자료구조 9장 과제

Global Business & Technology
201904385 우인경

1. 이진탐색트리를 이용한 맵 구현 및 실행

- 코드화면

<이진탐색트리>

```
class BSTNode:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.left = None
        self.right = None

def search_bst(n,key):
    if n == None: return None
    elif key == n.key: return n
    elif key < n.key: return search_bst(n.left, key)
    else: return search_bst(n.right, key)

def search_bst_iter(n, key):
    while n != None:
        if key == n.key: return n
        elif key < n.key: n = n.left
        else: n = n.right
    return None

def search_max_bst(n):
    while n != None and n.right != None:
        n = n.right
    return n
def search_min_bst(n):
    while n != None and n.left != None:
        n = n.left
    return n

def count_node(n):
    if n is None:
        return 0
    else:
        return 1 + count_node(n.left) + count_node(n.right)

def inorder(n):
    if n is not None:
        inorder(n.left)
        print(n.key, end= ' ')
        inorder(n.right)

def levelorder(n):
    que = []
    que.append(n)
    while len(que) != 0:
        n = que.pop(0)
        if n is not None:
            print(n.key, end = ' ')
            que.append(n.left)
            que.append(n.right)
```

```python
def count_leaf(n):
    if n is None:
        return 0
    elif n.left is None and n.right is None:
        return 1
    else:
        return count_leaf(n.left) + count_leaf(n.right)

def calc_height(n):
    if n is None:
        return 0
    hLeft = calc_height(n.left)
    hRight = calc_height(n.right)
    if (hLeft > hRight):
        return hLeft + 1
    else:
        return hRight + 1

def insert_bst(r,n):
    if n.key < r.key:
        if r.left is None:
            r.left = n
            return True
        else:
            return insert_bst(r.left, n)
    elif n.key > r.key:
        if r.right is None:
            r.right = n
            return True
        else:
            return insert_bst(r.right, n)
    else:
        return False

def delete_bst_case1(parent, node, root):
    if parent is None:
        root = None
    else:
        if parent.left == node:
            parent.left = None
        else:
            parent.right = None
    return root

def delete_bst_case2(parent, node, root):
    if node.left is not None:
        child = node.left
    else:
        child = node.right

    if node == root: root = child
    else:
        if node is parent.left:
            parent.left = child
        else:
            parent.right = child
    return root
def delete_bst_case3(parent, node, root):
    succp = node
    succ = node.right
    while succ.left != None:
        succp = succ
        succ = succ.left

    if succp.left == succ :
        succp.left = succ.right
    else:
        succp.right = succ.right
    node.key = succ.key
    node.value = succ.value
    node = succ;

    return root

def delete_bst(root,key):
    if root == None: return None

    parent = None
    node = root
    while node != None and node.key != key:
        parent = node
        if key < node.key: node = node.left
        else: node = node.right;

    if node == None: return None
    if node.left == None and node.right == None:
        root = delete_bst_case1(parent, node, root)
    elif node.left == None or node.right == None:
        root = delete_bst_case2(parent, node, root)
    else:
        root = delete_bst_case3(parent, node, root)
    return root
```

<이진탐색트리를 이용한 맵>

```
from BSTNode import *

class BSTMap():
    def __init__(self):
        self.root = None

    def isEmpty(self):
        return self.root == None
    def clear(self):
        self.root = None
    def size(self):
        return count_node(self.root)

    def search(self, key): return search_bst(self.root, key)
    def searchValue(self, key): return search_value_bst(self.root, key)
    def findMax(self): return search_max_bst(self.root)
    def findMin(slef): return search_min_bst(self.root)

    def insert(self, key, value=None):
        n = BSTNode(key, value)
        if self.isEmpty():
            self.root = n
        else:
            insert_bst(self.root, n)
    def delete(self, key):
        self.root = delete_bst(self.root, key)
    def display(self, msg = 'BSTMap: '):
        print(msg, end='')
        inorder(self.root)
        print()


#test program

if __name__ == '__main__':
    map = BSTMap()
    data = [35, 18, 7, 26, 12, 3, 68, 22, 30, 99]

    print("[삽입 연산]: ",data)
    for key in data:
        map.insert(key)
    map.display("[중위 순회: ")

    if map.search(26) != None:
        print("[탐색 26 ] : 성공")
    else: print("[탐색 26 ] : 실패")
    if map.search(25) != None:
        print("[탐색 25 ] : 성공")
    else: print("[탐색 25 ] : 실패")

    map.delete(3); map.display("[  3 삭제] : ")
    map.delete(68); map.display("[ 68 삭제] : ")
    map.delete(18); map.display("[ 18 삭제] : ")
    map.delete(35); map.display("[ 35 삭제] : ")
```

- 실행화면

```
>>>
================== RESTART: /Users/inkyung/Documents/BSTMap.py ==========
[삽입 연산]:  [35, 18, 7, 26, 12, 3, 68, 22, 30, 99]
[중위 순회: 3 7 12 18 22 26 30 35 68 99
[탐색 26 ] : 성공
[탐색 25 ] : 실패
[  3 삭제] : 7 12 18 22 26 30 35 68 99
[ 68 삭제] : 7 12 18 22 26 30 35 99
[ 18 삭제] : 7 12 22 26 30 35 99
[ 35 삭제] : 7 12 22 26 30 99
>>>
```

2. AVL 트리를 이용한 맵 구현 및 실행

- 코드화면

```python
from BSTMap import *

class AVLMap(BSTMap):
    def __init__(self):
        super().__init__()

    def insert(self, key, value=None):
        n = BSTNode(key, value)
        if self.isEmpty():
            self.root = n
        else:
            self.root = insert_avl(self.root, n)

    def display(self, msg = 'AVLMap : '):
        print(msg, end=' ')
        levelorder(self.root)
        print()

def rotateLL(A):
    B = A.left
    A.left = B.right
    B.right = A
    return B
def rotateRR(A):
    B = A.right
    A.right = B.left
    B.left = A
    return B
def rotateRL(A):
    B = A.right
    A.right = rotateLL(B)
    return rotateRR(A)
def rotateLR(A):
    B = A.left
    A.left = rotateRR(B)
    return rotateLL(A)

def reBalance(parent):
    hDiff = calc_height_diff(parent)

    if hDiff > 1:
        if calc_height_diff(parent.left) > 0:
            parent = rotateLL(parent)
        else:
            parent = rotateLR(parent)
    elif hDiff < -1:
        if calc_height_diff(parent.right) < 0:
            parent = rotateRR(parent)
        else:
            parent = rotateRL(parent)
    return parent

```

```
def insert_avl(parent, node):
    if node.key < parent.key:
        if parent.left != None:
            parent.left = insert_avl(parent.left, node)
        else:
            parent.left = node
        return reBalance(parent)

    elif node.key > parent.key:
        if parent.right != None:
            parent.right = insert_avl(parent.right, node)
        else:
            parent.right = node
        return reBalance(parent)
    else:
        print("중복된 키 에러")

def calc_height_diff(n):
    if n == None: return 0
    return calc_height(n.left) - calc_height(n.right)
def calc_height(n):
    if n == None: return 0
    hleft = calc_height(n.left)
    hright = calc_height(n.right)
    return ( max(hleft, hright) + 1)



#test program
if __name__ == '__main__':

    # case 1
    node = [7,8,9,2,1,5,3,6,4]
    map = AVLMap()

    for i in node:
        map.insert(i)
        map.display("AVL(%d): "%i)

    print(" 노드의 개수 = %d" % count_node(map.root))
    print(" 단말의 개수 = %d" % count_leaf(map.root))
    print(" 트리의 높이 = %d" % calc_height(map.root))

    #case 2
    node = [0,1,2,3,4,5,6,7,8,9]
    map = AVLMap()

    for i in node:
        map.insert(i)
        map.display("AVL(%d): "%i)

    print(" 노드의 개수 = %d" % count_node(map.root))
    print(" 단말의 개수 = %d" % count_leaf(map.root))
    print(" 트리의 높이 = %d" % calc_height(map.root))
```

- 실행화면

```
========== RESTART: /Users/inkyung/Documents/AVLMa
AVL(7):  7
AVL(8):  7 8
AVL(9):  8 7 9
AVL(2):  8 7 9 2
AVL(1):  8 2 9 1 7
AVL(5):  7 2 8 1 5 9
AVL(3):  7 2 8 1 5 9 3
AVL(6):  7 2 8 1 5 9 3 6
AVL(4):  7 3 8 2 5 9 1 4 6
 노드의 개수 = 9
 단말의 개수 = 4
 트리의 높이 = 4
AVL(0):  0
AVL(1):  0 1
AVL(2):  1 0 2
AVL(3):  1 0 2 3
AVL(4):  1 0 3 2 4
AVL(5):  3 1 4 0 2 5
AVL(6):  3 1 5 0 2 4 6
AVL(7):  3 1 5 0 2 4 6 7
AVL(8):  3 1 5 0 2 4 7 6 8
AVL(9):  3 1 7 0 2 5 8 4 6 9
 노드의 개수 = 10
 단말의 개수 = 5
 트리의 높이 = 4
>>>
```