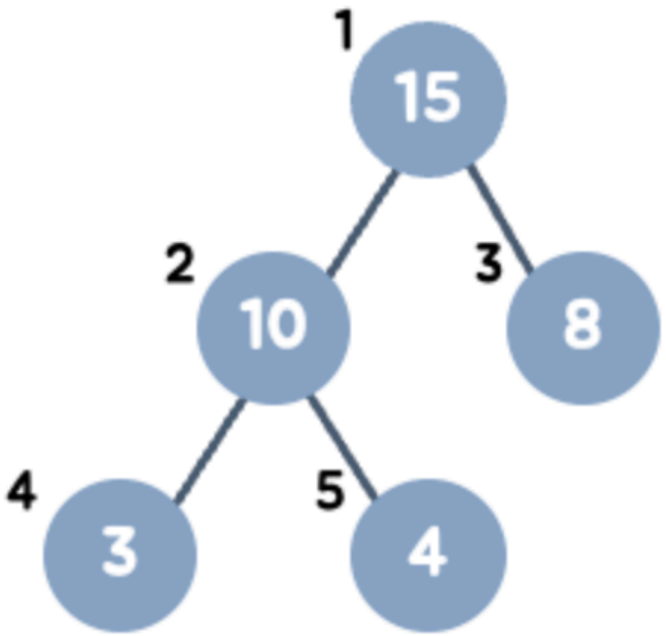
힙

생성일	@2025년 3월 28일 오전 10:09
태그	

힙이란?

- 최댓값, 최솟값을 빠르게 찾기 위한 자료형
- 우선순위 큐를 위해 만들어짐
- 완전 이진 트리의 일종이다.
- 각 노드의 키 값이 그 자식의 키 값보다 작지 않거나 크지 않은 완전 이진 트리이다.
    - 최대 힙
    - 최소 힙

완전 이진트리는 중복을 허용하지 않지만 힙은 허용한다.



특징

힙에서의 부모 자식 관계

- 오른쪽 자식의 인덱스 = (부모의 인덱스) * 2+1
- 왼쪽 자식의 인덱스 = (부모의 인덱스) * 2
- 부모의 인덱스 = (자식의 인덱스) / 2
- 중복값을 허용
- 부모-자식 간 정렬은 보장, 형제간의 정렬은 보장하지 않는 반 정렬 상태

사용이유

최솟값이나 최댓값을 찾기 위해

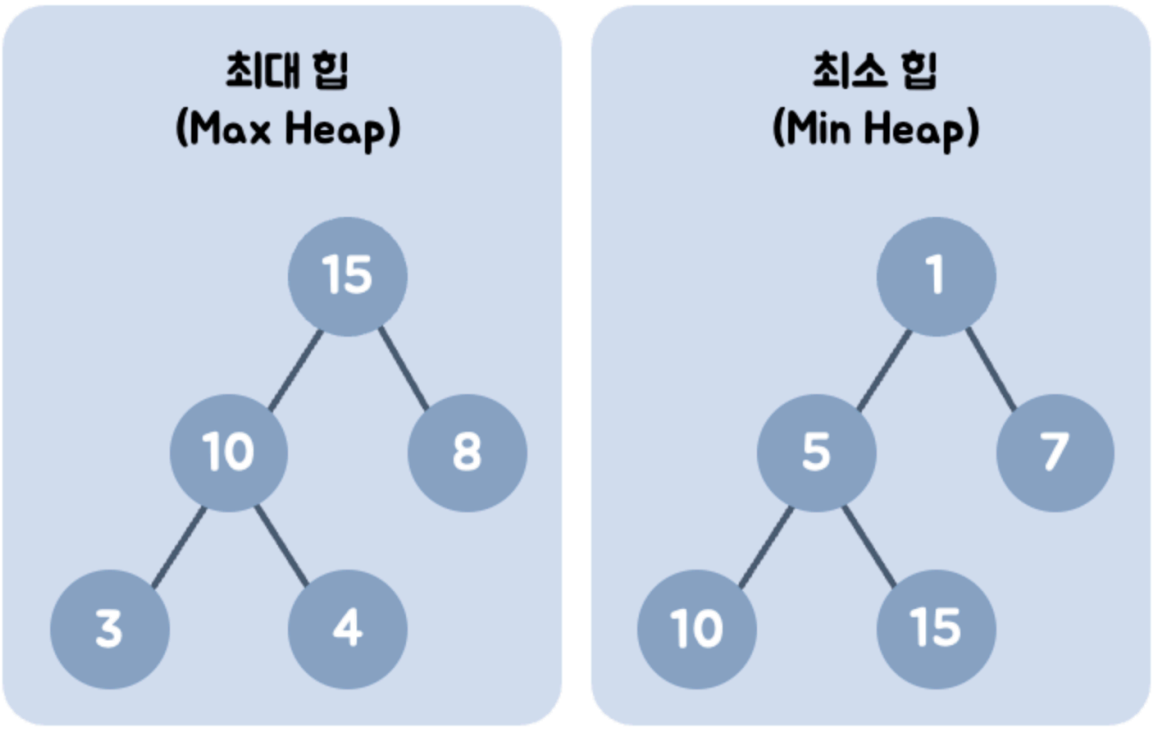
시간 복잡도

최솟값이나 최댓값을 찾기 위해 배열을 사용하면 O(n)만큼 시간이 걸린다.하지만 힙을 사용하면 O(logn)만큼 소요되므로, 배열을 사용할 때보다 빠르게 최솟값과 최댓값을 구할 수 있다.

우선순위 큐를 구현하는 표현 방법	삽입	삭제
순서 없는 배열	O(1)	O(n)
순서 없는 연결 리스트	O(1)	O(n)
정렬된 배열	O(n)	O(1)
정렬된 연결 리스트	O(n)	O(1)
힙(heap)	O(logn)	O(logn)

최대 힙 & 최소 힙

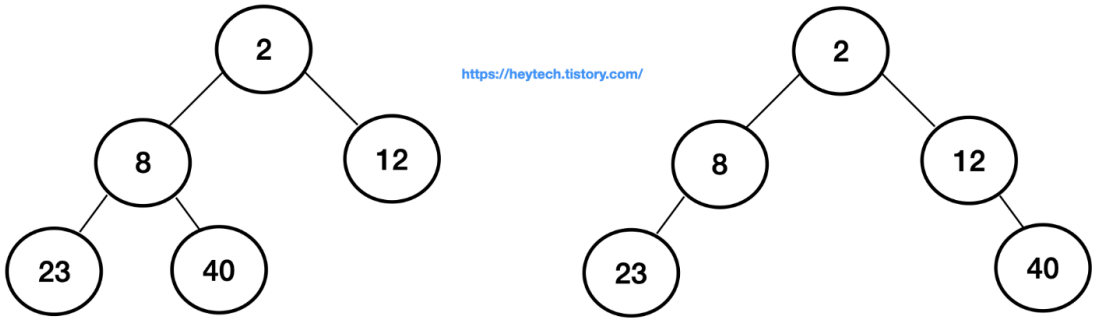
힙은 기본적으로 노드가 왼쪽부터 체워지는 완전 이진 트리이다.



완전 이진 트리

완전 이진 트리(Complete Binary Tree)란?

이진 트리에 노드를 삽입할 때 왼쪽부터 차례대로 삽입하는 트리이다.
위 그림과 같이 나타내며, 왼쪽이 비어있고 오른쪽이 채워져 있는 형태는 완전 이진 트리라고 할 수 없다.



완전 이진 트리(O)

완전 이진 트리(X)

이진 탐색 트리

이진 탐색과 연결리스트(linked-list)를 결합한 자료구조의 일종이다.

이진 탐색의 효율적인 탐색 능력을 유지하면서 빈번한 자료의 입력과 삭제를 가능하도록 한다.

각 노드에서 왼쪽의 자식 노드는 해당 노드보다 작은 값으로, 오른쪽의 자식 노드는 해당 노드보다 큰 값으로 이루어져 있다.

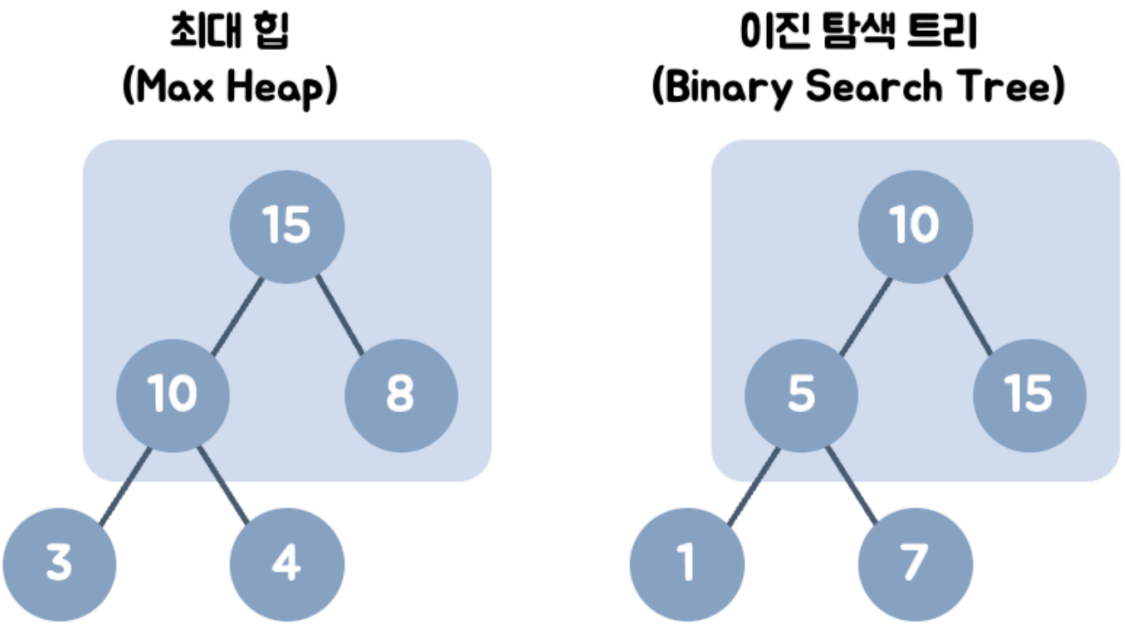
- 공통점

모두 완전 이진 트리이다.

- 차이점

최대힙의 경우에는 각 노드의 값이 자식 노드보다 크거나 같다.이진 탐색 트리는 각 노드의 왼쪽 자식은 더 작은 값으로, 오른쪽 자식은 더 큰 값으로 이루어져있다.

왼쪽 자식 노드 < 부모 노드 < 오른쪽 자식 노드



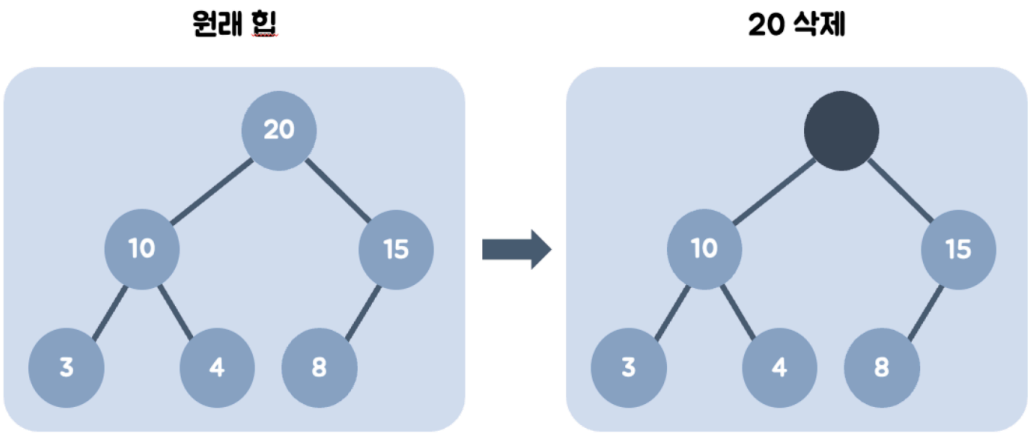
힙의 동작

데이터 삽입

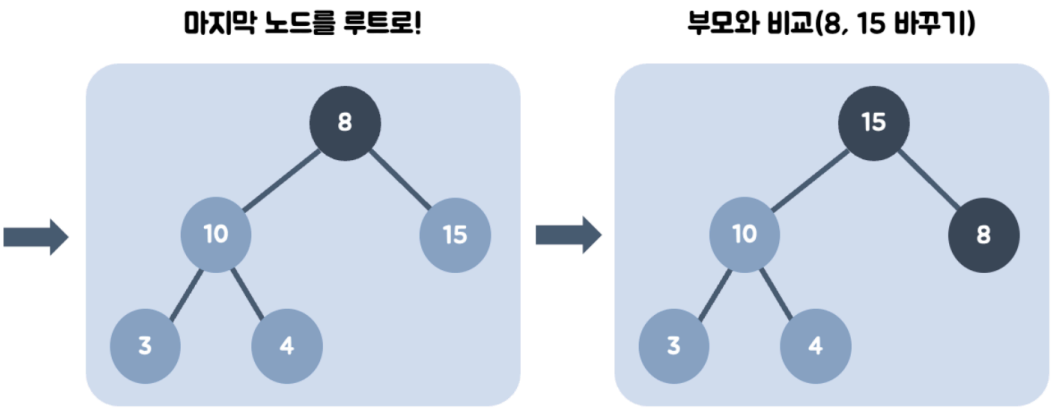
데이터 삭제

힙 자료구조의 목표는 바로 최대값이나 최소값을 알아내는 것이다.

때문에 데이터가 삭제 된다면 최대힙의 경우 가장 큰 값인 부모 노드의 값이 삭제된다.



최대값을 갖는 부모 노드를 삭제한다.



- 부모 노드가 비었으므로, 가장 최하단부 노드를 루트로 옮긴다.부모 노드인 8보다 값이 큰 자식 노드가 있는지 비교한다.
    - 왼쪽, 오른쪽 자식 노드 모두 부모 노드보다 클 경우왼쪽 자식 노드와 오른쪽 자식 노드를 비교하여, 더 큰 자식 노드와 부모 노드의 위치를 바꾼다.
    - 왼쪽, 오른쪽 자식 노드 중 하나만 부모 노드보다 클 경우둘 중에 부모 노드보다 큰 자식 노드와 부모 노드의 위치를 바꾼다.

구현 방법 - 우선 순위 큐

최소 힙

```
PriorityQueue<Integer> minHeap = new PriorityQueue<>();
```

이렇게 사용하여 컬렉션에 데이터를 넣으면 remove되는 peek의 값이 minHeap의 최소값이 된다.

- PriorityQueue는 기본적으로 오름차순 정렬이다.
- 숫자가 작을수록 우선순위가 높다.
- poll() peek()을 하면 가장 작은값으로 나온다
- 기본적으로 낮은 숫자부터 큰 숫자 오름차순이다.
- 즉 기본적으로 PriorityQueue는 MIn Heap 구조이다.
-

최대 힙

최대 힙을 사용하는 방법은 Comparator 값을 세팅해주어 사용할 수 있다.

```
PriorityQueue<Integer> maxHeap = new PriorityQueue<>(new Comparator
<Integer>() {
    @Override
    public int compare(Integer o1, Integer o2) {
        return -Integer.compare(o1, o2);
    }
});
```

Integer의 compare의 값이 default로 들어가게 되는데 이 값을 -로 바꾸어 주면 Max Heap이 된다.

🔍 어떻게 작동하나요?

기본 정렬 기준:

java 복사 편집

```
Integer.compare(3, 5); // 결과: -1 → 3이 5보다 작음 (작은 게 앞으로)
```

반대로 정렬:

java 복사 편집

```
-Integer.compare(3, 5); // 결과: 1 → 3이 5보다 크다고 판단 (큰 게 앞으로)
```

즉, `-Integer.compare(o1, o2)` 는 정렬 우선순위를 **내림차순(큰 수 먼저)** 으로 바꿔줘서 **Max Heap 구조가 되는 것**입니다.

맥스 힙 람다식도 가능

PriorityQueue<Integer> maxHeap = new PriorityQueue<>((a, b) → b - a);

```
//높은 숫자가 우선 순위인 int 형 우선순위 큐 선언
PriorityQueue<Integer> priorityQueueHighest = new PriorityQueue<>(Coll
ections.reverseOrder());
```

🎯 정리

힙 종류	선언 방법	우선순위 기준	결과
Min Heap	`new PriorityQueue<>()`	작은 수가 먼저	가장 작은 값부터
Max Heap	`new PriorityQueue<>(Comparator)`	큰 수가 먼저	가장 큰 값부터

PrioirityQueue

`PriorityQueue`는 **우선순위 큐**로,

항상 **우선순위가 가장 높은 값이 먼저 나오는 큐**입니다.

> 자바에서 PriorityQueue는 내부적으로 힙(Heap) 자료구조를 사용해 구현되어 있습니다.

✅ 주요 명령어(메서드)

메서드	설명
`add(E e)`	요소 추가 (`offer()` 과 거의 같음)
`offer(E e)`	요소 추가 (용량 제한 있을 때 예외 대신 false 반환)
`poll()`	**최우선순위** 요소 꺼내기 (삭제됨)
`peek()`	**최우선순위** 요소 확인 (삭제되지 않음)
`remove()`	최우선순위 요소 제거 (예외 발생 가능)
`isEmpty()`	큐가 비었는지 확인
`size()`	큐에 들어있는 요소 개수
`contains(Object o)`	해당 요소가 있는지 확인
`clear()`	큐 초기화 (모든 요소 제거)