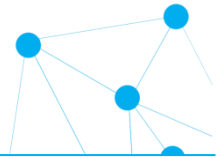
데이터구조 8장

08-1. 힙트리(우선순위큐) 개념

우선순위 큐(Priority queue)



- 우선순위 큐(Priority queue)
  - 우선순위를 가진 항목들을 저장하는 큐
  - 우선 순위가 높은 데이터가 먼저 나가게 됨
  - 가장 일반적인 큐로 생각할 수 있음
    - 스택이나 큐를 우선순위 큐로 구현할 수 있다.

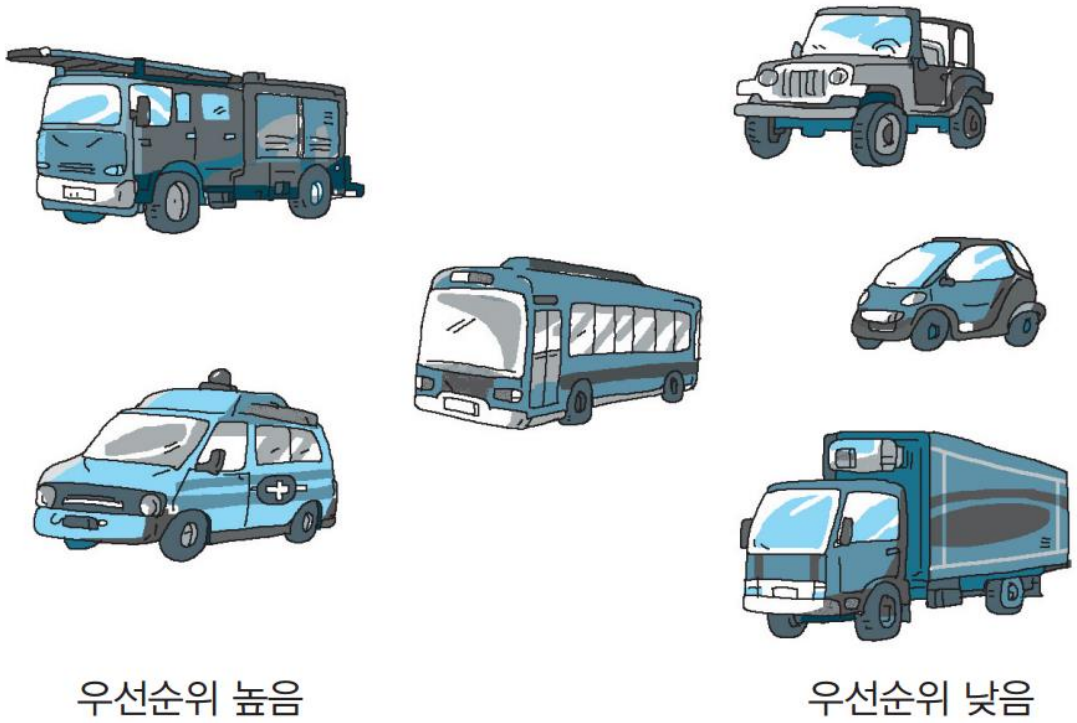
우선순위를 데이터가 들어온 시각으로 잡으면 우선순위 큐를 일반적인 큐처럼 동작시킬 수 있다.

자료구조	삭제되는 요소
스택	가장 최근에 들어온 데이터
큐	가장 먼저 들어온 데이터
우선순위큐	가장 우선순위가 높은 데이터

- 응용분야
  - 시뮬레이션, 네트워크 트래픽 제어, OS의 작업 스케쥴링 등

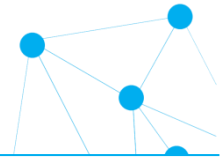
실생활에서의 우선순위

- 도로에서의 자동차 우선순위



08-2. 힙(우선순위큐)의 구현

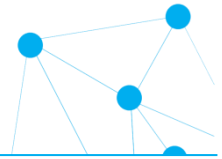
우선순위큐 구현방법



- 우선순위큐는 2가지로 구분
  - 최소 우선순위 큐
  - 최대 우선순위 큐

- 힙(heap)을 이용한 구현
  - 우선순위 큐를 위해 만들어진 자료구조(트리)
  - 일종의 반 정렬 상태를 유지

힙(heap)이란?

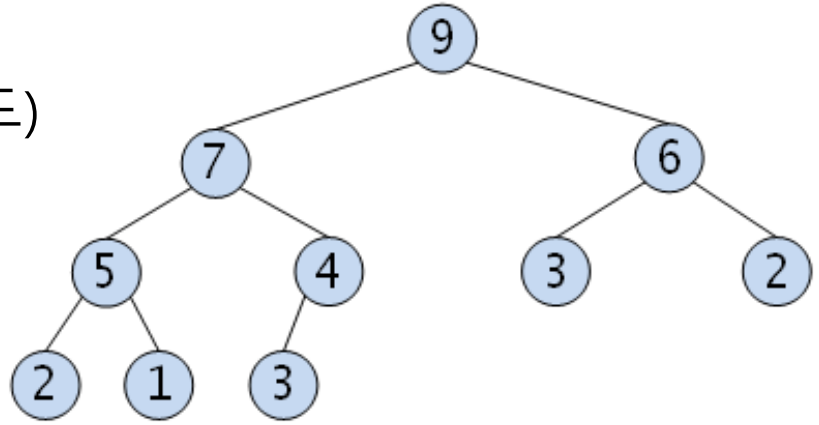


- 힙(Heap)
  - 완전이진트리 구조
  - 최대 힙, 최소 힙

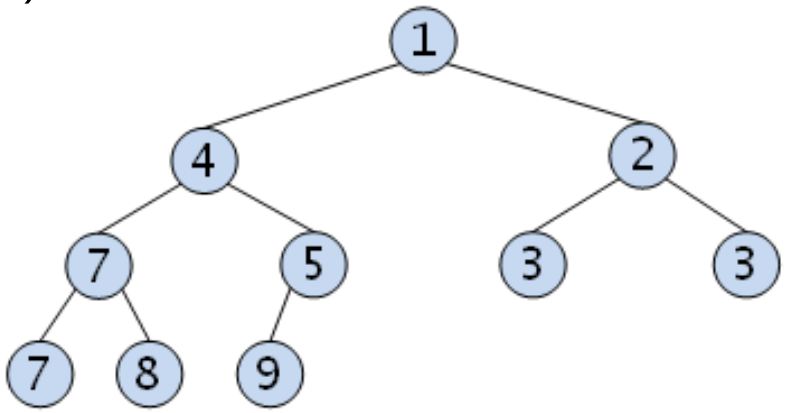
- 최대 힙(max heap)
  - 부모 노드의 키값이 자식 노드의 키값보다 크거나 같은 완전 이진 트리
- 최소 힙(min heap)
  - 부모 노드의 키값이 자식 노드의 키값보다 작거나 같은 완전 이진 트리

최대힙과 최소힙

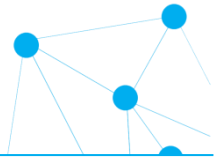
- 최대 힙(max heap)
  - key(부모노드) ≥ key(자식노드)



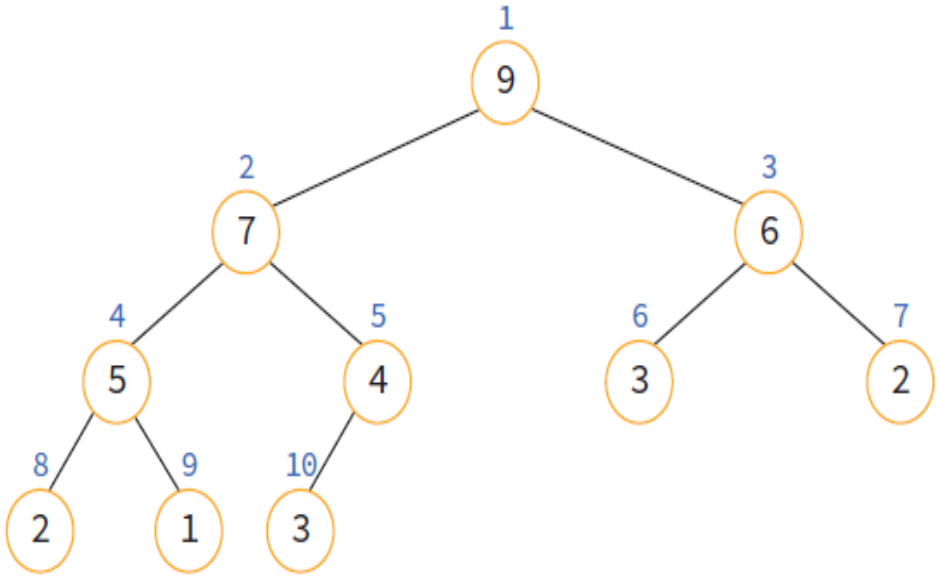
- 최소 힙(min heap)
  - key(부모노드) ≤ key(자식노드)



힙의 구현 : 배열 이용 구현



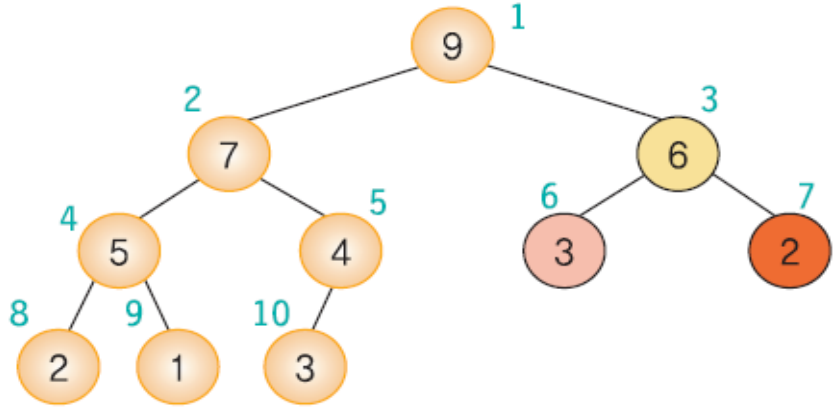
- 힙은 완전이진트리므로 보통 배열을 이용하여 구현하는 것이 효과적
  - 완전이진트리 ➔ 각 노드에 번호를 붙임 ➔ 배열의 인덱스

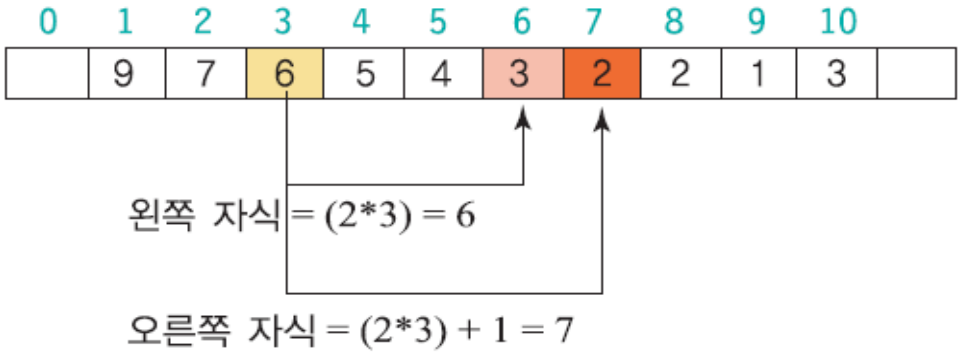


0	1	2	3	4	5	6	7	8	9	10	
	9	7	6	5	4	3	2	2	1	3	

힙의 구현

- 부모노드와 자식노드의 관계
  - 왼쪽 자식의 인덱스 = (부모의 인덱스)*2
  - 오른쪽 자식의 인덱스 = (부모의 인덱스)*2 + 1
  - 부모의 인덱스 = (자식의 인덱스)/2





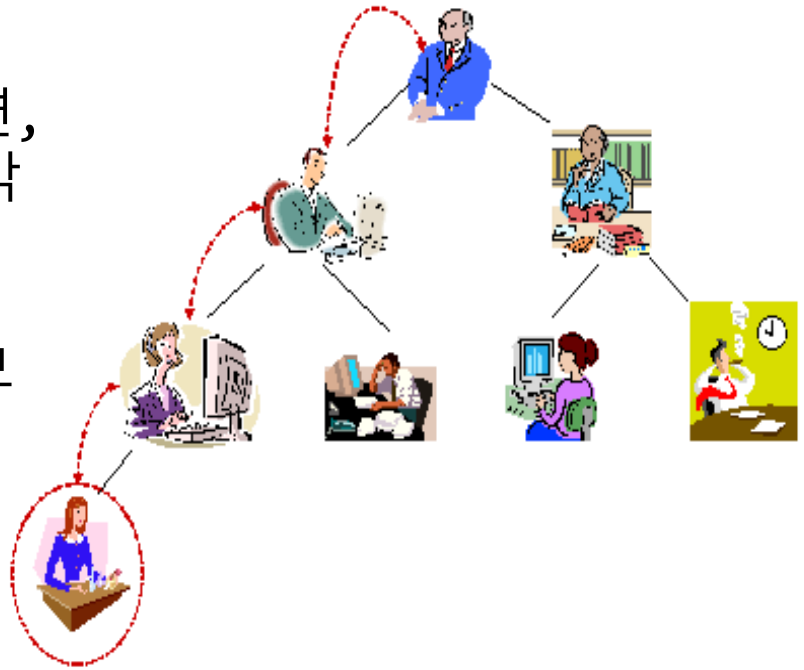
08-3. 힙의 연산

삽입 연산

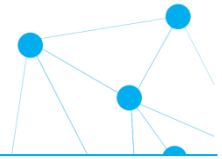
- **Upheap**
  - 힙에 있어서 삽입 연산은 회사에서 신입 사원이 들어오면 일단 말단 위치에 앉힌 다음에, 신입 사원의 능력을 봐서 위로 승진시키는 것과 비슷하다.

(1) 힙에 새로운 요소가 들어 오면, 일단 새로운 노드를 힙의 마지막 노드에 이어서 삽입

(2) 삽입 후에 새로운 노드를 부모 노드들과 교환해서 힙의 성질을 만족



Upheap 동작 알고리즘

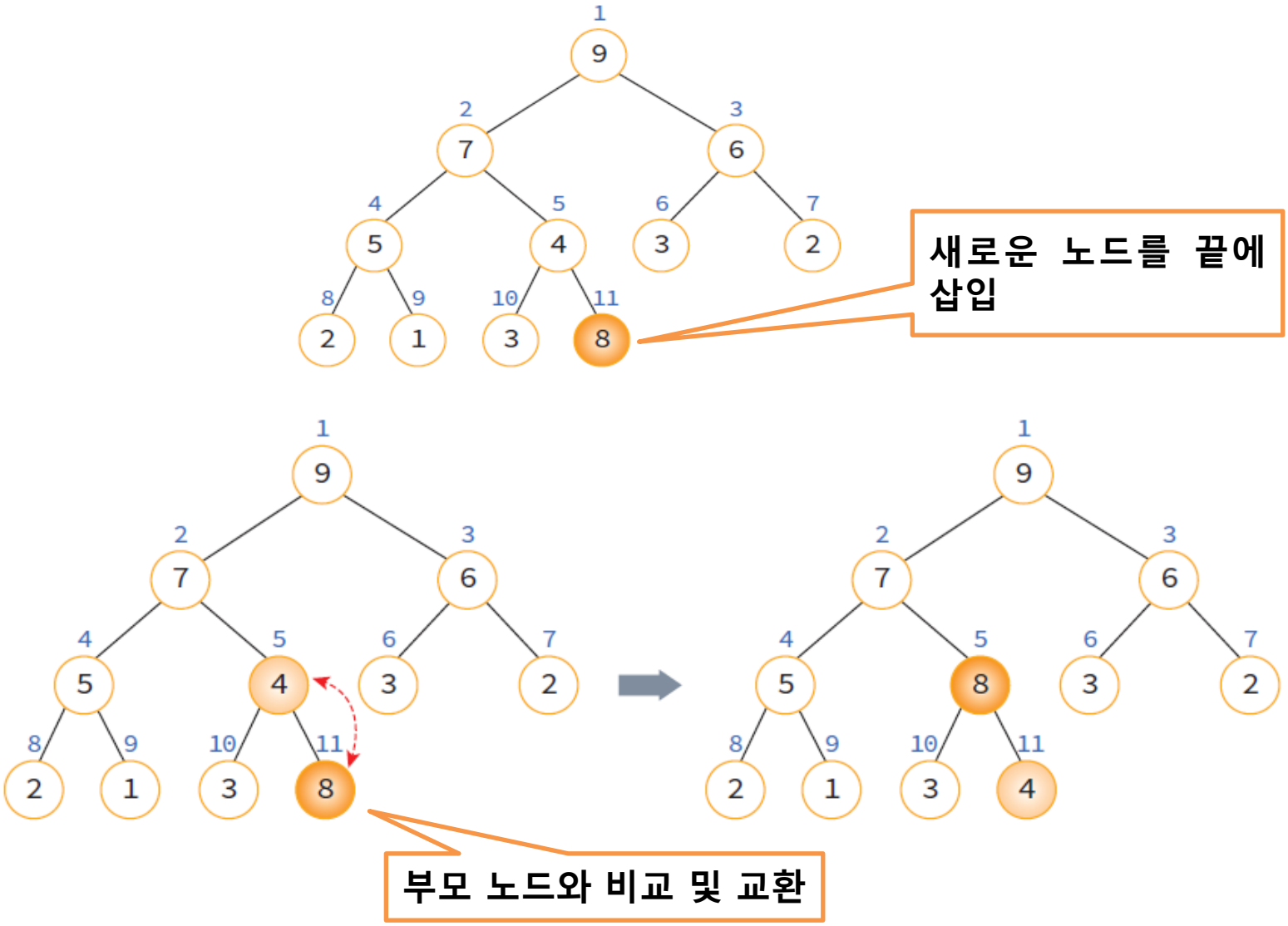


- 삽입된 노드에서 루트까지의 경로에 있는 노드들을 비교/교환
- 힙의 성질을 복원
  - 키 k가 부모 노드보다 작거나 같으면 upheap은 종료한다

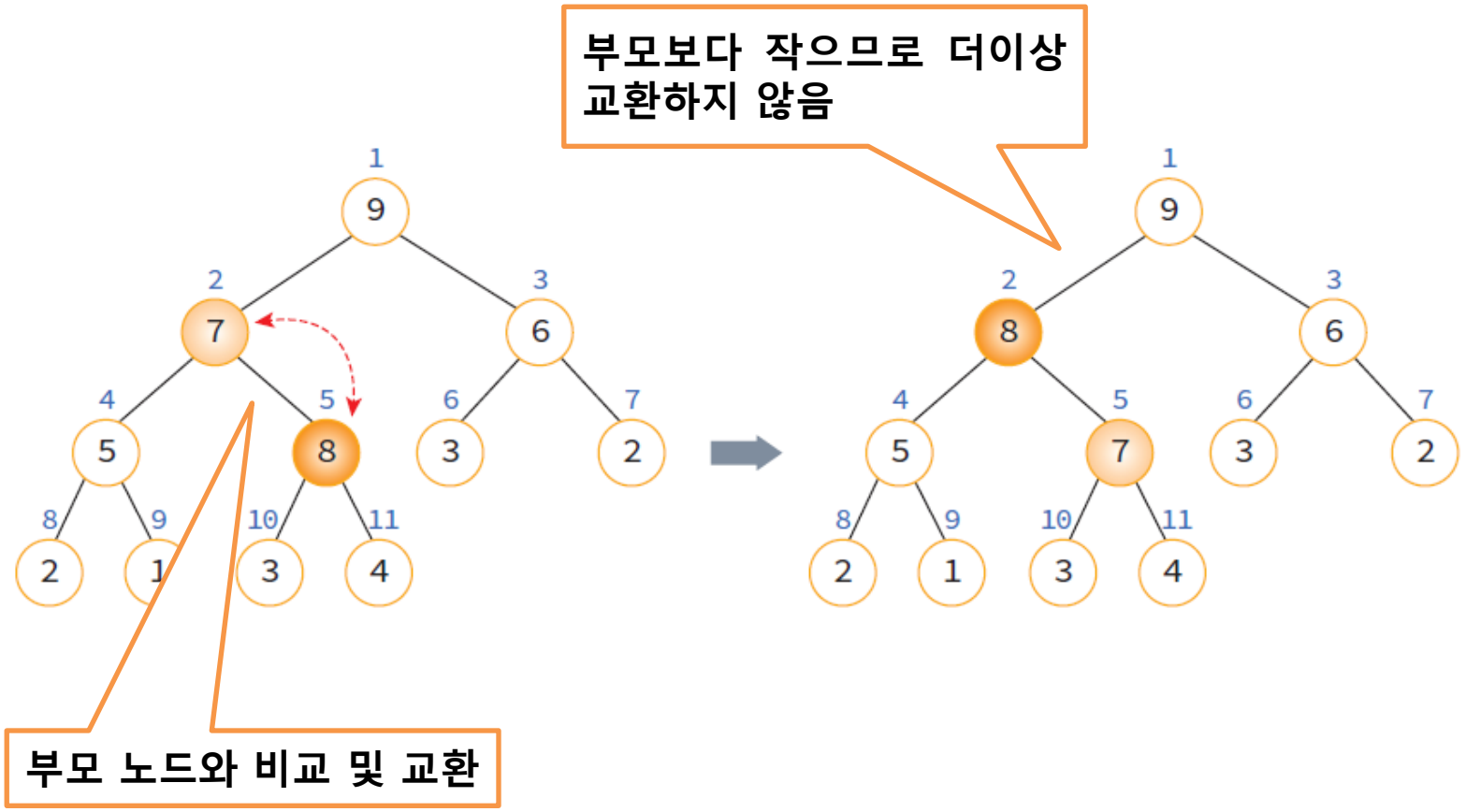
```
insert_max_heap(A, key) :
heap_size ← heap_size + 1;
i ← heap_size;
A[i] ← key;
while i ≠ 1 and A[i] > A[PARENT(i)]
            A[i] ↔ A[PARENT];
            i ← PARENT(i);
```

힙트리 A에 새로운 키 key삽입

Upheap 과정 예


1
9
2
7
3
6
4
5
5
4
6
3
7
2
8
2
9
1
10
3
11
8
새로운 노드를 끝에 삽입
1
9
2
7
3
6
4
5
5
4
6
3
7
2
8
2
9
1
10
3
11
8
부모 노드와 비교 및 교환
1
9
2
7
3
6
4
5
5
8
6
3
7
2
8
2
9
1
10
3
11
4

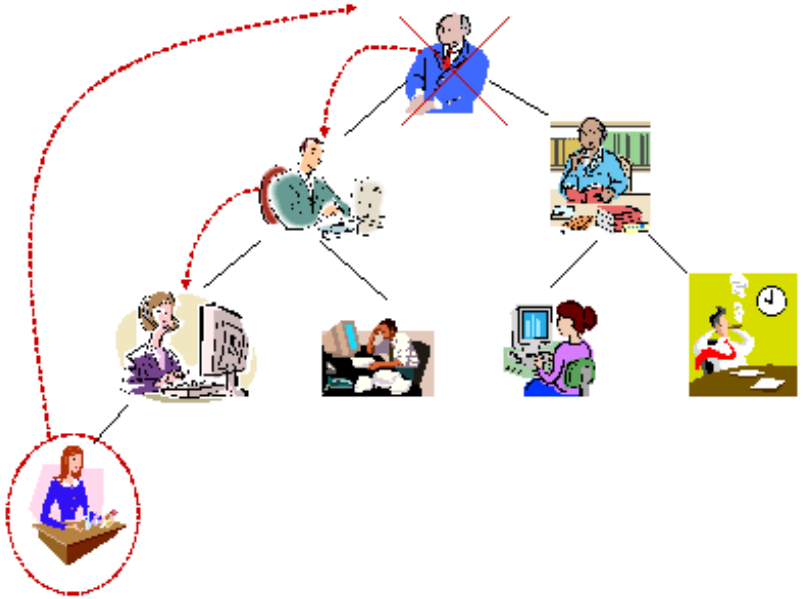
Upheap 과정 예



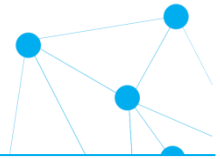
삭제 연산

- 힙 ➔ 우선순위큐 ➔ 우선순위가 높은 노드 삭제 ➔ 최대힙에서의 삭제 ➔ 항상 루트가 삭제됨
  - 가장 큰 키 값을 가진 노드를 삭제하는 것
- **Downheap**
  - 삭제 연산은 회사에서 사장의 자리가 비게 되면 먼저 제일 말단 사원을 사장 자리로 올린 다음에, 능력에 따라 강등시키는 것과 비슷하다.

(1) 루트 삭제
(2) 마지막 노드를 루트자리로 이동
(3) 루트에서부터 단말 노드까지의 경로에 있는 노드들을 교환하여 힙 성질을 만족

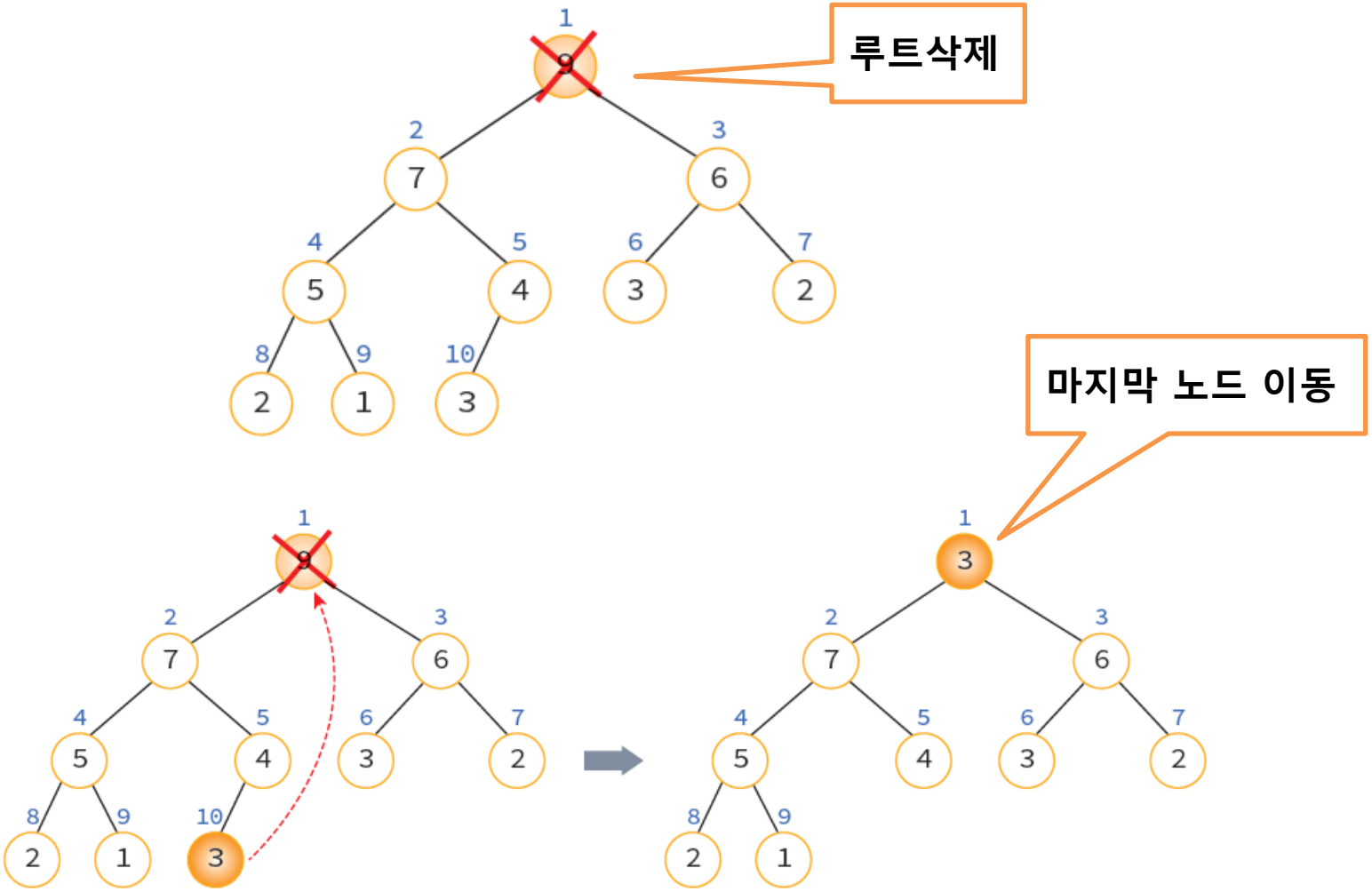


Downheap 동작 알고리즘

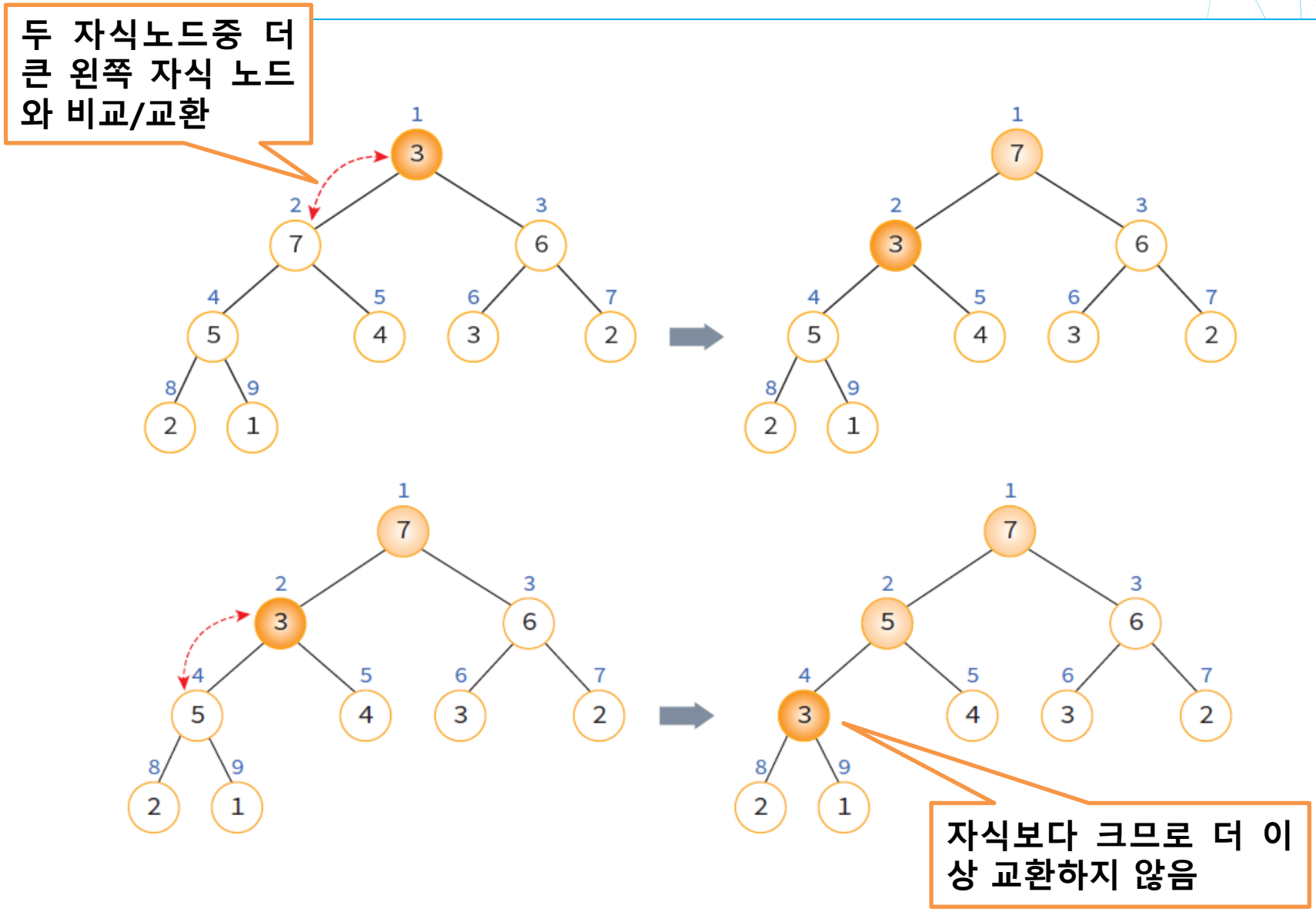


```
delete_max_heap(A):

item ← A[1];
A[1] ← A[heap_size];
heap_size ← heap_size-1;
i ← 2;

while i ≤ heap_size
        if i < heap_size and A[i+1] > A[i] then
                largest ← i+1;
        else
                largest ← i;
        if A[PARENT(largest)] > A[largest] then
                break;

        A[PARENT(largest)] ↔ A[largest];
        i ← CHILD(largest);

return item;
```

Downheap 과정 예

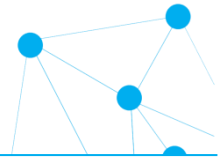


Downheap 과정 예



08-4. 힙 응용

힙응용 : 힙 정렬



- 힙을 이용하면 정렬 가능 : 힙 정렬
  - 먼저 정렬해야 할 n개의 요소들을 최대 힙에 삽입
  - 한번에 하나씩 요소를 힙에서 삭제하여 저장하면 정렬됨

- 힙정렬이 유용한 경우
  - 전체의 정렬이 아니라 가장 큰 값 몇 개만 필요할 때

힙응용 : 허프만 코드 트리

- 이진트리는 각 글자의 빈도가 알려져 있는 메시지의 내용을 압축하는데 사용될 수 있음
- 이런 종류의 이진트리 ➔ 허프만 코드 트리

A	80
B	16
C	32
D	36
E	123
F	22
G	16
H	51
I	71
...	
Z	1

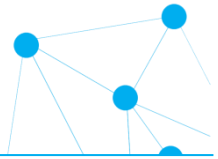
문자의 빈도수

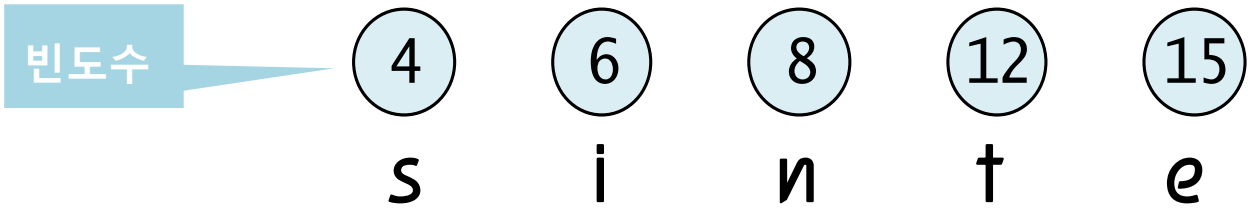
- 빈도수가 알려진 문자에 대한 고정길이코드와 가변길이 코드의 비교

글자	빈도수	고정길이코드			가변길이코드		
		코드	비트수	전체 비트수	코드	비트수	전체 비트수
A	17	0000	4	68	00	2	34
B	3	0001	4	12	11110	5	15
C	6	0010	4	24	0110	4	24
D	9	0011	4	36	1110	4	36
E	27	0100	4	108	10	2	54
F	5	0101	4	20	0111	4	20
G	4	0110	4	16	11110	5	20
H	13	0111	4	52	010	3	39
I	15	1000	4	60	110	3	45
J	1	1001	4	4	11111	5	5
합계	100			400			292

빈도가 높으면 적은 비트 할당, 빈도가 낮으면 많은 비트 할당 전체적인 비트수를 줄일 수 있다. 이를 통해 압축

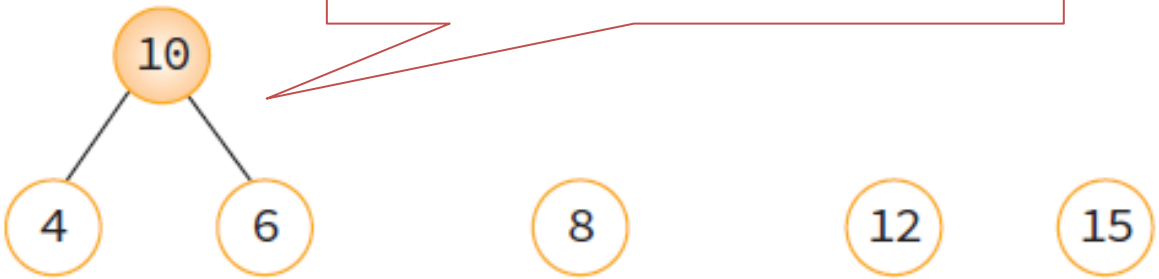
허프만 코드 생성 절차



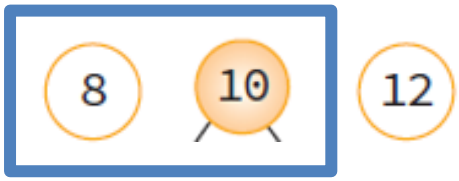


가장 적은 빈도수 두 글자 선택 후 이진트리 구성. 루트는 두 단말의 값을 합한 결과

적은 값을 얻는 과정에 최소힙 사용



허프만 코드 생성 절차

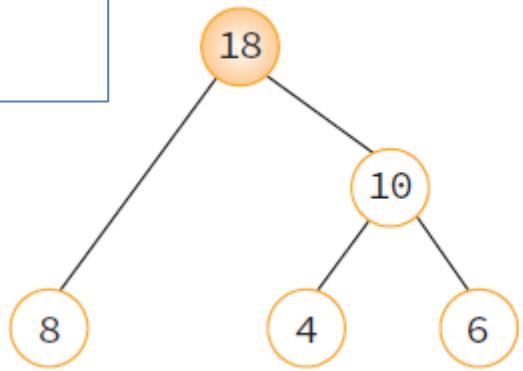


12

15

앞서 생성된 노드를 추가하여 기존 값들과 비교/정렬

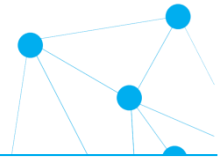
적은 값 두 개를 다시 선택하여 이진 트리 구성

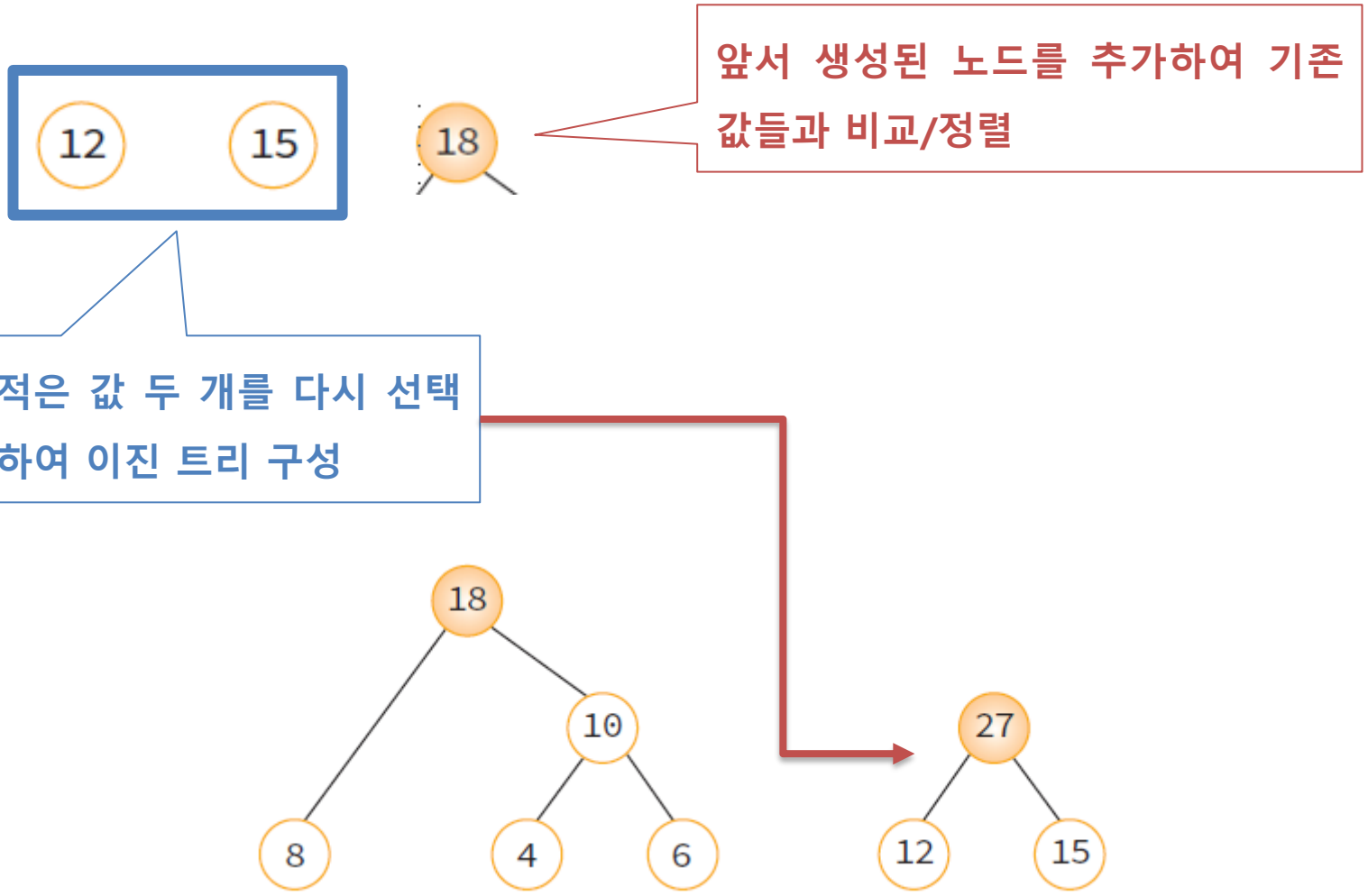


12

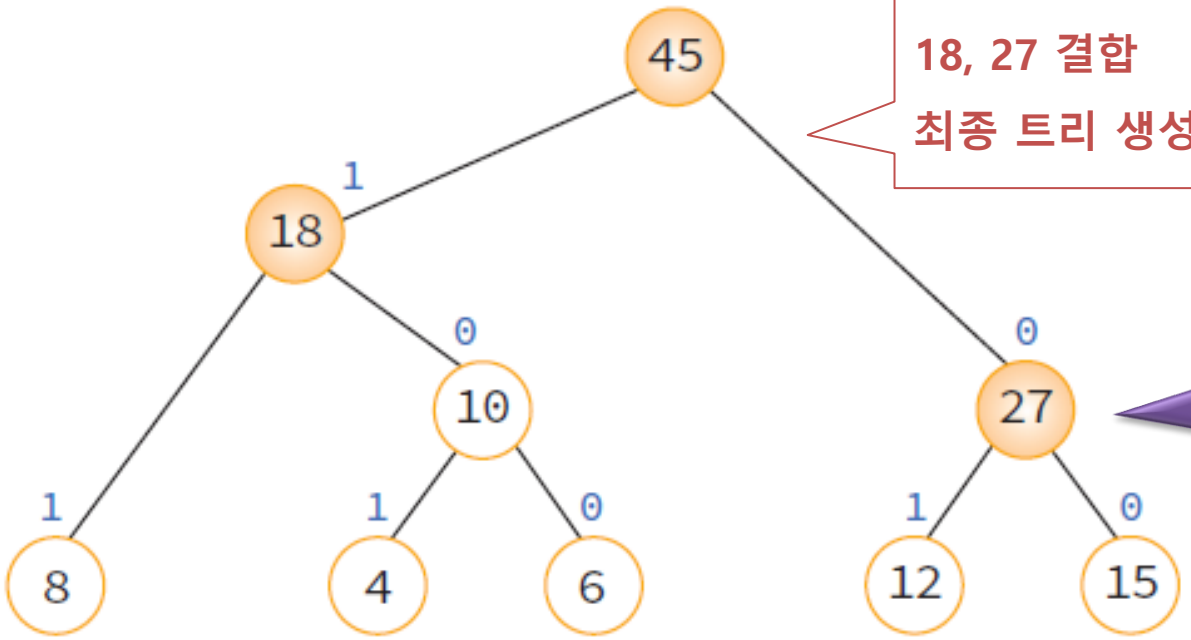
15

허프만 코드 생성 절차




12
15
18
앞서 생성된 노드를 추가하여 기존 값들과 비교/정렬
적은 값 두 개를 다시 선택하여 이진 트리 구성
18
10
8
4
6
27
12
15

허프만 코드 생성 절차



18, 27 결합
최종 트리 생성

오른쪽 간선은 비트 0,
왼쪽 간선은 비트 1

4	6	8	12	15
s	i	n	t	e
101	100	11	01	00

최종 할당된 비트들