2021-2 자료구조및실습 실습과제 07

학번	2020136129	이름	최수연

1) 각 문제에 대한 분석과 및 해결 방법

1. 교재 316쪽의 실습문제 P8.1을 먼저 구현하고, P8.2~P8.6 중에서 두 개를 선택해 구현하시오.

[문제분석 및 해결 방법]

(1) P8.1

- 교재 8.3절의 함수들과 테스트 코드를 이용하여 P8.1의 두 개의 이진 트리의 각 부모노드를 root1, root2로 설정하여 생성하였다. 단말 노드의 경우 단말 노드의 자식노드는 모두 None으로 설정하고, 왼쪽 자식과 오른쪽 자식의 유무와 어떤 자식노드인지에 따라 TNode를 설정하였다. 8.3절의 함수들을 통해 총 4가지의 순회 방법으로 각 이진 트리를 출력하고, 각 이진 트리별 노드의 개수, 단말의 개수, 트리의 높이를 출력하도록 하였다.

(2) P8.4

- 균형 잡힌 트리는 해당 노드에서 좌우 높이 차이가 1 이하로 균형 잡혀 있어야 하므로 먼저 해당 노드의 좌우 높이를 구하고, 좌우 높이를 뺀 값이 각각 1보다 작거나 같고, -1보다 작거나 같아야 하며, 그 자식노드들도 모두 검사하기 위해 순환을 사용하도록 하였다. 그 외에는 모두 False를 반환하였으며, False 반환 시 함수를 순환하지 않고 바로 빠져나온다. 계속 반복하여 root가 None이 되면 True를 반환한다.

(3) P8.6

- 좌우 대칭 연산은 if 문을 사용하여 왼쪽 자식과 오른쪽 자식이 모두 None이 아닐 때, 왼쪽 오른쪽 자식을 서로 바꾼다. 또는 왼쪽 자식이 None이고, 오른쪽 자식이 None이 아닐 때 오른쪽 자식을 왼쪽 자식으로 복사하고, 오른쪽 자식을 None으로 바꾼다. 또는 오른쪽 자식이 None이고, 왼쪽 자식이 None이 아닐 때 왼쪽 자식을 오른쪽 자식으로 복사하고, 왼쪽 자식을 None으로 바꾼다. 왼쪽 자식과 오른쪽 자식이 모두 None일 때, 왼쪽 오른쪽 자식 모두 None 그대로 둔다. 그리고 해당 reverse 함수가 모든 노드에 거쳐 갈 수 있도록 순환을 이용하여 반환 값을 설정한다.

2) 자신이 구현한 주요 코드

```
# tree1 트리 생성
g = TNode('G', None, None)
h = TNode('H', None, None)
e = TNode('E', g, h)
f = TNode('F', None, None)
d = TNode('D', None, None)
c = TNode('C', e, f)
b = TNode('B', d, None)
root1 = TNode('A', b, c)

# tree2 트리 생성
```

```
a = TNode('A', None, None)
b = TNode('B', None, None)
c = TNode('C', None, None)
d = TNode('D', None, None)
e = TNode('E', None, None)
x = TNode('/', a, b)
y = TNode('*', x, c)
z = TNode('*', y, d)
root2 = TNode('+', z, e)

# tree3 트리 생성
c = TNode('C', None, None)
d = TNode('D', None, None)
f = TNode('F', None, None)
x = TNode('X', None, None)
b = TNode('B', c, d)
e = TNode('E', x, f)
root3 = TNode('A', b, e)
```

```
# P8.4 균형 잡힌 트리 검사
def is_balanced(root):
    if root is None:
        return True
    else:
        hLeft = calc_height(root.left)
        hRight = calc_height(root.right)
        hLeft += 1
        hRight += 1
        if (hLeft - hRight) <= 1 and (hLeft - hRight) >= -1 and is_balanced(root.left) and is_balanced(root.right):
            return True
        else:
            return False
```

```
# P8.6 좌우 대칭 연산
def reverse(root):
    if root is None:
        return 0
    else:
        if root.left is not None and root.right is not None:
            root.left, root.right = root.right, root.left
        elif root.left is None and root.right is not None:
            root.left = root.right
            root.right = None
        elif root.left is not None and root.right is None:
            root.right = root.left
            root.left = None
```

```
        elif root.left is None and root.right is None:
            root.left = None
            root.right = None
        return reverse(root.left) or reverse(root.right)
```

3) 다양한 입력에 대한 테스트 결과

```
   In-Order : D B A G E H C F
  Pre-Order : A B D C E G H F
 Post-Order : D B G H E F C A
Level-Order : A B C D E F G H
tree1의 노드의 개수 = 8개
tree1의 단말의 개수 = 4개
tree1의 트리의 높이 = 4
tree1은 균형 잡힌 트리입니다.
reverse: A C B F E D H G

   In-Order : A / B * C * D + E
  Pre-Order : + * * / A B C D E
 Post-Order : A B / C * D * E +
Level-Order : + * E * D / C A B
tree2의 노드의 개수 = 9개
tree2의 단말의 개수 = 5개
tree2의 트리의 높이 = 5
tree2은 균형 잡힌 트리가 아닙니다.
reverse: + E * D * C / B A

   In-Order : C B D A E F
  Pre-Order : A B C D E F
 Post-Order : C D B F E A
Level-Order : A B E C D F
tree3의 노드의 개수 = 6개
tree3의 단말의 개수 = 3개
tree3의 트리의 높이 = 3
tree3은 균형 잡힌 트리입니다.
reverse: A E B F D C
```

4) 코드에 대한 설명 및 해당 문제에 대한 고찰

- 없음. (코드 설명은 **1) 각 문제에 대한 분석 및 해결 방법** 참고)

5) 이번 과제에 대한 느낀점

이번 과제는 2주가 주어져서 그런지 이전 과제보다는 어려웠던 것 같다. 그래도 과제 덕분에 이진 트리를 다시 한번 복습하면서 공부해보는 시간을 가질 수 있어서 좋았다.

6) 궁금한 점이나 건의사항

딱히 없습니다.

7) 자신이 구현한 전체 코드

 # queueClass 파일의 CircularQueue 클래스

```
MAX_QSIZE = 10

class CircularQueue:
    def __init__( self ):
        self.front = 0
        self.rear = 0
        self.items = [None] * MAX_QSIZE

    def isEmpty( self ): return self.front == self.rear
    def isFull( self ): return self.front == (self.rear+1)%MAX_QSIZE
    def clear( self ): self.front = self.rear

    def enqueue( self, item ):
        if not self.isFull():
            self.rear = (self.rear+1)%MAX_QSIZE
            self.items[self.rear] = item

    def dequeue( self ):
        if not self.isEmpty():
            self.front = (self.front+1)%MAX_QSIZE
            return self.items[self.front]

    def peek( self ):
        if not self.isEmpty():
            return self.items[(self.front+1)%MAX_QSIZE]

    def size( self ):
        return(self.rear - self.front + MAX_QSIZE)%MAX_QSIZE

    def display( self ):
```

```
        out = []
        if self.front < self.rear:
            out = self.items[self.front+1:self.rear+1]
        else:
            out = self.items[self.front+1:MAX_QSIZE]₩
                + self.items[0:self.rear+1]
        print("[f=%s,r=%d] ==> "%(self.front, self.rear), out)
```

 # 이진 트리

```
from queueClass import CircularQueue

class TNode:
    def __init__ (self, data, left, right):
        self.data = data
        self.left = left
        self.right = right

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

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

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

def levelorder(root) :
    queue = CircularQueue()
    queue.enqueue(root)
    while not queue.isEmpty() :
        n = queue.dequeue()
        if n is not None :
            print(n.data, end=' ')
            queue.enqueue(n.left)
            queue.enqueue(n.right)
```

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

    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 is_balanced(root):
        if root is None:
            return True
        else:
            hLeft = calc_height(root.left)
            hRight = calc_height(root.right)
            hLeft += 1
            hRight += 1
            if (hLeft - hRight) <= 1 and (hLeft - hRight) >= -1 and is_balanced(root.left) and
is_balanced(root.right):
                return True
            else:
                return False

    def reverse(root):
        if root is None:
            return 0
        else:
            if root.left is not None and root.right is not None:
                root.left, root.right = root.right, root.left
            elif root.left is None and root.right is not None:
```

```
            root.left = root.right
            root.right = None
        elif root.left is not None and root.right is None:
            root.right = root.left
            root.left = None
        elif root.left is None and root.right is None:
            root.left = None
            root.right = None
        return reverse(root.left) or reverse(root.right)

############################################################

g = TNode('G', None, None)
h = TNode('H', None, None)
e = TNode('E', g, h)
f = TNode('F', None, None)
d = TNode('D', None, None)
c = TNode('C', e, f)
b = TNode('B', d, None)
root1 = TNode('A', b, c)

print('₩n    In-Order : ', end='')
inorder(root1)
print('₩n   Pre-Order : ', end='')
preorder(root1)
print('₩n Post-Order : ', end='')
postorder(root1)
print('₩nLevel-Order : ', end='')
levelorder(root1)
print()

print("tree1의 노드의 개수 = %d개" % count_node(root1))
print("tree1의 단말의 개수 = %d개" % count_leaf(root1))
print("tree1의 트리의 높이 = %d" % calc_height(root1))

if(is_balanced(root1) == True):
    print("tree1은 균형 잡힌 트리입니다.")
else:
    print("tree1은 균형 잡힌 트리가 아닙니다.")

print("reverse: ", end ='')
reverse(root1)
levelorder(root1)
print()
```

```
#############################################################

a = TNode('A', None, None)
b = TNode('B', None, None)
c = TNode('C', None, None)
d = TNode('D', None, None)
e = TNode('E', None, None)
x = TNode('/', a, b)
y = TNode('*', x, c)
z = TNode('*', y, d)
root2 = TNode('+', z, e)

print('₩n    In-Order : ', end='')
inorder(root2)
print('₩n   Pre-Order : ', end='')
preorder(root2)
print('₩n Post-Order : ', end='')
postorder(root2)
print('₩nLevel-Order : ', end='')
levelorder(root2)
print()

print("tree2의 노드의 개수 = %d개" % count_node(root2))
print("tree2의 단말의 개수 = %d개" % count_leaf(root2))
print("tree2의 트리의 높이 = %d" % calc_height(root2))

if(is_balanced(root2) == True):
    print("tree2은 균형 잡힌 트리입니다.")
else:
    print("tree2은 균형 잡힌 트리가 아닙니다.")

print("reverse: ", end ='')
reverse(root2)
levelorder(root2)
print()

#############################################################

c = TNode('C', None, None)
d = TNode('D', None, None)
f = TNode('F', None, None)
b = TNode('B', c, d)
e = TNode('E', None, f)
root3 = TNode('A', b, e)
```

```
print('₩n     In-Order : ', end='')
inorder(root3)
print('₩n    Pre-Order : ', end='')
preorder(root3)
print('₩n  Post-Order : ', end='')
postorder(root3)
print('₩nLevel-Order : ', end='')
levelorder(root3)
print()

print("tree3의 노드의 개수 = %d개" % count_node(root3))
print("tree3의 단말의 개수 = %d개" % count_leaf(root3))
print("tree3의 트리의 높이 = %d" % calc_height(root3))

if(is_balanced(root3) == True):
    print("tree3은 균형 잡힌 트리입니다.")
else:
    print("tree3은 균형 잡힌 트리가 아닙니다.")

print("reverse: ", end ='')
reverse(root3)
levelorder(root3)
print()
print()
```