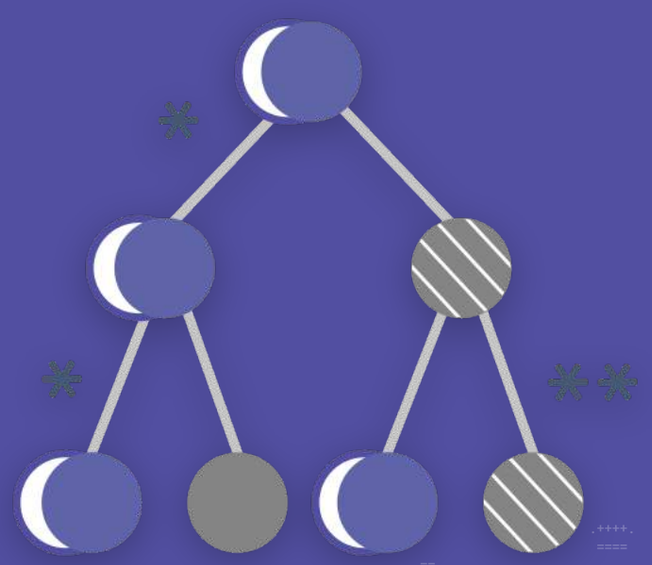
/* elice */

데이터구조 I

신현규 선생님 · 수 20:00



10월 11일 ~ 11월 7일

목차

주차별 커리큘럼

1주차 과정 소개, 배열, 연결리스트, 클래스

- 자료구조는 자료를 담는 주머니입니다. 배열, 연결 리스트의 개념과 장단점을 알아봅니다.

2주차 스택, 큐, 해싱

- 초급 자료구조와 자료를 저장·검색할 때 사용되는 해싱을 배워봅니다.

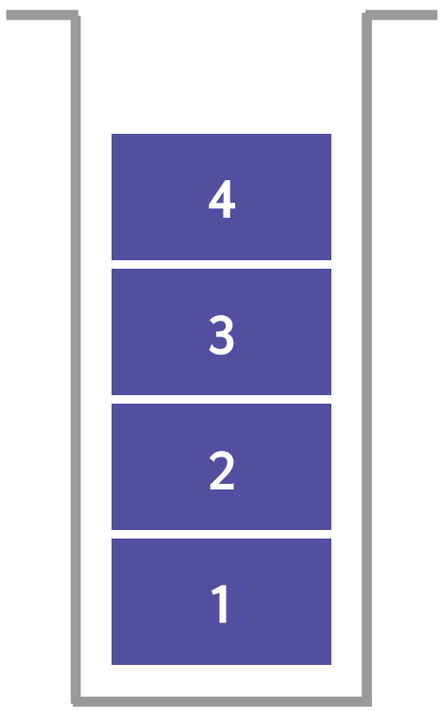
3주차 트리, 트리순회, 재귀호출

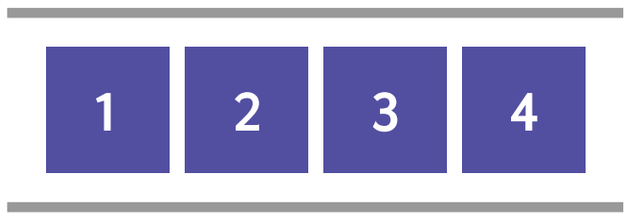
- 나무와 비슷하게 생긴 자료인 트리에 대해 배워보고 트리에서 자료를 탐색하는 알고리즘과 재귀호출을 배워봅니다.

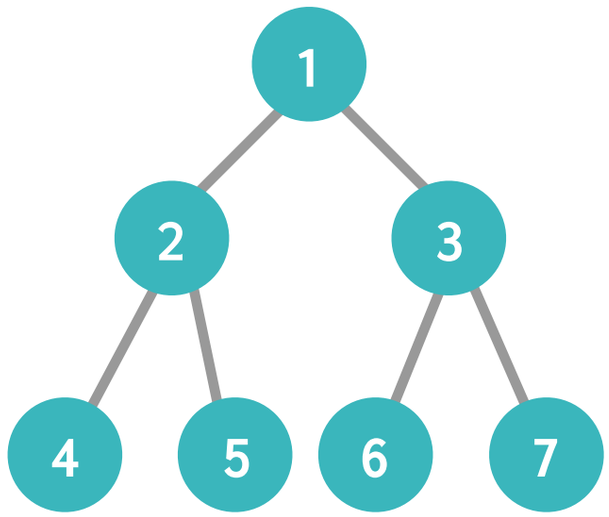
4주차 **재귀호출 응용 및 힙**

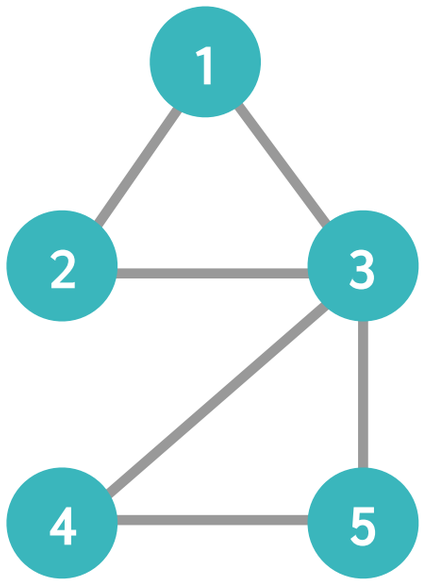
- 재귀호출로 해결할 수 있는 문제를 알아보고 그 의미를 찾아봅니다. 힙에 대해 알아보고,
  이를 이용하여 문제를 해결합니다.

대표적인 자료구조









스택 (Stack)

Last In First Out

큐 (Queue)

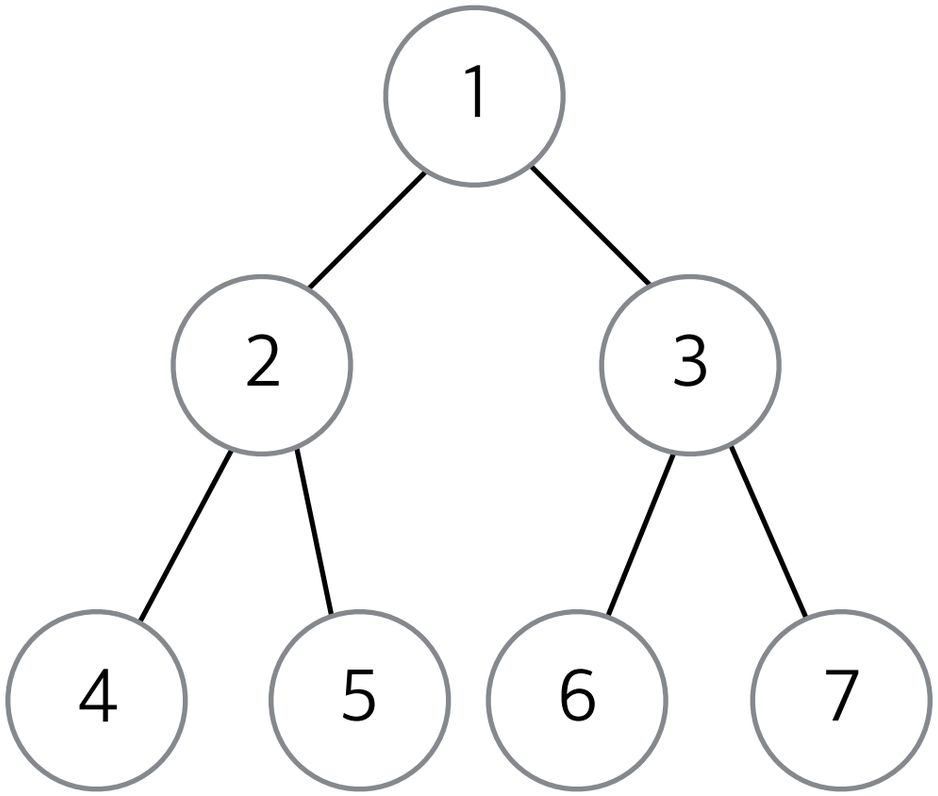
First In First Out

트리 (Tree)

그래프 (Graph)

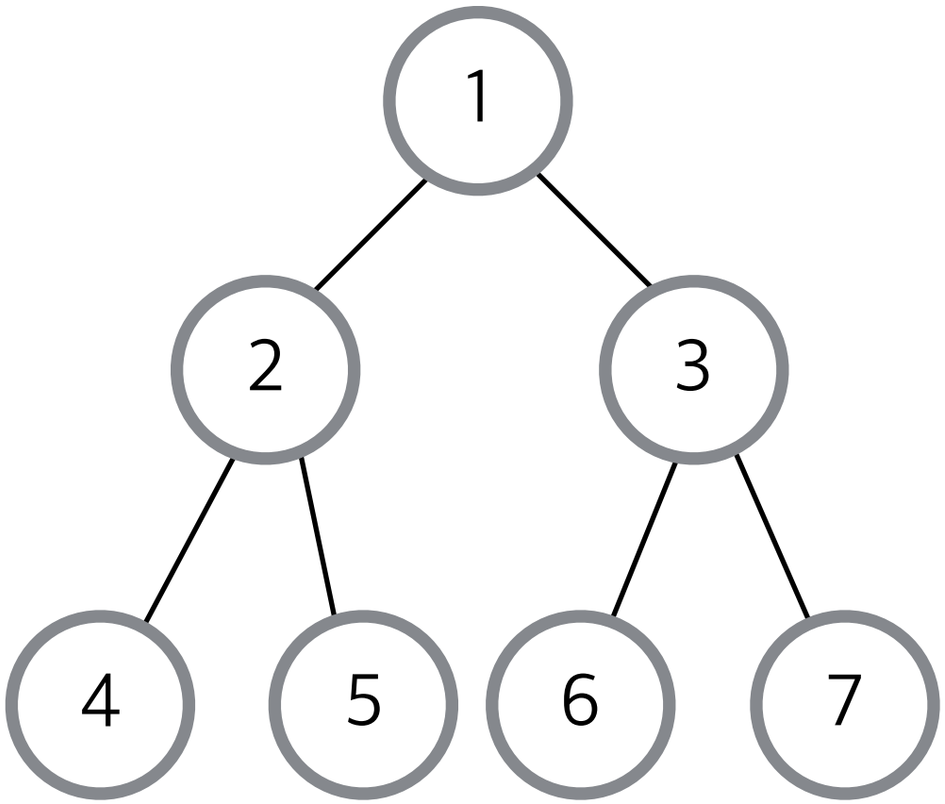
트리

아래와 같이 생긴 자료구조



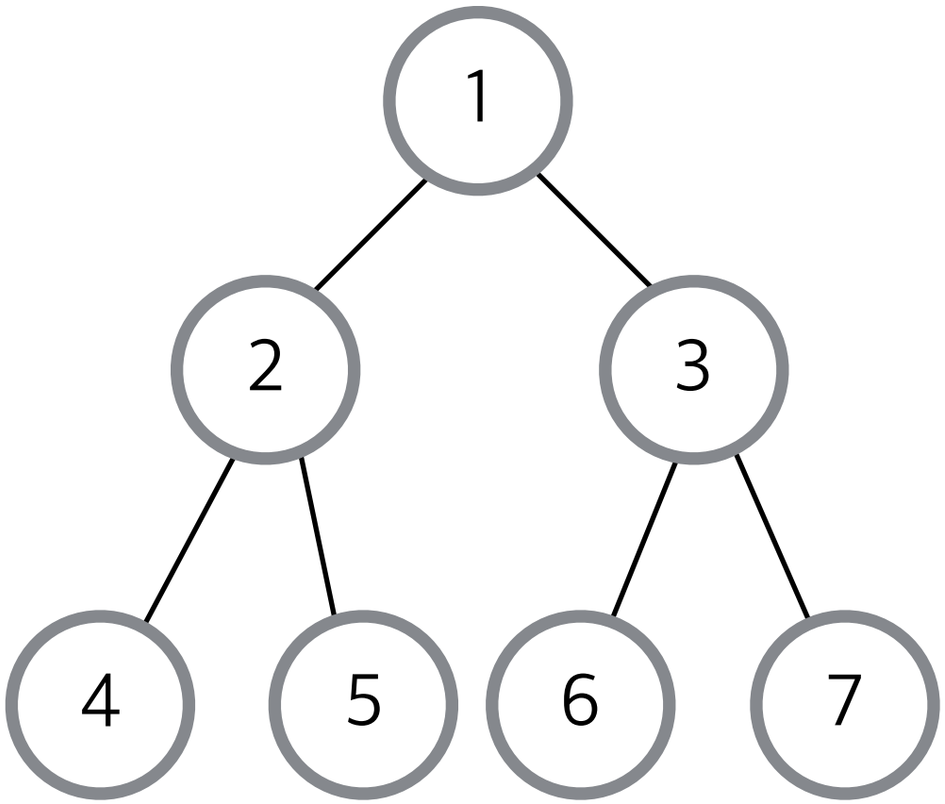
트리 순회

트리 내에 어떠한 자료가 담겨있는지를 알기 위함



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

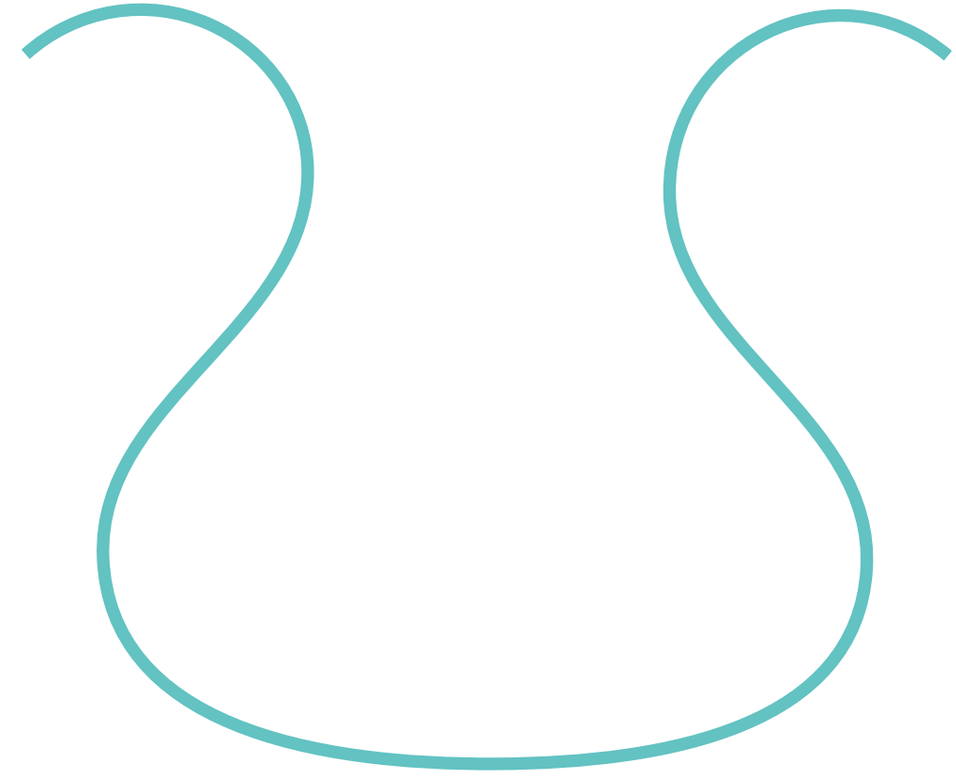
- **후위순회 :**

  L - R - Root 4 5 2 6 7 3 1

우선순위 큐

원소를 제거할 시,

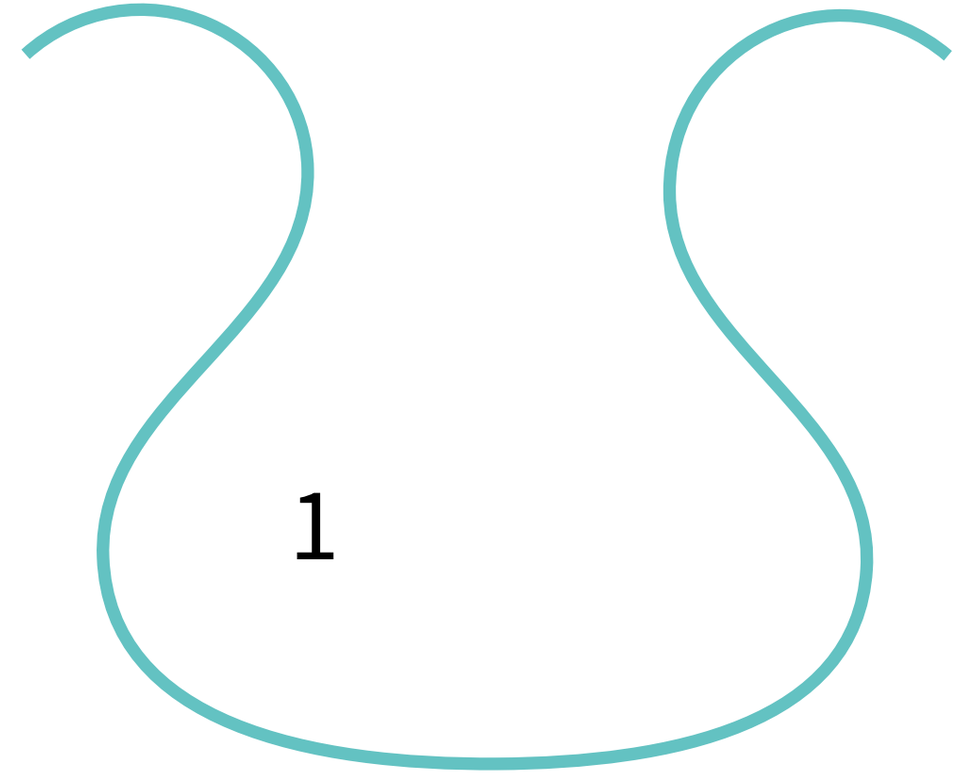
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

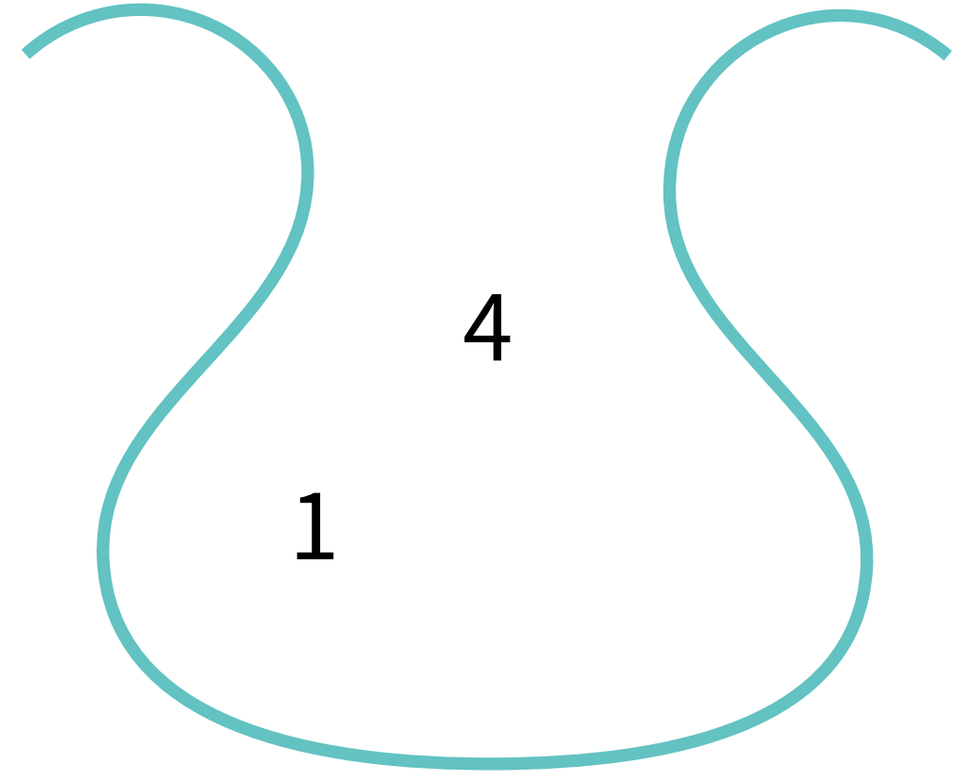
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

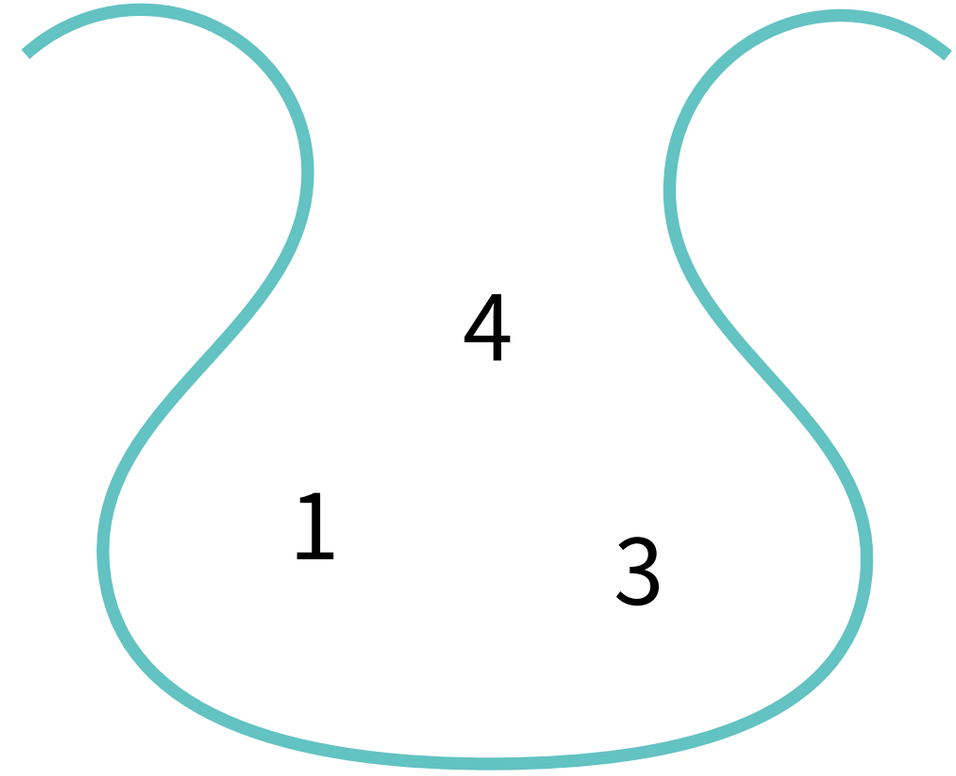
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

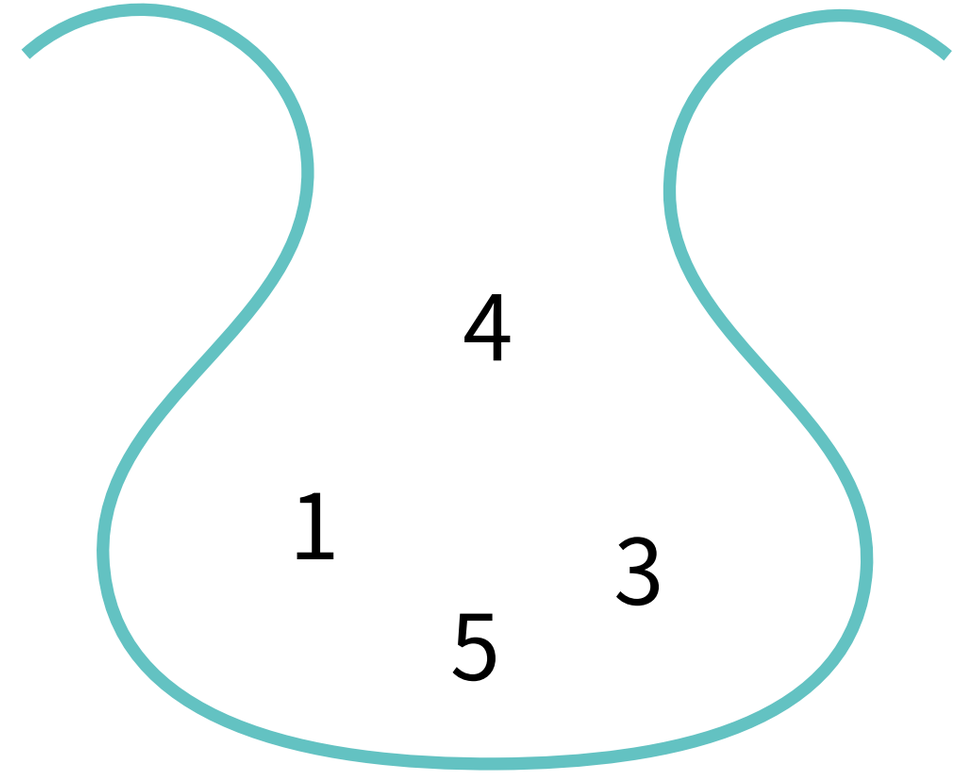
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

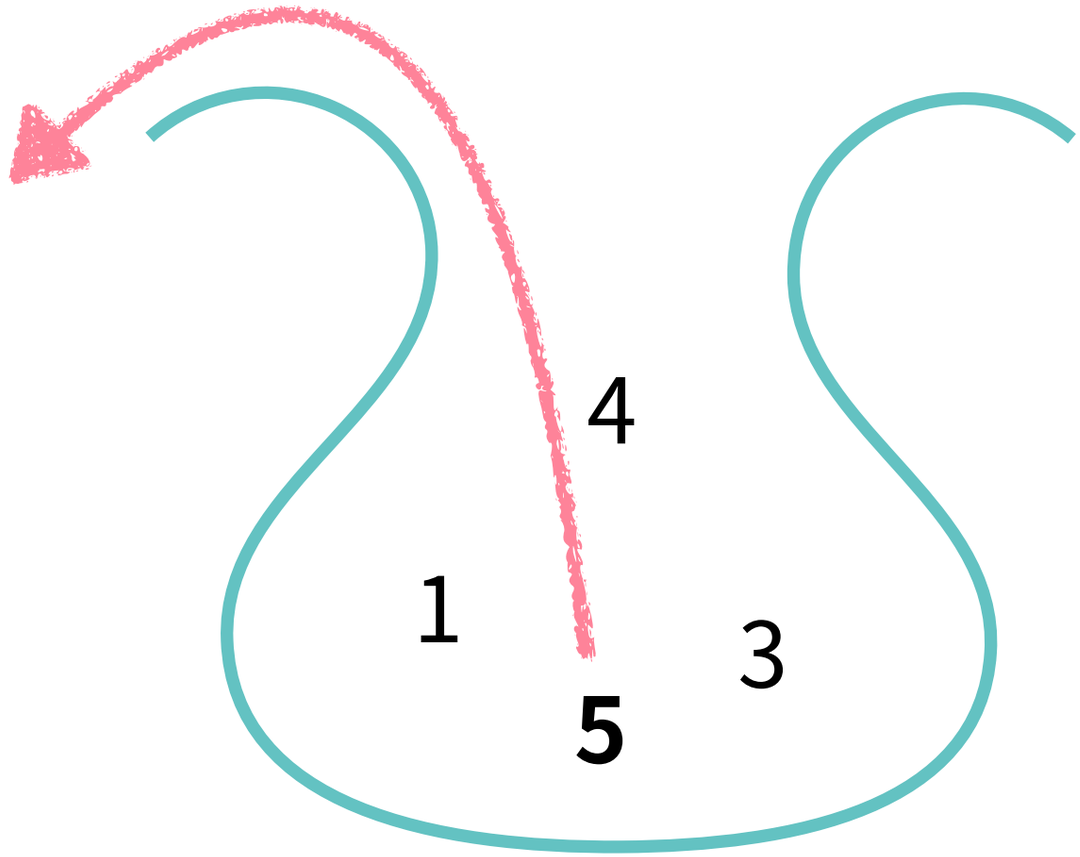
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

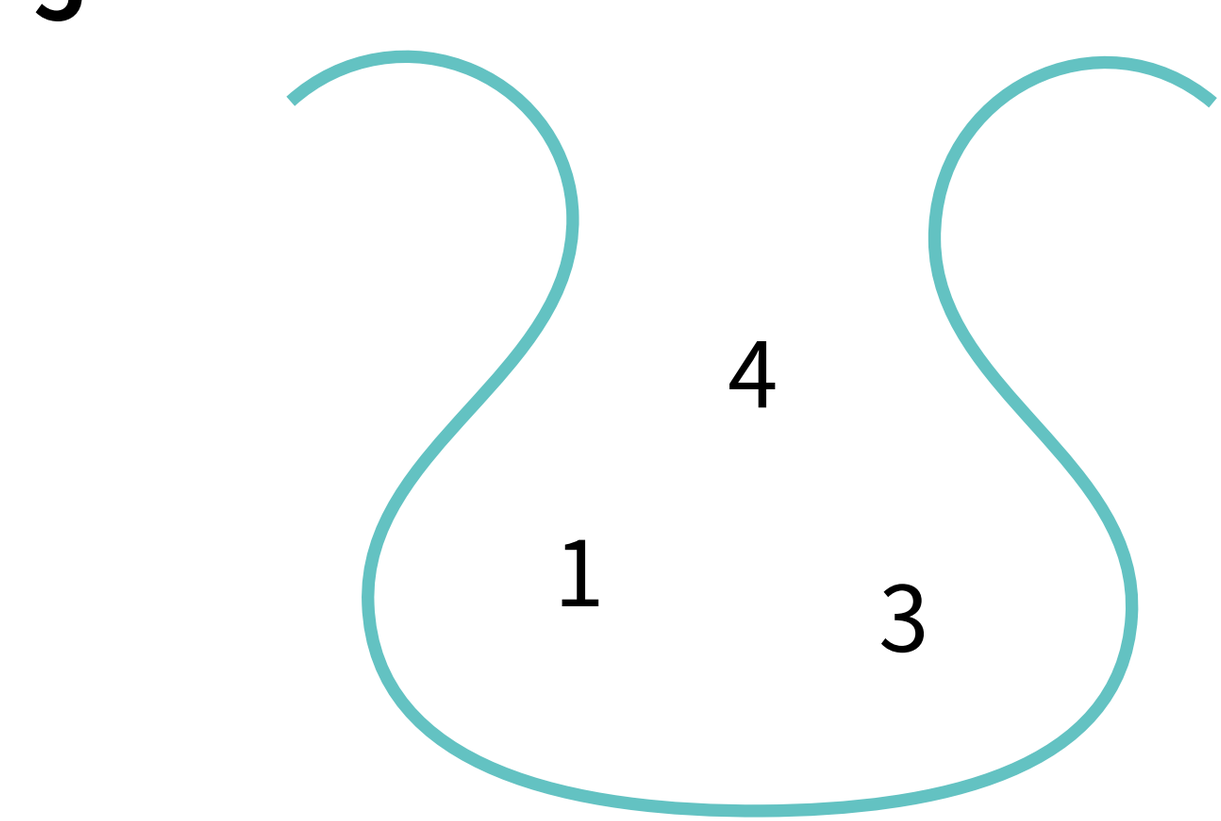
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

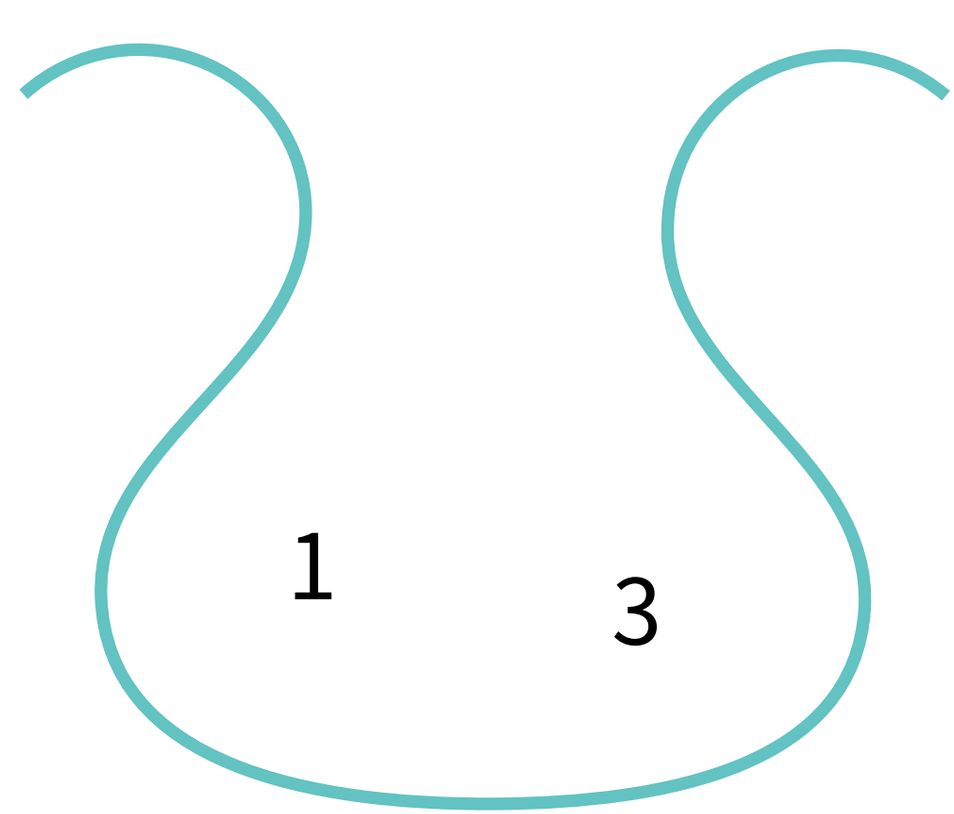
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

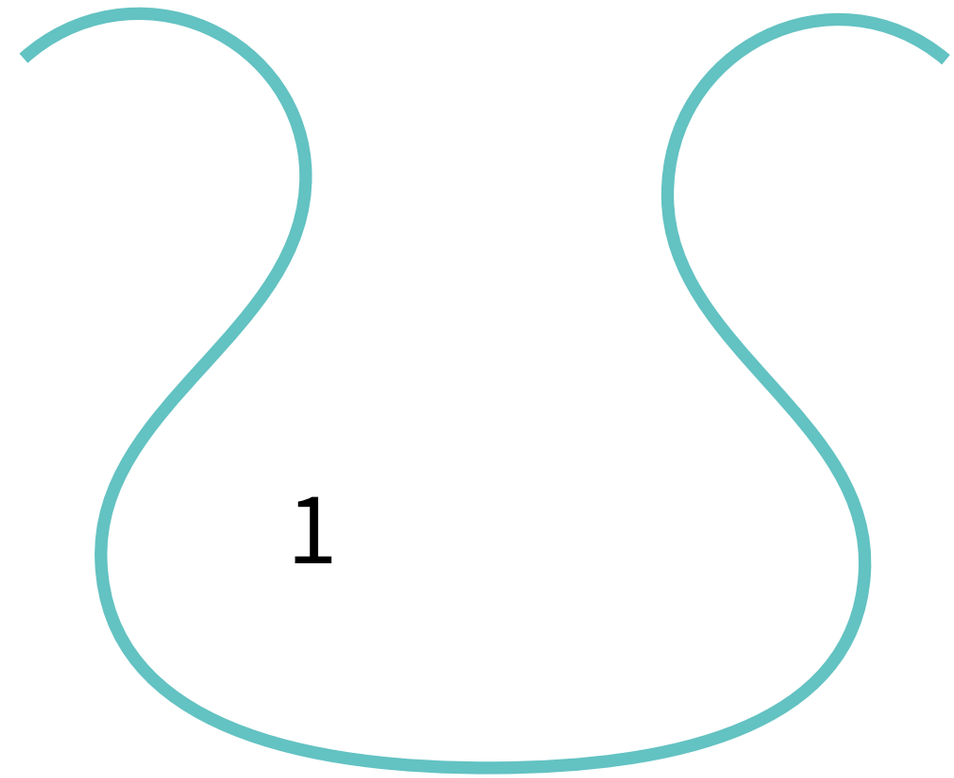


우선순위 큐

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5 4 3

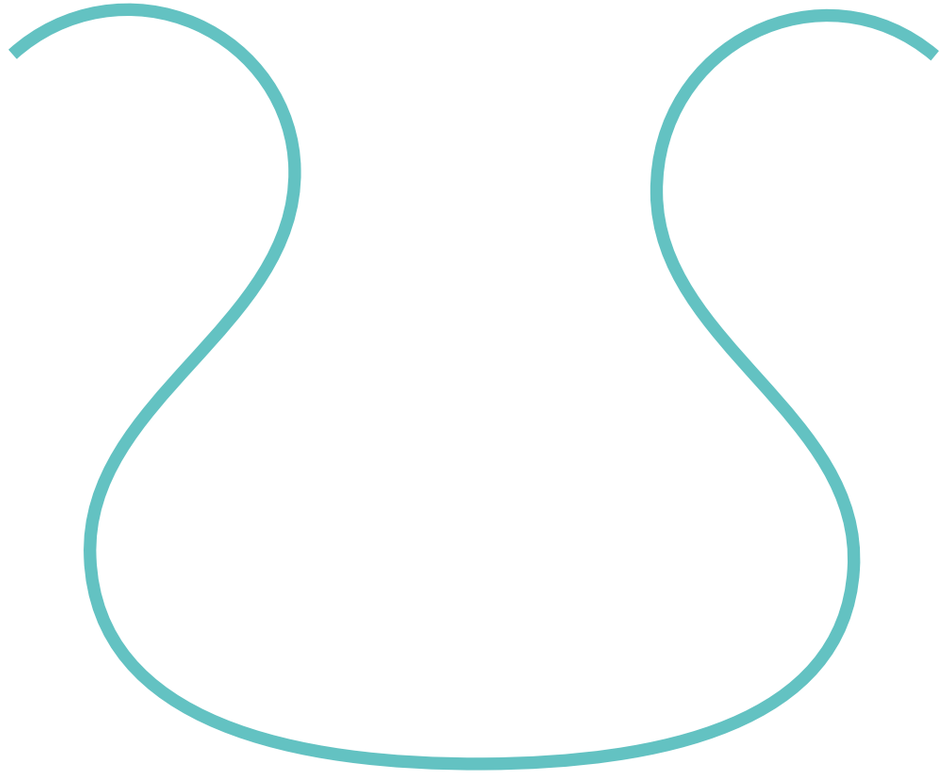


우선순위 큐

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

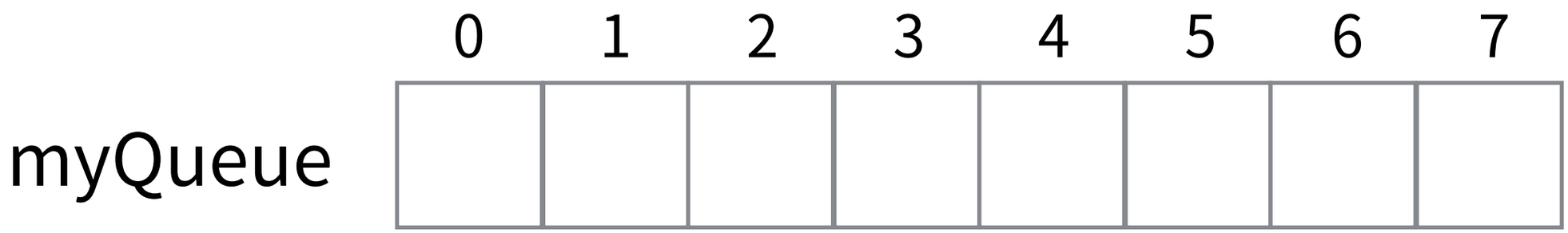
5 4 3 1



우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

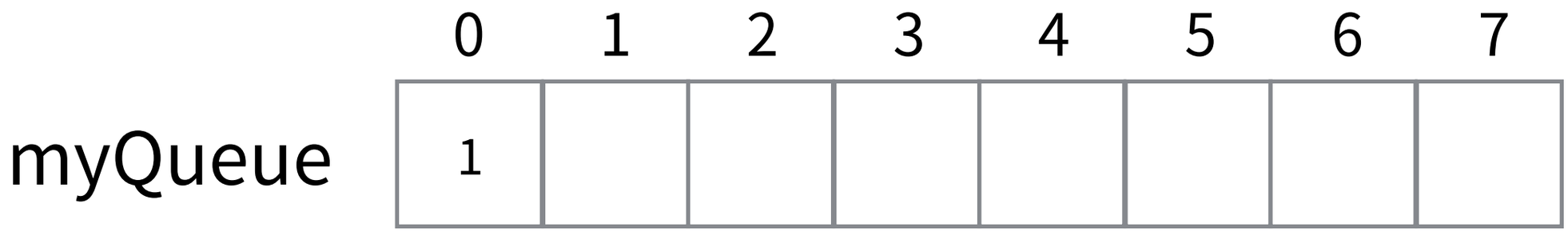
가장 우선순위가 높은 원소를 제거



우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

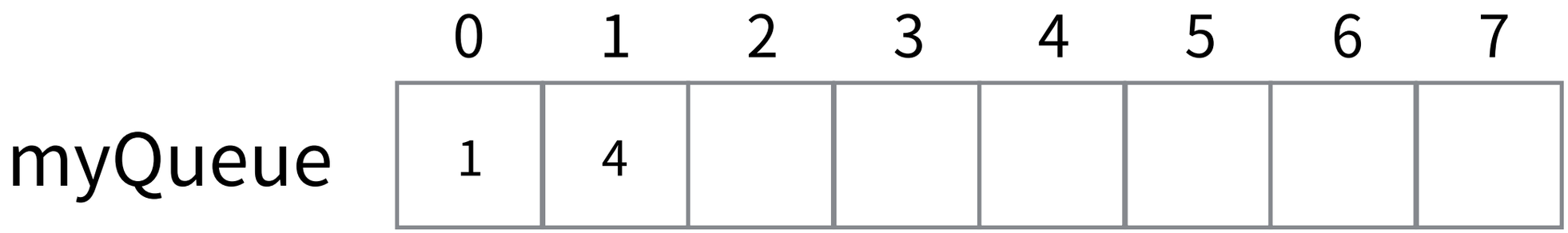
가장 우선순위가 높은 원소를 제거



우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

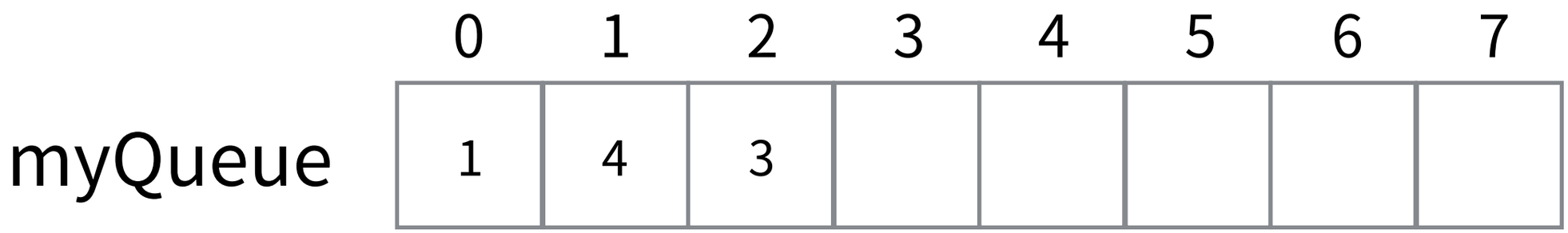
가장 우선순위가 높은 원소를 제거



우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

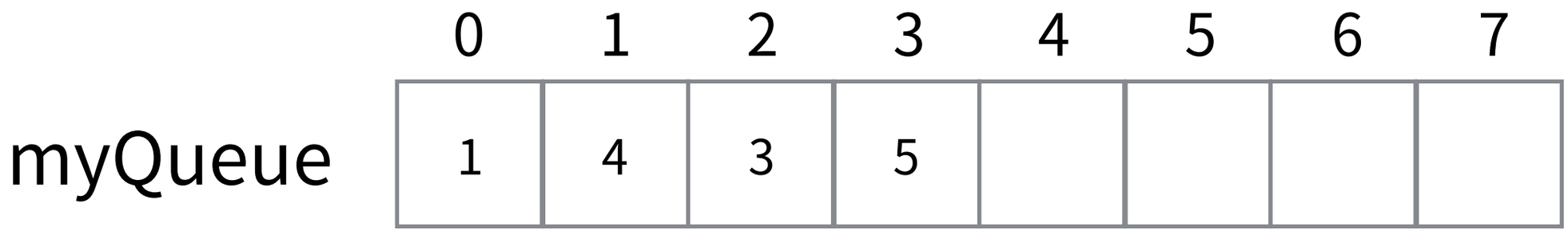
가장 우선순위가 높은 원소를 제거



우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

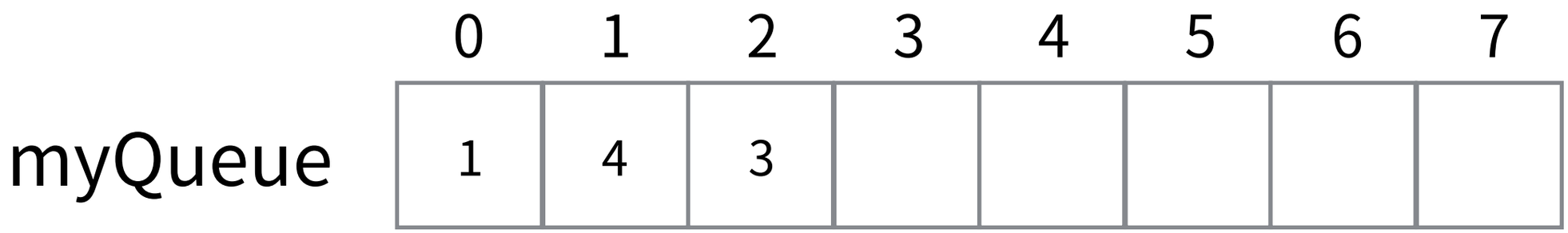


우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5

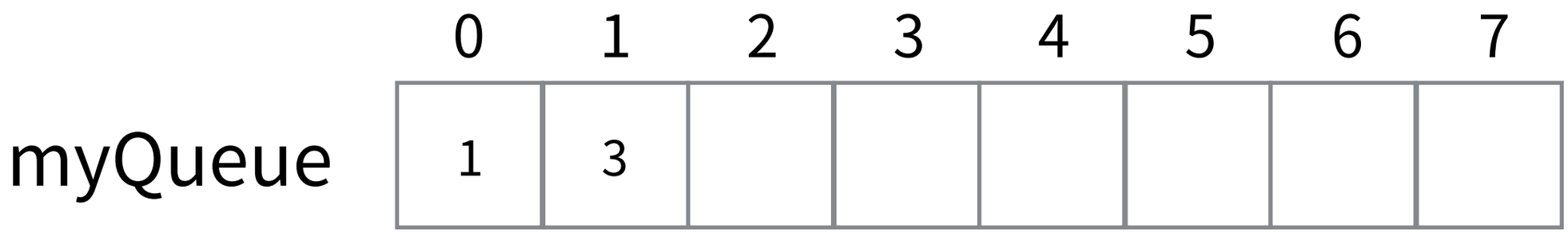


우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5 4

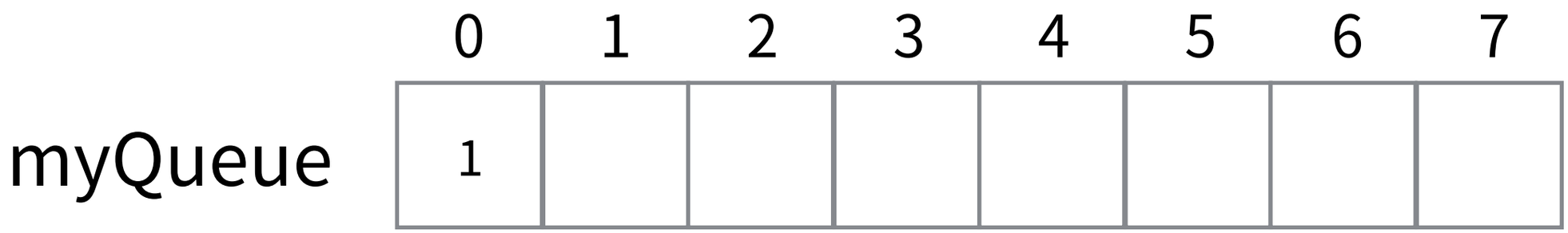


우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5 4 3

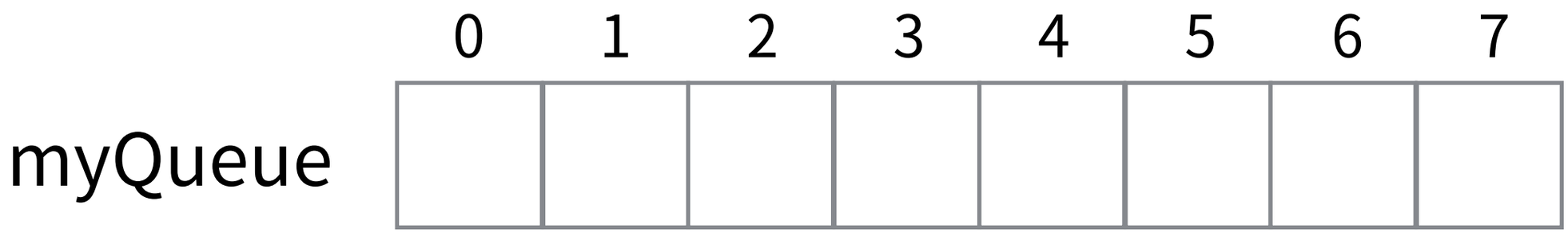


우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5 4 3 1

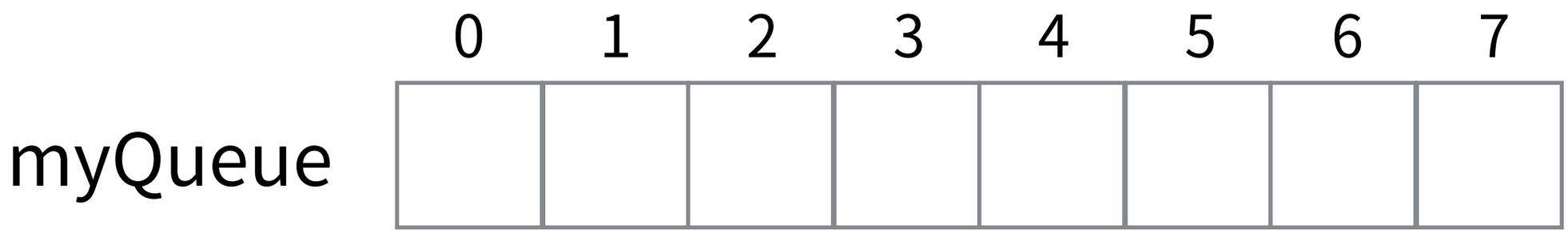


우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5 4 3 1　　삽입 : O(1)



우선순위 큐 : 배열로 구현하기

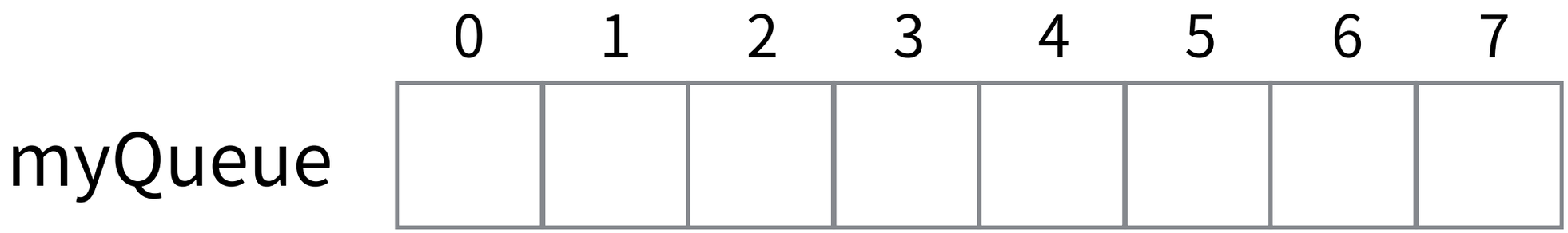
원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5 4 3 1

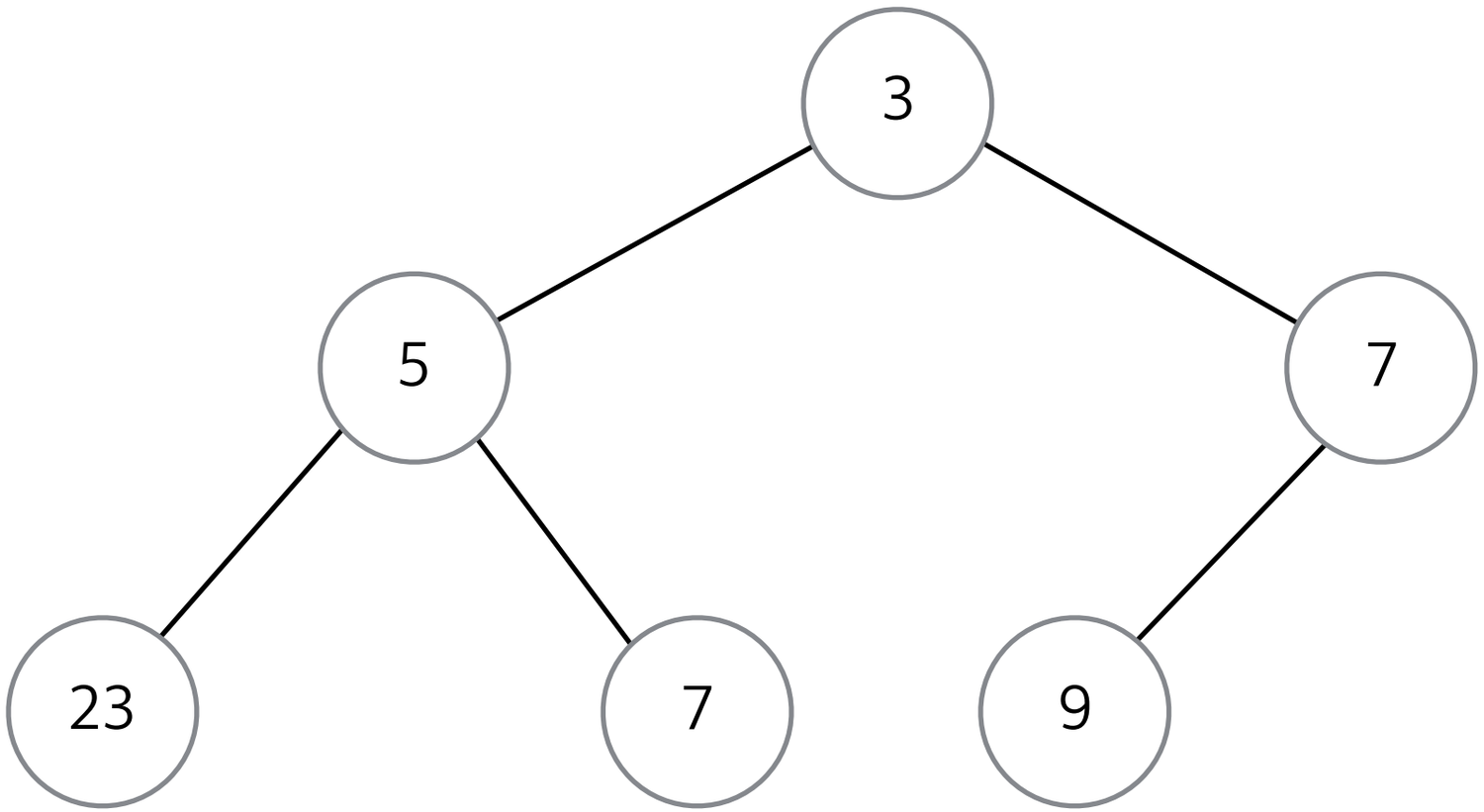
삽입 : O(1)

삭제 : O(n)



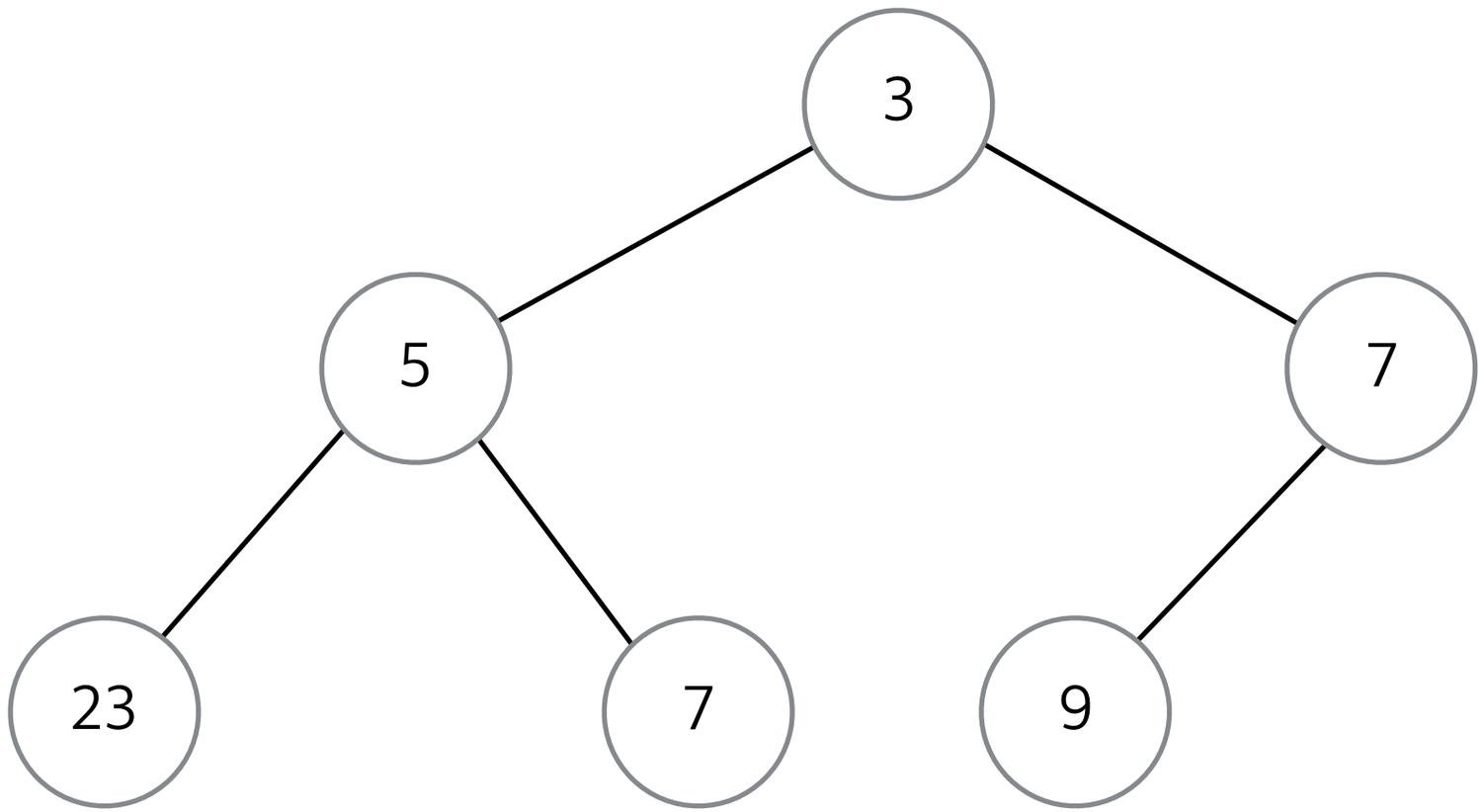
힙

부모의 값이 항상 자식보다 작은 완전 이진 트리



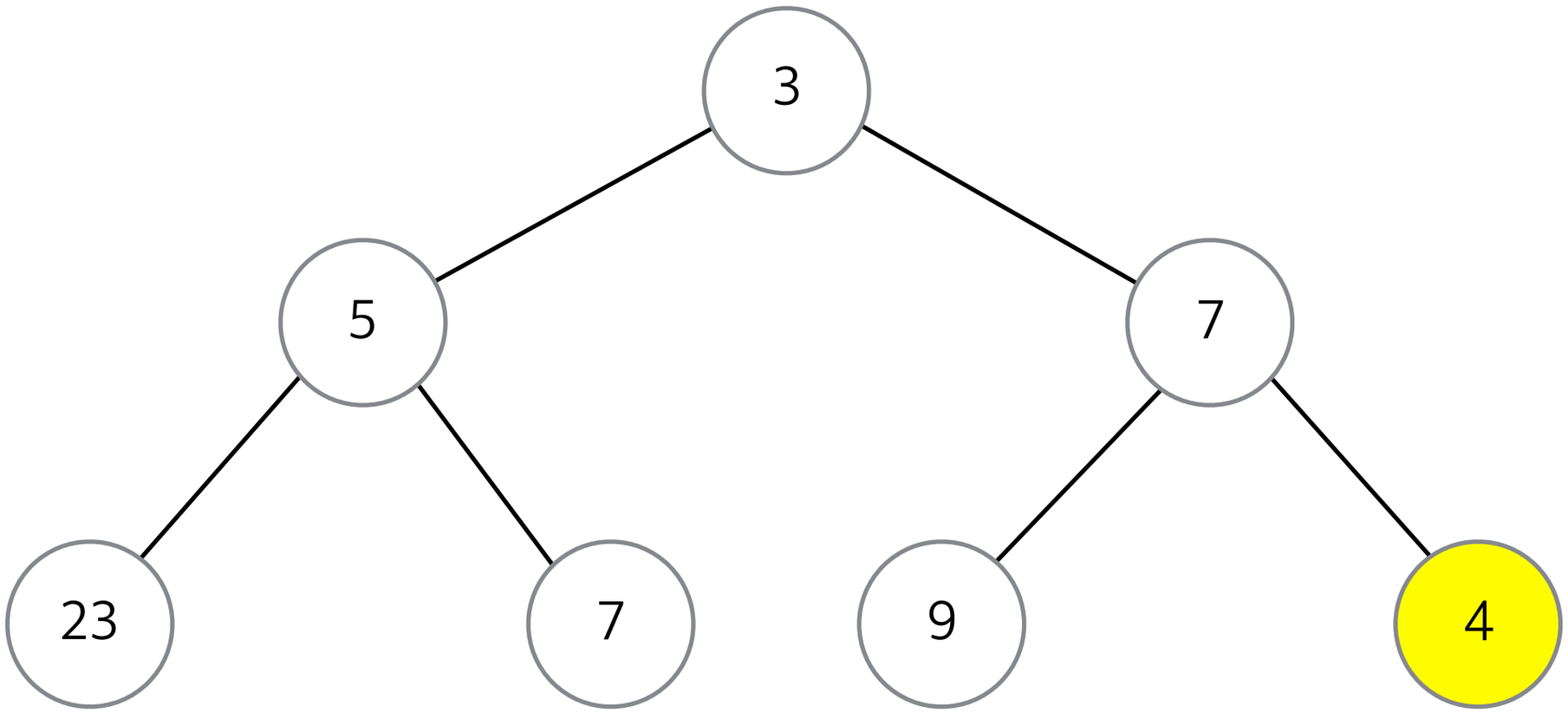
힙 : 값 삽입

heap.insert(4)



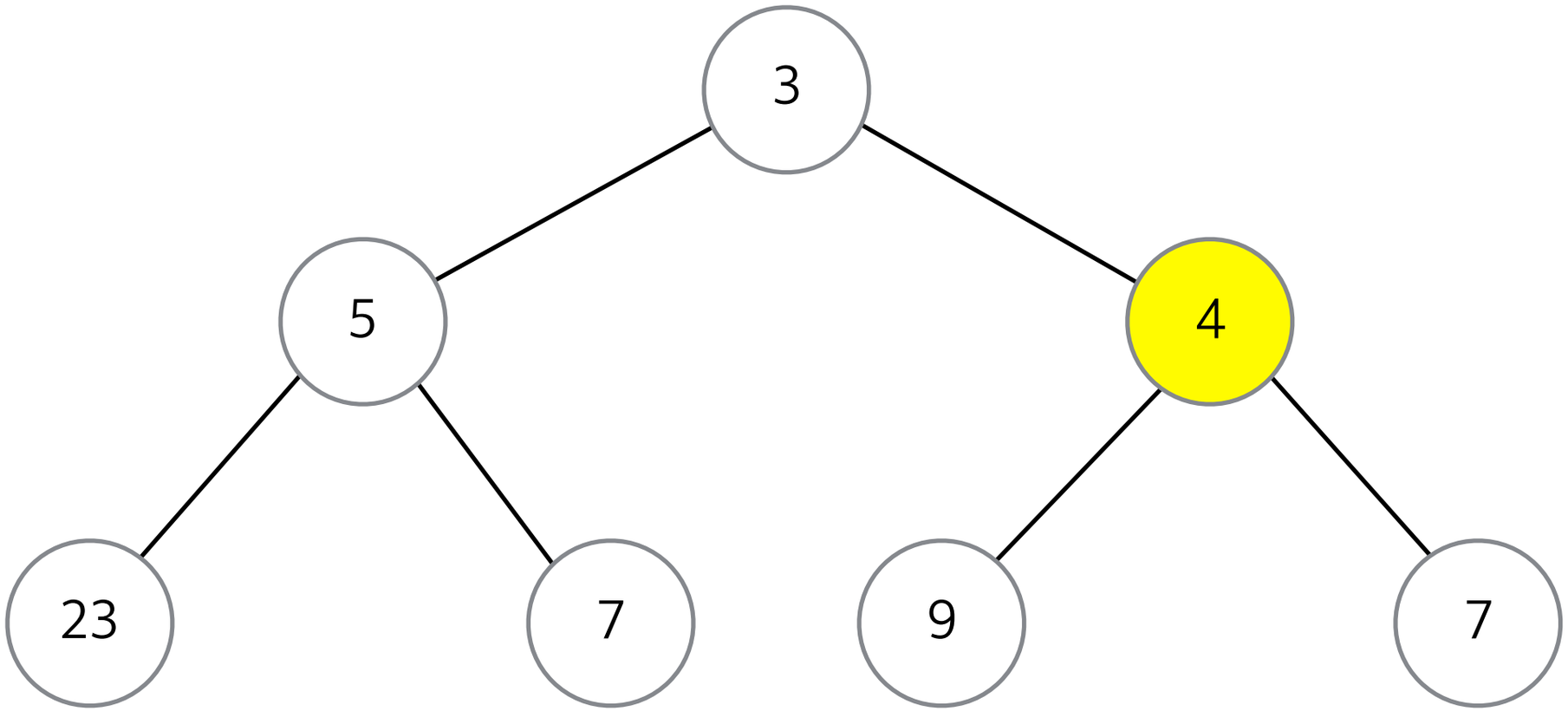
힙 : 값 삽입

heap.insert(4)



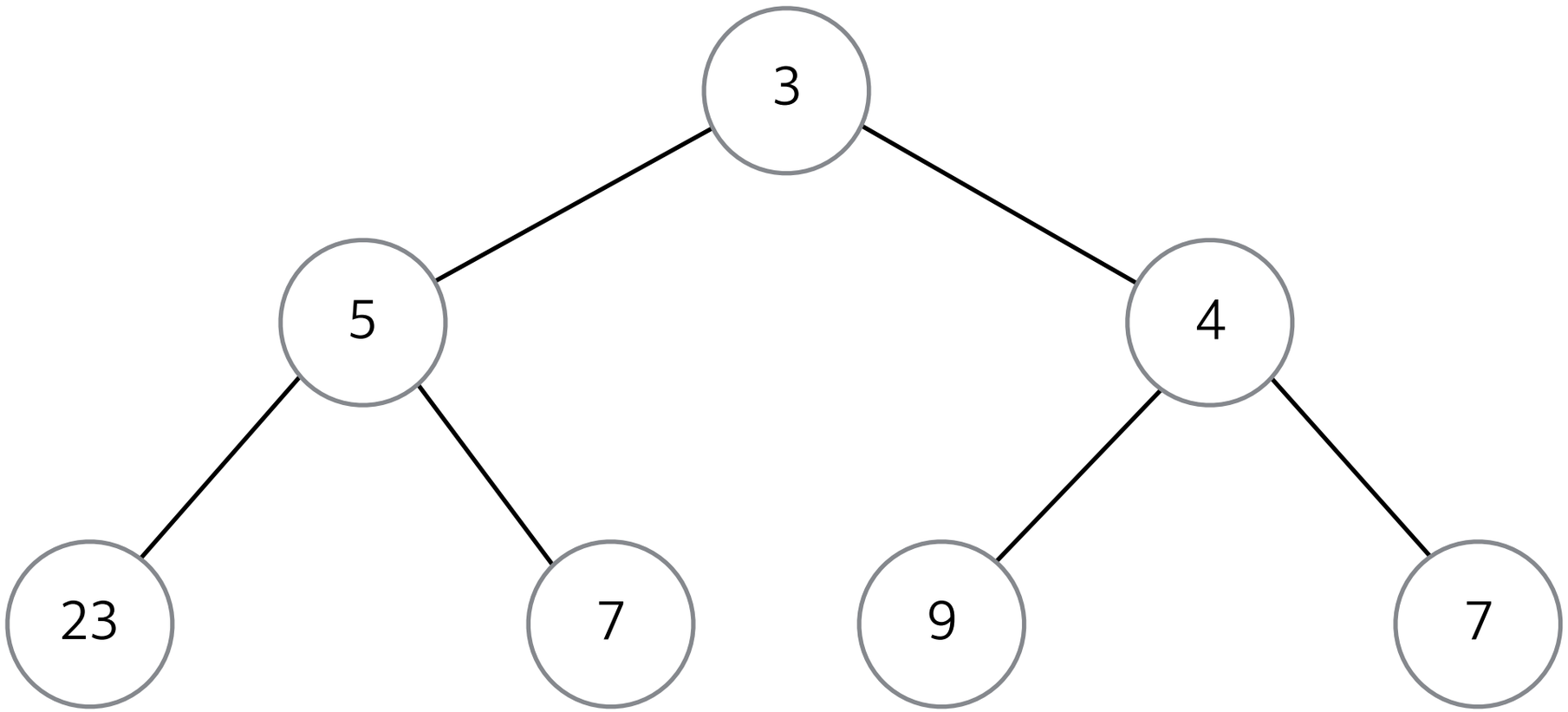
힙 : 값 삽입

heap.insert(4)



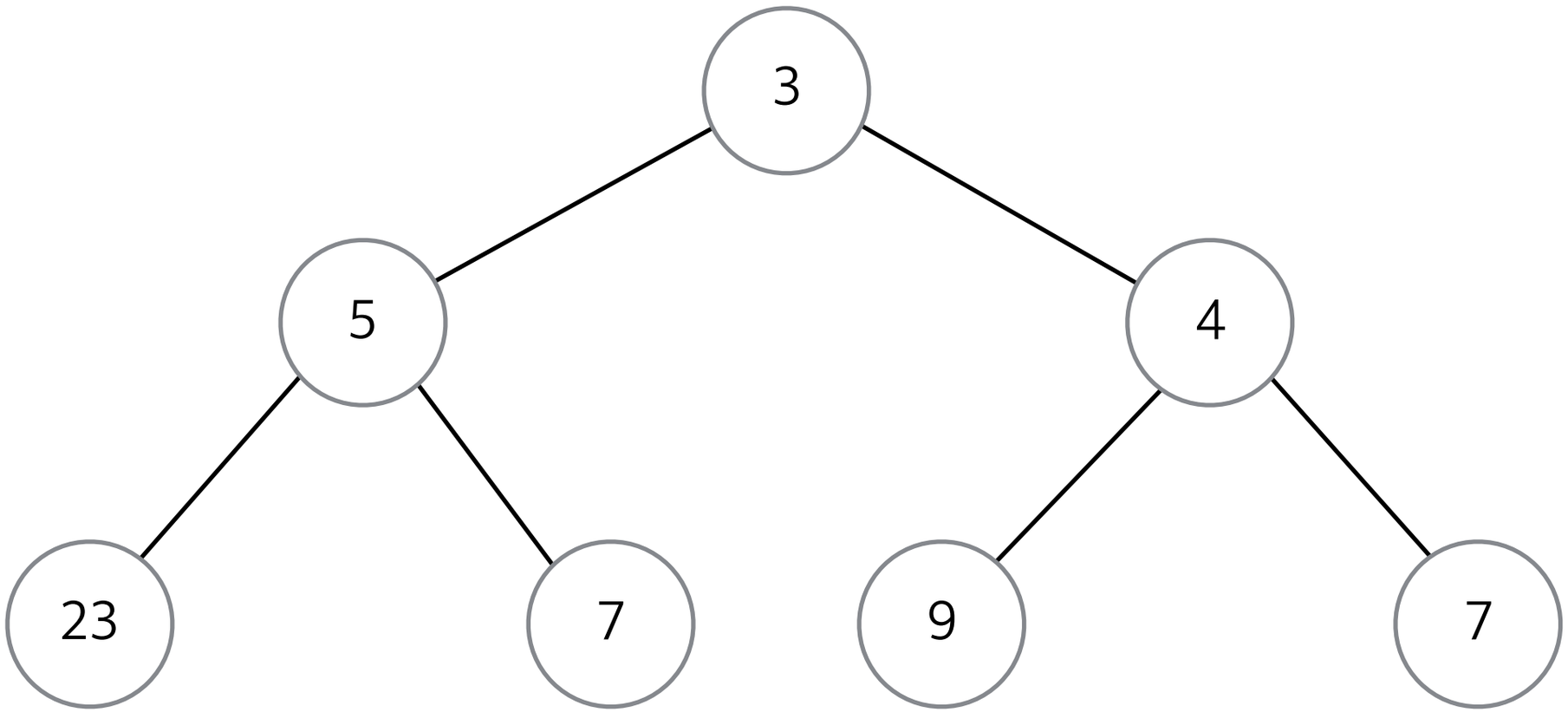
힙 : 값 삽입

heap.insert(4)



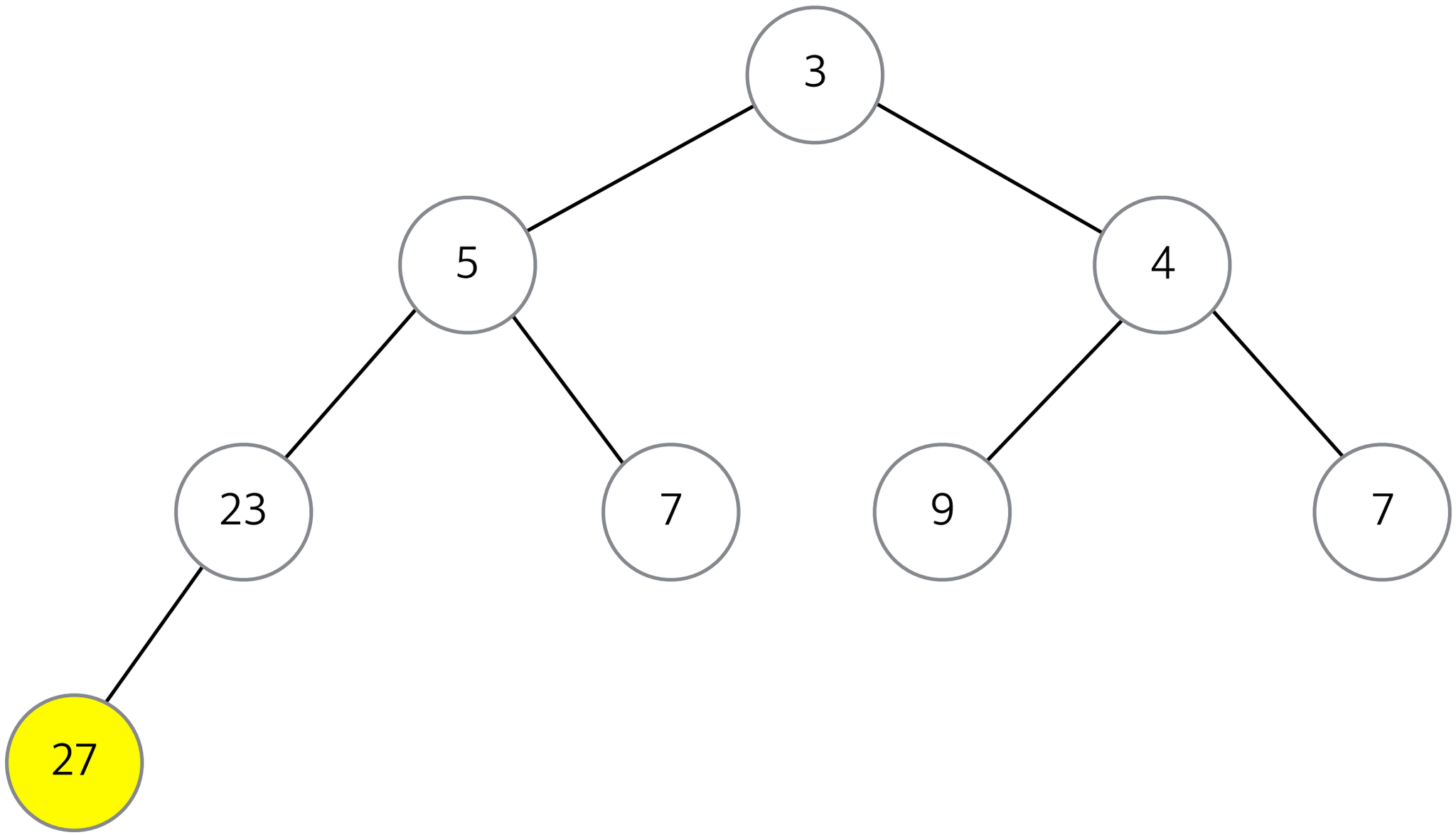
힙 : 값 삽입

heap.insert(27)



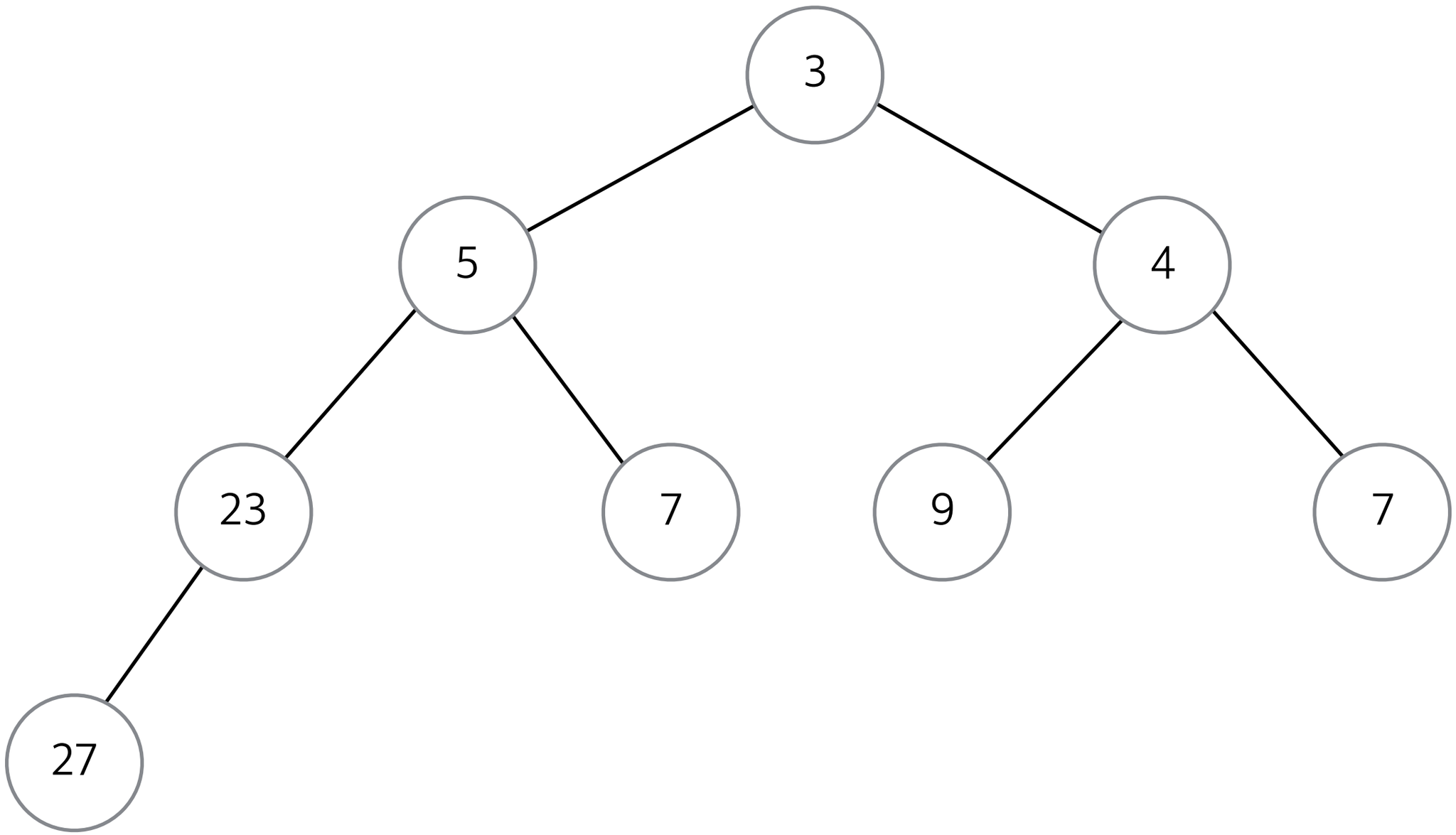
힙 : 값 삽입

heap.insert(27)



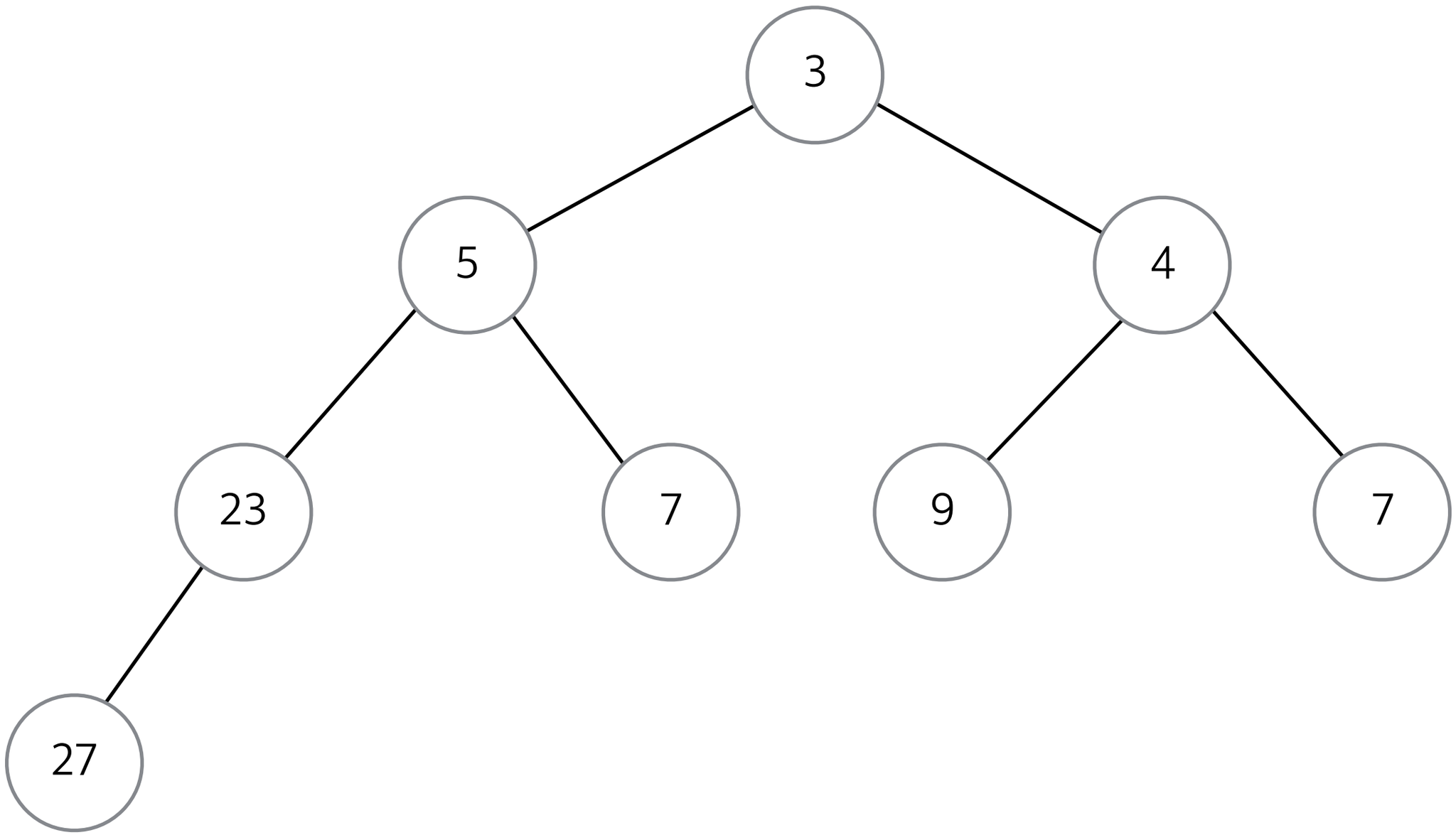
힙 : 값 삽입

heap.insert(27)



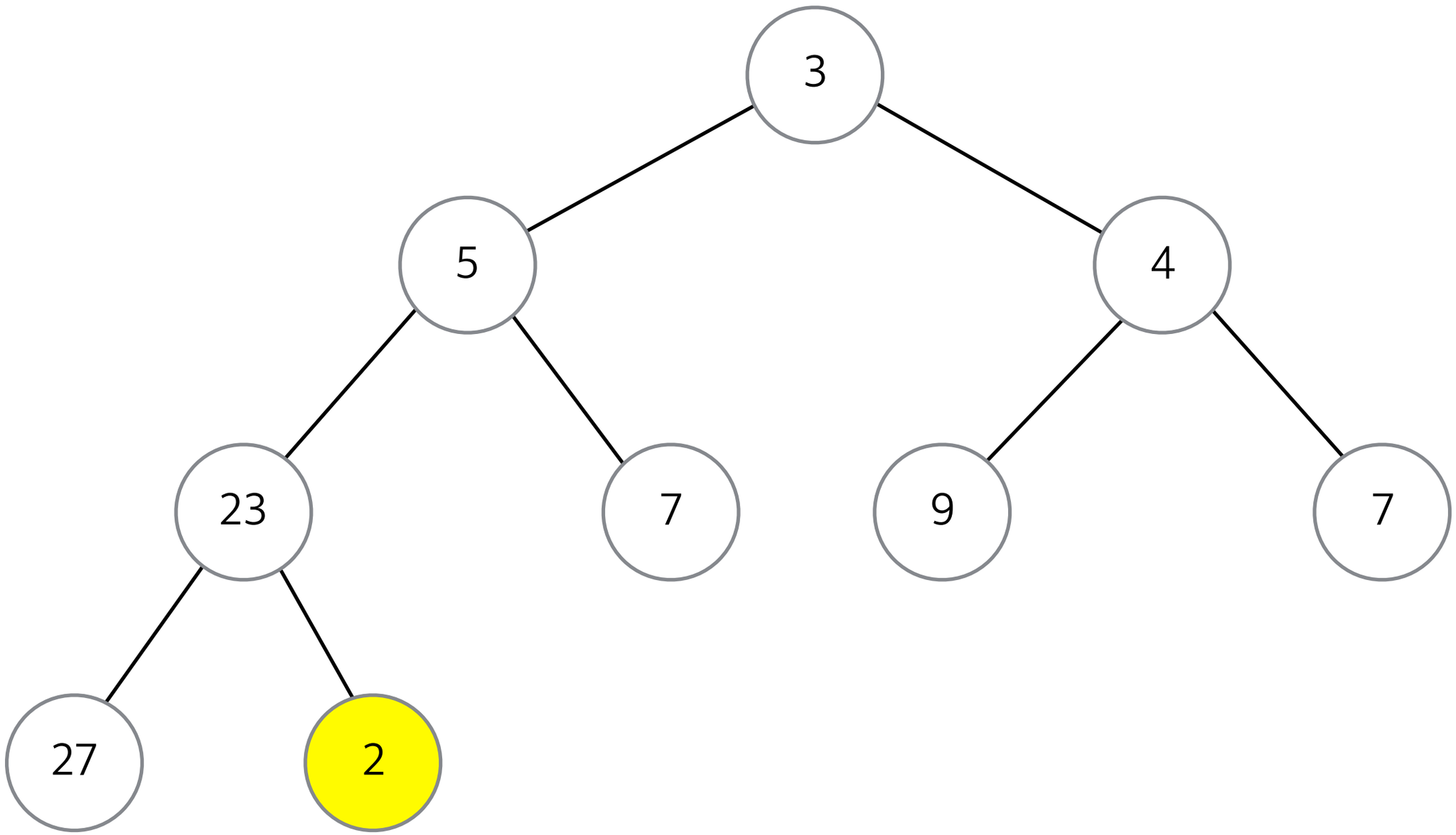
힙 : 값 삽입

heap.insert(2)



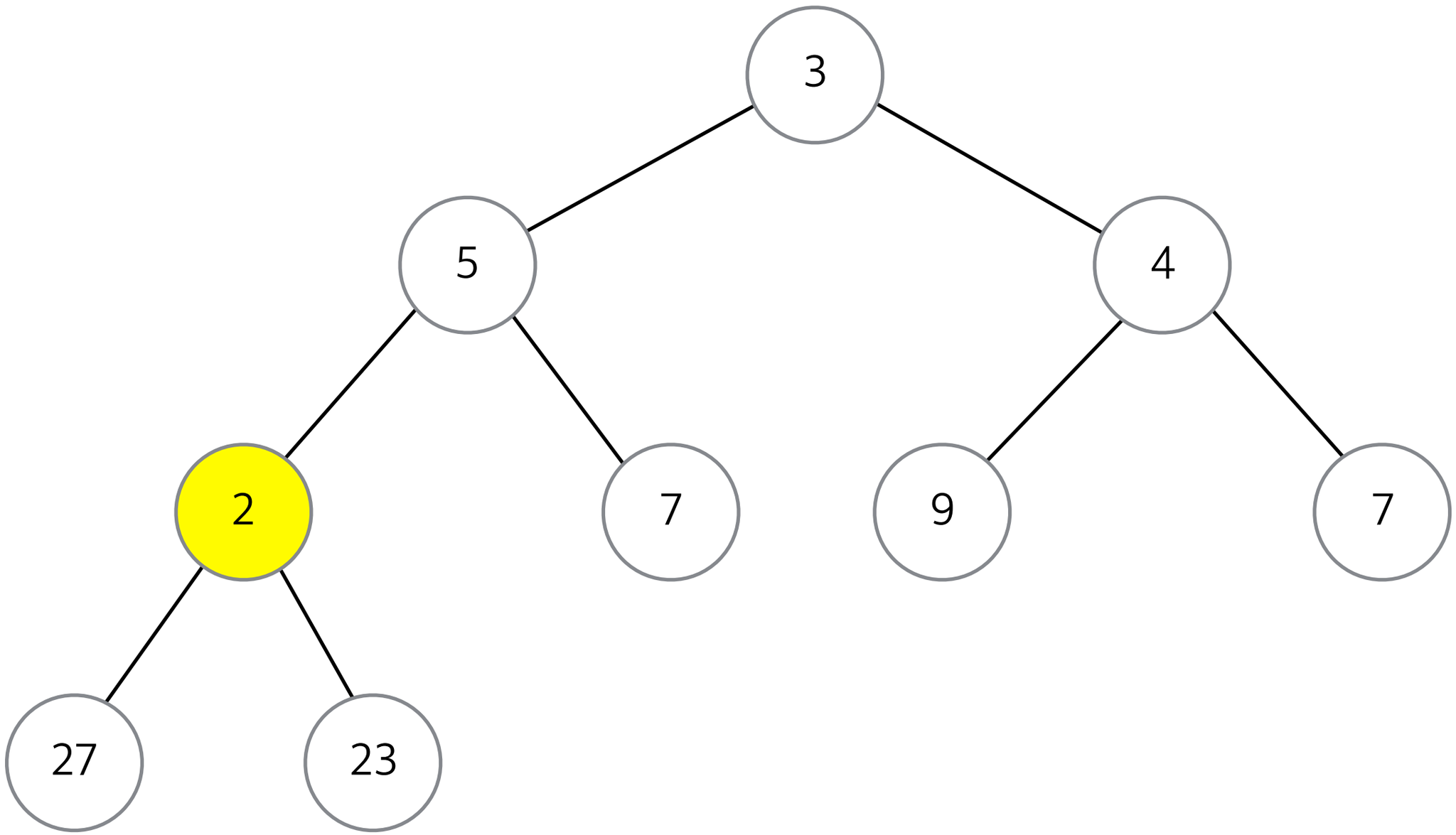
힙 : 값 삽입

heap.insert(2)



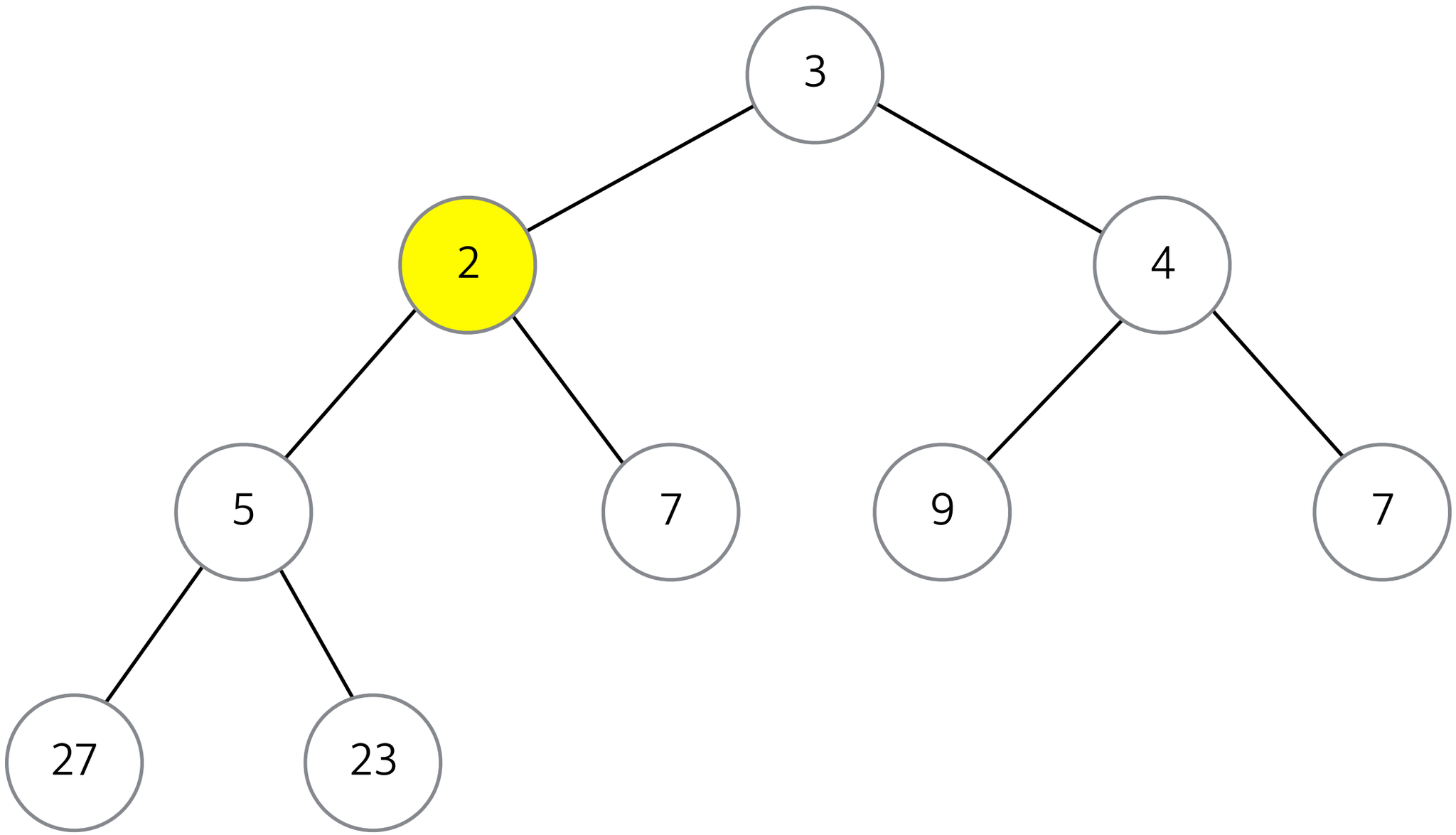
힙 : 값 삽입

heap.insert(2)



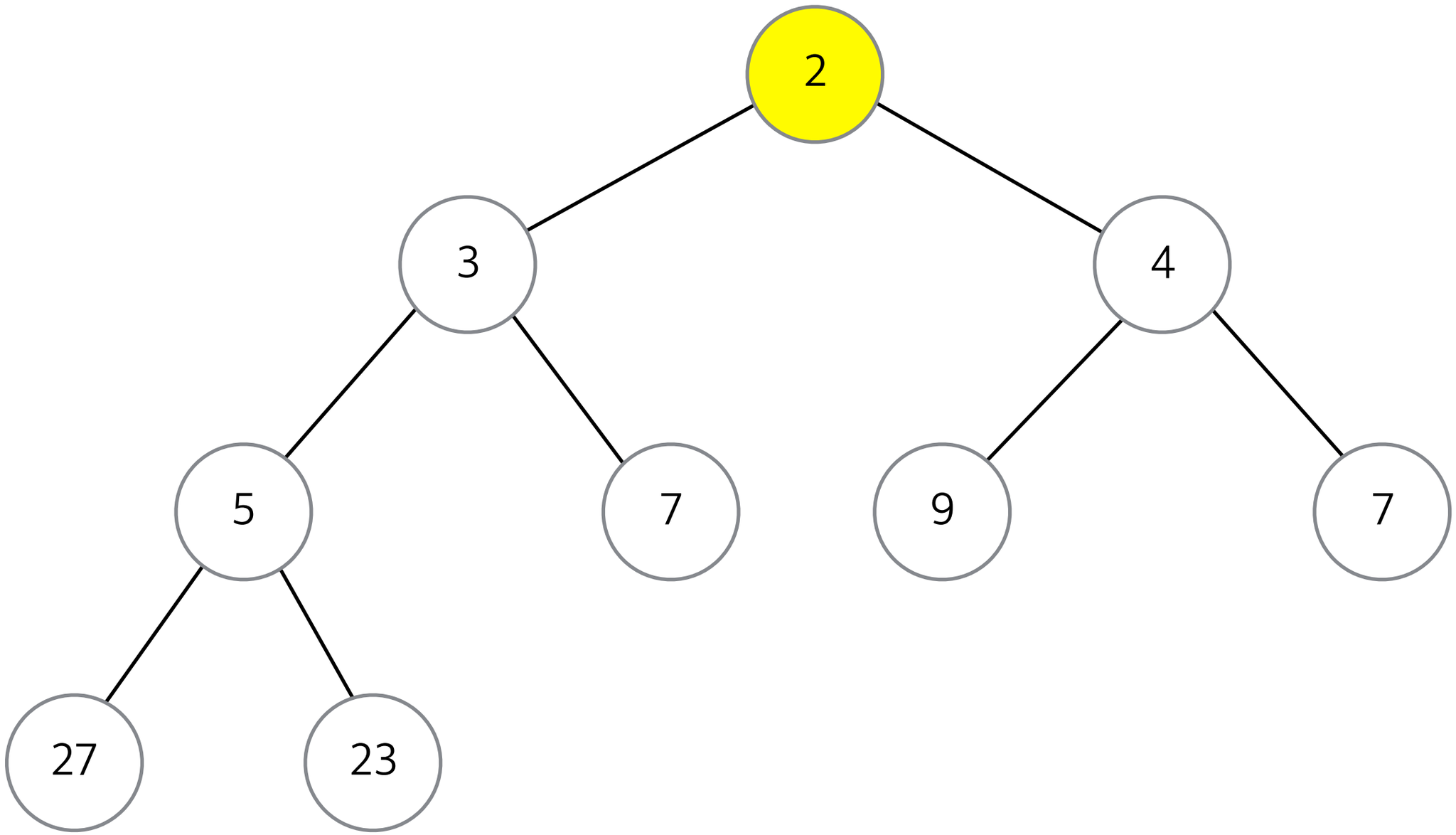
힙 : 값 삽입

heap.insert(2)



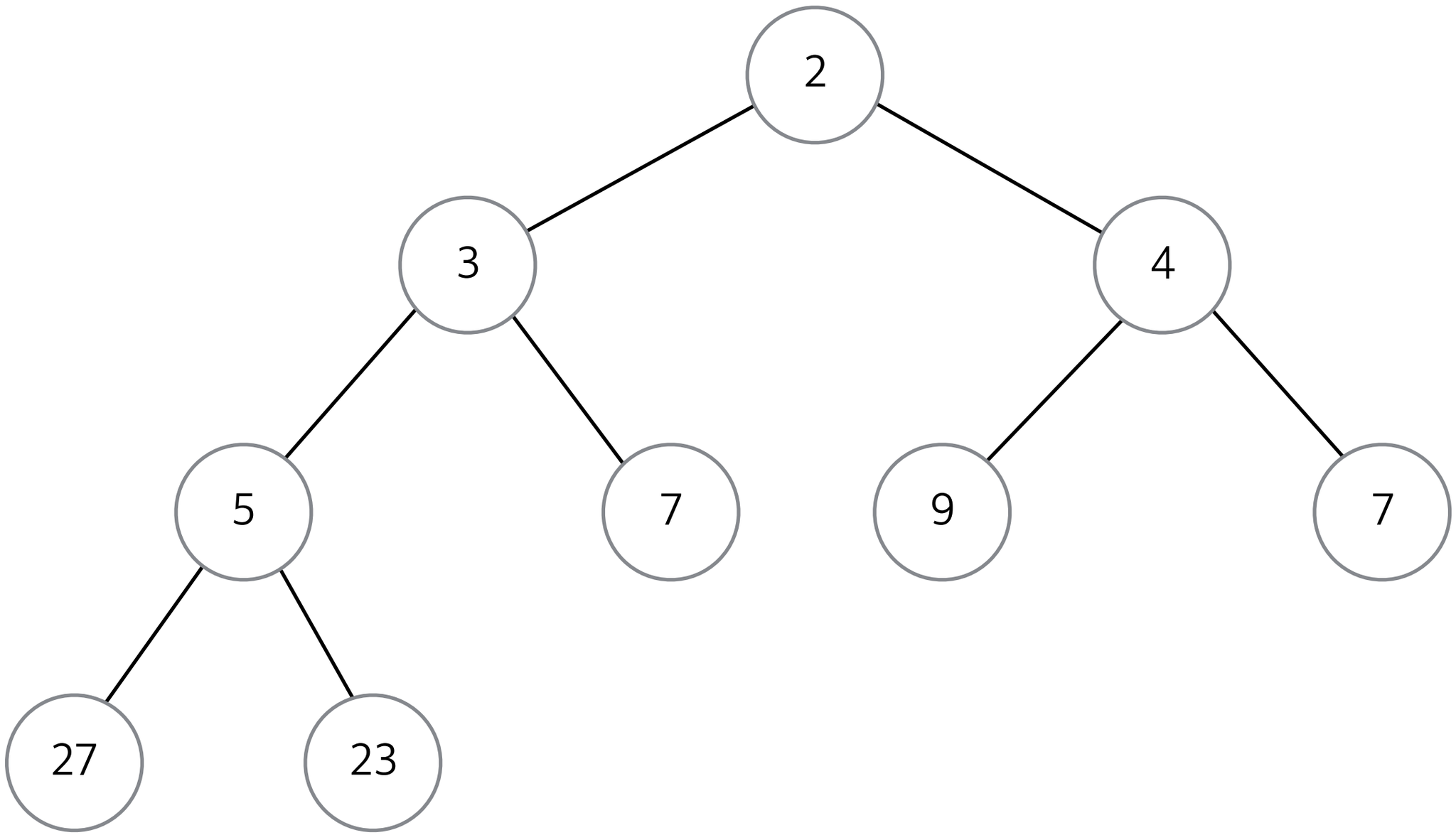
힙 : 값 삽입

heap.insert(2)

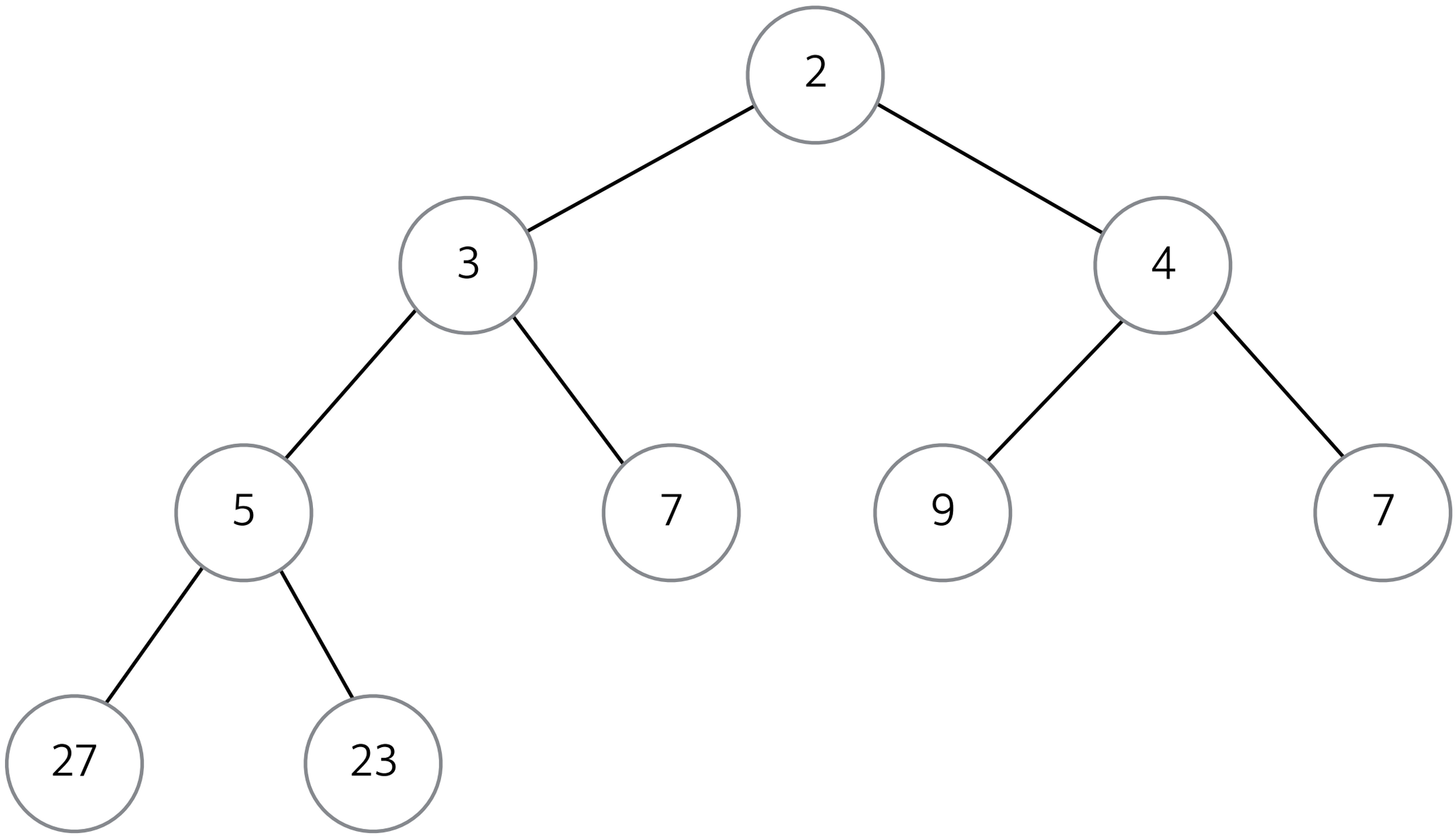


힙 : 값 삽입

heap.insert(2)

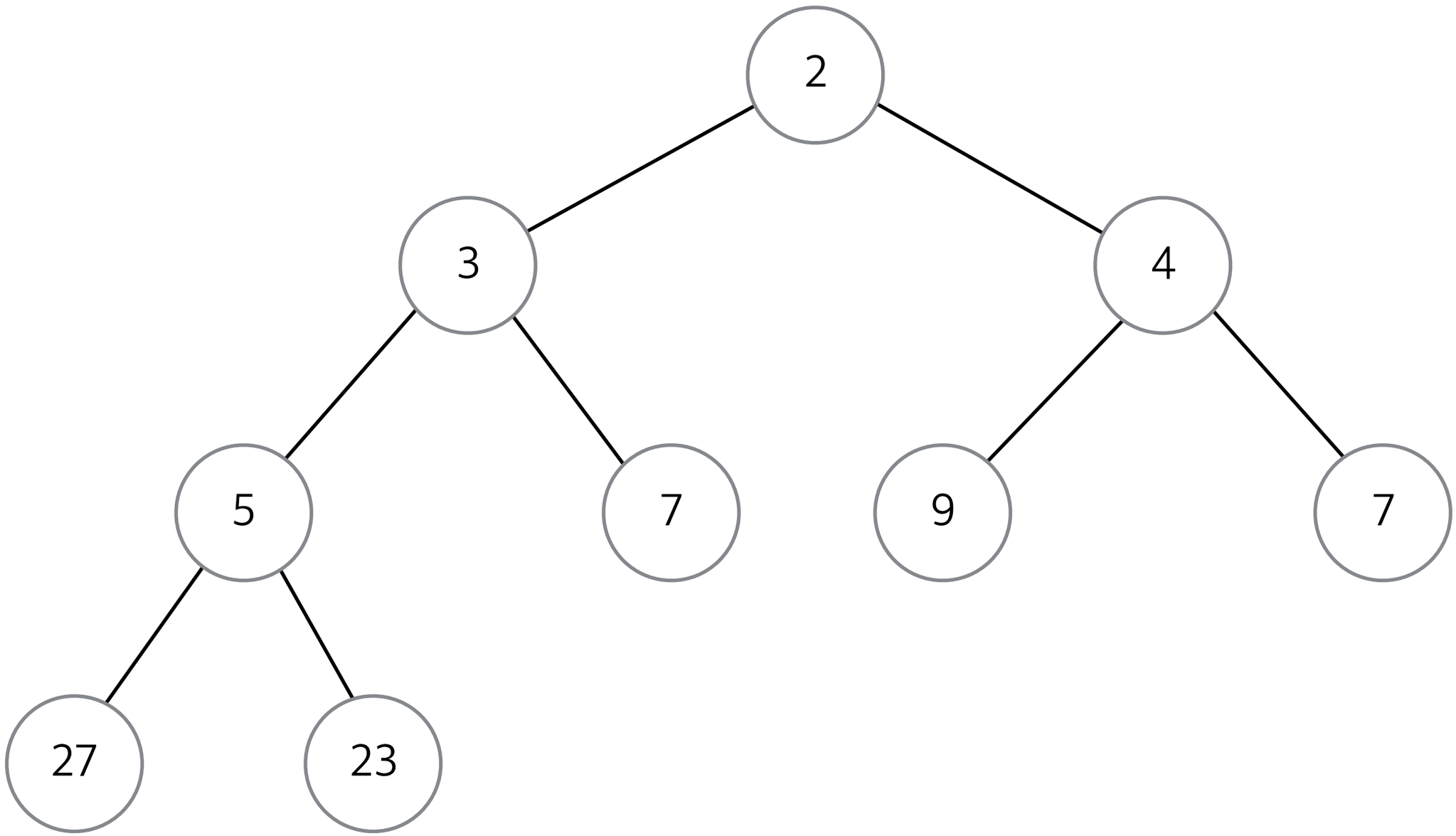


힙 : 값 삽입의 시간복잡도



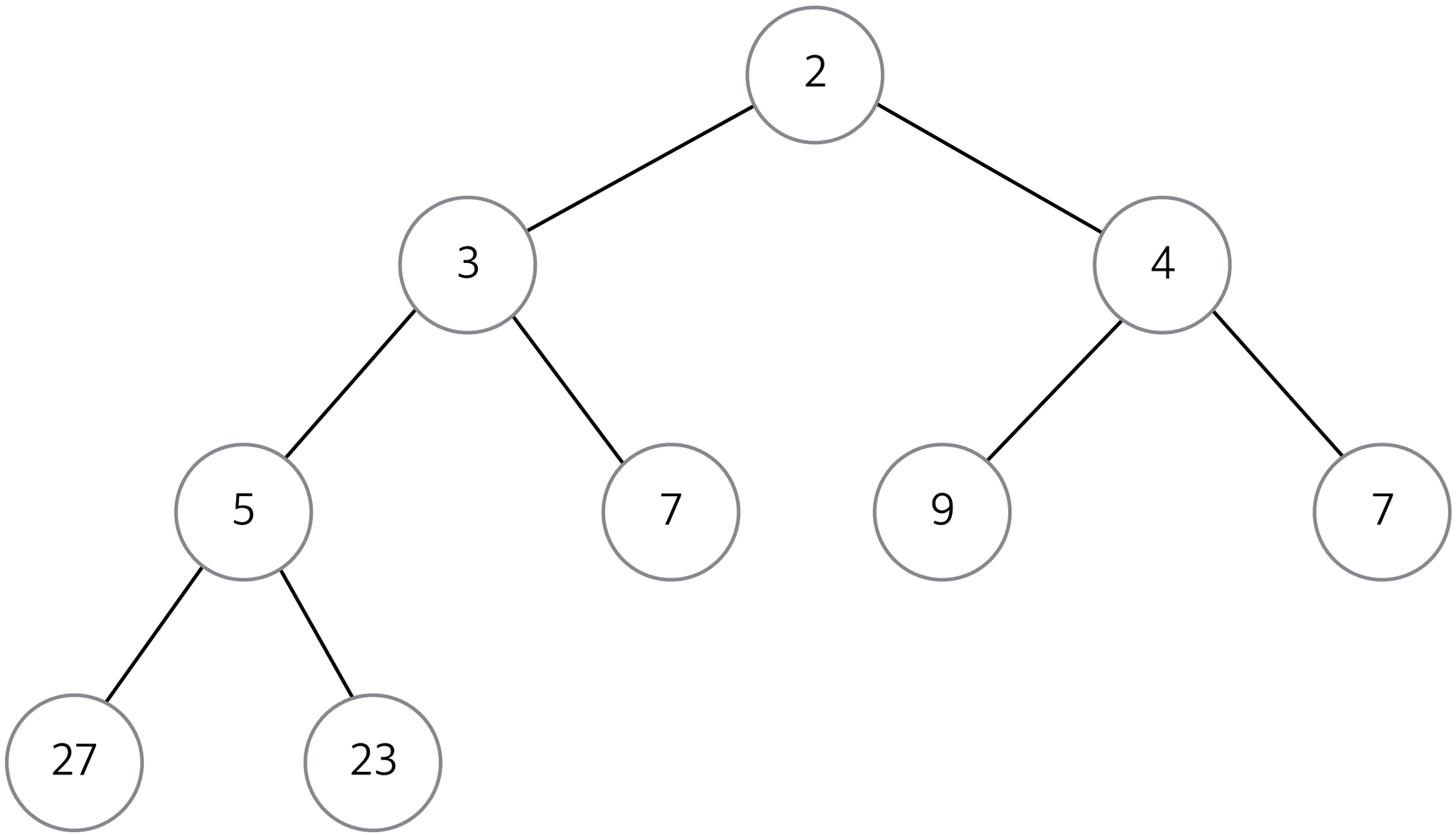
힙 : 값 삽입의 시간복잡도

O(log n)



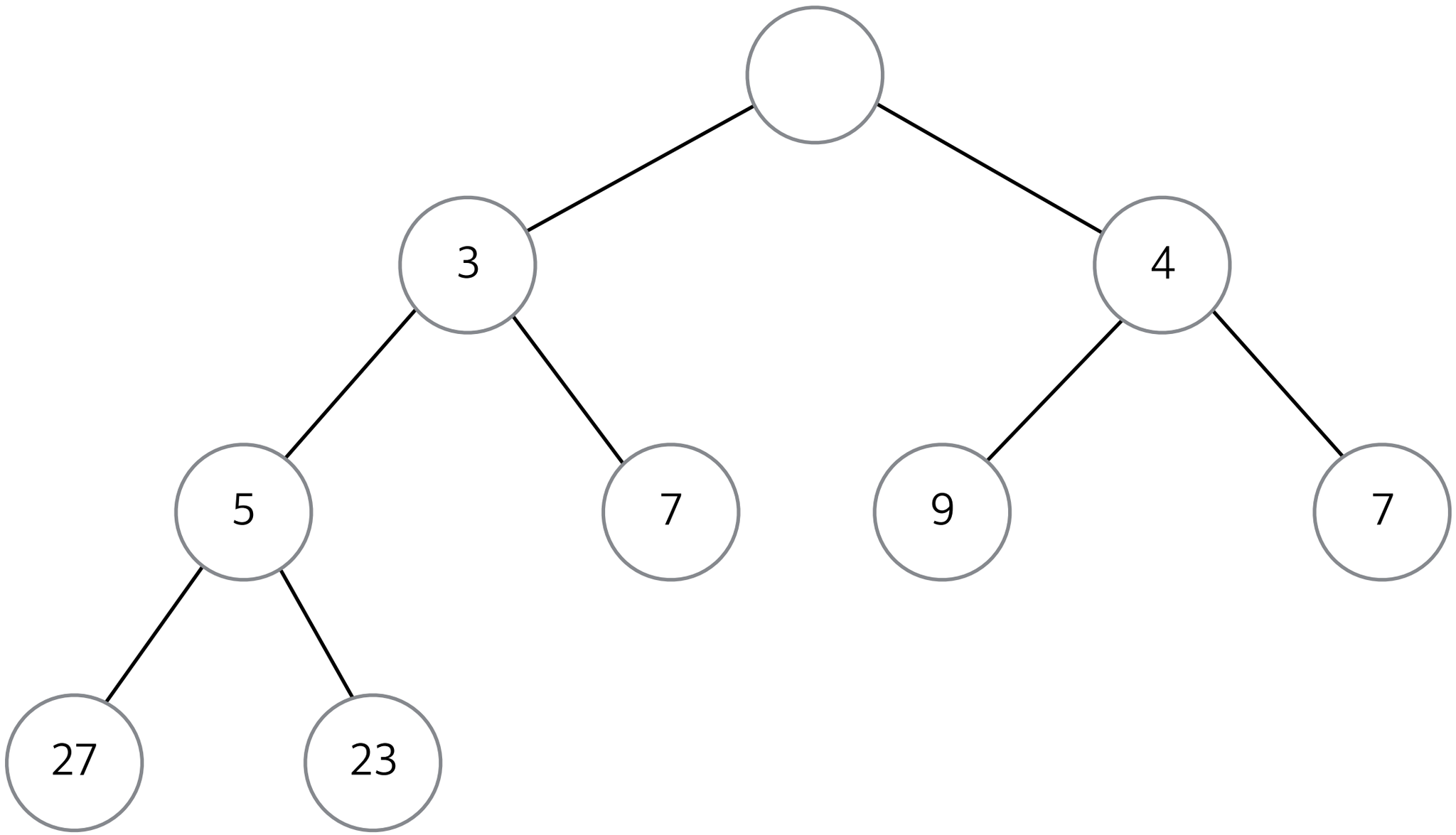
힙 : 값 삭제

heap.pop()



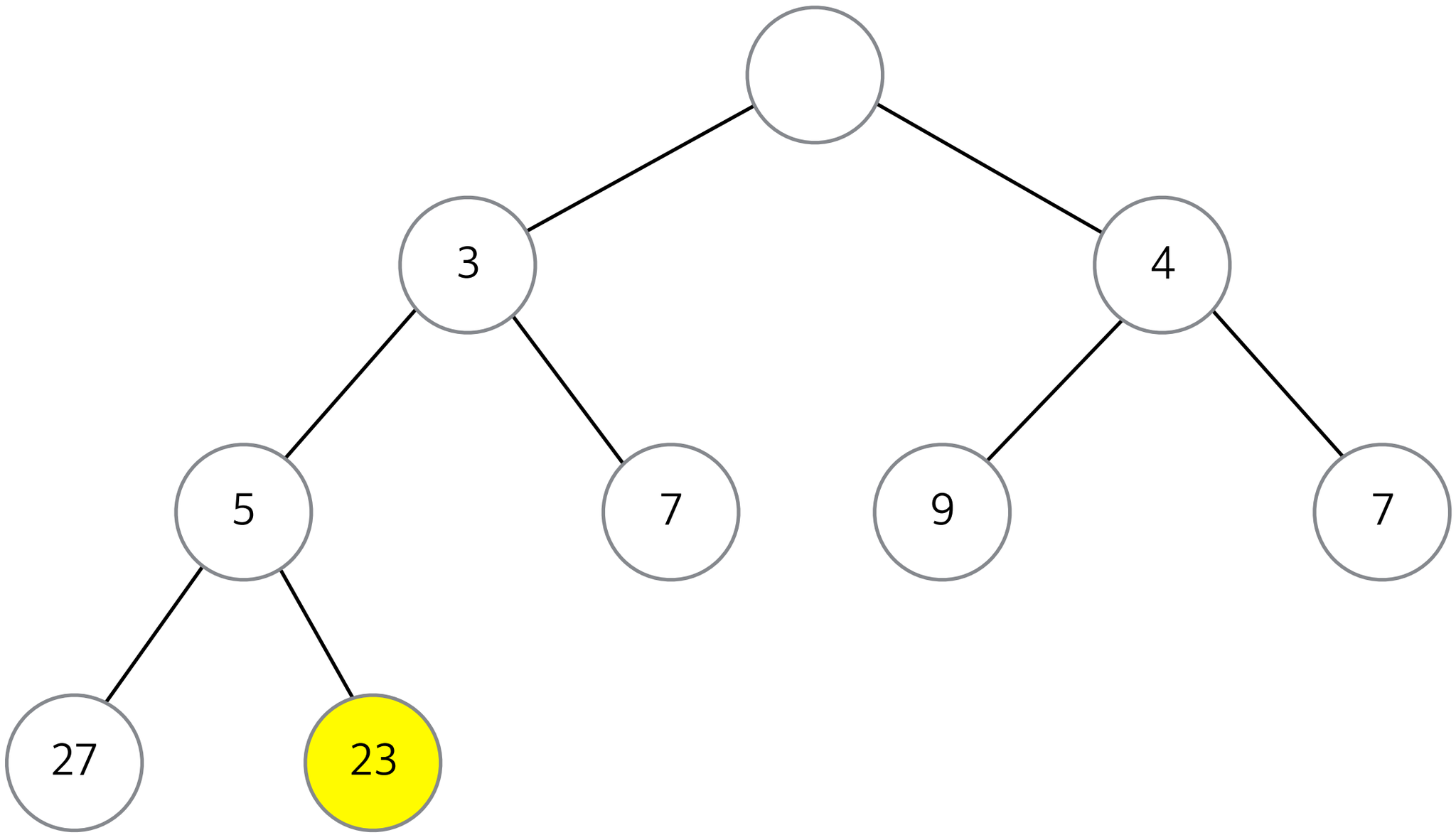
힙 : 값 삭제

heap.pop()



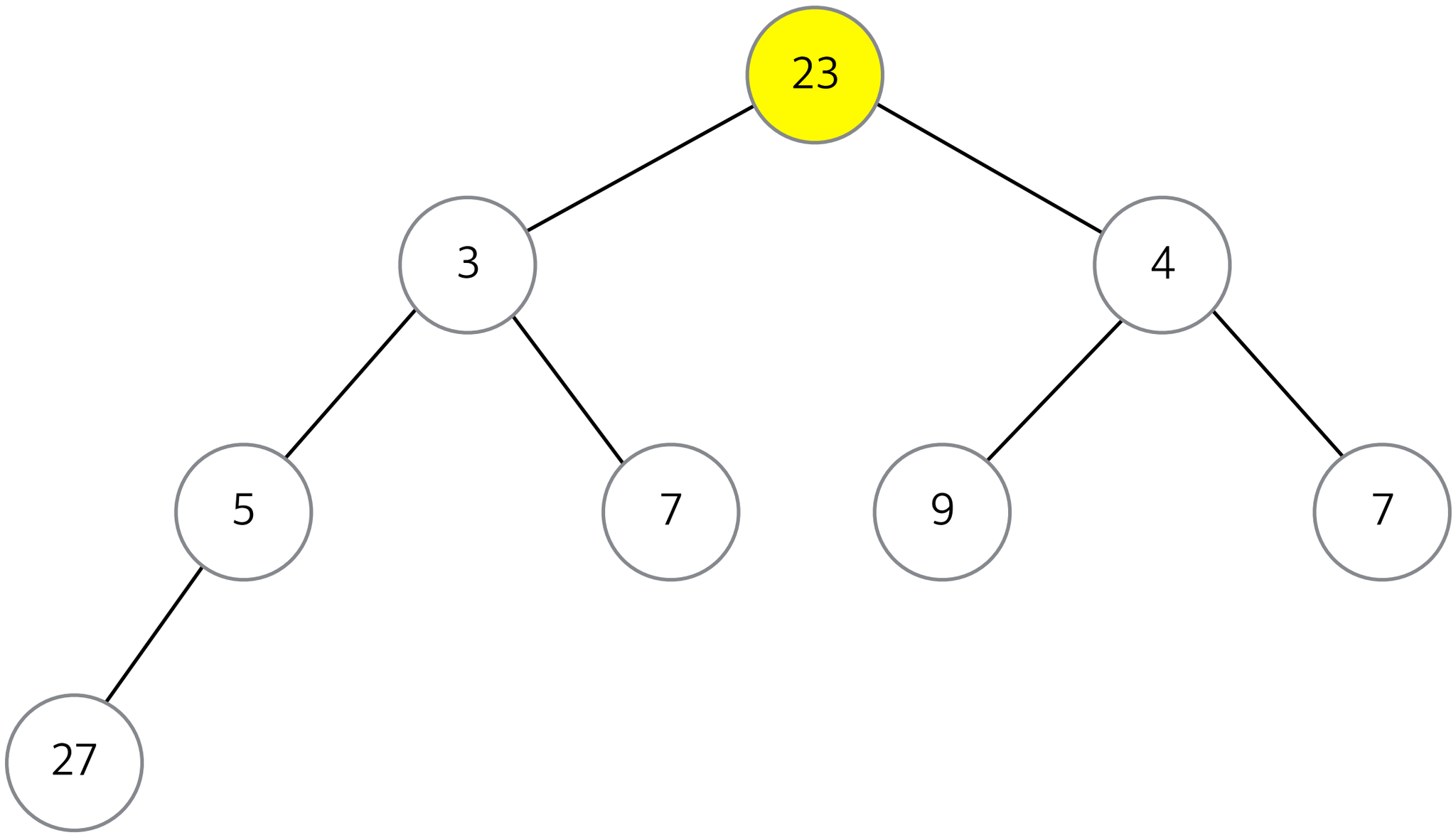
힙 : 값 삭제

heap.pop()



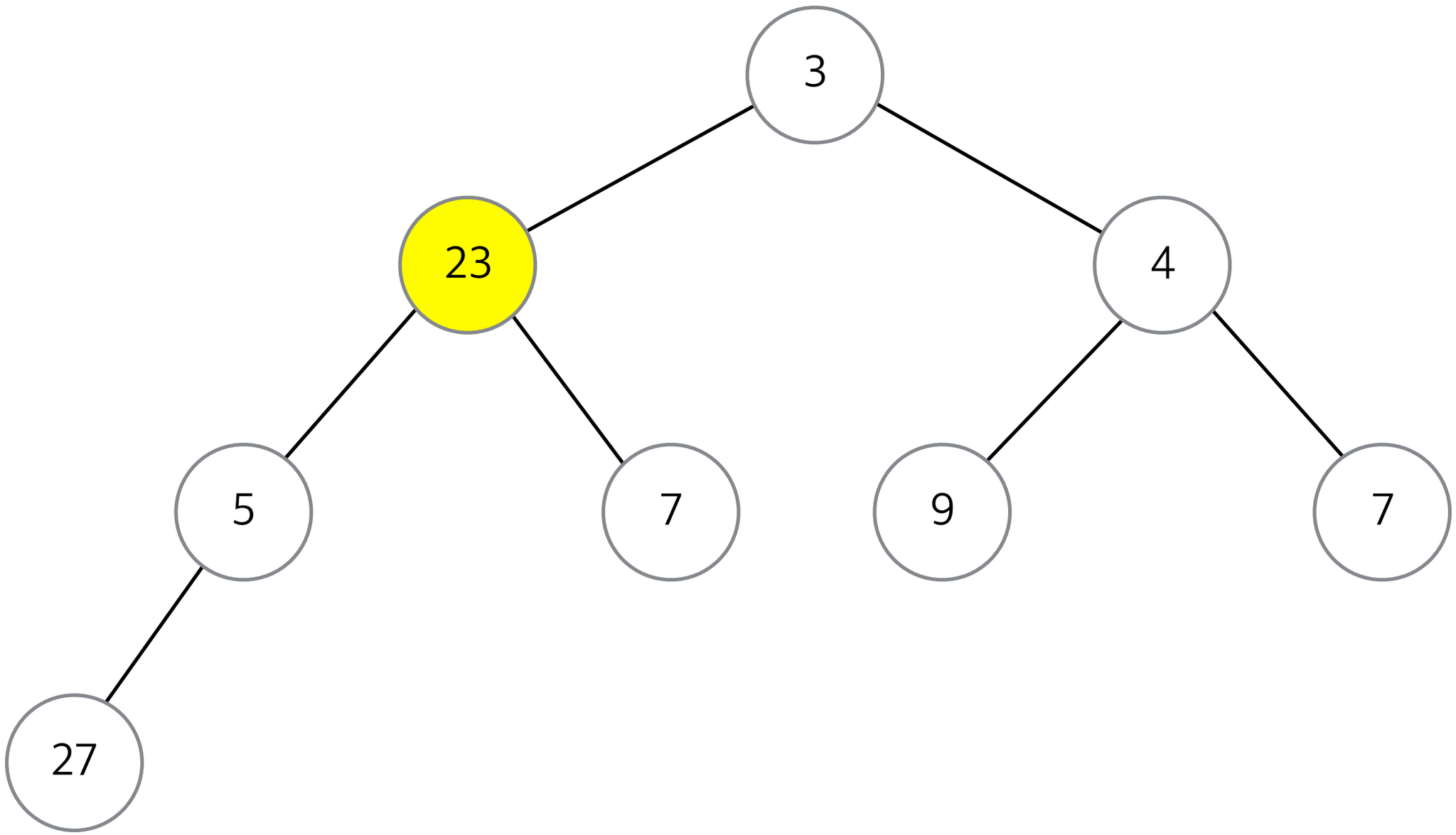
힙 : 값 삭제

heap.pop()



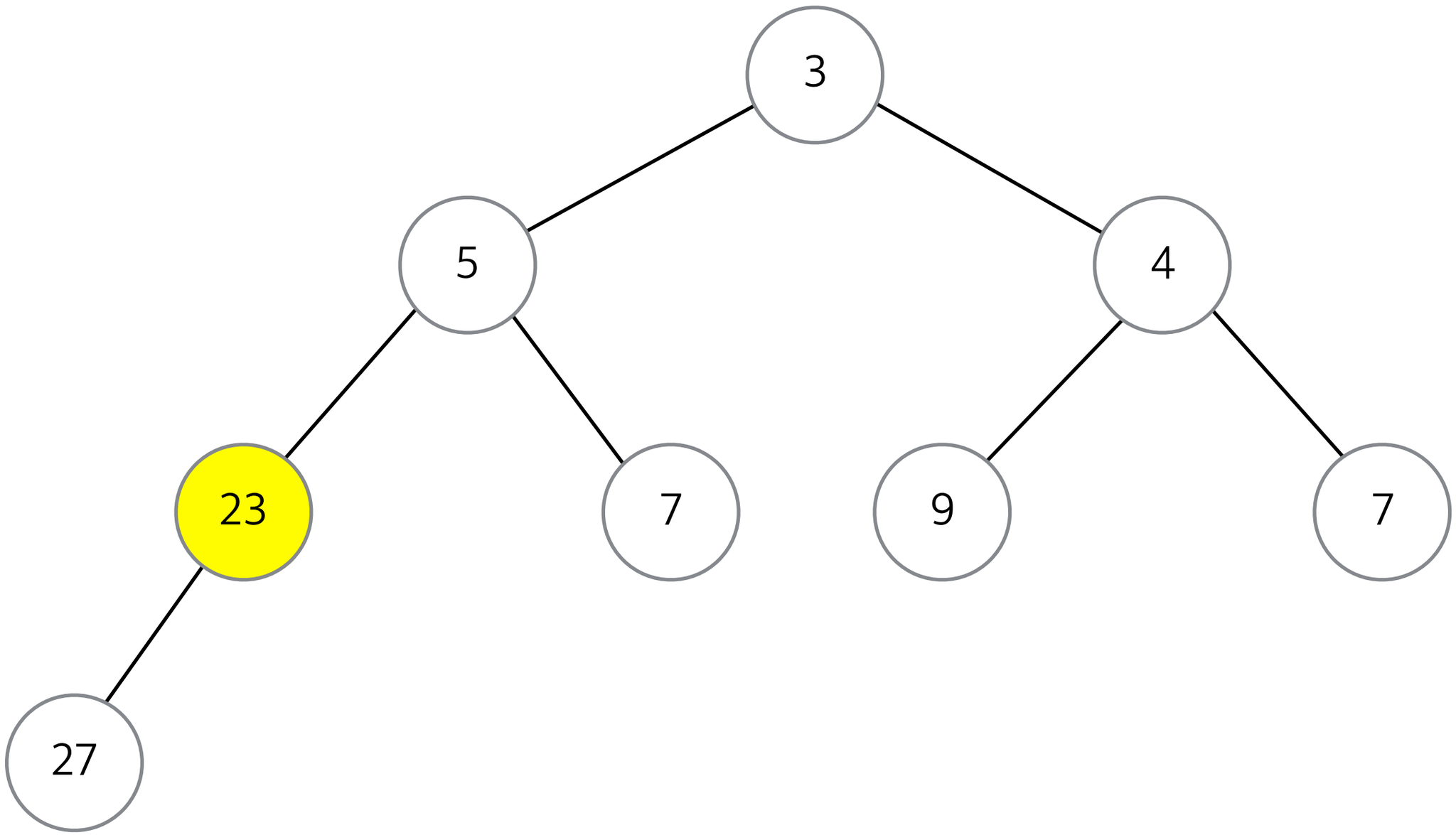
힙 : 값 삭제

heap.pop()



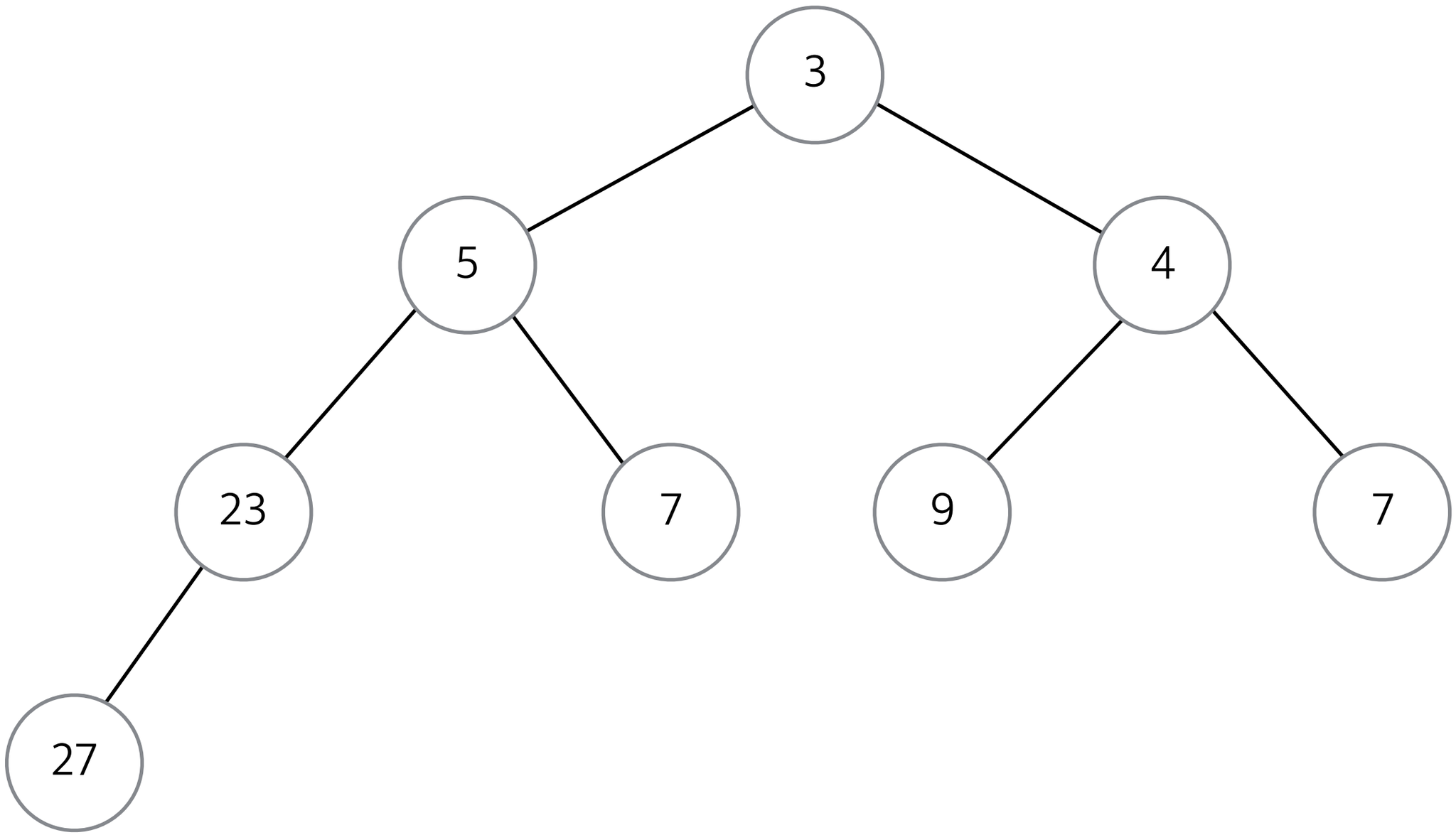
힙 : 값 삭제

heap.pop()



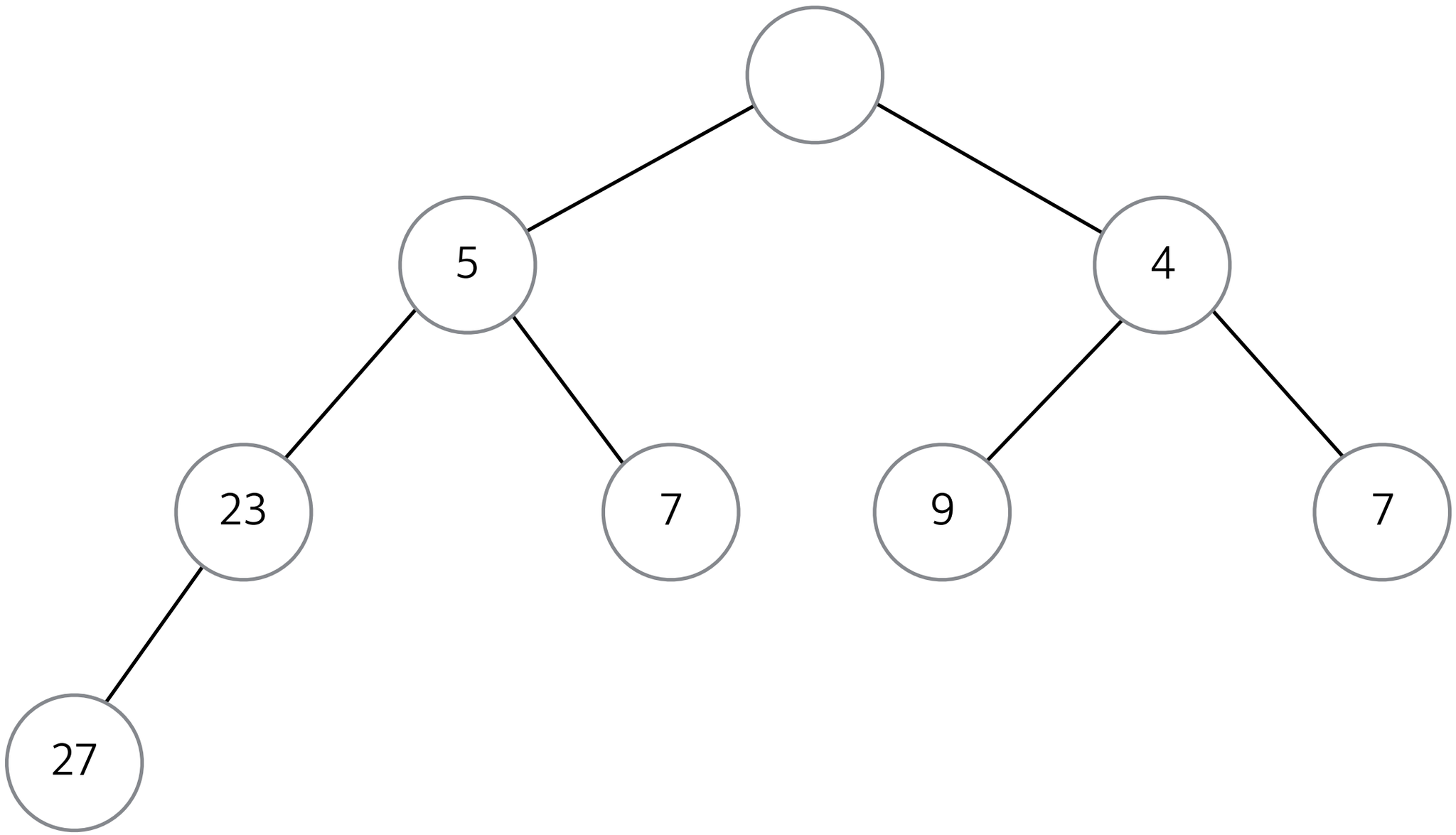
힙 : 값 삭제

heap.pop()



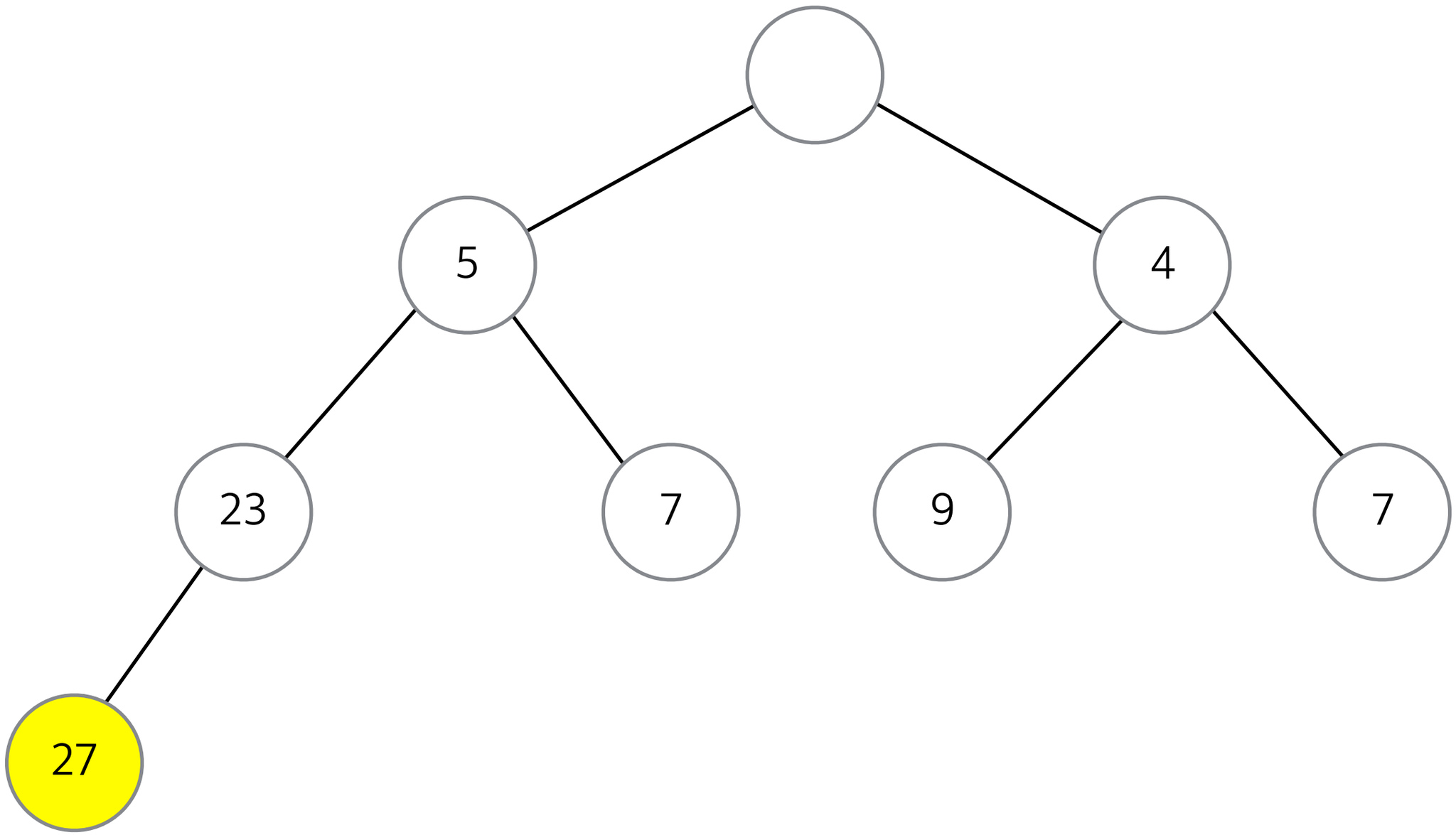
힙 : 값 삭제

heap.pop()



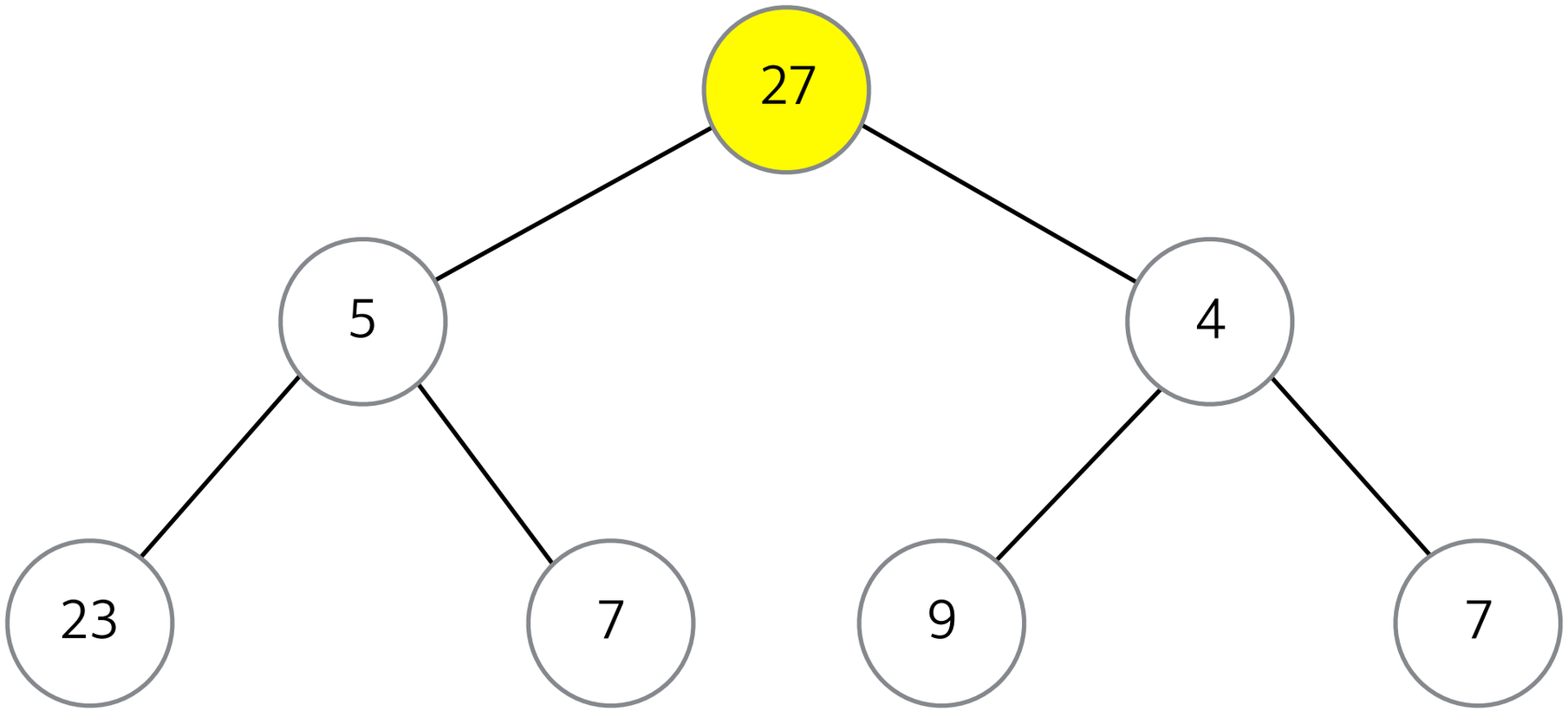
힙 : 값 삭제

heap.pop()



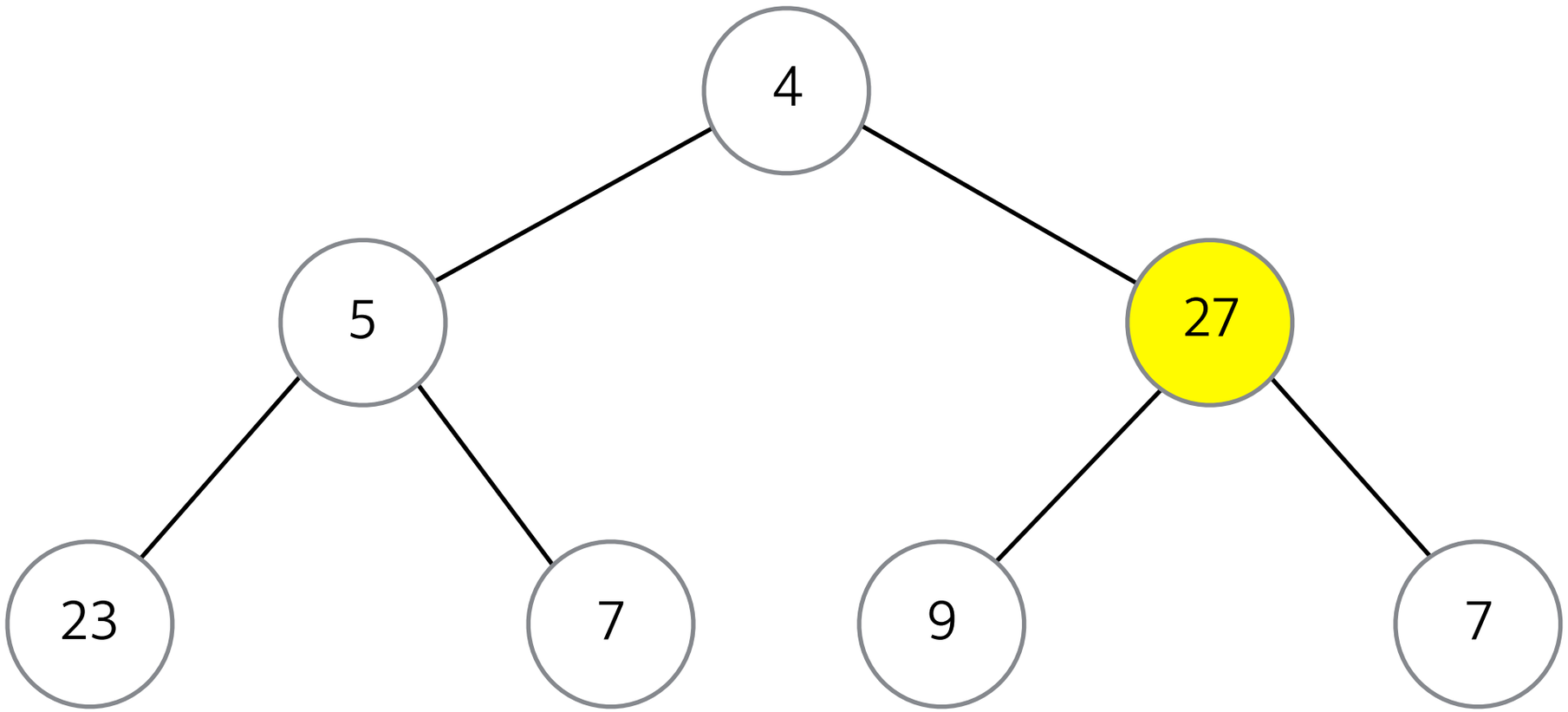
힙 : 값 삭제

heap.pop()



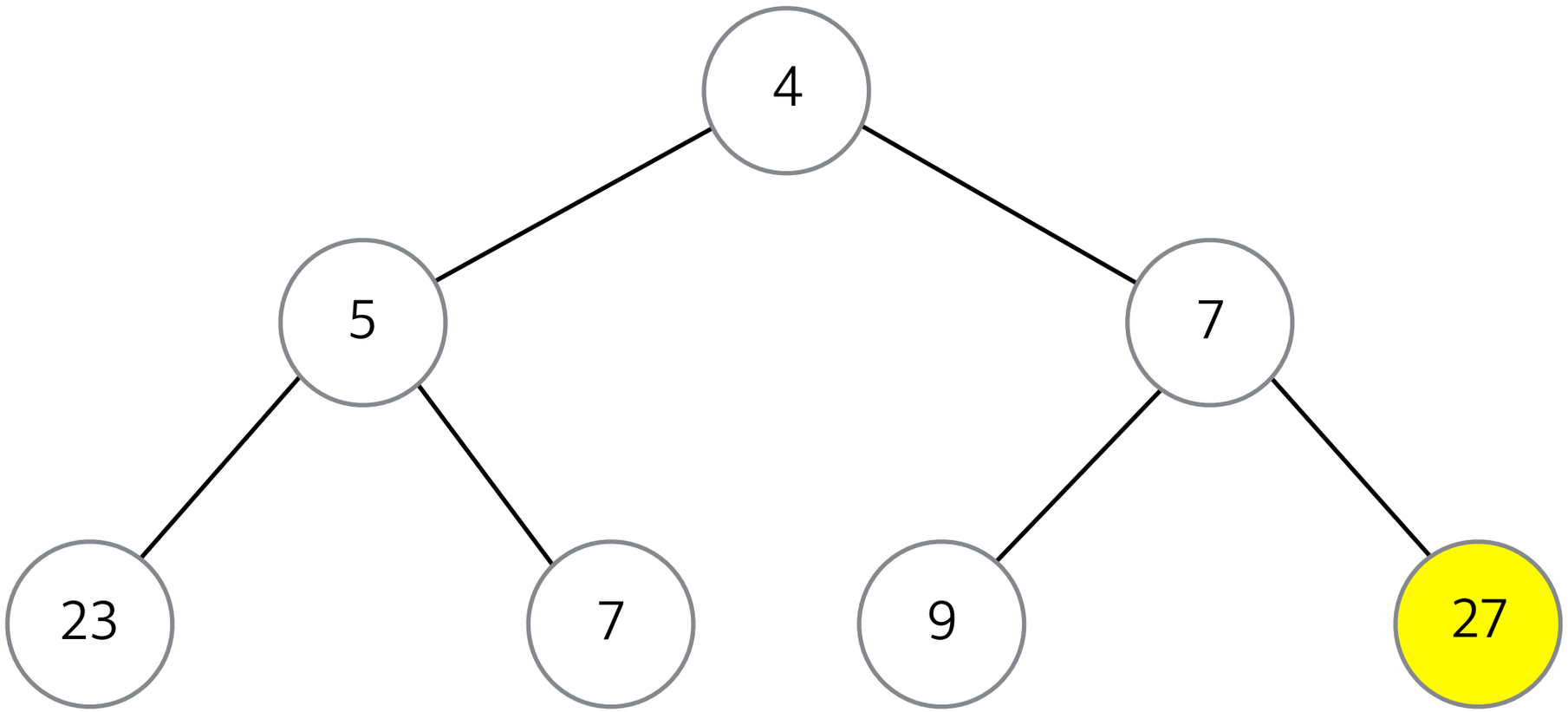
힙 : 값 삭제

heap.pop()



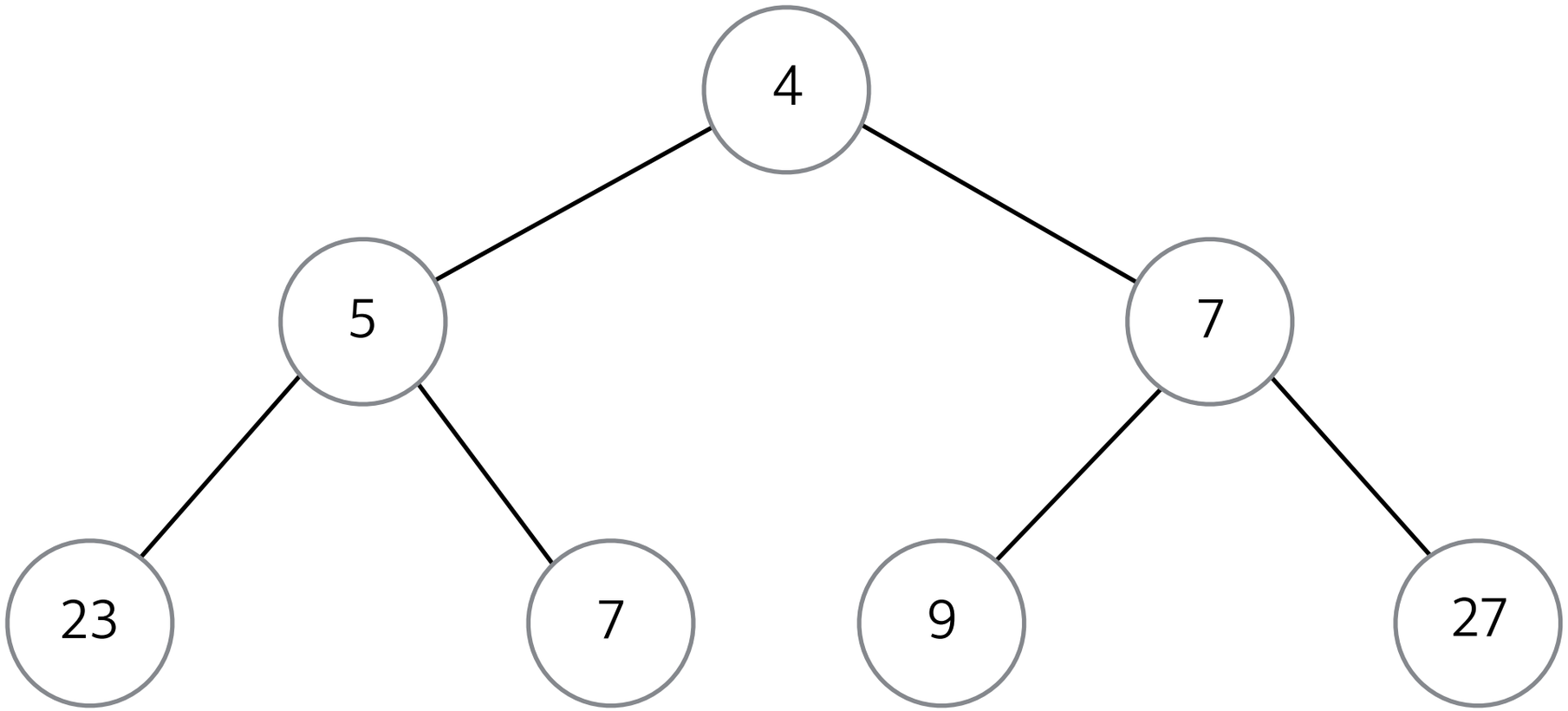
힙 : 값 삭제

heap.pop()

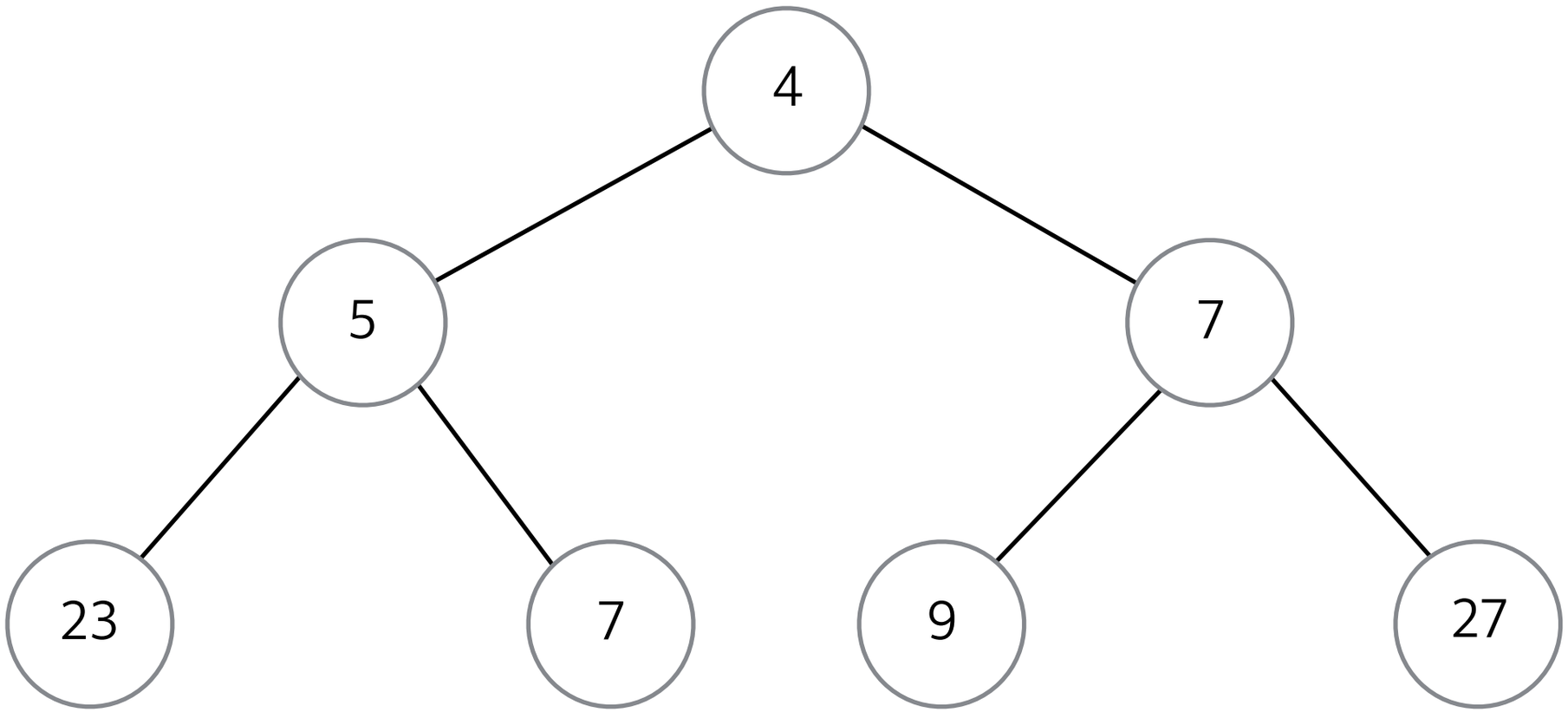


힙 : 값 삭제

heap.pop()

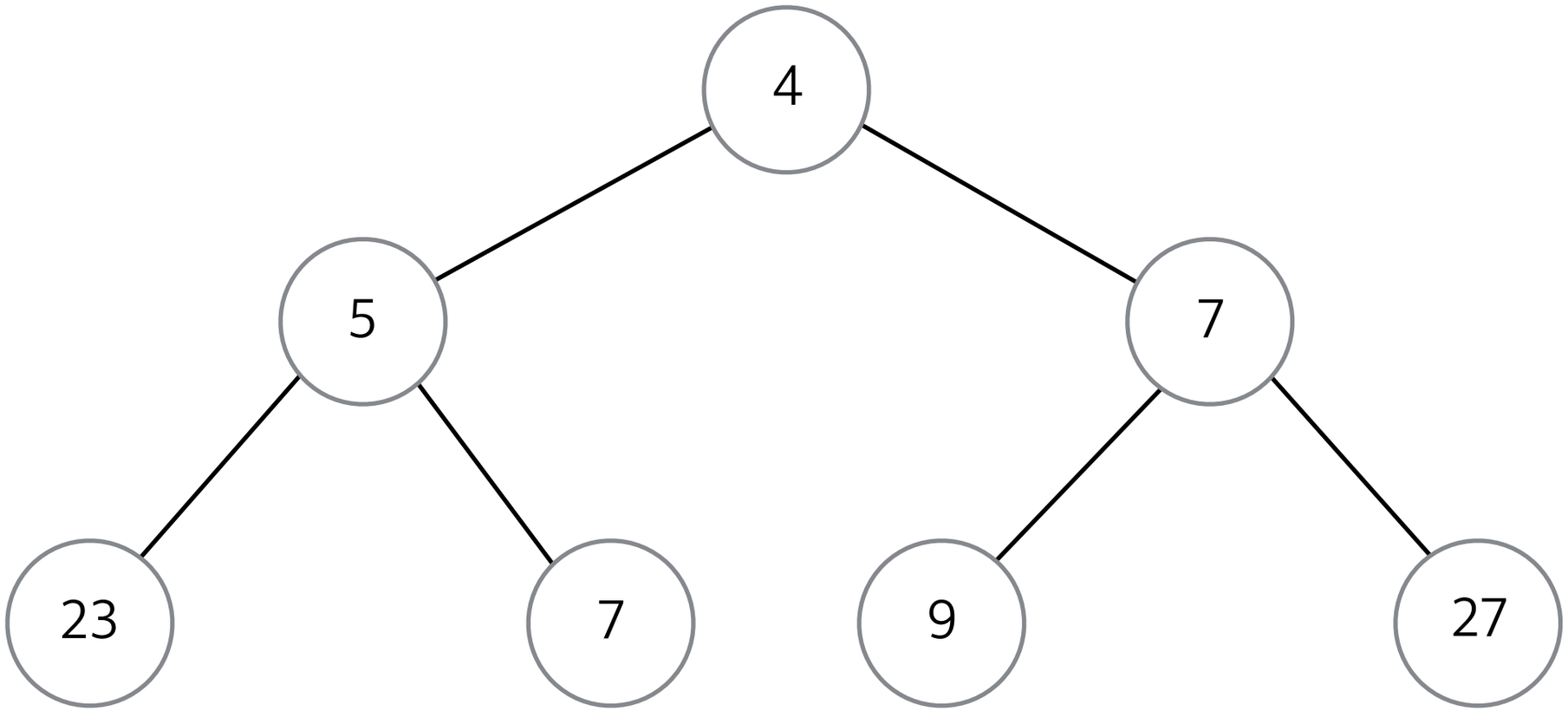


힙 : 값 삭제의 시간복잡도



힙 : 값 삭제의 시간복잡도

O(log n)



우선순위 큐의 구현 요약

	리스트	힙
값의 삽입	O(1)	O(log n)
값의 삭제	O(n)	O(log n)

[예제 2] 우선순위 큐 구현하기

힙을 구현하여 우선순위 큐를 완성하시오

시스템 입력의 예

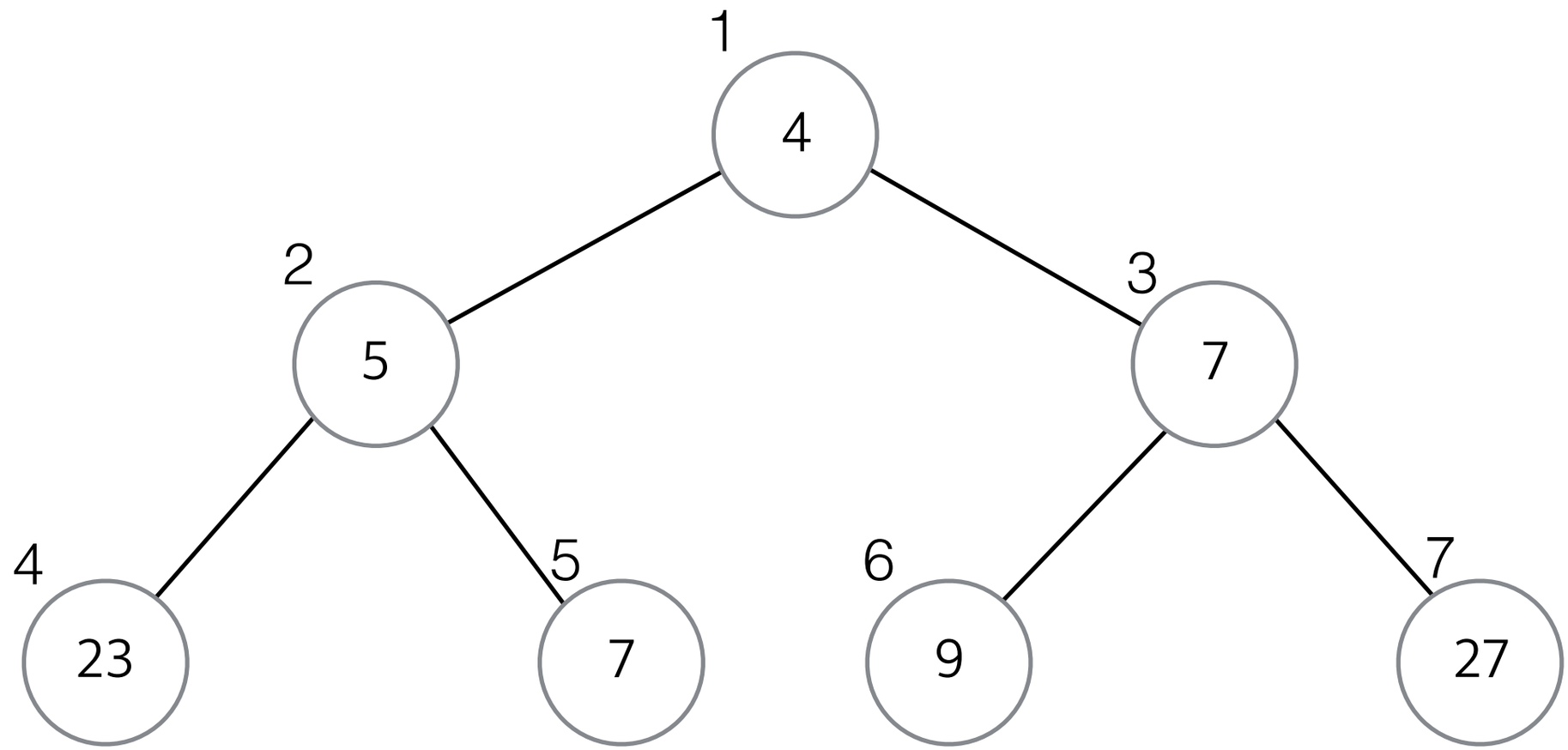
```
myPQ = priorityQueue()

myPQ.push(1)
myPQ.push(4)
myPQ.pop()

print(myPQ.top())
```

출력의 예

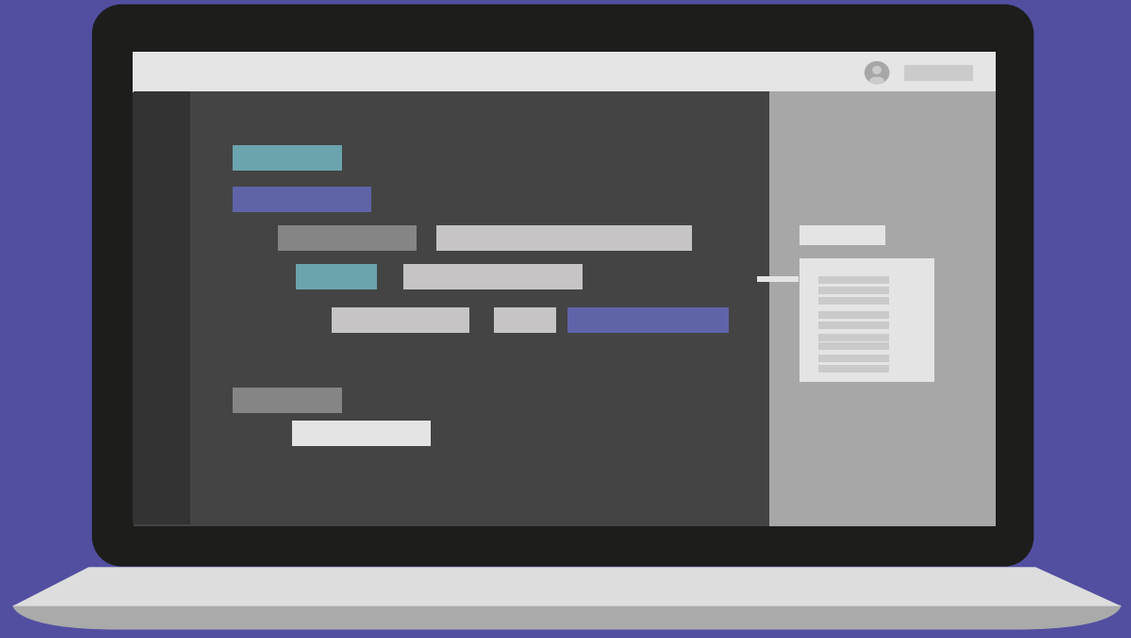
```
4
```

[예제 2] 우선순위 큐 구현하기

힙을 List를 이용하여 구현한다



[예제 1] 우선순위 큐 구현하기



의미 단위로 작성된 코드

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

Factorial(n) : n! 을 반환하는 함수

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

의미 단위로 작성된 코드

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

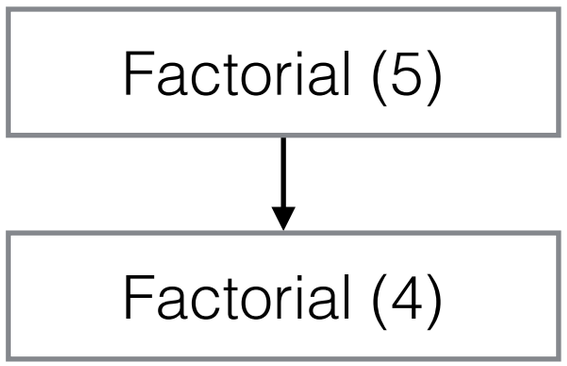
Factorial (5)

Factorial(n) : n! 을 반환하는 함수

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

의미 단위로 작성된 코드

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

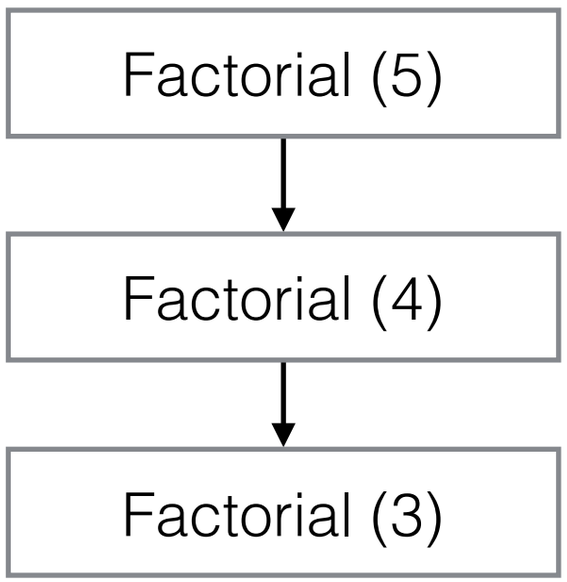


Factorial(n) : n! 을 반환하는 함수

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

의미 단위로 작성된 코드

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

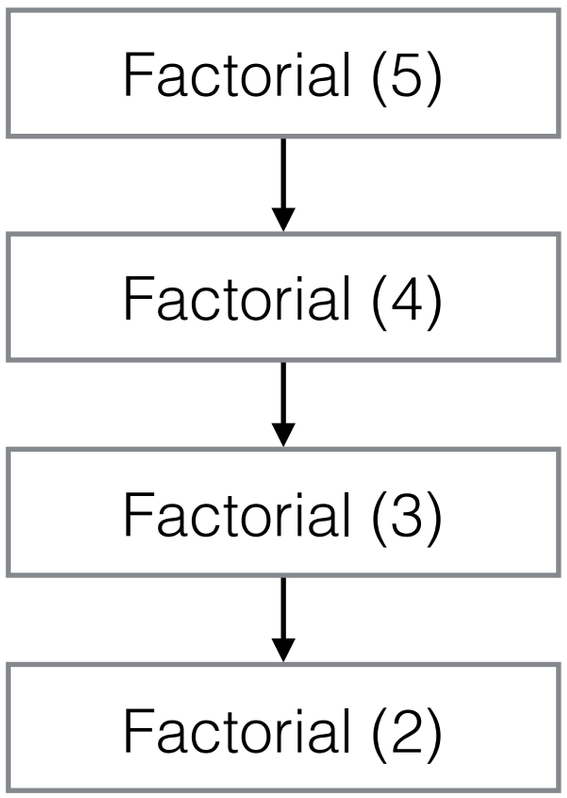


Factorial(n) : n! 을 반환하는 함수

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

의미 단위로 작성된 코드

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

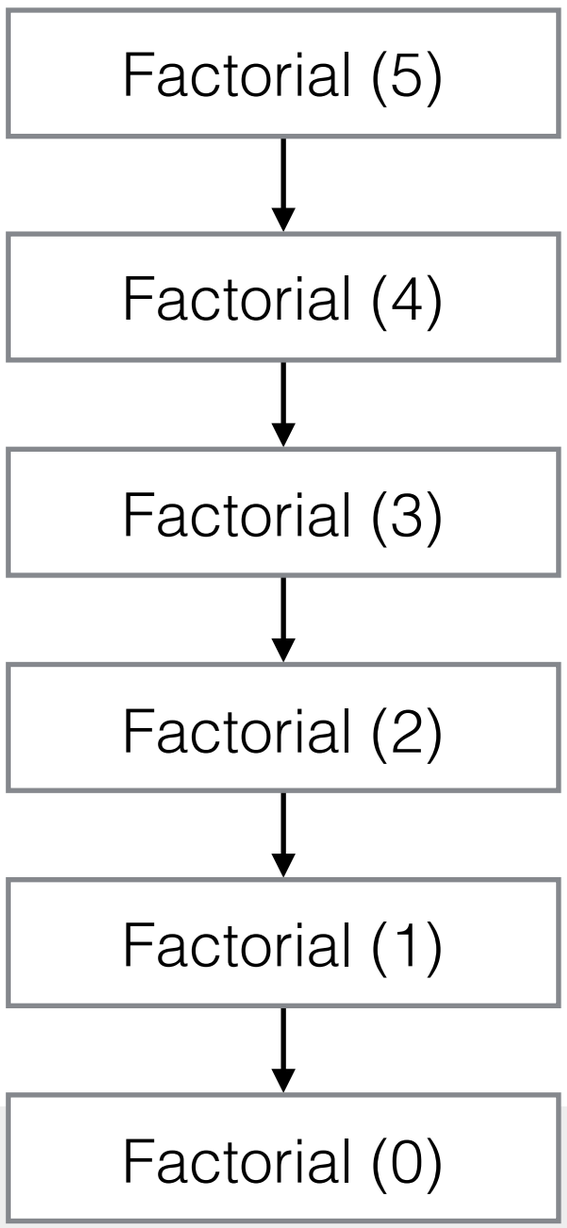


Factorial(n) : n! 을 반환하는 함수

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

의미 단위로 작성된 코드

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

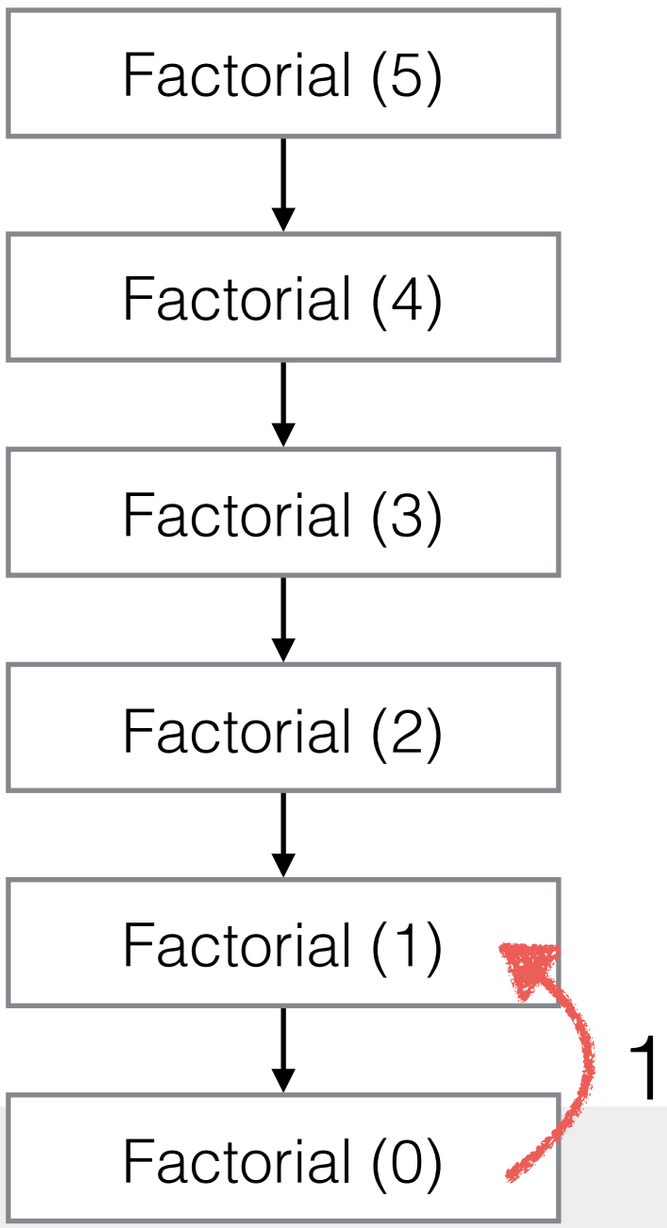


Factorial(n) : n! 을 반환하는 함수

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

의미 단위로 작성된 코드

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

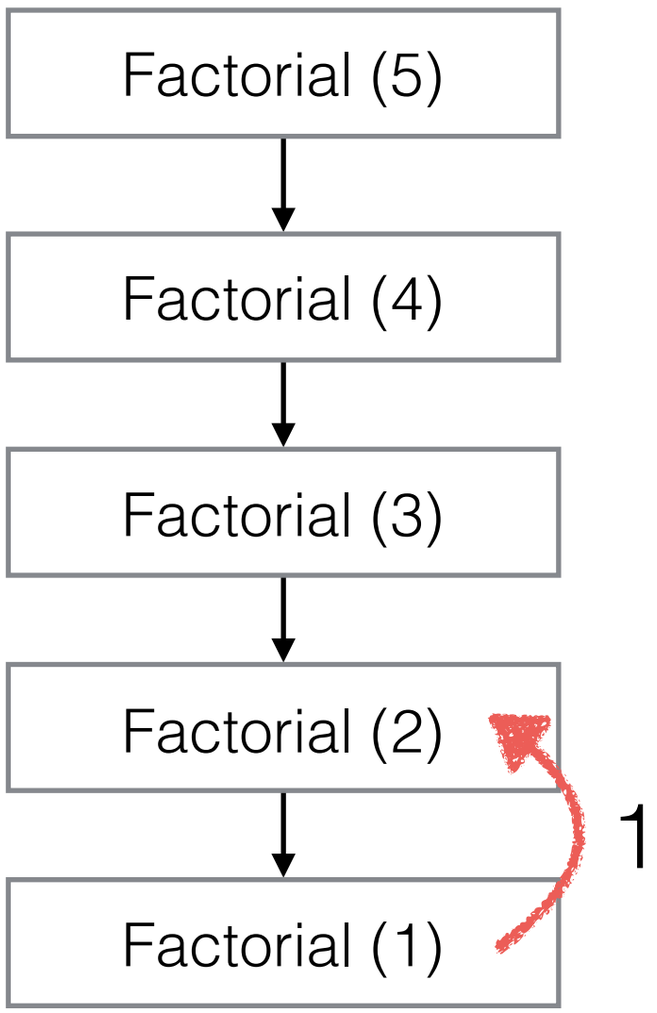


Factorial(n) : n! 을 반환하는 함수

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

의미 단위로 작성된 코드

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

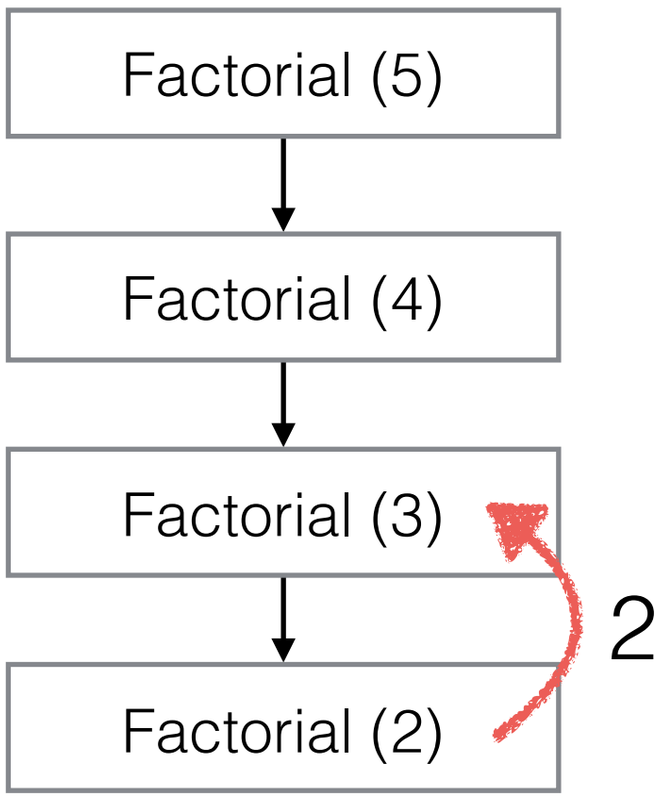


Factorial(n) : n! 을 반환하는 함수

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

의미 단위로 작성된 코드

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

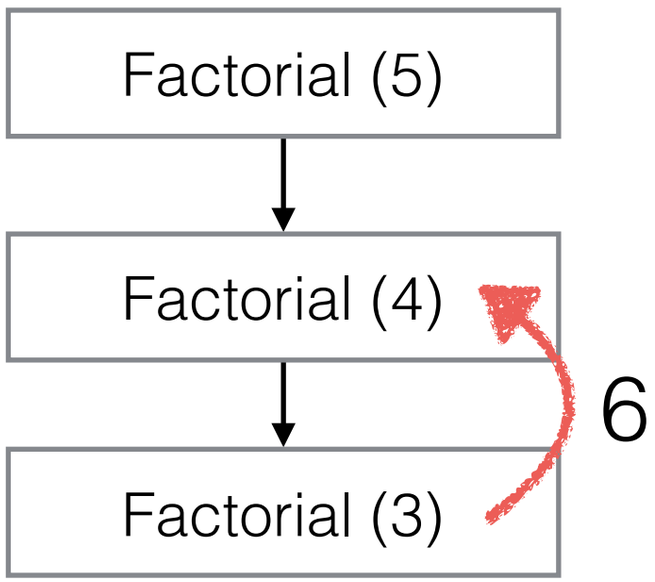


Factorial(n) : n! 을 반환하는 함수

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

의미 단위로 작성된 코드

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

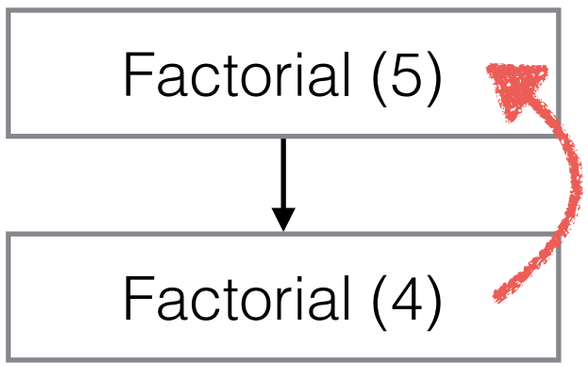


Factorial(n) : n! 을 반환하는 함수

```
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

24

```
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)
```

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

$$m^n = m \times m \times \ldots \times m$$

의미 단위로 작성된 코드

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

의미 단위로 작성된 코드

$$m^n = m \times m \times \ldots \times m$$

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

$$m^n = m \times m \times \ldots \times m$$

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

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

의미 단위로 작성된 코드

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

$m^n = (m^{(n/2)})^2$ n이 짝수일 경우

의미 단위로 작성된 코드

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

$$m^n = (m^{(n/2)})^2 \qquad n이 짝수일 경우$$

$$(m^{n-1}) \times m \qquad n이 홀수일 경우$$

의미 단위로 작성된 코드

$$m^n = m \times m \times \ldots \times m$$

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
    elif :
        return getPower(m, n-1) * m
```

의미 단위로 작성된 코드

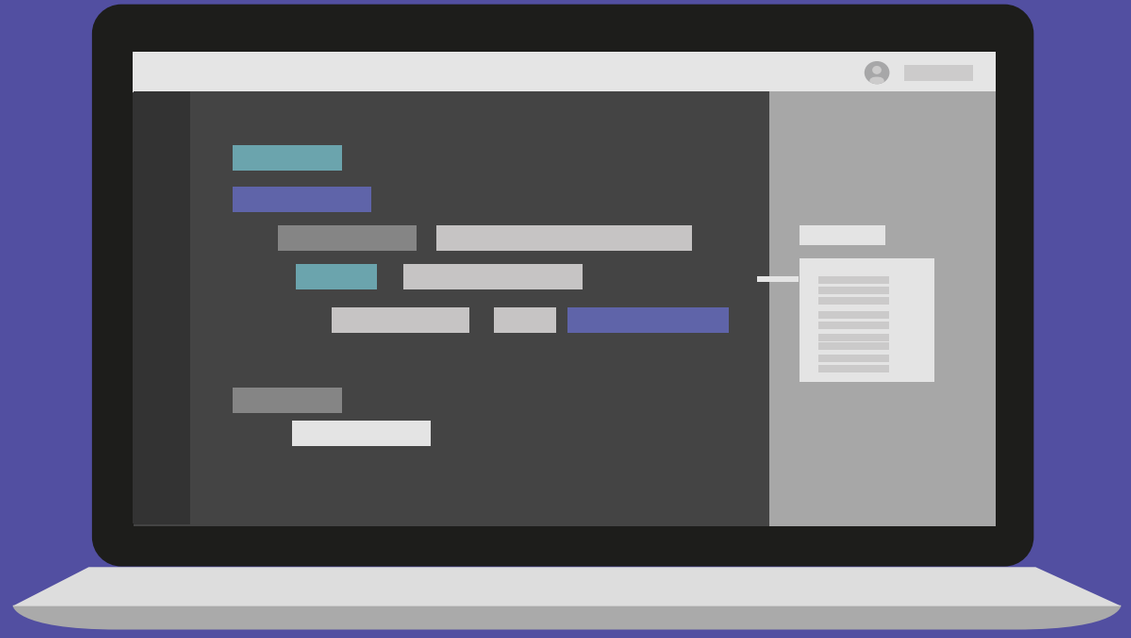
$m^n = m \times m \times \ldots \times m$

getPower(m, n) : m^n 을 반환하는 함수

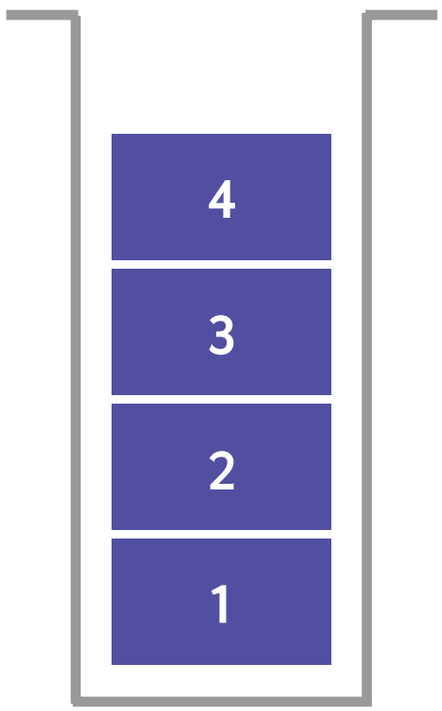
```
def getPower(m, n) :
    if n == 0 :
        return 1
    elif n % 2 == 0 :
        temp = getPower(m, n//2)
        return temp * temp
    elif :
        return getPower(m, n-1) * m
```

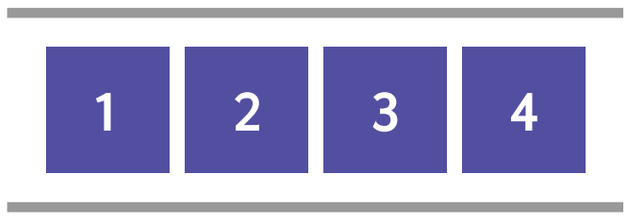
O(log n)

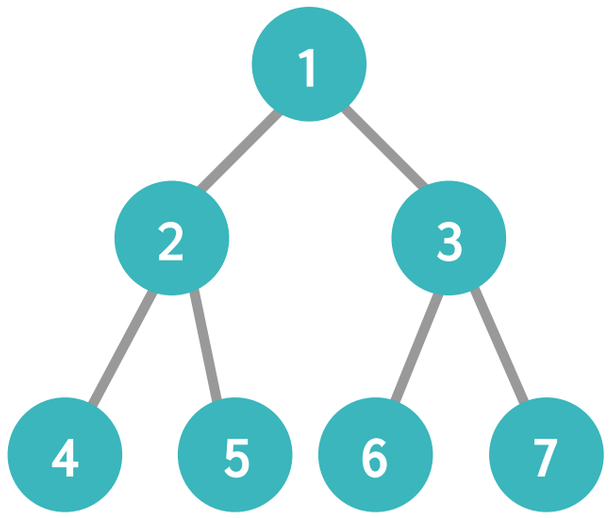
[예제 2] 거듭제곱 구하기

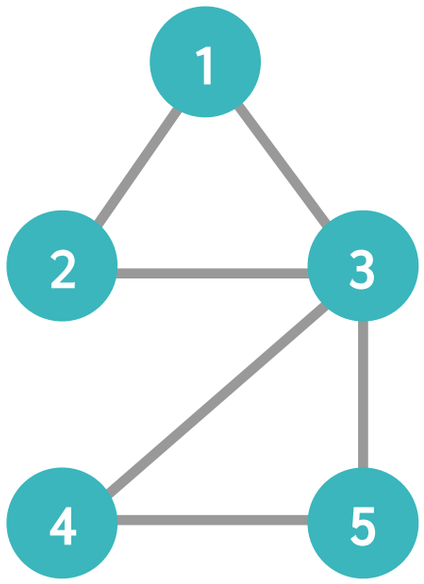


요약 : 자료구조









스택 (Stack)

Last In First Out

큐 (Queue)

First In First Out

트리 (Tree)

그래프 (Graph)

주차별 커리큘럼

1주차 과정 소개, 배열, 연결리스트, 클래스

- 자료구조는 자료를 담는 주머니입니다. 배열, 연결 리스트의 개념과 장단점을 알아봅니다.

2주차 스택, 큐, 해싱

- 초급 자료구조와 자료를 저장·검색할 때 사용되는 해싱을 배워봅니다.

3주차 트리, 트리순회, 재귀호출

- 나무와 비슷하게 생긴 자료인 트리에 대해 배워보고 트리에서 자료를 탐색하는 알고리즘과 재귀호출을 배워봅니다.

4주차 재귀호출 응용 및 힙

- 재귀호출로 해결할 수 있는 문제를 알아보고 그 의미를 찾아봅니다. 힙에 대해 알아보고,
  이를 이용하여 문제를 해결합니다.

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