CHAPTER 08

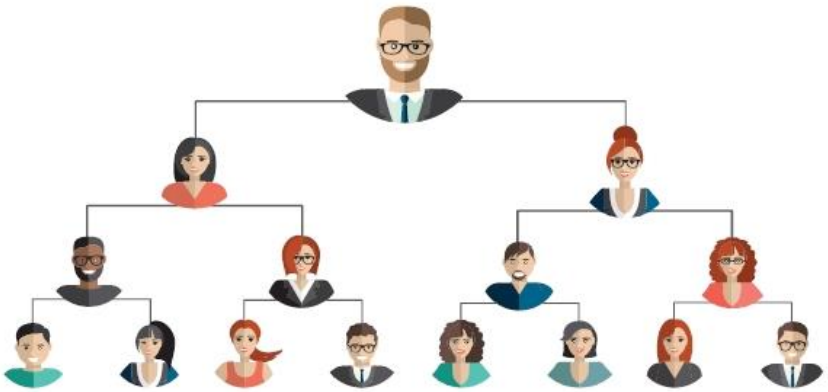
이진 트리

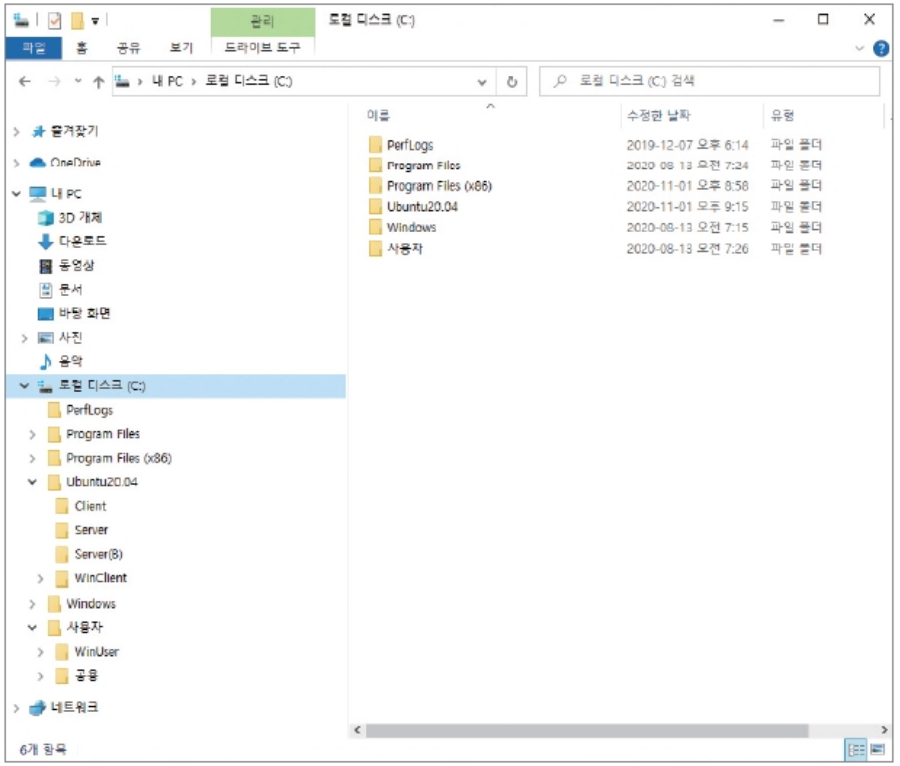
학습목표

- 이진 트리 개념을 파악한다.
- 파이썬으로 이진 트리를 구현하는 코드를 작성한다.
- 이진 트리 중 활용도가 높은 이진 탐색 트리를 이해하고 활용한다.
- 이진 탐색 트리를 활용하여 다양한 응용 프로그램을 작성한다.

Section 00 생활 속 자료구조와 알고리즘

- **트리 구조란?**
  - 회사 사장을 필두로 그 아래 직책들이 구성되어 있는 조직표 또는 컴퓨터의 상위 폴더 안에 하위 폴더들이 계속 이어져 있는 구조와 같은 구성

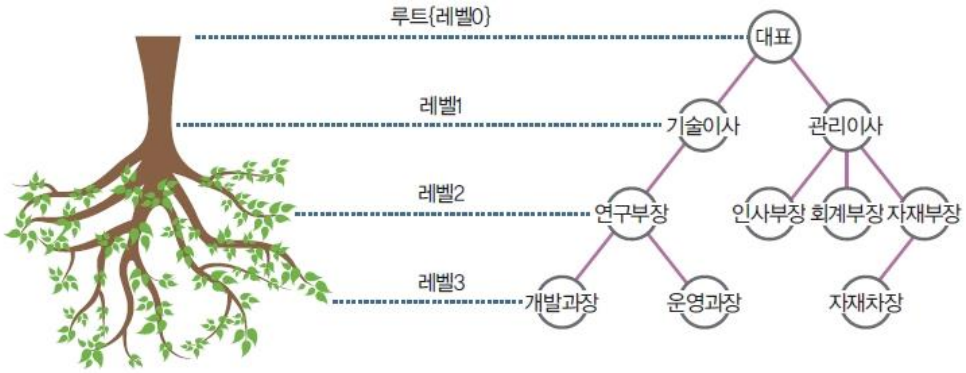




Section 01 이진 트리의 기본

이진 트리의 개념

- 트리(Tree) 자료구조는 나무를 거꾸로 뒤집어 놓은 형태



(a) 나무를 거꾸로 뒤집은 형태 (b) 트리 자료구조

그림 8-1 나무와 트리 자료구조 비교

- 트리 자료구조 용어

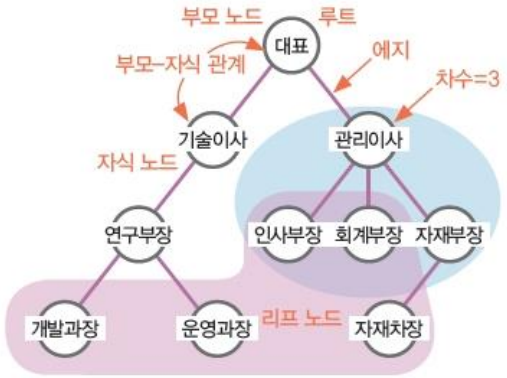


그림 8-2 트리 자료구조의 주요 용어

Section 01 이진 트리의 기본

- 이진 트리 예
  - 모든 노드의 자식이 최대 2개인 트리(자식이 2개 이하로 구성)

그림 8-3 **이진 트리 예**

- 전형적인 이진 트리

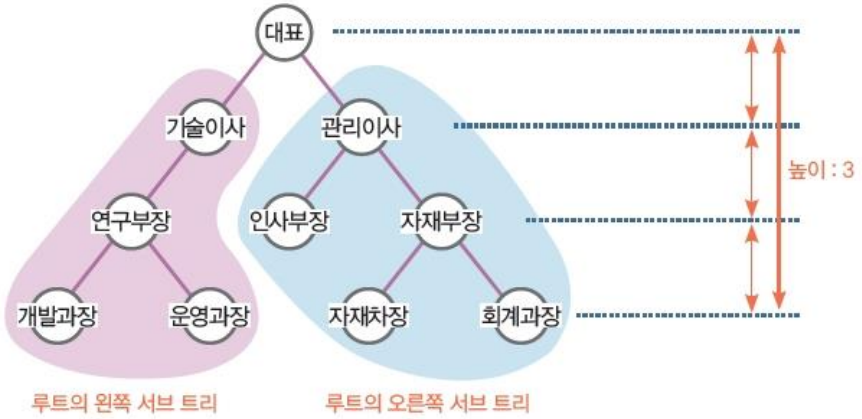


그림 8-4 **전형적인 이진 트리**

- **이진 트리의 종류**
  - 포화 이진 트리, 완전 이진 트리, 편향 이진 트리
    - **포화 이진 트리(full binary tree)**

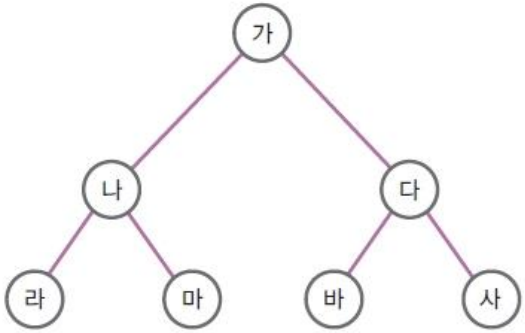


그림 8-5 포화 이진 트리 예

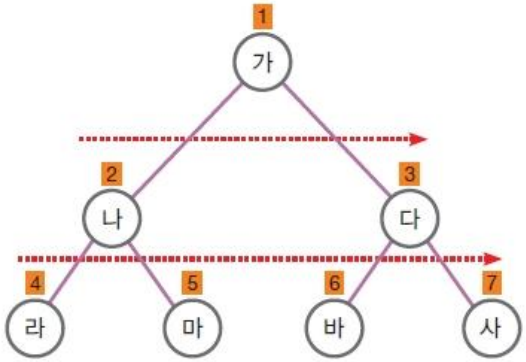


그림 8-6 포화 이진 트리의 번호 부여 순서

    - **완전 이진 트리(complete binary tree)**

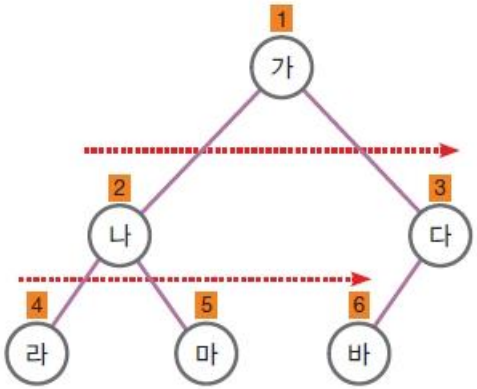
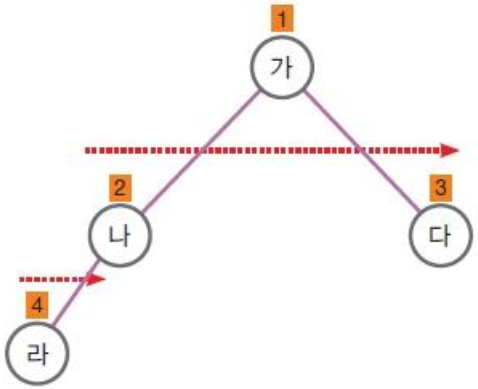


그림 8-7 완전 이진 트리 예



- 일반 이진 트리

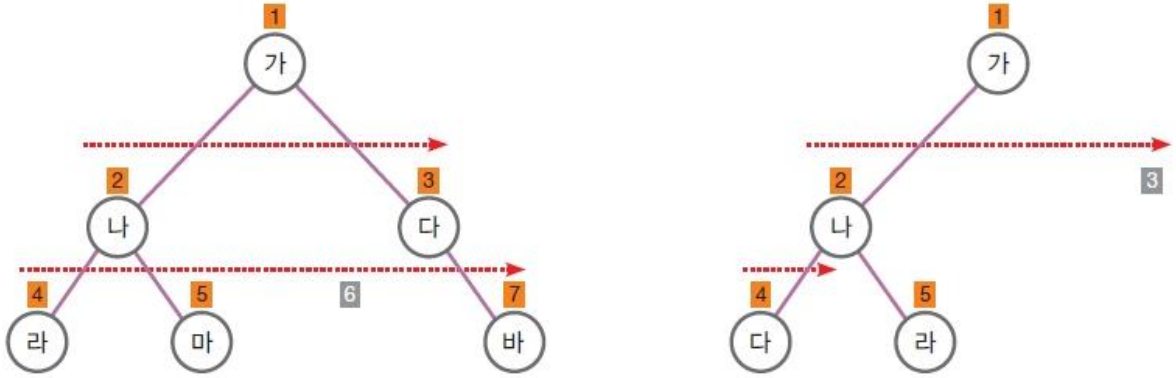
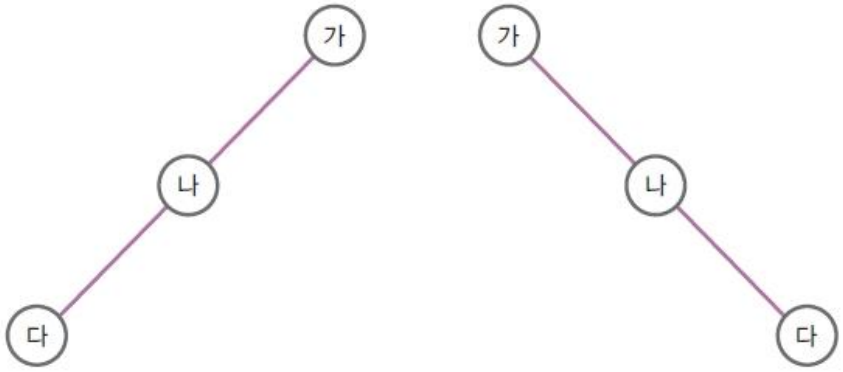


그림 8-8 **일반 이진 트리**

- **편향 이진 트리(skewed binary tree)**



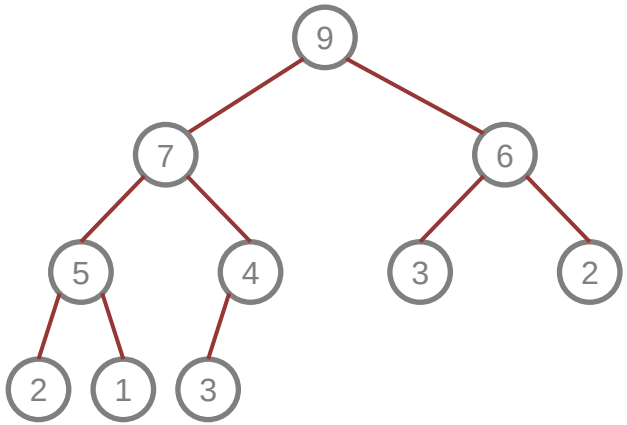
(a) 왼쪽 편향 이진 트리 (b) 오른쪽 편향 이진 트리

그림 8-9 **편향 이진 트리**

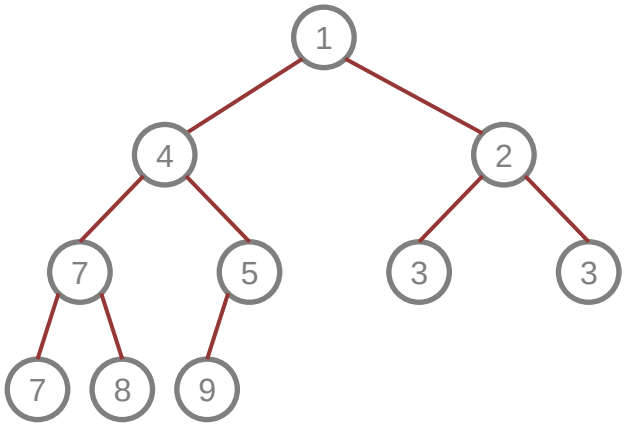
Section 01 이진 트리의 기본

- 힙(heap)
  - 완전 이진 트리의 일종으로 우선순위 큐(Priority Queue)를 위하여 만들어진 자료구조
  - 여러 개의 값들 중에서 최댓값이나 최솟값을 빠르게 찾아내도록 만들어진 자료구조
  - 일종의 반정렬 상태(느슨한 정렬 상태) 를 유지 → 큰 값이 상위 레벨에 있고 작은 값이 하위 레벨에 위치
  - 중복된 값을 허용

- 최대 힙 : 부모노드 >= 자식노드

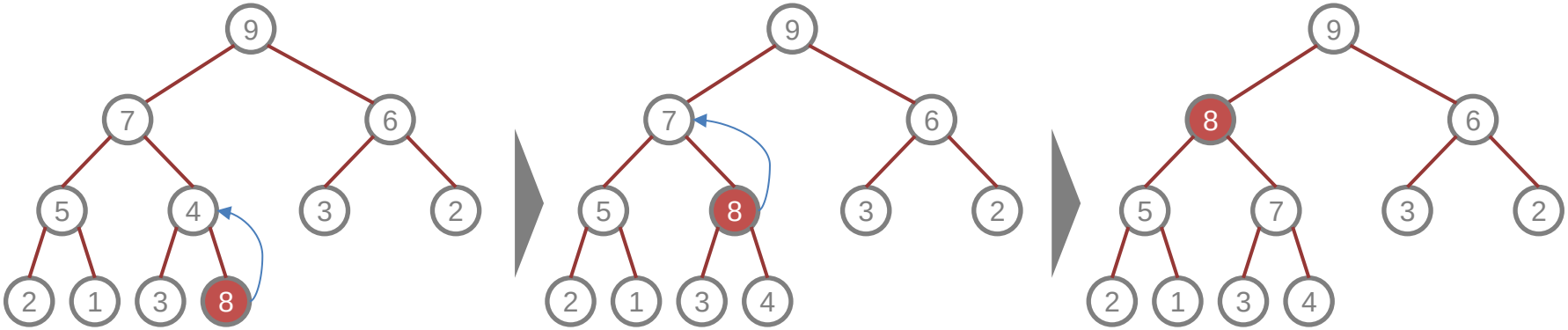


- 최소 힙 : 부모노드 <= 자식노드

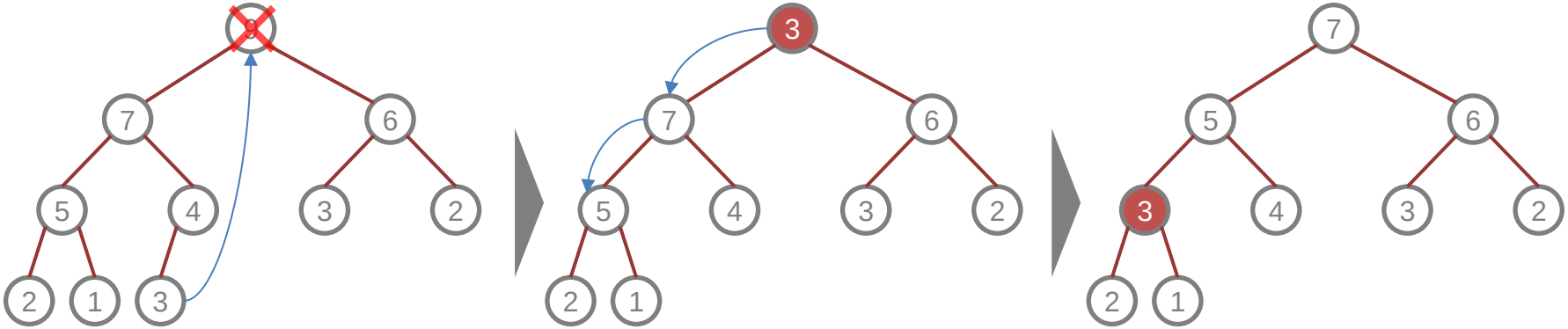


Section 01 이진 트리의 기본

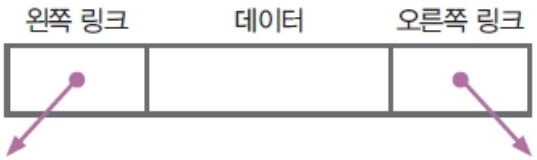
- 힙(heap)의 삽입
  - 새로운 노드를 힙의 마지막 노드에 삽입
  - 새로운 노드의 위치를 이동



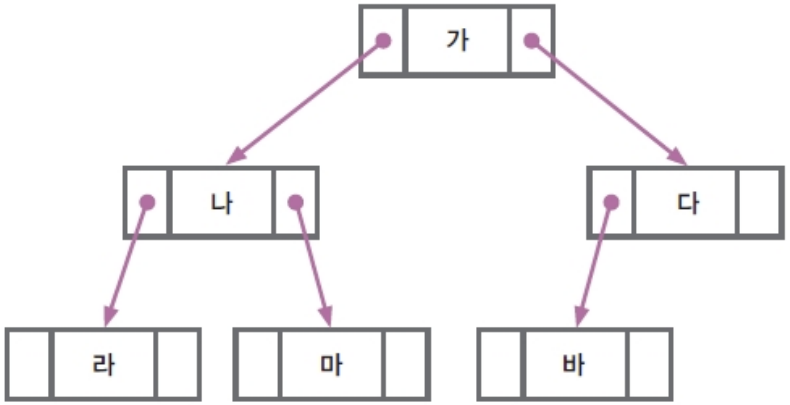
- 힙(heap)의 삭제
  - 삭제된 노드(최대힙에서 루트노드)에 마지막 노드를 가져옴
  - 이동된 노드 기준으로 힙을 재구성



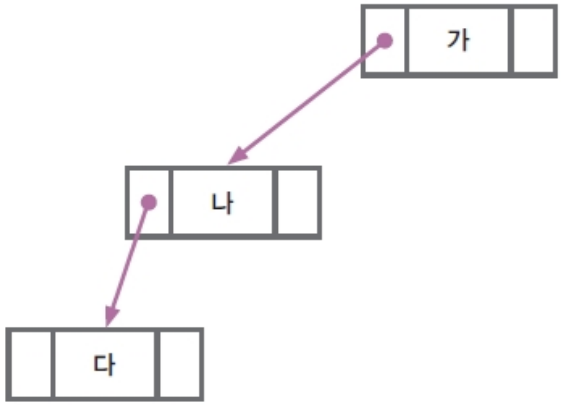
- **이진 트리의 노드 구조**



(a) 트리 노드의 구현을 위한 이중 연결 리스트



(b) 완전 이진 트리의 표현



(c) 편향 이진 트리의 표현

그림 8-10 **이중 연결 리스트를 이용한 트리 노드 표현**

이진 트리의 생성

- 높이가 2고 데이터가 6개인 완전 이진 트리 생성 예

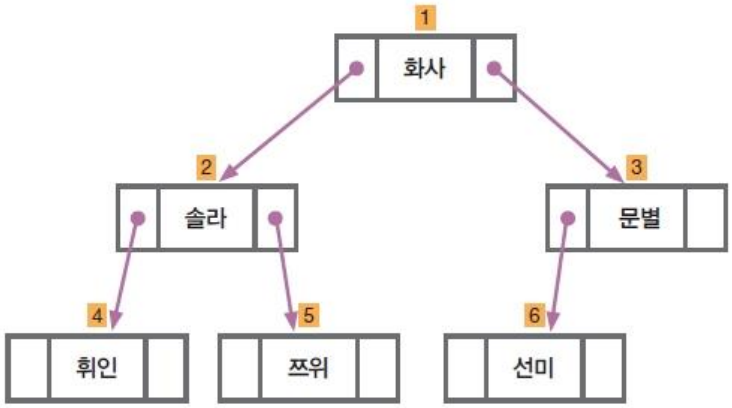
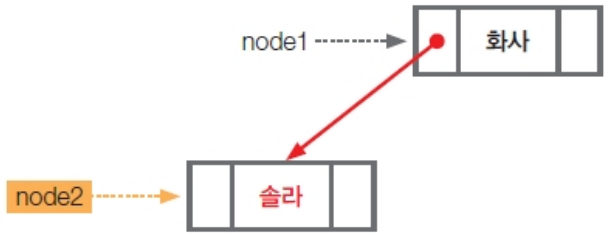


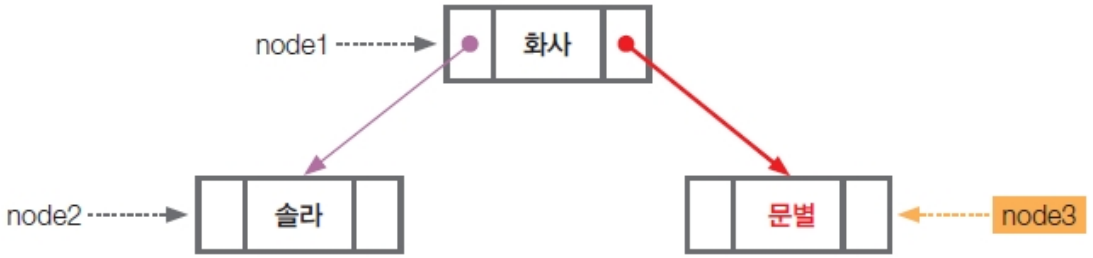
그림 8-11 생성할 이진 트리 예

1 루트 노드(화사)를 생성한다.

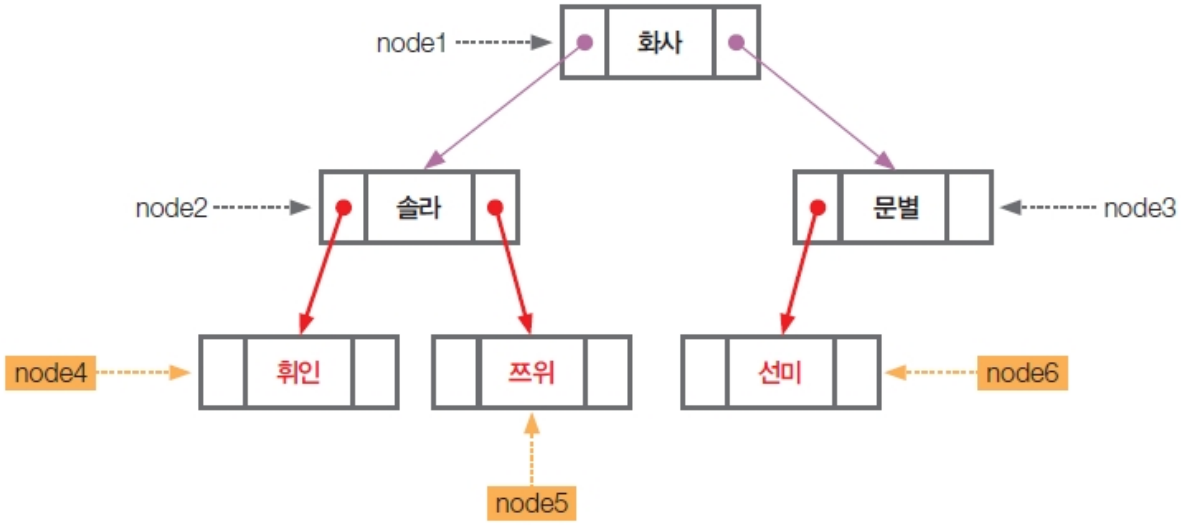
2 두 번째 노드(솔라)를 생성하고 루트 노드의 왼쪽 노드로 지정한다.



3 세 번째 노드(문별)를 생성하고 루트 노드의 오른쪽 노드로 지정한다.



4 네 번째부터 여섯 번째까지 노드를 생성하고 부모 노드와 연결한다.



Section 02 이진 트리의 간단 구현

Code08-01.py 높이가 2인 완전 이진 트리의 생성

```
1  class TreeNode() :          # 이진 트리 노드 생성
2      def __init__ (self) :
3          self.left = None
4          self.data = None
5          self.right = None
6
7  node1 = TreeNode()
8  node1.data = '화사'
9
10 node2 = TreeNode()
11 node2.data = '솔라'
12 node1.left = node2
13
14 node3 = TreeNode()
15 node3.data = '문별'
16 node1.right = node3
17
```

Section 02 이진 트리의 간단 구현

```
18 node4 = TreeNode()
19 node4.data = '휘인'
20 node2.left = node4
21
22 node5 = TreeNode()
23 node5.data = '쯔위'
24 node2.right = node5
25
26 node6 = TreeNode()
27 node6.data = '선미'
28 node3.left = node6
29
30 print(node1.data, end = ' ')
31 print()
32 print(node1.left.data, node1.right.data, end = ' ')
33 print()
34 print(node1.left.left.data, node1.left.right.data, node1.right.left.data, end = ' ')
```

실행 결과

```
화사
솔라 문별
휘인 쯔위 선미
```

Section 02 이진 트리의 간단 구현

▪ 이진 트리의 순회

▪ 순회 종류

- 이진 트리의 노드 전체를 한 번씩 방문하는 것을 순회(traversal)라고 함
- 노드 데이터를 처리하는 순서에 따라 전위 순회, 중위 순회, 후위 순회

▪ 전위 순회(preorder traversal)

❶ 현재 노드 데이터 처리
❷ 왼쪽 서브 트리로 이동
❸ 오른쪽 서브 트리로 이동

▪ 중위 순회(inorder traversal)

❶ 왼쪽 서브 트리로 이동
❷ 현재 노드 데이터 처리
❸ 오른쪽 서브 트리로 이동

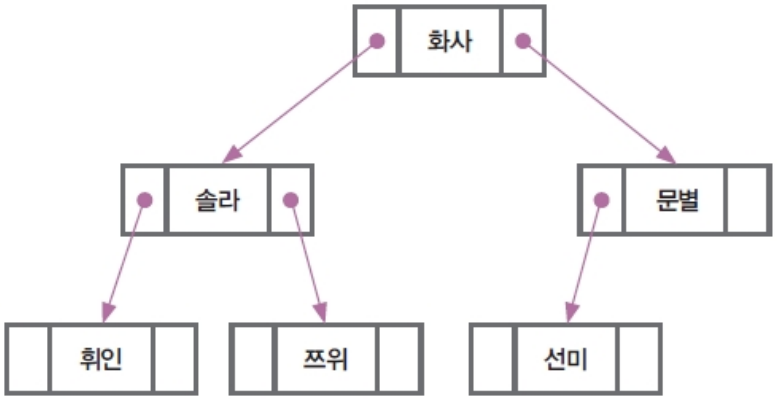
▪ 후위 순회(postorder traversal)

❶ 왼쪽 서브 트리로 이동
❷ 오른쪽 서브 트리로 이동
❸ 현재 노드 데이터 처리

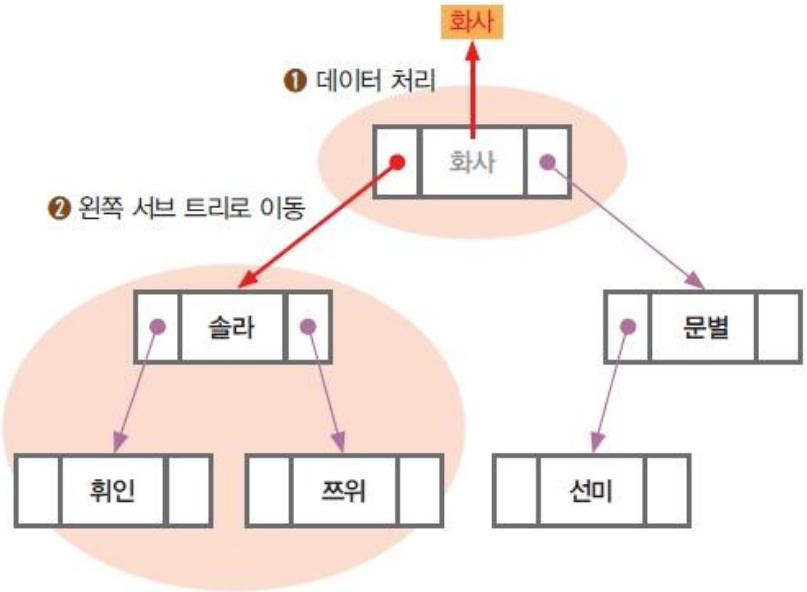
Section 02 이진 트리의 간단 구현

- 전위 순회 작동

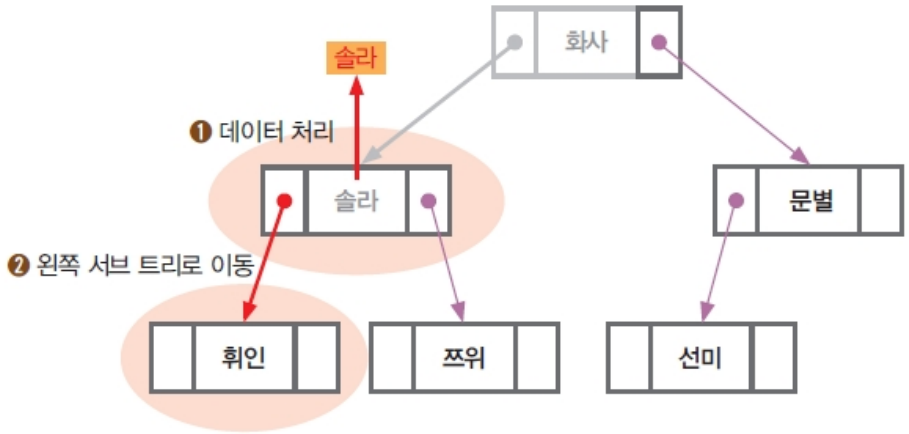
0 전위 순회할 이진 트리



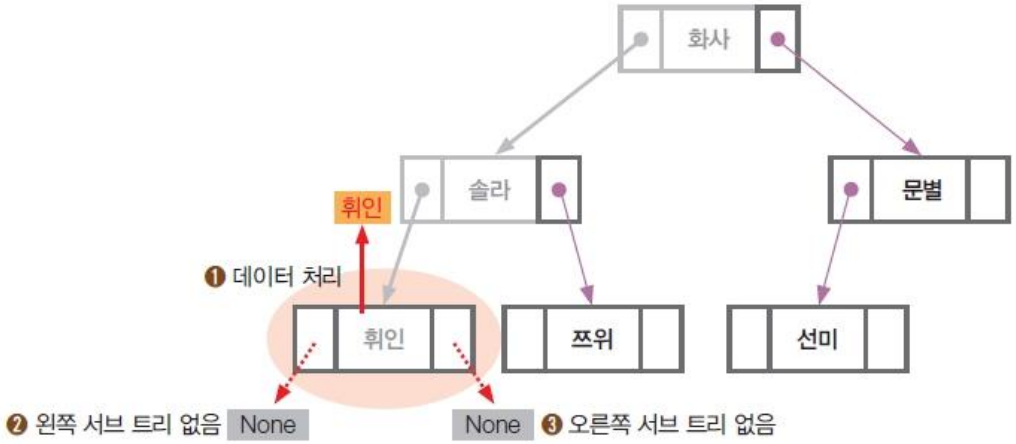
1 루트 노드(화사)의 데이터를 처리하고 왼쪽 서브 트리로 이동한다.



2 이동한 왼쪽 서브 트리의 솔라 데이터를 처리하고 다시 왼쪽 서브 트리로 이동한다.



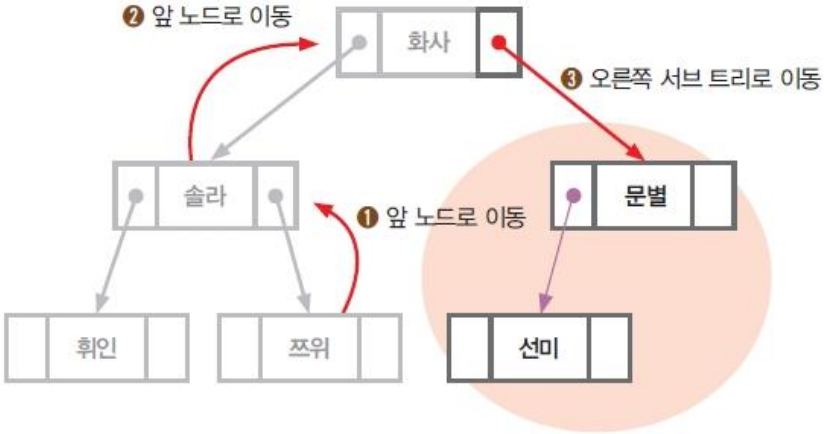
3 이동한 왼쪽 서브 트리의 휘인 데이터를 먼저 처리하고, 다시 왼쪽 서브 트리를 처리하려고 하나 왼쪽 서브 트리가 없어 오른쪽 서브 트리를 처리하려고 한다. 그런데 오른쪽 서브 트리도 없으므로 휘인 노드는 처리가 완료되었다.



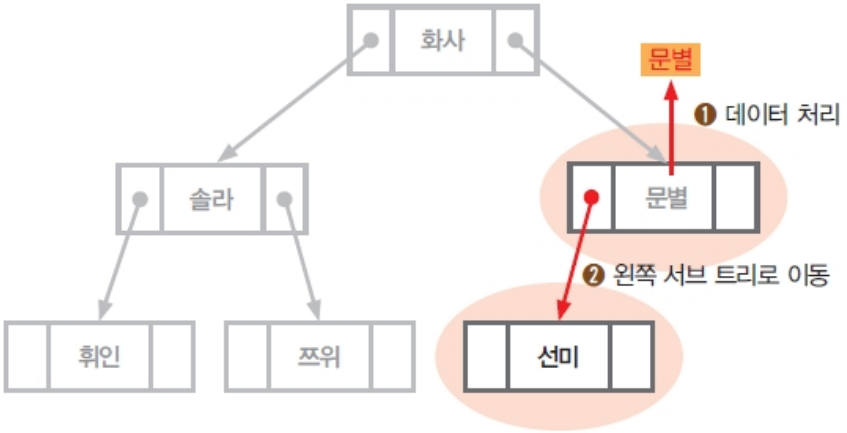
4 현재 노드(휘인 노드)는 더 이상 처리할 것이 없으므로 앞 노드로 올라가서 처리하지 않았던 오른쪽 서브 트리로 내려간다.

5 이동한 오른쪽 서브 트리의 쯔위 데이터를 먼저 처리하고, 왼쪽 서브 트리를 처리하려고 하나 왼쪽 서브 트리가 없어 오른쪽 서브 트리를 처리하려고 한다. 그런데 오른쪽 서브 트리도 없으므로 쯔위 노드는 처리가 완료되었다.

6 현재 노드(쯔위 노드)는 더 이상 처리할 것이 없으므로 앞 노드(솔라 노드)로 올라간다. 앞 노드도 처리가 완료되었으므로 다시 앞 노드(화사 노드)로 올라가 오른쪽 서브 트리로 이동한다.



7 이동한 오른쪽 서브 트리의 문별 데이터를 먼저 처리하고 다시 왼쪽 서브 트리로 이동한다.



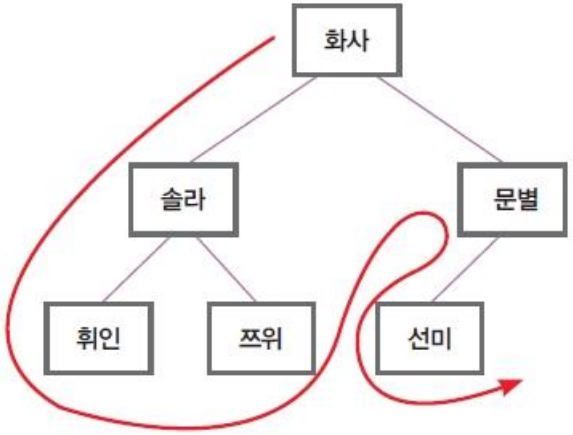
Section 02 이진 트리의 간단 구현

8 이동한 왼쪽 서브 트리의 선미 데이터를 먼저 처리하고, 다시 왼쪽 서브 트리를 처리하려고 하나 왼쪽 서브 트리가 없어 오른쪽 서브 트리를 처리하려고 한다. 그런데 오른쪽 서브 트리도 없으므로 선미 노드는 처리가 완료되었다. 그리고 트리의 모든 노드 순회가 완료되었다.

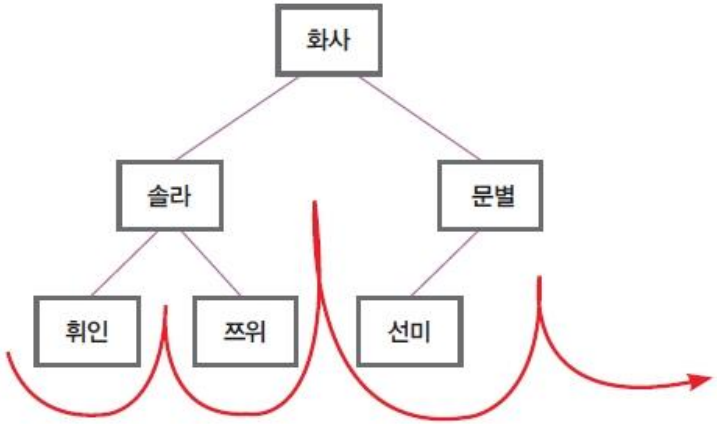
1 ~ 8 에서 출력된 데이터를 확인하면 화사, 솔라, 휘인, 쯔위, 문별, 선미 순이다.
즉, 전위 순회인 현재 데이터 → 왼쪽 서브 트리 → 오른쪽 서브 트리 순서로 출력된 것을 확인할 수 있다.

Section 02 이진 트리의 간단 구현

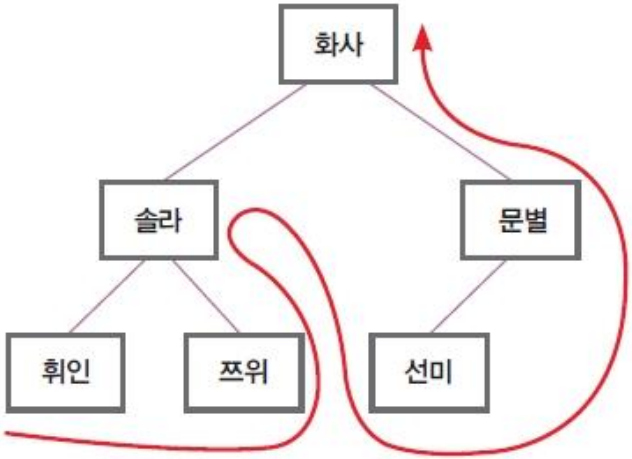
- 좀 더 간단한 트리 순회
  - 전위 순회 : 루트 → 왼쪽 → 오른쪽



  - 중위 순회 : 왼쪽 → 루트 → 오른쪽



  - 후위 순회 : 왼쪽 → 오른쪽 → 루트



Section 02 이진 트리의 간단 구현

- 이진 트리 순회 구현

Code08-02.py 이진 트리의 순회(재귀 함수 사용)

```
… # 생략(Code08-01.py의 1~28행과 동일)
29
30 def preorder(node) :
31     if node == None :
32         return
33     print(node.data, end = '->')
34     preorder(node.left)
35     preorder(node.right)
36
37 def inorder(node) :
38     if node == None :
39         return
40     inorder(node.left)
41     print(node.data, end = '->')
42     inorder(node.right)
43
44 def postorder(node) :
45     if node == None :
46         return
47     postorder(node.left)
48     postorder(node.right)
49     print(node.data, end = '->')
```

Section 02 이진 트리의 간단 구현

```
50
51 print('전위 순회 : ', end = ' ')
52 preorder(node1)
53 print('끝')
54
55 print('중위 순회 : ', end = ' ')
56 inorder(node1)
57 print('끝')
58
59 print('후위 순회 : ', end = ' ')
60 postorder(node1)
61 print('끝')
```

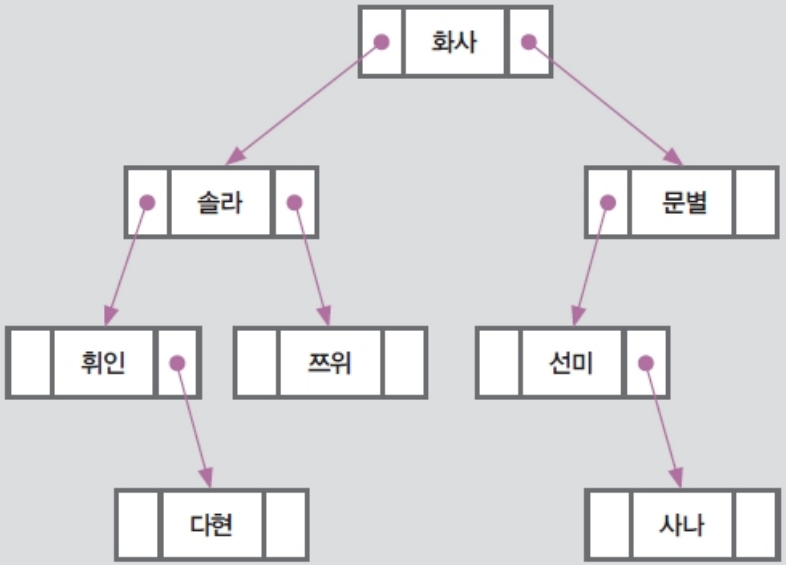
실행 결과

```
전위 순회 : 화사->솔라->휘인->쯔위->문별->선미->끝
중위 순회 : 휘인->솔라->쯔위->화사->선미->문별->끝
후위 순회 : 휘인->쯔위->솔라->선미->문별->화사->끝
```

Section 02 이진 트리의 간단 구현

SELF STUDY 8-1

Code08-02.py를 수정해서 다음 그림과 같이 만들고, 전위/중위/후위 순회를 시켜 보자.



실행 결과

```
전위 순회 : 화사->솔라->휘인->다현->쯔위->문별->선미->사나->끝
중위 순회 : 휘인->다현->솔라->쯔위->화사->선미->사나->문별->끝
후위 순회 : 다현->휘인->쯔위->솔라->사나->선미->문별->화사->끝
```

이진 탐색 트리의 특징

- 이진 트리 중 활용도가 높은 트리로, 데이터 크기를 기준으로 일정 형태로 구성함

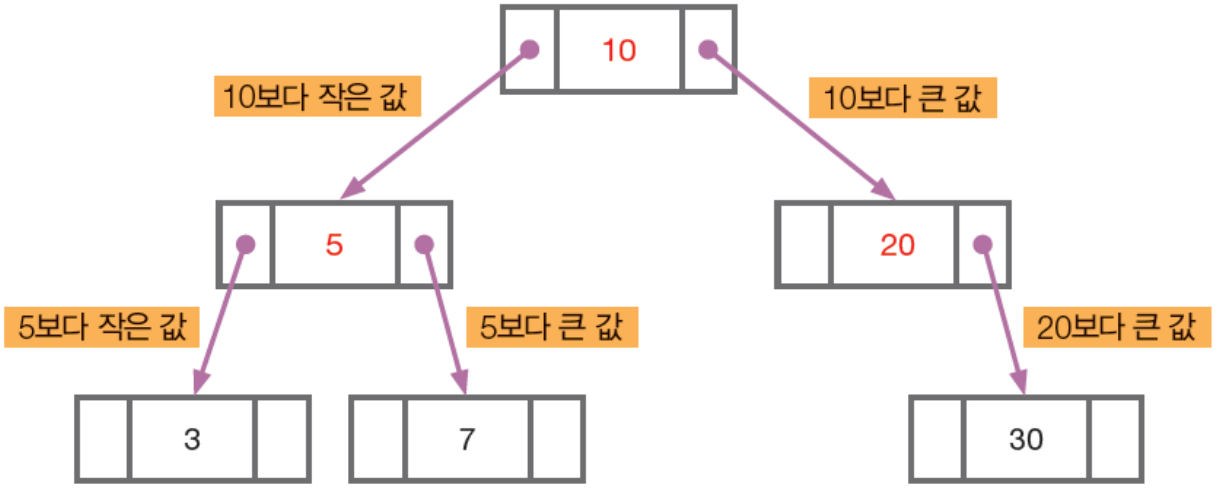


그림 8-12 이진 탐색 트리의 대표적인 형태

이진 탐색 트리 특징

❶ 왼쪽 서브 트리는 루트 노드보다 모두 작은 값을 가진다.

❷ 오른쪽 서브 트리는 루트 노드보다 모두 큰 값을 가진다.

❸ 각 서브 트리도 ❶, ❷ 특징을 갖는다.

❹ 모든 노드 값은 중복되지 않는다. 즉, 중복된 값은 이진 탐색 트리에 저장할 수 없다.

이진 탐색 트리의 생성

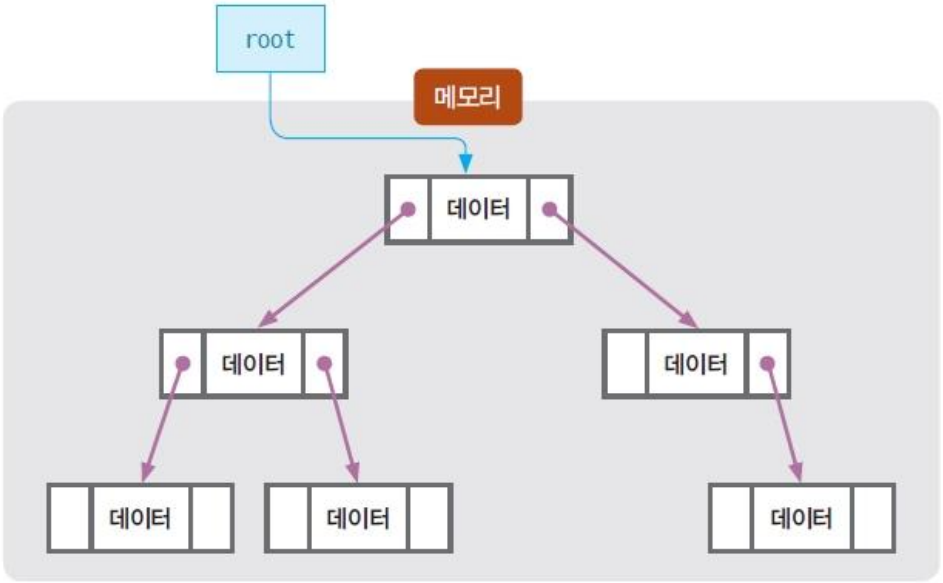


그림 8-13 생성할 이진 탐색 트리

- 메모리를 준비하고 root는 None으로 초기화

```
memory = []
root = None
```

Section 03 이진 탐색 트리의 일반 구현

- 배열에 있는 데이터를 차례대로 이진 탐색 트리에 삽입

```
nameAry = ['블랙핑크', '레드벨벳', '마마무', '에이핑크', '걸스데이', '트와이스']
```

- 첫 번째 데이터 삽입

```
❶ node = TreeNode()          # 노드 생성
❷ node.data = nameAry[0]     # 데이터 입력
❸ root = node                # 첫 번째 노드를 루트 노드로 지정
❹ memory.append(node)        # 생성한 노드를 메모리에 저장
```

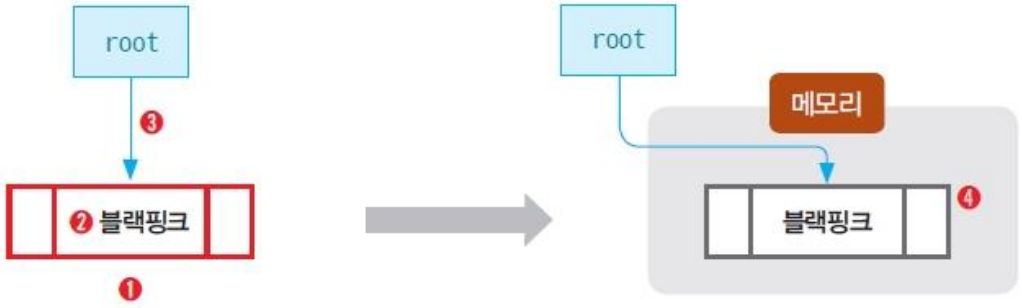


그림 8-14 첫 번째 데이터 삽입

- 두 번째 이후 데이터 삽입

```
name = '레드벨벳'                # 두 번째 데이터
node = TreeNode()                # 새 노드 생성
node.data = name                 # 새 노드에 데이터 입력
current = root                   # 현재 작업 노드를 루트 노드로 지정
if name < current.data :         # 입력할 값을 현재 작업 노드의 값과 비교
    current.left = node          # 작으면 새 노드를 왼쪽 링크로 연결
else :
    current.right = node         # 크면 새 노드를 오른쪽 링크로 연결
memory.append(node)              # 새 노드를 메모리에 저장
```

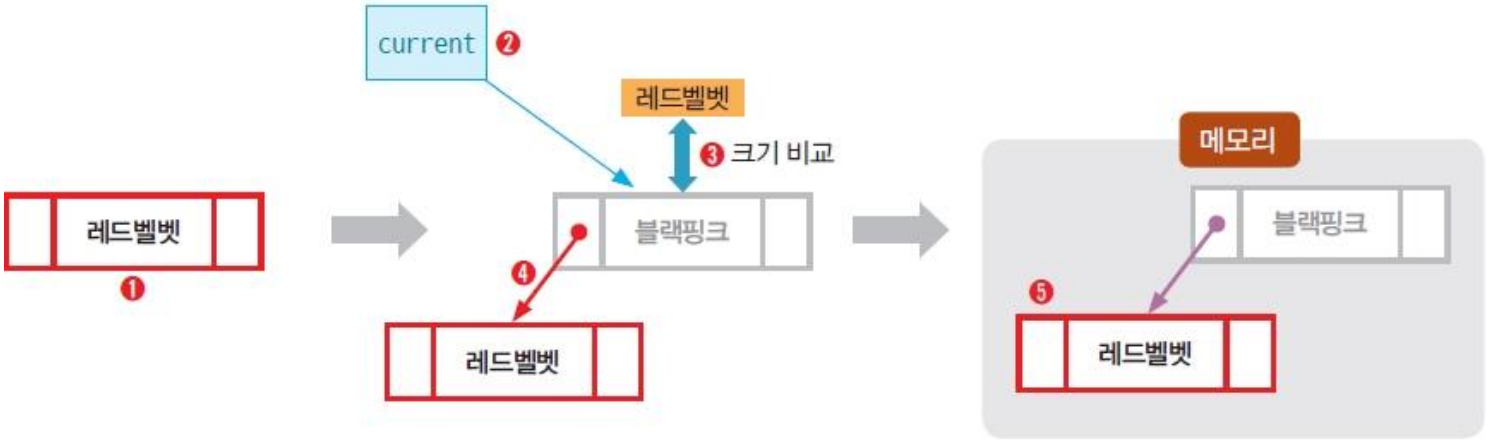


그림 8-15 두 번째 데이터 삽입

- **이진 탐색 트리에서 데이터를 삽입하는 일반적인 형태**
  - 레벨 2의 초기 상태인 이진 탐색 트리에 6 데이터를 삽입하는 과정

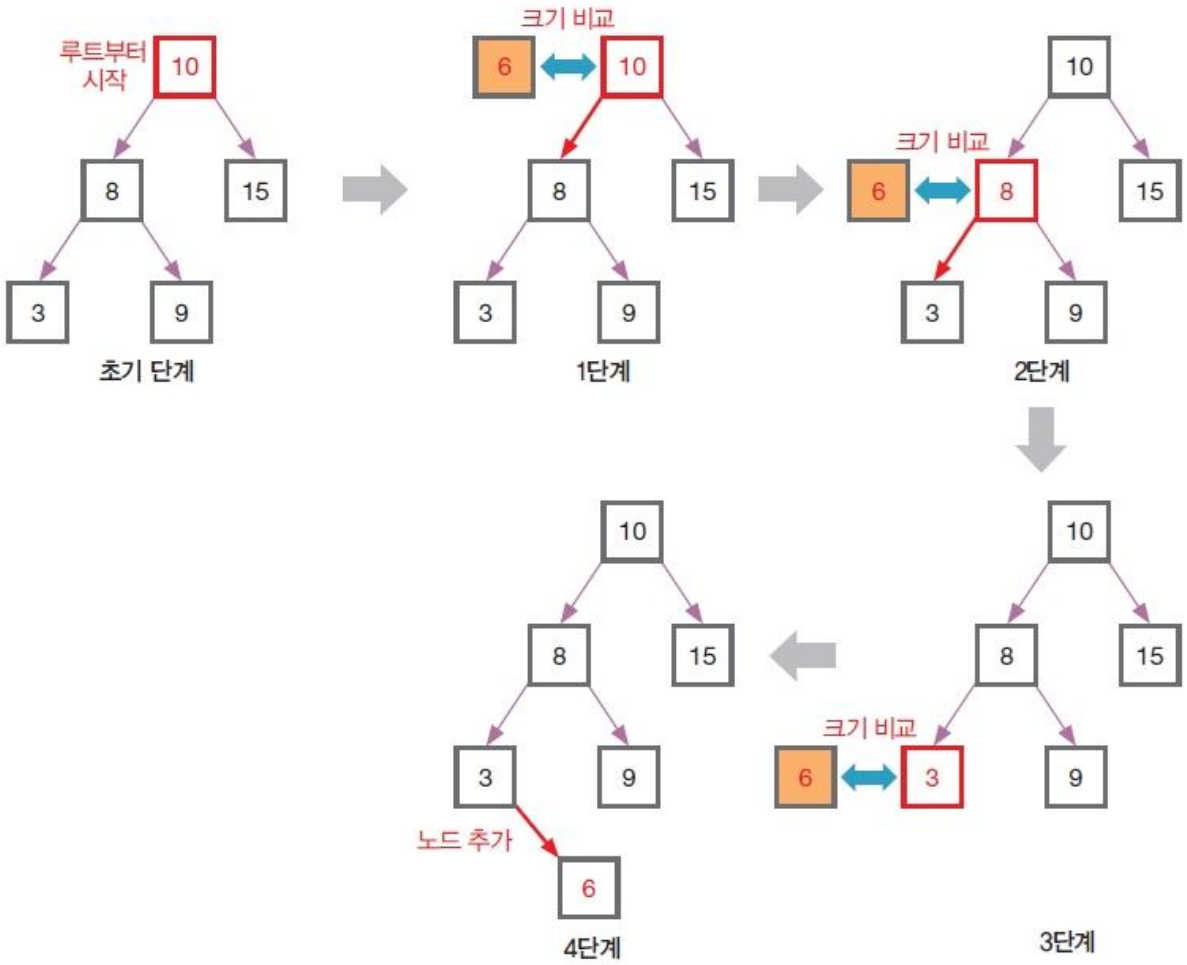


그림 8-16 레벨이 2인 이진 탐색 트리에서 6 데이터를 삽입하는 과정

Section 03 이진 탐색 트리의 일반 구현

- [그림 8-16]과 같이 반복적으로 자리를 찾아가는 과정을 코드 형태로 구현

```
name = 6                              # 위치를 찾을 새 데이터
node = TreeNode()                     # 새 노드 생성
node.data = name

❶current = root                       # 루트부터 시작
❷while True :                         # 무한 반복
    ❸if name < current.data :         # 1단계
        if current.left == None :     # 4단계
    ❹       current.left = node       # 4단계
            break
      ❺current = current.left         # 2~3단계
    else :
        if current.right == None :    # 4단계
    ❻       current.right = node      # 4단계
            break
        current = current.right
```

Section 03 이진 탐색 트리의 일반 구현

- 이진 탐색 트리에서 데이터를 삽입하는 과정 통합의 코드

Code08-03.py 이진 탐색 트리의 삽입 작동

```
## 함수 선언 부분 ##
class TreeNode() :
    def __init__ (self) :
        self.left = None
        self.data = None
        self.right = None

## 전역 변수 선언 부분 ##
memory = []
root = None
nameAry = ['블랙핑크', '레드벨벳', '마마무', '에이핑크', '걸스데이', '트와이스']

## 메인 코드 부분 ##
node = TreeNode()
node.data = nameAry[0]
root = node
memory.append(node)

for name in nameAry[1:] :

    node = TreeNode()
    node.data = name

```

Section 03 이진 탐색 트리의 일반 구현

```
24      current = root
25      while True :
26          if name < current.data :
27              if current.left == None :
28                  current.left = node
29                  break
30              current = current.left
31          else :
32              if current.right == None :
33                  current.right = node
34                  break
35              current = current.right
36
37      memory.append(node)
38
39  print("이진 탐색 트리 구성 완료!")
```

실행 결과

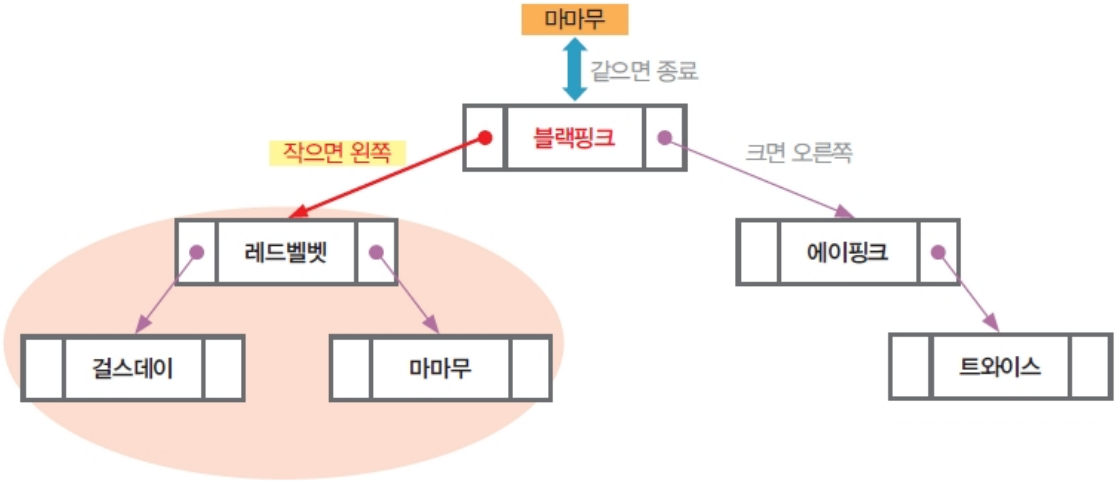
```
이진 탐색 트리 구성 완료!
```

이진 탐색 트리에서 데이터 검색

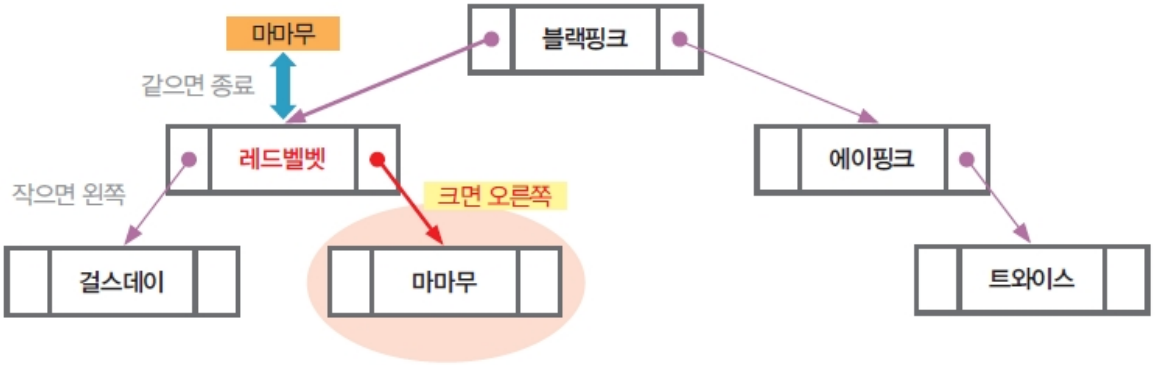
```
1 if 현재 작업 노드의 데이터 == 찾을 데이터 :
      탐색 종료
2 elif 현재 작업 노드의 데이터 < 찾을 데이터 :
      왼쪽 서브 트리 탐색
3 else :
      오른쪽 서브 트리 탐색
```

- 완성된 이진 탐색 트리에서 마마무를 찾는 예

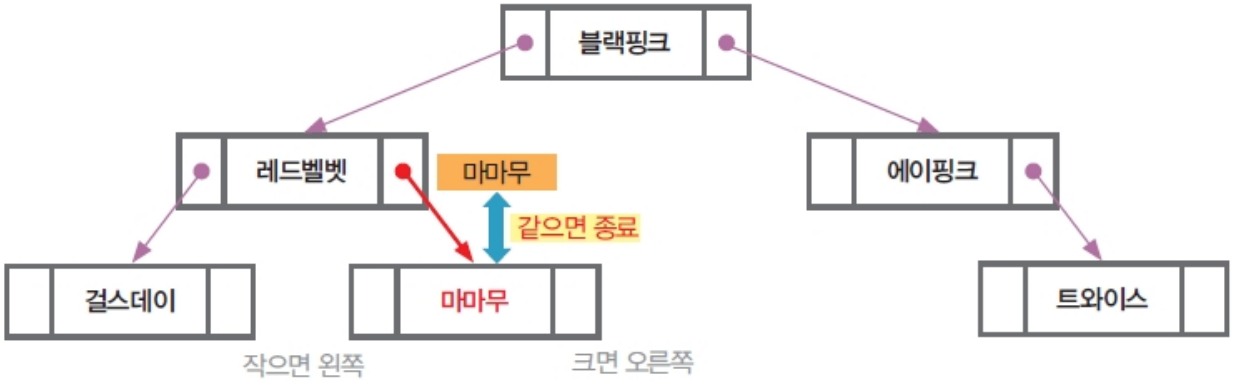
1 찾고자 하는 마마무를 루트 노드의 데이터와 비교한다. 마마무가 루트 노드의 데이터보다 작아 왼쪽으로 이동한다.



2 왼쪽 서브 트리에서도 동일하게 처리한다. 찾고자 하는 마마무가 왼쪽 서브 트리의 루트 노드보다 커 오른쪽으로 이동한다.



3 오른쪽 서브 트리에서도 동일하게 처리한다. 그런데 여기에서는 마마무를 찾았으므로 종료한다.



Section 03 이진 탐색 트리의 일반 구현

Code08-04.py 이진 탐색 트리의 검색 작동

```
## 함수 선언 부분 ##
... # 생략(Code08-03.py의 2~37행과 동일)

findName = '마마무'

current = root
while True :
    if findName == current.data :
        print(findName, '을(를) 찾음.')
        break
    elif findName < current.data :
        if current.left == None :
            print(findName, '이(가) 트리에 없음')
            break
        current = current.left
    else :
        if current.right == None :
            print(findName, '이(가) 트리에 없음')
            break
        current = current.right
```

실행 결과

```
마마무 을(를) 찾음.
```

Section 03 이진 탐색 트리의 일반 구현

SELF STUDY 8-2

Code08-04.py를 수정해서 nameAry에 잇지와 여자친구를 추가하자. 그리고 검색할 이름을 input() 함수로 입력받은 후 검색하도록 하자.

실행 결과

```
찾을 그룹이름-->잇지
잇지 을(를) 찾았음.
찾을 그룹이름-->소녀시대
소녀시대 이(가) 트리에 없음
```

이진 탐색 트리에서 데이터 삭제

- 리프 노드(맨 아래쪽 노드)를 삭제하는 경우

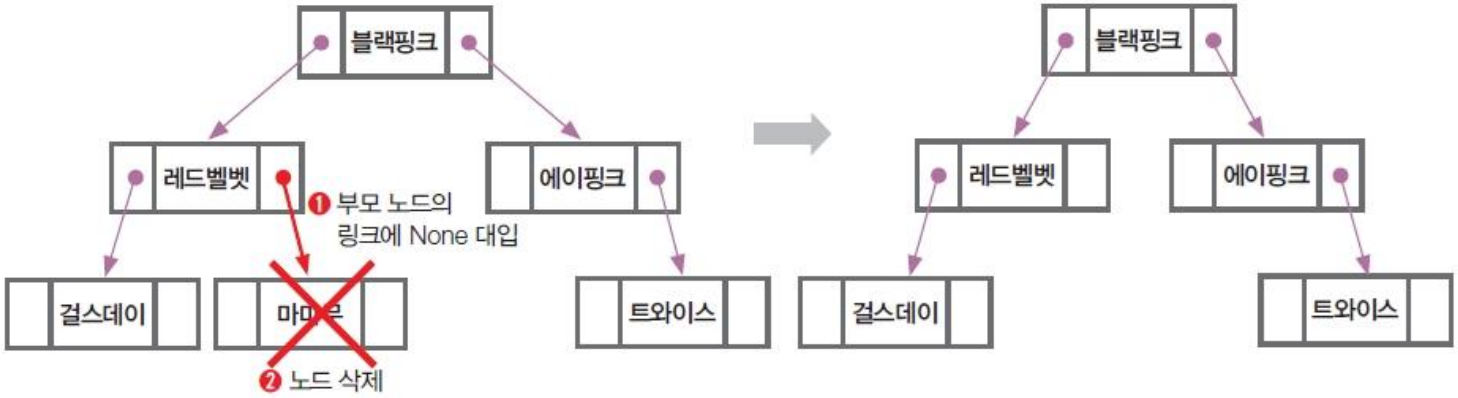


그림 8-17 리프 노드 삭제

- 현재 노드가 부모 노드의 왼쪽 링크인지, 오른쪽 링크인지 구분하고자 코드 형태로 구현

```
if parent.left == current :        # 부모 노드 왼쪽 링크와 삭제할 노드가 같으면
    parent.left = None             # 부모 노드의 왼쪽에 None 대입
else :                             # 부모 노드 오른쪽 링크와 삭제할 노드가 같으면
    parent.right = None            # 부모 노드의 오른쪽에 None 대입

del(current)                       # 노드 삭제
```

- 자식 노드가 하나인 노드를 삭제하는 경우

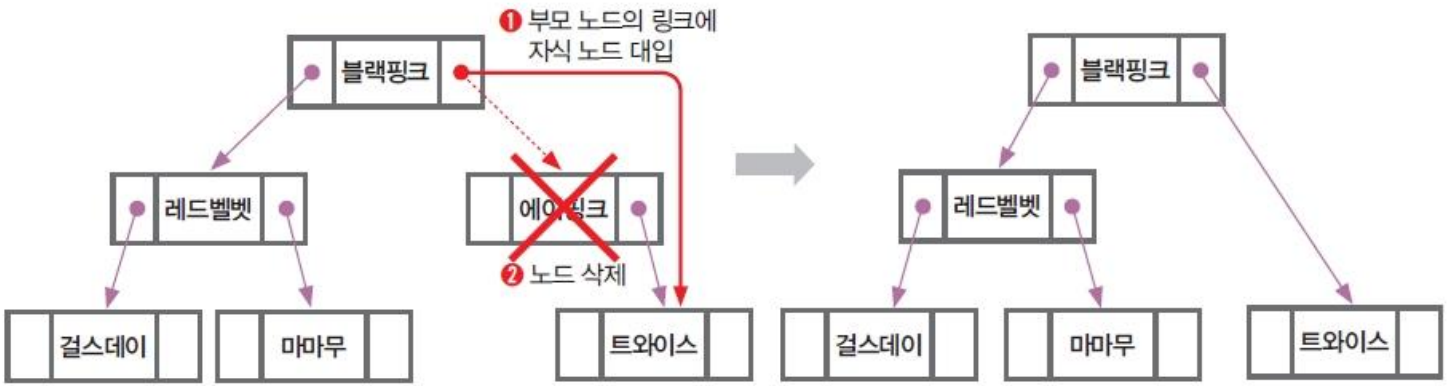


그림 8-18 자식 노드가 1개인 노드 삭제

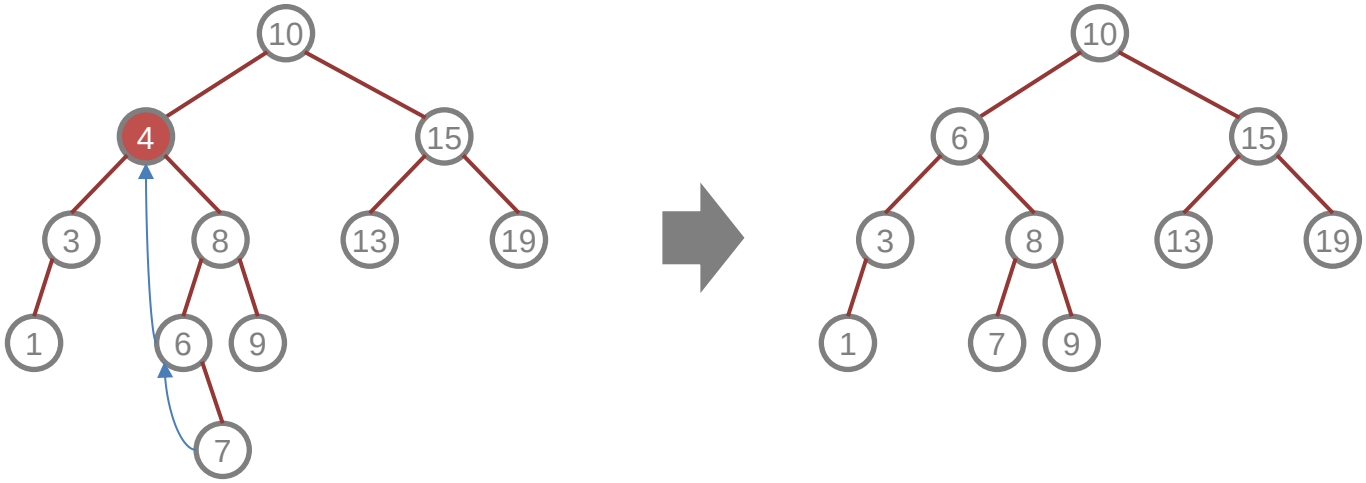
- **왼쪽 자식 노드가 있는 경우**

```
if parent.left == current :          # 부모 노드 왼쪽 링크와 삭제할 노드가 같으면
    parent.left = current.left       # 부모 노드의 왼쪽 링크에 왼쪽 자식 노드 대입
else :                               # 부모 노드 오른쪽 링크와 삭제할 노드가 같으면
    parent.right = current.left      # 부모 노드의 오른쪽 링크에 왼쪽 자식 노드 대입

del(current)
```

- **오른쪽 자식 노드가 있는 경우**

```
if parent.left == current :          # 부모 노드 왼쪽 링크와 삭제할 노드가 같으면
    parent.left = current.right      # 부모 노드의 왼쪽 링크에 오른쪽 자식 노드 대입
else :                               # 부모 노드 오른쪽 링크와 삭제할 노드가 같으면
    parent.right = current.right     # 부모 노드의 오른쪽 링크에 오른쪽 자식 노드 대입

del(current)
```

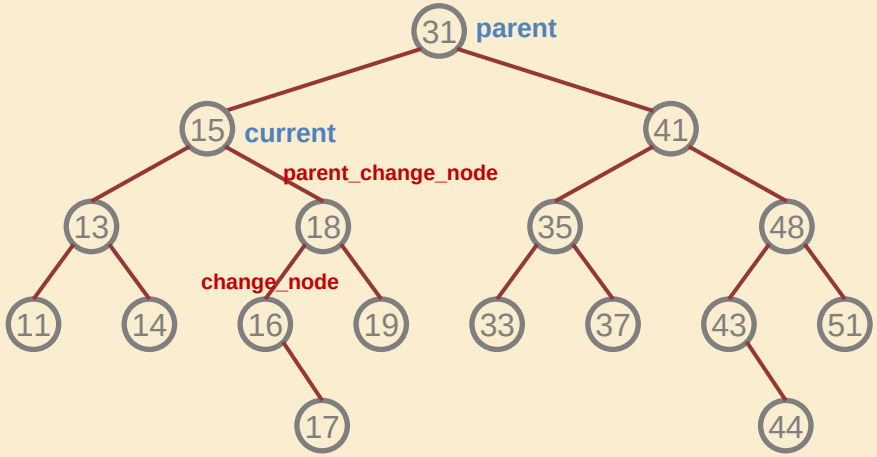
- 자식 노드가 둘 있는 노드를 삭제하는 경우
  - 삭제할 Node의 오른쪽 자식 중 가장 작은 값으로 대체(Parent Node와 연결)
  - 또는 삭제할 노드의 왼쪽 자식 중, 가장 큰 값으로 대체(Parent Node와 연결)



Section 03 이진 탐색 트리의 일반 구현

- **자식 노드가 둘 있는 경우(Code08-051.py)**

```
change_node = current.right
parent_change_node = current.right

while change_node.left != None :
    parent_change_node = change_node
    change_node = change_node.left

if change_node.right != None :
    parent_change_node.left = change_node.right
else :
    parent_change_node.left = None

# change node 이동
parent.left = change_node
change_node.left = current.left
change_node.right = current.right
```



- 이진 탐색 트리에서 데이터 삭제 완성(코드로 구현)

Code08-05.py 이진 탐색 트리의 삭제 작동

```
1   ## 함수 선언 부분 ##
…   # 생략(Code08-03.py의 2~37행과 동일)
38
39  deleteName = '마마무'
40
41  current = root
42  parent = None
43  while True :
44      if deleteName == current.data :
45
```

```
46          if current.left == None and current.right == None :
47              if parent.left == current :
48                  parent.left = None
49              else :
50                  parent.right = None
51              del(current)
52
53          elif current.left != None and current.right == None :
54              if parent.left == current :
55                  parent.left = current.left
56              else :
57                  parent.right = current.left
58              del(current)
59
60          elif current.left == None and current.right != None :
61              if parent.left == current :
62                  parent.left = current.right
63              else :
64                  parent.right = current.right
65              del(current)
66
67          print(deleteName, '이(가) 삭제됨.')
68          break
69      elif deleteName < current.data :
70          if current.left == None :
71              print(deleteName, '이(가) 트리에 없음')
```

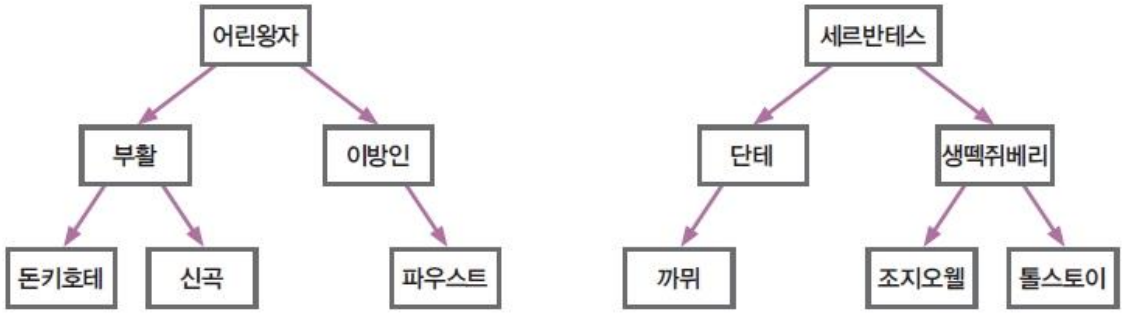
```
72              break
73          parent = current
74          current = current.left
75      else :
76          if current.right == None :
77              print(deleteName, '이(가) 트리에 없음')
78              break
79          parent = current
80          current = current.right
```

실행 결과

```
마마무 이(가) 삭제됨.
```

Section 04 이진 탐색 트리의 응용

- 이진 탐색 트리는 데이터를 보관하고 검색할 때 효율적
  - 도서관에 새로 입고된 책 정보를 이진 탐색 트리에 보관해서 검색하는 예



(a) 책 이름 이진 탐색 트리 (b) 작가 이름 이진 탐색 트리

그림 8-19 책 이름 및 작가 이름으로 생성한 이진 탐색 트리

```
import random
bookAry = [['어린왕자', '생떽쥐베리'], ['이방인', '까뮈'], ['부활', '톨스토이'],
           ['신곡', '단테'], ['돈키호테', '세르반테스'], ['동물농장', '조지오웰'],
           ['데미안', '헤르만헤세'], ['파우스트', '괴테'], ['대지', '펄벅']]
bookAry = random.shuffle(bookAry)
```

```
rootBook, rootAuth = None, None
```

Section 04 이진 탐색 트리의 응용

- 책 이름 트리와 작가 이름 트리를 따로 구현

Code08-06.py 도서관 책 목록의 보관과 검색

```
import random
## 함수 선언 부분 ##
class TreeNode() :
    def __init__ (self) :
        self.left = None
        self.data = None
        self.right = None

## 전역 변수 선언 부분 ##
memory = []
rootBook, rootAuth = None, None
bookAry = [['어린왕자', '생떽쥐베리'], ['이방인', '까뮈'], ['부활', '톨스토이'],
           ['신곡', '단테'], ['돈키호테', '세르반테스'], ['동물농장', '조지오웰'],
           ['데미안', '헤르만헤세'], ['파우스트', '괴테'], ['대지', '펄벅']]
random.shuffle(bookAry)

```

```
17 ## 메인 코드 부분 ##
18
19 ### 책 이름 트리 ###
20 node = TreeNode()
21 node.data = bookAry[0][0]
22 rootBook = node
23 memory.append(node)
24
25 for book in bookAry[1:] :
26     name = book[0]
27     node = TreeNode()
28     node.data = name
29
30     current = rootBook
31     while True :
32         if name < current.data :
33             if current.left == None :
34                 current.left = node
35                 break
36             current = current.left
37         else :
38             if current.right == None :
39                 current.right = node
40                 break
41             current = current.right
42
43     memory.append(node)
44
```

```
print("책 이름 트리 구성 완료!")

### 작가 이름 트리 ###
node = TreeNode()
node.data = bookAry[0][1]
rootAuth = node
memory.append(node)

for book in bookAry[1:] :
    name = book[1]
    node = TreeNode()
    node.data = name

    current = rootAuth
    while True :
        if name < current.data :
            if current.left == None :
                current.left = node
                break
            current = current.left
        else :
            if current.right == None :
                current.right = node
                break
            current = current.right

    memory.append(node)

print("작가 이름 트리 구성 완료!")
```

```
74
75 ## 책 이름 및 작가 이름 검색 ##
76 bookOrAuth = int(input('책검색(1), 작가검색(2)-->'))
77 findName = input('검색할 책 또는 작가-->')
78
79 if bookOrAuth == 1 :
80     root = rootBook
81 else :
82     root = rootAuth
83
84 current = root
85 while True :
86     if findName == current.data :
87         print(findName, '을(를) 찾음.')
88
89         break
90     elif findName < current.data :
91         if current.left == None :
92             print(findName, '이(가) 목록에 없음')
93             break
94         current = current.left
95     else :
96         if current.right == None :
97             print(findName, '이(가) 목록에 없음')
98             break
99         current = current.right
```

실행 결과

```
책 이름 트리 구성 완료!
작가 이름 트리 구성 완료!
책검색(1), 작가검색(2)-->1
검색할 책 또는 작가-->대지
대지 을(를) 찾음.
```

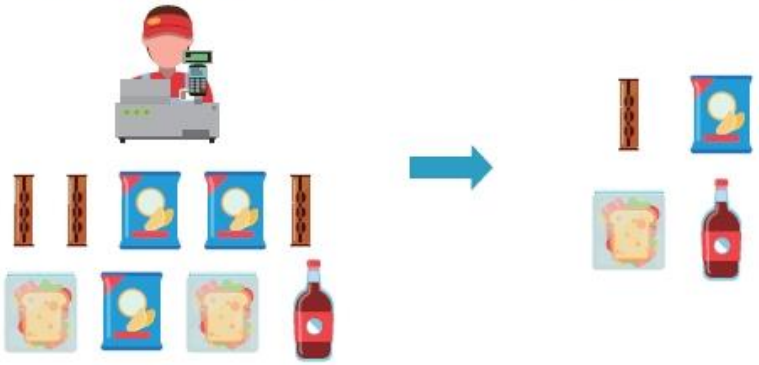
응용예제 01 편의점에서 판매된 물건 목록 출력하기

난이도★★☆☆☆

예제 설명

편의점에서는 매일 다양한 물품을 판매한다. 하루에 판매하는 물건은 당연히 중복해서 여러 개 판매한다. 마감 시간에 오늘 판매된 물건 종류를 살펴볼 때는 중복된 것은 하나만 남기도록 한다. 이진 탐색 트리를 활용해서 중복된 물품은 하나만 남기자.

편의점에서 하루 판매된 물건(중복○)　　오늘 판매된 물건(중복×)



실행 결과

```
Python
File  Edit  Shell  Debug  Options  Window  Help
========== RESTART: C:WCookDataWEx08-01.py ==========
오늘 판매된 물건(중복O) --> ['레쓰비캔커피', '레쓰비캔커피', '레쓰비캔커피', '도시락', '도시락', '삼각김밥', '레쓰비캔커피', '도시락', '코카콜라',
'삼다수', '레쓰비캔커피', '레쓰비캔커피', '레쓰비캔커피', '츄파춥스', '츄파춥스', '레쓰비캔커피', '코카콜라', '츄파춥스', '삼각김밥', '코카콜라']

이진 탐색 트리 구성 완료!

오늘 판매된 종류(중복X)--> 레쓰비캔커피 도시락 삼각김밥 코카콜라 삼다수 츄파춥스
>>>
                                                                    Ln: 22  Col: 4
```

응용예제 02 폴더 및 하위 폴더에 중복된 파일 이름 찾기

난이도★★★☆☆

예제 설명

특정 폴더를 지정해서 해당 폴더 및 그 하위 폴더에 모든 파일을 조회한다. 그리고 이름이 동일한 파일이 있으면 그 이름을 출력한다. 예로 C:/Program Files/Common Files/ 폴더 및 그 하위 폴더 아래에는 이름이 동일한 파일이 몇 개 있다. 단 여러 번 중복되더라도 한 번만 출력하자.

실행 결과

Thank You