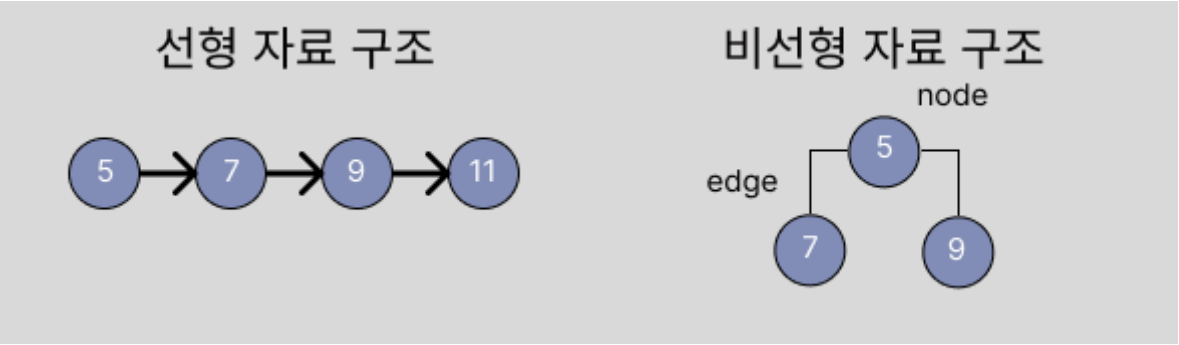
# 트리와 힙 - Concept

#트리와힙

## 트리란?

- 선형이 아닌 비선형으로 나타낸 자료구조 중 하나
- 선형 구조 vs 비선형 구조
  - 선형 구조는 내용을 나열한다. 그렇기에 내용과 내용은 서로 동등한 관계로 이루어져 있다.
  - 비선형 구조는 계층이 존재한다. 최상위 노드(node, 선형 구조에서 원소에 해당)는 하위 계층 노드와 간선(edge, 선형 구조에서 내용과 내용을 이어주는 선)으로 이루어져 있다. 계층이 있기에 상위 계층과 하위 계층은 동등하지 않다.




- 가령, 직무별로 나열하면 회사 아래에 각 직무에 따른 부서가 있으며 부서별 관계가 동등하지 않으므로 비선형 구조로 표현할 수 있다. 반대로 영화 제목을 가나다 순으로 나열하는 건 가나다 순으로 일차원적으로 차례차례 표시하면 그만이므로 선형 구조로 표현할 수 있다.
- 트리는 비선형 구조 중에서 순환을 돌지 않는 구조를 의미한다. 즉, 하위 노드끼리 연결된 간선이 있어, 하위 노드와 하위 노드를 거쳐 상위 노드로 갈 수 있는 경우, 트리가 아니다.
  - 예시 : 0 -> 1, 0 -> 2, 0 -> 3, 1 -> 2
  - 0번 노드에서 1, 2, 3을 전부 갈 수 있기에 0번이 상위 노드, 1, 2, 3 번이 하위 노드이다. 근데 하위 노드 1번에서 2번, 2번에서 0번으로 거쳐갈 수 있어 1 -> 2 -> 0 으로 순환이 가능하므로 트리라 볼 수 없다.

## 트리 관련 용어

- 노드 : 트리 자료 구조에 존재하는 원소
- 간선 : 트리 내부의 원소들 간의 관계를 나타내기 위해 잇는 선
- 리프 : 연결된 간선이 하나 밖에 없는 노드
- 깊이 : 어떤 노드에 도달하기 위해 지나쳐야 하는 간선의 수
- 높이 : 최상위 노드를 시작으로 가장 깊은 곳까지 들어갈 때 지나치는 간선의 수

## 간단한 트리 표현

```
struct treeNode {
        string label; // 저장할 자료
        TreeNode *parent // 부모 노드(상위 노드)를 가리키는 포인터
        vector<TreeNode*> children; // 자손 노드들(하위 노드들)을 가리키는 포인터 배열
}
```

- 위 트리는 현재 노드가 하위 노드와 연결된 간선이 n개. 즉, 직접적으로 연결된 하위 노드가 n개인 경우 하위 노느들로 이동하기 위해 하위 노드들의 주소를 별도로 저장하고 있다
- 추후, 나올 이진 검색 트리의 경우, 직접적으로 연결된 하위 노드가 두 개 밖에 없기에 보통 left, right로 구분하여 하위 노드의 주소를 기록한다.

## 트리의 순회

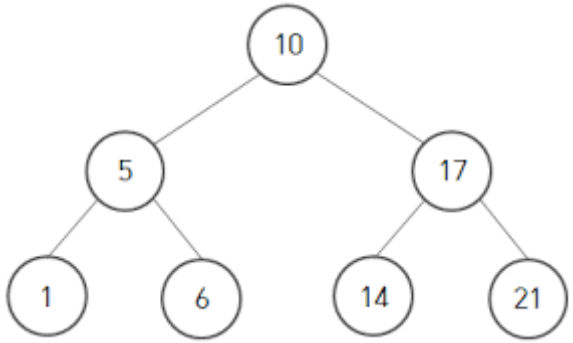
- 선형으로 구성된 배열과 달리 트리는 간선 및 노드의 개수에 따라 구조가 달라지므로 배열처럼 손쉽게 순회가 불가하다.

```
// 주어진 트리의 각 노드에 저장된 값을 모두 출력
void printLabels(TreeNode* root) {
        // 루트에 저장된 값을 출력한다
        std::cout << root->label << std::endl;
        // 각 자손들을 루트로 하는 서브트리에 포함된 값들을 재귀적으로 출력
        for (int i = 0; i < root->children.size(); ++i)
                printLabels(root->children[i]);
}
```

- 위와 같이 코드를 구성하면 root와 연결된 각각의 하위 노드를 순회하면서 값을 볼 수 있다

## 이진 검색 트리

- 이진 트리는 각 노드가 왼쪽과 오른쪽 최대 자식 노드를 두 개까지 가질 수 있는 트리를 의미한다.
- 이진 트리는 왼쪽 자식 노드를 본인보다 작은 값을 넣고 오른쪽 자식 노드를 본인보다 큰 값을 넣는다.




- 위 그림을 보자. 최상위 노드(root)와 비교해서 작은 것이 왼쪽 큰 것이 오른쪽으로 들어간 게 보이는 가? 즉, 새로운 원소가 들어올 때마다 root 부터 시작해서 비교를 하면서 왼쪽 또는 오른쪽으로 들어가는 것이다.
  - 예시] 그림에서 21을 빼고 지금 21을 넣는다 생각해 보자.

```
21 > 10 -> right
21 > 17 -> right
17의 right는 비어 있으므로 right 신규 node 생성 후에 21이 17의 right node를 차지
```

- 검색 또한 노드를 넣는 것처럼 대소 비교를 통해 left와 right 중 어느 쪽으로 갈 지 정하면서 값을 찾아나간다. 한 번 들어갈 때마다 비교해야 할 값이 절반 씩 줄어드는 셈이므로 lg n의 효율을 보인다.
- 순회
  - 전위 순회 : root(최상위 노드)를 먼저 방문. 그 뒤 왼쪽 자식 -> 오른쪽 자신 순서다.
    - 예시] 10 -> 5 -> 1 -> 6 -> 17 -> 14 -> 21
  - 중위 순회 : 왼쪽 자식 -> 최상위 노드 -> 오른쪽 자식 순서다. 각 자식들을 돌아볼 때 최하위 노드부터 돌아본다.
    - 예시] 1 -> 5 -> 6 -> 10 -> 14 -> 17 -> 21
  - 후위 순회 : 왼쪽 자식 -> 오른쪽 자식 -> 최상위 노드 순서다. 중위와 달리 각 자식의 노드를 돌아볼 때 동일 깊이를 우선한다.

        - 예시] 1 -> 6 -> 5 -> 14 -> 21 -> 17 -> 10
- 삭제
    - 노드 t를 지울 시, 노드 t의 자식 트리들을 합친 신규 트리를 만들고 그 트리가 노드 t의 자리를 대신한다
    - 예시] 위 그림에서 17을 삭제
        - 17의 하위 노드들인 14와 21이 붕 뜬다.
        - 14와 그 하위 노드들은 21보다 작을 수 밖에 없다. 그러므로 우측의 21을 기점으로 한 서브 트리를 14의 우측 하위 노드들과 재귀적으로 합치면 된다.
        - 결과적으로 21은 14의 우측 최상단 하위 노드가 되는 셈이다.

## 우선순위 큐와 힙

- 우선순위 큐는 원하는 자료값에 가까울 수록 상위 노드에 위치하는 큐.
- 힙은 이진 트리이며 우선 순위와 같은 특정 규칙을 만족하고자 만들어낸 자료구조다.
- 힙의 구성 규칙
    - 마지막 레벨(레벨 == 깊이)을 제외한 모든 레벨에 노드가 꽉 차 있을 것
    - 마지막 레벨에 노드가 있을 때는 항상 가장 왼쪽부터 순서대로 채워져 있을 것

## 힙의 특징

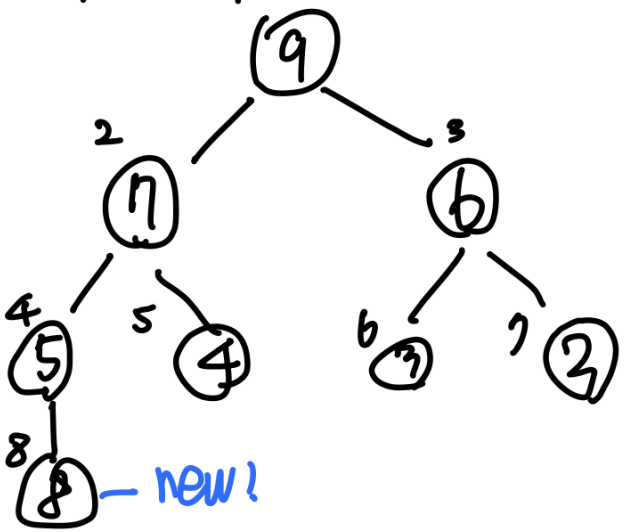
- 힙은 배열 원소와 대응이 된다.
    - array : A
    - root : A[0], 1st depth : left(A[1]) - right(A[2]), 2nd depth : left(A[3], A[4]) - right(A[5], A[6]), ...
    - 위 내용을 정리하면 아래와 같다
        - A[i]에 대응하는 노드의 왼쪽 자손은 A[2 * i + 1]
        - A[i]에 대응하는 노드의 오른쪽 자손은 A[2 * i + 2]
        - A[i]에 대응하는 노드이 부모는 A[floor((i - 1) / 2)]
    - 힙에 n 개의 노드가 있으면 배열 A의 원소는 1 ~ n - 1까지 순차적으로 넣을 수 있다. 이 때문에 힙의 구성 규칙이 엄격한 이유이다.
- 반대로 위 배열과 힙이 대응하는 부분을 노려서 힙을 배열로 구현하는 것도 가능하다.

## 최대 힙과 최소 힙

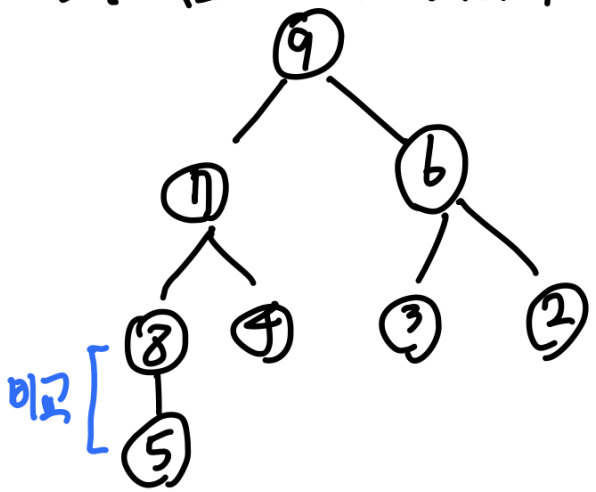
- 최대 힙은 큰 값일 수록 상위 노드에 위치하고 최소 힙은 작은 값일 수록 상위 노드에 위치한다.
- 최대 힙을 어떻게 만들 것인가?
  - 힙이 배열에 대응하는 점을 고려한다.
  - 최대 힙은 배열이 오름차순으로 정렬되었을 때가 기준이다. 그렇기에 힙의 깊이가 얕을 수록, 노드가 좌측에 위치할 수록 값이 크다

- 삽입 방법

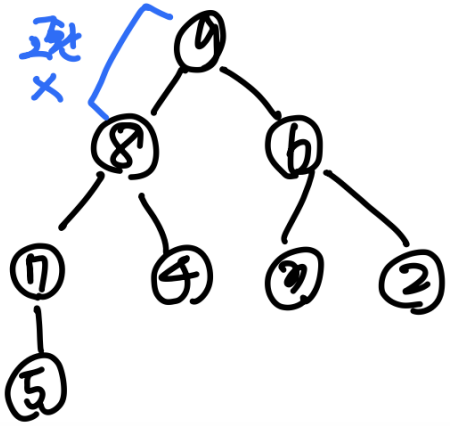
1. 가장 마지막 노드 위치 다음 위치에 신규 원소 추가



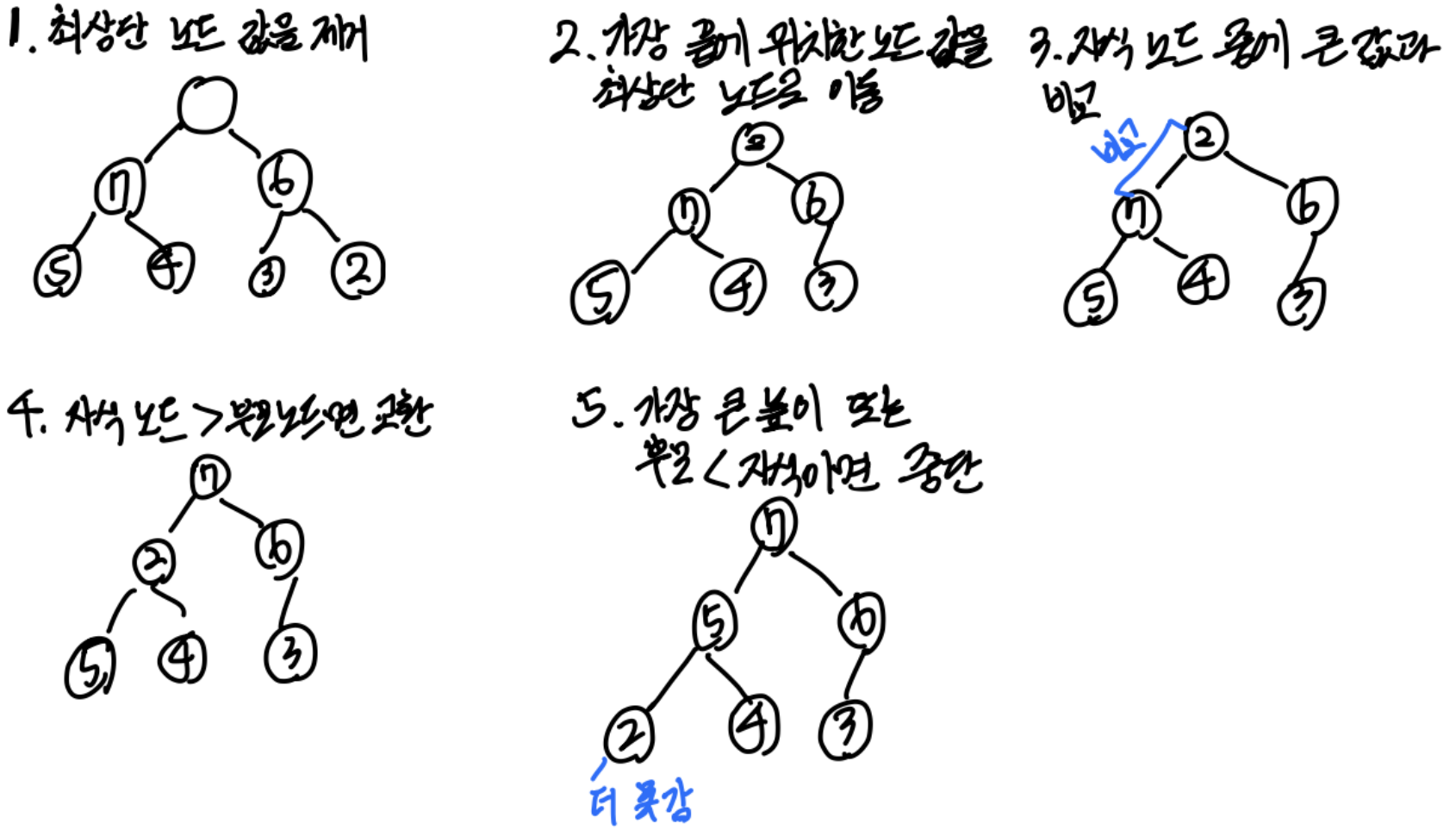

2. 부모 노드와 비교하여 자식 > 부모면 자리 교환




3. 자식 > 부모면 교환 계속



- 제거 방법




---

- 최소 힙의 삽입, 제거는 위 최대 힙의 삽입, 제거 방식에서 비교 연산자만 정반대면 된다.

## 그래서 트리와 힙은 언제 쓰면 됩니까?

1. 출발지와 목적지가 존재하거나 최단 경로를 구하거나 문자열 검색을 하는 등. 다종 다양한 곳에서 사용한다. 솔직히 명확한 예시와 종류를 조사하는 것보다 관련된 문제를 많이 풀어보면서 체화 시키는 게 빠르다.
2. "단시간", "경로" 같은 단어가 나오면 비선형 자료구조를 쓸 확률이 높다. 특히, 다음 부터 배우는 그래프를 보면 왜 그러는 지 알 수 있을 것이다.