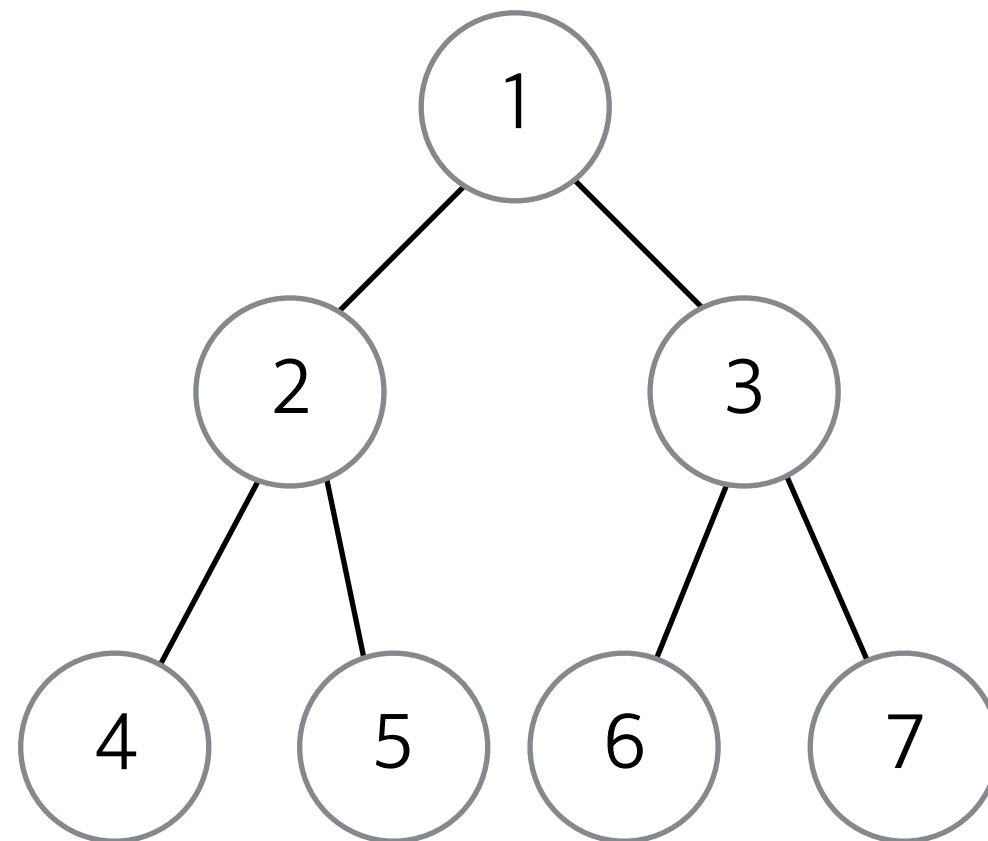
트리 (Tree)



그래프 (Graph)

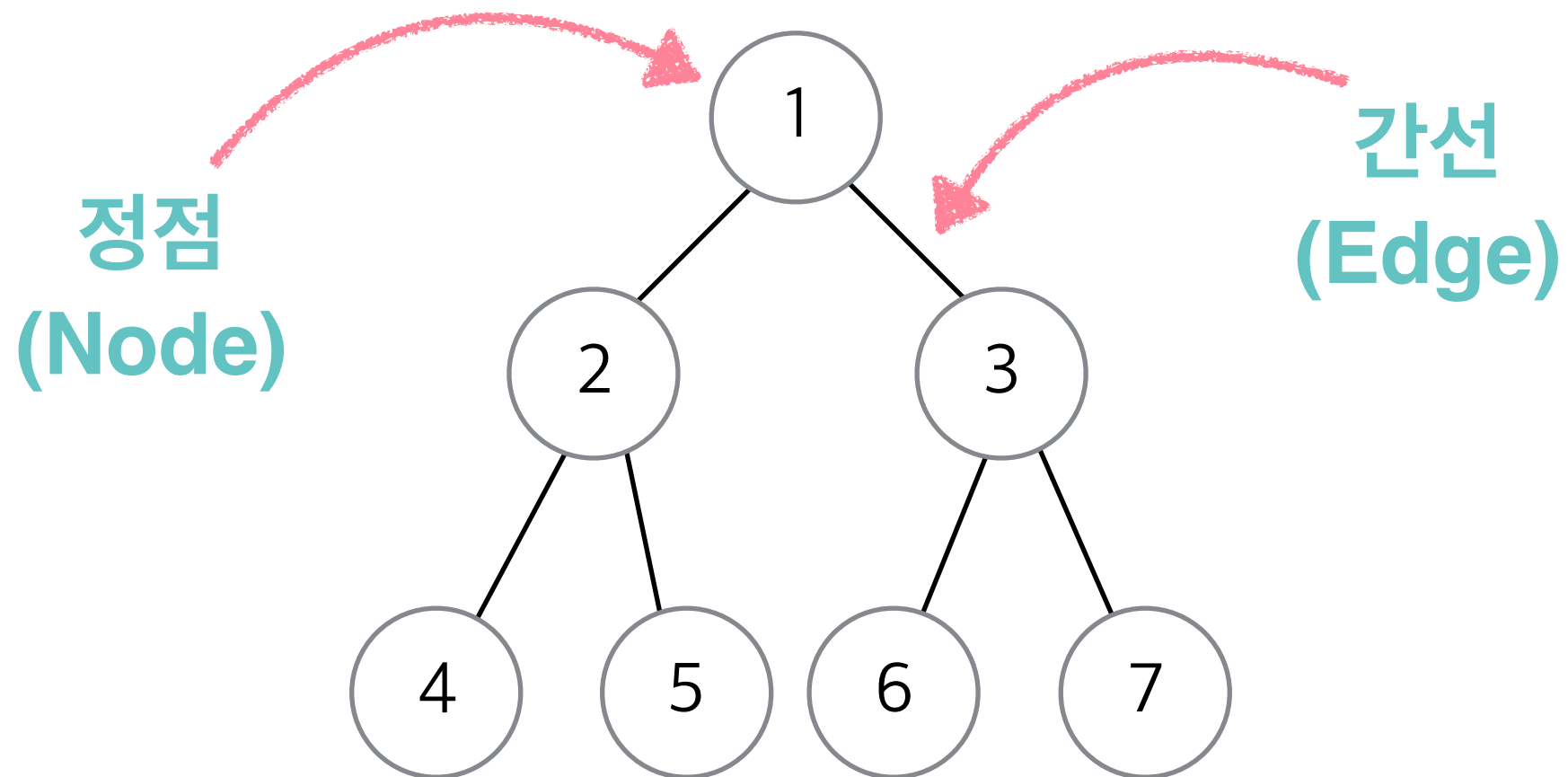
트리

아래와 같이 생긴 자료구조



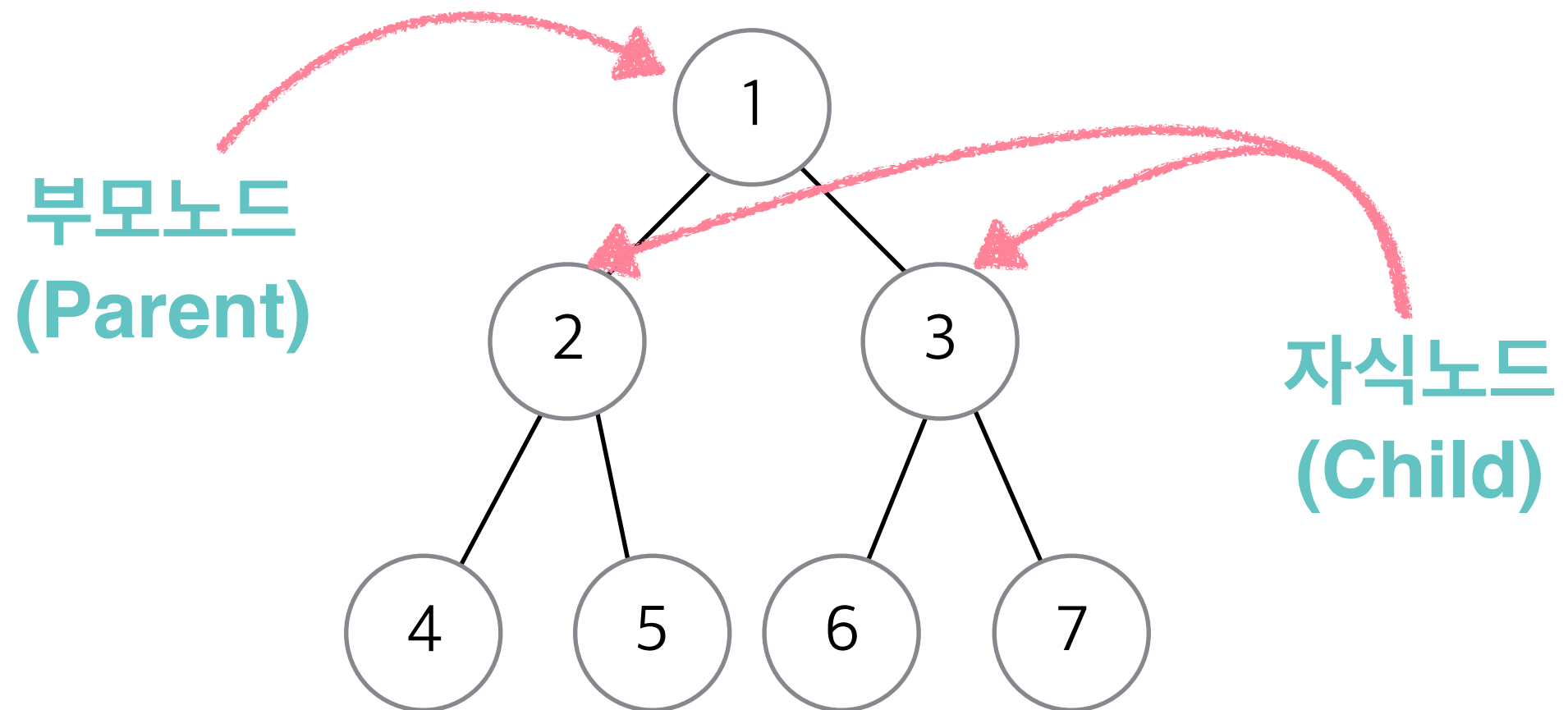
트리

아래와 같이 생긴 자료구조



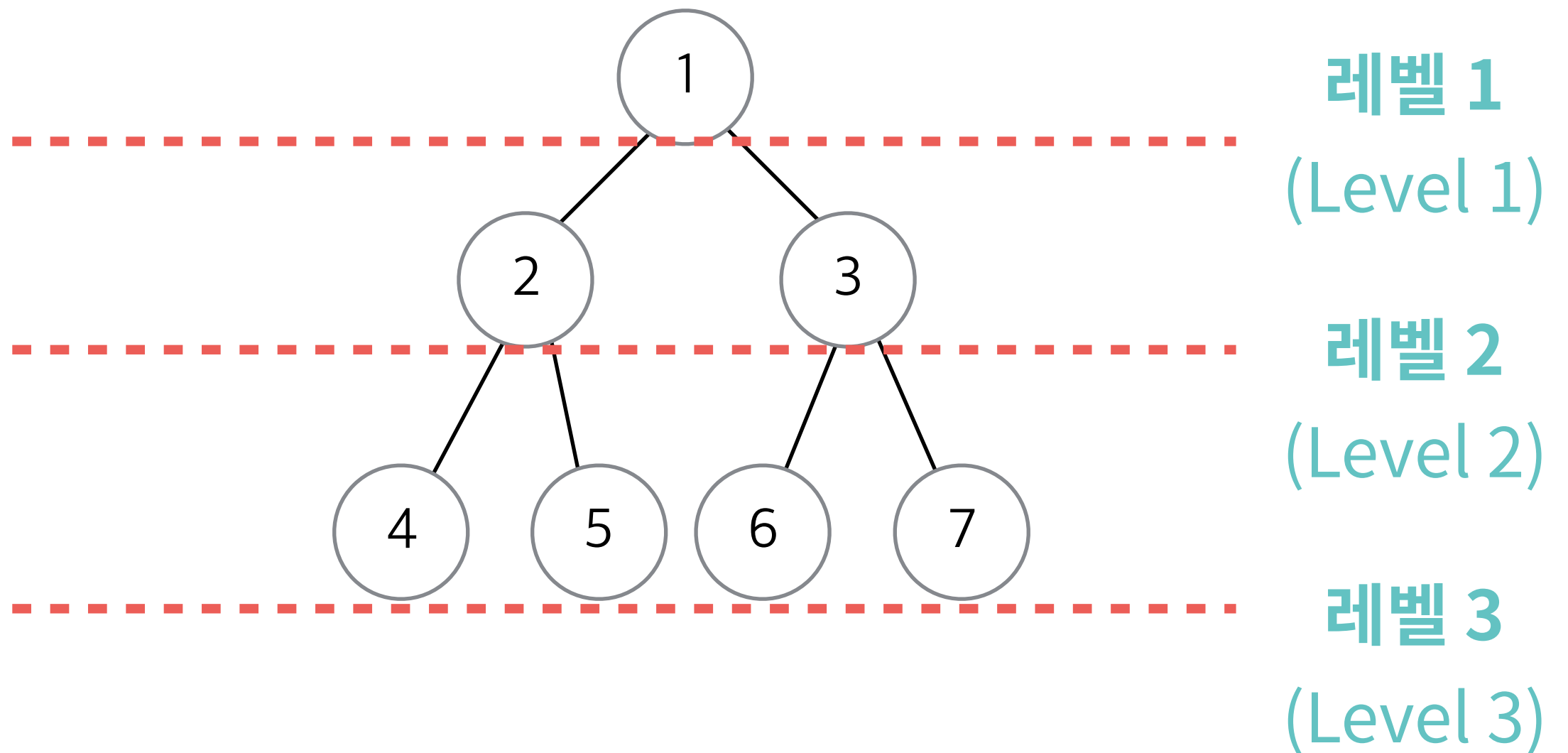
트리

아래와 같이 생긴 자료구조



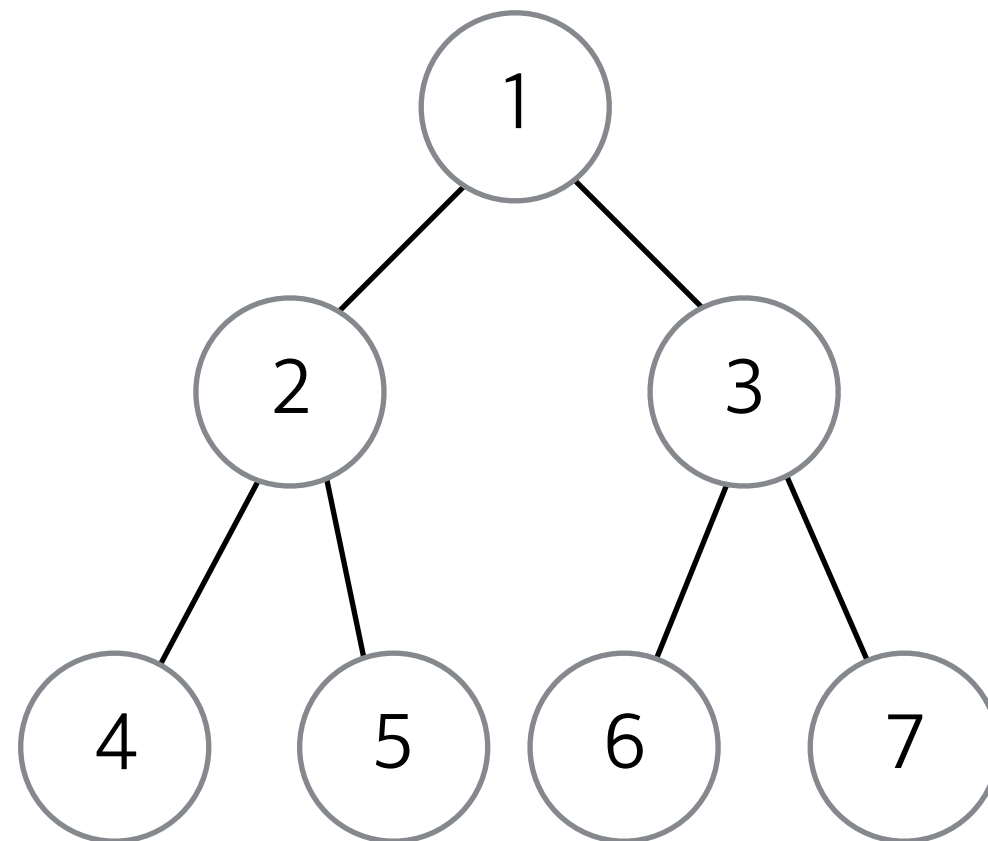
트리

아래와 같이 생긴 자료구조



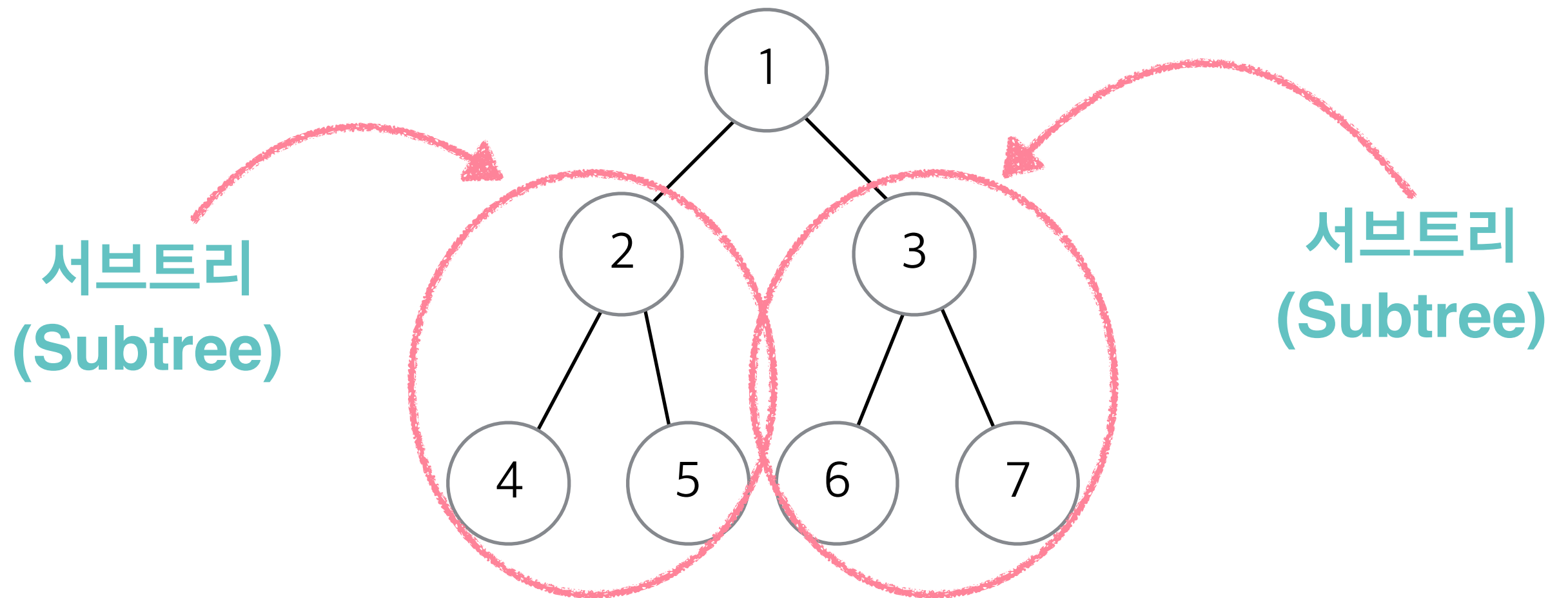
트리의 재귀적 성질

트리는 그 안에 또 트리가 존재한다



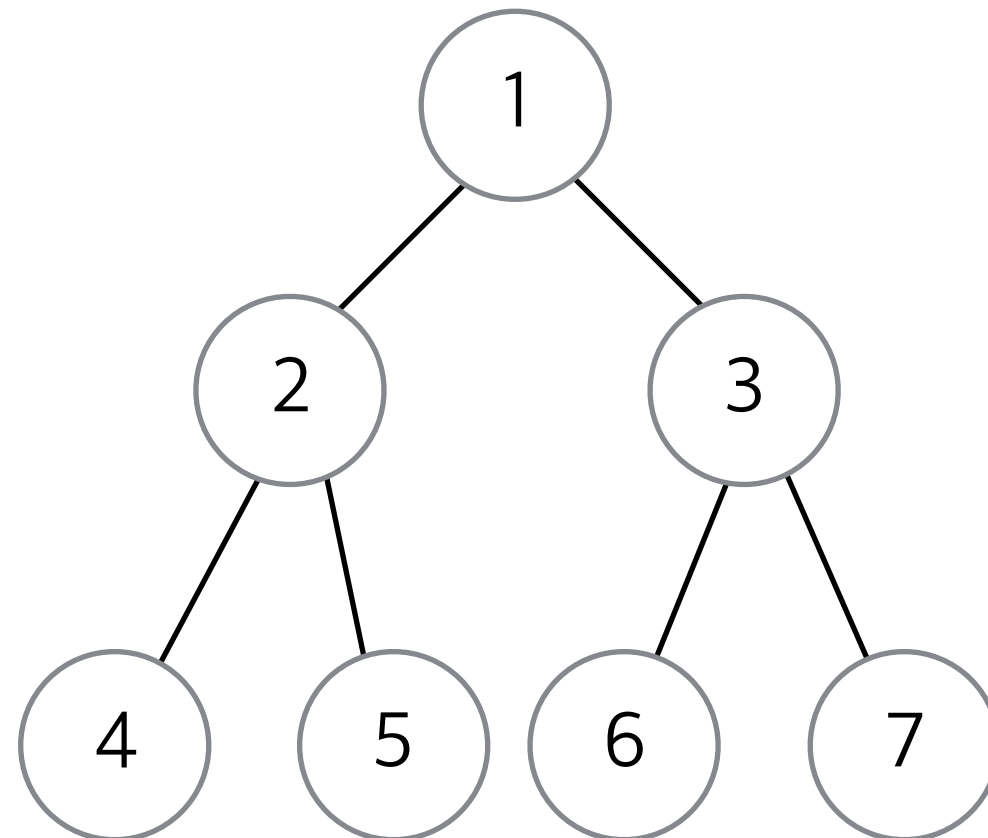
트리의 재귀적 성질

트리는 그 안에 또 트리가 존재한다



트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함



`/* elice */`

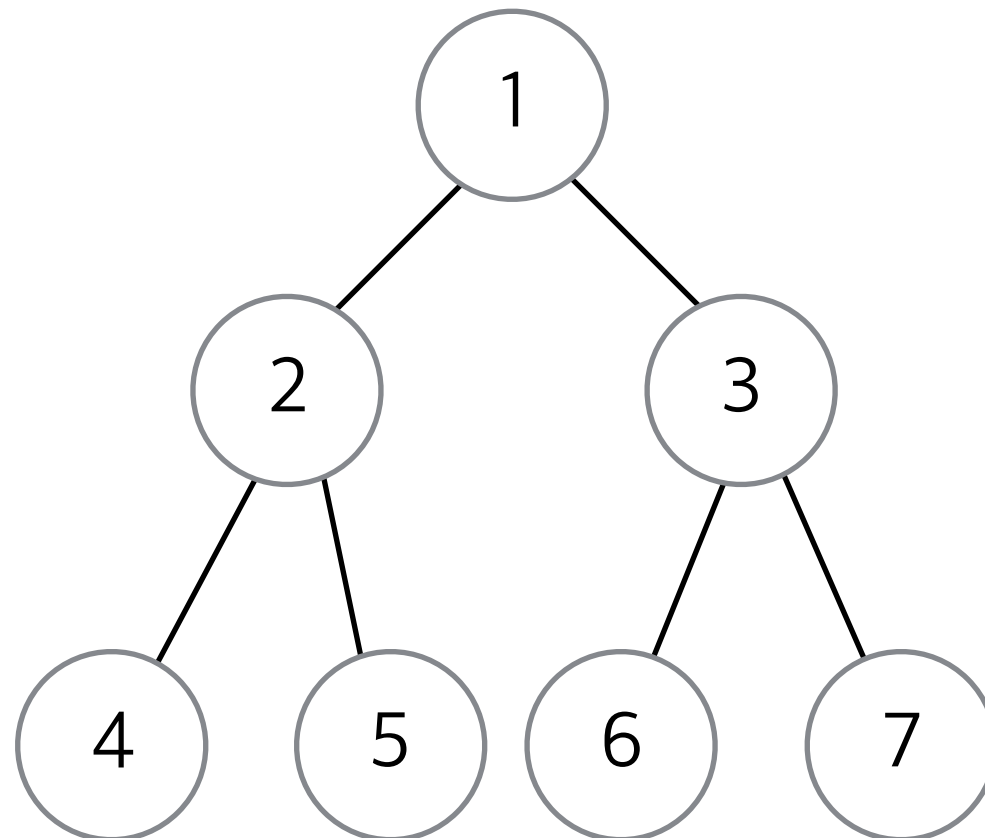
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

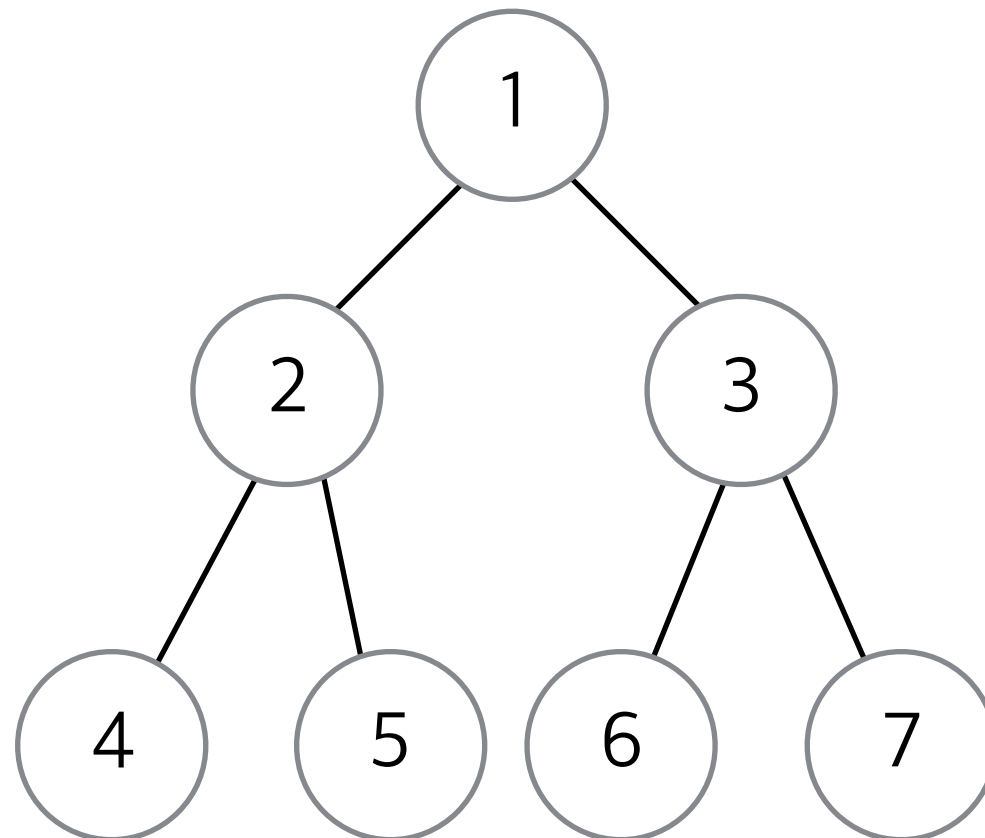
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

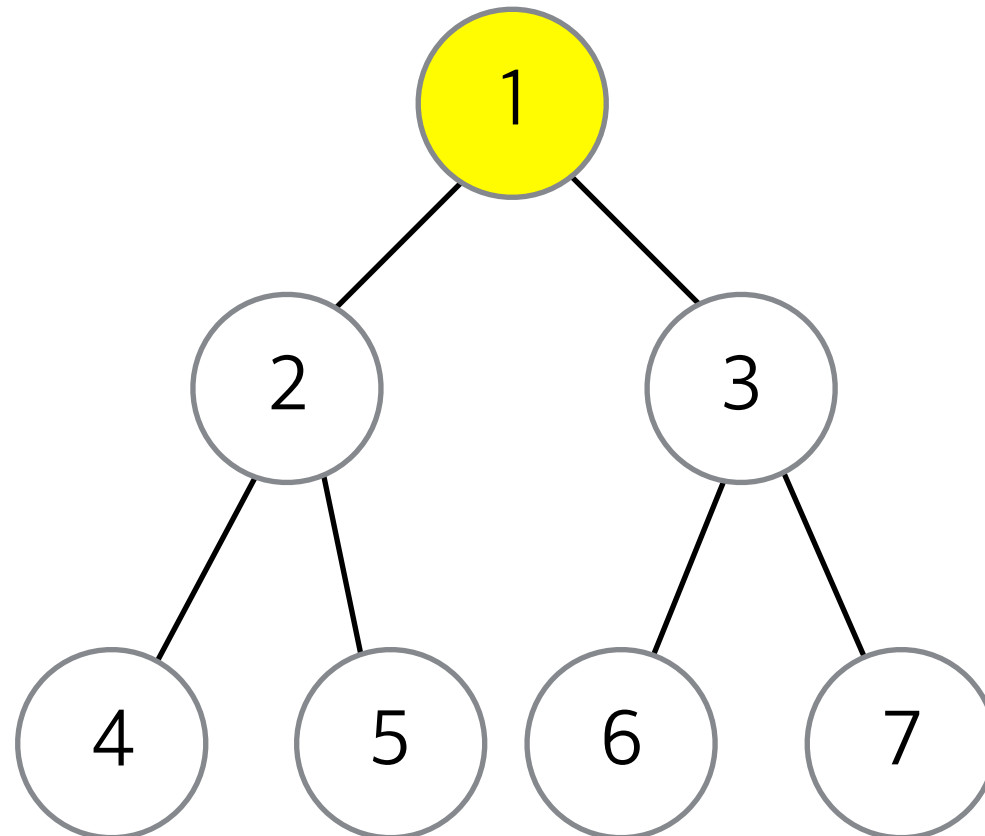
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



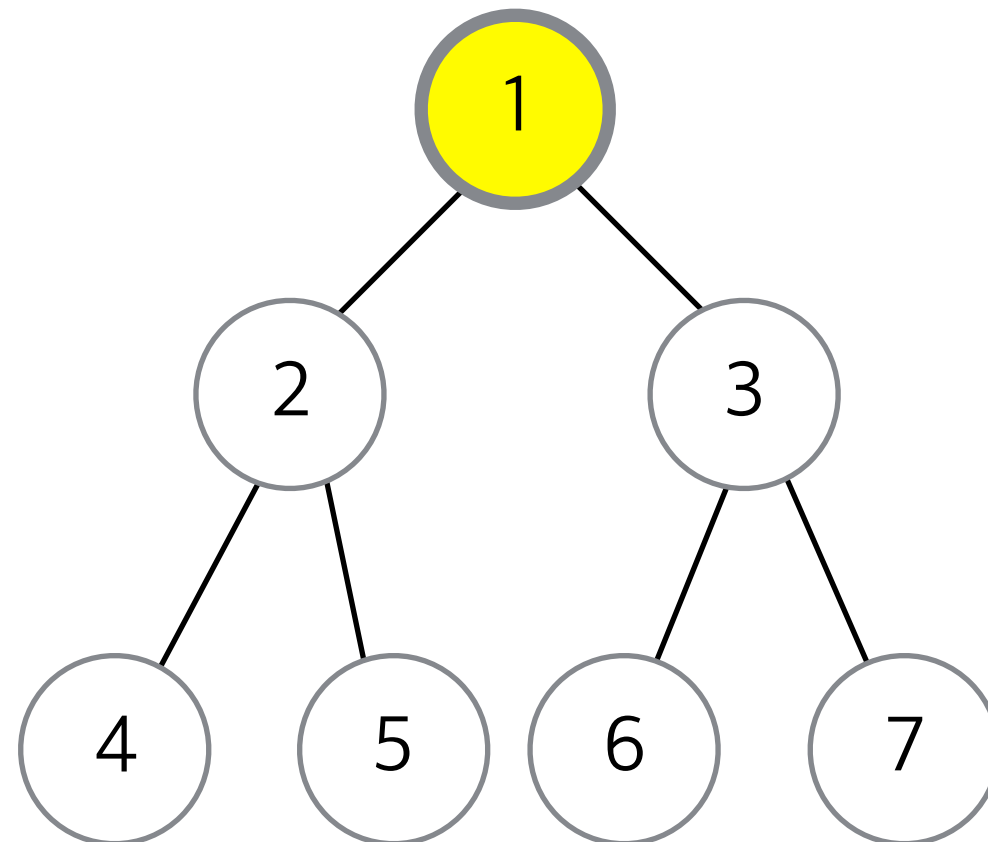
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



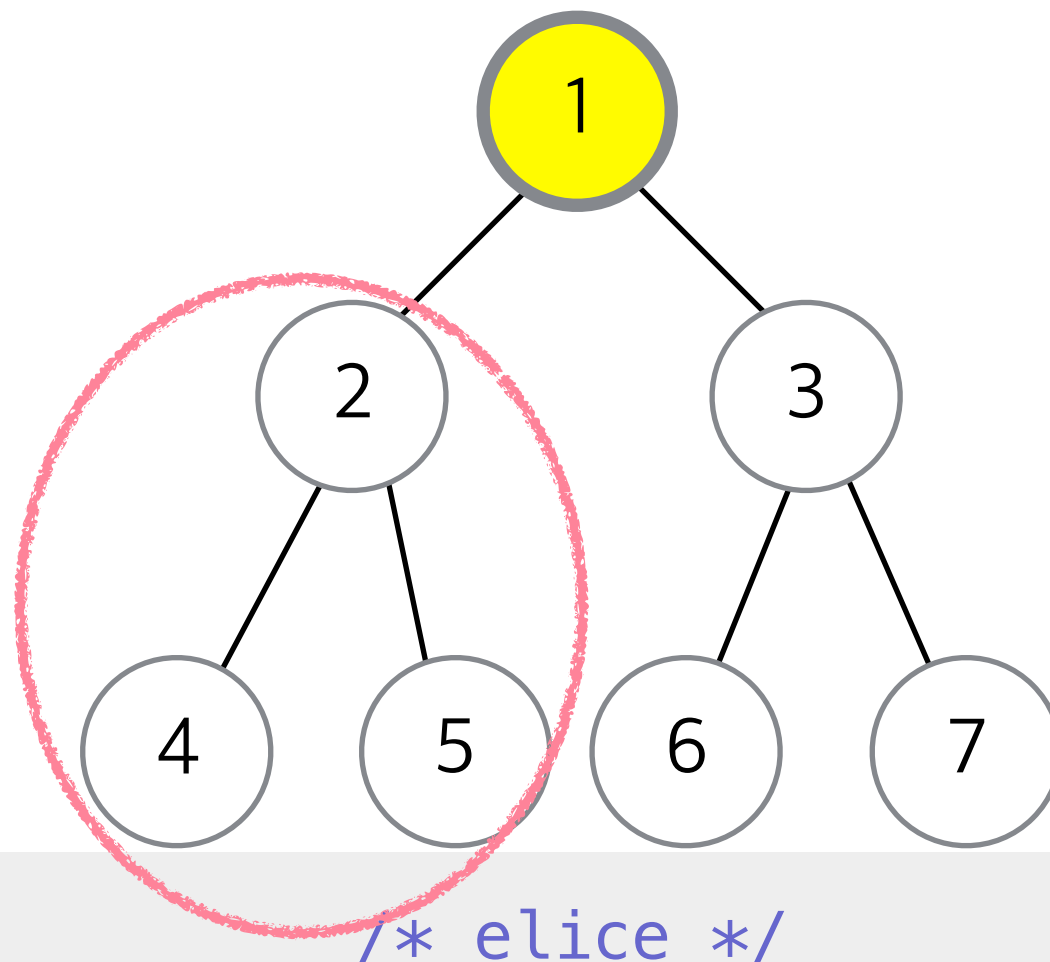
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



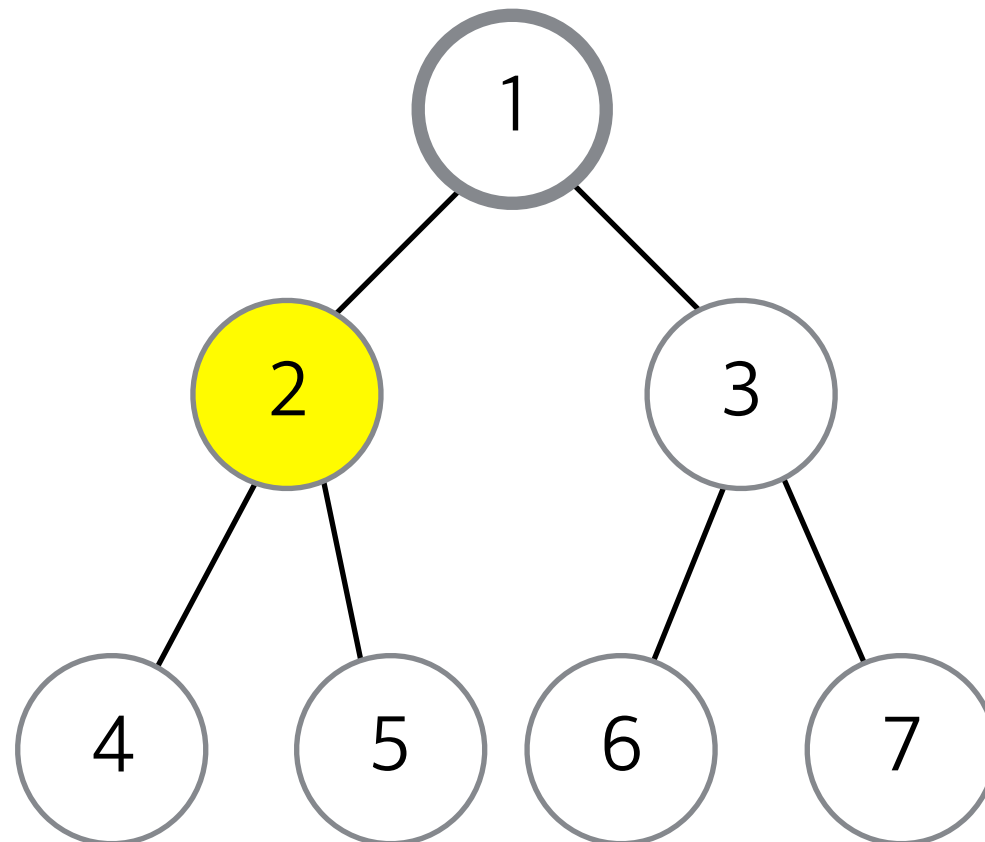
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



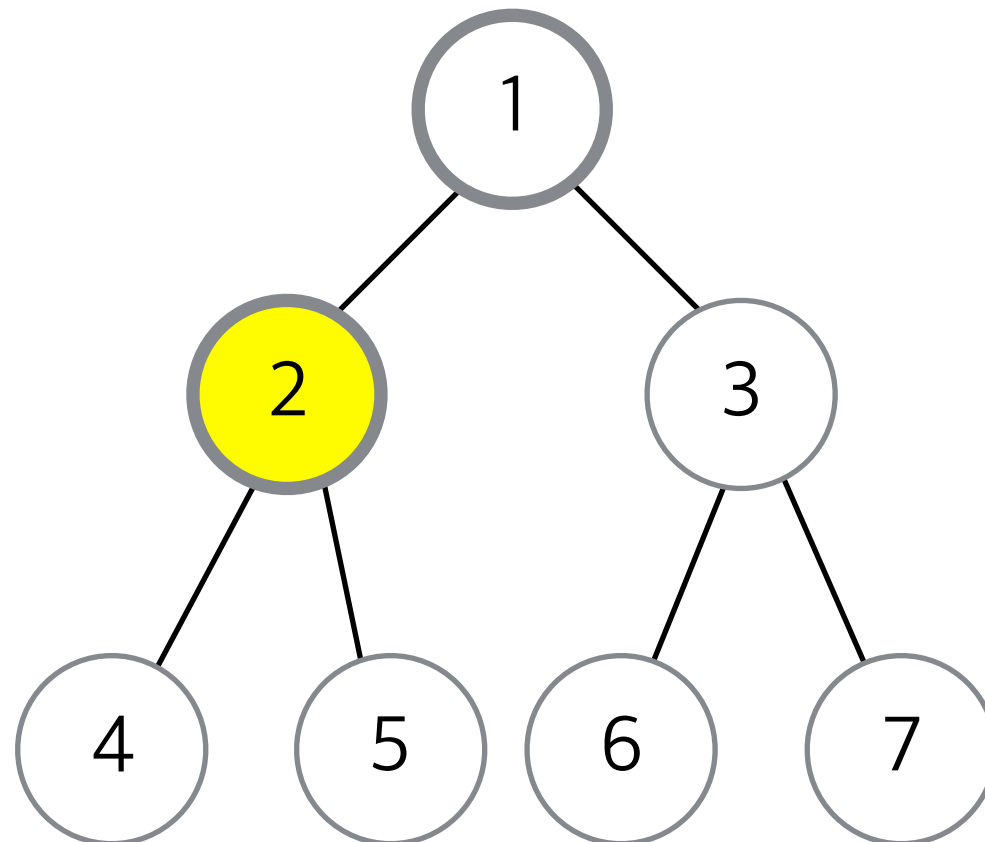
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

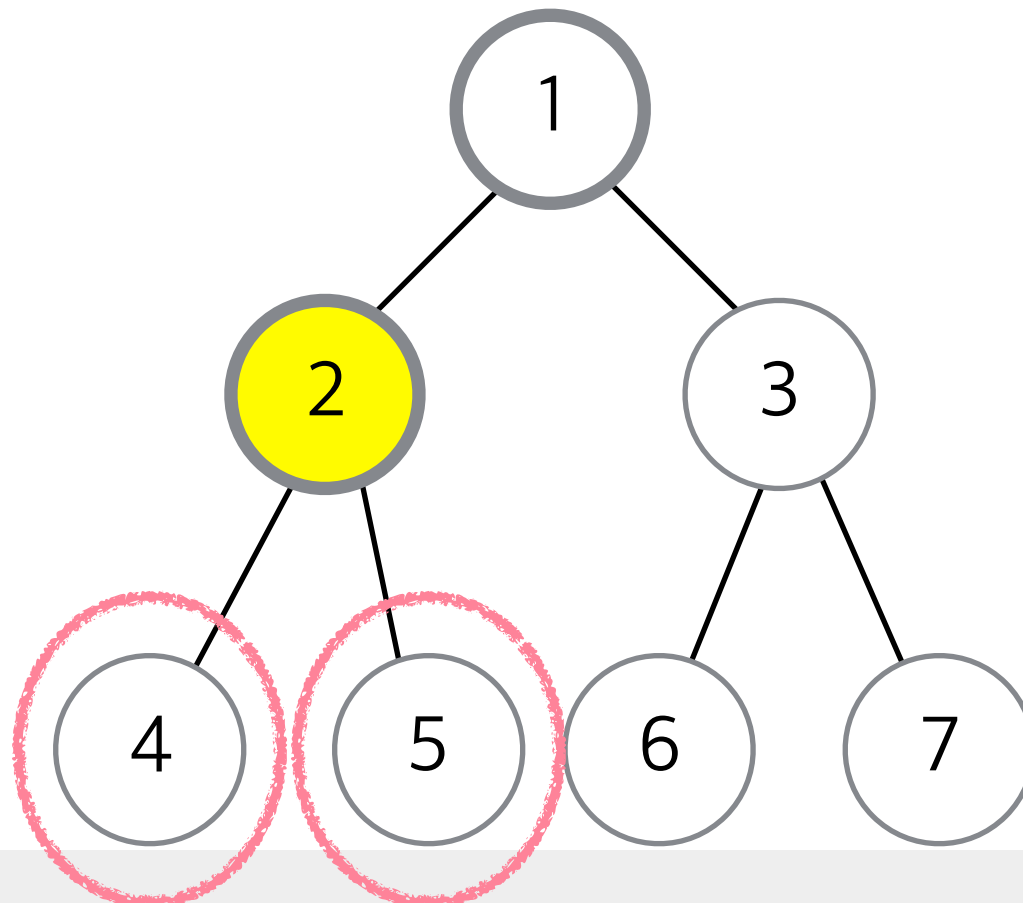
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



/* elice */

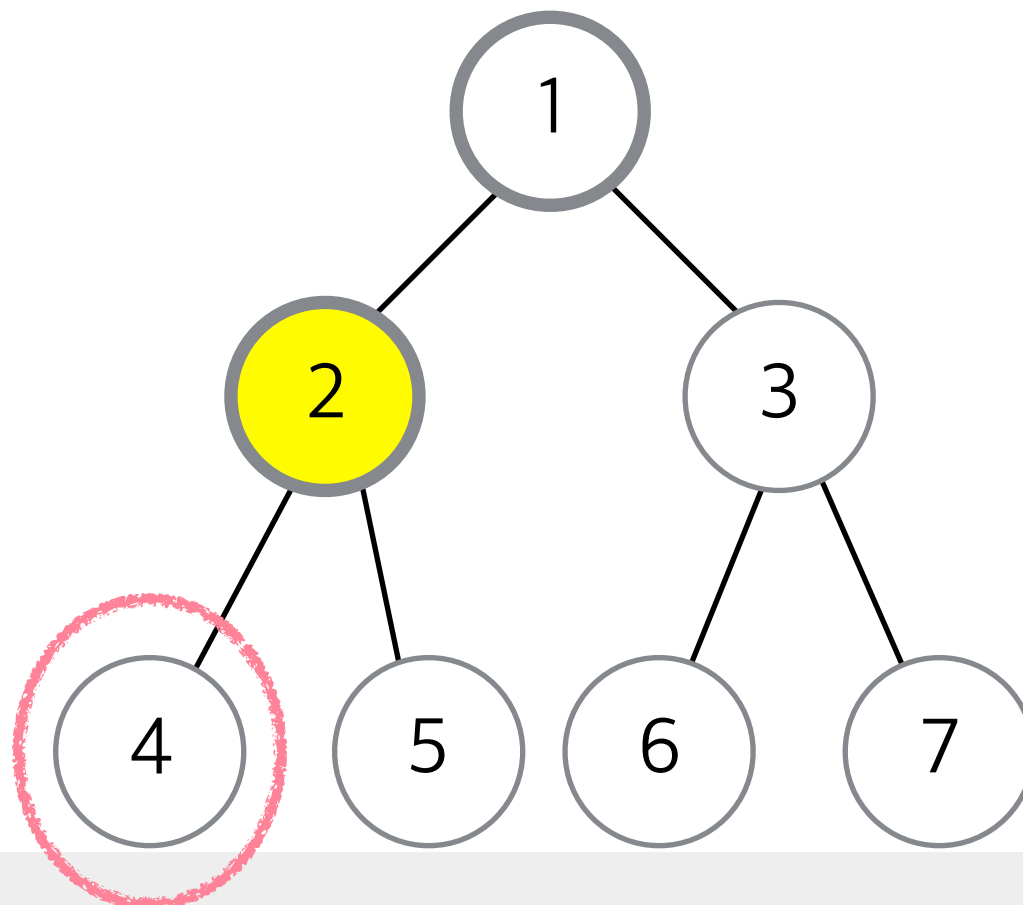
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

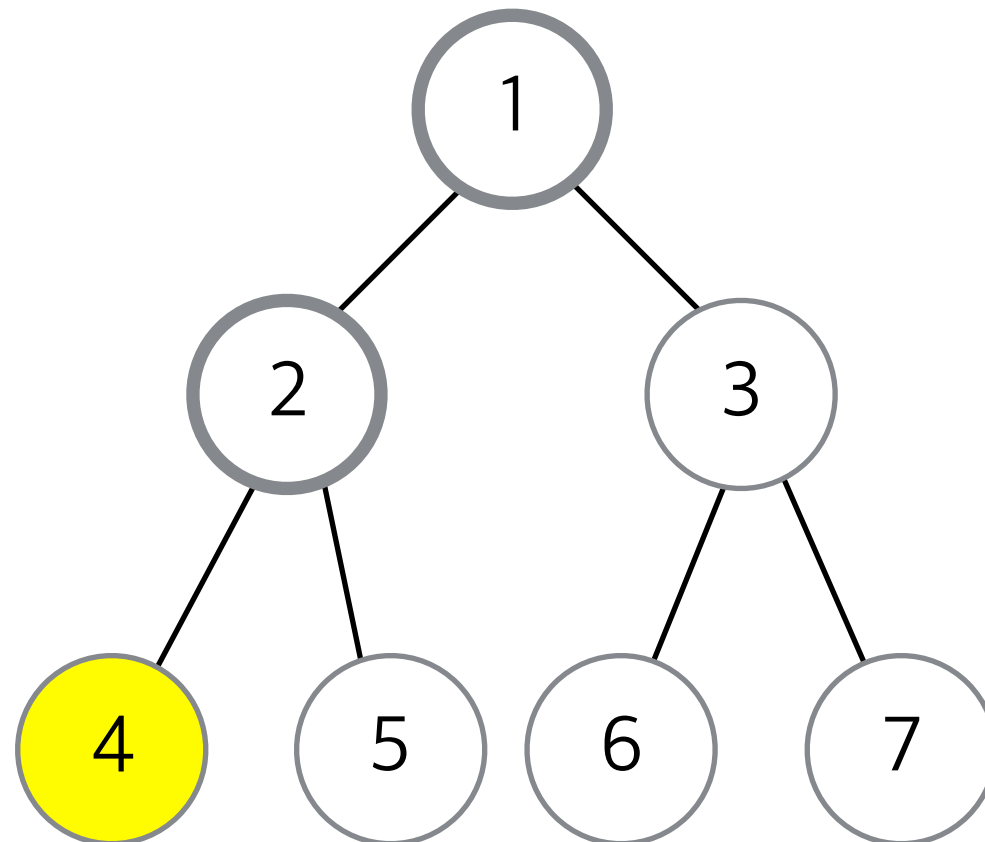
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

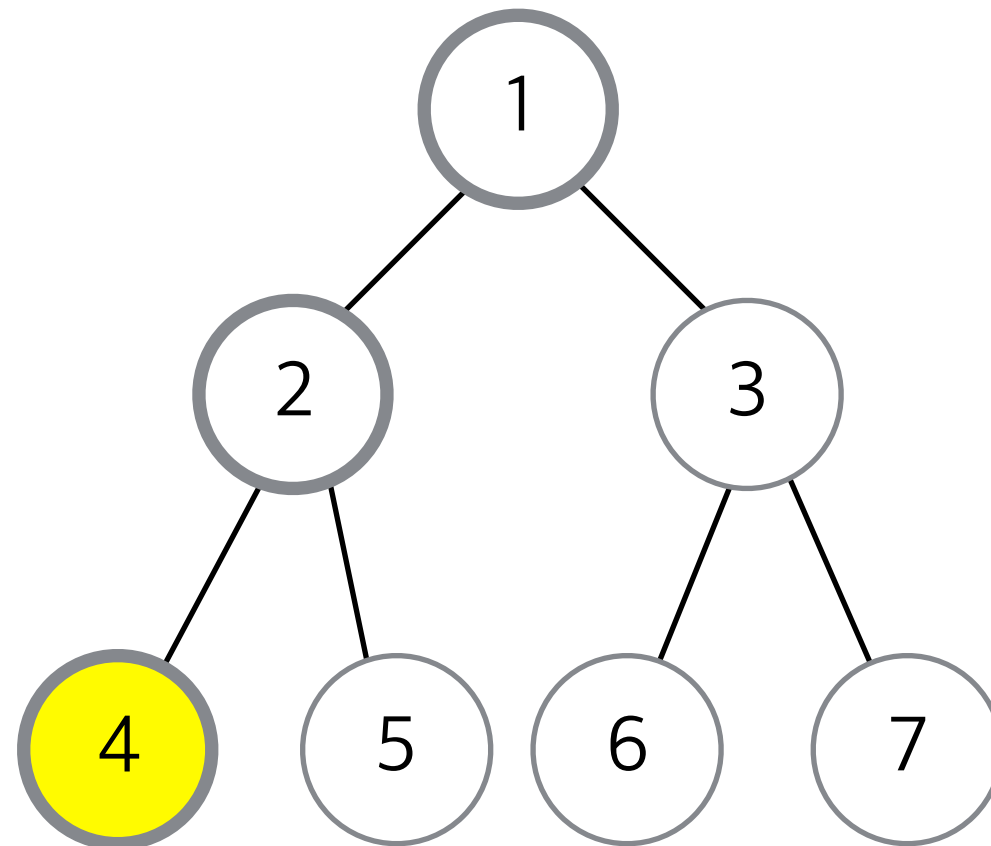
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

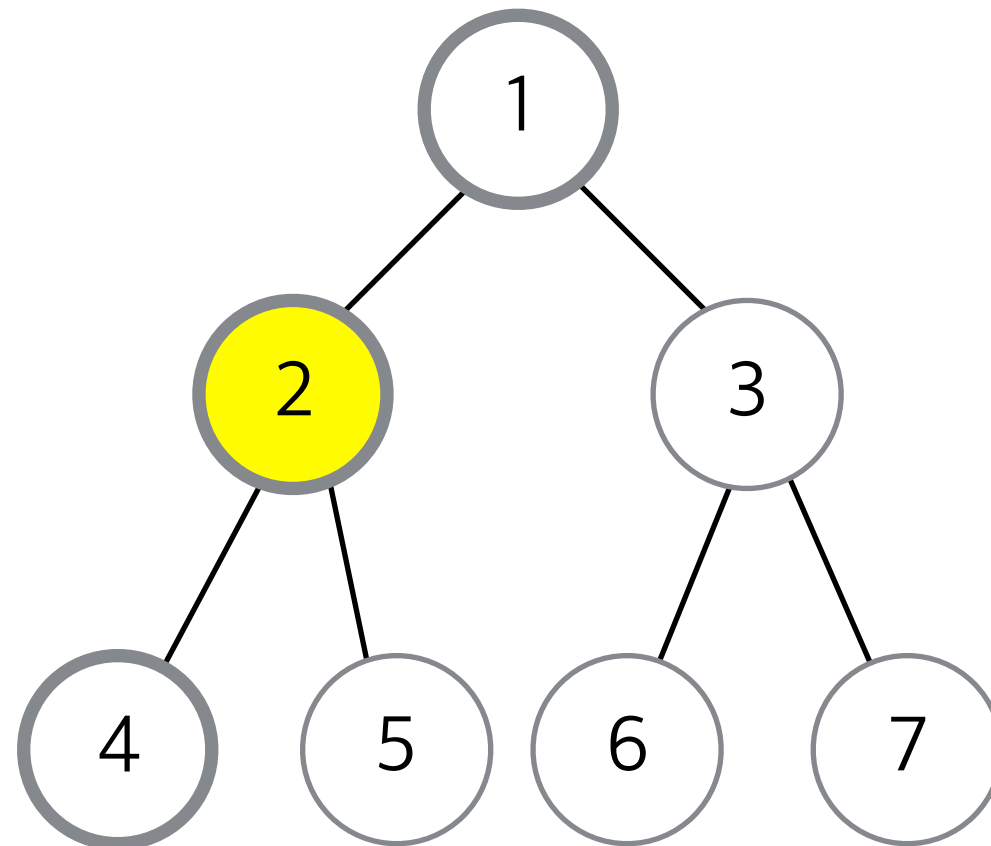
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

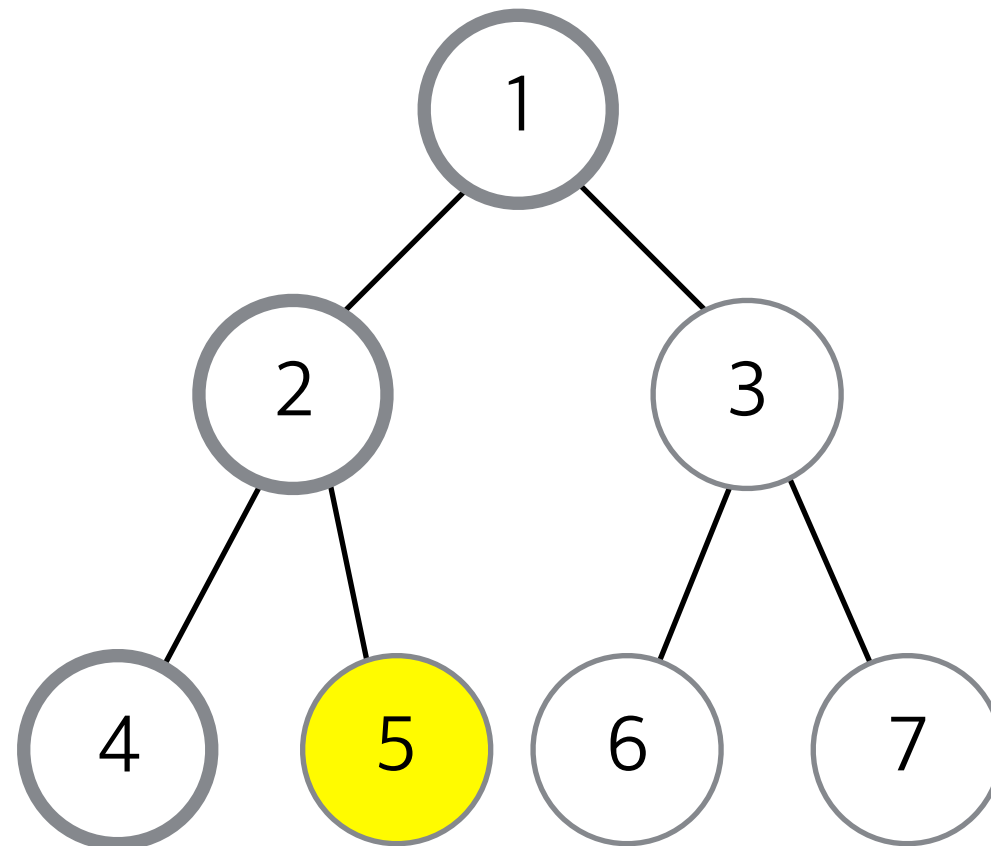
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

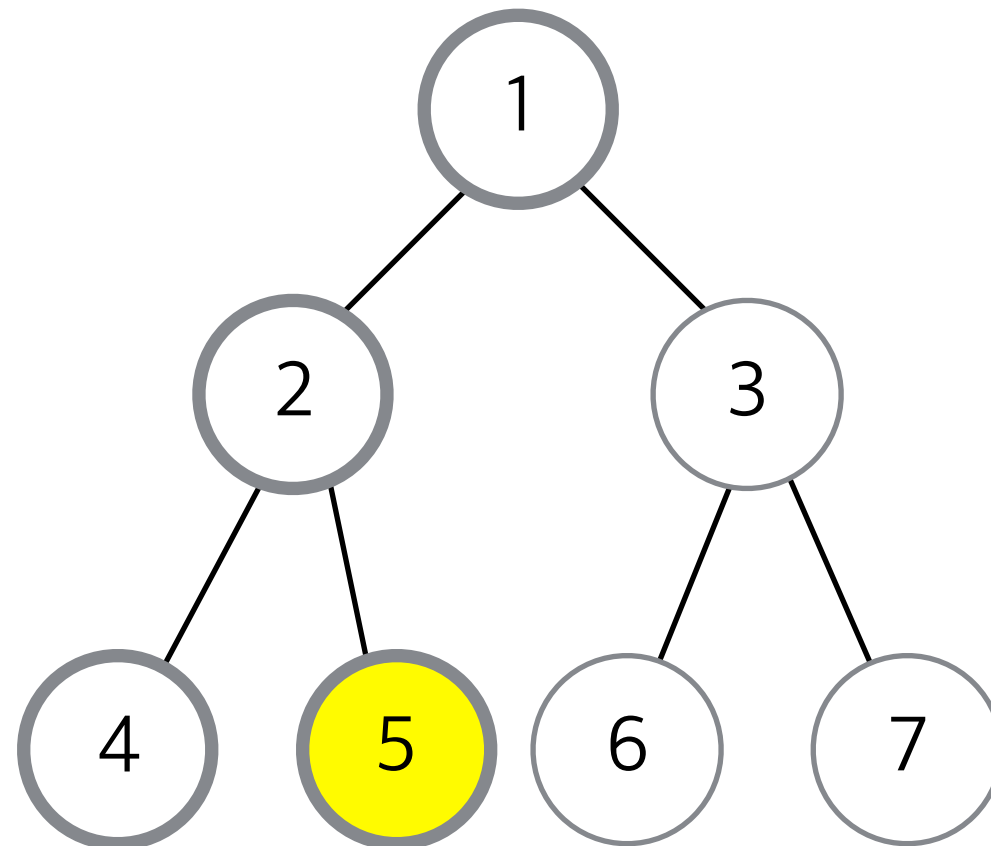
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

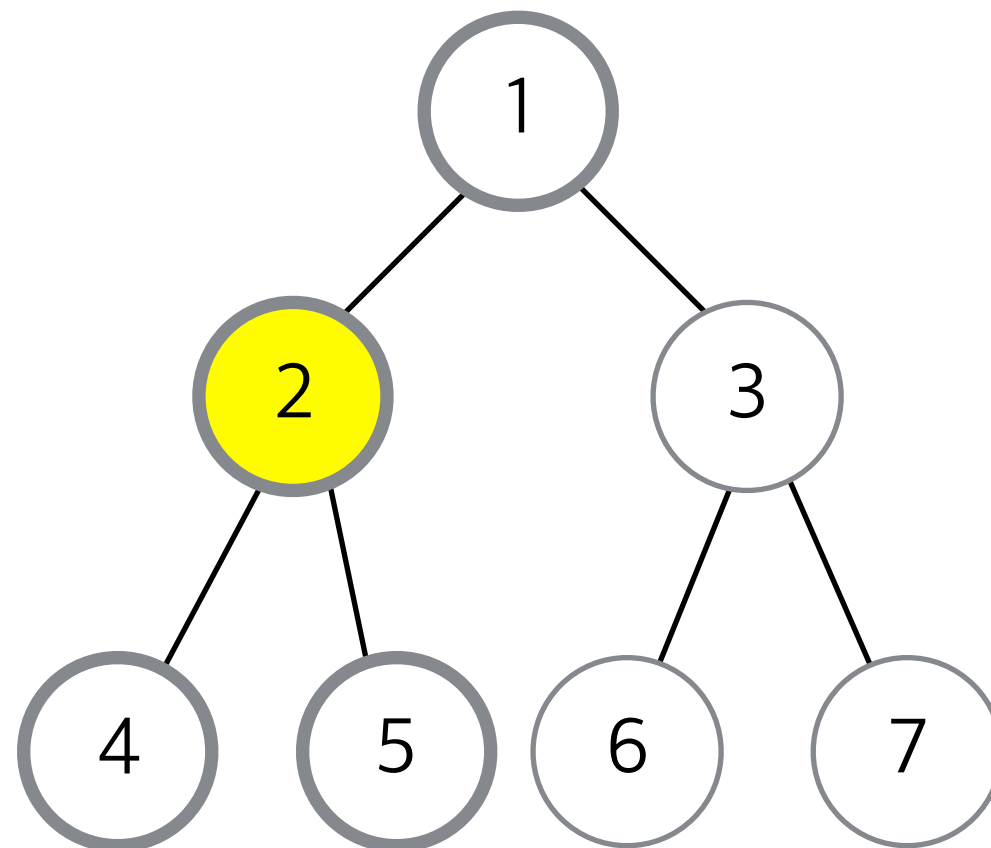
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

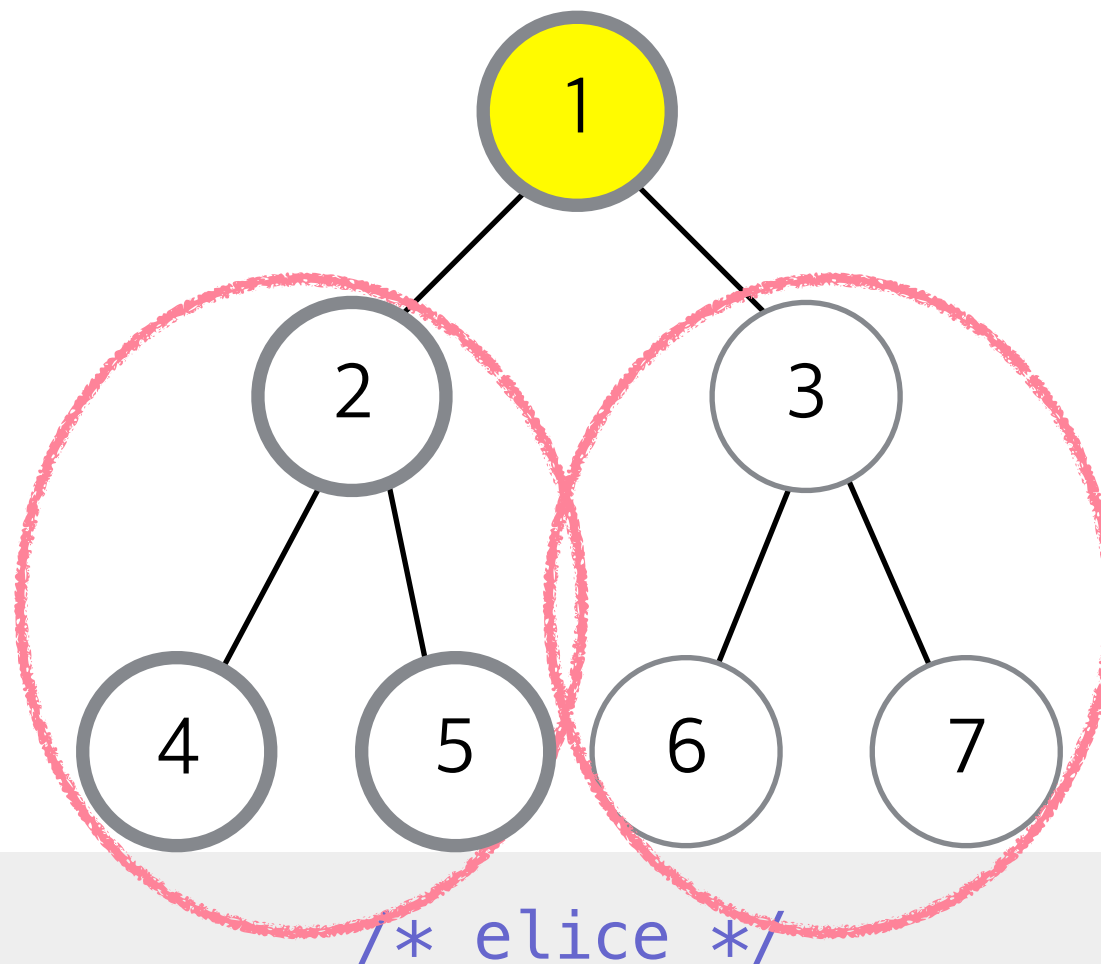
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



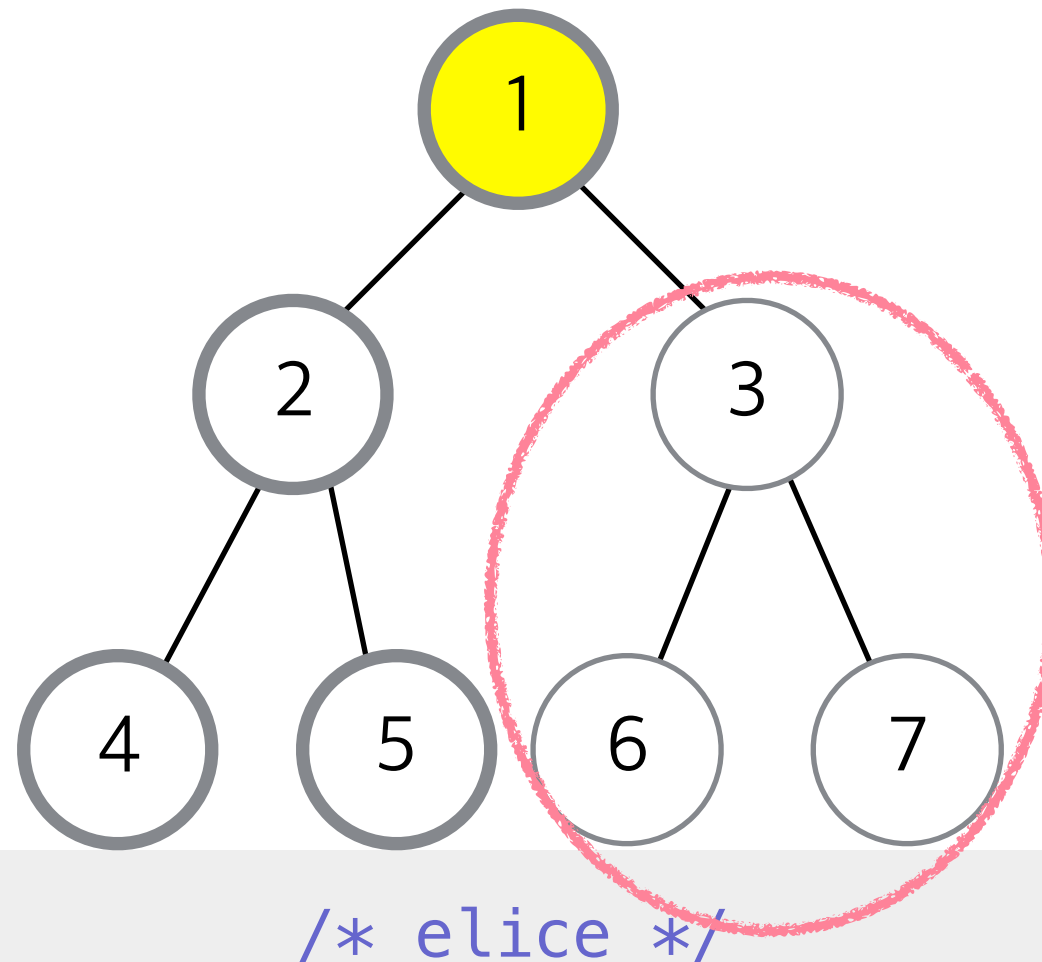
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



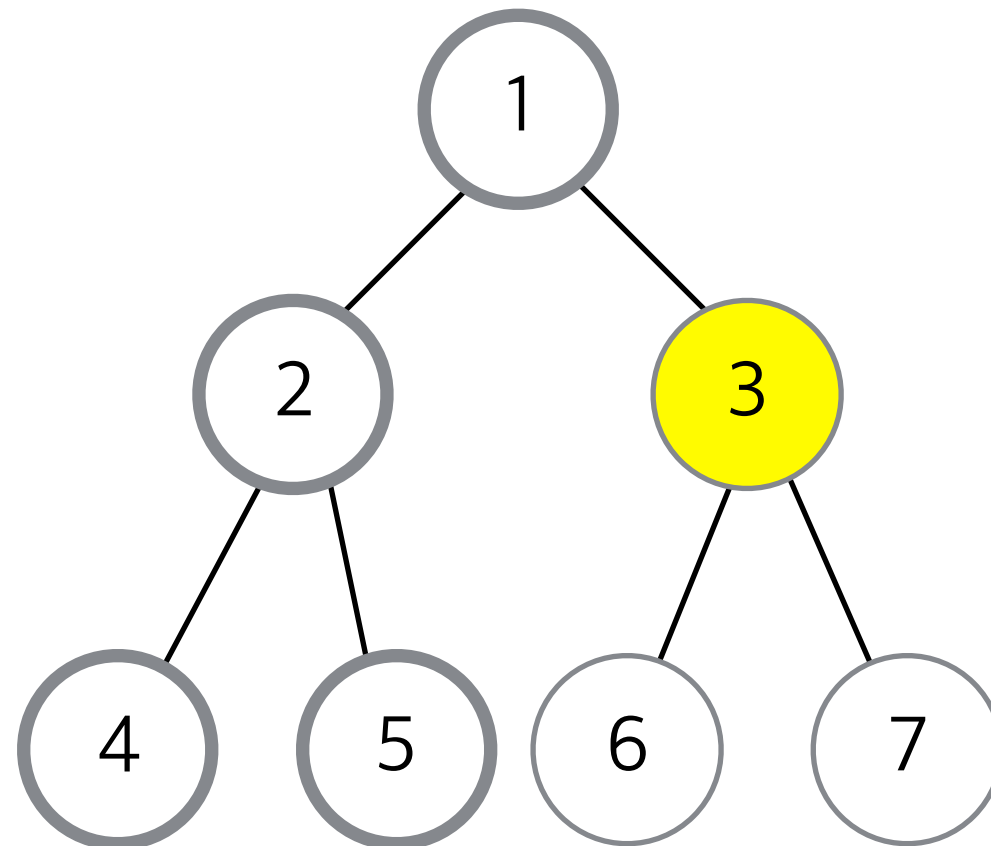
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

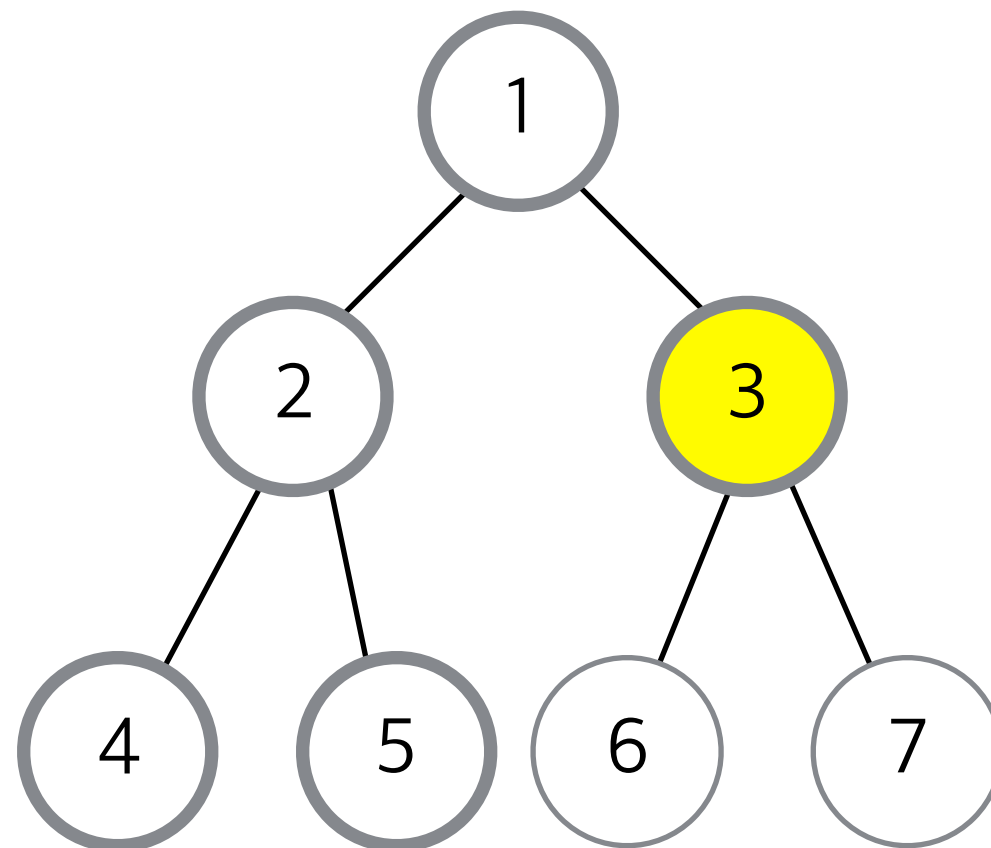
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

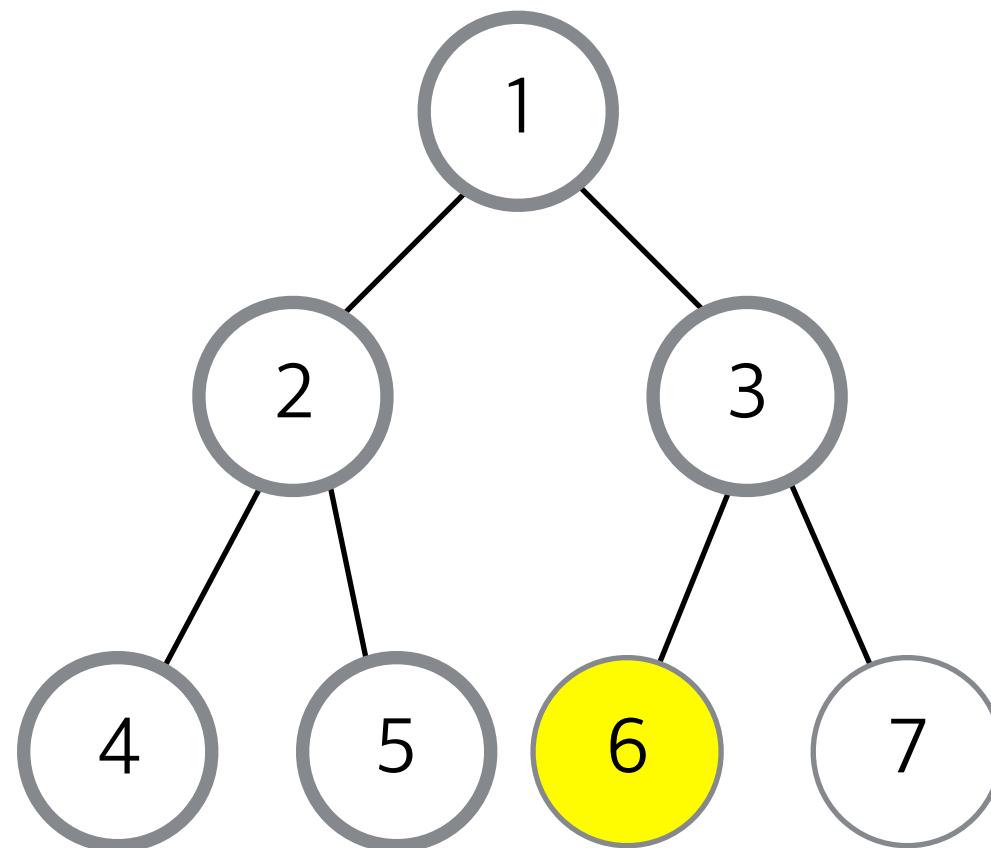
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

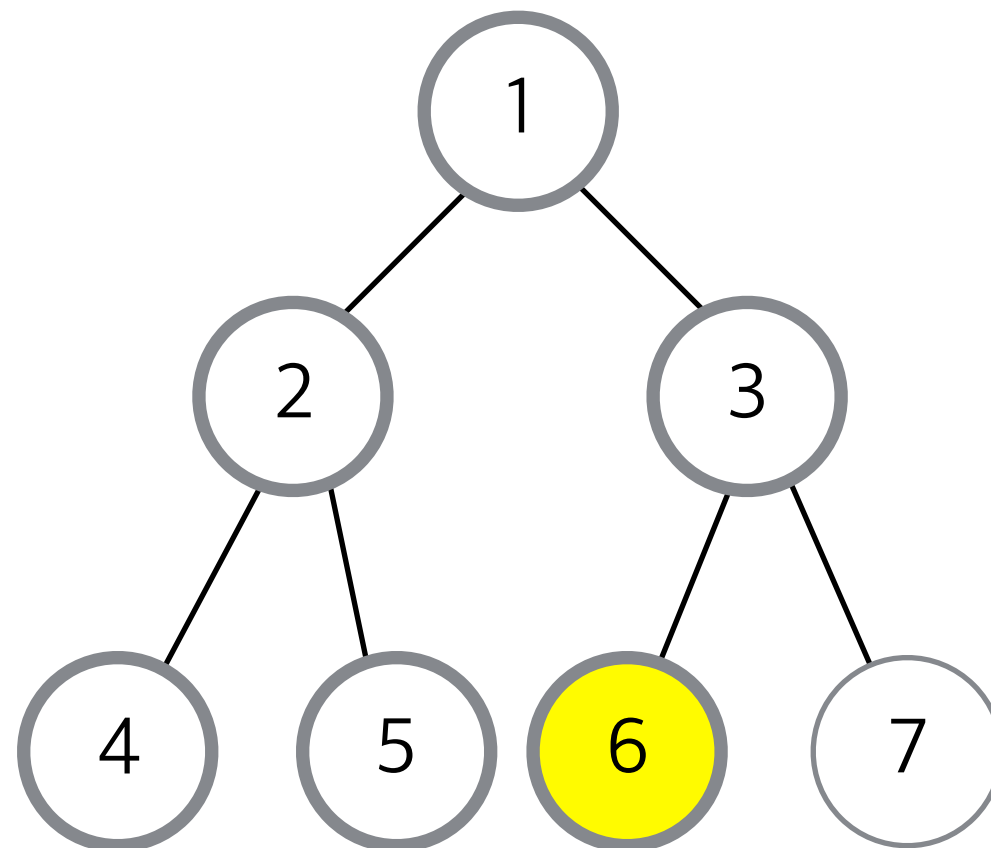
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

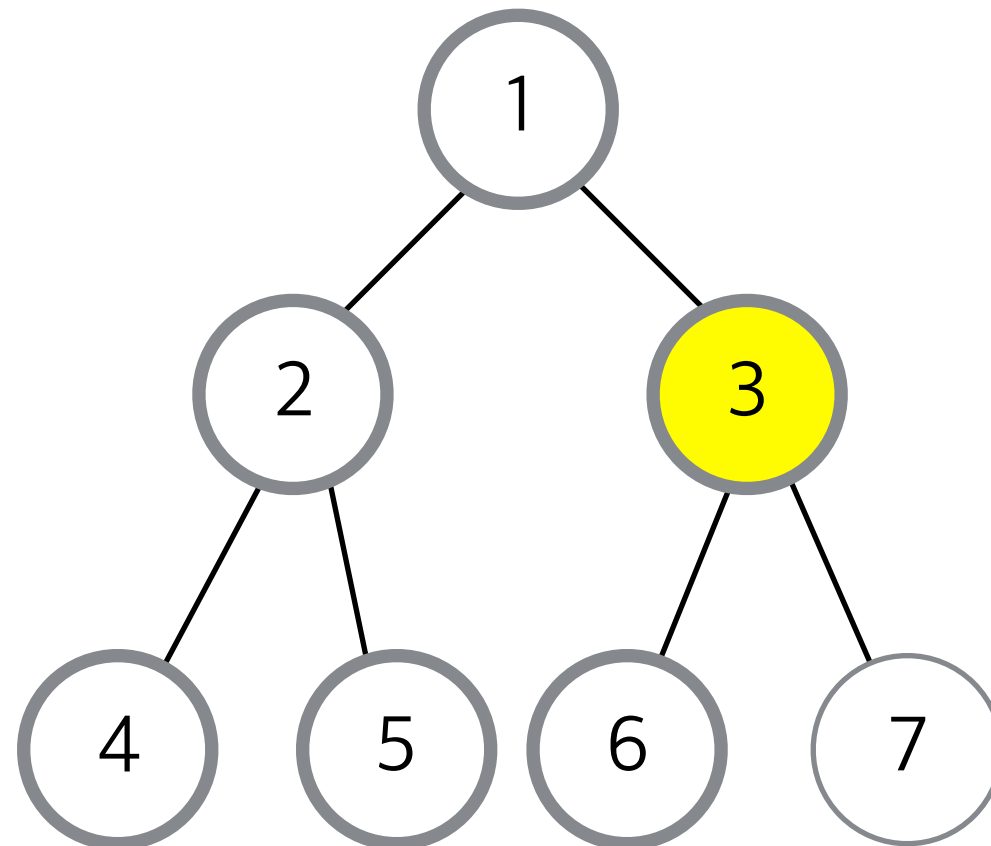
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

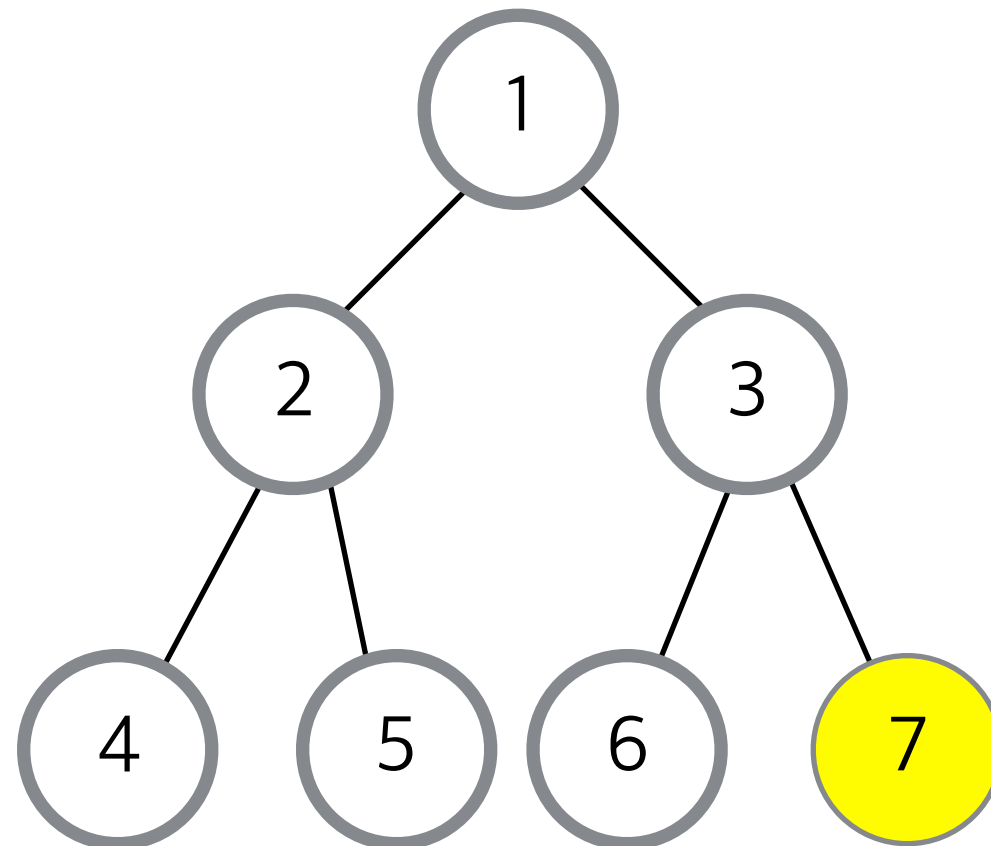
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

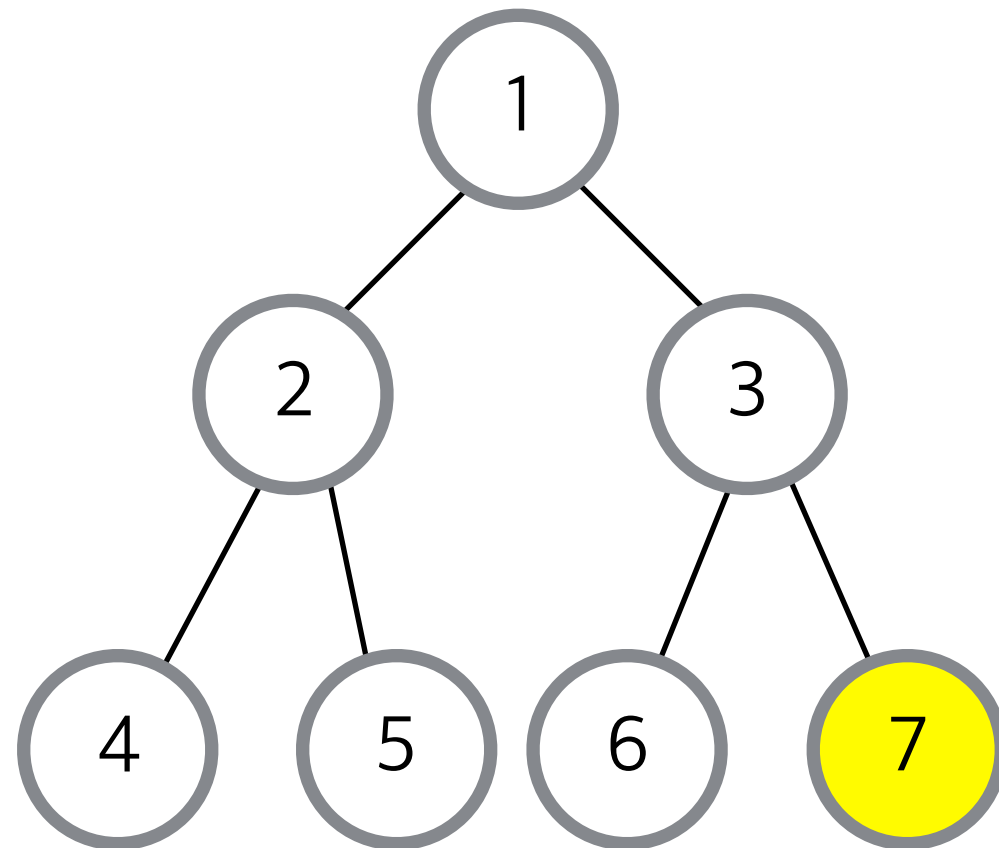
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

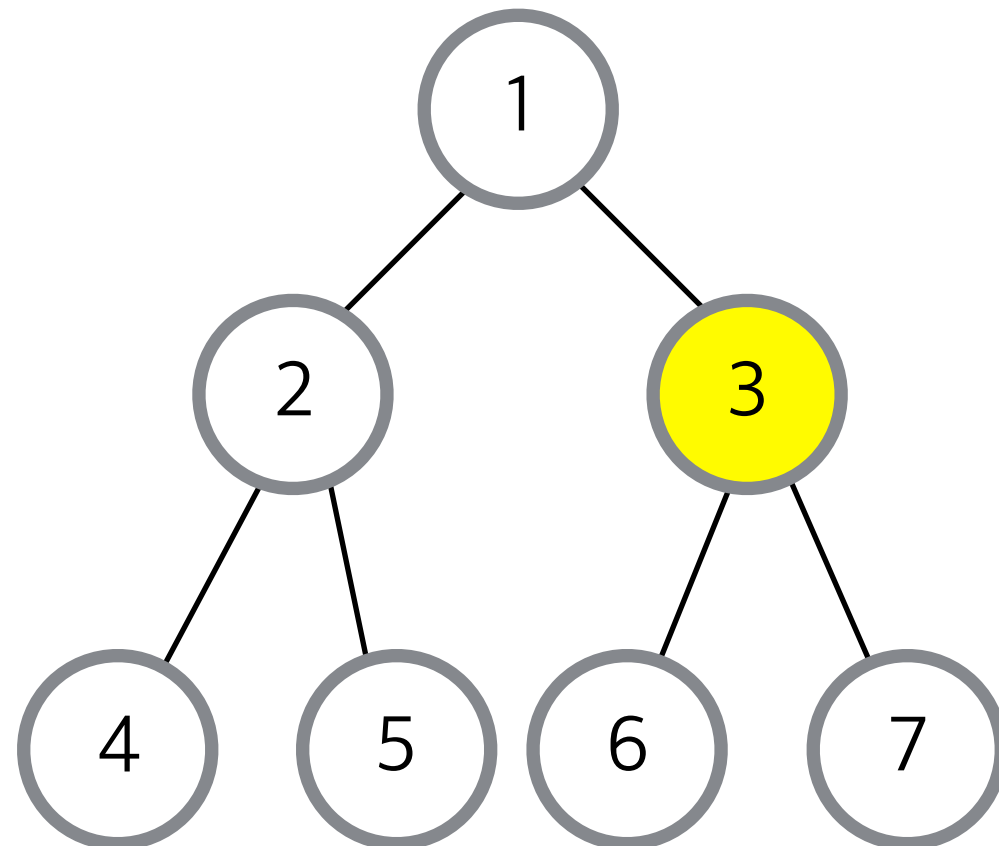
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

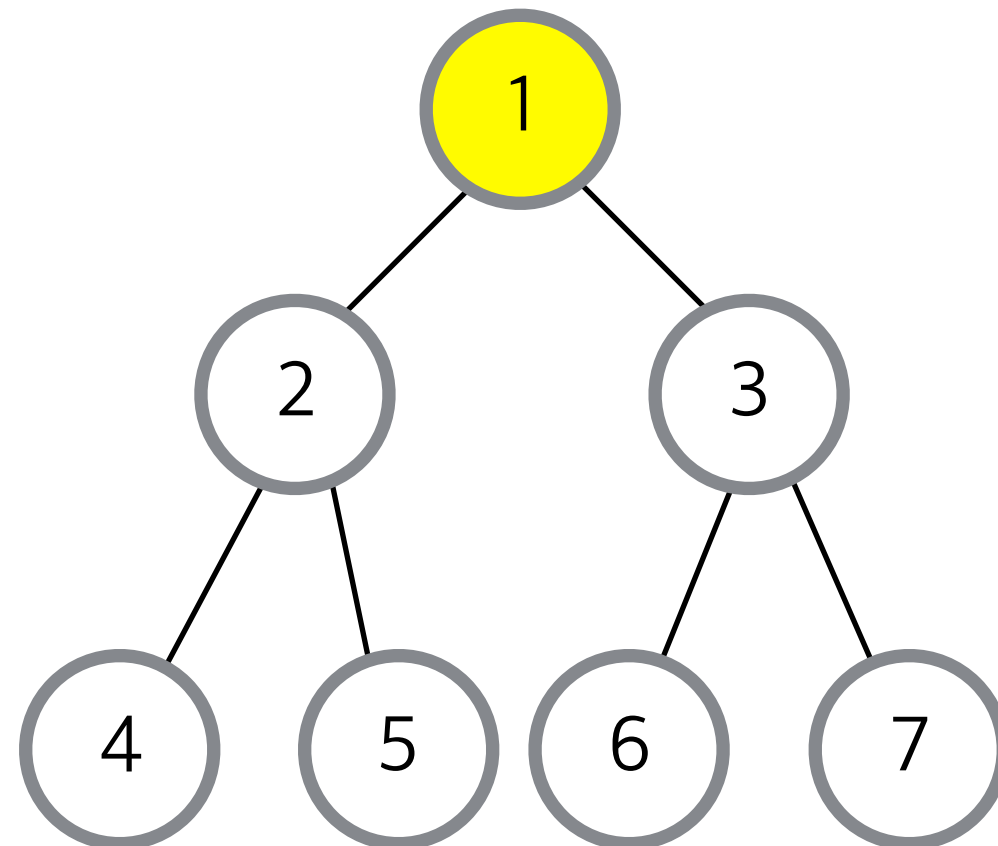
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

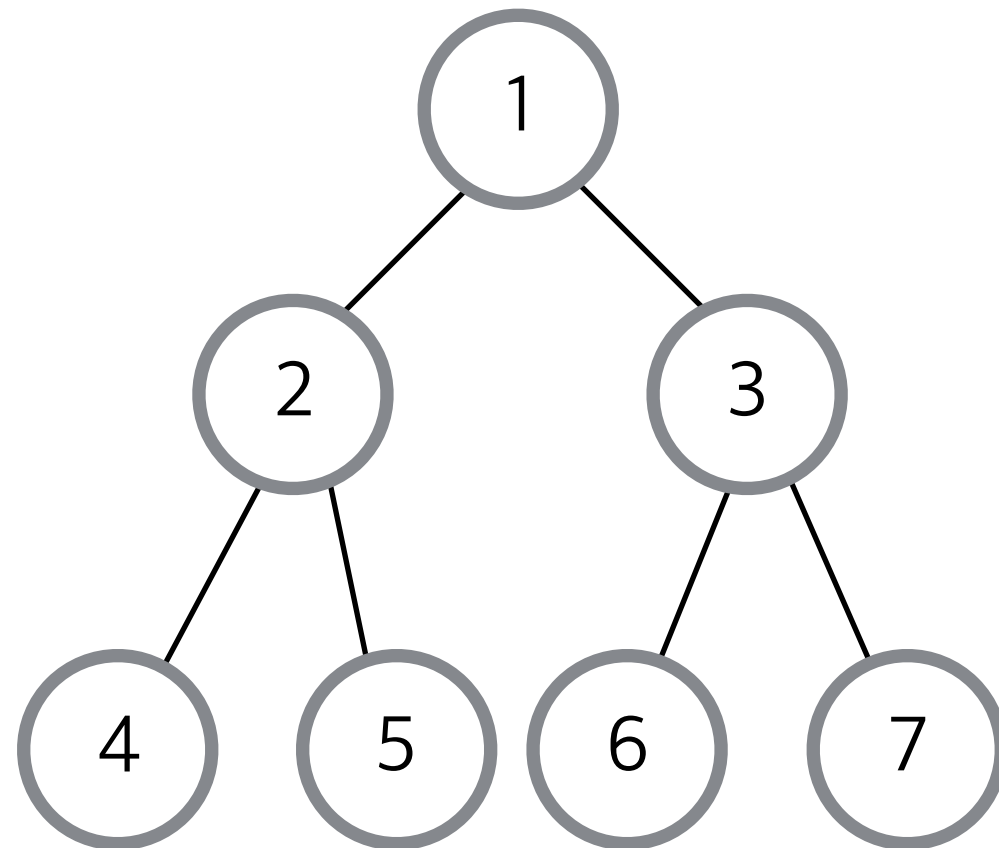
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

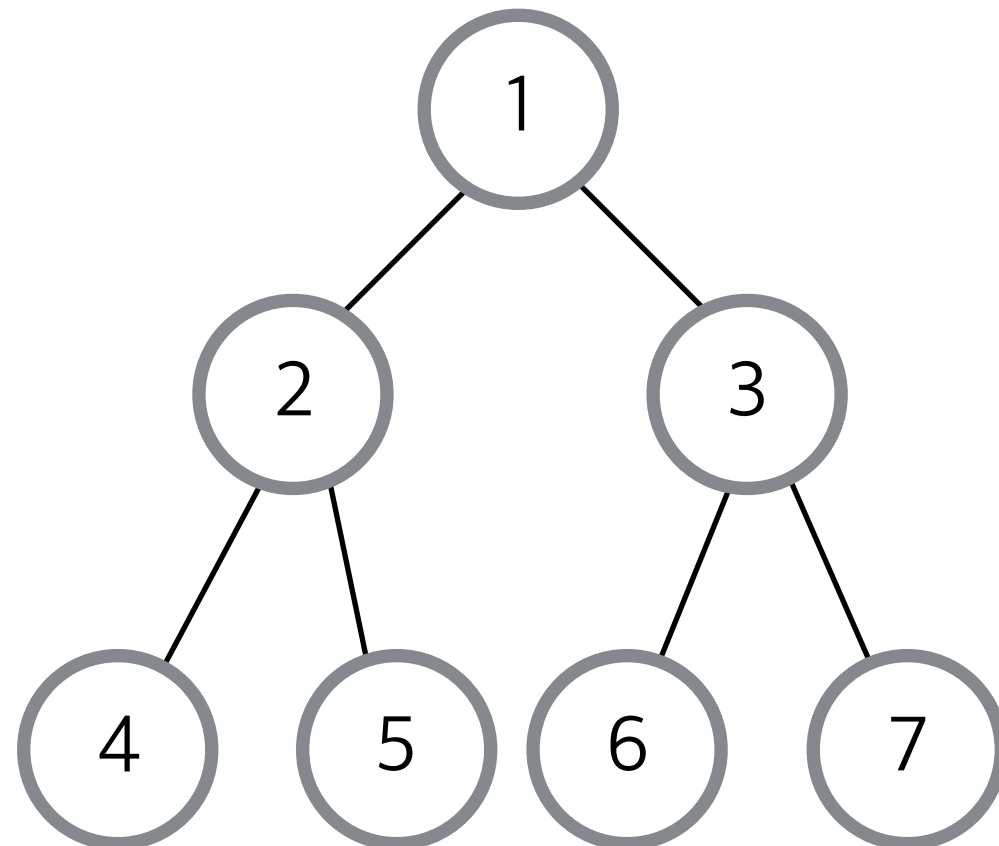
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R 1 2 4 5 3 6 7

중위순회 : L - Root - R

후위순회 : L - R - Root



`/* elice */`

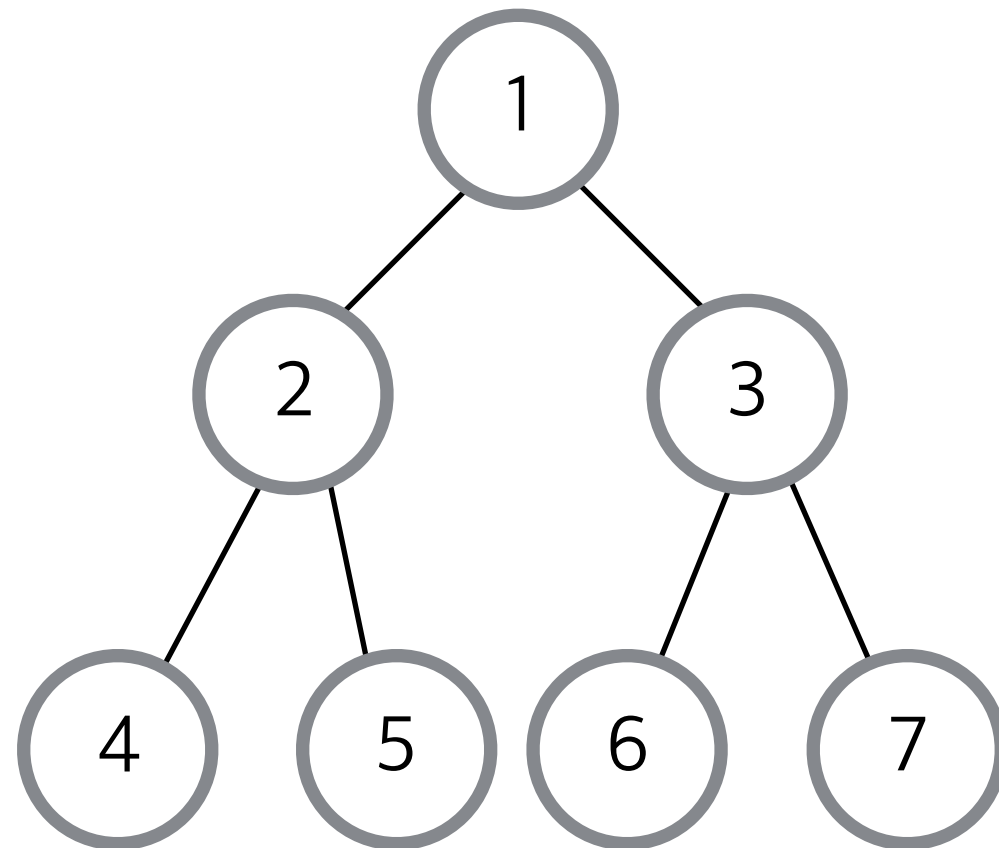
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R 1 2 4 5 3 6 7

중위순회 : L - Root - R

후위순회 : L - R - Root



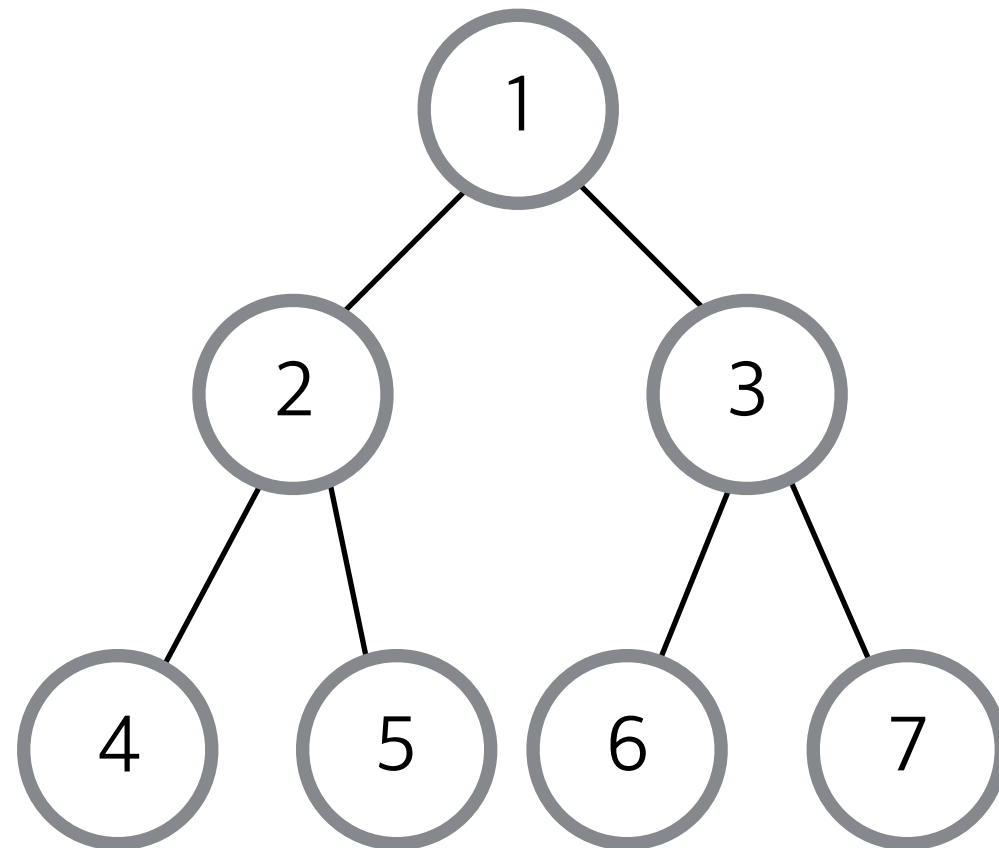
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R 1 2 4 5 3 6 7

중위순회 : L - Root - R 4 2 5 1 6 3 7

후위순회 : L - R - Root



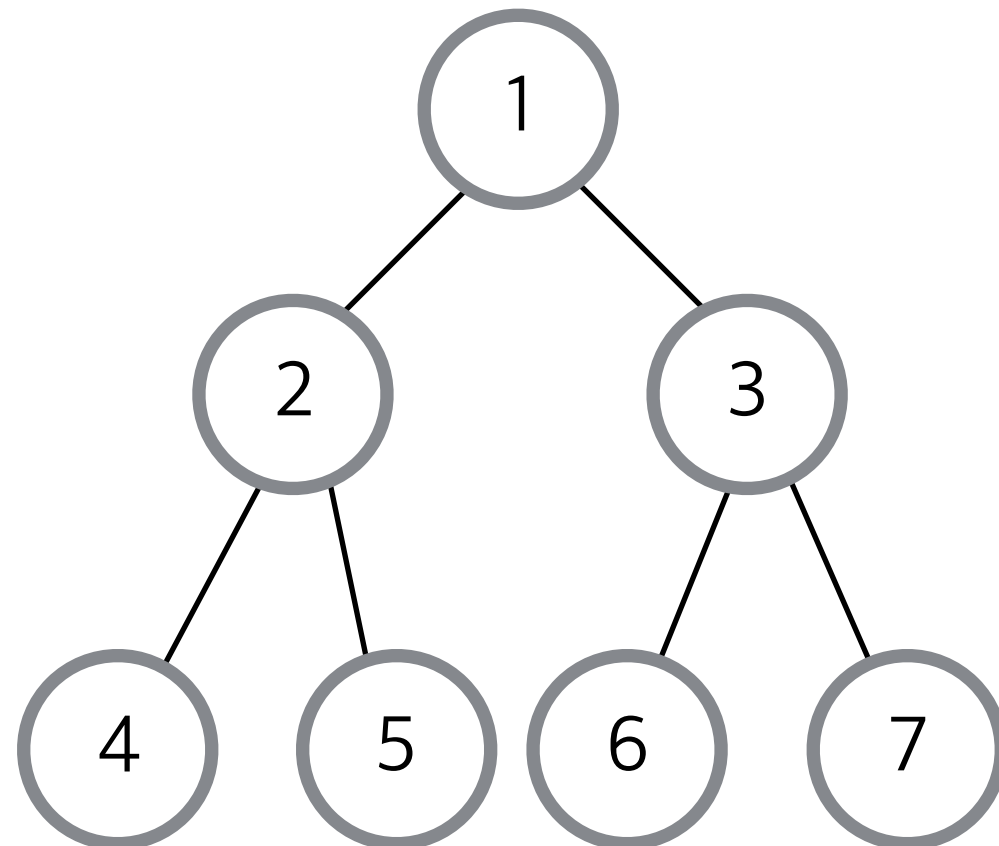
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R 1 2 4 5 3 6 7

중위순회 : L - Root - R 4 2 5 1 6 3 7

후위순회 : L - R - Root



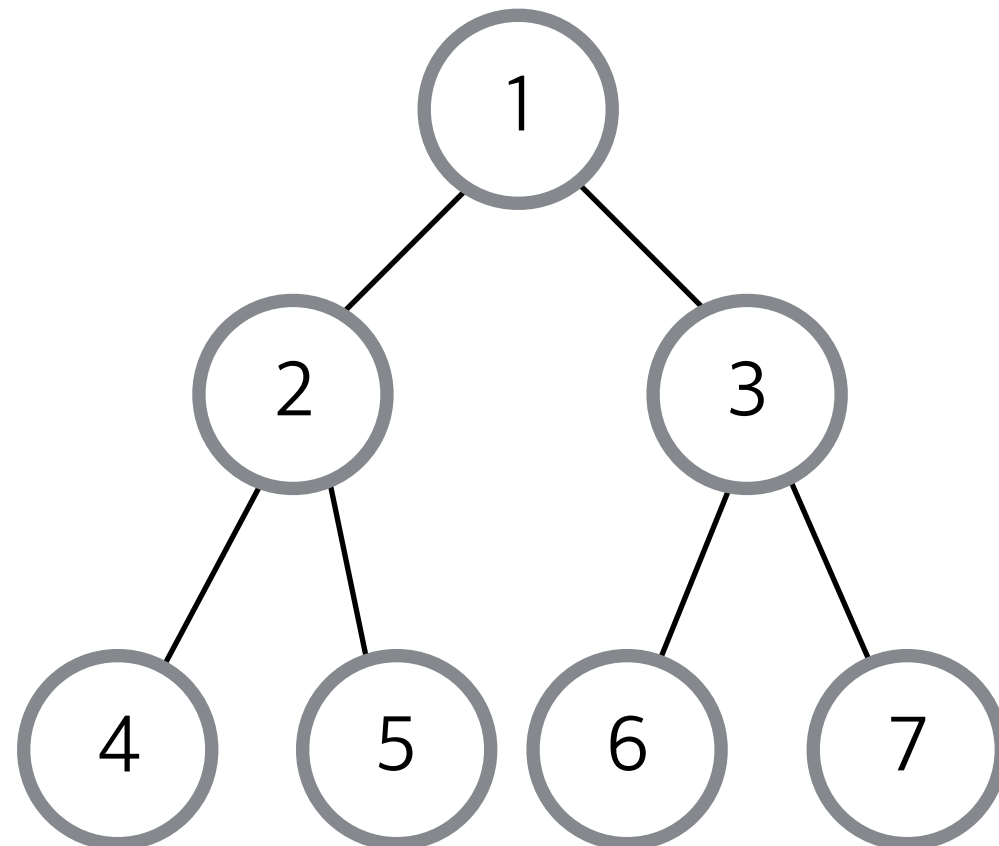
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함

전위순회 : Root - L - R 1 2 4 5 3 6 7

중위순회 : L - Root - R 4 2 5 1 6 3 7

후위순회 : L - R - Root 4 5 2 6 7 3 1



[예제 1] 이진트리 순회

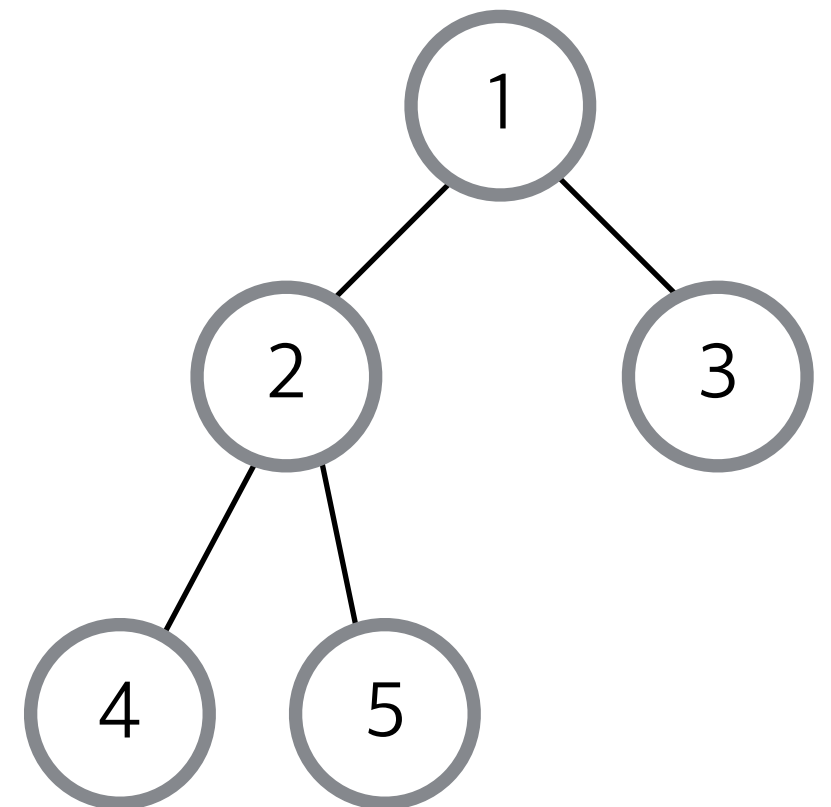
이진 트리가 주어질 때, 트리를 순회한 결과를 출력하라

입력의 예

```
5
1 2 3
2 4 5
3 -1 -1
4 -1 -1
5 -1 -1
```

출력의 예

```
1 2 4 5 3
4 2 5 1 3
4 5 2 3 1
```



[예제 1] 이진트리 순회

이진 트리가 주어질 때, 트리를 순회한 결과를 출력하라

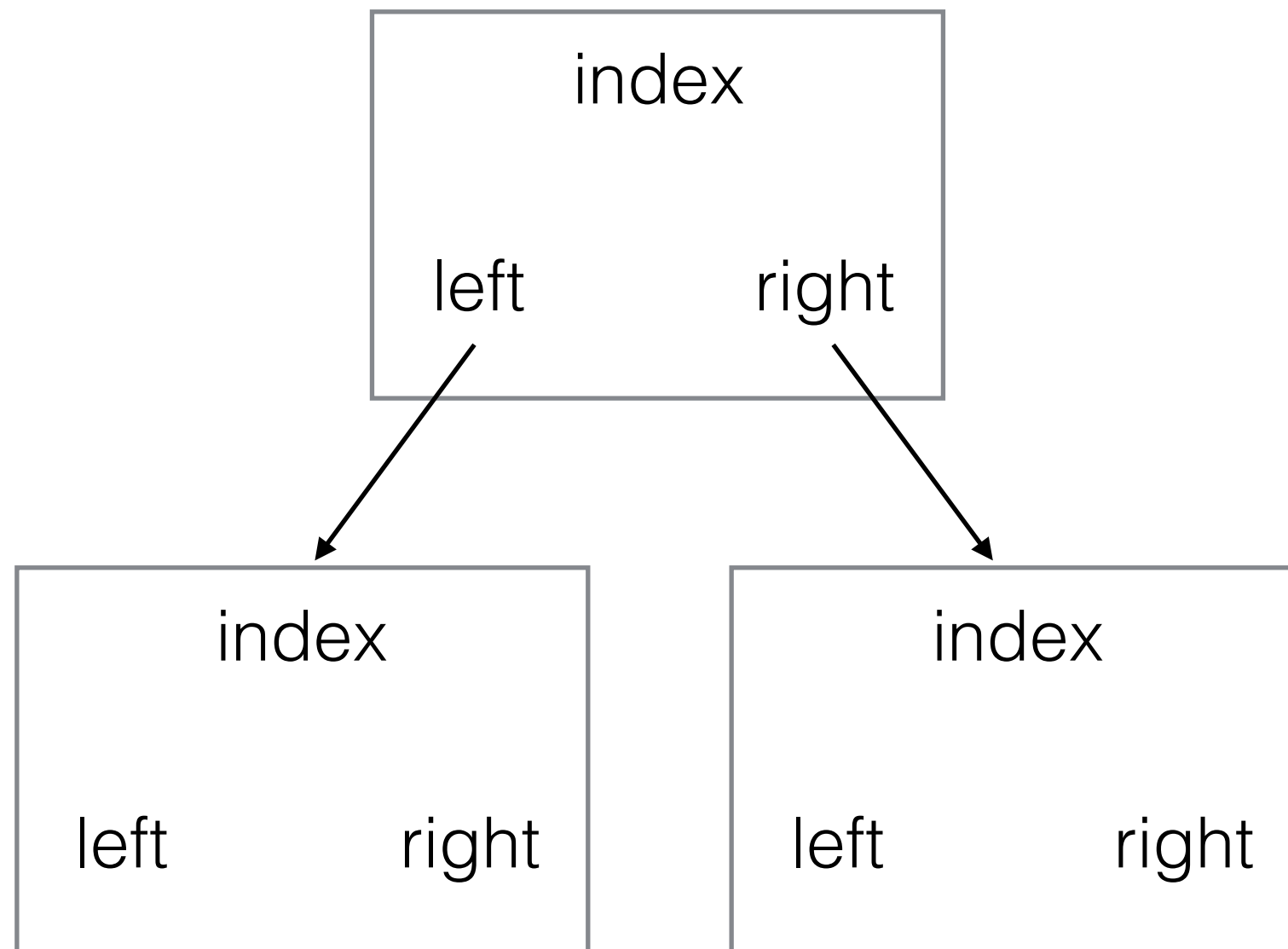
```
class Tree
```



[예제 1] 이진트리 순회

이진 트리가 주어질 때, 트리를 순회한 결과를 출력하라

class Tree



`/* elice */`

[예제 1] 이진트리 순회



```
/* elice */
```

[예제 2] 이진트리 만들기

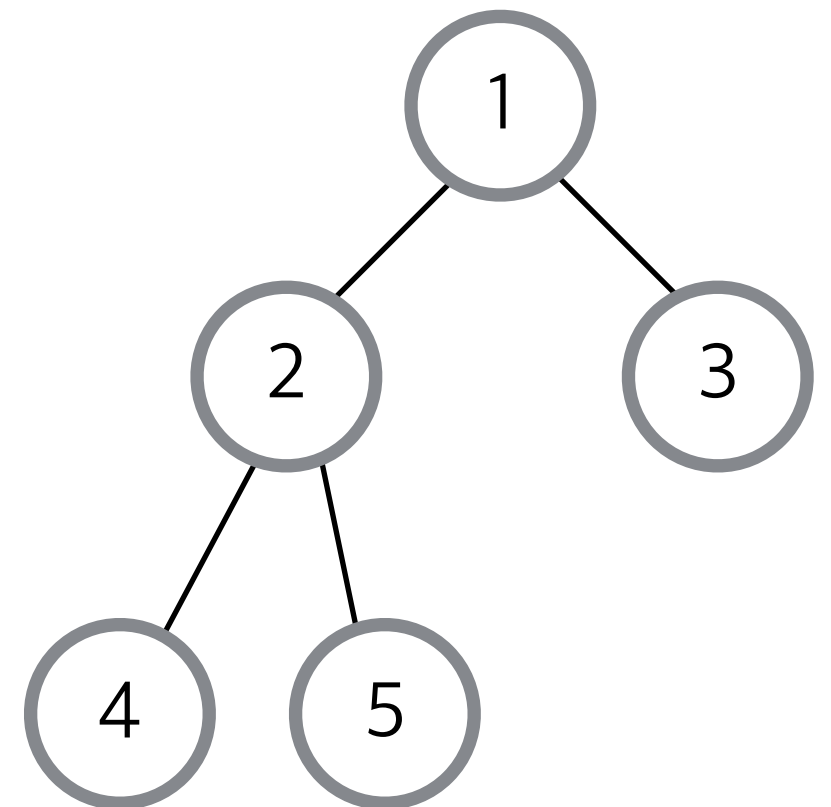
이진 트리의 입력이 주어질 때, 이진트리 만들기
만들어진 이진트리로 전위/중위/후위순회 결과를 출력

입력의 예

```
5
1 2 3
2 4 5
3 -1 -1
4 -1 -1
5 -1 -1
```

출력의 예

```
1 2 4 5 3
4 2 5 1 3
4 5 2 3 1
```

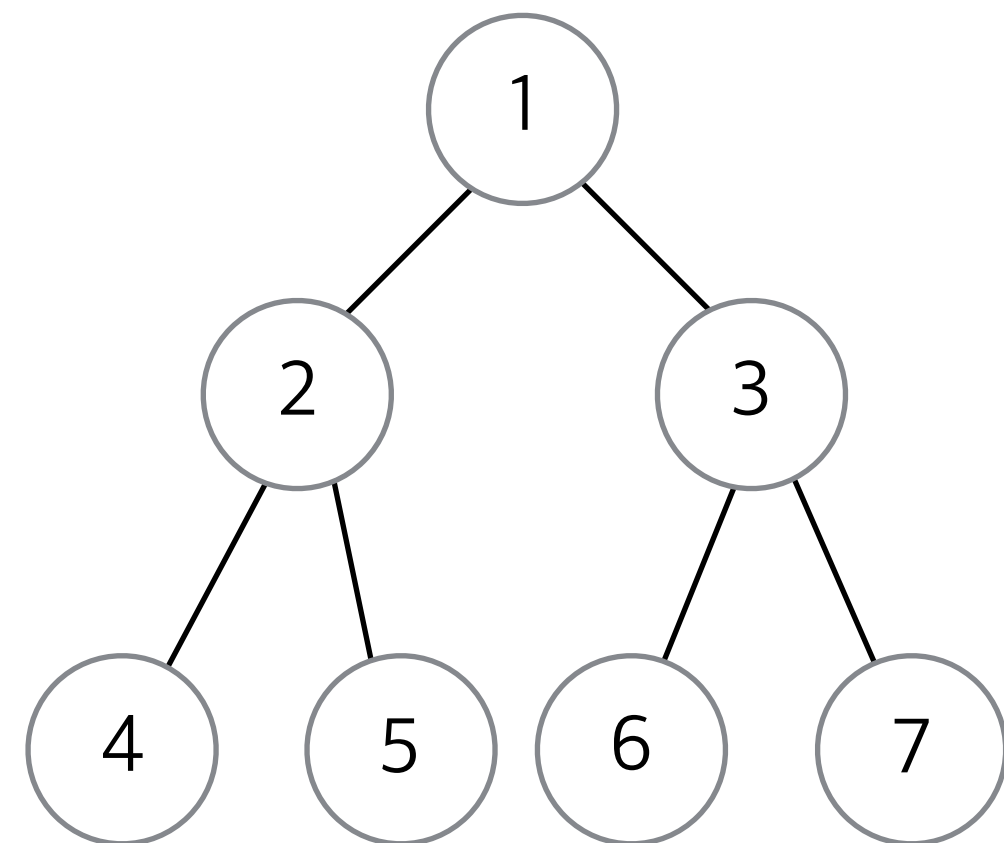


[예제 2] 이진트리 만들기



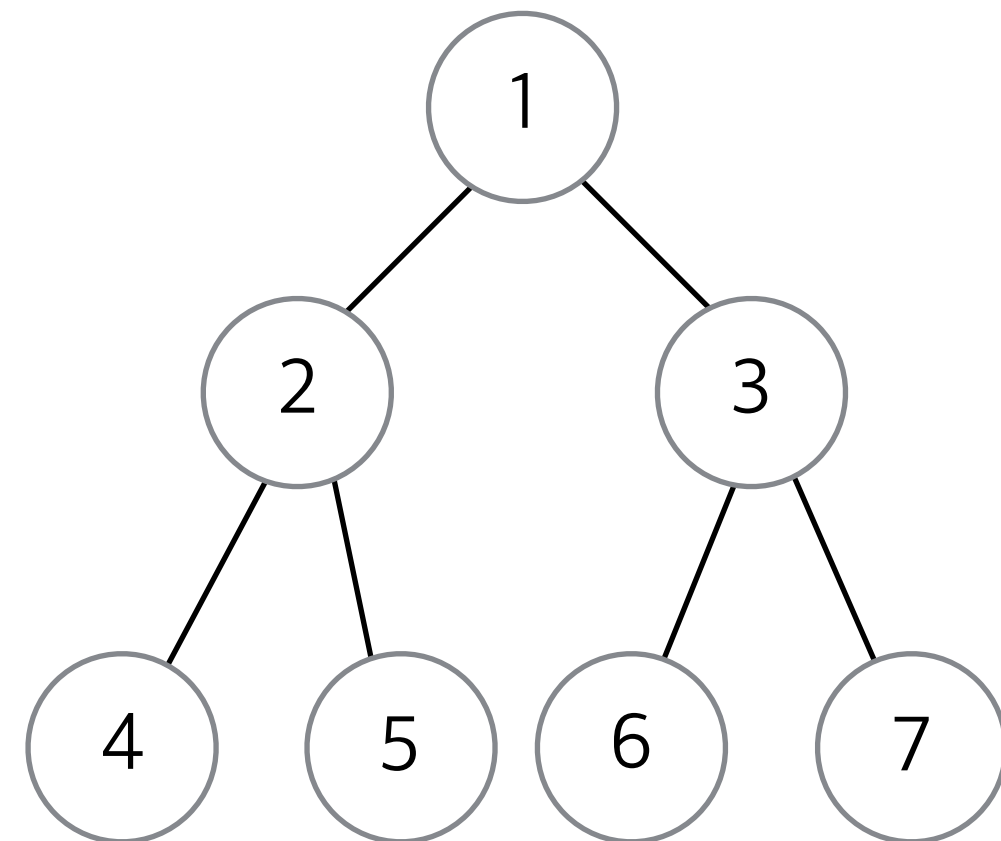
```
/* elice */
```

도대체 왜 하필 트리여야만 하는가?



도대체 왜 하필 트리여야만 하는가?

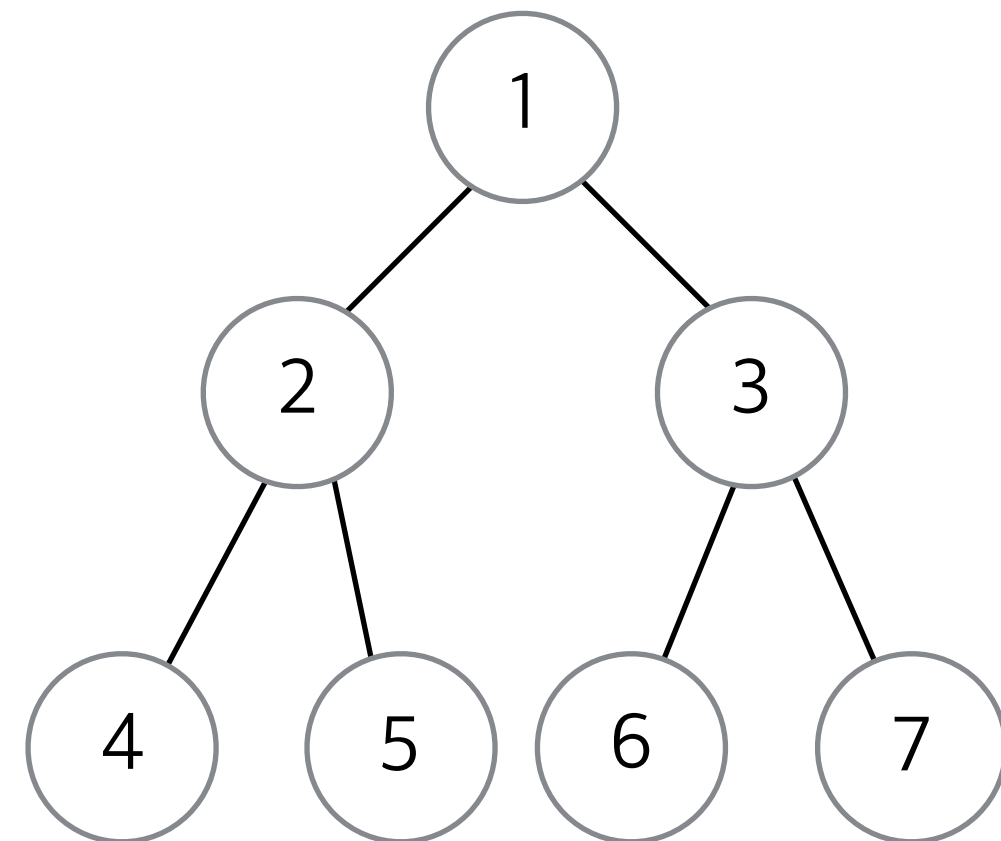
- 1) 정점에 무슨 자료를 담는가?
- 2) 간선은 어떤 의미인가?



도대체 왜 하필 트리여야만 하는가?

1) 정점에 무슨 자료를 담는가? 코드가 실행되는 상태

2) 간선은 어떤 의미인가? 코드 A가 코드 B를 부른다

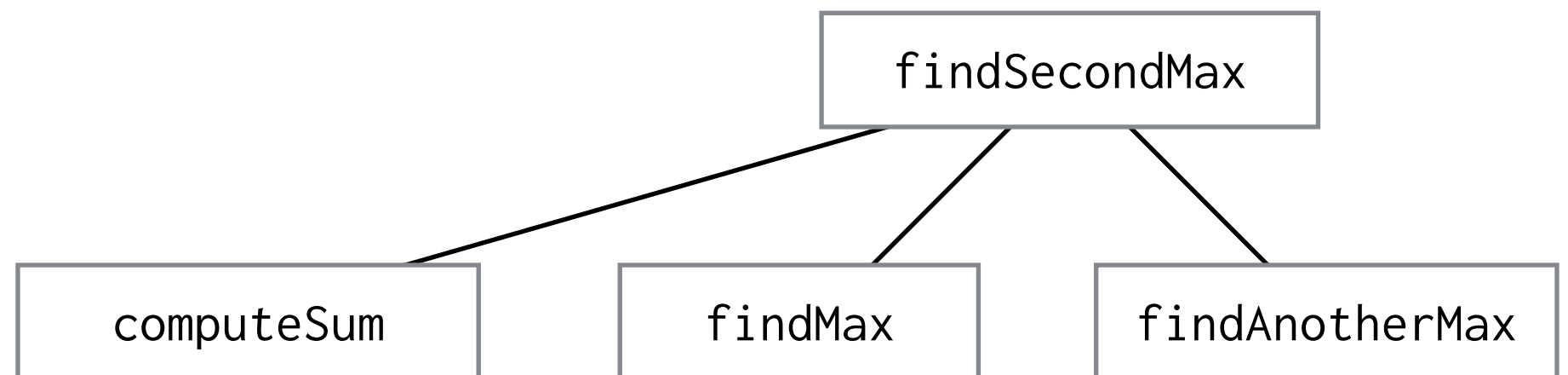


도대체 왜 하필 트리여야만 하는가 ?

```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)  
  
    maxValue = findMax(myMatrix)  
    secondMaxValue = findAnotherMax(myMatrix, maxValue)  
  
    return (sum, maxValue, secondMaxValue)
```

도대체 왜 하필 트리여야만 하는가 ?

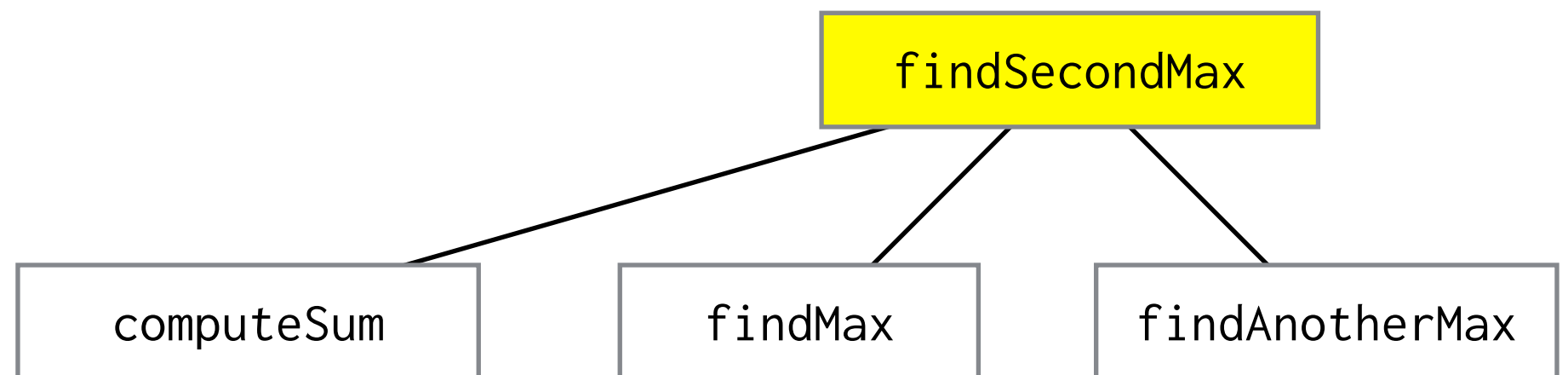
```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)  
  
    maxValue = findMax(myMatrix)  
    secondMaxValue = findAnotherMax(myMatrix, maxValue)  
  
    return (sum, maxValue, secondMaxValue)
```



도대체 왜 하필 트리여야만 하는가 ?

→

```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)  
  
    maxValue = findMax(myMatrix)  
    secondMaxValue = findAnotherMax(myMatrix, maxValue)  
  
    return (sum, maxValue, secondMaxValue)
```

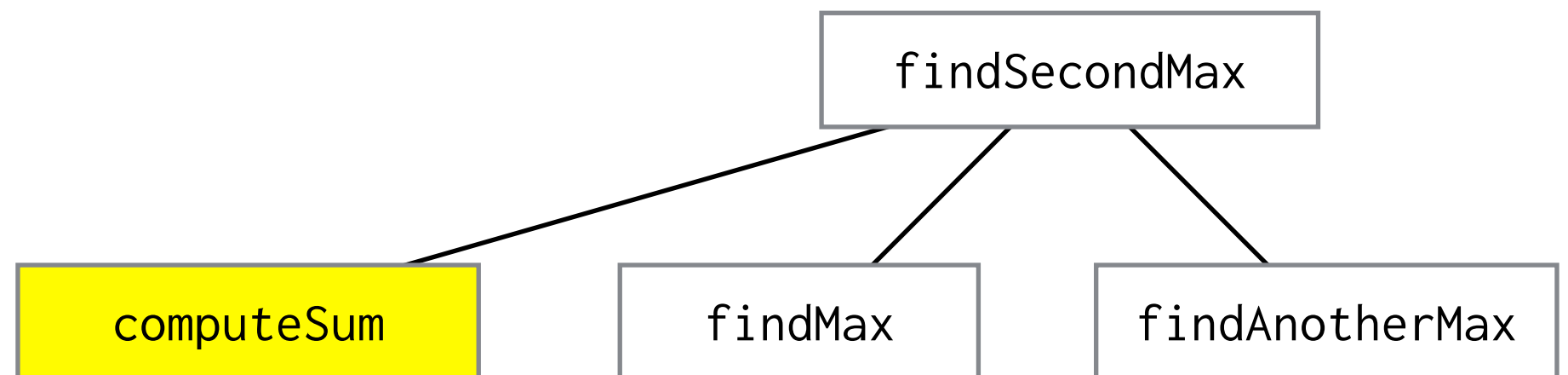


도대체 왜 하필 트리여야만 하는가 ?

```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)  
  
    maxValue = findMax(myMatrix)  
    secondMaxValue = findAnotherMax(myMatrix, maxValue)  
  
    return (sum, maxValue, secondMaxValue)
```

→

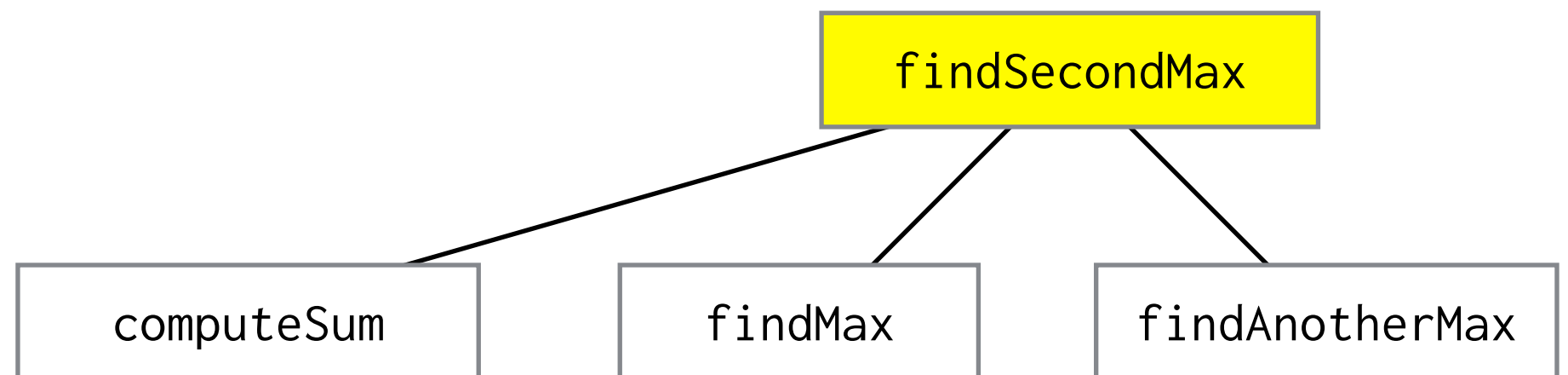
```
def computeSum(myMatrix) :  
    return sum(myMatrix)
```



도대체 왜 하필 트리여야만 하는가 ?

→

```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)  
  
    maxValue = findMax(myMatrix)  
    secondMaxValue = findAnotherMax(myMatrix, maxValue)  
  
    return (sum, maxValue, secondMaxValue)
```



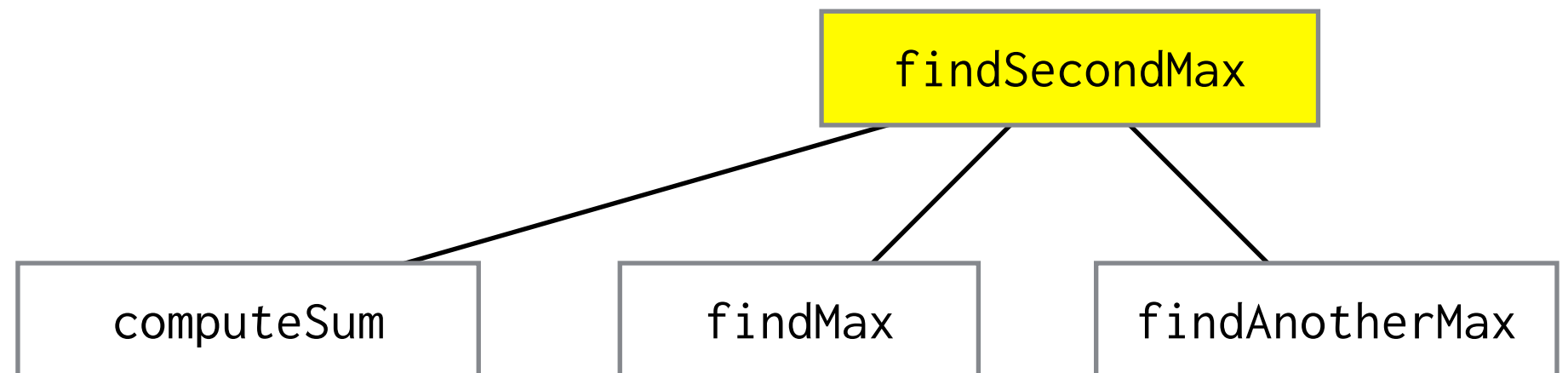
도대체 왜 하필 트리여야만 하는가 ?

```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)
```



```
    maxValue = findMax(myMatrix)  
    secondMaxValue = findAnotherMax(myMatrix, maxValue)
```

```
    return (sum, maxValue, secondMaxValue)
```

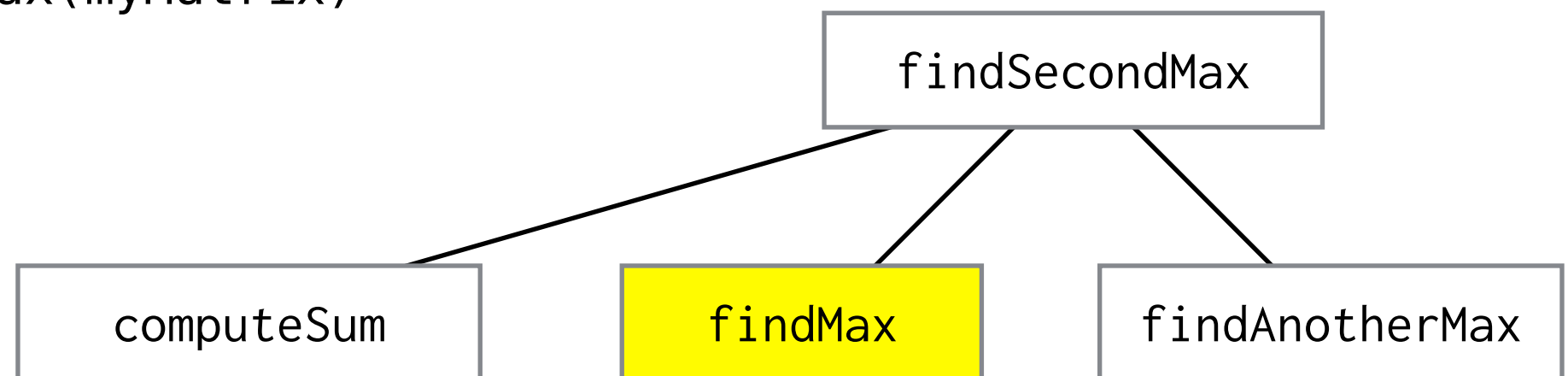


도대체 왜 하필 트리여야만 하는가?

```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)  
  
    maxValue = findMax(myMatrix)  
    secondMaxValue = findAnotherMax(myMatrix, maxValue)  
  
    return (sum, maxValue, secondMaxValue)
```

→

```
def findMax(myMatrix) :  
    return max(myMatrix)
```



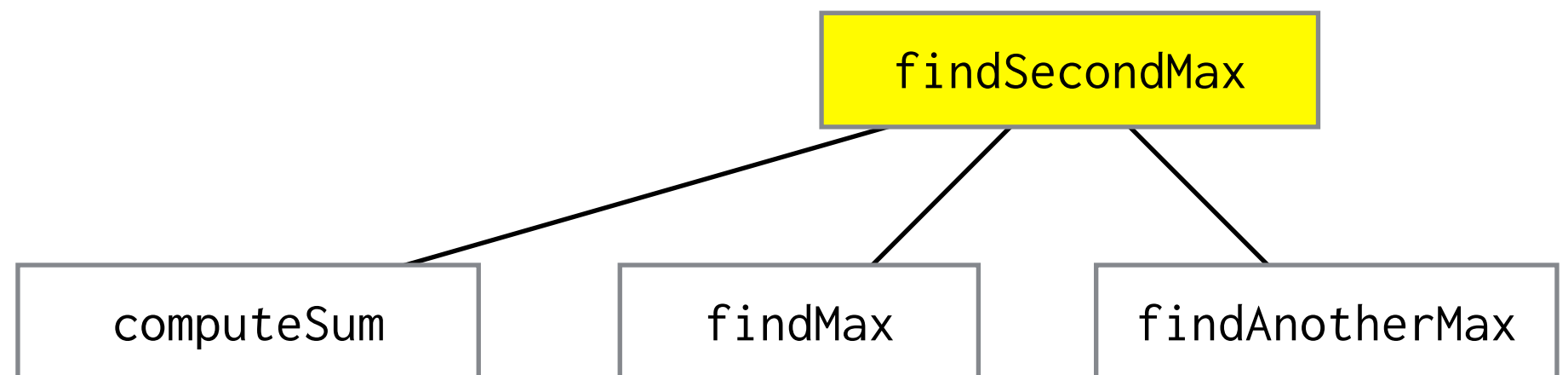
도대체 왜 하필 트리여야만 하는가 ?

```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)
```



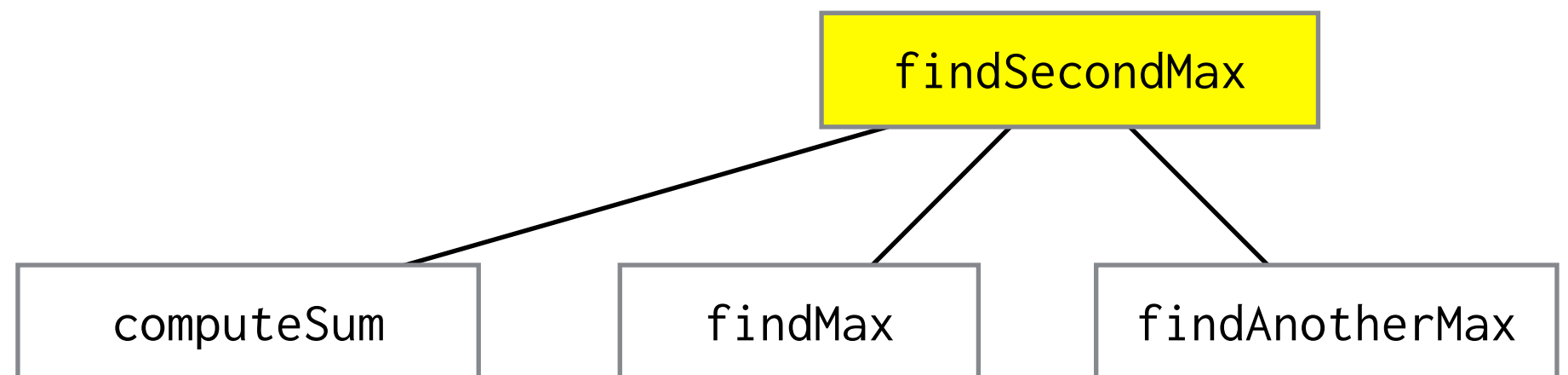
```
    maxValue = findMax(myMatrix)  
    secondMaxValue = findAnotherMax(myMatrix, maxValue)
```

```
    return (sum, maxValue, secondMaxValue)
```



도대체 왜 하필 트리여야만 하는가 ?

```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)  
  
    maxVal = findMax(myMatrix)  
    secondMaxVal = findAnotherMax(myMatrix, maxVal)  
  
    return (sum, maxVal, secondMaxVal)
```



도대체 왜 하필 트리여야만 하는가 ?

```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)  
  
    maxValue = findMax(myMatrix)  
    secondMaxValue = findAnotherMax(myMatrix, maxValue)  
  
    return (sum, maxValue, secondMaxValue)
```

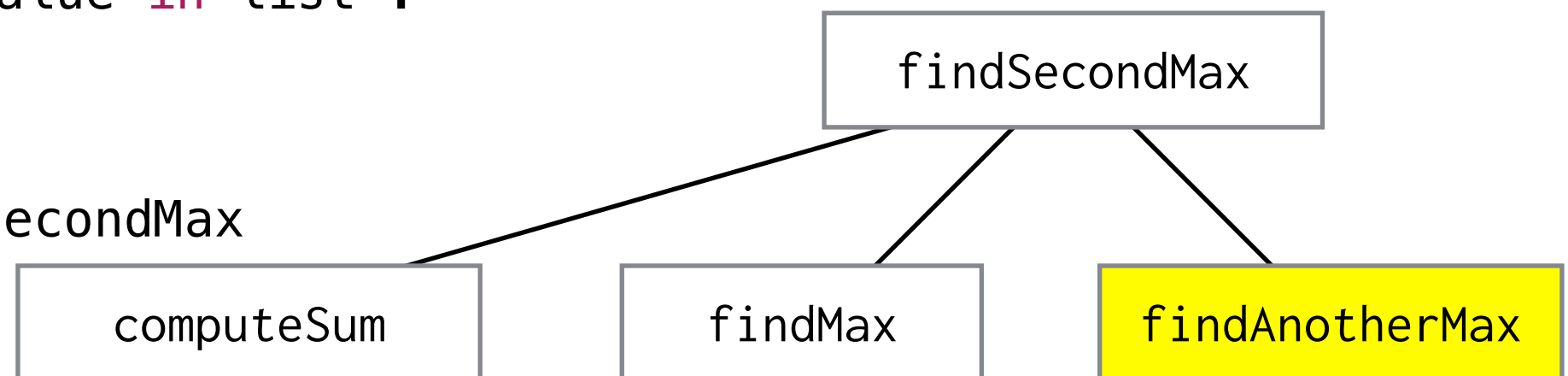
→

```
def findAnotherMax (myMatrix, value) :  
    secondMax = -1
```

```
    for list in myMatrix :  
        for value in list :
```

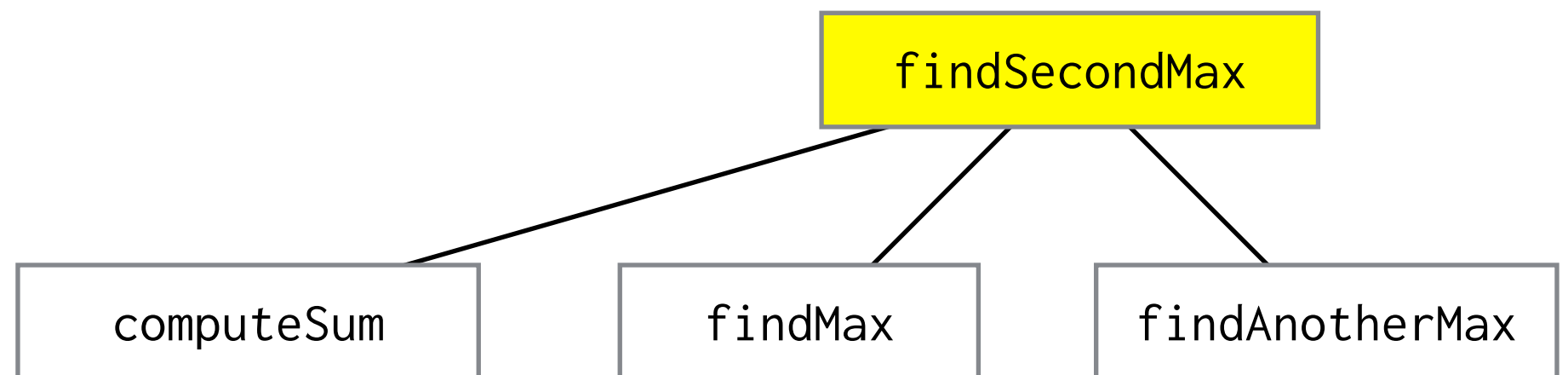
...

```
    return secondMax
```



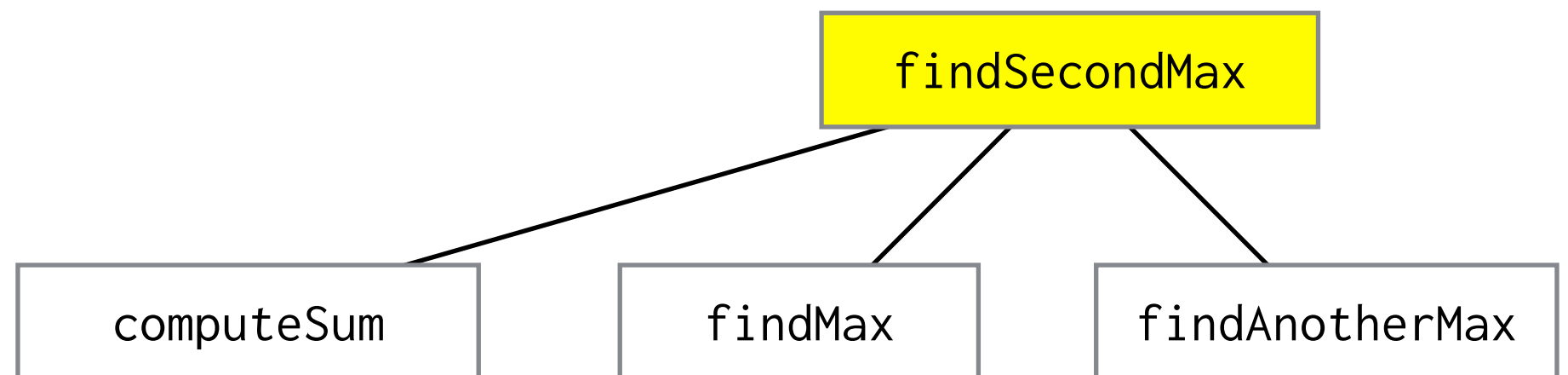
도대체 왜 하필 트리여야만 하는가 ?

```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)  
  
    →    maxValue = findMax(myMatrix)  
        secondMaxValue = findAnotherMax(myMatrix, maxValue)  
  
    return (sum, maxValue, secondMaxValue)
```



도대체 왜 하필 트리여야만 하는가 ?

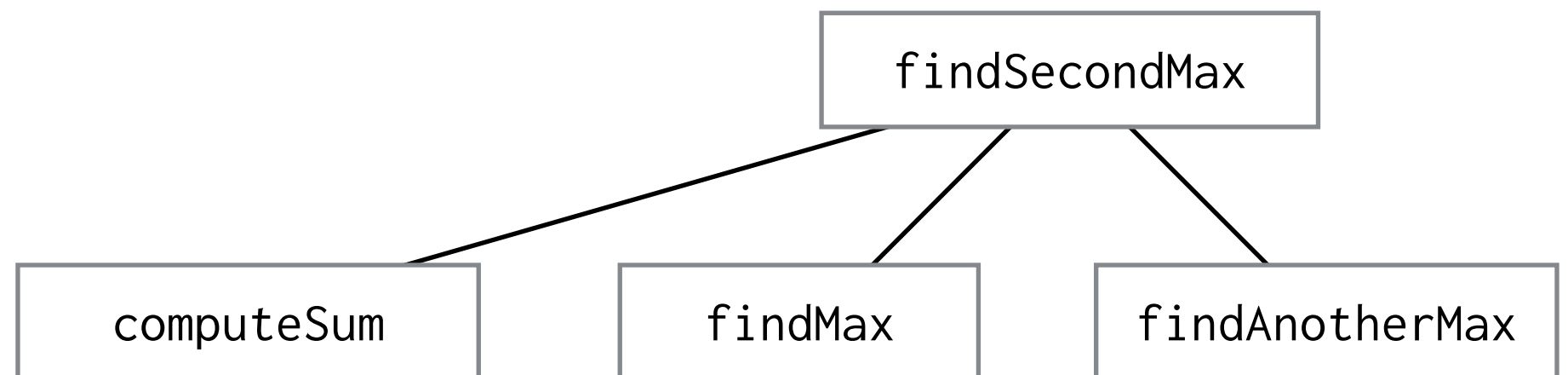
```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)  
  
    maxValue = findMax(myMatrix)  
    secondMaxValue = findAnotherMax(myMatrix, maxValue)  
  
→    return (sum, maxValue, secondMaxValue)
```



도대체 왜 하필 트리여야만 하는가 ?

```
def findSecondMax(myMatrix) :  
    sum = computeSum(myMatrix)  
  
    maxValue = findMax(myMatrix)  
    secondMaxValue = findAnotherMax(myMatrix, maxValue)  
  
→ return (sum, maxValue, secondMaxValue)
```

코드 실행 = 트리를 후위순회



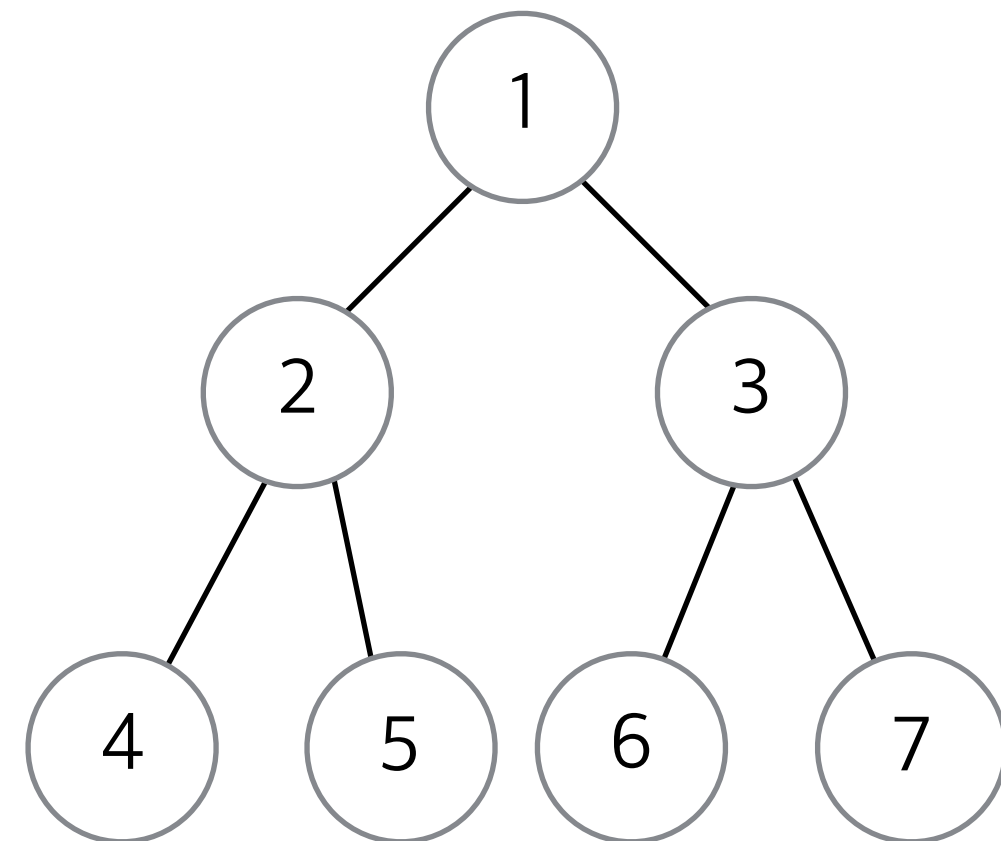
도대체 왜 하필 트리여야만 하는가?

1) 정점에 무슨 자료를 담는가?

코드가 실행되는 상태

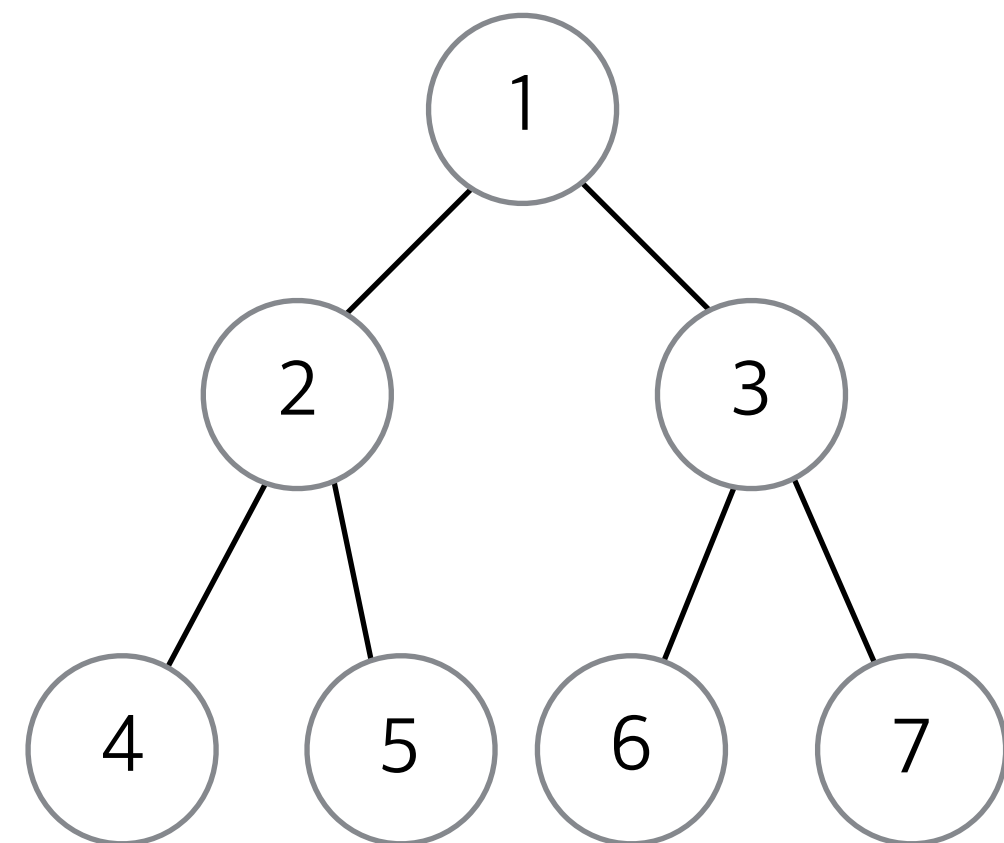
2) 간선은 어떤 의미인가?

코드 A가 코드 B를 부른다



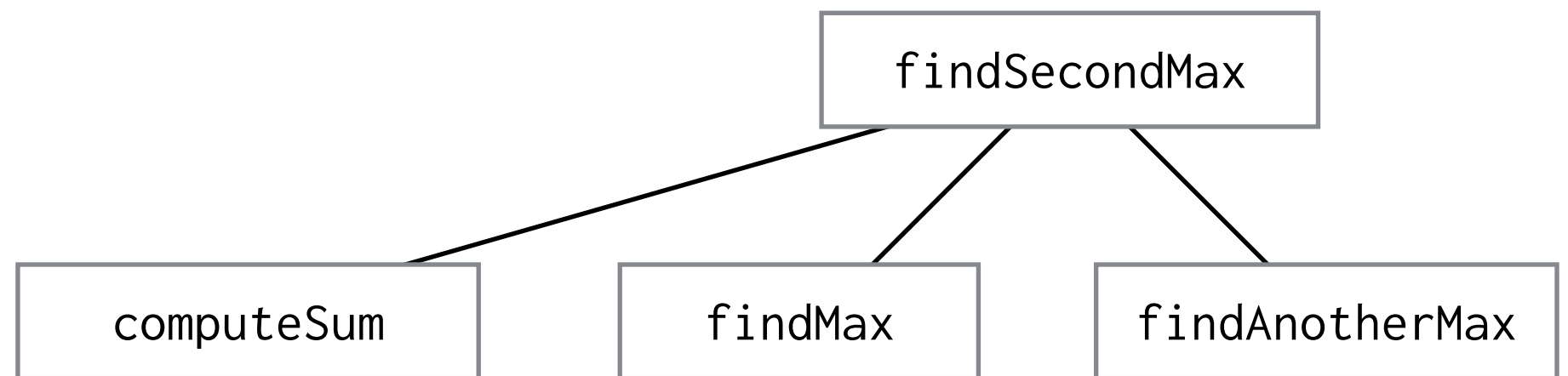
도대체 왜 하필 트리여야만 하는가 ?

코드가 실행되고 있는 상태를 나타내는 자료구조



좋은 코드가 가져야 할 필수 조건

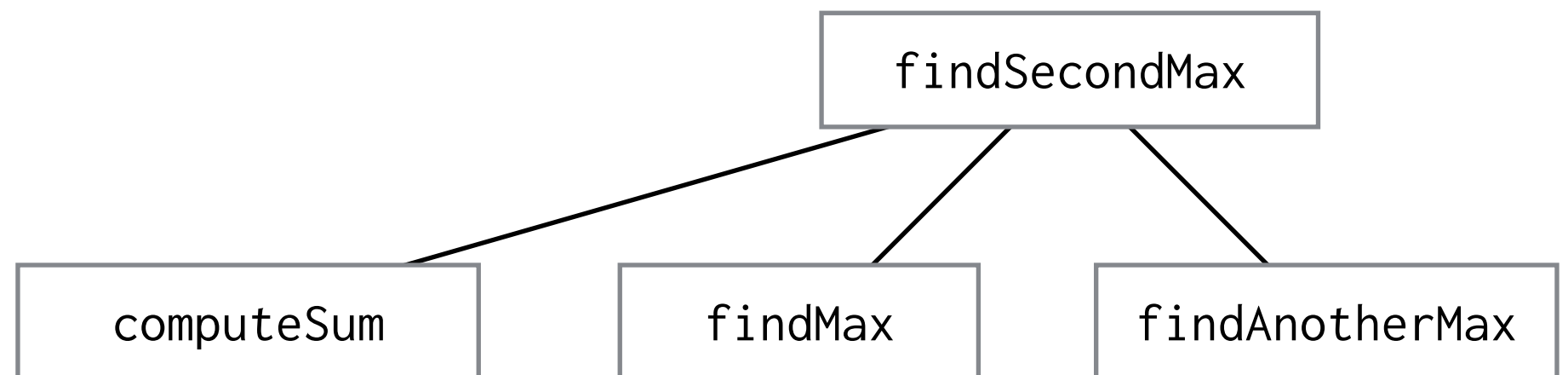
좋은 코드 = 트리만 봐도 그 기능을 알 수 있는 코드



좋은 코드가 가져야 할 필수 조건

좋은 코드 = 트리만 봐도 그 기능을 알 수 있는 코드

= 함수를 중심으로 작성된 코드

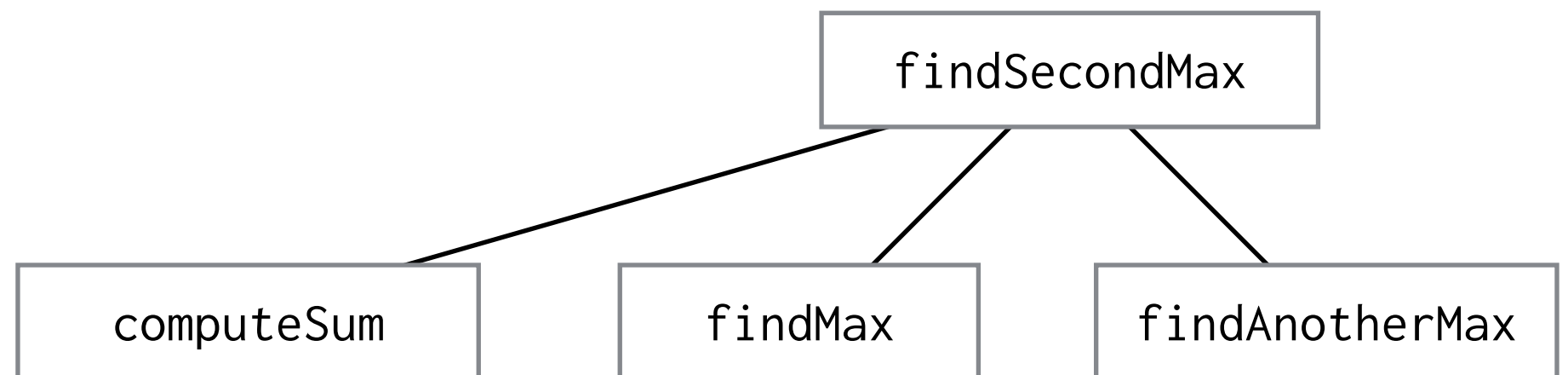


좋은 코드가 가져야 할 필수 조건

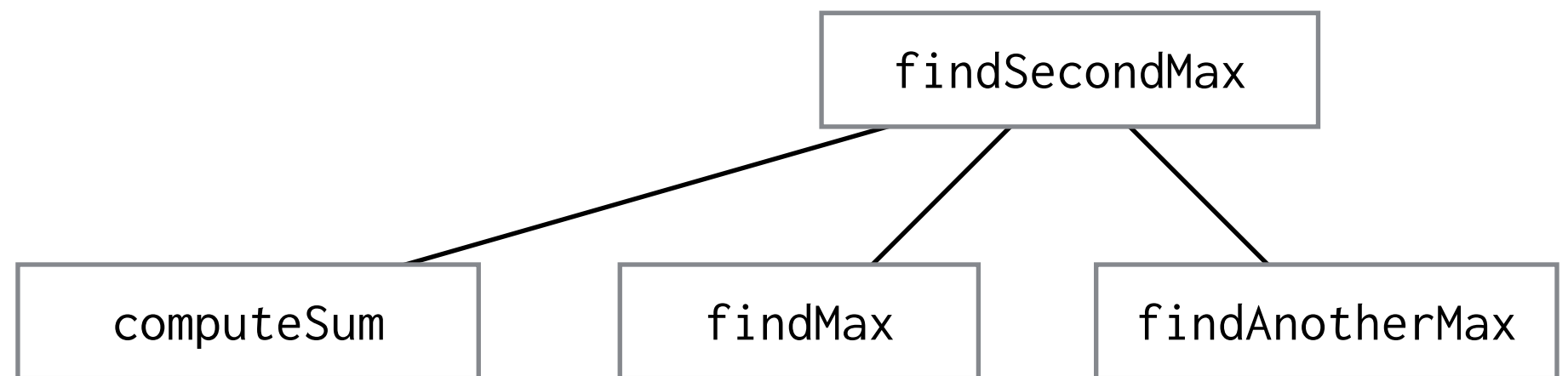
좋은 코드 = 트리만 봐도 그 기능을 알 수 있는 코드

= 함수를 중심으로 작성된 코드

= 의미 단위로 작성된 코드



의미 단위로 작성된 코드



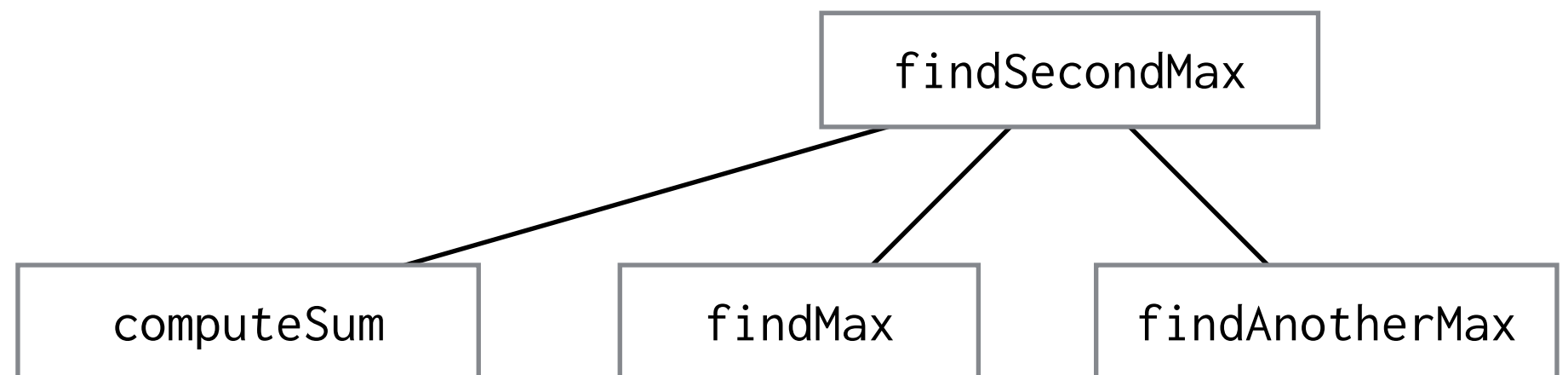
의미 단위로 작성된 코드

`findSecondMax(m)` : 주어진 행렬 `m`에서 (합, 최대값, 두 번째 최대값)을 반환하는 함수

`computeSum(m)` : 주어진 행렬 `m`에서 원소의 합을 반환하는 함수

`findMax(m)` : 주어진 행렬 `m`에서 최대값을 반환하는 함수

`findAnotherMax(m, v)` : 주어진 행렬 `m`에서 `v`를 제외한 최대값을 반환하는 함수



의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$

의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$

Factorial(n) : n! 을 반환하는 함수

의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$

Factorial(n) : n! 을 반환하는 함수

```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```

의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$

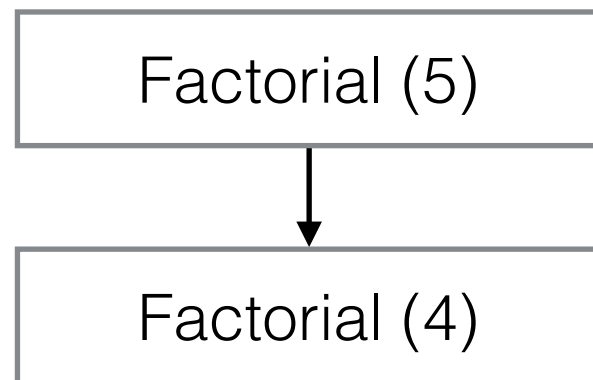
Factorial (5)

Factorial(n) : n! 을 반환하는 함수

```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```

의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$

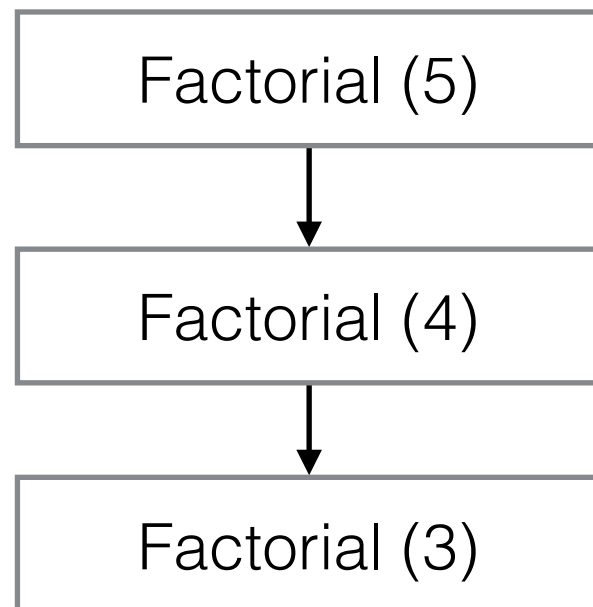


Factorial(n) : n! 을 반환하는 함수

```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```

의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$

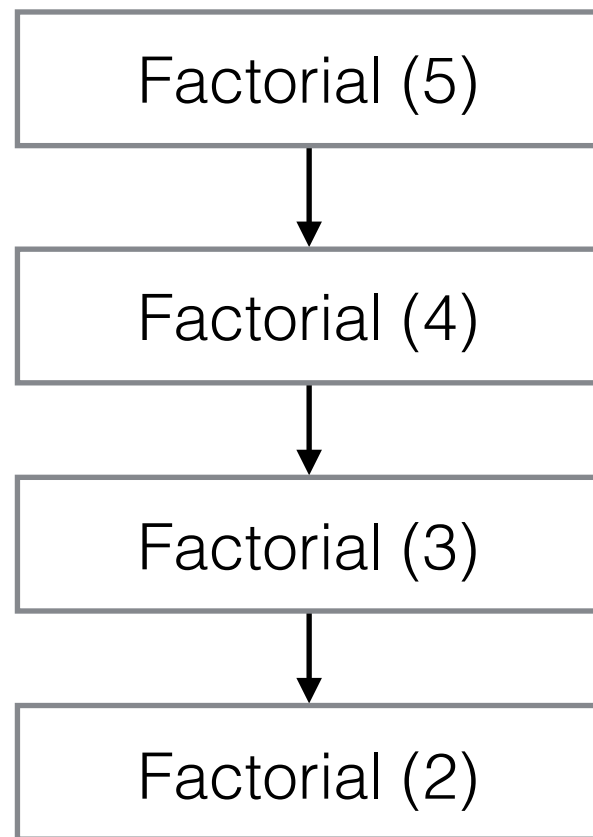


Factorial(n) : n! 을 반환하는 함수

```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```


의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$

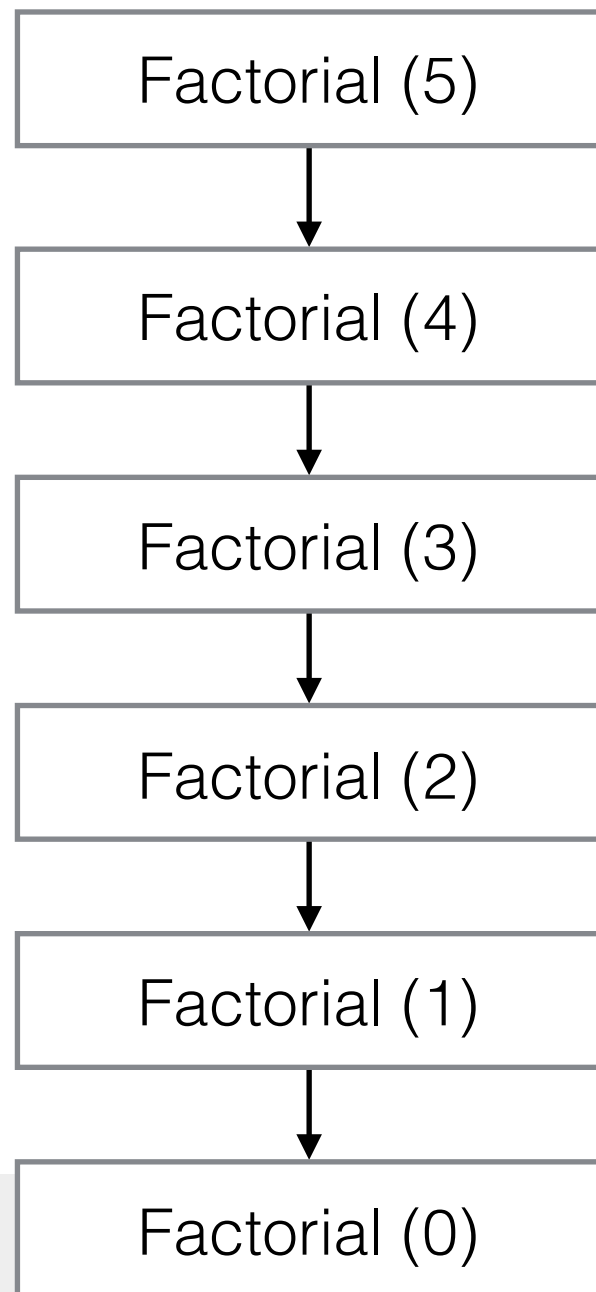


Factorial(n) : n! 을 반환하는 함수

```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```

의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$

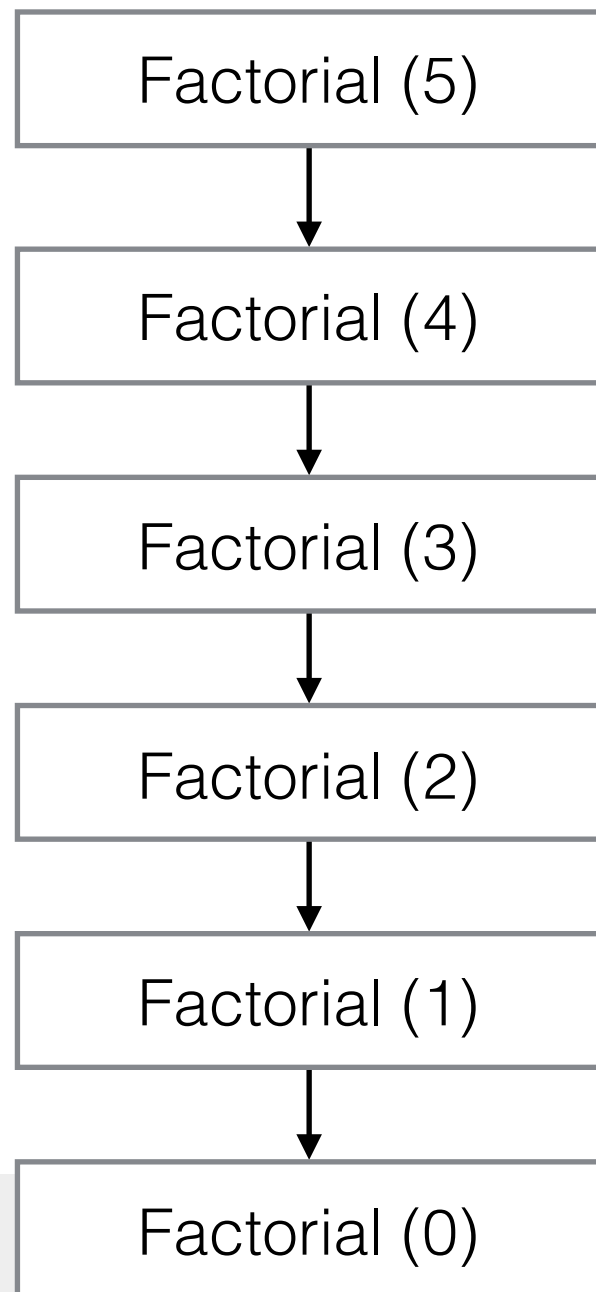


Factorial(n) : n! 을 반환하는 함수

```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```

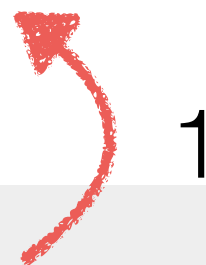
의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$



Factorial(n) : n! 을 반환하는 함수

```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```

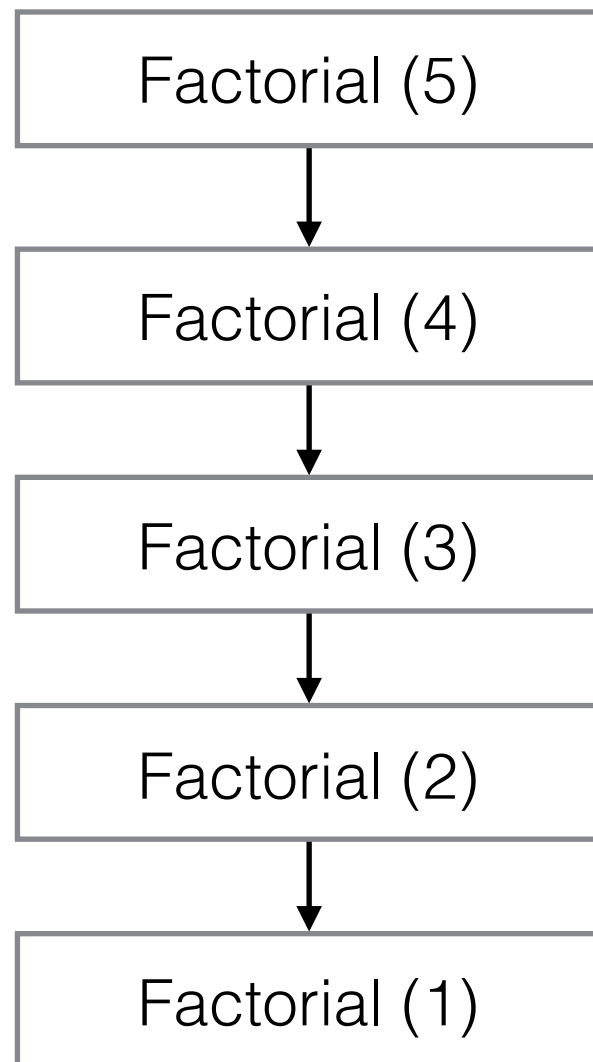


1

`/* elice */`

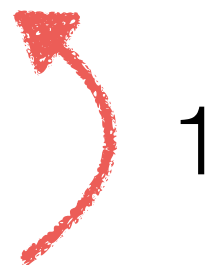
의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$



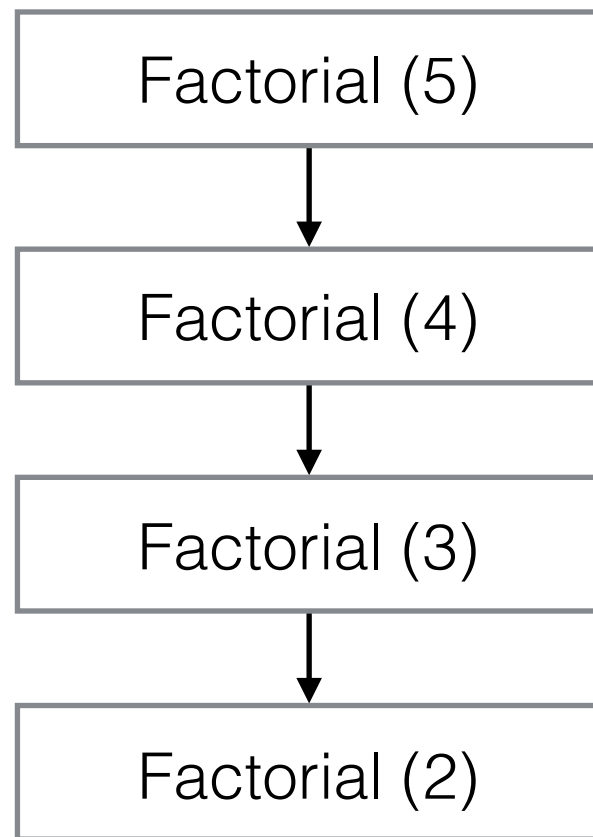
Factorial(n) : n! 을 반환하는 함수

```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```



의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$



Factorial(n) : n! 을 반환하는 함수

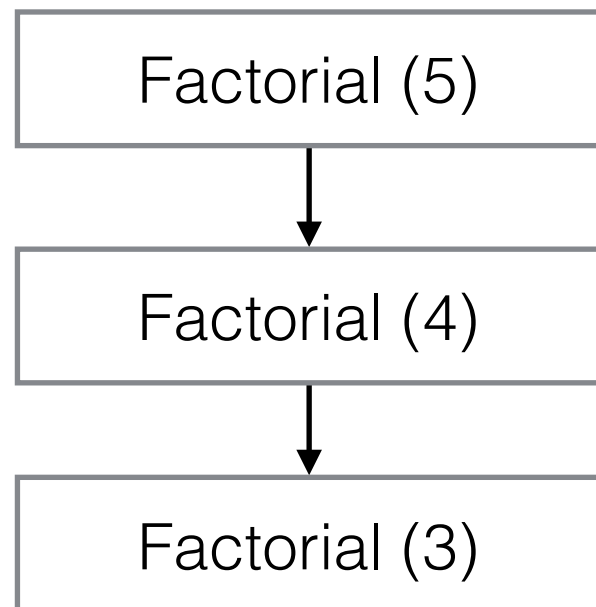
```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```




2

의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$



Factorial(n) : n! 을 반환하는 함수

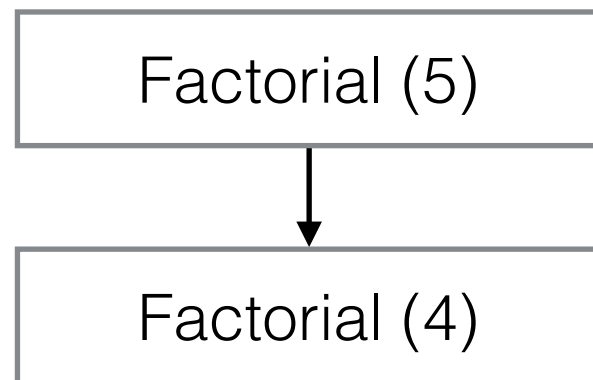


```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```

A red arrow points from the code block to the 'Factorial (4)' box in the flowchart.

의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$



Factorial(n) : n! 을 반환하는 함수

24

```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```

의미 단위로 작성된 코드

120

Factorial (5)

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$

Factorial(n) : n! 을 반환하는 함수

```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```


의미 단위로 작성된 코드

$$n! = 1 \times 2 \times \dots \times (n-1) \times n$$

120

Factorial (5)

Factorial(n) : n! 을 반환하는 함수

```
def Factorial(n) :  
    if n == 0 :  
        return 1  
    else :  
        return n * Factorial(n-1)
```

O(n)

의미 단위로 작성된 코드

$$m^n = m \times m \times \dots \times m$$

의미 단위로 작성된 코드

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

의미 단위로 작성된 코드

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

```
def getPower(m, n) :  
    if n == 0 :  
        return 1  
    else :  
        return m * getPower(m, n-1)
```

의미 단위로 작성된 코드

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

```
def getPower(m, n) :  
    if n == 0 :  
        return 1  
    else :  
        return m * getPower(m, n-1)
```

O(n)

의미 단위로 작성된 코드

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

의미 단위로 작성된 코드

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

$$m^n = (m^{(n/2)})^2 \quad n \text{이 짝수일 경우}$$

의미 단위로 작성된 코드

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

$$m^n = (m^{(n/2)})^2 \quad n \text{이 짝수일 경우}$$

$$(m^{n-1}) \times m \quad n \text{이 홀수일 경우}$$

의미 단위로 작성된 코드

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

```
def getPower(m, n) :  
    if n == 0 :  
        return 1  
    elif n % 2 == 0 :  
        temp = getPower(m, n//2)  
        return temp * temp  
    else :  
        return getPower(m, n-1) * m
```

의미 단위로 작성된 코드

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

```
def getPower(m, n) :  
    if n == 0 :  
        return 1  
    elif n % 2 == 0 :  
        temp = getPower(m, n//2)  
        return temp * temp  
    else :  
        return getPower(m, n-1) * m
```

$O(\log n)$

[예제 3] 거듭제곱 구하기



```
/* elice */
```

요약

의미단위로 작성된 코드가 좋은 코드이다

→ 코드를 이해한다 = 각 함수가 무슨 일을 하는지 설명할 수 있다

트리는 코드가 실행되고 있는 상태를 나타내는 자료구조이다

→ 물론, 코드를 의미 단위로 나타냈을 때 파악이 가능한 사실이다

코드를 하나하나 따라가는 것은 컴퓨터가 해야 할 일이다

→ 우리는 앞으로 코드가 하는 일, 더 나아가 코드의 의미에 집중한다

퀴즈 및 설문조사



`/* elice */`

감사합니다!

신현규

E-mail : hyungyu.sh@kaist.ac.kr

Kakao : yougatup

/* elice */

문의 및 연락처

academy.elice.io

contact@elice.io

facebook.com/elice.io

blog.naver.com/elicer