SWIFT
Data Structure - Heap

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Heap

힙(Heap)는 일종의 이진 트리(Tree)를 단일 배열을 이용하여 구현한 구조 가지는 자료구조로서 여러개의 값들 중에서 가장 큰 값이나 가장 작은 값을 빠르게 찾아내도록 하기위해서 만들어진 자료구조입니다.

힙은 완전 이진 트리와 같은 특성을 가지고 있으며 이진 탐색 트리와 비교시 약간 느슨한 상태의 반 정렬 상태를 가지는 형태를 취합니다.

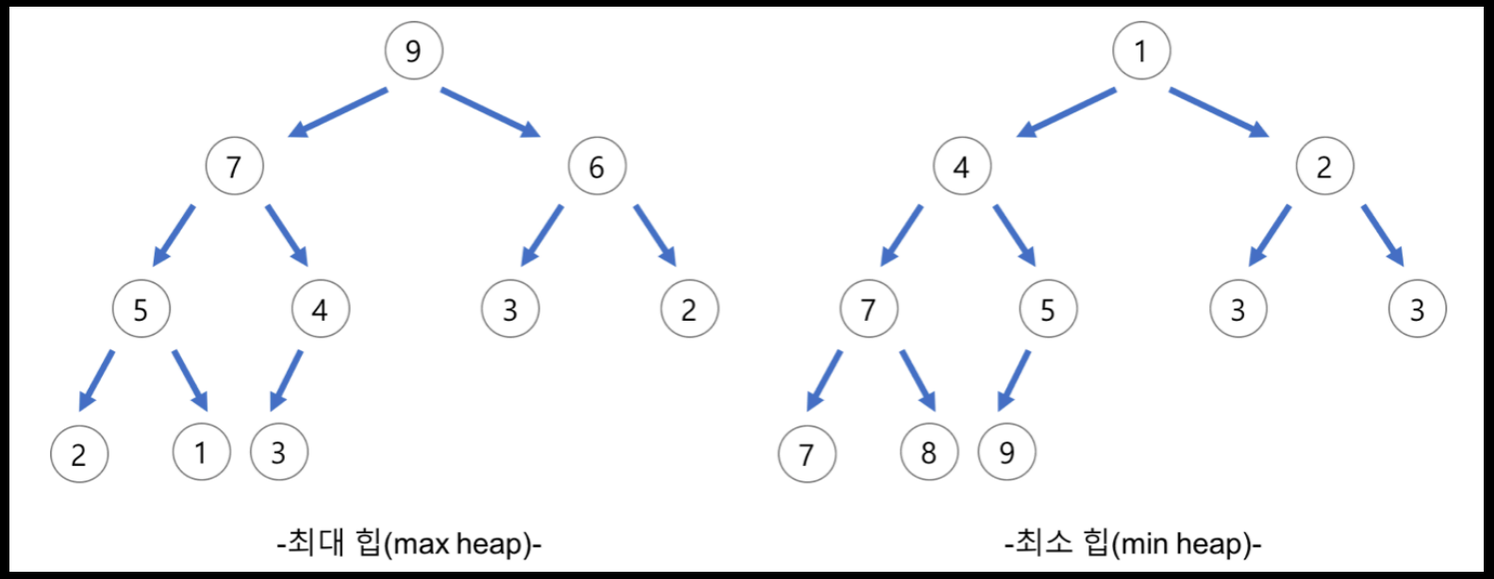
힙은 크게 정렬 상태에 따라서 아래와 같이 분류됩니다.

최대 힙(Max Heap) : 부모 노드의 키 값이 자식 노드의 키 값보다 크거나 같은 완전 이진 트리

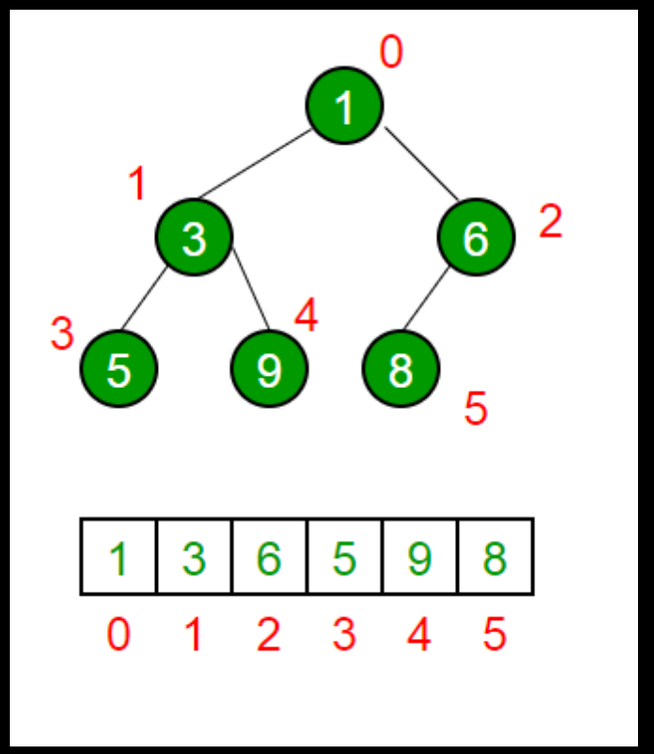
최소 힙(Min Heap) : 부모 노드의 키 값이 자식 노드의 키 값보다 작거나 같은 완전 이진 트리

Concept

Heap의 기본 컨셉은 아래와 같습니다.

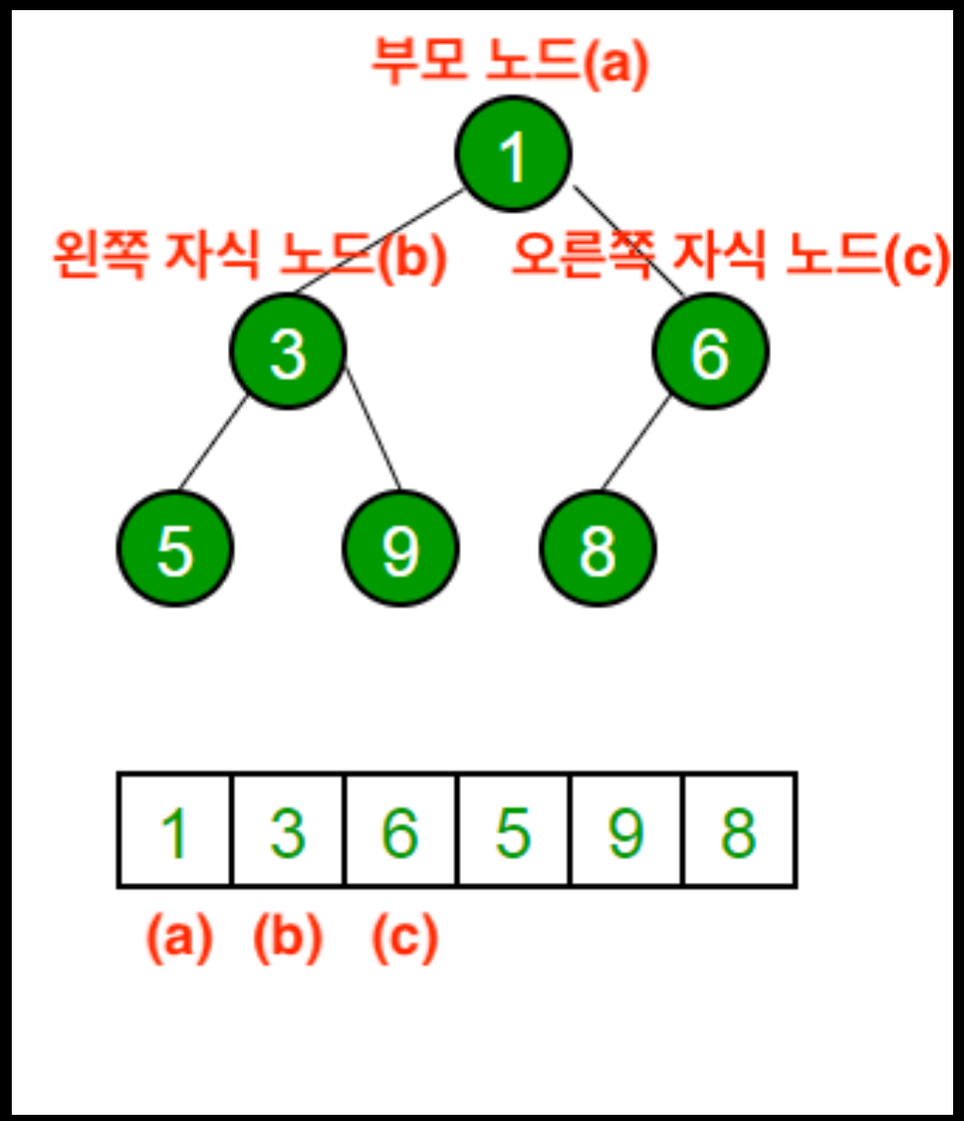


-최대 힙(max heap)- -최소 힙(min heap)-



Concept

Heap에서의 부모 노드와 자식 노드의 관계 공식은 아래와 같습니다.



(a) 부모의 인덱스 = (자식의 인덱스-1) / 2
Ex) 0 = 0 / 2, 0 = 1 / 2

(b) 왼쪽 자식의 인덱스 = (부모의 인덱
스) * 2 + 1
Ex) 1 = (0 / 2) + 1

(c) 오른쪽 자식의 인덱스 = (부모의 인덱스) * 2 + 2
Ex) 1 = (0 / 2) + 2

Concept

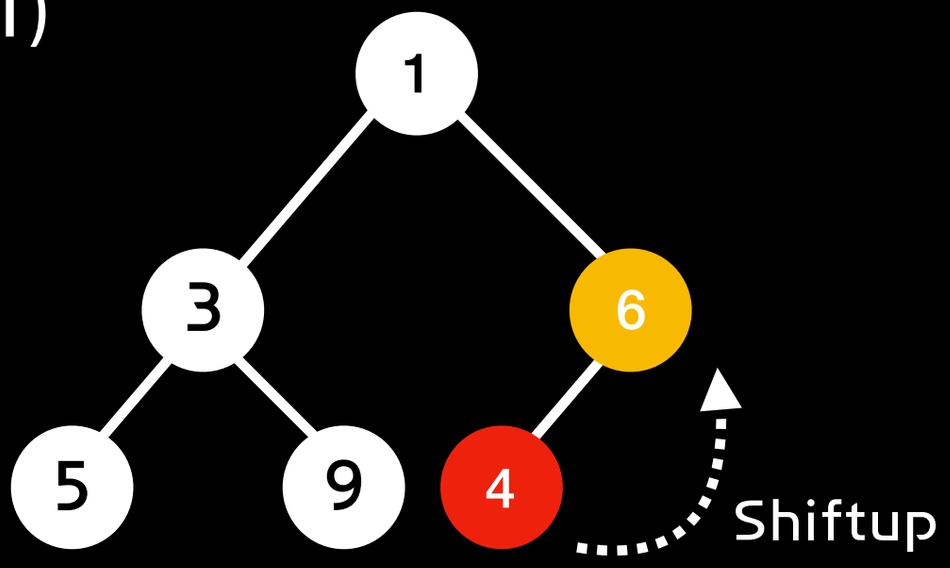
최소 Heap에서의 데이터 삽입 과정을 살펴보면 아래와 같습니다.

예) 1, 3, 6, 5, 9 에서 데이터 4 추가 시
-> 1, 3, 4, 5, 9, 6

1. 우선 데이터 추가 시 배열 가장 마지막에 데이터를 삽입
2. 위의 부모 노드와의 비교 후 입력된 데이터가 더 작을 경우 부모 인덱스와 현재 인덱스 및 데이터 값 교체(ShiftUp)
3. 마지막 루트 노드(최 상단 노드)까지 2번의 과정을 반복
4. 최상단 노드까지 비교를 마치면 최종 데이터 삽입 완료

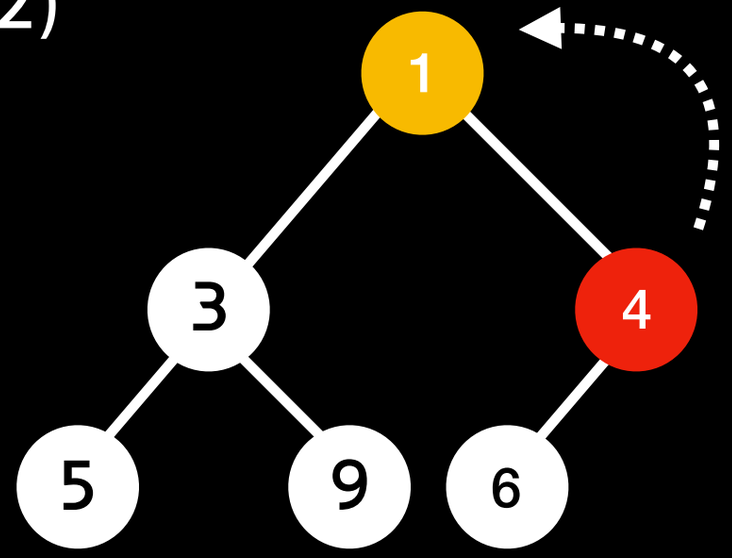
Concept

(1)



[1, 3, 6, 5, 9, 4]

(2)



[1, 3, 4, 5, 9, 6]

Concept

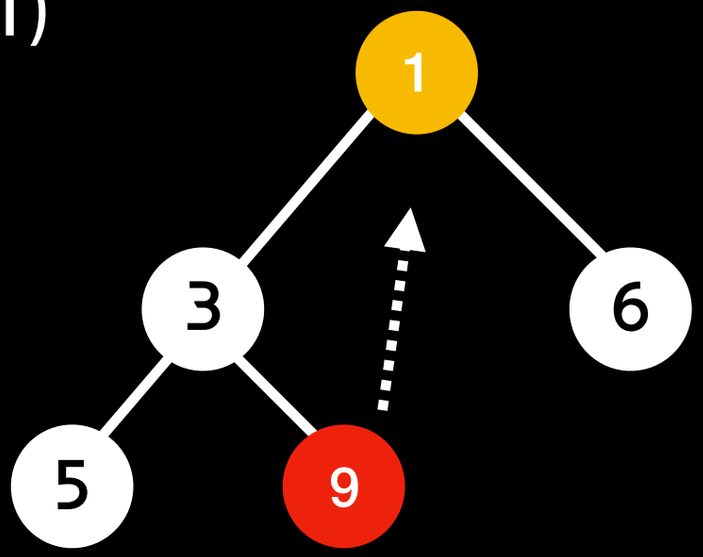
최소 Heap에서의 루트 데이터 삭제 과정을 살펴보면 아래와 같습니다.

예) 1, 3, 6, 5, 9에서 데이터 1 삭제 시
-> 3, 5, 6, 9

1. 루트 노드 삭제
2. 마지막 노드를 루트로 이동
3. 루트로 옮겨진 노드 아래의 자식 노드와 힙 조건 검사 후 조건이 맞지 않으면 인덱스 및 데이터 교체(ShiftDown)
4. 제일 마지막 자식 노드를 만날때까지 3번의 과정을 반복
5. 마지막 자식 노드까지 검사를 완료하면 삭제 과정 완료

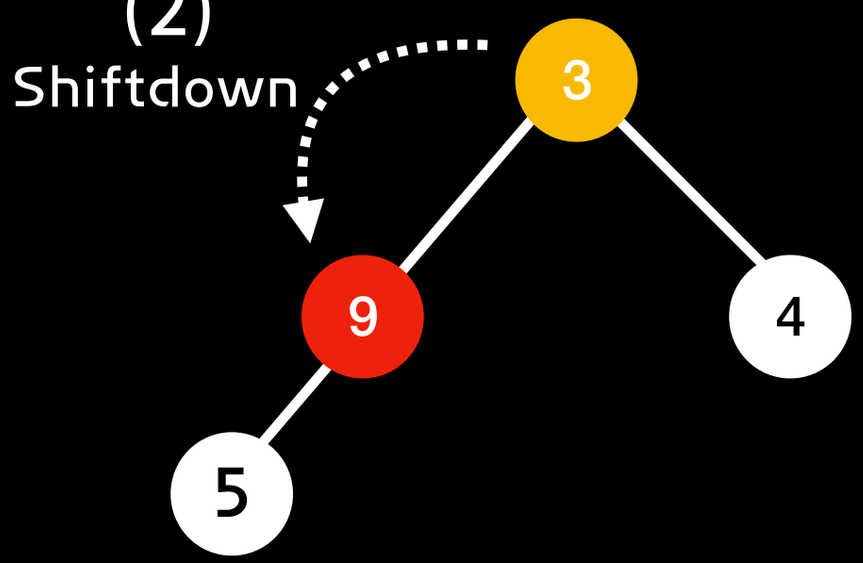
Concept

(1)



[1, 3, 6, 5, 9]

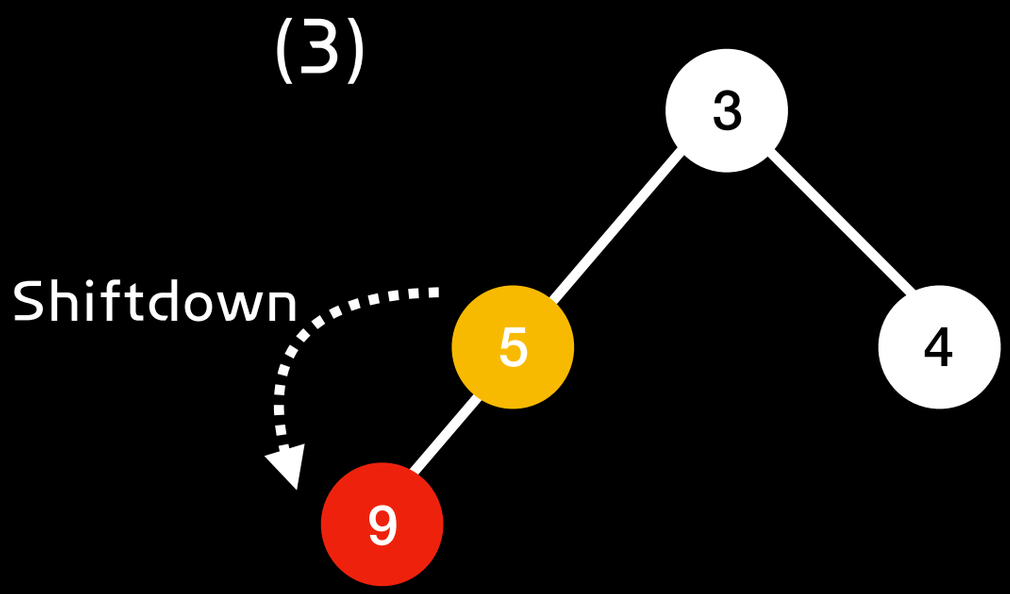
(2)



[3, 9, 4, 5]

Concept

(3)



[3, 5, 4, 9]

Concept

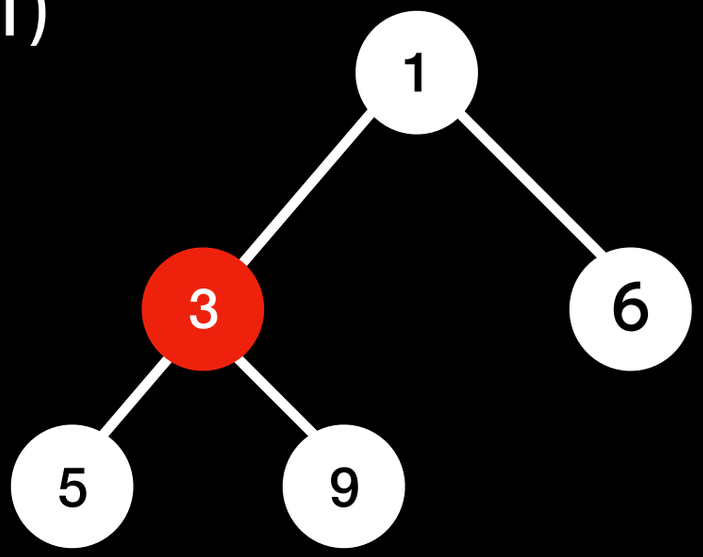
최소 Heap에서의 중간 데이터 삭제 과정을 살펴보면 아래와 같습니다.

예) 1, 3, 6, 5, 9에서 데이터 3 삭제 시
-> 1, 5, 6, 9

1. 삭제할 인덱스가 가장 마지막일 경우는 데이터 삭제 후 바로 종료
2. 제거할 데이터를 제일 마지막에 데이터와 교체(swap)
3. 교체한 후 마지막 데이터 삭제
4. 최초 교체했던 인덱스 기준으로 ShifDown 정렬
5. 이후 교체했던 인덱스 기준으로 ShifUp 정렬
6. 최종 정렬된 이후 삭제 과정 종료

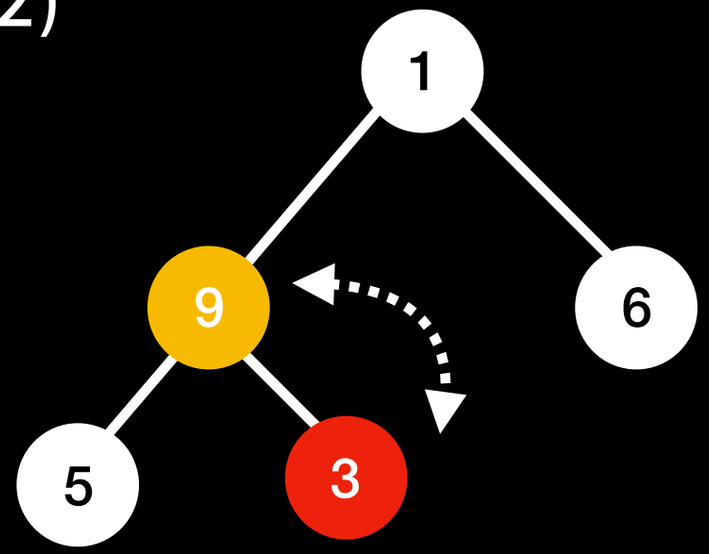
Concept

(1)



[1, 3, 6, 5, 9]
Index : 1

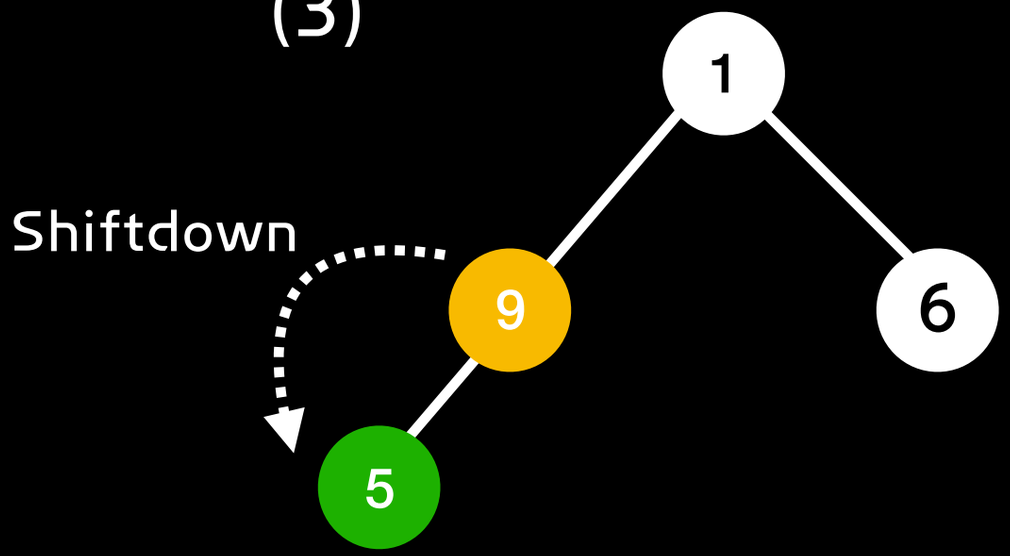
(2)



[1, 9, 6, 5, 3]
Index : 1

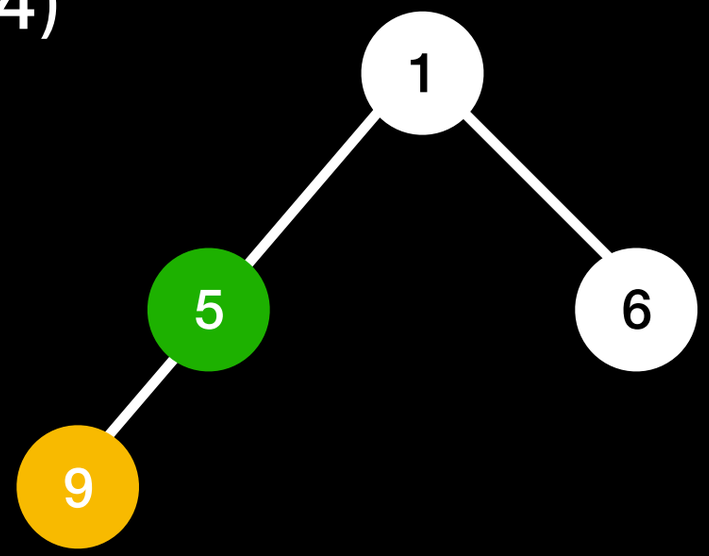
Concept

(3)



[1, 9, 6, 5]
Index : 1

(4)



[1, 5, 6, 9]
Index : 1

Features

Heap의 특징을 살펴보면 아래와 같습니다.

- 최대, 최소 값을 쉽고 빠르게 찾을 수 있음
- 이진 트리를 1차 배열로 저장하여 표현
- 삽입, 삭제 평균 실행 시간은 O(logn)
- 완전 이진 트리의 일종으로 우선순위 큐에서 활용
- 힙은 일종의 반정렬 상태(느슨한 정렬 상태)를 유지
- 힙 트리에서는 중복된 값을 허용
- 힙 정렬(O(nLog n)), 우선 순위 큐(O(log n)), 그래프 알고리즘 등에서 활용

Implementation

Swift를 활용하여 Heap을 구현해보겠습니다.
우선 필요한 메소드는 아래와 같습니다.

- init : 리스트를 초기화하는 함수
- leftChildIndex : 왼쪽 자식 노드 인덱스 반환
- rightChildIndex : 오른쪽 자식 노드 인덱스 반환
- parentIndex : 부모 노드 인덱스 반환
- insert : 데이터 입력(마지막 혹은 특정 노드 위치)
- remove : 루트 노드 및 특정 노드 삭제
- siftDown : 자식 노드와의 비교 후 힙 정렬
- siftUp : 부모 노드와의 비교 후 힙 정렬
- count : 현재 리스트의 크기를 반환
- peek : 루트 노드(최상위 노드) 반환
- isEmpty : 현재 리스트의 크기가 비어있는지 체크

Implementation

```
class Heap<Element : Comparable> {

    var elements: [Element] = []
    let sort: (Element, Element) -> Bool

    init(sort: @escaping (Element, Element) -> Bool, elements: [Element] = []) {
        self.sort = sort
        self.elements = elements

        if !elements.isEmpty {
            for i in stride(from: elements.count / 2 - 1, through: 0, by: -1){
                siftDown(from: i)
            }
        }
    }

    func leftChildIndex(ofParentAt index: Int) -> Int {
        return 2*index + 1
    }

    func rightChildIndex(ofParentAt index: Int) -> Int {
        return 2*index + 2
    }

    func parentIndex(ofChildAt index: Int) -> Int {
        return (index - 1) / 2
    }
}
```

Implementation

```
extension Heap {
    func insert(_ element: Element) {
        elements.append(element)
        siftUp(from: elements.count - 1)

    }

    func remove() -> Element? {
        guard !isEmpty else {
            return nil
        }
        elements.swapAt(0, count - 1)
        defer {
            siftDown(from: 0)
        }
        return elements.removeLast()
    }

    func remove(at index: Int) -> Element? {
        guard index < elements.count else {  return nil }

        if index == elements.count - 1 {
            return elements.removeLast()
        }else {
            elements.swapAt(index, elements.count-1)
            siftDown(from: index)
            siftUp(from: index)
        }
        return elements.removeLast()
    }
}
```

Implementation

```
extension Heap {
    func siftDown(from index: Int) {
        var parent = index

        while true {
            let left = leftChildIndex(ofParentAt: parent)
            let right = rightChildIndex(ofParentAt: parent)

            var candidate = parent // 탐색할 아이 지정
            if left < count , sort(elements[left], elements[candidate]) {
                candidate = left
            }
            if right < count , sort(elements[right], elements[candidate]) {
                candidate = right
            }
            if candidate == parent {  // 종료조건
                return
            }

            elements.swapAt(parent, candidate)
            parent = candidate
        }
    }

    func siftUp(from index: Int) {
        var child = index // 마지막 인덱스

        while true {
            let parent = parentIndex(ofChildAt: child)

            if child > 0 && sort(elements[child], elements[parent]) { // > 이니까
                elements.swapAt(child, parent)
                child = parent
            }else {
                return
            }
        }
    }
}
```

Implementation

```
extension Heap {
    func peek() -> Element? {
        return elements.first
    }

    var count: Int {
        return elements.count
    }

    var isEmpty: Bool {
        return elements.isEmpty
    }
}
```

References

[1] [스위프트:자료구조] Heap: 힙 자료구조 (1 / 2) : Heap 이란? : https://the-brain-of-sic2.tistory.com/55

[2] [자료구조] 힙(heap)이란 : https://the-brain-of-sic2.tistory.com/63https://gmlwjd9405.github.io/2018/05/10/data-structure-heap.html

[3] (Swift) - 힙(Heap) - 자료구조에 대해 : https://sooooprmx.com/archives/6548

[4] Swift, Data Structure, Priority Queue : https://devmjun.github.io/archive/PriorityQueue

[5] [Swift 자료구조 ch07] Priority Queue : https://kor45cw.tistory.com/243?category=722252

References

[6] 자료구조 - 힙(Heap)이란? : https://galid1.tistory.com/485

[7] [프로그래머스 고득점Kit] #3 힙 : https://velog.io/@wan088/프로그래머스-고득점Kit-3-힙

[8] [자료구조](C++) 힙(heap) 삽입, 삭제, 정렬, 출력 구현하기(Max Heap) : http://blog.naver.com/PostView.nhn?blogId=kartmon&logNo=221543374899&parentCategoryNo=&categoryNo=39&viewDate=&isShowPopularPosts=false&from=postView

[9] 자료구조 힙(Heap) : https://yeolco.tistory.com/60

[10] 자료구조 :: 힙(1) "배열을 이용한 힙, 힙정렬" : http://egloos.zum.com/printf/v/700682

Thank you!