4주차 실습과제

20135151 이갑성

**public** **class** TreeNode //노드를 찍어내는 빵틀

{

String data;

TreeNode left, right;

**public** TreeNode() //생성자

{

**this**.data = **null**;

**this**.left = **null**;

**this**.right = **null**;

}

}

**public** **class** BST {

TreeNode root; // 루트 노드

**public** BST() // 생성자

{

}

**public** **void** insert(String x) // 자료를 넣는 메소드

{

**if** (root == **null**) { // 루트가 null인 경우

TreeNode newNode = **new** TreeNode(); // 노드 생성

newNode.data = x;

newNode.left = **null**;

newNode.right = **null**;

root = newNode; // 생성된 노드가 곧 루트가 된다.

} **else** {

insertKey(root, x); // 루트가 null이 아닌 경우 insertKey메소드으로 넘겨주어

// insert해준다.

}

}

**public** **void** insertKey(TreeNode subroot, String x) {

**if** (subroot.data.compareTo(x) < 0){ // x가 subroot.data의 값보다 큰경우 ==>

// 오른쪽이동

**if** (subroot.right == **null**) { // subroot.right의 노드가 null인경우 ==> 바로

// null자리에 노드 생성

TreeNode newNode = **new** TreeNode();

newNode.data = x;

newNode.left = **null**;

newNode.right = **null**;

subroot.right = newNode; // 생성한 노드가 곧 subroot의 오른쪽자식이다.

} **else** { // null이 아닌경우 ==> 재귀를 이용해 insertKey()메소드를 돌린다.

insertKey(subroot.right, x);

}

} **else** **if** (subroot.data.compareTo(x) > 0) { // x가 subroot.data의 값보다 작은경우

// ==> 왼쪽이동

**if** (subroot.left == **null**) { // subroot.left의 노드가 null인경우 ==> 바로

// null자리에 노드 생성

TreeNode newNode = **new** TreeNode();

newNode.data = x;

newNode.left = **null**;

newNode.right = **null**;

subroot.left = newNode; // 생성한 노드가 곧 subroot의 왼쪽자식이다.

} **else** { // null이 아닌경우 ==> 재귀를 이용해 insertKey()메소드를 돌린다.

insertKey(subroot.left, x);

}

}

}

**public** **void** inorderPrint() // 중위순회 프린트하는 메소드

{

inorder(root); // 중위순회하는 메소드에 루트를 넘겨준다.

}

**public** **void** inorder(TreeNode Node1) // 중위순회 메소드

{

**if** (Node1 != **null**) {

inorder(Node1.left);

System.***out***.println(Node1.data);

inorder(Node1.right);

}

}

**public** **int** find(TreeNode root, String x) {

**int** isright = 0; // 존재 여부를 저장하는 변수, 초기값 0으로 초기화(false로 초기화)

TreeNode T = root; // 재귀를 하면서 루트의 위치가 변동이 됨으로 임의의 TreeNode T를 root와 복사를 해준다.

//T를 안해도 됨 매개변수 루트라서

**if** (T != **null**) { // root를 복사한 T가 null이 아닐때

**if** (T.data.compareTo(x) == 0) { // T의 data값이 x와 같은지

**return** 1;// 찾게 되면 1을 리턴한다.

} **else** **if** (T.data.compareTo(x) > 0) { //T의 data값이 x와 비교했을때 data가 더 크면

isright = find(T.left, x); //T의 왼쪽으로 이동한 Subroot값을 find메소드에 넣어준다.

} **else** **if** (T.data.compareTo(x) < 0) //T의 data값이 x와 비교했을때 x가 더 크면

isright = find(T.right, x); //T의 오른쪽으로 이동한 Subroot값을 find메소드에 넣어준다.

}

**return** isright;

}

**public** **void** findMax() {

TreeNode T = root;

**while** (T.right != **null**) {

T = T.right;

}

System.***out***.println("제일 큰 값: " + T.data);

}

**public** **void** findMin() {

TreeNode T = root;

**while** (T.left != **null**) {

T = T.left;

}

System.***out***.println("제일 작은 값: " + T.data);

}

}

**import** java.util.Scanner;

**public** **class** BST\_main {

**public** **static** **void** main(String args[]) {

Scanner sc = **new** Scanner(System.***in***);

**int** isright = 0;

String findNode = **null**;

BST a = **new** BST();

a.insert("A");

a.insert("C");

a.insert("X");

a.insert("B");

a.insert("F");

a.insert("G");

a.insert("S");

a.inorderPrint();

System.***out***.println("트리에서 찾고 싶은 값을 입력하세요(타입은 String): ");

findNode = sc.next();

isright = a.find(a.root, findNode);

**if**(isright == 0)

System.***out***.println(findNode + "는 트리에 존재하지않습니다.");

**else**

System.***out***.println(findNode + "는 트리에 존재합니다.");

a.findMax();

a.findMin();

}

}

