땅울림
자구 스터디

Made by 규정

9주차

1	2	3	4
priority queue	선택 정렬, 삽입 정렬	heap	트리 순회

1. priority queue

1 priority queue

우선순위 큐

- 큐에 우선순위라는 개념을 도입한 자료구조
- 우선순위가 높은 데이터부터 나감

자료 구조	삭제되는 요소
스택(Stack)	가장 최근에 들어온 데이터
큐(Queue)	가장 먼저 들어온 데이터
우선순위 큐(Priority Queue)	가장 우선순위가 높은 데이터

1 priority queue

우선순위 큐의 구현

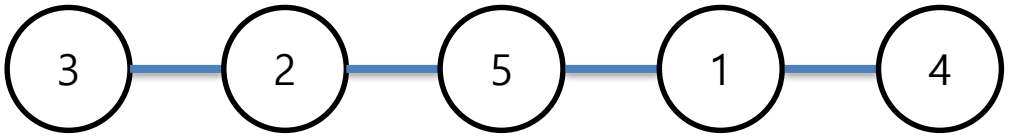
- 배열
- 연결 리스트
- 힙

pq의 표현방법	삽입	삭제
정렬 없는 배열	O(1)	O(n)
정렬 없는 리스트	O(1)	O(n)
정렬된 배열	O(n)	O(1)
정렬된 리스트	O(n)	O(1)
힙	O(logn)	O(logn)

1 priority queue

시퀀스로 구현한 우선순위큐

- 정렬되지 않은 리스트



insert : O(1)

그냥 맨 앞에 넣으면 되니까 O(1)

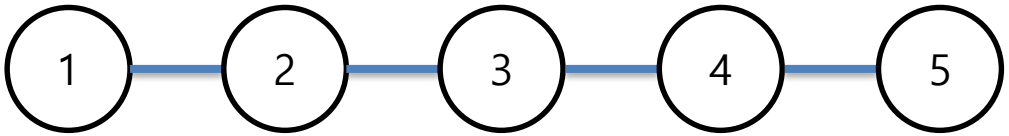
removeMin : O(N)

가장 작은 수를 찾아야하므로 O(N)

1 priority queue

시퀀스로 구현한 우선순위큐

- 정렬된 리스트



insert : O(N)

숫자가 들어갈 위치를 찾아야 하니까 O(N)

removeMin : O(1)

맨 앞의 수 삭제하면 끝이니까 O(1)

2. pq를 이용한 정렬

2 Selection sort

삽입을 빠르게!

선택 정렬(Selection sort)

- 리스트에 정렬 따위 없이 다 때려박아놓고
  최소값을 하나씩 **선택**해서 꺼내면서 정렬

- 정렬되지 않은 리스트
  insert : O(1)
  removeMin : O(N)

- 시간복잡도 : O(N^2)

2 Selection sort

정렬되지 않은 pq

	Sequence S	Priority Queue P
Input:	(7,4,8,2,5,3,9)	()
Phase 1		
(a)	(4,8,2,5,3,9)	(7)
(b)	(8,2,5,3,9)	(7,4)
..	.. ..	
(g)	()	(7,4,8,2,5,3,9)
Phase 2		
(a)	(2)	(7,4,8,5,3,9)
(b)	(2,3)	(7,4,8,5,9)
(c)	(2,3,4)	(7,8,5,9)
(d)	(2,3,4,5)	(7,8,9)
(e)	(2,3,4,5,7)	(8,9)
(f)	(2,3,4,5,7,8)	(9)
(g)	(2,3,4,5,7,8,9)	()

insert를 N번 =>O(N)

removeMin을 N번
=> O(N^2)

2 Insertion sort

꺼낼 때 빠르게!

삽입 정렬(Insertion sort)

- 리스트에 **삽입**할 때 정렬시키면서 삽입하고 맨 앞에부터 꺼내면서 정렬
- 정렬된 리스트

  insert : O(N)
  removeMin : O(1)
- 시간복잡도 : O(N^2)

2 Selection sort

정렬된 pq

	Sequence S	Priority queue P
Input:	(7,4,8,2,5,3,9)	()
Phase 1		
(a)	(4,8,2,5,3,9)	(7)
(b)	(8,2,5,3,9)	(4,7)
(c)	(2,5,3,9)	(4,7,8)
(d)	(5,3,9)	(2,4,7,8)
(e)	(3,9)	(2,4,5,7,8)
(f)	(9)	(2,3,4,5,7,8)
(g)	()	(2,3,4,5,7,8,9)
Phase 2		
(a)	(2)	(3,4,5,7,8,9)
(b)	(2,3)	(4,5,7,8,9)
..	..	..
(g)	(2,3,4,5,7,8,9)	()

insert를 N번 => O(N^2)

removeMin을 N번 => O(N)

2 sort

삽입,삭제 정렬은 In-place 가능

- 별도의 공간을 사용하지 않고 주어진 배열 내에서 정렬을 완성시키는 방법(swap)

- http://hsp1116.tistory.com/33

3. heap

3 heap

- 시퀀스로 pq를 구현하면 정렬 여부 상관없이 O(N^2)의 성능
- 개선시킬 방법 : heap 구조 사용

3 heap

heap

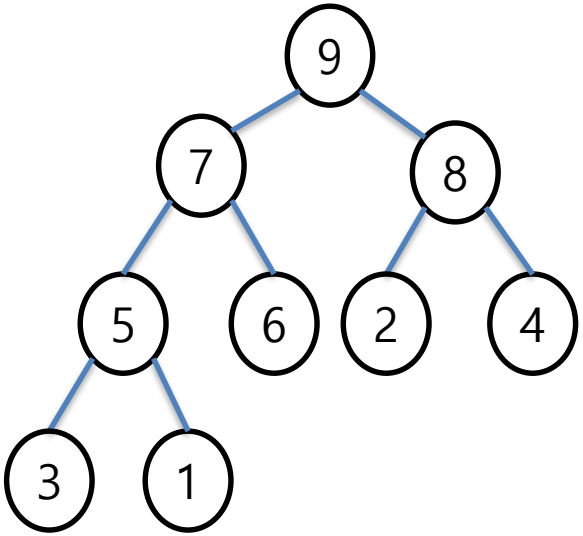
- 최댓값이나 최솟값을 빠르게 찾아내기 위한 자료구조
- 완전 이진 트리로 표현
- 부모 노드의 키 값이 자식 노드의 키 값보다 항상 작거나 같음 (최소힙일 때)
  => 루트의 키 값이 최솟값이 됨

※ 헷갈주의) 이진 탐색 트리(BST)는 왼쪽 자식은 더 작고, 오른쪽 자식은 더 큼

3 heap의 종류

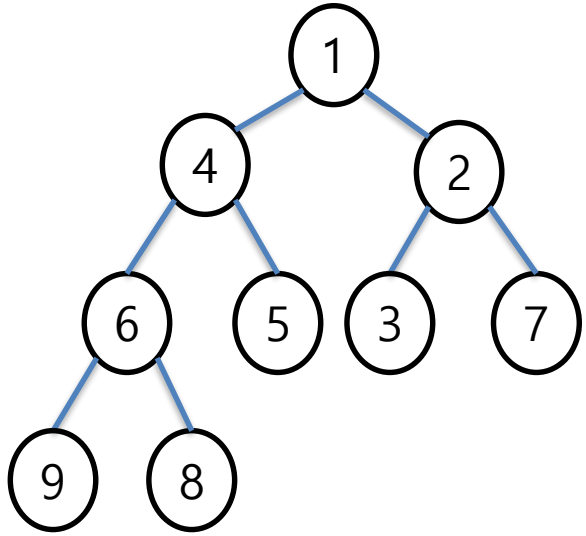
· 최대 힙(max heap)

부모 노드의 키 값이 항상 더 크거나 같음



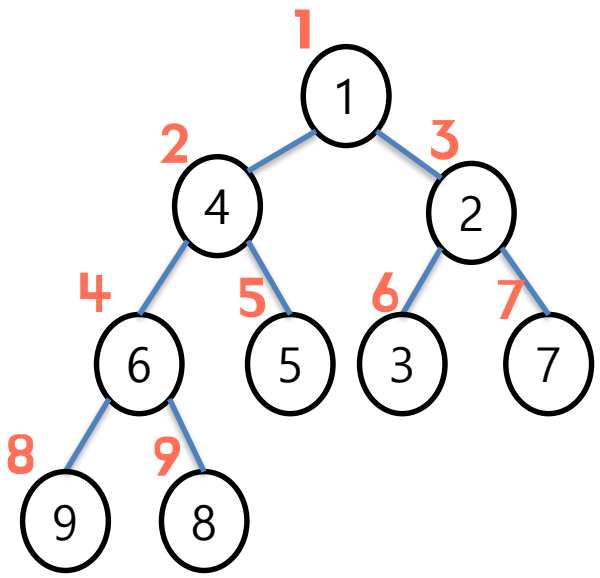
· 최소 힙(min heap)

부모 노드의 키 값이 항상 더 작거나 같음



3 heap의 구현

- 완전이진트리이므로 배열로 구현 가능
- 1-based로 구현하는게 편함(인덱스 1부터 시작)
- 자신의 인덱스 * 2 = 왼쪽 자식의 인덱스
  자신의 인덱스 * 2 + 1 = 오른쪽 자식의 인덱스



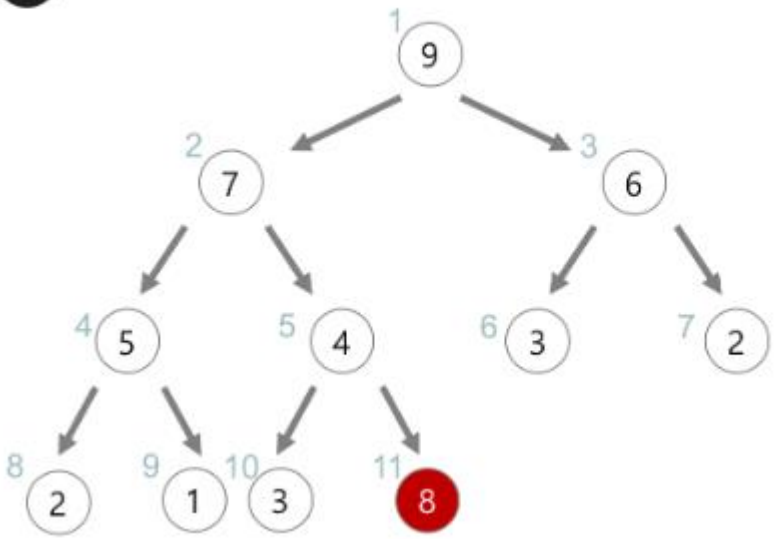
0	1	2	3	4	5	6	7	8	9
-	1	4	2	6	5	3	7	9	8

heap의 삽입

- 새로운 노드를 일단 힙의 마지막에 삽입
- 힙의 성질을 만족할 때까지 부모 노드와 교환 (upheap)

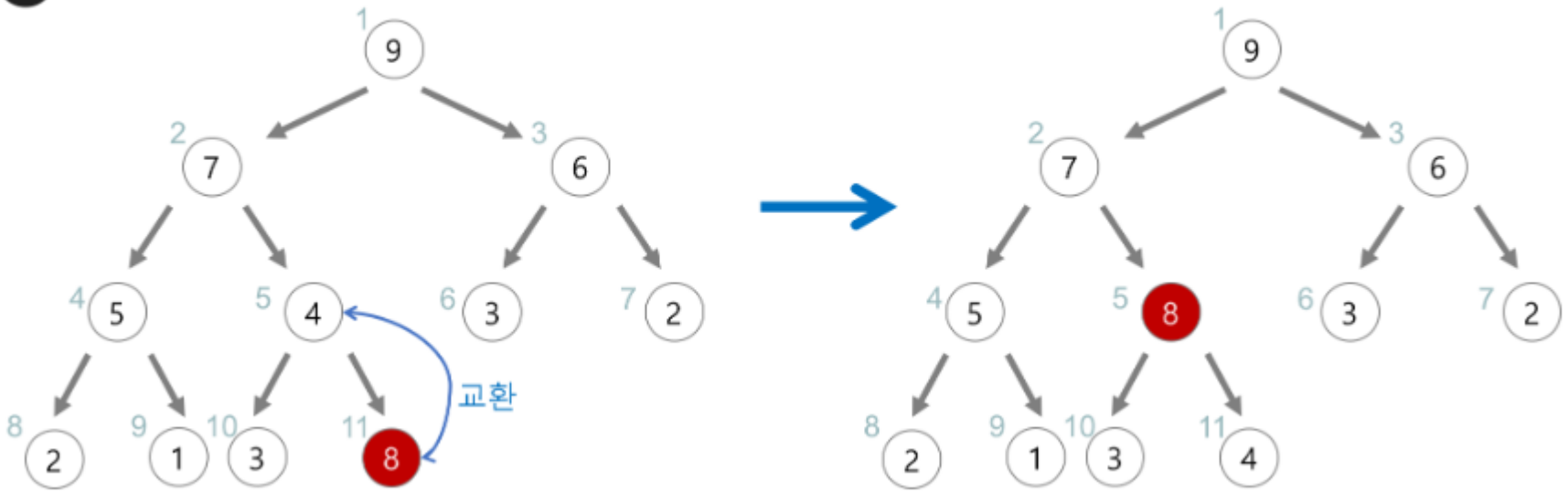
max heap

1 인덱스순으로 가장 마지막 위치에 이어서 새로운 요소 8을 삽입

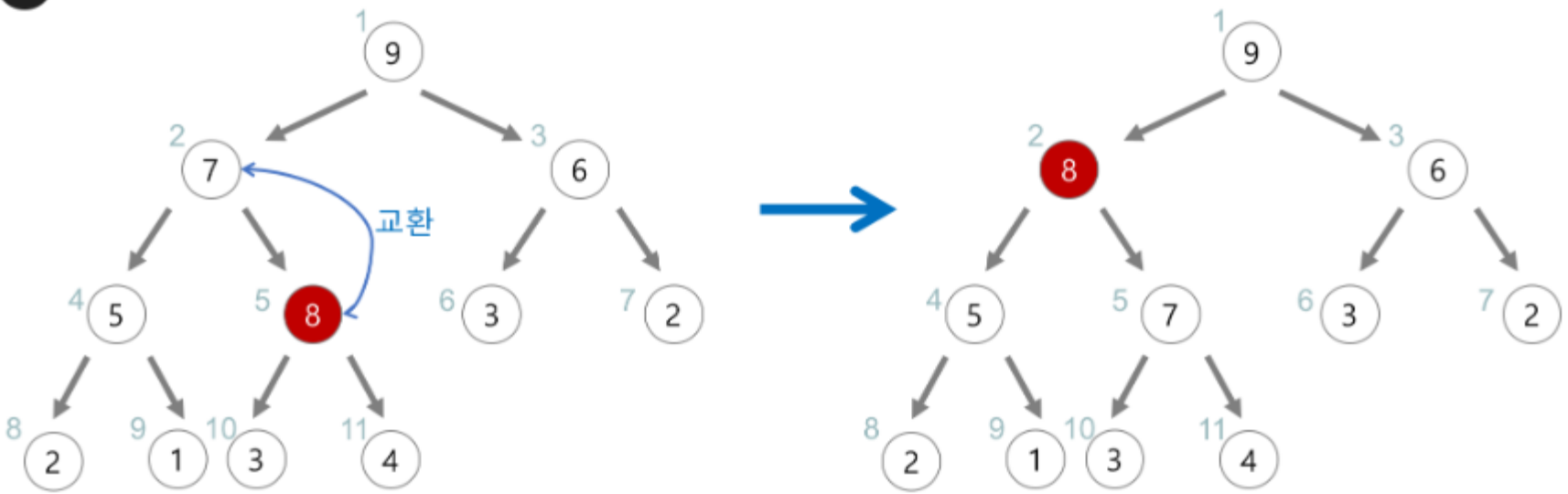


---

2 부모 노드 4 < 삽입 노드 8 이므로 서로 교환



3 부모 노드 7 < 삽입 노드 8 이므로 서로 교환

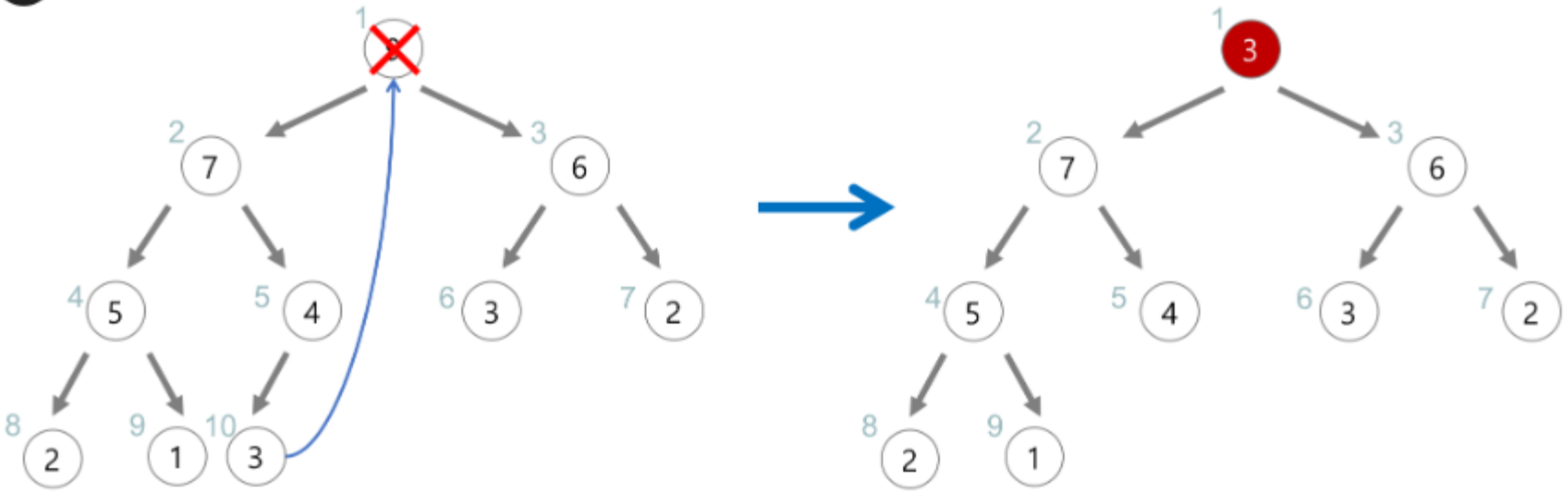


4 부모 노드 9 > 삽입 노드 8 이므로 더 이상 교환하지 않는다.

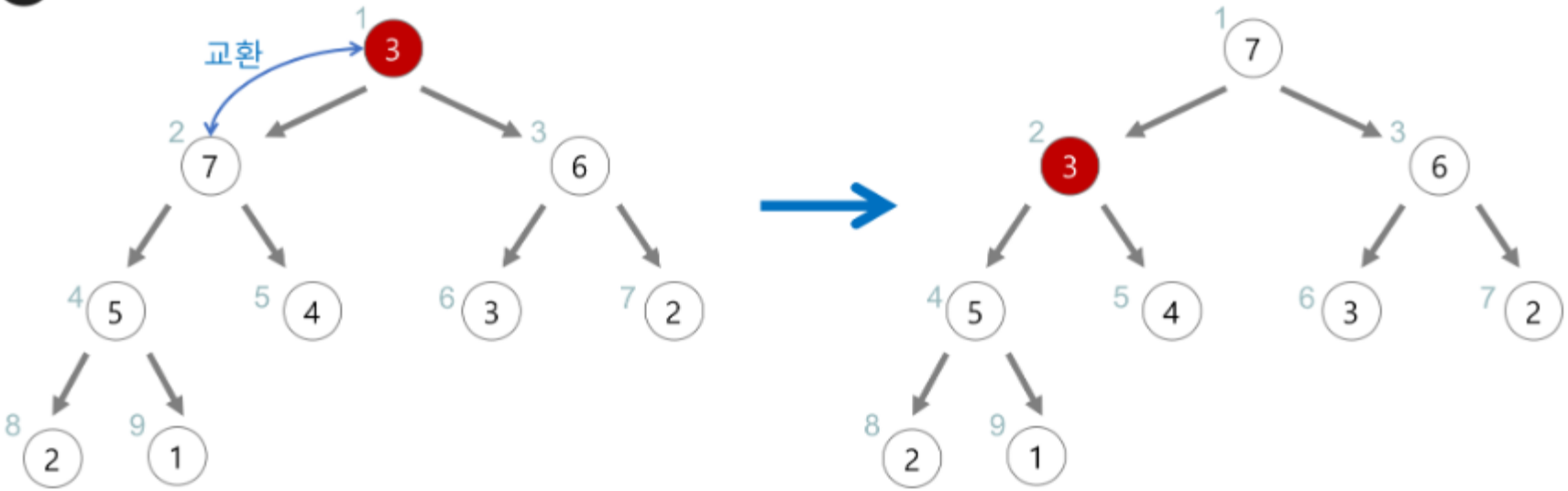
heap의 삭제

- 루트 노드를 삭제
- 힙의 마지막 노드를 루트로 옮김
- 힙의 성질을 만족할 때까지 **자식** 노드와 교환 (downheap)

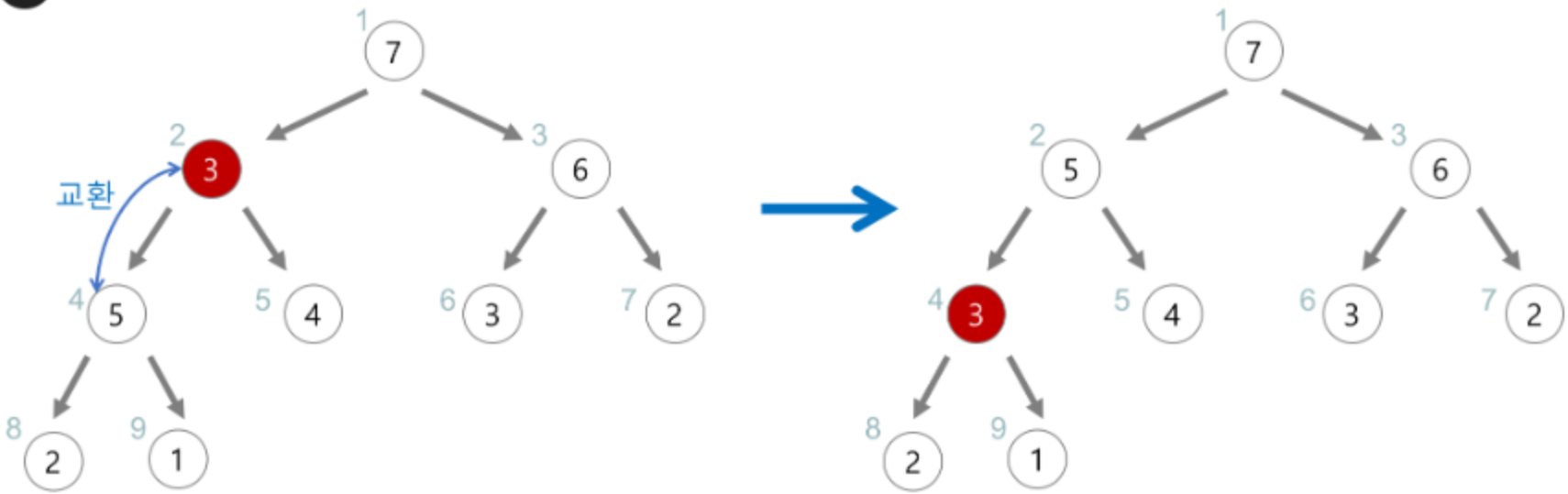
1 최댓값인 루트 노드 9를 삭제. (빈자리에는 최대 힙의 마지막 노드를 가져온다.)



2 삽입 노드와 자식 노드를 비교. 자식 노드 중 더 큰 값과 교환. (자식 노드 7 > 삽입 노드 3 이므로 서로 교환)



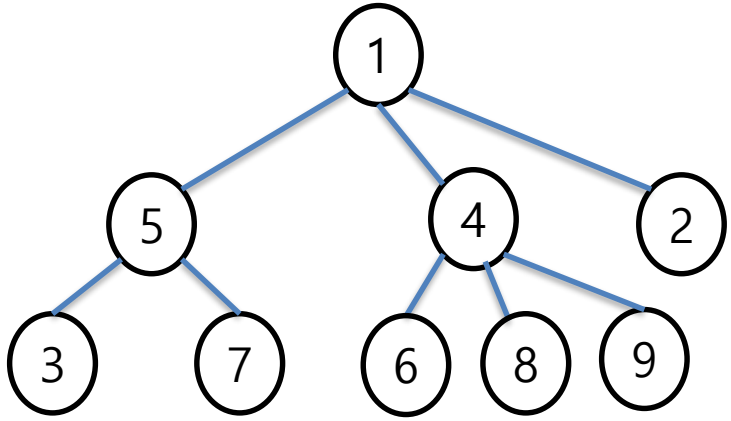
3 삽입 노드와 더 큰 값의 자식 노드를 비교. 자식 노드 5 > 삽입 노드 3 이므로 서로 교환



4 자식 노드 1, 2 < 삽입 노드 3 이므로 더 이상 교환하지 않는다.

4. 트리의 순회

4 BFS



첫째줄에 노드의 개수(N)이 주어진다.
다음 N-1개의 줄에 트리의 연결 정보(u, v)가 주어진다.
입력 (u, v) 는 v가 u의 자식 노드라는 의미이다.
트리를 입력받아 BFS탐색을 한 결과를 출력해라.

입력 제한 : 1<=N<=100, 1 <= u, v <= 1000

예제 입력

```
9
1 5
1 4
1 2
5 3
5 7
4 6
4 8
4 9
```

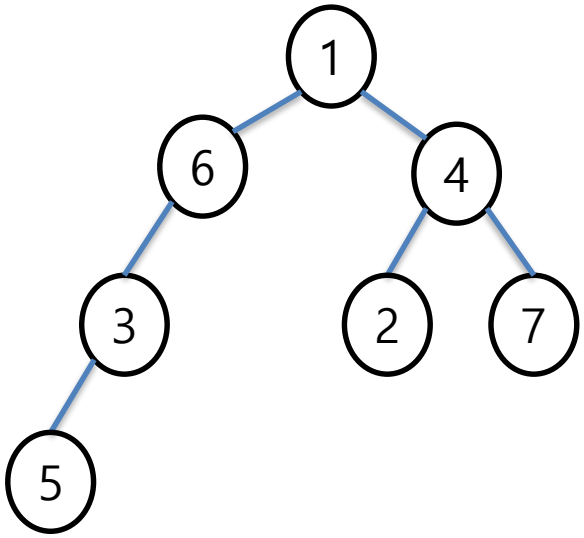
예제 출력

```
1 5 4 2 3 7 6 8 9
```

4 트리의 순회

11725_트리의 부모 찾기

문제 : 트리가 주어지고, 루트가 1일 때 각 노드의 부모를 출력하는 문제



예제 입력

```
7
1 6
6 3
3 5
4 1
2 4
4 7
```

예제 출력

```
4
6
1
3
1
4
```

입력이 부모–자식 순서대로 들어오는 게 아님!

4 트리의 순회

BFS로 풀이

1. 일단 양쪽으로 트리를 연결시켜놓고 루트부터 탐색한다.

2. 한 번 방문한 노드는 체크해서 다시 방문하지 않도록 한다.

3. 루트에서부터 타고 내려오면서 각 노드마다 부모노드를 저장

4. 이런 느낌으로 구현은 알아서 ㅎㅎ

감사합니다.

Made by 규정