SWIFT
Data Structure - Binary Search Tree

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Binary Search Tree

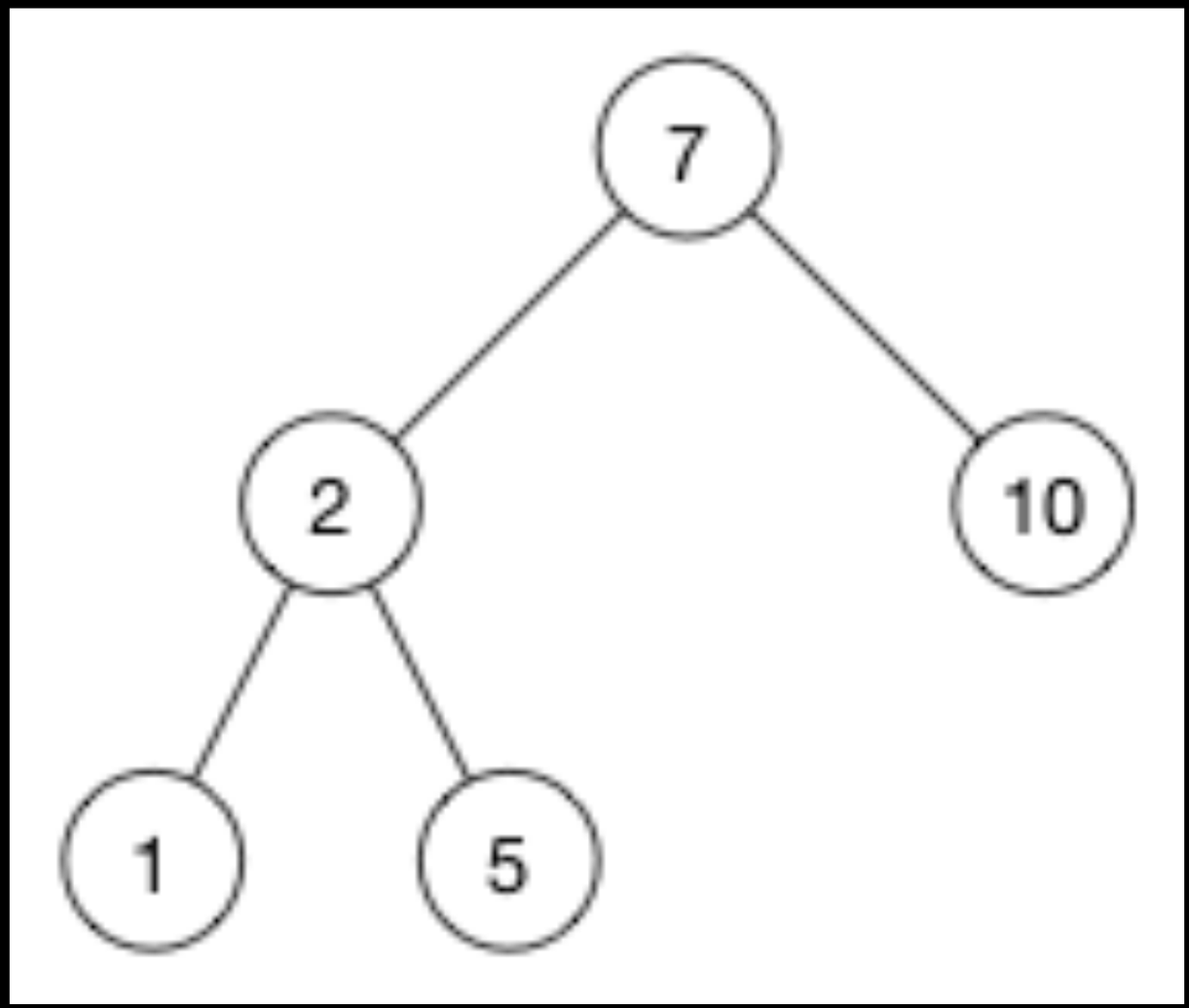
이진 탐색 트리(Binary Search Tree)는 이진 트리의 한 종류로서 삽입과 삭제에서 특정한 규칙을 두어 검색을 빠르게 할 수 있도록 도와주는 자료구조입니다.

이진 탐색 트리의 기본 구조는 이진 탐색(Binary Search)와 연결 리스트(Linked List)를 결합한 구조를 가지고 있습니다.

이진 탐색의 효율적인 탐색 능력(O(logn))을 유지하면서 빈번한 자료 입력과 삭제(O(1))를 가능하게끔 고안한 형태입니다.

Binary Search Tree

아래는 이진 탐색 트리(Binary Search Tree)의 한 예시입니다.



Binary Search Tree

이진 탐색 트리(Binary Search Tree)는 아래와 같은 규칙을 가진 트리입니다.

- 형태

1. 각 노드마다 값이 있다.
2. 루트 노드를 기준으로 왼쪽 자식 노드는 루트보다 값이 작다.(<)
3. 루트 노드를 기준으로 오른쪽 자식 노드는 루트보다 값이 크다.(>)
4. 왼쪽과 오른쪽 서브 트리도 이진 탐색 트리이다.

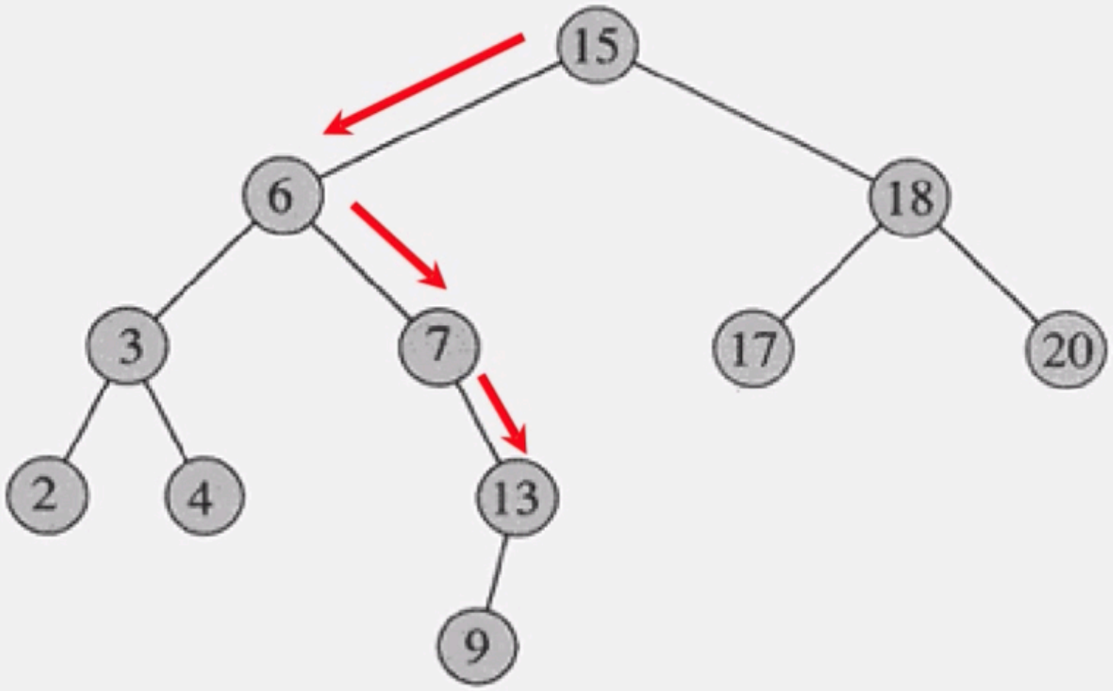
Search

- 탐색

1. 이진 탐색 트리가 공백 상태이면 탐색 실패
2. 탐색하는 키가 현재 트리의 루트 키와 같으면 루트 반환
3. 탐색 키 값과 루트 키값을 비교하여 작으면 왼쪽, 크면 오른쪽 서브트리를 반복적으로 탐색

Search

탐색에 대한 예시를 살펴보면 아래와 같습니다.



키 13을 검색: 15 → 6 → 7 → 13

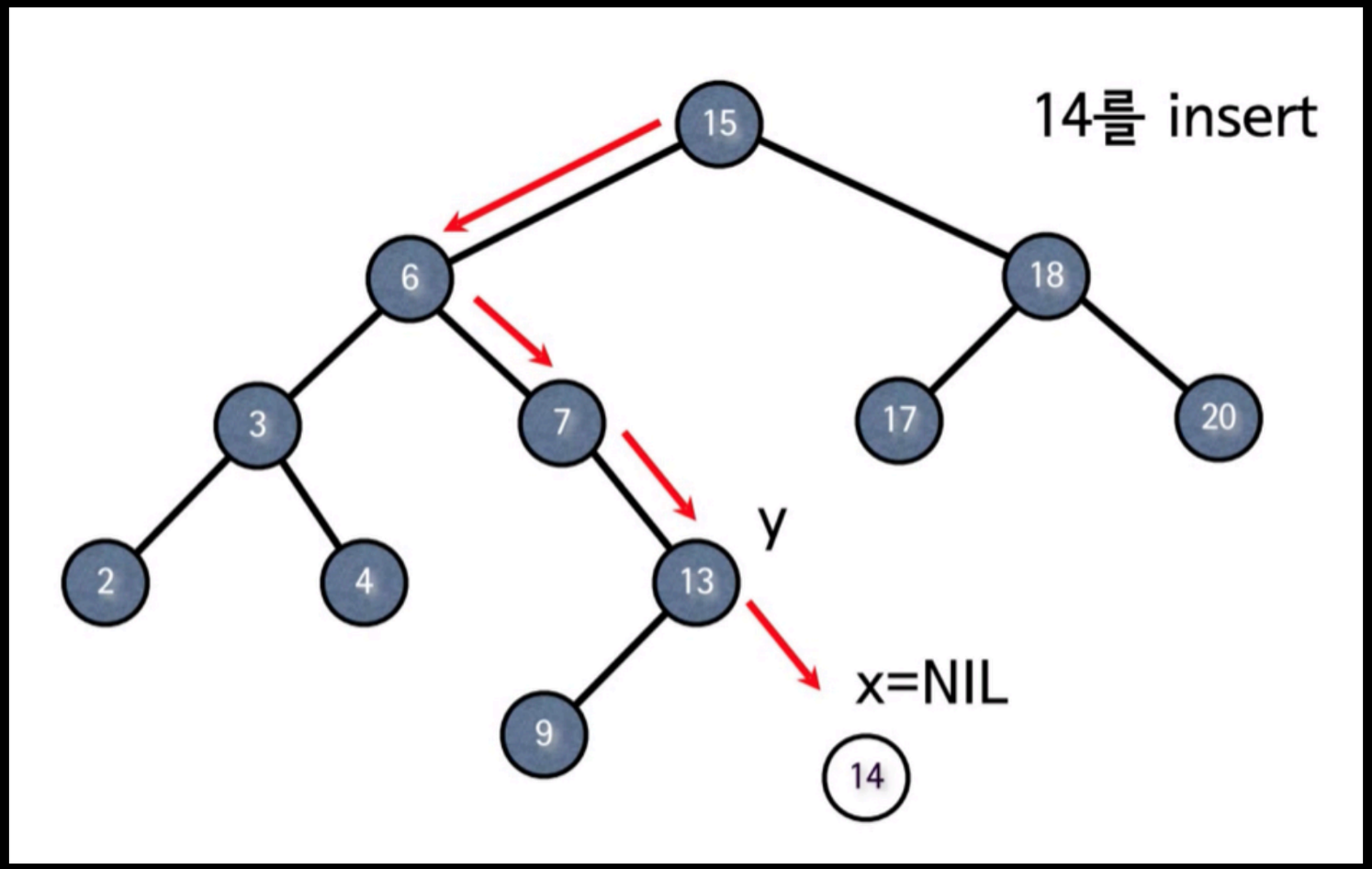
Insert

- 삽입(Insert)

1. 삽입 전에 키 값 검색을 수행(키 값에 대해 중복을 허용 안할 경우)
2. 키가 있을 경우 중복 허용을 안한다면 삽입 불가
3. 키 값 중복을 허용할 경우 1, 2번 과정 생략 가능
4. 루트 기준으로 삽입하려는 키 값이 작으면 왼쪽, 크면 오르쪽 서브 트리를 반복적으로 탐색
5. 최종 비어있는 하단부에 최종 삽입

Insert

삽입(Insert)에 대한 예시를 살펴보면 아래와 같습니다.



Delete

- 삭제(Delete)

삭제의 경우는 아래의 같은 경우에 따라서 방법이 다릅니다.

1. 자식 노드가 없는 노드(리프 노드) 삭제 :
바로 해당 노드 삭제

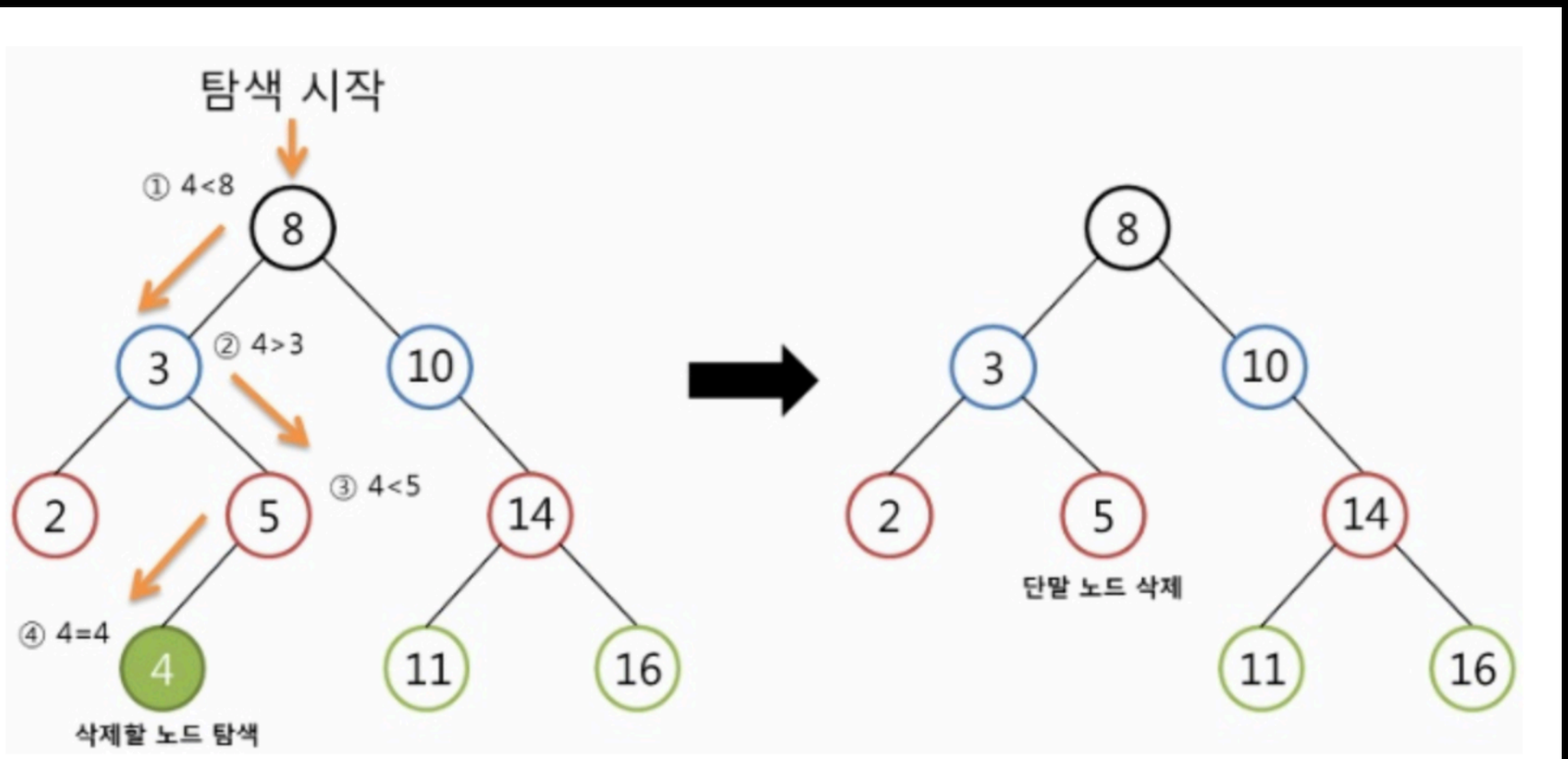
2. 자식 노드가 1개인 노드 삭제 :
해당 노드를 삭제하고 그 위치에 해당 노드의 자식노드를 대입

3. 자식 노드가 2개인 노드 삭제 :
삭제하고자 하는 노드의 값을 해당 노드의 왼쪽 서브트리에서 가장 큰 값으로 대체하거나, 오른쪽 서브트리에서 가장 작은 값으로 대체한 뒤, 해당 노드(왼쪽 서브트리에서 가장 큰 값을 가지는 노드 또는 오른쪽 서브트리에서 가장 작은 값을 가지는 노드)를 삭제한다.

Delete

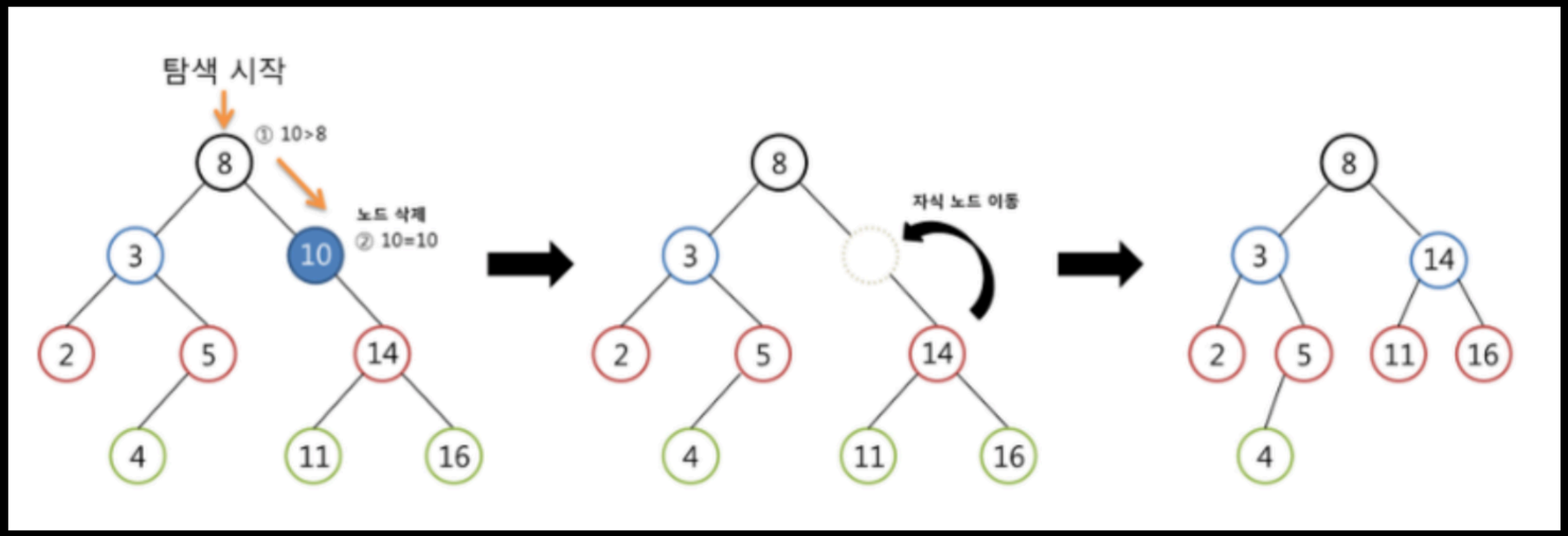
삭제 과정에 대해서 경우에 따른 예시를 살펴보면 다음과 같습니다.

1. 자식노드가 없는 노드(리프 노드) 경우



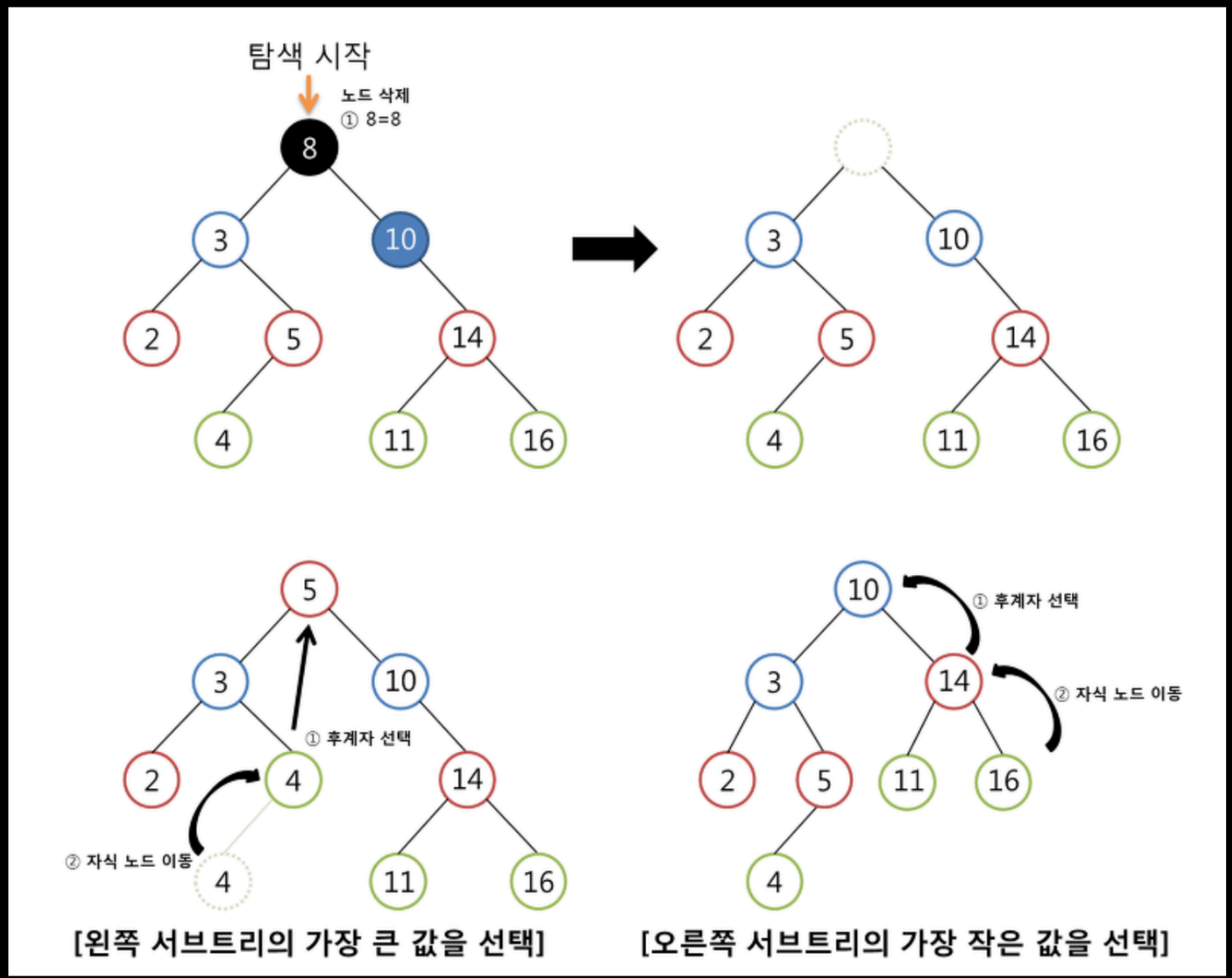
Delete

2. 자식 노드가 1개인 노드인 경우



Delete

3. 자식 노드가 2개인 노드인 경우



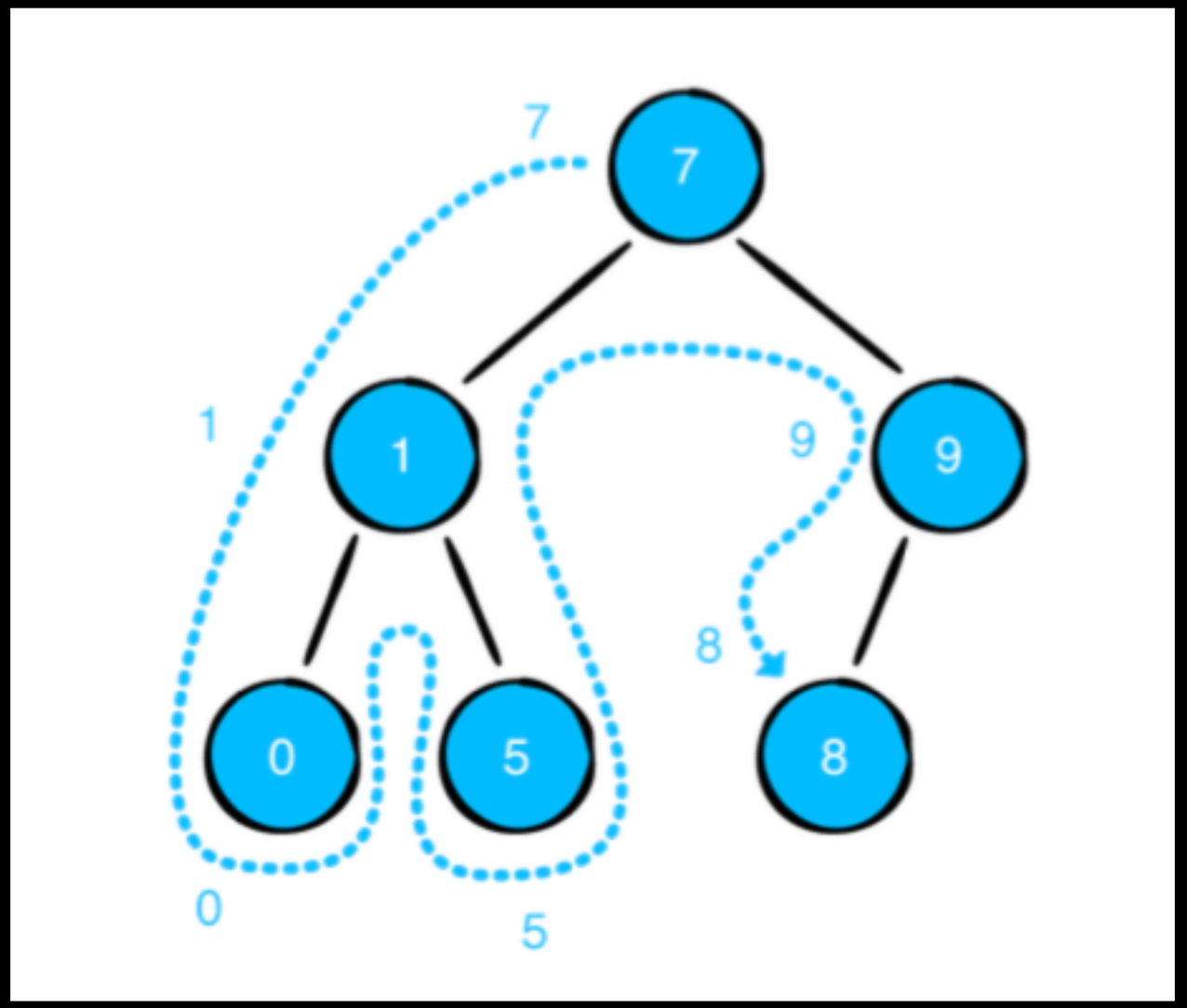
Traversal

이전의 이진 트리에서도 설명하였지만 이진 탐색 트리(Binary Search Tree)도 역시 데이터를 접근(순회)하는 방법은 아래와 같이 3가지 방식이 있습니다.

1. 전위 순회(Preorder Traversal) : 루트(현재) 노드 -> 왼쪽 자식 노드-> 오른쪽 자식 노드를 재귀적으로 방문하는 방식
2. 중위 순회(Inorder Traversal) : 왼쪽 자식 노드-> 오른쪽 자식 노드 -> 루트 노드를 재귀적으로 방문하는 방식
3. 후위 순회(Postorder Traversal) : 왼쪽 자식 노드-> 오른쪽 자식 노드 -> 루트 노드를 재귀적으로 방문하는 방식

Traversal

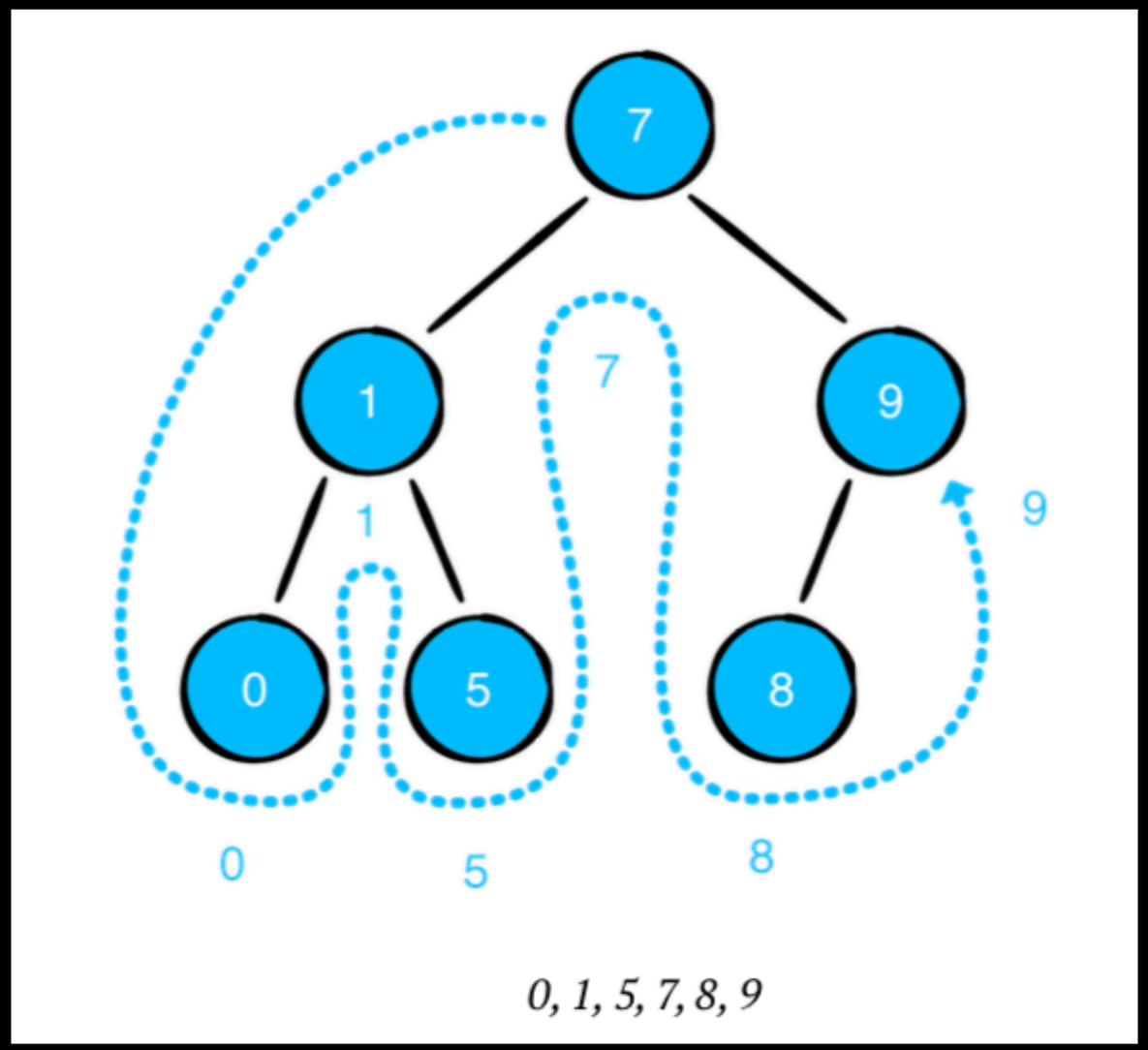
전위 순회(Preorder Traversal)



루트(현재) 노드 -> 왼쪽 자식 노드-> 오른쪽 자식 노드

Traversal

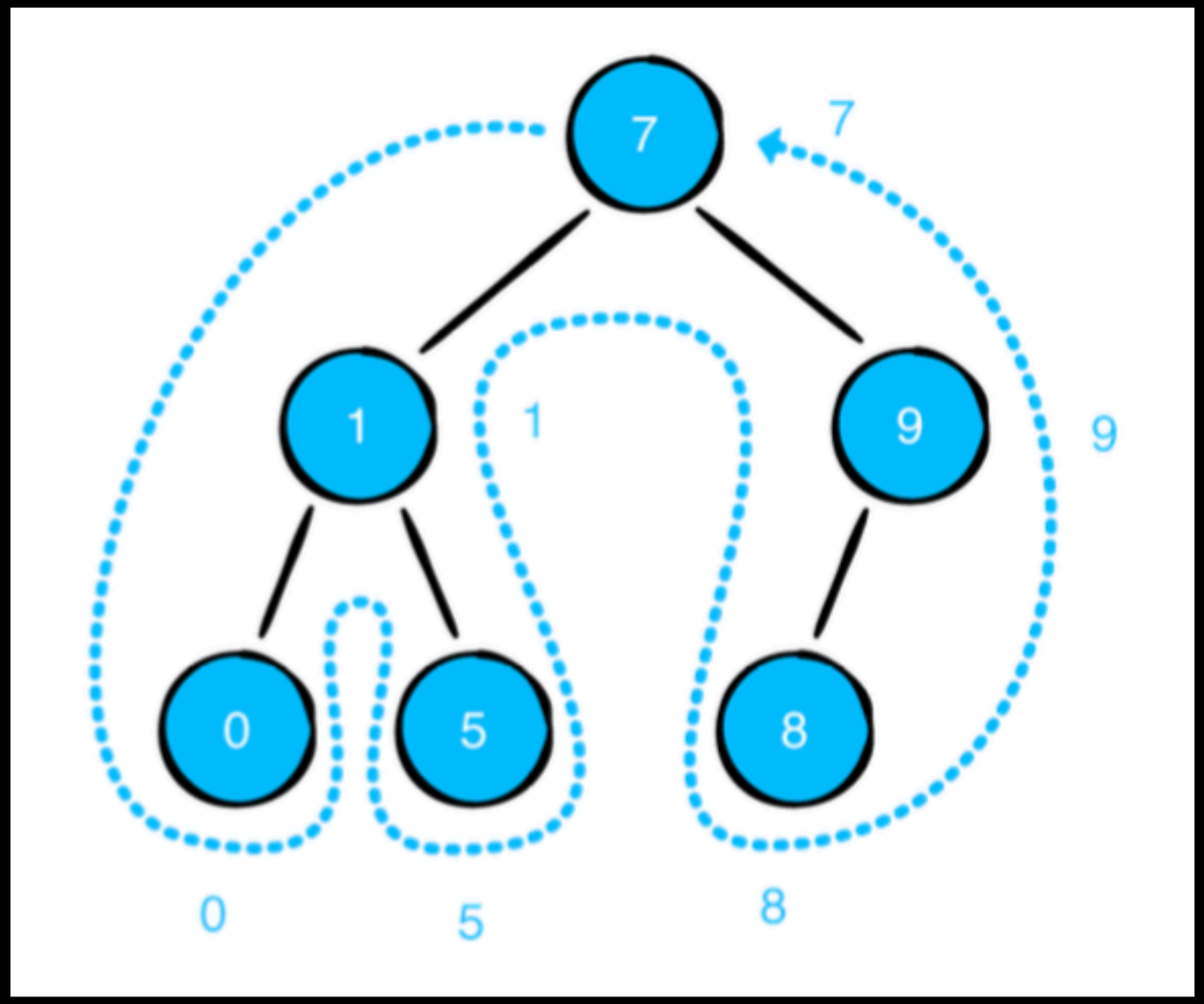
중위 순회(Inorder Traversal)



왼쪽 자식 노드-> 오른쪽 자식 노드 -> 루트 노드

Traversal

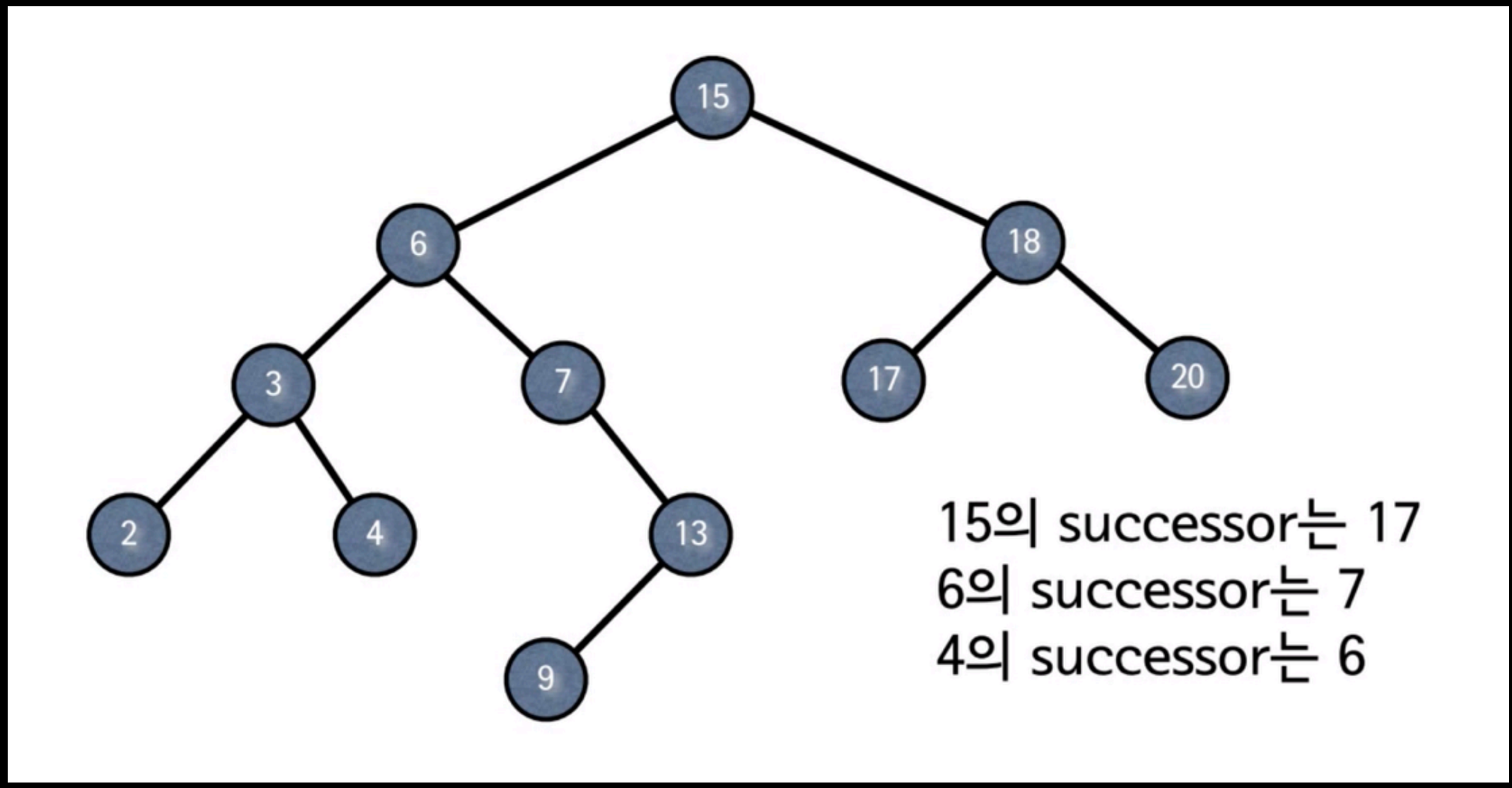
후위 순회(Postorder Traversal)



왼쪽 자식 노드-> 오른쪽 자식 노드 -> 루트 노드

Successor

특정 노드 키값보다 큰 것 중에서 가장 작은 노드를 말합니다.



Successor

Successor 값을 구할 경우 아래와 같은 세가지 경우가 존재할 수 있습니다.

1. 노드 X의 오른쪽 서브 트리가 존재하는 경우 :
오른쪽 서브 트리의 최소값이 바로 Successor(Ex. 15의 Successor는 17)

2. 오른쪽 서브 트리가 없는 경우 :
오른쪽 서브 트리가 없는 경우 부모 노드로 올라간다. 올라가는 과정에서 오른쪽으로 올라갈 경우 해당 노드가 바로 Successor(Ex. 17의 Successor는 18, 13의 Successor는 15)

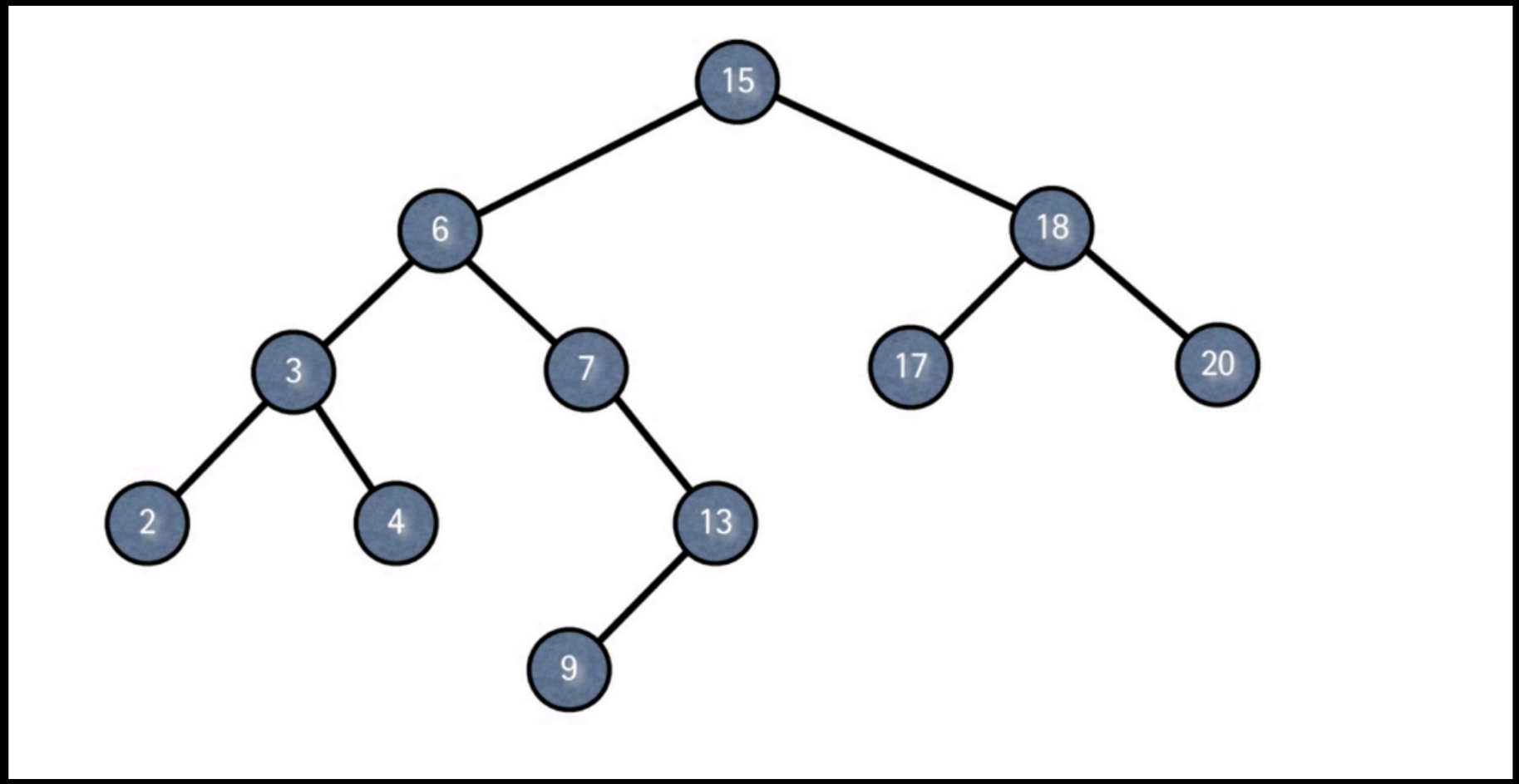
3. 1, 2 과정을 거쳤는데는 값을 찾지 못하면 Successor가 없는 것 (Ex. 20의 Successor는 없음)

Predecessor

특정 노드 키값보다 작은 것 중에서 가장 큰 노드를 말합니다.

Successor 값을 구할때와 반대의 과정을 거치면 Predecessor 값을 찾을 수 있습니다.

Predecessor



6의 Predecessor는 4
17의 Predecessor는 15
3의 Predecessor는 2

Implementation

Swift를 활용하여 가장 기본적인 Binary Search Tree를 구현해 보겠습니다.

우선 기본적으로 필요한 메소드는 아래와 같습니다.

- init : 이진 탐색 트리를 초기화하고 루트 노드 값을 입력하는 함수
- insert : 노드를 추가하는 함수
- remove : 특정 노드를 삭제하는 함수
- search : 특정 키 값을 검색하는 함수
- traversePreOrder : 전위 순회를 수행하는 함수
- traverseInOrder : 중위 순회를 수행하는 함수
- traversePostOrder : 후위 순회를 수행하는 함수

Implementation

```
public class BinarySearchTree<T: Comparable> {
    fileprivate(set) public var value: T
    fileprivate(set) public var parent: BinarySearchTree?
    fileprivate(set) public var left: BinarySearchTree?
    fileprivate(set) public var right: BinarySearchTree?

    public init(value: T) {
        self.value = value
    }

    public convenience init(array: [T]) {
        precondition(array.count > 0)
        self.init(value: array.first!)

        for v in array.dropFirst() {
          insert(value: v)
        }
    }
}
```

Implementation

```
extension BinarySearchTree {
    public func insert(value: T) {
        if value < self.value {
            if let left = left {
                left.insert(value: value)
            } else {
                left = BinarySearchTree(value: value)
                left?.parent = self
            }
        } else {
            if let right = right {
                right.insert(value: value)
            } else {
                right = BinarySearchTree(value: value)
                right?.parent = self
            }
        }
    }
}
```

Implementation

```
extension BinarySearchTree {
   @discardableResult public func remove() -> BinarySearchTree? {
        let replacement: BinarySearchTree?

        if let right = right {
            replacement = right.minimum()
        } else if let left = left {
            replacement = left.maximum()
        } else {
            replacement = nil
        }

        replacement?.remove()

        replacement?.right = right
        replacement?.left = left
        right?.parent = replacement
        left?.parent = replacement

        reconnectParentTo(node:replacement)

        parent = nil
        left = nil
        right = nil

        return replacement
    }

    private func reconnectParentTo(node: BinarySearchTree?) {
        if let parent = parent {
            if isLeftChild {
                parent.left = node
            } else {
                parent.right = node
            }
        }

        node?.parent = parent
    }
}
```

Implementation

```
extension BinarySearchTree {
    public func search(value: T) -> BinarySearchTree? {
        var node: BinarySearchTree? = self

        while let n = node {
            if value < n.value {
                node = n.left
            } else if value > n.value {
                node = n.right
            } else {
                return node
            }
        }

        return nil
    }
}
```

Implementation

```
extension BinarySearchTree {
    public func traversePreOrder(process: (T) -> Void) -> [T] {
        var results = [T]()

        process(value)
        results.append(value)
        results += left?.traversePreOrder(process: process) ?? []
        results += right?.traversePreOrder(process: process) ?? []

        return results
    }

    public func traverseInOrder(process: (T) -> Void)-> [T] {
        var results = [T]()

        results += left?.traverseInOrder(process: process) ?? []
        process(value)
        results.append(value)
        results += right?.traverseInOrder(process: process) ?? []

        return results
    }

    public func traversePostOrder(process: (T) -> Void) -> [T] {
        var results = [T]()

        results += left?.traversePostOrder(process: process) ?? []
        results += right?.traversePostOrder(process: process) ?? []
        process(value)
        results.append(value)

        return results
    }
}
```

Implementation

```
extension BinarySearchTree: CustomStringConvertible {
    public var description: String {
        var s = ""

        if let left = left {
            s += "(\(left.description)) <- "
        }

        s += "\(value)"

        if let right = right {
            s += " -> (\(right.description))"
        }

        return s
    }

    public func toArray() -> [T] {
        return map { $0 }
    }
}
```

Implementation

```
let tree = BinarySearchTree<Int>(value: 7)
tree.insert(value: 2)
tree.insert(value: 5)
tree.insert(value: 10)
tree.insert(value: 9)
tree.insert(value: 1)

print(tree)
// ((1) <- 2 -> (5)) <- 7 -> ((9) <- 10)

print(tree.toArray())
// [1, 2, 5, 7, 9, 10]

let toDelete = tree.search(value: 2)
toDelete?.remove()

print(tree)
// ((1) <- 5) <- 7 -> ((9) <- 10)
```

References

[1] Swift, Data Structure, Binary Search Trees : https://devmjun.github.io/archive/BinarySearchTree

[2] [Swift] Binary Search Tree (이진 탐색 트리) 를 Swift로 구현해보자 + search / insert / delete : https://eunjin3786.tistory.com/17

[3] 스위프트: 이진 탐색 트리(1 / 2): #BinarySearchTree: #자료구조: #배열과 비교: #트리: #탐색: #삽입: #삭제 : https://the-brain-of-sic2.tistory.com/26

[4] [Swift] 이진 탐색 트리 : https://milyo-codingstories.tistory.com/60

[5] [Swift] Binary Search Tree (이진 탐색 트리) 를 Swift로 구현해보자 + search / insert / delete : https://eunjin3786.tistory.com/17

References

[6] 이진 검색 트리(Binary Search Tree) : https://kka7.tistory.com/78

[7] [Swift] 이진 탐색 트리 : https://milyo-codingstories.tistory.com/60

[8] 이진 탐색 트리 : https://ko.wikipedia.org/wiki/이진_탐색_트리

[9] 자료구조 :: 트리(4) "이진탐색트리" : http://egloos.zum.com/printf/v/699558

[10] 이진 탐색 트리 (Binary Search Tree) : https://velog.io/@hanrimjo/이진-탐색-트리-Binary-Search-Tree-8gk6ablvfx

Thank you!