e-learning

2020 학년도 1 학기

컴퓨터 정보과

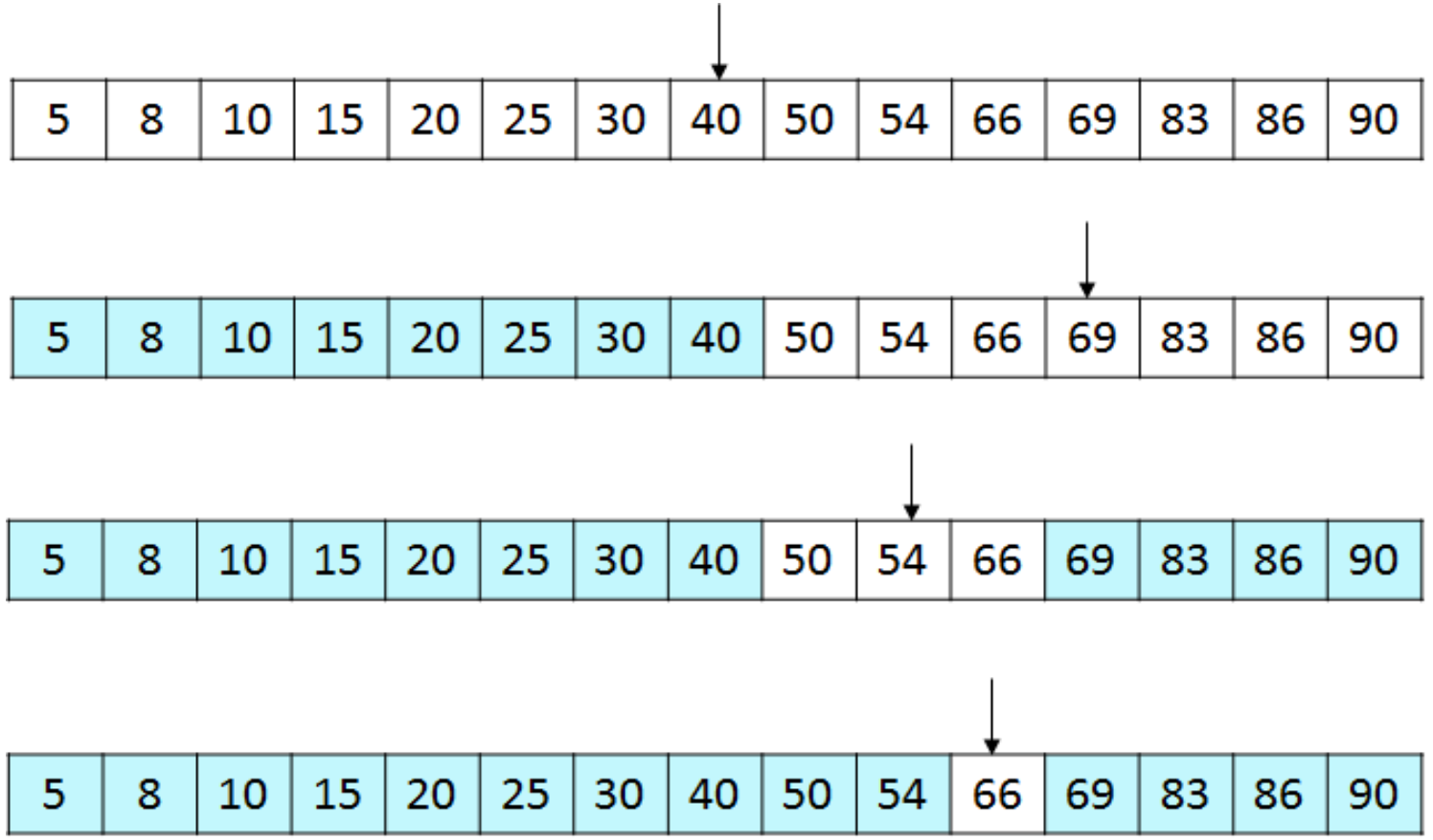
자료구조(Data Structures)

담당교수 : 김주현

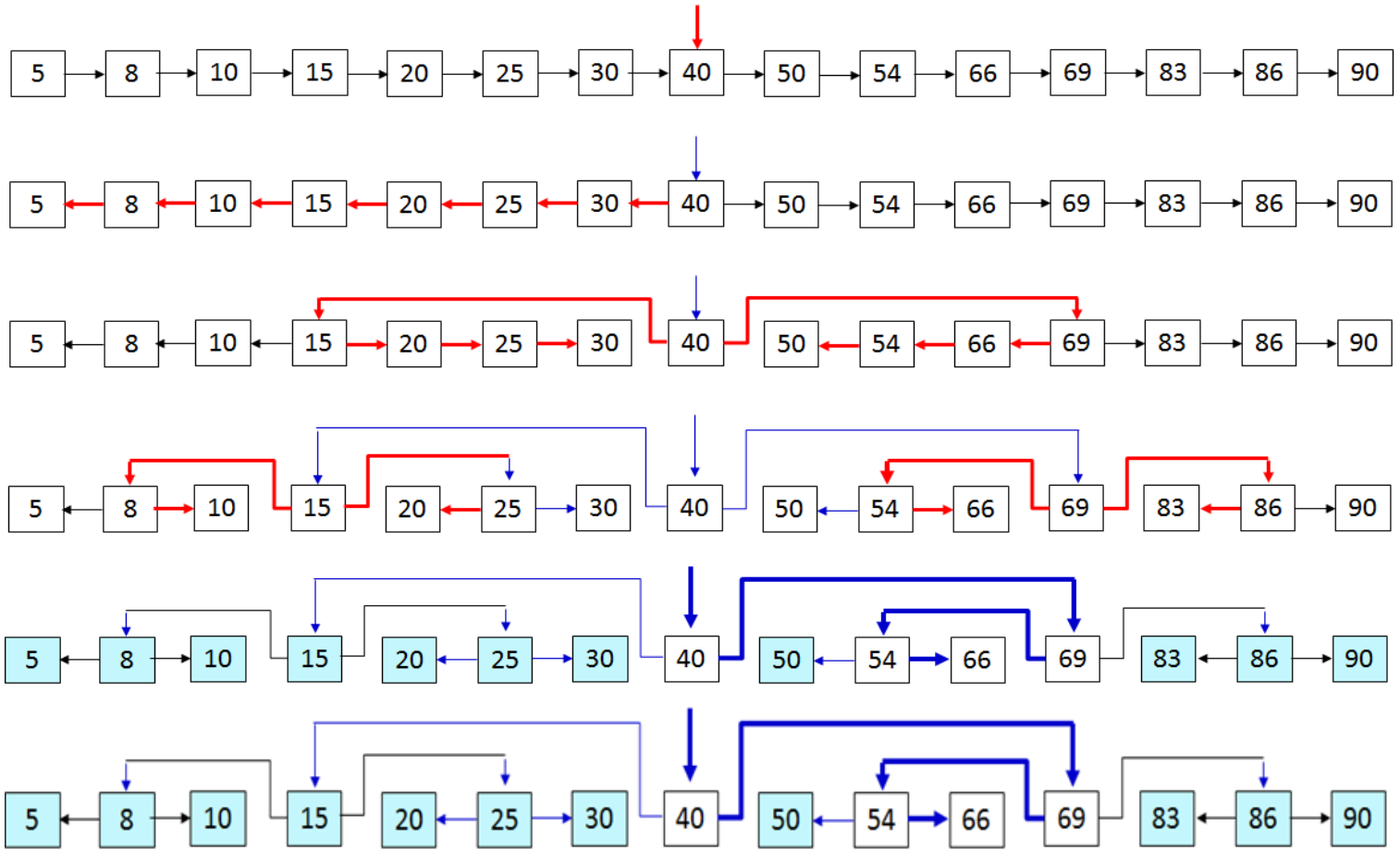
제 6 주차 / 제 1 차시

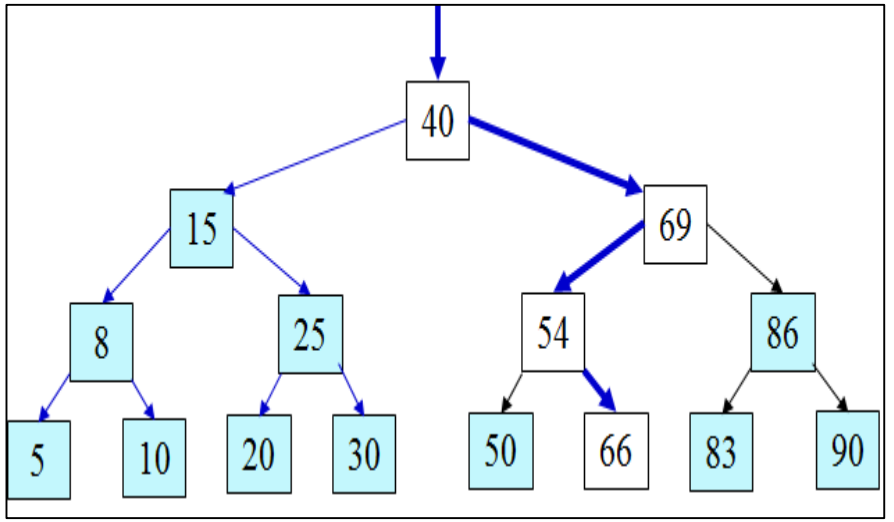
DIT 동의과학대학교
DONG-EUI INSTITUTE OF TECHNOLOGY

이진 탐색 과정

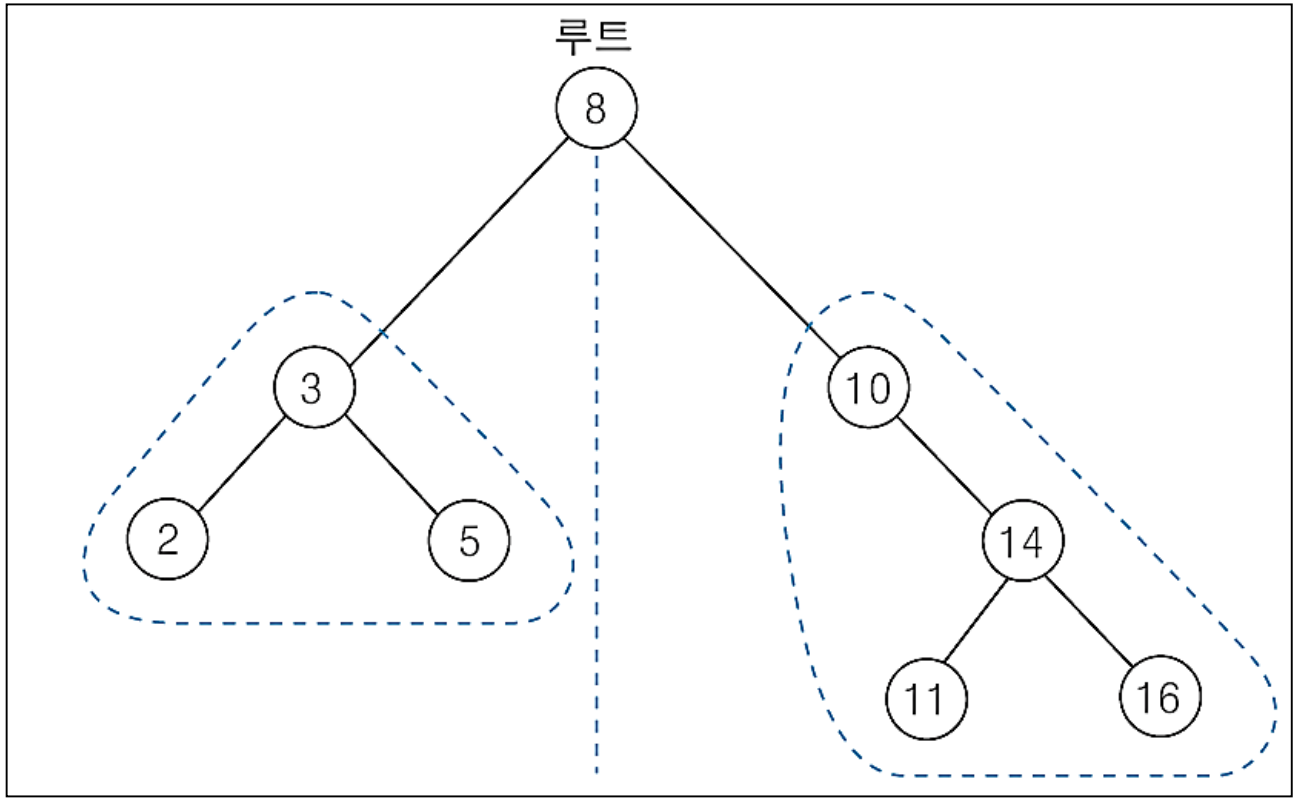

5 8 10 15 20 25 30 40 50 54 66 69 83 86 90
5 8 10 15 20 25 30 40 50 54 66 69 83 86 90
5 8 10 15 20 25 30 40 50 54 66 69 83 86 90
5 8 10 15 20 25 30 40 50 54 66 69 83 86 90

선형자료를 이진트리로 변환하는 과정





- # 이진 탐색 트리(binary search tree)
  - ✓ 이진 트리에 탐색을 위한 조건을 추가하여 정의한 자료구조
  - ✓ 이진 탐색 트리의 정의
    - (1) 모든 원소는 서로 다른 **유일한 키**를 갖는다.
    - (2) **왼쪽** 서브트리에 있는 원소의 키들은 그 **루트의 키보다 작다**.
    - (3) **오른쪽** 서브트리에 있는 원소의 키들은 그 **루트의 키보다 크다**.
    - (4) 왼쪽 서브트리와 오른쪽 서브트리도 이진 탐색 트리이다



이진 탐색 트리에서의 탐색 연산

루트에서 시작한다.

탐색할 키값 x를 루트 노드의 키값과 비교한다.

(키 값 x = 루트노드의 키 값)인 경우 :

☞ 원하는 원소를 찾았으므로 탐색연산 성공

(키 값 x < 루트노드의 키 값)인 경우 :

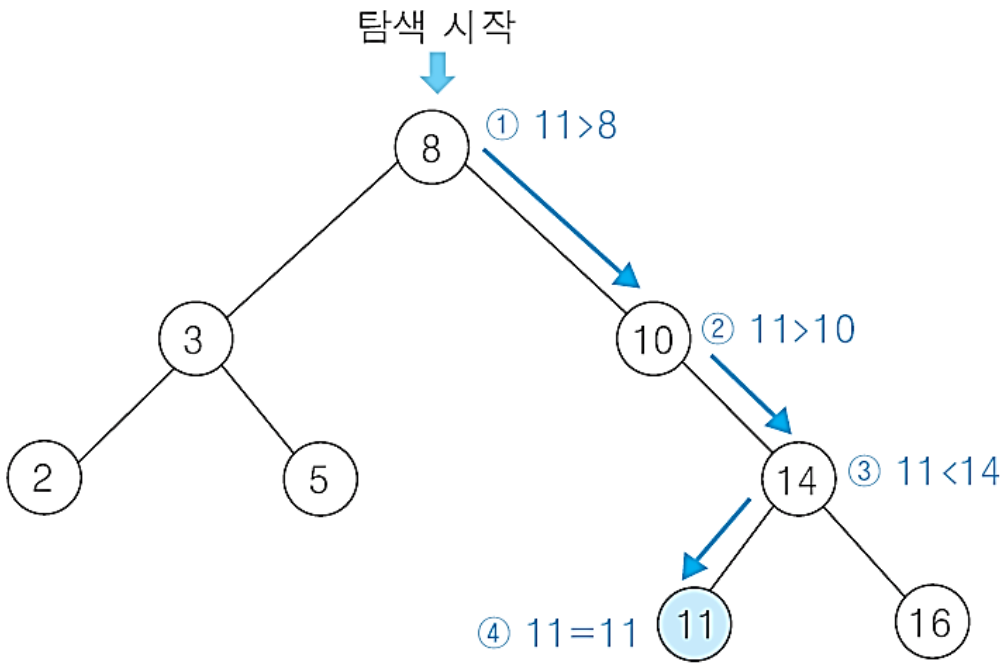
☞ 루트노드의 왼쪽 서브트리에 대해서 탐색연산 수행

(키 값 x > 루트노드의 키 값)인 경우 :

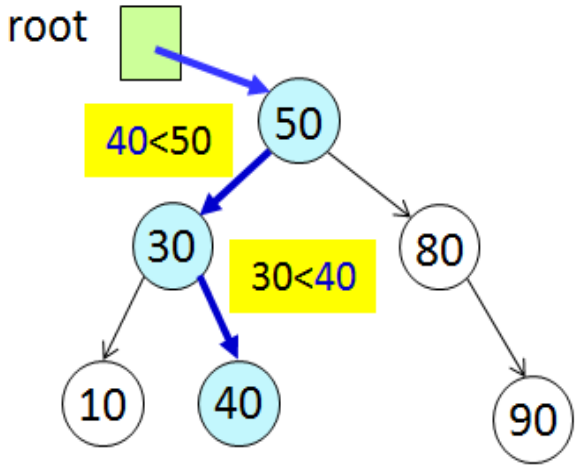
☞ 루트노드의 오른쪽 서브트리에 대해서 탐색연산 수행

서브트리에 대해서 순환적으로 탐색 연산을 반복한다.

원소 11 탐색하기



원소 40 탐색하기



이진탐색트리의 삽입 연산

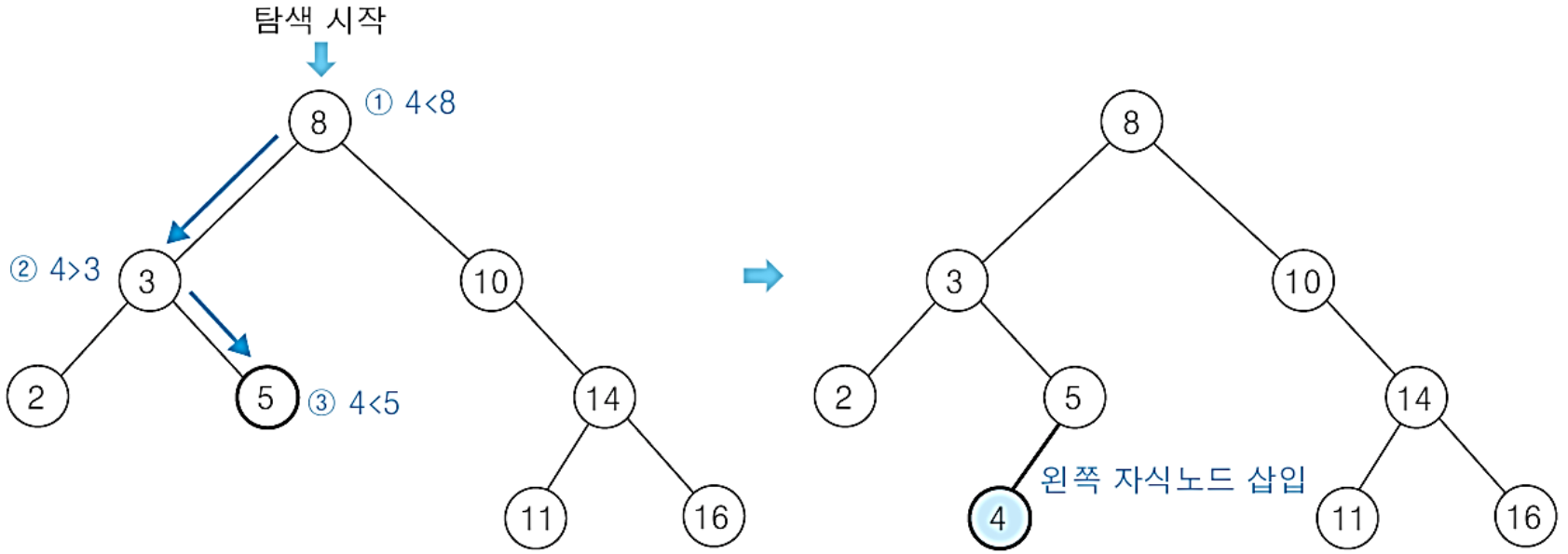
1) 먼저 탐색 연산을 수행

삽입할 원소와 같은 원소가 트리에 있으면 삽입할 수 없으므로, 같은 원소가 트리에 있는지 탐색하여 확인한다.

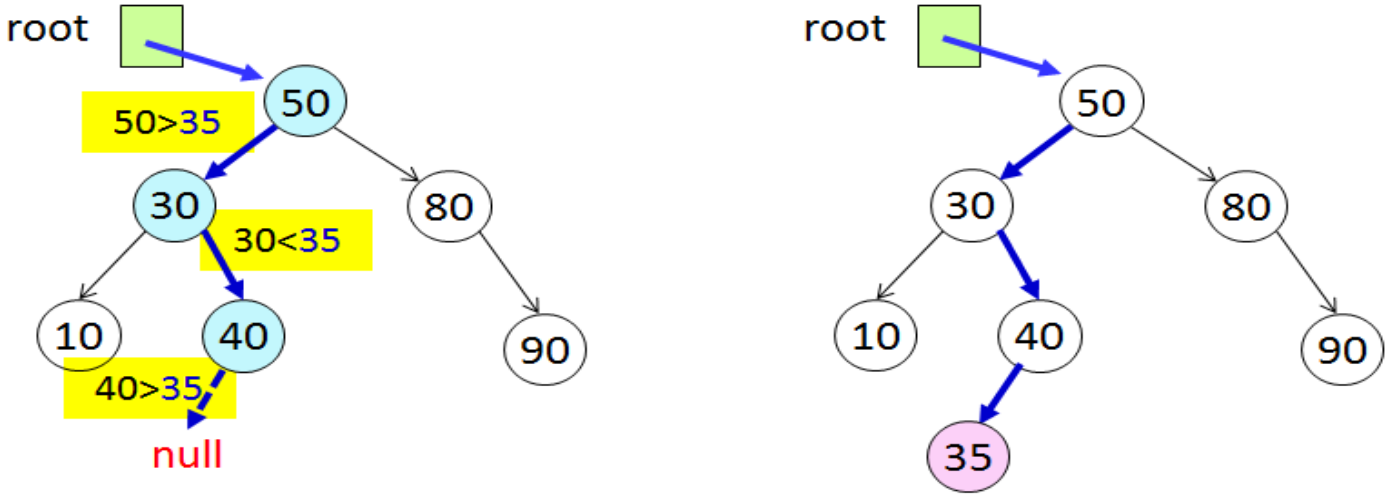
탐색에서 탐색 실패가 결정되는 위치가 삽입 원소의 자리가 된다.

2) 탐색 실패한 위치에 원소를 삽입한다.

원소 4 삽입하기



원소 35 삽입하기



이진탐색트리의 삭제 연산

1) 먼저 탐색 연산을 수행

삭제할 노드의 위치를 알아야하므로 트리를 탐색한다.

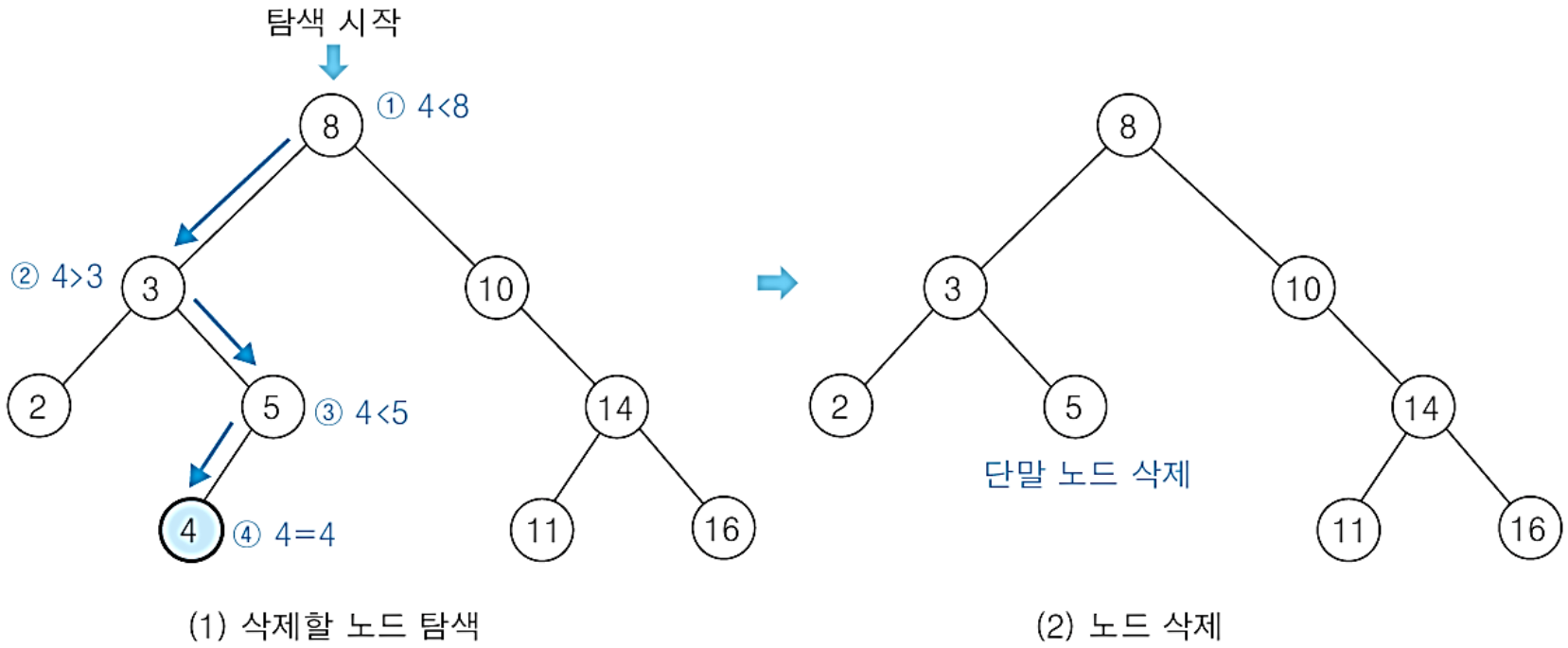
2) 탐색하여 찾은 노드를 삭제한다.

- 삭제 노드의 경우
  - ✓ 삭제할 노드가 단말노드인 경우 : 차수가 0인 경우
  - ✓ 삭제할 노드가 하나의 자식노드를 가진 경우 : 차수가 1인 경우
  - ✓ 삭제할 노드가 두개의 자식노드를 가진 경우 : 차수가 2인 경우

- 노드의 삭제 후에도 이진 탐색 트리를 유지해야 하므로 삭제 노드의 경우에 대한 후속 처리(이진 탐색 트리의 재구성 작업)가 필요하다.

삭제할 노드가 단말노드인 경우

원소 4 삭제하기

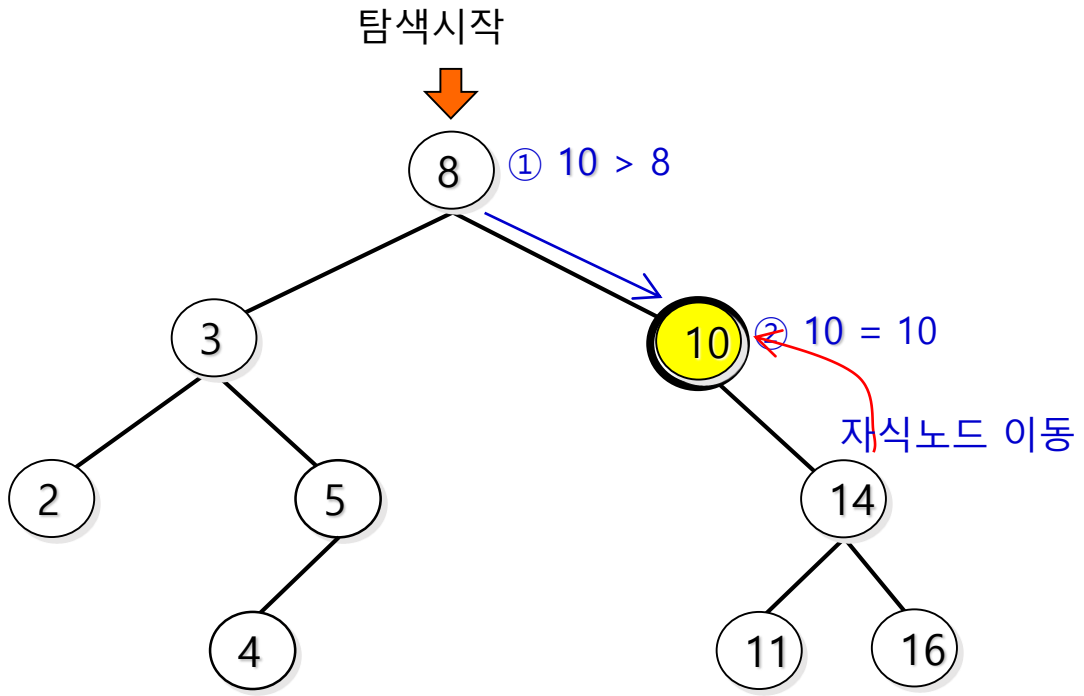


(1) 삭제할 노드 탐색

(2) 노드 삭제

삭제할 노드가 하나의 자식노드를 가진 경우

- 자식 노드가 하나인 노드, 즉 차수가 1인 노드의 삭제 연산
  - 노드를 삭제하면, 자식 노드는 트리에서 연결이 끊어져서 고아가 된다.
  - 후속 처리 : 이진 탐색 트리의 재구성
    - ✓ 삭제한 부모노드의 자리를 자식노드에게 물려준다.
  - 예) 노드 10을 삭제하는 경우

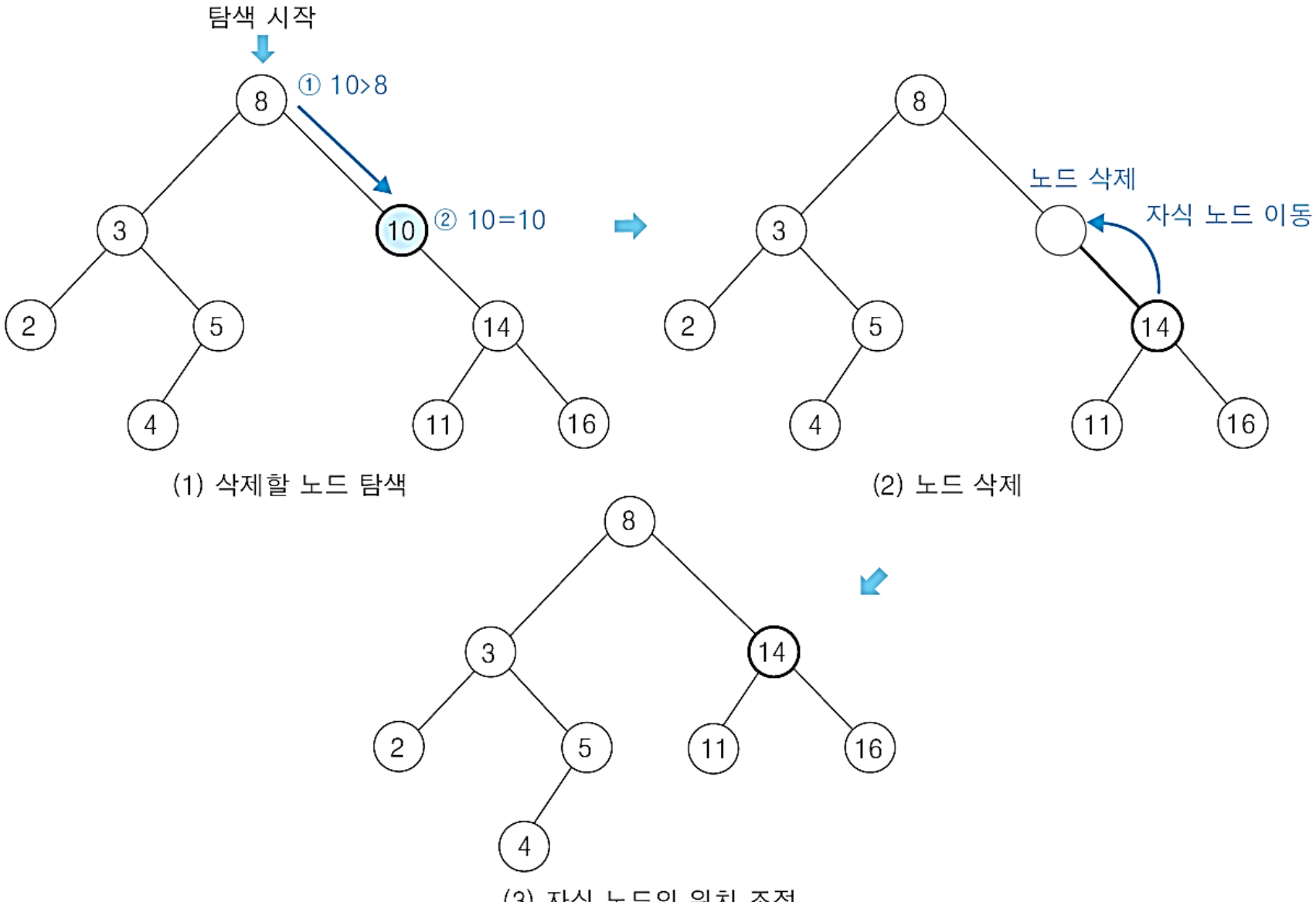


1단계: 삭제할 노드 탐색

2단계: 탐색한 노드 삭제

3단계: 후속처리

삭제할 노드가 하나의 자식노드를 가진 경우 삭제 과정



(1) 삭제할 노드 탐색

(2) 노드 삭제

(3) 자식 노드의 위치 조정

삭제할 노드가 두 개의 자식노드를 가진 경우

- **자식 노드가 둘인 노드, 즉 차수가 2인 노드의 삭제 연산**
  - ➢ 노드를 삭제하면, 자식 노드들은 트리에서 연결이 끊어져 고아가 된다.
  - ➢ **후속 처리 : 이진 탐색 트리의 재구성**
    - ✓ 삭제한 노드의 자리를 자손 노드들 중에서 선택한 후계자에게 물려준다.

  - ➢ 후계자 선택 방법

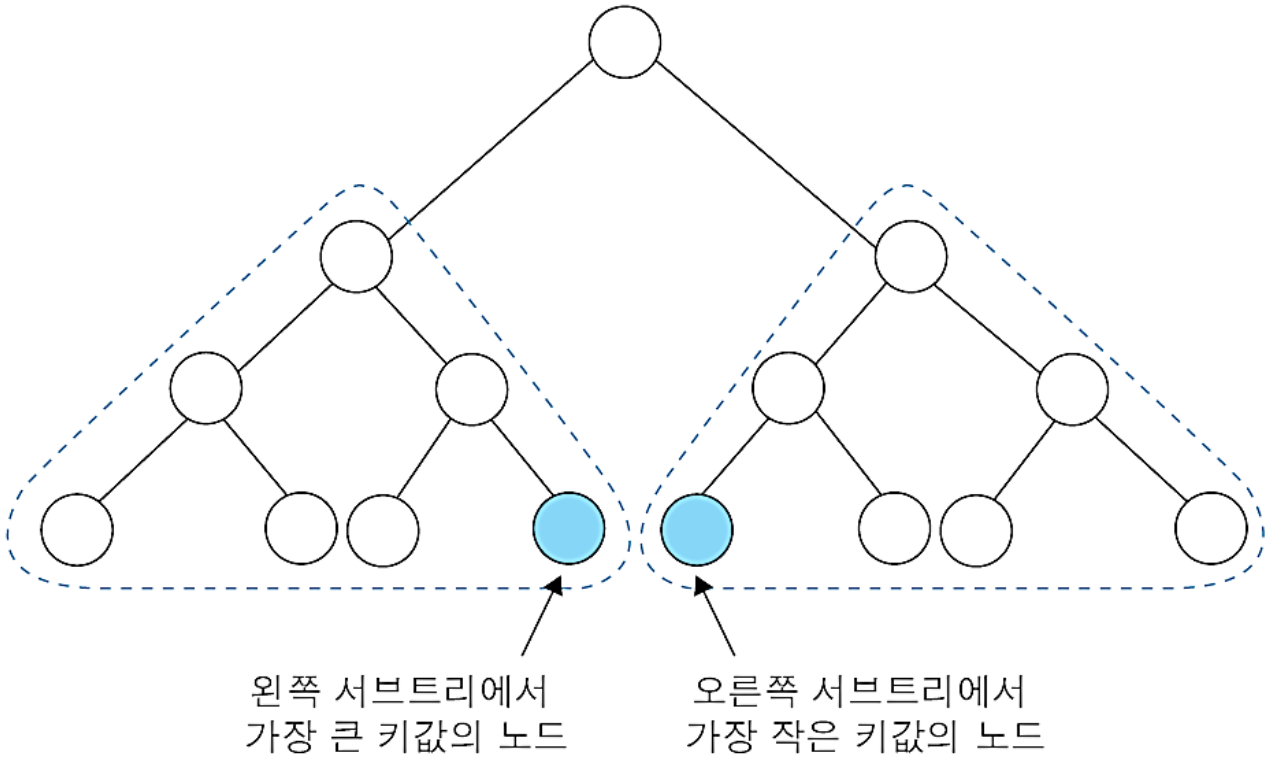
    방법1) 왼쪽 서브트리에서 가장 큰 자손노드 선택

    왼쪽 서브트리의 오른쪽 링크를 따라 계속 이동하여 오른쪽 링크 필드가 NULL인 노드 즉, 가장 오른쪽에 있는 노드가 후계자가 된다.

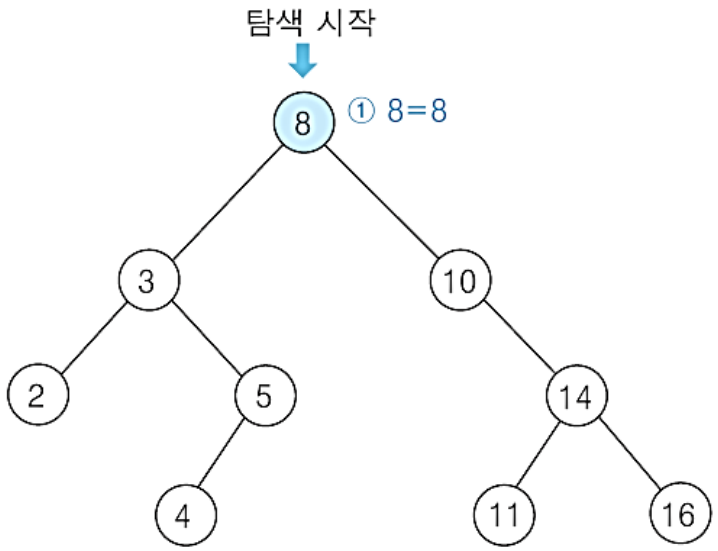
    방법2) 오른쪽 서브트리에서 가장 작은 자손노드 선택

    오른쪽 서브트리에서 왼쪽 링크를 따라 계속 이동하여 왼쪽 링크 필드가 NULL인 노드 즉, 가장 왼쪽에 있는 노드가 후계자가 된다.

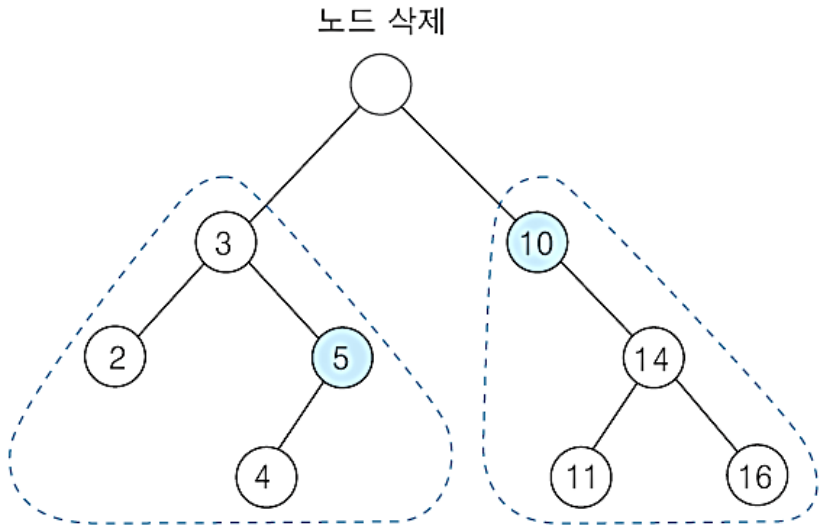
삭제한 노드의 자리를 물려받을 수 있는 후계자 노드



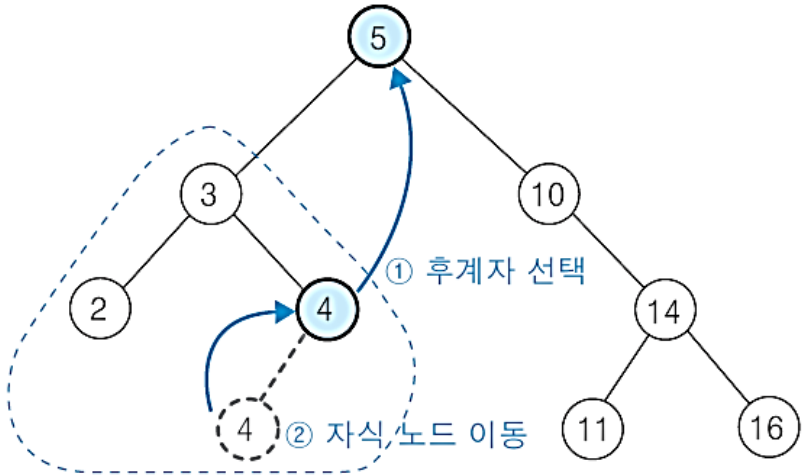
노드 8을 삭제하는 경우



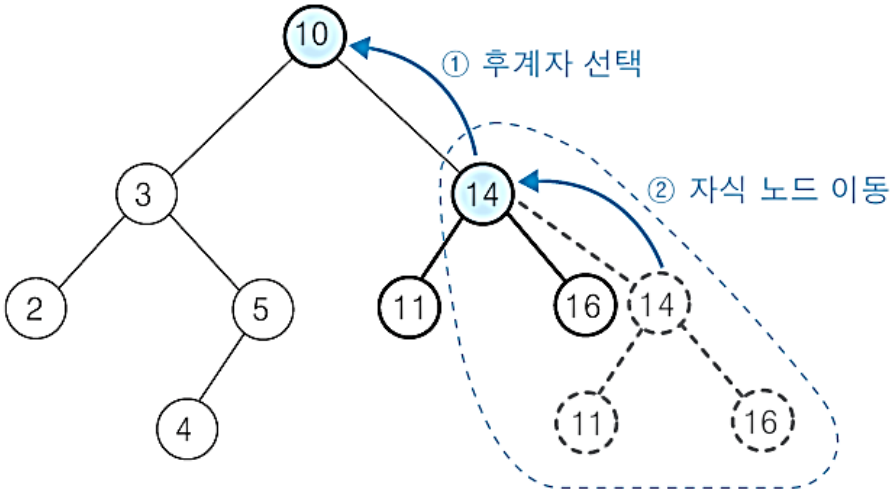
(1) 삭제할 노드 탐색



(2) 노드 삭제



(a) 트리 재구성_노드 5를 후계자로 선택한 경우



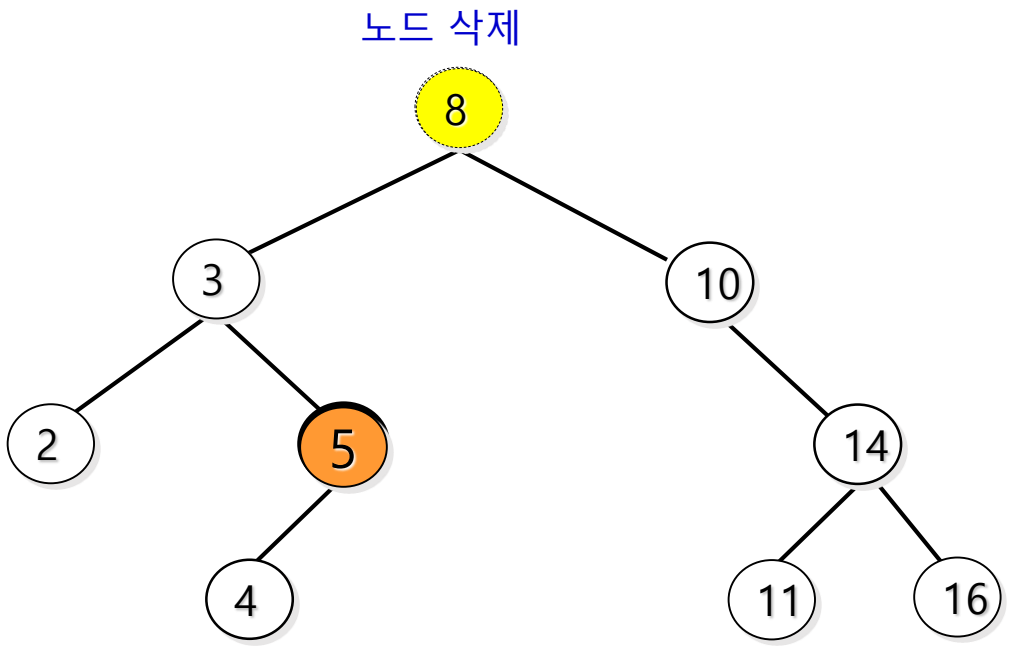
(b) 트리 재구성_노드 10을 후계자로 선택한 경우

- **노드 5를 후계자로 선택한 경우**

① 후계자 노드 5를 원래자리에서 삭제하여, 삭제노드 8의 자리를 물려준다.

② 후계자노드 5의 원래자리는 자식노드 4에게 물려주어 이진 탐색 트리를 재구성한다.

(☞ 자식노드가 하나인 노드 삭제연산의 후속처리 수행!)



1단계: 노드 **탐색**

2단계: 노드 **삭제**

3단계: 삭제한 노드의 자리를 **후계자에게 물려주기**

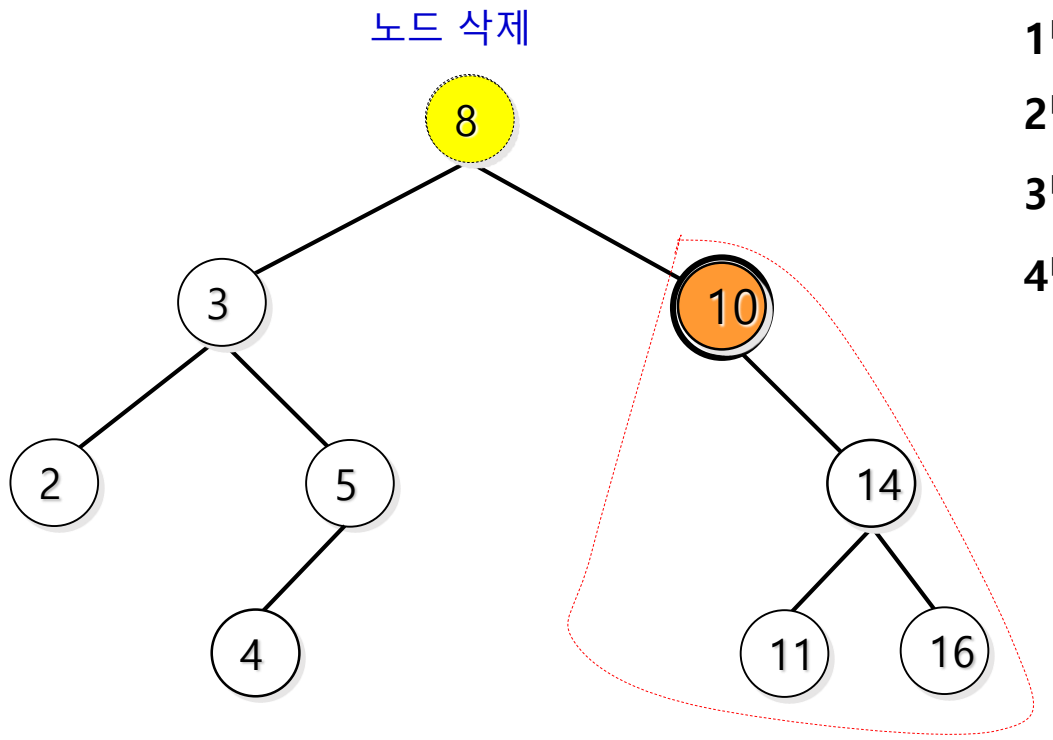
4단계: 후계자노드의 **원래자리를 자식노드에게 물려주기**

- **노드 10을 후계자로 선택한 경우**

  ① 후계자 노드 10을 원래자리에서 삭제하여, 삭제노드 8의 자리를 물려준다.

  ② 후계자노드 10의 원래자리는 자식노드 14에게 물려주어 이진 탐색 트리를 재구성한다.

  (☞ 자식노드가 하나인 노드 삭제연산의 후속처리 수행!)



1단계: 노드 **탐색**

2단계: 노드 **삭제**

3단계: 삭제한 노드의 자리를 **후계자에게 물려주기**

4단계: 후계자노드의 **원래자리를 자식노드에게 물려주기**

BST.java

```
class TreeNode{
        char data;
        TreeNode left;
        TreeNode right;
}

class BinarySearchTree{
        private TreeNode root = new TreeNode();

        public TreeNode insertKey(TreeNode root, char x){
                TreeNode p = root;
                TreeNode newNode = new TreeNode();
                newNode.data = x;
                newNode.left = null;
                newNode.right = null;
```

```
            if(p == null)
                return newNode;
            else if(newNode.data < p.data){
                p.left = insertKey(p.left, x);
                return p;
            }
            else if(newNode.data > p.data){
                p.right = insertKey(p.right, x);
                return p;
            }
            else return p;
        }

        public void insertBST(char x){
            root = insertKey(root, x);
        }
```

```
public TreeNode searchBST(char x){
        TreeNode p = root;
        while(p != null){
                if(x < p.data)  p = p.left;
                else if (x > p.data) p = p.right;
                else return p;
        }
        return p;
}

public void inorder(TreeNode root){
        if(root != null){
                inorder(root.left);
                System.out.printf(" %c", root.data);

                inorder(root.right);
        }
}
```

```
        public void printBST(){
                inorder(root);
                System.out.println();
        }
}
```

```
class BST{
        public static void main(String args[]){
                BinarySearchTree bsT = new BinarySearchTree();
                bsT.insertBST('G');
                bsT.insertBST('I');
                bsT.insertBST('H');
                bsT.insertBST('D');
                bsT.insertBST('B');
                bsT.insertBST('M');
                bsT.insertBST('N');
                bsT.insertBST('A');
                bsT.insertBST('J');
                bsT.insertBST('E');
                bsT.insertBST('Q');
```

```
                System.out.printf("₩nBinary Tree >>> ");
                bsT.printBST();

                System.out.printf("Is There ₩"A₩" ? >>> ");
                TreeNode p1 = bsT.searchBST('A');
                if(p1 != null)
                        System.out.printf("Searching Success! Searched key : %c ₩n", p1.data);
                else
                        System.out.printf("Searching fail!! There is no %c ₩n", p1.data);

                System.out.printf("Is There ₩"Z₩" ? >>> ");
                TreeNode p2 = bsT.searchBST('Z');
                if(p2 != null)
                        System.out.printf("Searching Success! Searched key : %c ₩n", p2.data);
                else
                        System.out.printf("Searching fail!! ₩n");
        }
}
```

Node.java

```
public class Node <Key extends Comparable<Key>, Value> {
        private Key   id;
        private Value name;
        private Node<Key, Value>  left, right;

        public Node(Key newId, Value newName) { // 노드 생성자
                id   = newId;
                name = newName;
                left = right = null;
        }
        // get과 set 메소드들
        public Key getKey()    { return id; }
        public Value getValue() { return name; }
        public Node<Key, Value> getLeft()    { return left; }
        public Node<Key, Value> getRight()  { return right;}
        public void setKey(Key     newId)       { id    = newId; }
        public void setValue(Value newName) { name  = newName; }
        public void setLeft(Node<Key, Value>   newLeft) { left  = newLeft; }
        public void setRight(Node<Key, Value>  newRight){ right = newRight;}
}
```

BST.java

```
public class BST<Key extends Comparable<Key>, Value>{
        public Node<Key, Value> root;
        public Node<Key, Value> getRoot() { return root; }
        public BST(Key newId, Value newName){
                // BST 생성자    // get, put, min, deleteMin, delete
                root = new Node<Key, Value>(newId, newName);
        }
        // 메소드들 선언
        public Value get(Key k) {return get(root, k);}
        public Value get(Node<Key, Value> n, Key k) {
                if (n == null)  return null;   // k를 발견 못함
                int t = n.getKey().compareTo(k);
                if (t > 0)          return get(n.getLeft(), k);
                                    // if (k < 노드 n의 id) 왼쪽 서브 트리 탐색
                else if (t < 0) return get(n.getRight(), k);
                                    // if (k > 노드 n의 id) 오른쪽서브 트리 탐색
                else                        return (Value) n.getValue(); // k를 가진 노드 발견
        }
```

```
public void put(Key k, Value v) {root = put(root, k, v);}
public Node<Key, Value> put(Node<Key, Value> n, Key k, Value v){
        if (n == null)  return new Node<Key, Value>(k, v);
        int t = n.getKey().compareTo(k);
        if (t > 0)  n.setLeft(put(n.getLeft(), k, v));
        // if (k < 노드 n의 id) 왼쪽 서브 트리에 삽입
        else if (t < 0) n.setRight(put(n.getRight(), k, v));
        // if (k > 노드 n의 id) 오른쪽 서브 트리에 삽입
        else n.setValue(v); // 노드 n의 name을v로 갱신
        return n;
}

public Key min() {
        if (root == null) return null;
        return (Key) min(root).getKey();}
private Node<Key, Value> min(Node<Key, Value> n) {
        if (n.getLeft() == null)  return n;
        return min(n.getLeft());
}
```

```
        public void deleteMin() {
                if (root == null) System.out.println("empty 트리");
                root = deleteMin(root);
        }
        public Node<Key, Value> deleteMin(Node<Key, Value> n) {
                if (n.getLeft() == null) return n.getRight();
                // if (n의 왼쪽 자식==null) n의 오른쪽 자식 리턴
                n.setLeft(deleteMin(n.getLeft()));
                // if (n의 왼쪽 자식≠null), n의 왼쪽 자식으로 재귀 호출
                return n;
        }

        public void deleteMax() {
                if (root == null)  System.out.println("empty 트리");
                        root = deleteMax(root);
    }
```

```
private Node<Key, Value> deleteMax(Node<Key, Value> n) {
        if (n.getRight() == null)
                return n.getLeft();
        n.setRight(deleteMax(n.getRight()));
        return n;
}

public void delete(Key k) {
        root = delete(root, k);
}
```

```
public void delete(Key k) {root = delete(root, k);}
public Node<Key, Value> delete(Node<Key, Value> n, Key k) {
        if (n==null) return null;
        int t = n.getKey().compareTo(k);
        if (t > 0)      n.setLeft(delete(n.getLeft(), k));
        // if (k < 노드 n의 id) 왼쪽 자식으로 이동
        else if (t < 0) n.setRight(delete(n.getRight(), k));
        // if (k > 노드 n의 id) 오른쪽 자식으로 이동
        else {  // 삭제할 노드 발견
                if (n.getRight() == null) return n.getLeft();  // case 0, 1
                if (n.getLeft()  == null) return n.getRight(); // case 1
                Node<Key, Value> target = n;   // case 2 Line10-13
                n = min(target.getRight());
                // 삭제할 노드 자리로 옮겨올 노드 찾아서 n이 가리키게 함
                n.setRight( deleteMin(target.getRight()));
                n.setLeft(target.getLeft());
        }
        return n;
}
```

```
        public void print(Node<Key, Value> root) {
                System.out.printf("₩ninorder:₩n");
                inorder(root);
        }
        public void inorder(Node<Key, Value> n){        // 중위 순회
                if (n != null) {
                        inorder(n.getLeft());   // n의 왼쪽 서브 트리를 순회하기 위해
                        System.out.print(n.getKey()+" ");  // 노드 n 방문
                        inorder(n.getRight());  // n의 오른쪽 서브 트리를 순회하기 위해
                }
        }
}
```

BSTMain.java

```
public class BSTMain {
        public static void main(String[] args) {
                BST<Integer, String> t = new BST<Integer, String>(500, "Apple");

                t.put(600, "Banana");
                t.put(200, "Melon");
                t.put(100, "Orange");
                t.put(400, "Tangerine");
                t.put(250, "Kiwi");
                t.put(150, "Grape");
                t.put(800, "Strawberry");
                t.put(700, "Cherry");
                t.put(50,  "Pear");
                t.put(350, "Lemon");
                t.put(10,  "Watermelon");
```

```
                t.print(t.root);System.out.println();
                System.out.println(t.get(200));
                t.delete(250);
                t.print(t.root);System.out.println();
                t.put(250, "Kiwi");
                t.print(t.root);System.out.println();
        }
}
```

Reference

- 자바로 배우는 쉬운 자료구조, 이지영, 한빛아카데미
- 자바와 함께하는 자료구조의 이해, 양성봉, 생능출판

언제 어디서나 즐공, 열공, 진공하세요.

감사합니다

DIT 동의과학대학교
DONG-EUI INSTITUTE OF TECHNOLOGY

DIT 동의과학대학교
DONG-EUI INSTITUTE OF TECHNOLOGY