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.

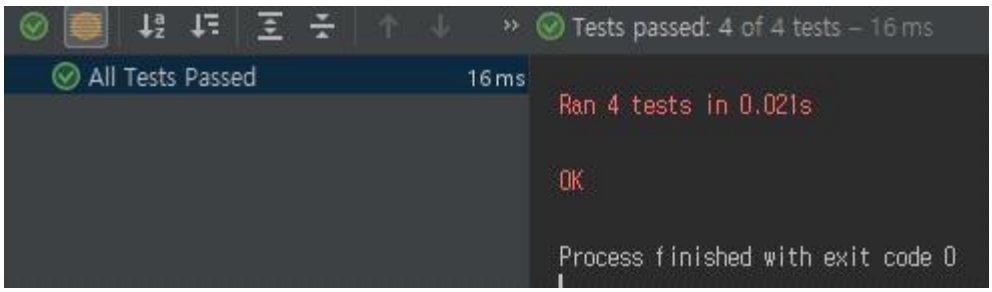
만약 삭제하려는 key값과 node의 key값이 같다면 deleted에 True를 저장한 뒤 node의 왼쪽 자식과 오른쪽 자식이 있는지를 판단하여 둘 다 있다면 parent의 현재 node 값, child에는 node의 오른쪽 노드 값을 저장하고 child의 왼쪽 자식 노드가 없어질 때까지 반복하여 가장 마지막 자식 값을 child에 저장하고, child의 left 값으로 node의 left값을 넣는다.

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

우리는 지난 실습시간에 컴퓨터공학에서 부모-자식 관계로 '연결되어있는' 데이터를 표현할 때 가장 널리 사용하는 트리(Tree)를 살펴보았다. 이러한 트리 중에서 정렬된 트리(Ordered Tree)라고도 불리는 이진 탐색 트리(Binary Search Tree)가 있다. BST(Binary Search Tree)는 노드(Node)의 키(Key)값을 기준으로 각 노드를 정렬하여 트리 형태로 저장한다.

BST는 대부분의 연산에 O(logn) 시간 복잡도를 가지기 때문에 성능을 필요로 하는 응용에서 자주 사용된다. Insert, Delete, Search에서 이미 정렬되어 있다는 특징으로 평균 O(logn) 최악의 경우 O(n)이 걸린다. 컴퓨터공학 다양한 분야에서 활용되는 BST지만 그대로 사용하기에는 결정적인 문제가 있다. 만약 Root의 키가 '최대값' 혹은 '최소값'으로 설정되면 어떻게 될까? 스스로 생각해보자.

2 이진 탐색 트리(Binary Search Tree)

BST에서 사용할 노드는 key, value와 함께 left, right, 그리고 parent도 멤버 변수로 가지고 있어야 한다. 이 parent는 Insert, Delete할 때 활용된다.

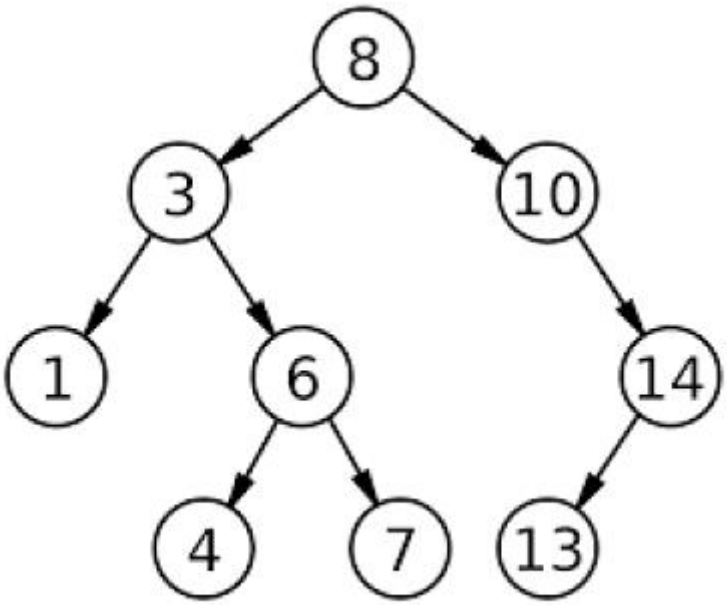


Figure 1: 이진 탐색 트리.png

BST는 멤버 변수(필드)로 root 노드와 크기(size)를 가진다. 이는 Insert, Delete가 일어날 때마다 변화되어야 한다.

is_bst

위의 함수는 BST를 검증하는 함수이다. 일정한 규칙이 있기 때문에 BST는 모든 노드를 순회하며 규칙에 따르고 있는지 확인하는 것으로 검증할 수 있다.

2.1 insert_node()

Listing 6: insert sudo code

```
1  void insert(Node*& root, int key, int value) {
2    if (!root)
3      root = new Node(key, value);
4    else if (key == root->key)
5      root->value = value;
6    else if (key < root->key)
7      insert(root->left, key, value);
8    else  // key > root->key
9      insert(root->right, key, value);
10 }
```

BST에 노드를 추가하는 것은 간단하다. 우리가 지난 시간에 트리를 순회했던 것을 이용하여 '조건에 맞는' 위치를 찾아가 연결해주면 된다. 위의 코드는 insert 멤버 함수를 재귀적으로 구현한 수도 코드다.

2.2 delete_node()

BST를 구현할 때 가장 어려운 것은 노드를 지우는 작업이다. 서로 '연결되어 있는' 형태이기 때문에 아무 고려없이 연결을 끊어버릴 수 없다. 노드를 지울 때에는 크게 4가지 상황이 있다.

- Left, Right Node가 모두 존재
- Left 노드가 존재
- Right 노드가 존재
- 자식 노드가 없음

지우려는 노드의 상황을 위의 조건에 맞게 따져본 후 삭제 작업을 해야한다.

Listing 7: insert sudo code

```
def binary_tree_delete(self, key):
    if key < self.key:
        self.left_child.binary_tree_delete(key)
        return
    if key > self.key:
        self.right_child.binary_tree_delete(key)
        return
    # delete the key here
    if self.left_child and self.right_child: # if both children are present
        successor = self.right_child.find_min()
        self.key = successor.key
        successor.binary_tree_delete(successor.key)
    elif self.left_child:   # if the node has only a *left* child
        self.replace_node_in_parent(self.left_child)
    elif self.right_child:  # if the node has only a *right* child
        self.replace_node_in_parent(self.right_child)
    else:
        self.replace_node_in_parent(None) # this node has no children
```

위는 delete_node()의 수도코드다. 하지만 그대로 구현하기 너무 어려울 경우 자신의 논리로 구현해도 무방하다. 자신만의 논리, 알고리즘을 세우기 위해선 아래 그림과 같이 직접 각 상황에 대하여 순서를 따져보아야 한다.