20강, 이진 탐색 트리(Binary Search Trees) (1)

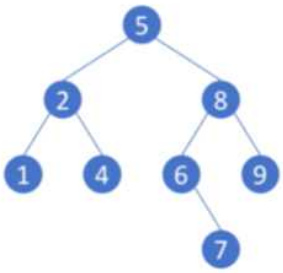
1) 이진 탐색 트리

모든 노드에 대해서

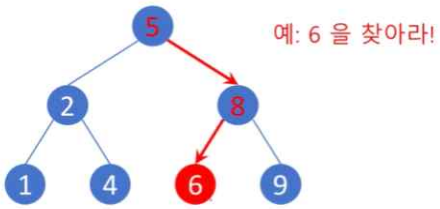
: 왼쪽 서브트리에 있는 데이터는 모두 현재 노드의 값보다 작고

: 오른쪽 서브트리에 있는 데이터는 모두 현재 노드의 값보다 큰

성질을 만족하는 이진 트리 (중복되는 데이터는 없다는 것을 가정)



2) 데이터를 찾기에 쉬워진다



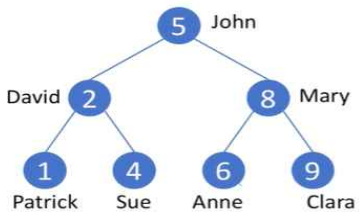
3) (정렬된) 배열을 이용한 이진 탐색과 비교-> 3강 참고

(장점) 데이터 원소의 추가, 삭제가 용이

(단점) 공간 소요가 큼

4) 이진 탐색 트리의 추상적 자료 구조

데이터 표현 : 각 노드는 (key, value)의 쌍으로



5) 연산의 정의

insert(key,data) : 트리에 주어진 데이터 원소를 추가
-입력 인자: 키, 데이터 원소
-리턴: 없음
remove(key) : 특정 원소를 트리로부터 삭제
lookup(key) : 특정 원소를 검색
-입력 인자: 찾으려는 대상 키
-리턴: 찾은 노드, 그것의 부모 노드 (각각, 없으면 None)
inorder() : 키의 순서대로 데이터 원소를 나열
min(), max() : 최소 키, 최대 키를 가지는 원소를 각각 탐색

6) 이진 탐색 트리에 원소 삽입 (연습 문제)

예) 3을 삽입하라!

7) 문제 (이진 탐색 트리의 원소 삽입 연산 구현)

초기 코드에 주어진 class Node 와 class BinSearchTree 를 기반으로, 이진 탐색 트리 (binary search tree) 에 새로운 원소를 삽입하는 insert(key, data) 연산의 구현을 완성하세요. class BinSearchTree 에는 이미 insert(key, data) 메서드가 구현되어 있습니다. 이것을 그대로 이용하고, class Node 의 insert(key, data) 메서드를 재귀적 방법으로 구현하세요. 강의에서 언급한 바와 같이, 이미 트리 안에 들어 있는 것과 같은 (중복된) 키를 이용하여 삽입을 시도하는 경우에는 KeyError 예외를 발생시켜야 합니다.
[참고 1] inorder() 메서드의 구현은 그대로 두세요. 테스트에 이용됩니다.
[참고 2] solution() 함수의 구현도 그대로 두세요. 이것을 없애면 테스트가 되지 않습니다.
[참고 3] 코드 실행 을 눌렀을 때 통과하는 것은 아무런 의미가 없습니다.

```
class Node:

    def __init__(self, key, data):
        self.key = key
        self.data = data
        self.left = None
        self.right = None


    def insert(self, key, data):
        if key<self.key:
            if self.left:
                self.left.insert(key,data)
            else:
                self.left=Node(key,data)
        elif key>self.key:
```

```
            if self.right:
                self.right.insert(key,data)
            else:
                self.right=Node(key,data)
        else:
            raise KeyError('Key %s already exists.' % key)

    def inorder(self):
        traversal = []
        if self.left:
            traversal += self.left.inorder()
        traversal.append(self)
        if self.right:
            traversal += self.right.inorder()
        return traversal


class BinSearchTree:

    def __init__(self):
        self.root = None


    def insert(self, key, data):
        if self.root:
            self.root.insert(key, data)
        else:
            self.root = Node(key, data)


    def inorder(self):
        if self.root:
            return self.root.inorder()
        else:
            return []


def solution(x):
    return 0
```