13. 탐색 트리

13.1 키와 Comparable 타입 (1)

- Comparable 인터페이스

  public int compareTo(Object object)

- 다음 세 가지 가능성 중의 하나에 대한 정보를 가지고 있는 정수 c를 리턴
  - 만일 c<0이면, this 객체는 주어진 object보다 작음
  - 만일 c=0이면, this 객체는 주어진 object와 같음
  - 만일 c>0이면, this 객체는 주어진 object보다 큼

- 디스크 주소를 위한 인터페이스

```
interface Address {
   public Object get(Comparable key);
}
```

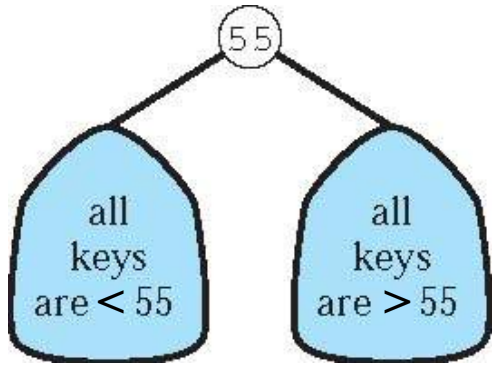
키와 Comparable 타입(2)

- 탐색 트리가 만족해야 하는 조건
  - 탐색 트리 구조 내에 있는 키는 유일함
  - 각각의 키는 그것이 표현하는 데이터의 주소를 가지고 있음
  - 키의 타입은 java.lang.Comparable 인터페이스를 구현함
    - 크기 비교가 가능
  - 주소의 타입은 Address 인터페이스를 구현함
    - 키에 저장되어 있는 주소에 위치하고 있는 Object 리턴 가능
  - 저장된 키를 참조할 때, 실제로는 키-주소 쌍을 참조하는 것

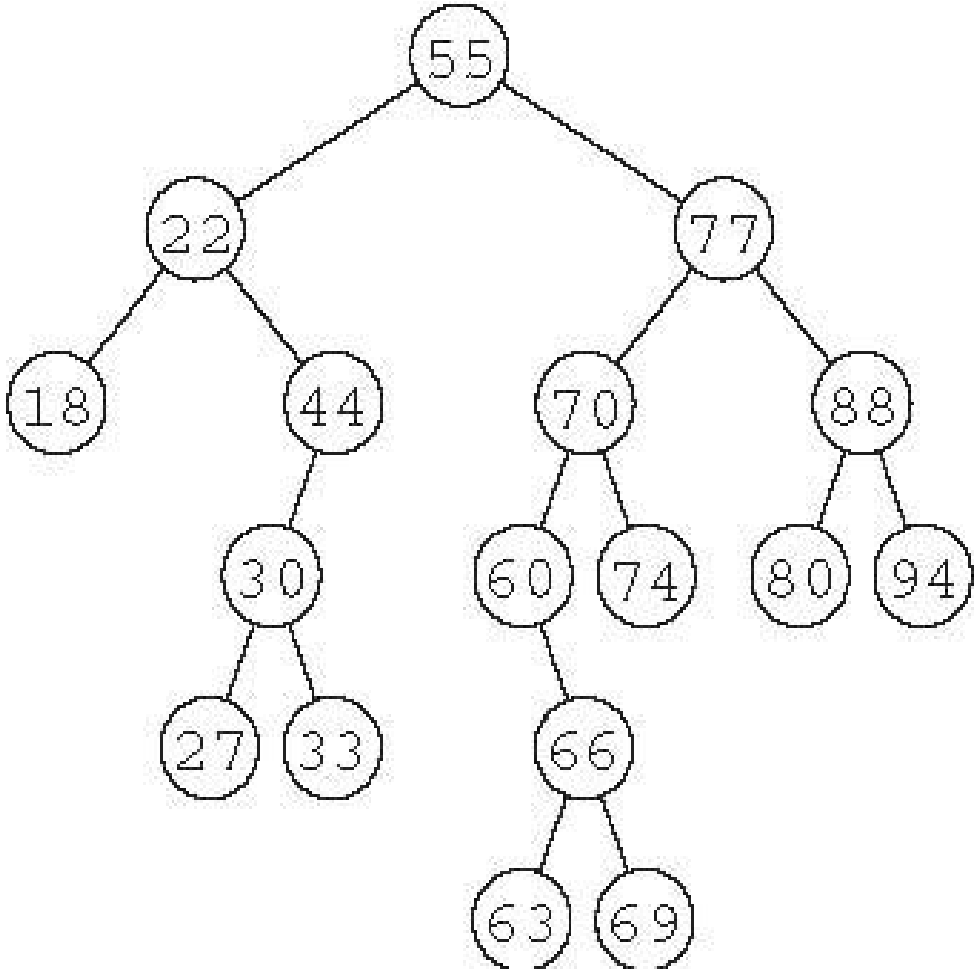
13.2 이진 탐색 트리 (1)

- 정의
  - 이진 탐색 트리(BST: binary search tree)는 각각의 노드가 BST 특성을 만족하는 키-주소 쌍을 가지고 있는 이진 트리

- BST 특성
  - 트리에 있는 각각의 키에 대해, 왼쪽 서브트리에 있는 모든 키는 이것보다 작고, 오른쪽 서브트리에 있는 모든 키는 이것보다 큼



이진 탐색 트리의 예



이진 탐색 트리 (2)

- 이진 탐색 트리의 순회
  - 이진 탐색 트리의 중위 순회는 키를 오름차순으로 방문

- BST 검색 알고리즘
  - 입력 : 이진 탐색 트리 T와 키.
  - 출력 : 키를 위한 데이터 주소 또는 키가 T에 없다면 null.

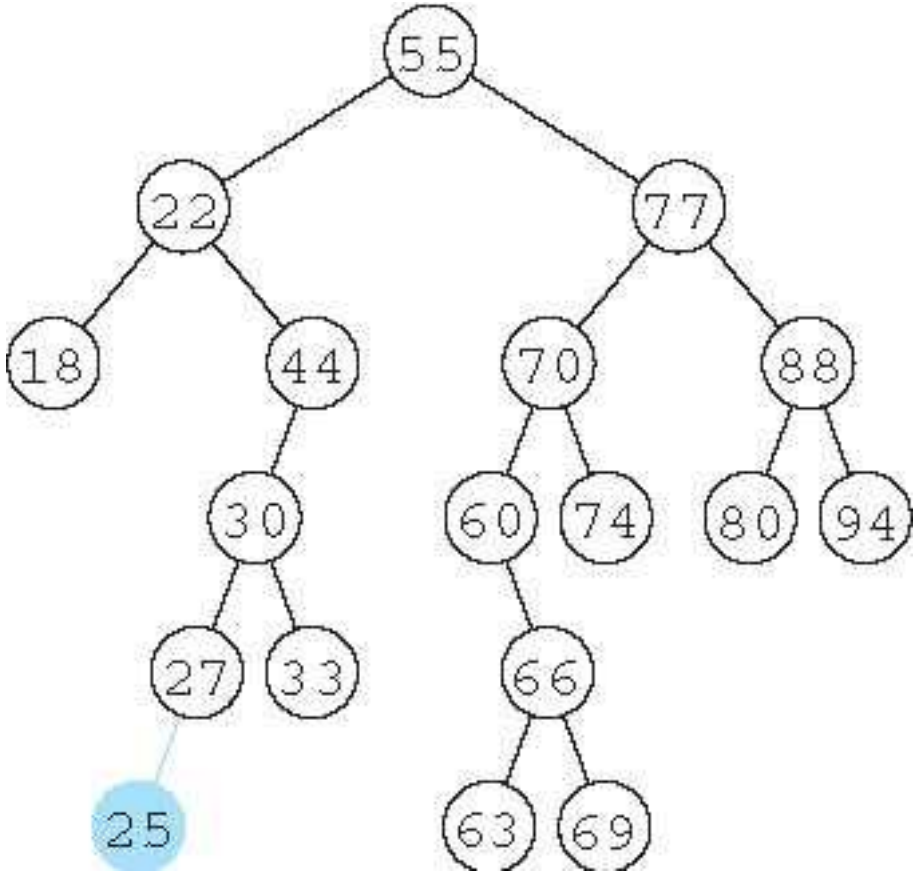
  1. 만일 T가 공백이면, null을 리턴.
  2. 만일 key < root.key 이면, 키를 찾기 위해 왼쪽 순환 탐색에 의해 반환되는 값을 리턴.
  3. 만일 key > root.key 이면, 키를 찾기 위한 오른쪽 순환 탐색에 의해 반환되는 값을 리턴.
  4. root.address를 리턴.

BST 삽입

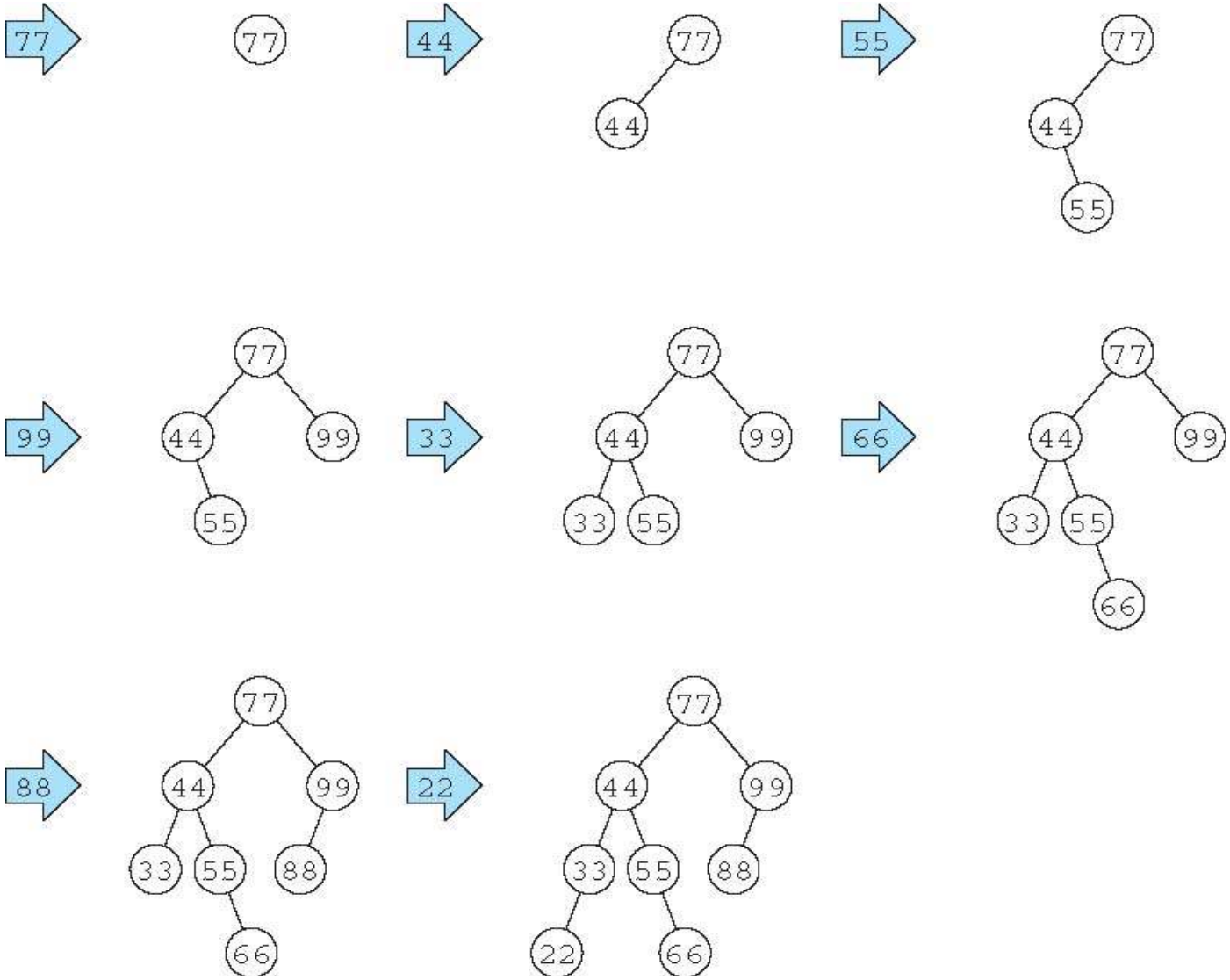
- BST 삽입 알고리즘
  - 입력 : 이진 탐색 트리 T와 키-주소 쌍.
  - 출력 : 키가 T에 이미 있다면 false, 아니면 true.
  - 후조건 : 키-주소 쌍은 T에 있음.

1. 만일 T가 공백이면 키-주소 쌍을 포함하는 단독 트리를 만들고, true를 리턴.
2. 만일 key < root.key 이면, 순환적으로 왼쪽 서브트리에서 키-주소 쌍의 삽입에 의해 반환되는 값을 리턴.
3. 만일 key < root.key 이면, 순환적으로 오른쪽 서브트리에서 키-주소 쌍의 삽입에 의해 반환되는 값을 리턴.
4. false를 리턴.

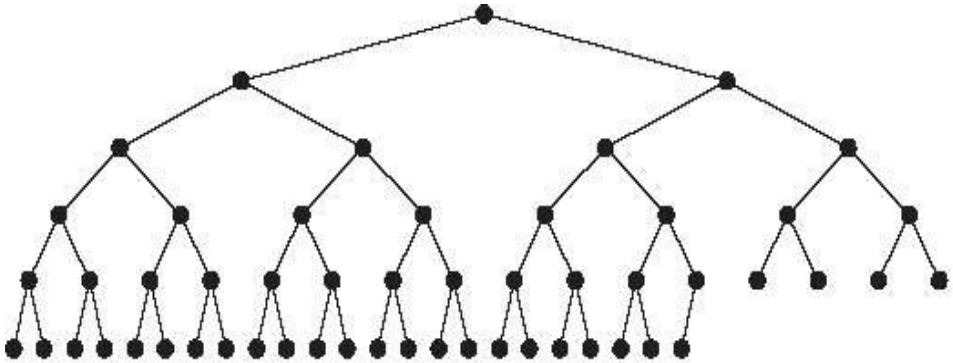
BST 삽입의 예



BST 구축 예


77
77
44
77
44
55
77
44
55
99
77
44
99
55
33
77
44
99
33
55
66
77
44
99
33
55
66
88
77
44
99
33
55
88
66
22
77
44
99
33
55
88
22
66

BST의 최선과 최악의 경우



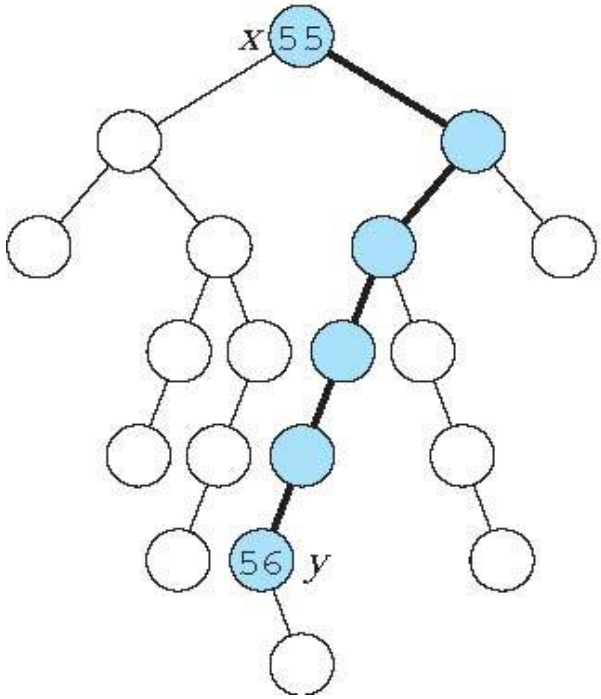
BST 최소값

- BST 최소값 알고리즘
  - 입력 : 공백이 아닌 이진 탐색 트리 T.
  - 출력 : T에 있는 최소 노드.

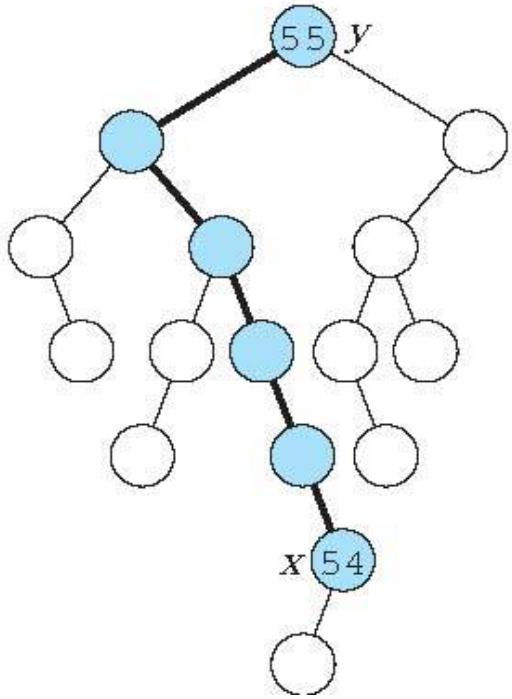
  1. x를 T의 루트라 하자.
  2. x.left가 공백이 아닌 동안 x=x.left로 설정.
  3. x를 리턴.

노드의 중위 후속자(1)

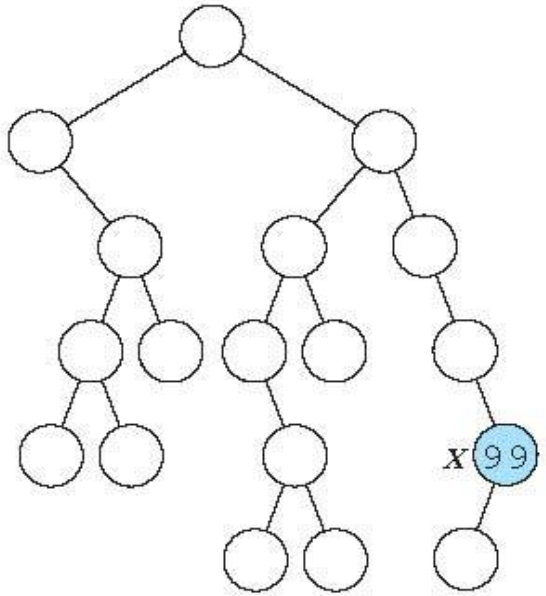
사례 **1:** x의 후속자는 오른쪽 서브트리의 최소 원소임



사례 **2:** x의 후속자는 값이 크면서 가장 가까운 조상임



사례 **3:** 트리에서 가장 오른쪽 원소는 후속자를 가지지 않음



노드의 중위 후속자(2)

- 세 가지 가능성
  1. 만일 x가 오른쪽 서브트리를 가지고 있다면, y는 이 서브트리에서 가장 왼쪽에 있는 노드이다.
  2. 그렇지 않고, 만일 x가 오른쪽 조상을 가지고 있다면, y는 가장 가까운 오른쪽 조상이다.
  3. 그렇지 않다면, x는 트리에서 가장 오른쪽에 있는 원소이므로 중위 후속자를 가지지 않는다.

노드의 중위 후속자(3)

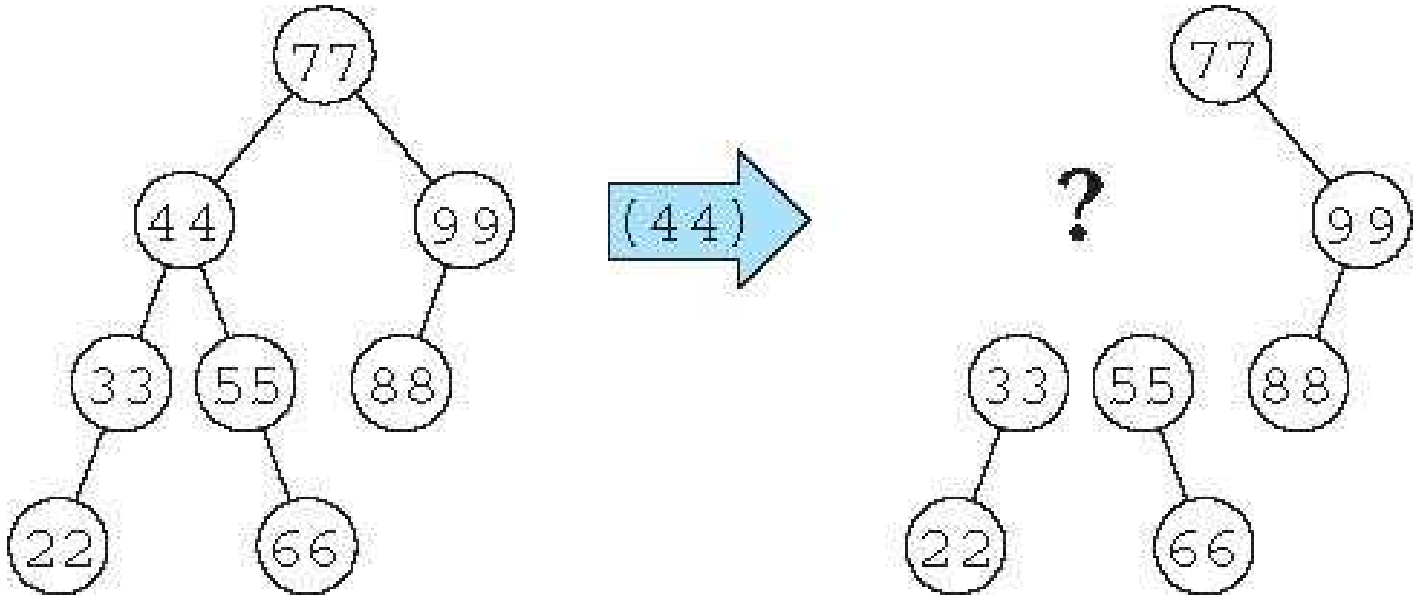
- BST 중위 후속자 알고리즘
  - 입력 : 공백이 아닌 이진 탐색 트리 T와 노드 x.
  - 출력 : x의 중위 후속자 또는 x가 T의 최대 원소일 경우에는 nil.

1. 만일 x의 오른쪽 서브트리가 공백이 아니면, 그것의 최소값을 리턴(알고리즘 13.3).
2. x가 오른쪽 자식인 동안 x=x.parent로 설정.
3. x.parent(nil이 될 수 있음)를 리턴.

부모로 이동하는 방법:
스택을 이용, 또는 노드 자체에 부모링크 구현

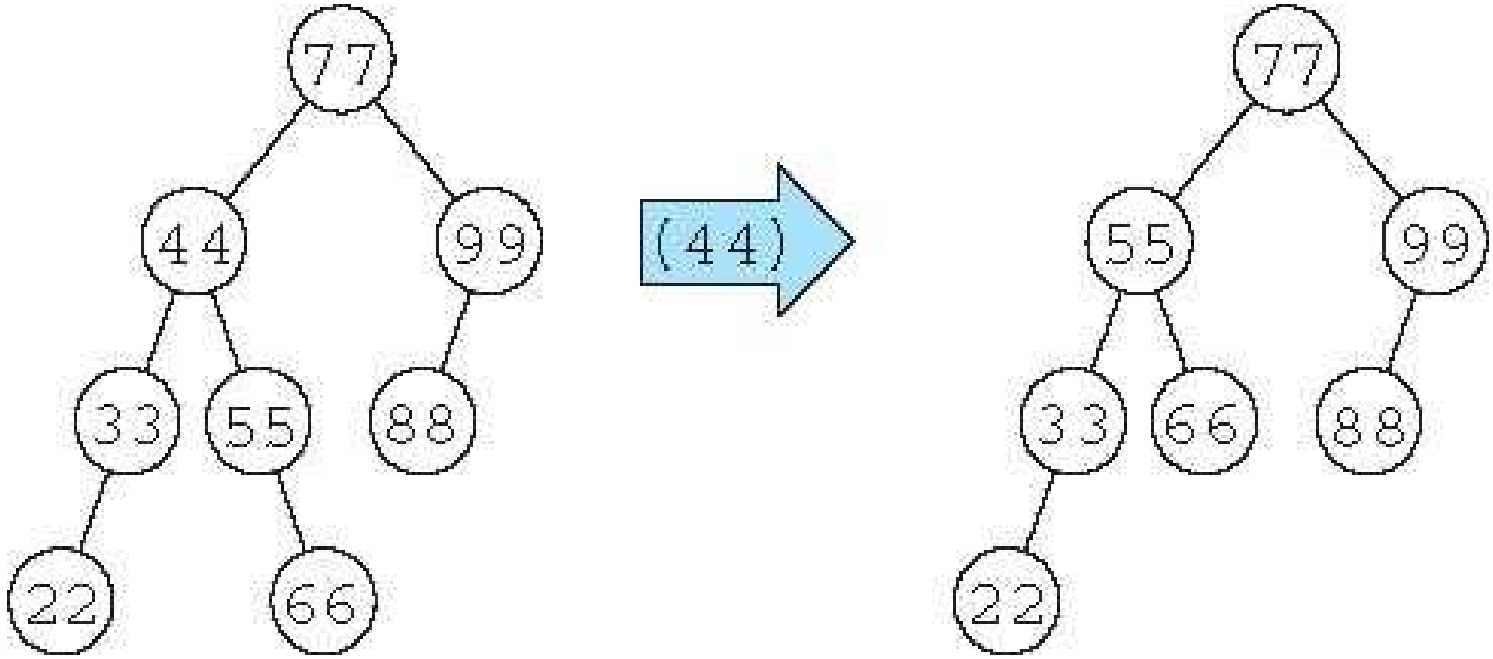
BST 삭제 (1)

- 노드 **44**를 삭제하기 위한 부적절한 시도



BST 삭제 (2)

- 노드 **44**를 삭제하는 올바른 방법



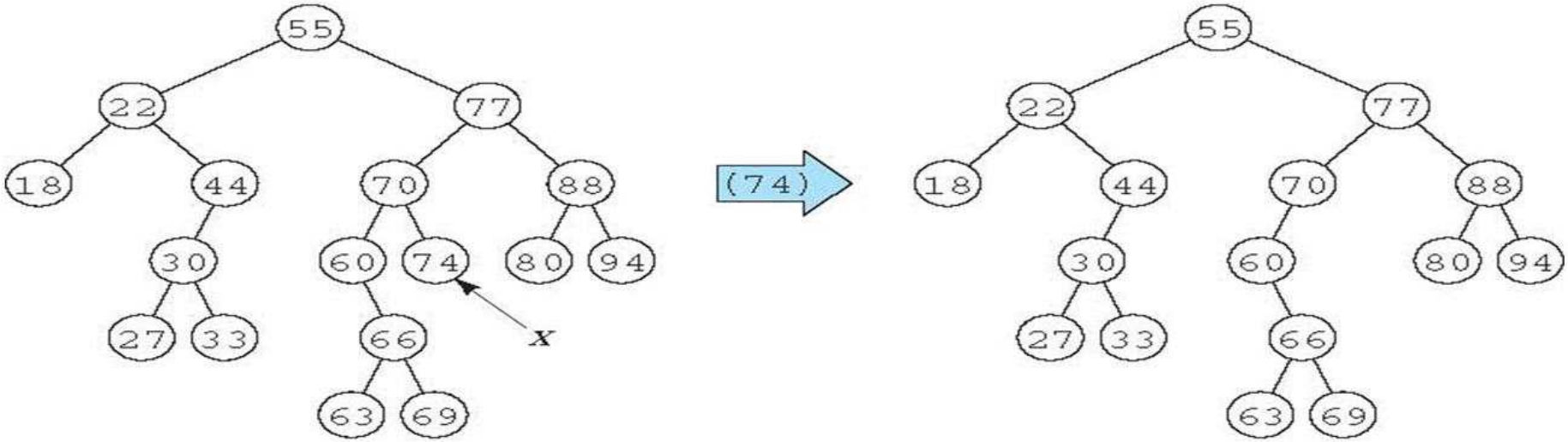
BST 삭제 알고리즘

- 입력 : 이진 탐색 트리 T와 키.
- 출력 : T에서 키를 발견할 수 없으면 false; 아니면 true.
- 후조건 : 키는 T에 없음.

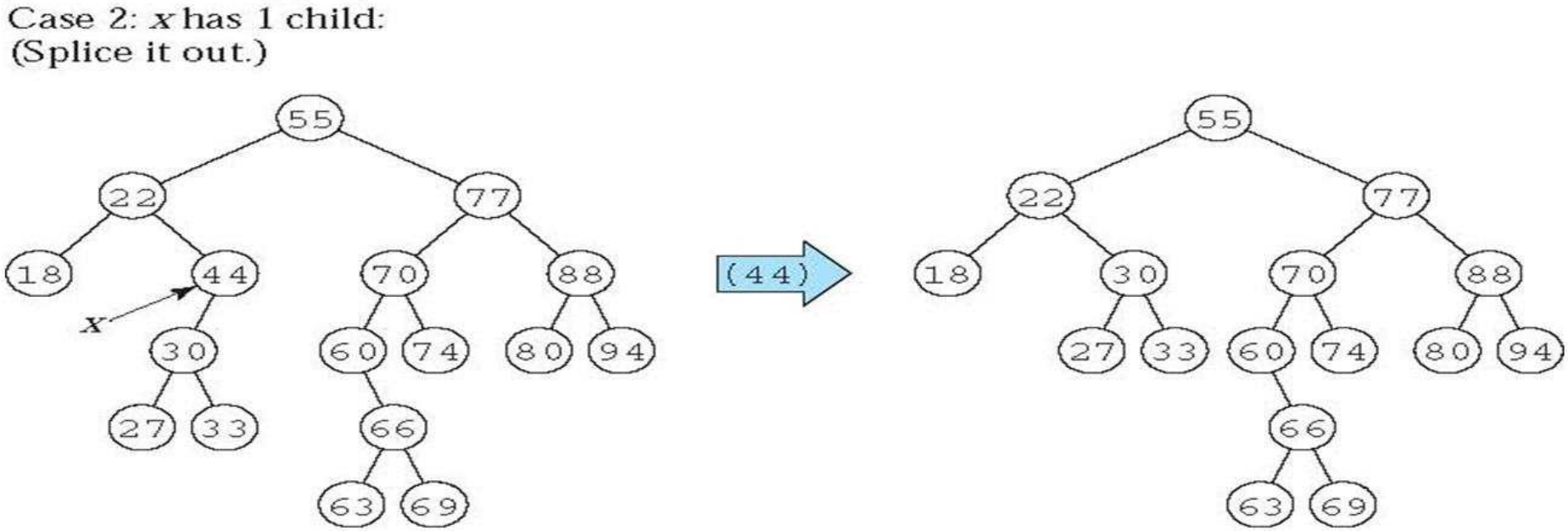
1. 만일 T가 공백이면, false를 리턴.
2. 만일 key < root.key 이면, 순환적으로 왼쪽 서브트리로부터 키의 삭제에 의해 반환되는 값을 리턴.
3. 만일 key > root.key 이면, 순환적으로 오른쪽 서브트리로부터 키의 삭제에 의해 반환되는 값을 리턴.
4. 만일 T가 단독 트리이면, 이것을 공백 트리로 만들고, true를 리턴.
5. 만일 왼쪽 서브트리가 공백이면, T의 오른쪽 서브트리의 루트, 왼쪽, 오른쪽 필드를 T 자체에 복사하고, true를 리턴.
6. 만일 오른쪽 서브트리가 공백이면, T의 왼쪽 서브트리의 루트, 왼쪽, 오른쪽 필드를 T 자체에 복사하고, true를 리턴.
7. 오른쪽 서브트리에 대해 deleteMinimum을 적용하여 반환되는 노드를 루트와 교체한다.

라인 4의 경우

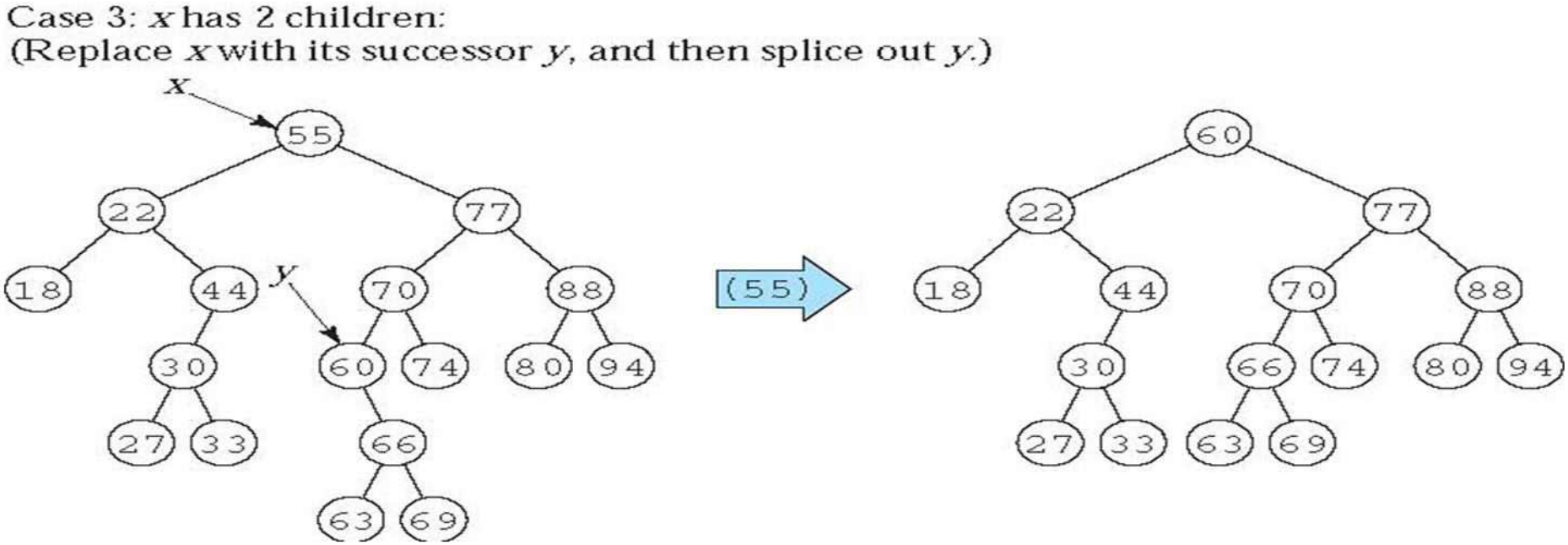
Case 1: x has 0 children:
(No side effects.)



라인 5와 6의 경우


Case 2: x has 1 child:
(Splice it out.)
55
22
77
18
44
70
88
x
30
60
74
80
94
27
33
66
63
69
(44)
55
22
77
18
30
70
88
27
33
60
74
80
94
66
63
69

라인 7의 경우


Case 3: x has 2 children:
(Replace x with its successor y, and then splice out y.)
x
y
55
22
77
18
44
70
88
30
60
74
80
94
27
33
66
63
69
(55)
60
22
77
18
44
70
88
30
66
74
80
94
27
33
63
69

BST 최소값 삭제

- BST 최소값 삭제 알고리즘
  - 입력 : 공백이 아닌 이진 탐색 트리 T.
  - 출력 : T에 있는 최소 노드 y.
  - 후조건 : y는 T에서 제거됨.

1. 만일 왼쪽 서브트리가 공백이면, 이것을 오른쪽과 교체한 다음 현재 노드를 리턴.
2. 만일 왼쪽 서브트리가 리프이면, 이것을 왼쪽과 교체한 다음 현재 노드를 리턴.
3. 최소값 삭제를 왼쪽에 적용하여 반환되는 노드를 리턴.

13.4 BST 성능

- BST의 세 가지 연산, 탐색, 삽입, 삭제 연산에서 비교 횟수는 트리의 높이에 비래하게 된다
- 이진 탐색 트리의 삽입과 탐색
  - $B(n) = \Theta(1)$
  - $A(n) = \Theta(\lg n)$
  - $W(n) = \Theta(n)$
- BST 탐색과 삽입 알고리즘에 대한 평균 시간 복잡도

$$A(n) = \Theta(1.39 \lg n) = \Theta(\lg n)$$