



2020 학년도 1 학기

컴퓨터 정보과

자료구조(Data Structures)

담당교수 : 김주현

제 6 주차 / 제 3 차시

```
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 W\"AW\" ? >>> ");
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 W\"ZW\" ? >>> ");
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");
}
}
```

```
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; }  
}
```




```
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 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
        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("Inorder: %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  
    }  
}  
}
```

```
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 즐^공 열^공, 진공하세요.

감사합니다

