201201871 서현택

**1.목표**

이진탐색트리 구현

**2.과제를 해결하는 방법**

1. 이진탐색트리에 대한 이해
2. insert, delete, search 인터페이스 구현

**3.과제를 해결한 방법**

1. search_by_tree

self._root에 key=key인 노드를 넣고, self._root가 없다면 False 반환하여 탐색 결과가 False 라는 것을 나타낸다. 찾으려는 key가 현재 노드의 key값과 일치하면 True를 반환하고, 적으면 현재 노드의 왼쪽 자식 노드를 search_by_key에 넣고 순환시켜 탐색한다. 찾으려는 key가 현재 노드의 key값보다 크다면 현재 노드의 오른쪽 자식 노드를 search_by_key에 넣고 순환시켜 탐색한다.

2. insert_node

현재 노드가 없다면, self._root에 Node를 입력하여 추가하고, size를 1 늘린다. 그 외의 경우엔 __insert_node를 통하여 삽입을 한다.

__insert_node에서는 삽입하려는 노드의 key값과 현재 노드의 key값을 비교하여 적거나 같으면 왼쪽 자식 노드가 있는지 판단한 후, 있다면 왼쪽 자식 노드에서 __insert_node를 넣어 재귀를 통해 삽입을 진행한다. 왼쪽 자식 노드가 없다면 왼쪽 노드에 추가하려는 노드를 삽입한다.

반대로 삽입하려는 노드의 key값이 현재 노드의 key값보다 크다면 오른쪽 자식 노드가 있는지 판단한 후, 있다면 오른쪽 자식 노드에서 __insert_node을 넣어 재귀를 통해 삽입을 진행한다. 없다면 오른쪽 노드에 추가하려는 노드를 삽입한다.

3. delete_node

self._root와 deleted 값에 _deleted_node의 값을 저장한 뒤 deleted 값을 반환하여 삭제 여부를 확인한다.

_deleted_node에서는 node가 없다면, node와 False를 반환하여 삭제가 되지 않았음을 나타낸다.

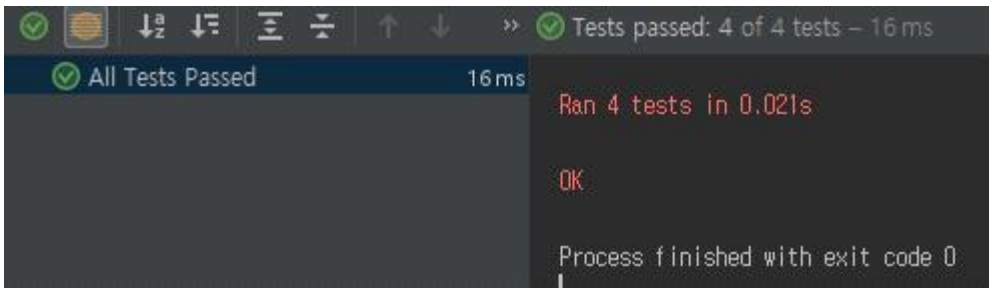
만약 삭제하려는 key값과 node의 key값이 같다면 deleted에 True를 저장한 뒤 node의 왼쪽 자식과 오른쪽 자식이 있는지를 판단하여 둘 다 있다면 parent의 현재 node 값, child에는 node의 오른쪽 노드 값을 저장하고 child의 왼쪽 자식 노드가 없어질 때까지 반복하여 가장 마지막 자식 값을 child에 저장하고, child의 left 값으로 node의 left값을 넣는다.

만약 parent 노드와 node가 같지 않다면 parent의 왼쪽 노드 값으로 child의 오른쪽 node 값을 저장하고, child의 오른쪽 노드 값으로 node의 오른쪽 노드 값을 저장한다. node에는 child의 값을 저장한다.

이 외에 node가 왼쪽 노드 또는 오른쪽 노드를 가진다면 node에 node의 왼쪽 노드 또는 오른쪽 노드를 저장한다.
만약 삭제하려는 key값이 node의 key값보다 적다면 node의 left와 deleted에 _deleted에 node의 왼쪽 노드를 넣어 순환시킨 값을 저장한다.
만약 삭제하려는 key값이 node의 key 값보다 크다면 node의 right와 deleted에 _deleted에 node의 오른쪽 노드를 넣어 순환시킨 값을 저장한다.
마지막으로 이런 과정을 통해 저장된 node와 deleted 값을 반환한다.

**4.결과화면**

(1) 테스트 코드




(2)BST_sample

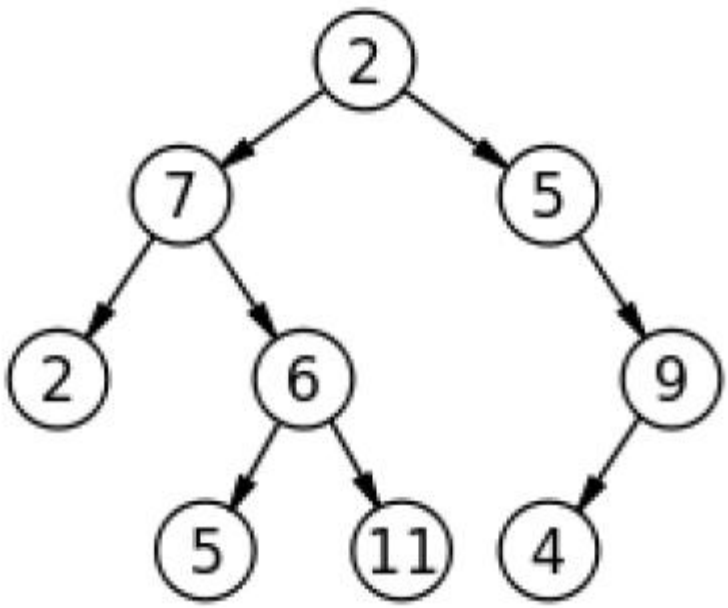
# 1 개요

트리(Tree)는 컴퓨터공학에서 부모-자식 관계로 '연결되어있는' 데이터를 표현할 때 가장 널리 사용되는 자료형이다. '노드(Node)'와 '에지(Edge)'로 구성되는 트리는 그래프(Graph)의 일종이라고 볼 수 있다. 그래프(Graph)에서 상위 노드(부모)로 가는 에지나 순환이 없으면 트리로 표현할 수 있다.

그렇다면 트리는 어디에 어떻게 활용해야할까? 사용할 장소나 상황을 고려하기 전에 활용하는 방법을 먼저 학습하자. 이번 실습과 과제는 트리를 활용하기 위해 '순회(Traversal)'를 배운다.

# 2 트리(Tree)

트리와 그래프에서 노드는 '키(Key)'값과 '밸류(Value)'값을 가진다. 또한 에지는 시작과 끝을 가지고 있다. 이것을 구현할 때에는 노드와 에지를 각각 클래스로 정의하여 객체를 생성하는 방법과 노드 클래스에 에지 정보를 포함하는 방법이 있다. 이번 실습과 과제에서는 구현의 편의성을 위하여 노드 클래스에 에지 정보를 포함하도록 한다.



# 3 트리 순회(Tree Traversal)

트리를 활용하기 위해서는 각 노드에 접근하는 방법을 알아야한다. 트리는 하나의 루트(Root)로 시작하여 여러개의 잎(Leaf) 노드로 구성되기 때문에 일정한 규칙으로 순회하기 위한 알고리즘이 존재한다. 대표적인 트리 순회 알고리즘은 **Preorder**, **Inorder**, **Postorder**, 그리고 **Levelorder**가 있다. 이중에서 가장 많이 사용되는 것은 Preorder, Inorder, Postorder다.

Preorder는 트리를 순회할 때 '부모를 먼저' 처리하는 방법이다. Inorder는 '자식중 절반'을 처리한 후 부모를 처리한다. Postorder는 '자식을 모두 처리'한 후 부모를 처리한다.

트리 순회 그리고 Levelorder

Listing 3: 재귀를 활용한 Preorder

```
1  def print_pre(tree):
2      print(tree.get_key(), tree.get_value())
3      if tree.get_left() is not None:
4          print_pre(tree.get_left())
5      if tree.get_right() is not None:
6          print_pre(tree.get_right())
7
8
9  def print_in(tree):
10     if tree.get_left() is not None:
11         print_in(tree.get_left())
12     print(tree.get_key(), tree.get_value())
13     if tree.get_right() is not None:
14         print_in(tree.get_right())
15
16
17 def print_post(tree):
18     if tree.get_left() is not None:
19         print_post(tree.get_left())
20     if tree.get_right() is not None:
21         print_post(tree.get_right())
22     print(tree.get_key(), tree.get_value())
23
24
25 def print_level(tree):
26     node_queue = [tree]
27     result = list()
28
29     while len(node_queue) > 0:
30         node = node_queue.pop(0)
31         print(node.get_key(), node.get_value())
32         result.append(node.get_key())
33         if node.get_left() is not None:
```

```
34                 node_queue.append(node.get_left())
35             if node.get_right() is not None:
36                 node_queue.append(node.get_right())
37
38     return result
```

Figure ??에 대하여 위의 순회를 수행해보자.