QUEUE

큐

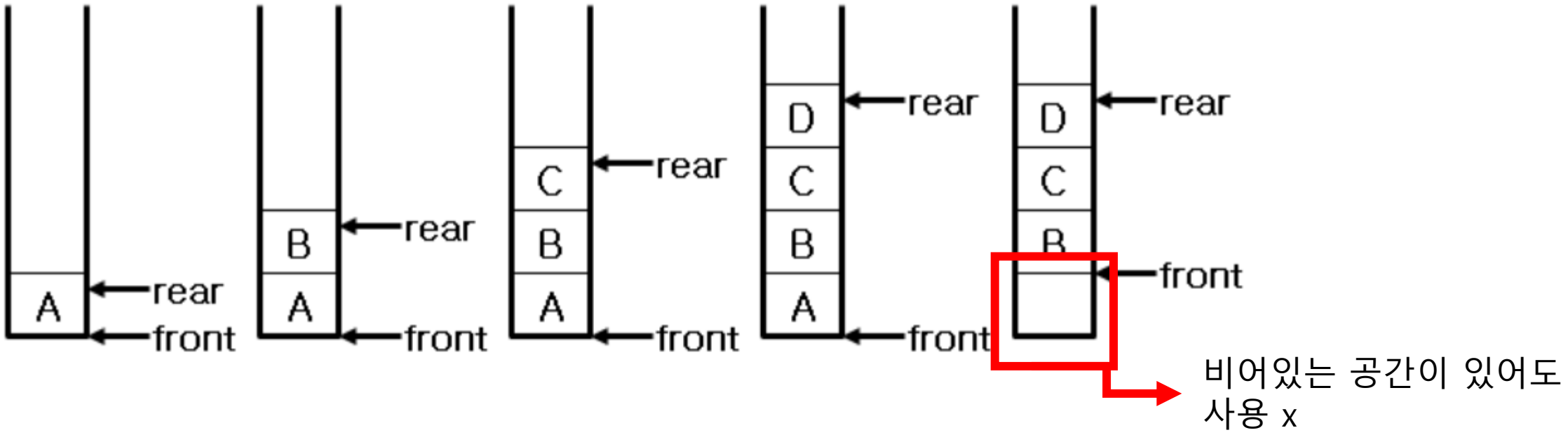
큐
먼저 들어간 데이터를 먼저 꺼내는 방식(FIFO: 선입선출)의 구조

용어
rear : 꼬리, 삽입이 이루어짐 (=Enqueue)
front: 머리, 삭제가 이루어짐 (=Dequeue)

https://monsieursongsong.tistory.com/5

선형 큐의 문제점

인덱스를 감소하지 않고 증가하는 방식
=> 데이터가 없어도 사용 X



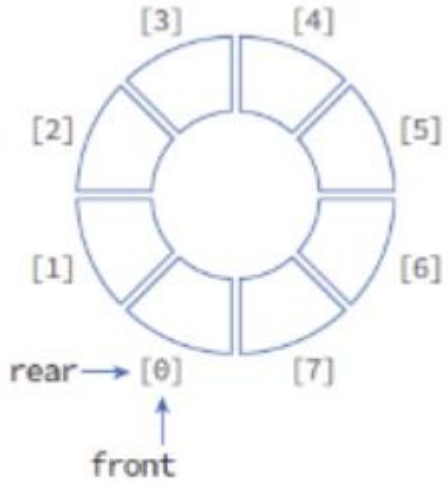
https://m.blog.naver.com/nabilera1/222062019192

큐의 멤버 함수

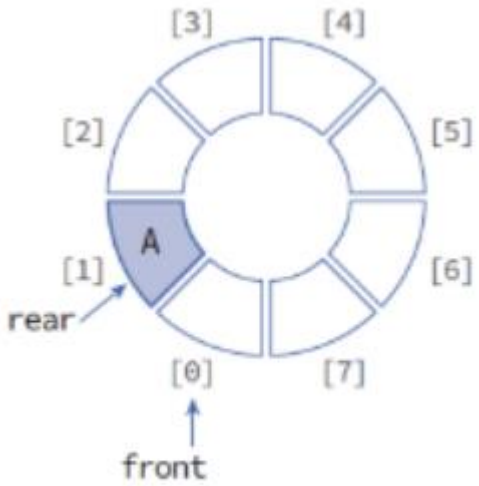
- `q.front()` : q의 맨 앞(front)의 원소 리턴
- `q.back()` : q의 맨 뒤(rear)의 원소 리턴
- `q.push(x)` : q의 맨 뒤에 x 원소 추가
- `q.pop()` : q의 맨 앞(front)의 원소 삭제
- `q.size()` : q의 사이즈(원소의 개수) 리턴
- `q.empty()` : q의 사이즈(원소의 개수)가 0인지 아닌지 확인 (q가 비어있다면 1, 아니라면 0을 리턴)

https://cocoon1787.tistory.com/237

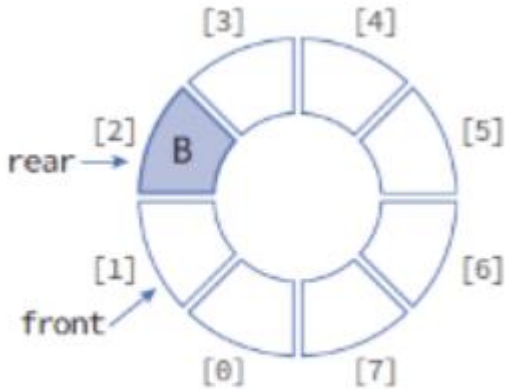
원형 큐



(a) 초기상태



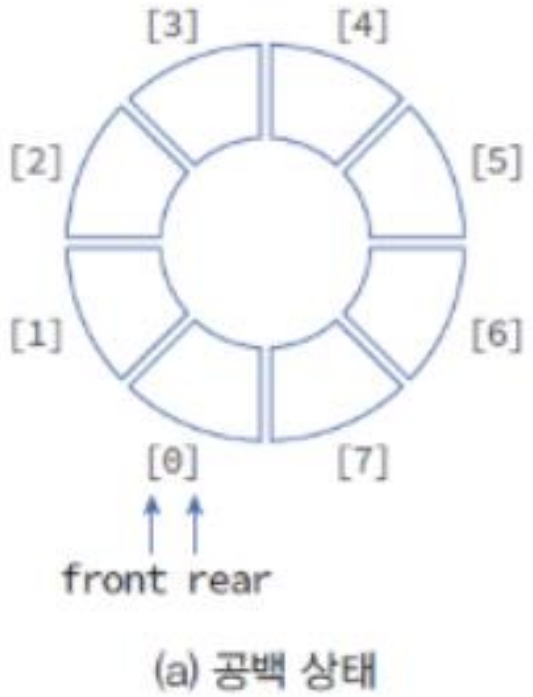
(b) A 삽입



(d) 삭제

https://yjg-lab.tistory.com/115

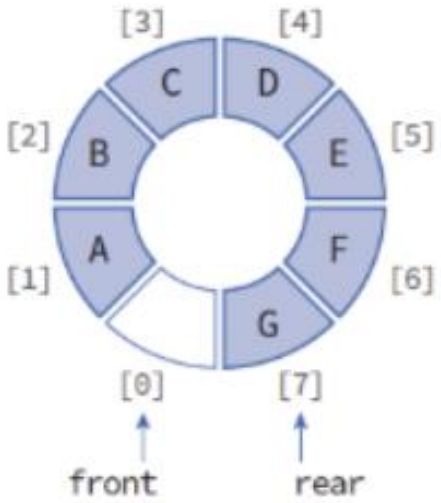
공백 판단



(a) 공백 상태

```
int is_empty()
{
    return (front == rear)
}
```

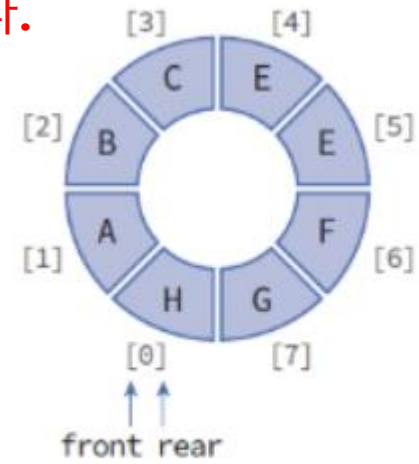
포화 판단



(b) 포화 상태

```
Int is_full()
{
        return (front == ((rear+1)%
MAX_QUEUE_SIZE)))
}
```

공백과 포화상태를 구별하기 위해 하나의 공간을 비워둔다.



(c) 오류 상태

원형 큐 구현(1)

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_QUEUE_SIZE 5

typedef int element;
typedef struct {  // 원형 큐 타입
    element  data[MAX_QUEUE_SIZE];
    int  front, rear;
} QueueType;
// 오류 함수
void error(char *message)
{
    fprintf(stderr, "%s\n", message);
    exit(1);
}
// 원형 큐 초기화
void init_queue(QueueType *q)
{
    q->front = q->rear = 0  ; // 0이 아니라 다른 수로 해도 상관 없습니다. 원형이기 때문입니다.
}
// 공백 상태 검출 함수
int is_empty(QueueType *q)
{
    return ( q->front == q->rear );
}
// 포화 상태 검출 함수
int is_full(QueueType *q)
{
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}
```

원형 큐 구현(2)

```c
// 원형 큐 출력 함수
void queue_print(QueueType *q)
{
    printf("QUEUE(front=%d rear=%d) = ", q->front, q->rear);
    if (!is_empty(q)) {
        int i = q->front;
        do {
            i = (i + 1) % (MAX_QUEUE_SIZE);
            printf("%d | ", q->data[i]);
        } while ( i == q->rear );
    }
    printf("\n");
}
// 삽입 함수
void enqueue(QueueType *q, element item)
{
    if (is_full(q))
        error("큐가 포화상태입니다");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    q->data[ q->rear ] = item;
}
// 삭제 함수
element dequeue(QueueType *q)
{
    if (is_empty(q))
        error("큐가 공백상태입니다");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[ q->front];
}
```

메인 코드
와
실행

```c
// 메인
int main(void)
{
    QueueType queue;
    int element;

    init_queue(&queue);
    printf("--데이터 추가 단계--\n");
    while (!is_full(&queue))
    {
        printf("정수를 입력하시오: ");
        scanf("%d",  &element  );
        enqueue(&queue, element);
        queue_print(&queue);
    }
    printf("큐는 포화상태입니다.\n\n");

    printf("--데이터 삭제 단계--\n");
    while (!is_empty(&queue))
    {
        element = dequeue(&queue);
        printf("꺼내진 정수: %d \n", element);
        queue_print(&queue);
    }
    printf("큐는 공백상태입니다.\n");
    return 0;
}
```

```
--데이터 추가 단계--
정수를 입력하시오: 10
QUEUE(front=0 rear=1) = 10 ¦
정수를 입력하시오: 20
QUEUE(front=0 rear=2) = 10 ¦ 20 ¦
정수를 입력하시오: 30
QUEUE(front=0 rear=3) = 10 ¦ 20 ¦ 30 ¦
정수를 입력하시오: 40
QUEUE(front=0 rear=4) = 10 ¦ 20 ¦ 30 ¦ 40 ¦
큐는 포화상태입니다.

--데이터 삭제 단계--
꺼내진 정수: 10
QUEUE(front=1 rear=4) = 20 ¦ 30 ¦ 40 ¦
꺼내진 정수: 20
QUEUE(front=2 rear=4) = 30 ¦ 40 ¦
꺼내진 정수: 30
QUEUE(front=3 rear=4) = 40 ¦
꺼내진 정수: 40
QUEUE(front=4 rear=4) =
큐는 공백상태입니다.
```

TREE

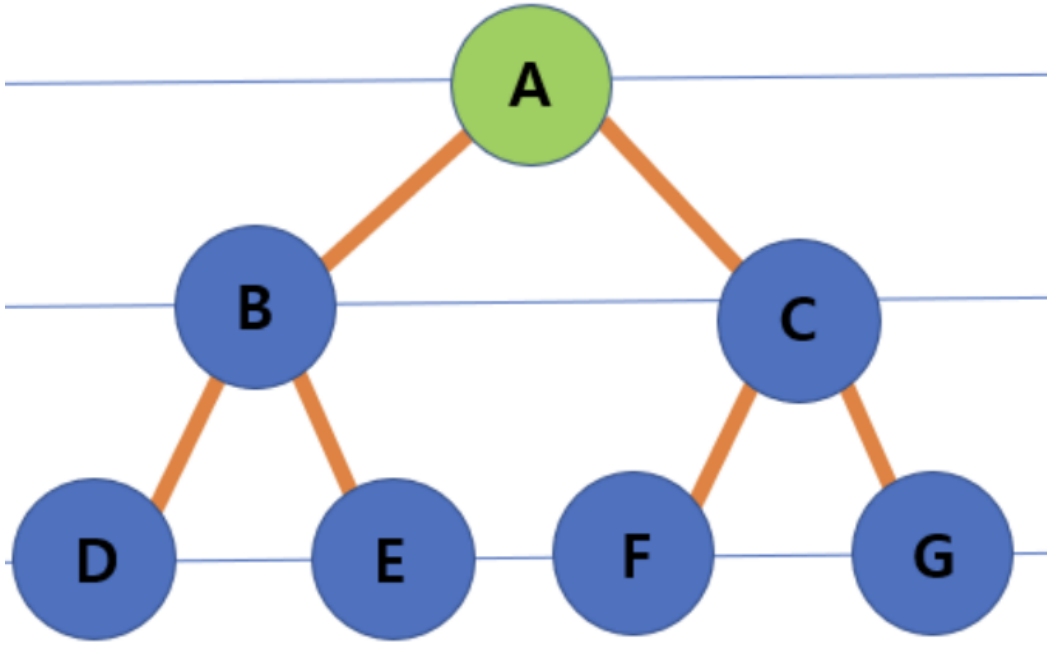
트리

트리

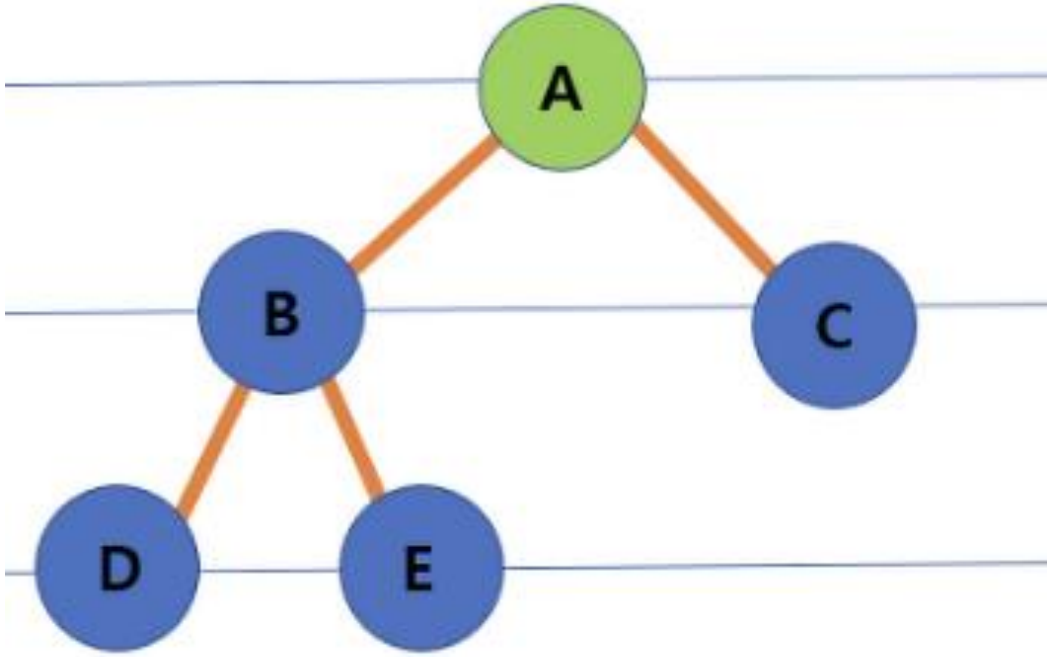
- 계층적인 구조
- 사이클을 포함하지 않음 (부모-자식 관계)
- 특별 노드(루트)가 존재
- 서로 독립적인 서브트리를 가짐

이진 트리

: 자식 노드가 최대 2개



포화 이진 트리

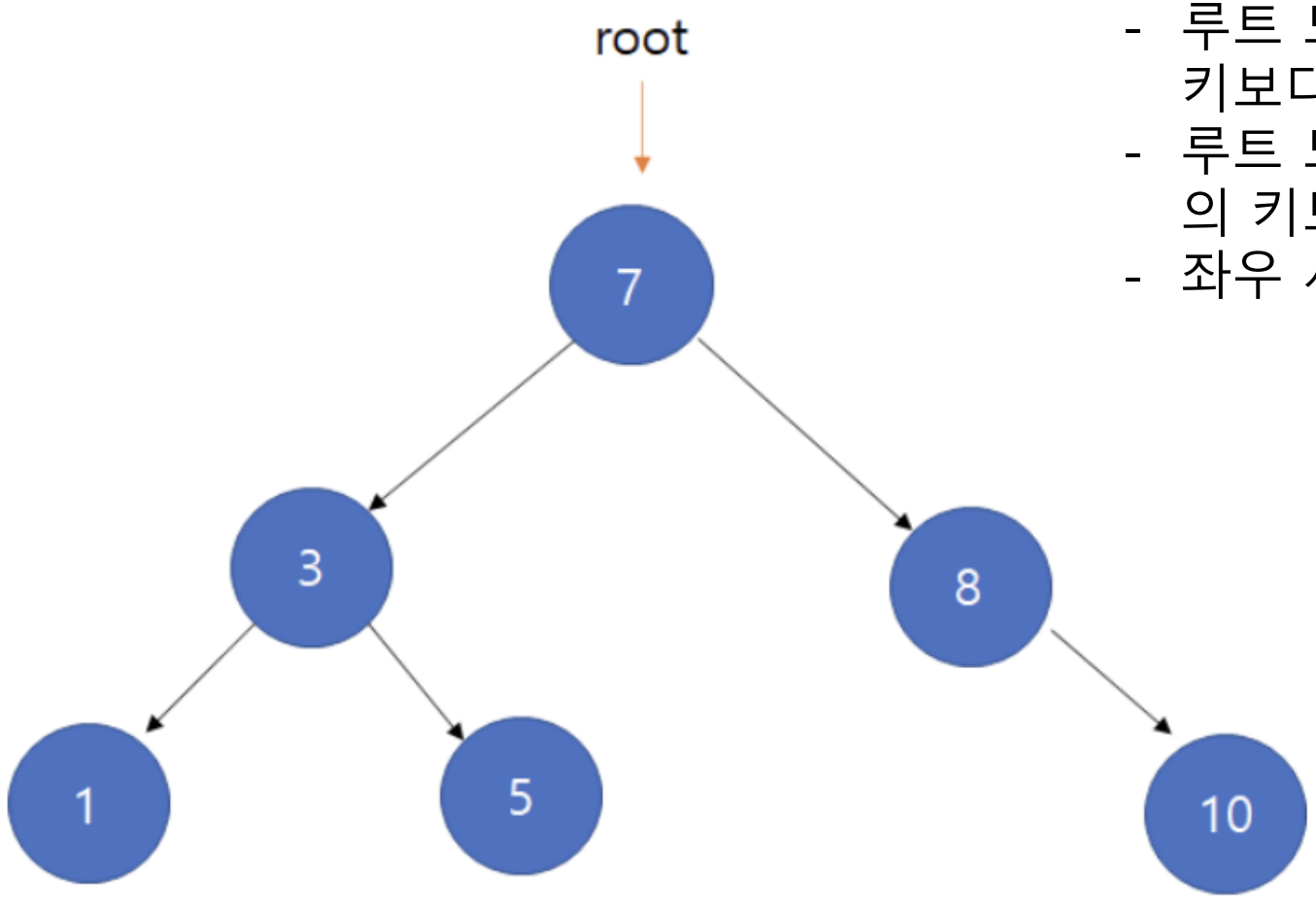


완전 이진 트리

왼쪽부터 차례대로 채워진 트리

https://hsc-tech.tistory.com/7

이진 탐색 트리



이진 탐색 트리(Binary Search Tree)

조건

- 중복되지 않는 키
- 루트 노트의 왼쪽 서브 트리는 해당 노드의 키보다 작은 키를 가짐
- 루트 노드의 오른쪽 서브 트리는 해당 노드의 키보다 큰 키를 갖는 노드들로 이루어짐
- 좌우 서브 트리도 모두 이진 탐색 트리

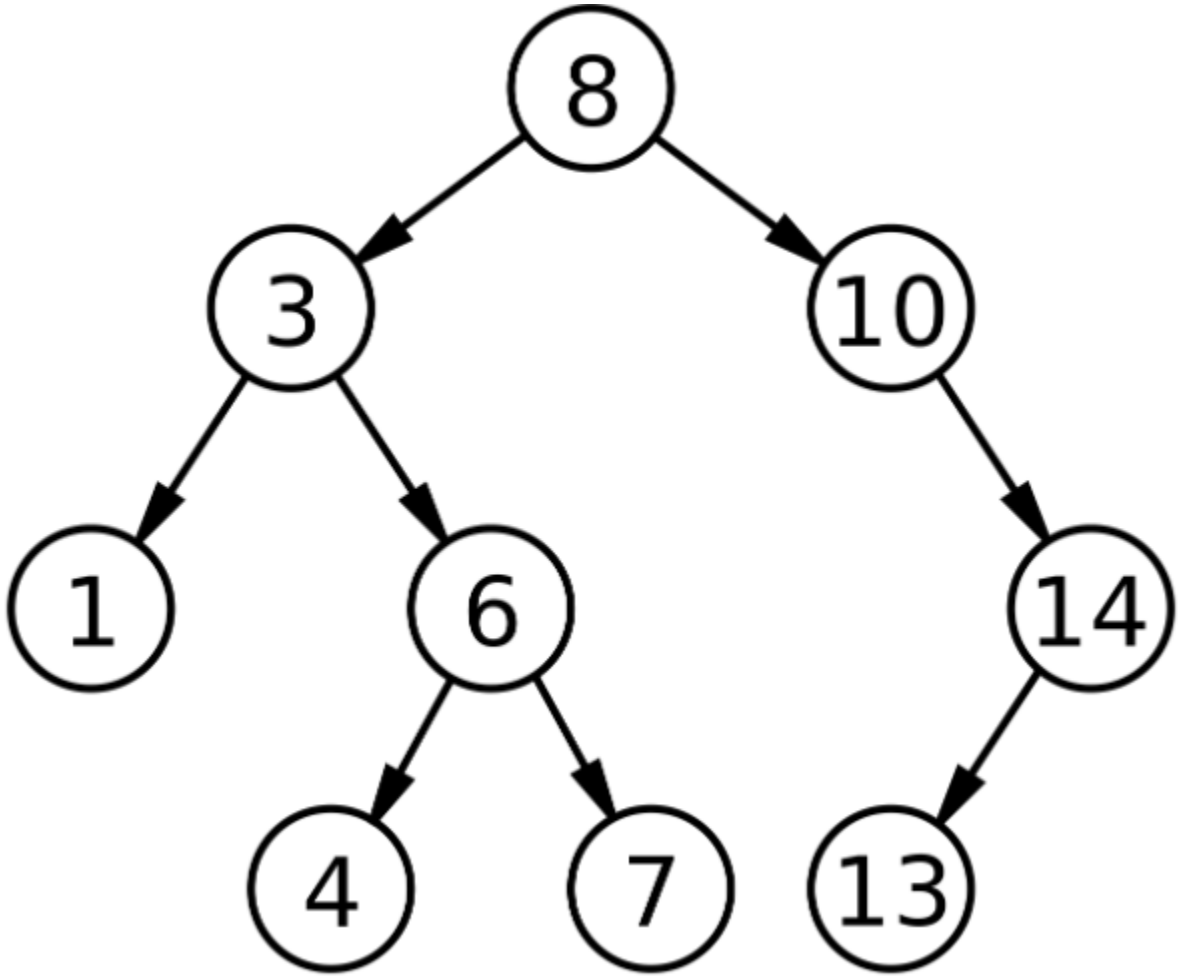
https://kangworld.tistory.com/51

순회 방법

전위 순회 : 루트 -> 왼쪽 -> 오른쪽
8 – 3 – 1 - 6 - 4 - 7 - 10 - 14 – 13

중위 순회 : 왼쪽 -> 루트 -> 오른쪽
1 - 3 - 4 - 6 - 7 - 8 - 10 - 13 – 14

후위 순화 : 왼쪽 -> 오른쪽 -> 루트
1 - 4 - 7 - 6 - 3 - 13 - 14 - 10 - 8

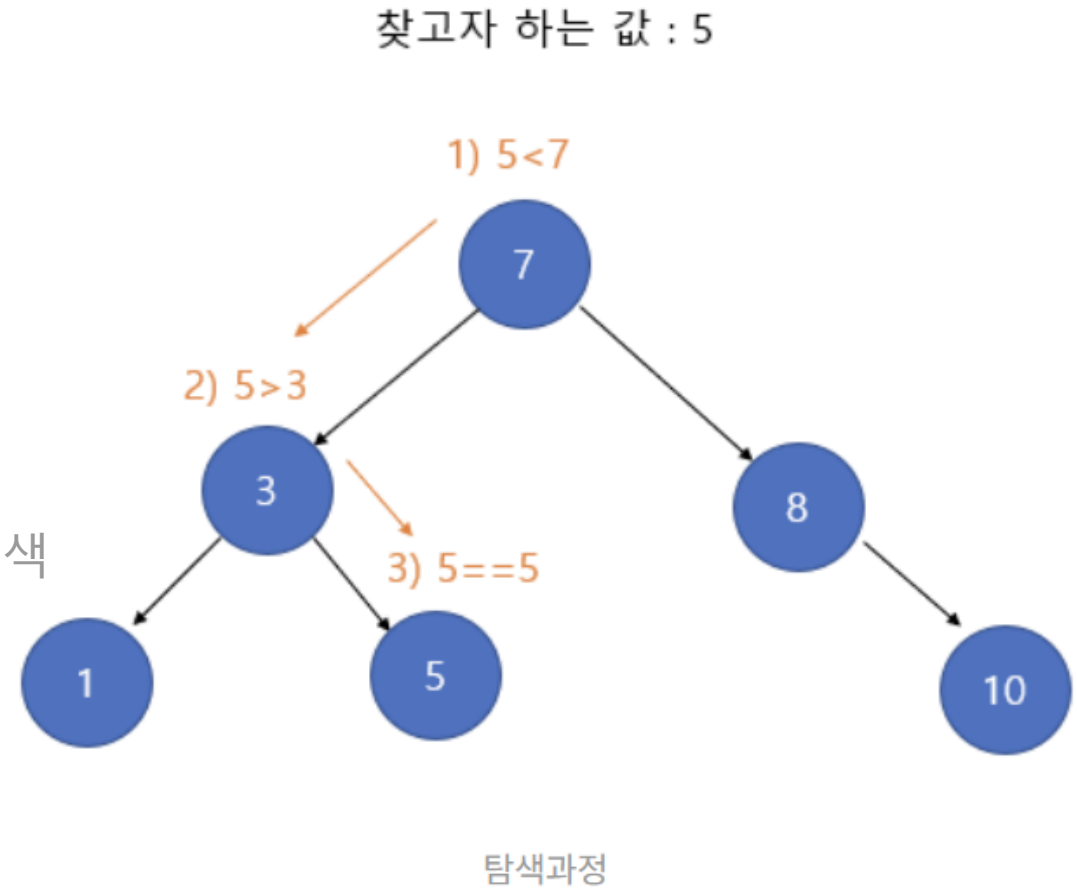


https://kangworld.tistory.com/51

탐색

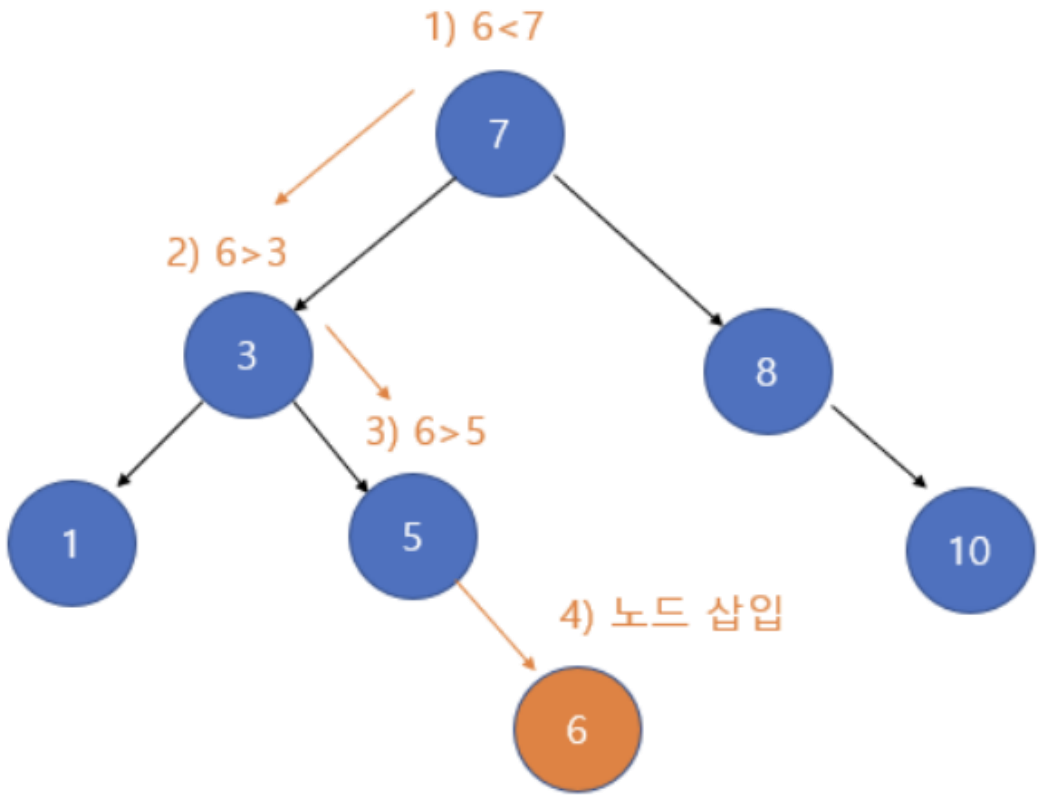
```
TreeNode* search(TreeNode* root, int key){
    if(root == NULL){    // 값을 찾지 못한 경우
        return NULL;
    }

    if(key == root->key){    // 값을 찾음
        return root;
    }
    else if(key < root->key){    // 왼쪽 서브트리 탐색
        search(root->left, key);
    }
    else if(key > root->key){    // 오른쪽 서브트리 탐색
        search(root->right, key);
    }

}
```

출처: https://code-lab1.tistory.com/10 [코드 연구소:티스토리]



탐색과정

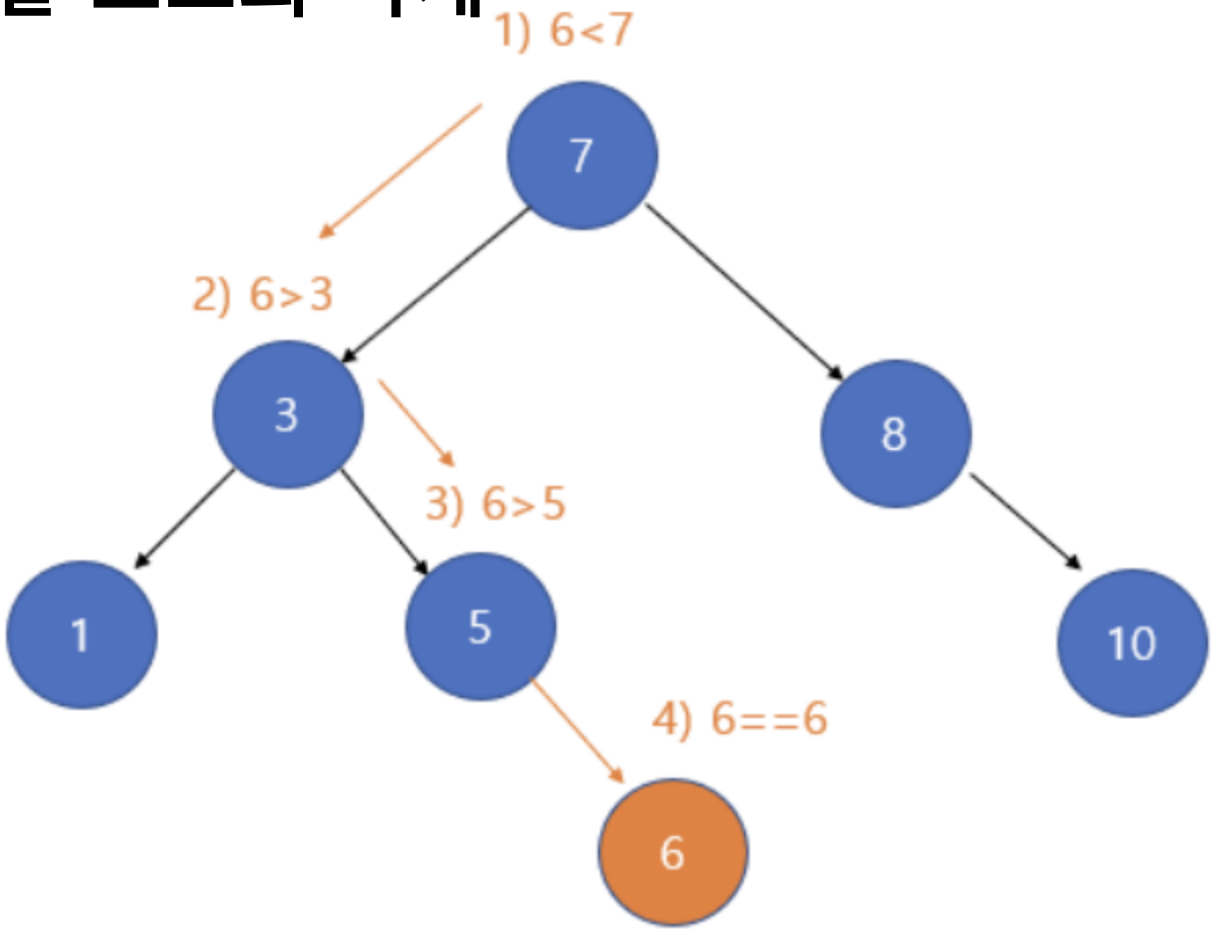
삽입




```
void insert(TreeNode** root, int key){
    TreeNode* ptr;     // 탐색을 진행할 포인터
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));    // newNode 생성
    newNode->key = key;
    newNode->left = newNode->right = NULL;

    if(*root == NULL){    // 트리가 비어 있을 경우
        *root = newNode;
        return;
    }

    ptr = *root;    // root 노드부터 탐색 진행

    while(ptr){
        if(key == ptr->key){    // 중복값
            printf("Error : 중복값은 허용되지 않습니다!\n");
            return;
        }else if(key < ptr->key){    // 왼쪽 서브트리
            if(ptr->left == NULL){    // 비어있다면 추가
                ptr->left = newNode;
                return;
            }else{    // 비어있지 않다면 다시 탐색 진행
                ptr= ptr->left;
            }
        }else{    // key > ptr->key 오른쪽 서브트리
            if(ptr->right == NULL){    // 비어있다면 추가
                ptr->right = newNode;
                return;
            }else{    // 비어있지 않다면 다시 탐색 진행
                ptr = ptr->right;
            }
        }
    }

}
```

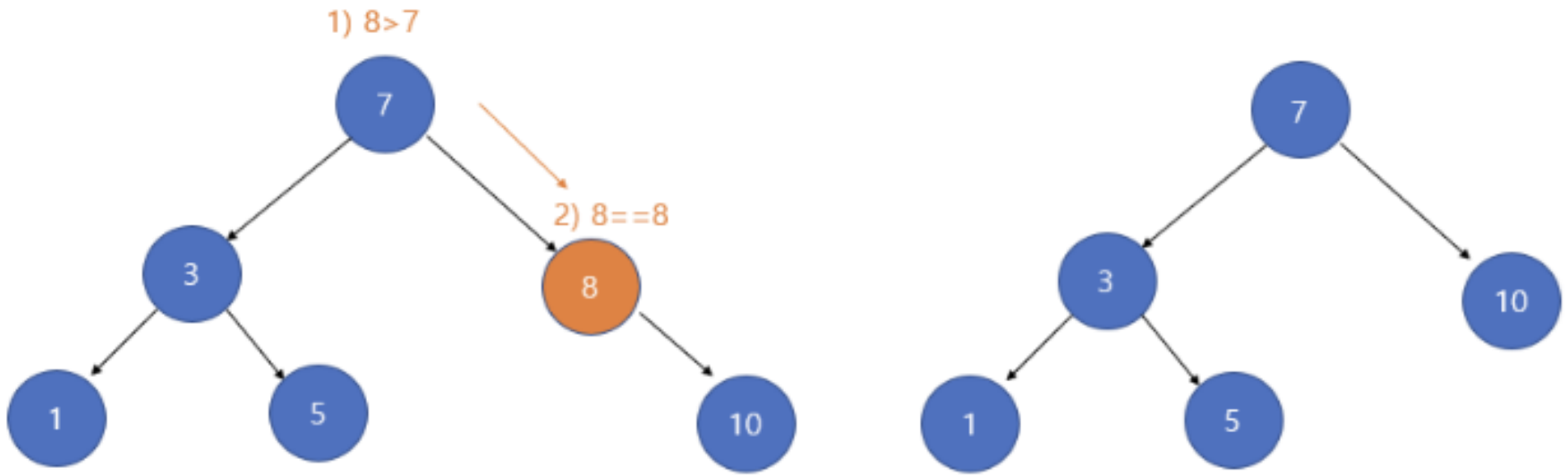
출처: https://code-lab1.tistory.com/10 [코드 연구소:티스토리]

삭제(1)
: 단말 노드의 삭제



삭제할 노드의 부모 노드가 있다면 부모 노드의 자식노드를 NULL 설정 후 , 삭제할 노드를 삭제 (메모리 해제)

삭제(2)
: 서브 트리의 노드일 경우

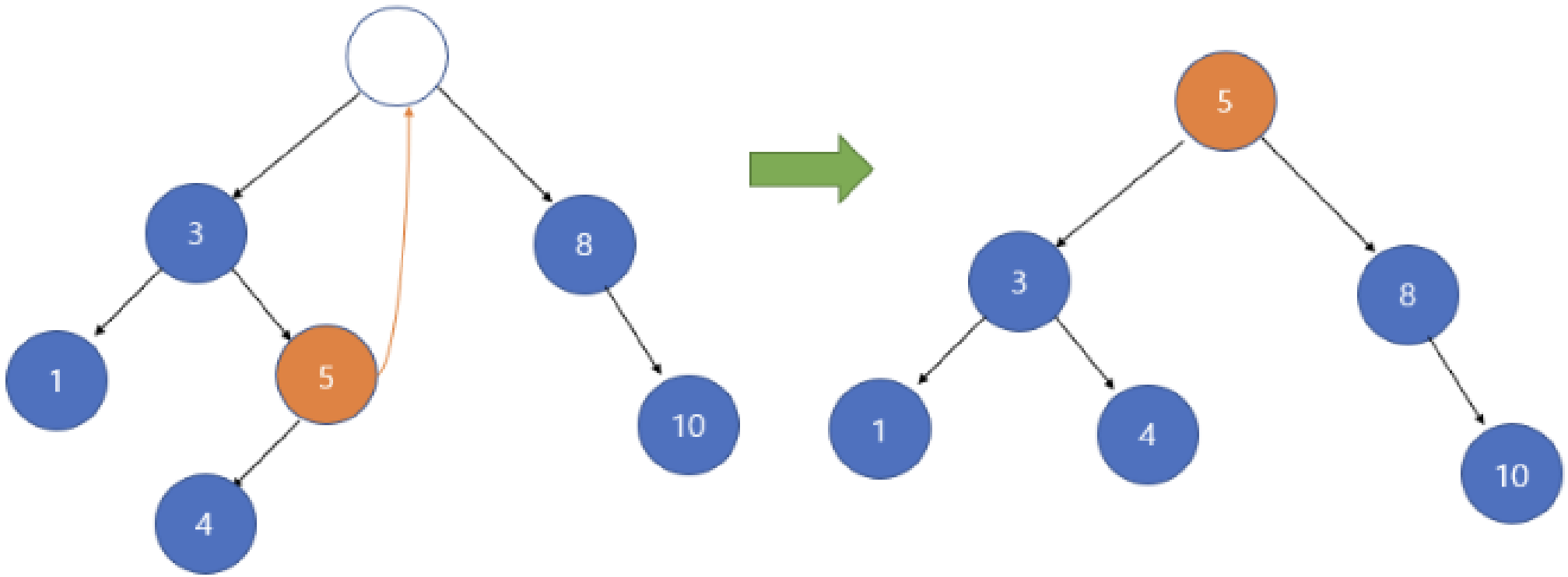


노드 삭제 과정

삭제할 노드의 자식 노드를 삭제할 노드의 부모 노드가 가리키도록 설정 후, 해당 노드 삭제

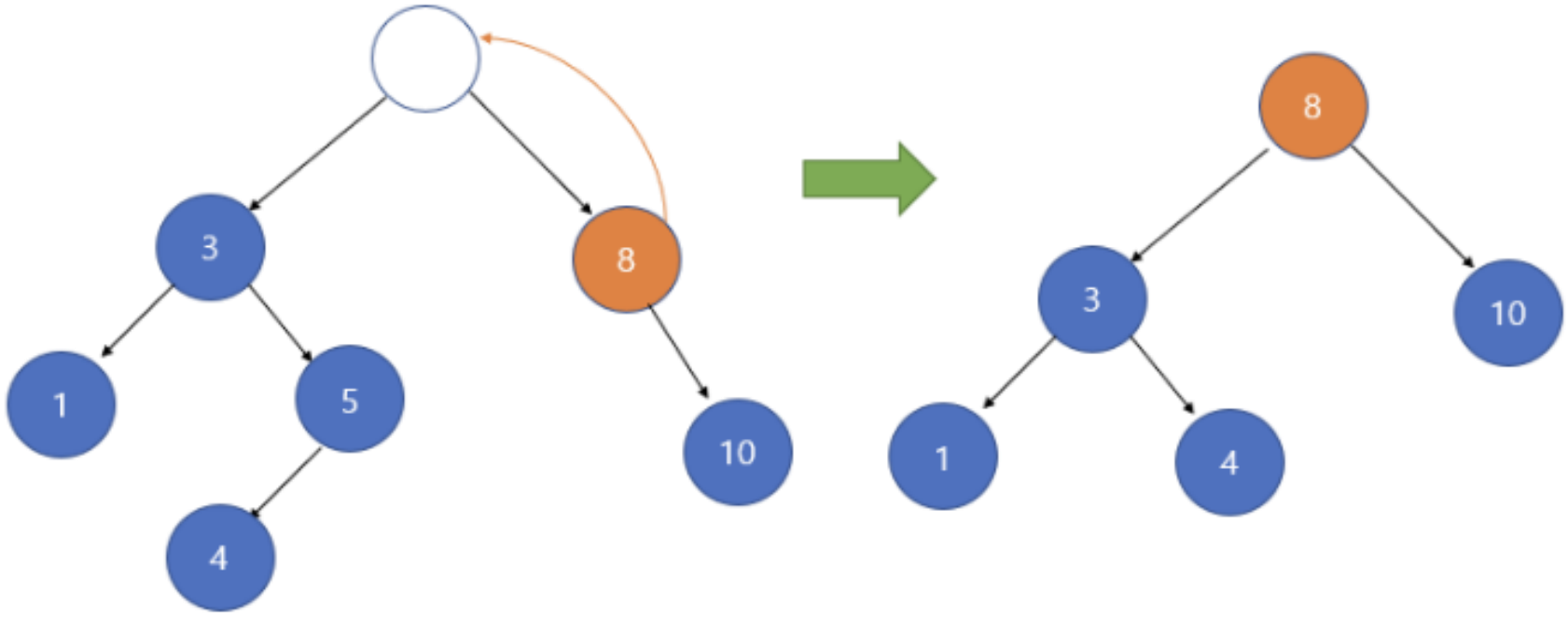
삭제(3)
: 루트 삭제

방법 1
삭제할 노드 왼쪽 서브 트리의 가장 큰 자손을 해당 노드에 올린다



삭제(3)
: 루트 삭제

방법 2
삭제할 노드 오른쪽 서브 트리의 가장 작은 자손을 해당 노드의 자리에 올린다



출처: https://code-lab1.tistory.com/10 [코드 연구소:티스토리]

Map

1) Map이란?

- map은 각 노드가 key와 value 쌍으로 이루어진 트리입니다. **특히, 중복을 허용하지 않습니다.**
- 따라서 map은 first, second가 있는 pair 객체로 저장되는데 first- key로 second- value로 저장됩니다.

2) MAP 기본 형태

- **map <key, value> map1;**

3) MAP 정렬

- **map은 자료를 저장할때 내부에서 자동으로 정렬합니다.**
- map은 key를 기준으로 정렬하며 **오름차순으로 정렬**합니다.
- 만약 **내림차순**으로 정렬하고 싶은 경우와 같이 사용하면 됩니다.

  map <int, int, greater> map1;

- (만약 다른 방법으로 int데이터를 내림차순으로 정렬하고 싶을 경우, 데이터에 -(마이너스)를 붙여 삽입하여 처리하면 내림차순으로 정렬됩니다.)

4) MAP 사용방법

1) 헤더 포함

- map을 사용하려면 헤더에 **#include <map>** 처리를 해야 합니다.

2) map 선언하기

- map의 기본 구조는 **map <key type, value type> 이름**입니다.

3) map에 찾고자 하는 데이터가 있는 지 확인하기(search)

- map에서 데이터를 찾을 때는 iterator을 사용합니다.
- 데이터를 끝까지 찾지 못했을 경우, iterator는 map.end()를 반환합니다.

4) MAP 사용방법

4) map에 데이터 삽입

- map은 중복을 허용하지 않습니다. insert를 수행할때, key가 중복되면 insert가 수행되지 않습니다.
- 중복되면 그것은 key의 역할을 제대로 하지 않습니다.

5) 반복문 데이터 접근 (first, second)

```
//인덱스기반
for (auto iter = m.begin() ; iter !=  m.end(); iter++)
{
        cout << iter->first << " " << iter->second << endl;
}
cout << endl;
```

4) MAP 사용방법

6) map에서 삭제하기

- map에서 데이터를 삭제하기 위해 활용할 함수는 erase와 clear입니다.
- `m.erase(m.begin()+2); m.erase("Alice");`
- `m.erase(m.begin(), m.end()); m.clear();`