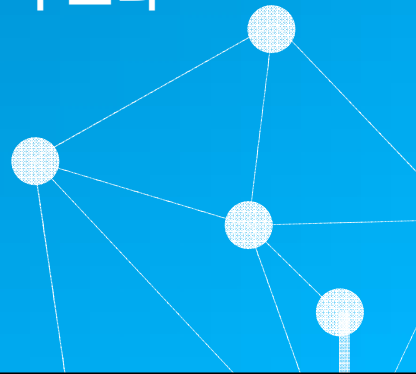


09 CHAPTER

탐색 트리



9장. 학습 목표



- 이진탐색트리의 개념과 연산들을 이해한다.
- 이진탐색트리의 효율성을 이해한다.
- 이진탐색트리 균형화의 의미를 이해한다.
- AVL 트리의 원리를 이해한다.
- 탐색트리를 이용한 맵의 구현을 이해한다.

9.1 탐색트리란?

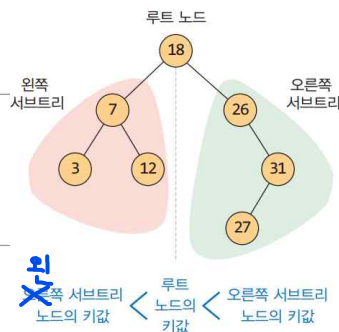
- 탐색트리는 탐색을 위한 트리 기반의 자료구조이다.

탐색트리란?

- 탐색을 위한 트리 기반의 자료구조이다.
- 이진탐색트리
 - 효율적인 탐색을 위한 이진트리 기반의 자료구조
 - 삽입, 삭제, 탐색: $O(\log n)$

정의 9.1 이진탐색트리

- 모든 노드는 유일한 키를 갖는다.
- 왼쪽 서브트리의 키들은 루트의 키보다 작다.
- 오른쪽 서브트리의 키들은 루트의 키보다 크다.
- 왼쪽과 오른쪽 서브트리도 이진탐색트리이다.



9.2 이진탐색트리의 연산



- 이진탐색트리: 노드 구조
- 탐색연산
- 삽입연산
- 삭제연산
- 이진 탐색 트리의 성능 분석

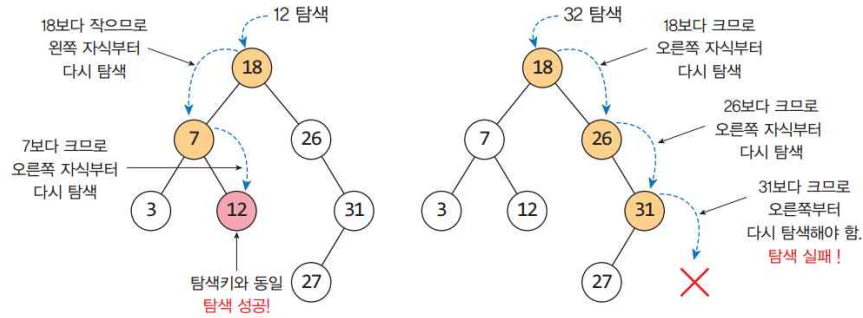
이진탐색트리: 노드 구조



- 노드의 구조
 - (탐색키, 키에 대한 값)의 형태

```
class BSTNode:                                # 이진탐색트리를 위한 노드 클래스
    def __init__(self, key, value):            # 생성자: 키와 값을 받음
        self.key = key                        # 키 (key)
        self.value = value                    # 값 (value)
        self.left = None                      # 왼쪽 자식에 대한 링크
        self.right = None                     # 오른쪽 자식에 대한 링크
```

탐색 연산: 키를 이용한 탐색



파이썬으로 쉽게 배우는
자료 구조

7

탐색 연산: 순환과 반복

- 순환 구조와 반복 구조로 구현할 수 있음

```
# 이진탐색트리 탐색연산(순환 함수)
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)
```

```
# 이진탐색트리 탐색연산(반복 함수)
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
```

- 값을 이용한 탐색은?
 - 모든 노드를 검사해야 함

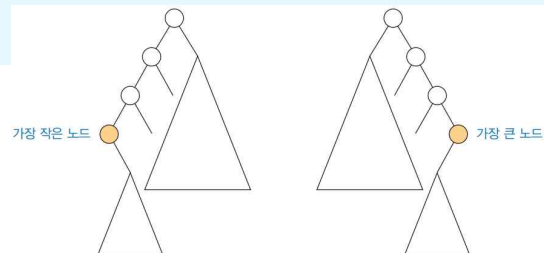
파이썬으로 쉽게 배우는
자료 구조

8

탐색 연산: 최대와 최소 노드

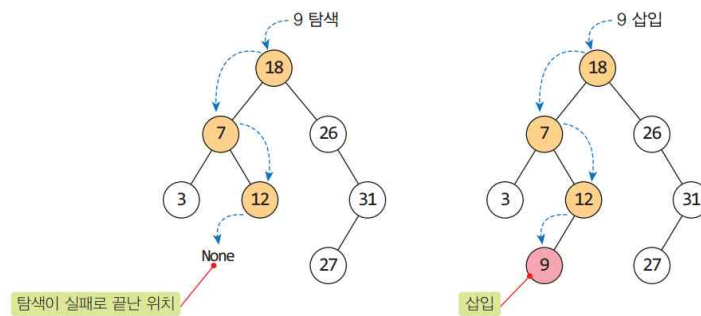
```
def search_max_bst(n) :                # 최대 값의 노드 탐색
    while n != None and n.right != None:
        n = n.right
    return n
```

```
def search_min_bst(n) :                # 최소 값의 노드 탐색
    while n != None and n.left != None:
        n = n.left
    return n
```



삽입 연산

- 탐색에 실패한 위치 → 노드를 삽입해야 하는 위치



삽입 연산 알고리즘

이진탐색트리 삽입연산 (노드를 삽입함): 순환구조 이용

```
def insert_bst(r, n):  
    if n.key < r.key:  
        # 삽입할 노드의 키가 루트보다 작으면  
        if r.left is None:  
            # 루트의 왼쪽 자식이 없으면  
            r.left = n  
            # n은 루트의 왼쪽 자식이 됨.  
            return True  
        else:  
            # 루트의 왼쪽 자식이 있으면  
            return insert_bst(r.left, n) # 왼쪽 자식에게 삽입하도록 함  
    elif n.key > r.key:  
        # 삽입할 노드의 키가 루트보다 크면  
        if r.right is None:  
            # 루트의 오른쪽 자식이 없으면  
            r.right = n  
            # n은 루트의 오른쪽 자식이 됨.  
            return True  
        else:  
            # 루트의 오른쪽 자식이 있으면  
            return insert_bst(r.right, n) # 오른쪽 자식에게 삽입하도록 함  
    else:  
        # 키가 중복되면  
        return False  
        # 삽입하지 않음
```

파이썬으로 풀이
자료구조

11

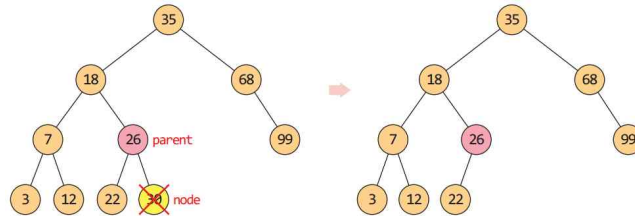
삭제 연산

- 노드 삭제의 3가지 경우
 1. 삭제하려는 노드가 단말 노드일 경우
 2. 삭제하려는 노드가 하나의 왼쪽이나 오른쪽 서브 트리 중 하나만 가지고 있는 경우
 3. 삭제하려는 노드가 두개의 서브 트리 모두 가지고 있는 경우

파이썬으로 풀이
자료구조

12

Case 1: 단말 노드 삭제



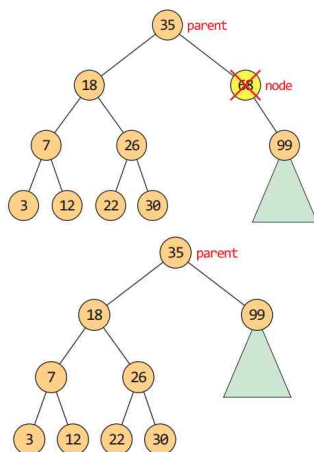
```
def delete_bst_case1 (parent, node, root) :
    if parent is None:
        root = None
    else :
        if parent.left == node :
            parent.left = None
        else :
            parent.right = None
    return root
```

삭제할 단말 노드가 루트이면
공백 트리가 됨
삭제할 노드가 부모의 왼쪽 자식이면
부모의 왼쪽 링크를 None
오른쪽 자식이면
부모의 오른쪽 링크를 None
root가 변경될 수도 있으므로 반환

파이썬으로 쉽게 배우는
자료구조

13

Case2: 자식이 하나인 노드의 삭제



```
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
```

삭제할 노드가
child는 왼쪽
삭제할 노드가
child는 오른쪽
없애려는 노드가
이제 child가
삭제할 노드가
부모의 왼쪽
부모의 오른쪽
root가 변경될

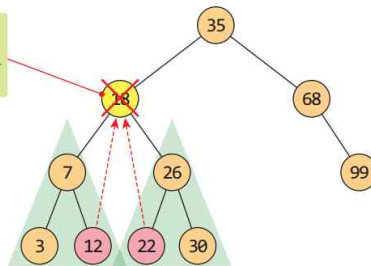
파이썬으로 쉽게 배우는
자료구조

14

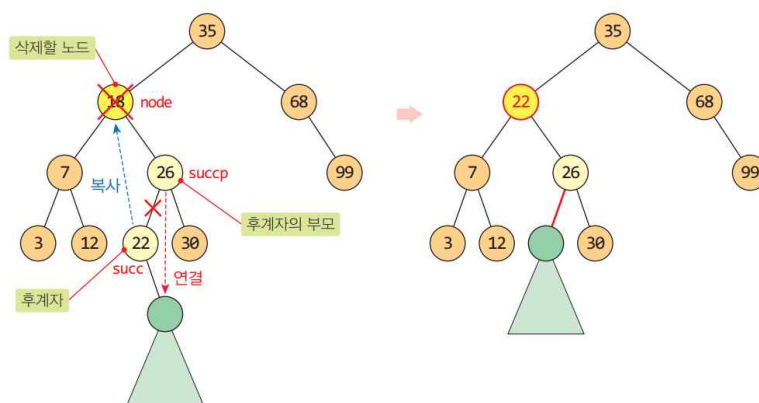
Case 3: 두 개의 자식을 가진 노드 삭제

- 가장 비슷한 값을 가진 노드를 삭제 위치로 가져옴
- 후계 노드의 선택

삭제할 위치에 왼쪽 서브트리의 가장 큰 노드나 오른쪽 서브트리의 가장 작은 노드가 들어가면 이진탐색트리의 조건을 계속 만족한다.



예) 노드 18 삭제



Case 3: 삭제 알고리즘

```
def delete_bst_case3 (parent, node, root) :
    succp = node                # 후계자의 부모 노드
    succ = node.right           # 후계자 노드
    while (succ.left != None) : # 후계자와 부모노드 탐색
        succp = succ
        succ = succ.left

    if (succp.left == succ) :   # 후계자가 왼쪽 자식이면
        succp.left = succ.right # 후계자의 오른쪽 자식 연결
    else :                      # 후계자가 오른쪽 자식이면
        succp.right = succ.right # 후계자의 왼쪽 자식 연결
        node.key = succ.key      # 후계자의 키와 값을
        node.value = succ.value  # 삭제할 노드에 복사
        node = succ;            # 실제로 삭제하는 것은 후계자 노드

    return root                # 일관성을 위해 root반환
```

파이썬으로 쉽게 배우는
자료구조

17

삭제 연산: 전체 코드

```
# 이진탐색트리 삭제연산 (노드를 삭제함)
def delete_bst (root, key) :
    if root == None : return None                # 공백 트리

    parent = None                                # 삭제할 노드의 부모 탐색
    node = root                                  # 삭제할 노드 탐색
    while node != None and node.key != key :     # parent 탐색
        parent = node
        if key < node.key : node = node.left
        else : node = node.right;

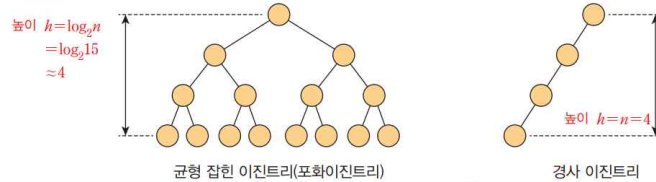
    if node == None : return None                # 삭제할 노드가 없음
    if node.left == None and node.right == None: # case 1: 단말 노드
        root = delete_bst_case1 (parent, node, root)
    elif node.left==None or node.right==None :   # case 2: 유일한 자식
        root = delete_bst_case2 (parent, node, root)
    else :                                        # case 3: 두 개의 자식
        root = delete_bst_case3 (parent, node, root)
    return root                                  # 변경된 루트 노드를 반환
```

파이썬으로 쉽게 배우는
자료구조

18

이진탐색트리의 성능

- 탐색, 삽입, 삭제 연산의 시간 트리의 높이에 비례함



연산	함수	최선의 경우 (균형트리)	최악의 경우 (경사트리)
키를 이용한 탐색	search_bst() search_bst_iter()	$O(\log_2 n)$	$O(n)$
값을 이용한 탐색	search_value_bst()	$O(n)$	$O(n)$
최대/최소 노드 탐색	search_max_bst() search_min_bst()	$O(\log_2 n)$	$O(n)$
삽입	insert_bst()	$O(\log_2 n)$	$O(n)$
삭제	delete_bst()	$O(\log_2 n)$	$O(n)$

파이썬으로 쉽게 배우는
자료구조

19

9.3 이진탐색트리를 이용한 맵

- 이진탐색트리를 이용한 맵 클래스
- 테스트 프로그램

파이썬으로 쉽게 배우는
자료구조

20

이진탐색트리를 이용한 맵 클래스

```
class BSTMap():
    # 이진탐색트리를 이용한 맵
    def __init__(self):
        # 생성자
        self.root = None
        # 트리의 루트 노드

    def isEmpty(self): return self.root == None
    # 맵 공백검사

    def clear(self): self.root = None
    # 맵 초기화

    def size(self): return count_node(self.root)
    # 레코드(노드) 수 계산

    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(self): return search_min_bst(self.root)

    def insert(self, key, value=None):
        # 삽입 연산
        n = BSTNode(key, value)
        # 키와 값으로 새로운 노드 생성
        if self.isEmpty():
            # 공백이면
            self.root = n
            # 루트노드로 삽입
        else:
            # 공백이 아니면
            insert_bst(self.root, n)
            # insert_bst() 호출

    def delete(self, key):
        # 삭제 연산
        self.root = delete_bst(self.root, key)
        # delete_bst() 호출
```

파이썬으로 쉽게 배우는
자료구조

21

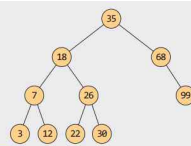
테스트 프로그램

```
map = BSTMap()
data = [35, 18, 7, 26, 12, 3, 68, 22, 30, 99]

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

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

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



```
C:\Python\
[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
```

파이썬으로 쉽게 배우는
자료구조

22

9.4 심화 학습: 균형이진탐색트리

- AVL 트리란?
- AVL 트리의 삽입 연산
- AVL 트리를 이용한 맵

AVL 트리란?

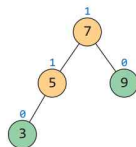
정의 9.2 AVL 트리

AVL 트리는 모든 노드에서 왼쪽 서브트리와 오른쪽 서브트리의 높이 차이가 1을 넘지 않는 이진탐색 트리이다. 즉 모든 노드의 균형 인수는 0 이나 ± 1 이 되어야 한다.

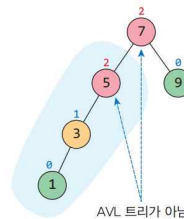
- Adelson-Velskii와 Landis에 의해 1962년에 제안된 트리
- 평균, 최선, 최악 시간복잡도: $O(\log n)$ 보장

- **균형 인수** : 왼쪽서브트리 높이 - 오른쪽서브트리 높이

● 균형 인수 +2, -2
● 균형 인수 +1, -1
● 균형 인수 0



AVL 트리



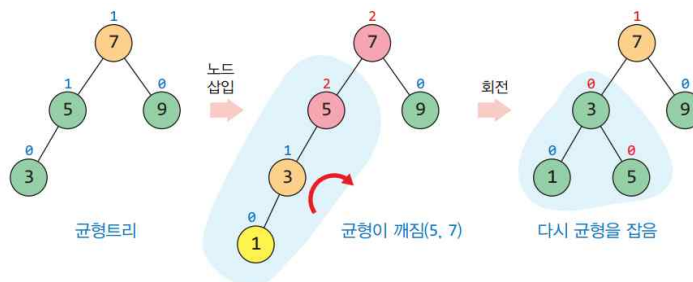
AVL 트리가 아님

AVL 트리의 연산

- 탐색연산: 이진탐색트리와 동일
- 삽입과 삭제 시 균형 상태가 깨질 수 있음
- 삽입 연산
 - 삽입 위치에서 루트까지의 경로에 있는 조상 노드들의 균형 인수에 영향을 미침
 - 삽입 후에 불균형 상태로 변한 가장 가까운 조상 노드(균형 인수가 ± 2 가 된 가장 가까운 조상 노드)의 서브 트리들에 대하여 다시 재균형
 - 삽입 노드부터 균형 인수가 ± 2 가 된 가장 가까운 조상 노드까지 회전

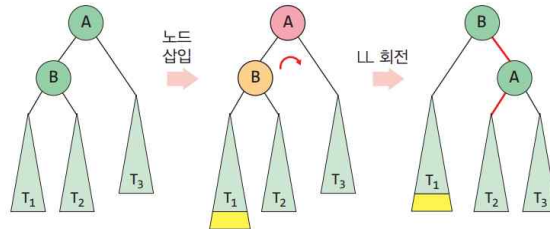
AVL 트리의 삽입연산

- 노드 1을 트리에 추가 → 균형이 깨짐



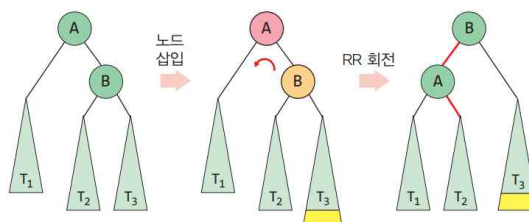
- 균형이 깨지는 4가지 경우
 - LL, LR, RL, RR 타입

LL 회전 방법



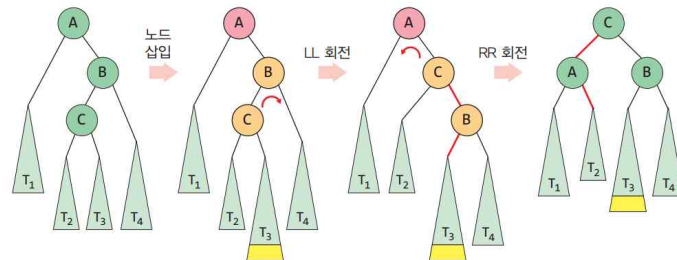
```
def rotateLL(A) :
    B = A.left          # 시계방향 회전
    A.left = B.right
    B.right = A
    return B           # 새로운 루트 B를 반환
```

RR 회전 방법



```
def rotateRR(A) :
    B = A.right          # 반 시계방향 회전
    A.right = B.left
    B.left = A
    return B             # 새로운 루트 B를 반환
```

RL 회전 방법

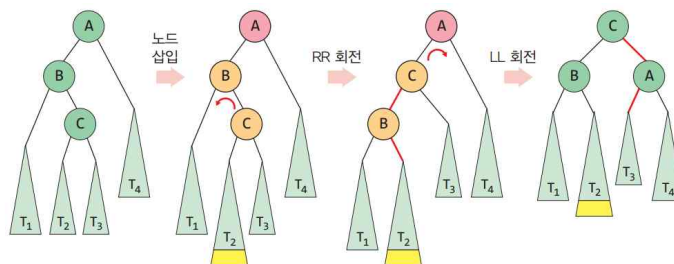


```
def rotateRL(A) :
    B = A.right
    A.right = rotateLL(B)    # LL회전
    return rotateRR(A)      # RR회전
```

파이썬으로 쉽게 배우는
자료구조

29

LR 회전 방법



```
def rotateLR(A) :
    B = A.left
    A.left = rotateRR(B)    # RR회전
    return rotateLL(A)      # LL회전
```

파이썬으로 쉽게 배우는
자료구조

30

재균형 함수

```
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
```

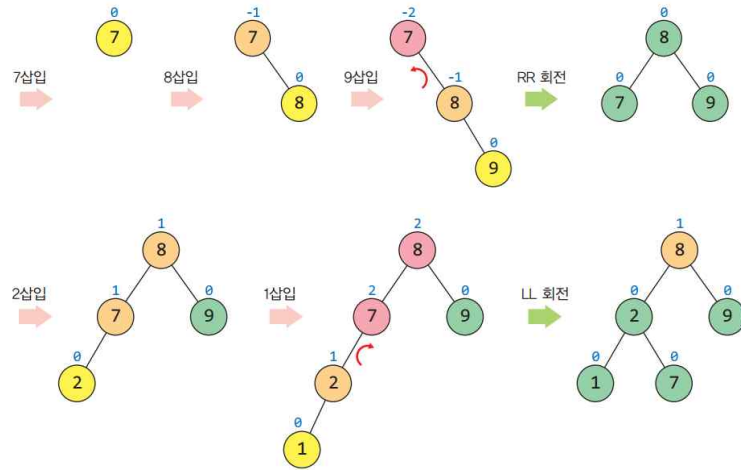
AVL 트리의 삽입함수

```
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")
```


AVL트리 구축의 예

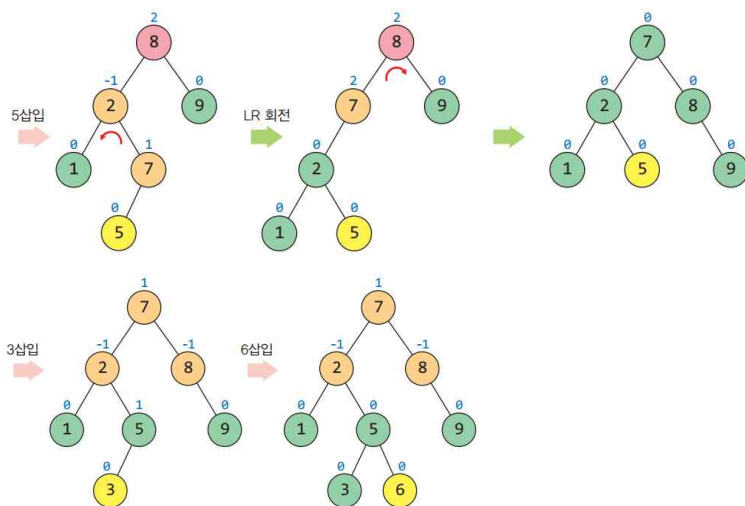
[7, 8, 9, 2, 1, 5, 3, 6, 4]



파이썬으로 쉽게 배우는
자료구조

33

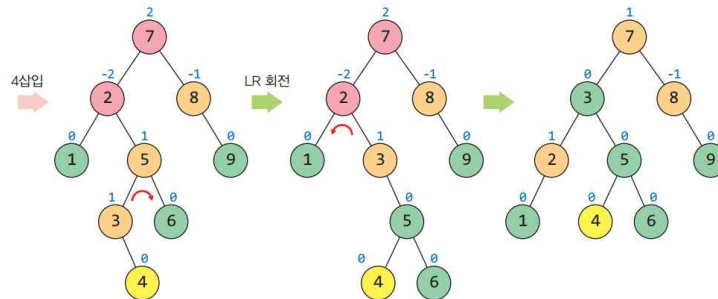
AVL트리 구축의 예(계속)



파이썬으로 쉽게 배우는
자료구조

34

AVL트리 구축의 예(계속)



파이썬으로 쉽게 배우는
자료구조

35

AVL 트리를 이용한 맵

```
node = [7,8,9,2,1,5,3,6,4] # node = [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 ))
```

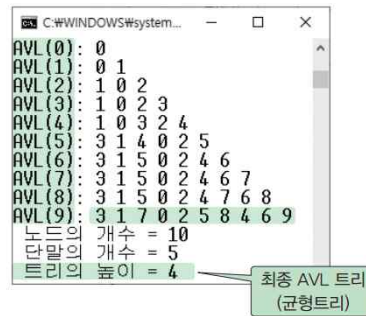
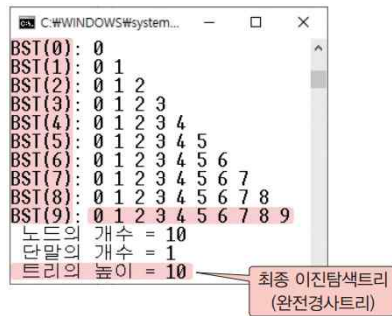
```
C:\WINDOWS\system32\cmd.exe
AVL(7): 7
AVL(8): 7 8
AVL(9): 8 7 9
AVL(2): 8 7 9 2
AVL(1): 8 2 9 1 7
AVL(5): 7 2 8 1 5 9
AVL(3): 7 2 8 1 5 9 3
AVL(6): 7 2 8 1 5 9 3 6
AVL(4): 7 3 8 2 5 9 1 4 6
노드의 개수 = 9
단말의 개수 = 4
트리의 높이 = 4
```

파이썬으로 쉽게 배우는
자료구조

36

이진탐색트리와 AVL트리 비교

- [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] 입력



9장 연습문제, 실습문제

