

Homework #03

제출일 : 2019년 5월 26일 일요일 (eCampus)

제출물 : 학번.java 파일 하나

Hard copy 제출 필요 없음

전체 구현 30점

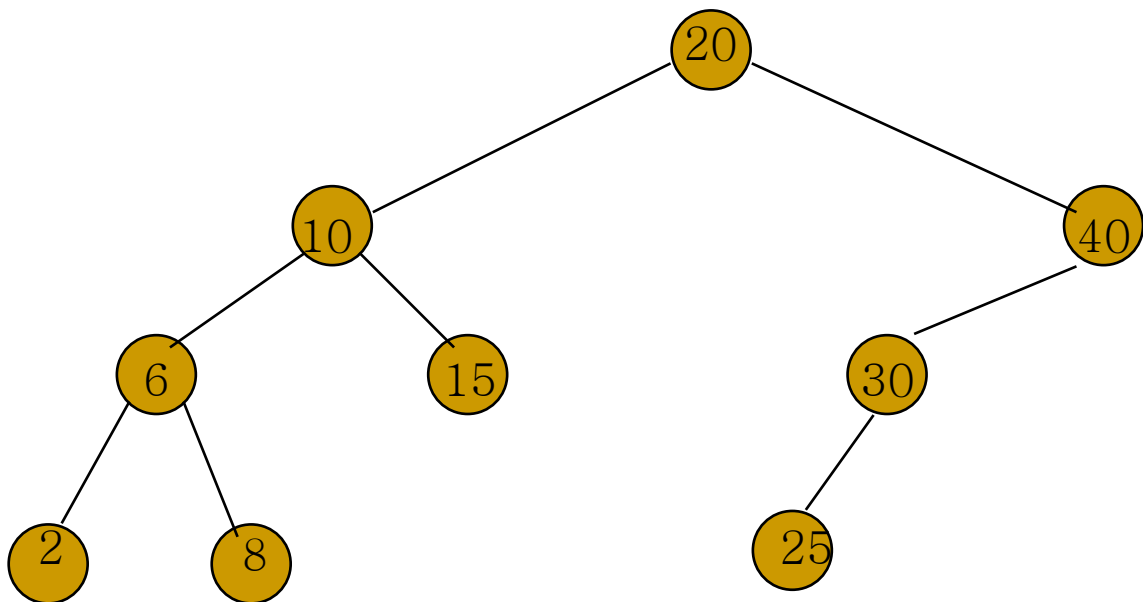
Insert only (Delete 구현 안함) 10점

hw03.zip 파일 : LabTest.java hw03.java lab.in lab.out labins.in labins.out

제출

hw03.java 를 학번.java 로 변경하여 이 파일 한 개만 제출할 것.

HW 03은 Binary Search Tree를 구현하는 문제이다.



hw03.java 에서 이 Binary Search Tree는 class BST <T extends KeyValue> class에 정의되어 있다. 각각의 노드는 class TreeNode <U extends KeyValue> 에 정의되어 있다. 또한 각 TreeNode <T>가 가지는 data는 Item라고 하는 클래스에 정의되어 있는데 이 Item

에는 key와 value를 저장할 수 있도록 했다. 위 그림에서는 key 값만을 보여준다.

수행 예는 다음과 같다.

```
sanghwan@sanghwan-VirtualBox: ~/dbox/classes191/ds/hw/hw193
File Edit View Search Terminal Help

sanghwan@sanghwan-VirtualBox:~/dbox/classes191/ds/hw/hw193$ java LabTest
BST > ins 10 1
Pre Order : [10(1)]
In Order : [10(1)]
Post Order : [10(1)]
Number of Nodes : 1
Height : 1

BST > ins 5 2
Pre Order : [10(1)][5(2)]
In Order : [5(2)][10(1)]
Post Order : [5(2)][10(1)]
Number of Nodes : 2
Height : 2

BST > ins 20 8
Pre Order : [10(1)][5(2)][20(8)]
In Order : [5(2)][10(1)][20(8)]
Post Order : [5(2)][20(8)][10(1)]
Number of Nodes : 3
Height : 2

BST > get 20
Item 20 8
Pre Order : [10(1)][5(2)][20(8)]
In Order : [5(2)][10(1)][20(8)]
Post Order : [5(2)][20(8)][10(1)]
Number of Nodes : 3
Height : 2

BST > del 5
Pre Order : [10(1)][20(8)]
In Order : [10(1)][20(8)]
Post Order : [20(8)][10(1)]
Number of Nodes : 2
Height : 2

BST > 
```

사용자가 사용하는 명령어의 syntax는 다음과 같다. LabTest 클래스의 main() 함수에 정의되어 있다. 이미 사용자 명령어는 구현이 되어 있다. 모든 명령에 대해 명령을 수행하고 나면 트리가 제대로 구성되었는지를 파악하기 위해 전체 트리를 pre order, in order, post order로 traverse하여 보여준다. 또한 트리의 모든 노드의 수를 보여준다. 마지막으로 트리의 height를 보여준다.

- quit

프로그램의 수행을 끝낸다.

- ins **key value**

이 명령은 **key-value**쌍을 하나의 BST에 하나의 노드로 추가하는 것이다. 만약 주어

진 key가 존재하는 경우는 새로운 key-value를 추가하지 않고, Key Exists라고 출력한다.

- `get key`

`key`에 정의된 값을 `key`로 가지는 노드를 찾아서 `key` 와 `value`의 값을 보여준다.

- `del key`

`key`에 지정된 값을 `key`로 가지는 노드를 찾아서 그 노드를 삭제한다.

이 과제에서는 BST라는 클래스를 구현하여 binary search tree를 구현하였다. BST의 멤버 함수 중 구현되어 있지 않은 부분을 구현하면 된다. 참고로 함수를 구현할 때 item으로부터 key나 value 값을 읽는 것은 `GetKey()` 함수나 `GetValue()` 함수를 사용한다.

- `boolean Insert(T item);`

`item.GetKey()` 값을 `key`로 하는 노드를 생성하여 알고리즘에 의해 정해진 위치에 삽입한다. 이 때 이 노드의 `data` field는 `item`으로 한다. 만약 같은 `item.GetKey()` 값을 가지는 노드가 존재하면 삽입하지 않고 그냥 `return false` 한다. 그렇지 않으면 `return true` 한다.

- `T Get(T item);`

`item.GetKey()` 값을 `key`로 가지는 노드를 찾아서 그 노드의 `data` field를 return한다. 만약 주어진 키를 가지는 노드가 존재하지 않으면 `return null`을 한다.

- `boolean Delete(T item);`

`item.GetKey()` 값을 가지는 노드를 찾아서 삭제한다. 만약 주어진 `key`값을 가지는 노드가 존재하지 않으면 `return false`하고, 존재하면 `return true` 한다.

단. degree가 2인 노드를 삭제하는 경우 왼쪽 subtree의 가장 큰 key를 가지는 원소를 찾아서 삭제할 노드로 치환하도록 한다.

- `void PreOrder(TreeNode<T> t);`

`t`로 시작하는 노드로부터 pre order로 traverse 하는 재귀 함수이다.

- `void InOrder(TreeNode<T> t);`

`t`로 시작하는 노드로부터 in order로 traverse 하는 재귀 함수이다.

- `void PostOrder(TreeNode<T> t);`

`t`로 시작하는 노드로부터 postorder order로 traverse 하는 재귀 함수이다.

- `int Count(TreeNode<T> t);`

`t`이 가리키는 노드를 root로 하는 subtree에 존재하는 노드의 수를 return 하는 재귀 함수이다.

- `int Height(TreeNode<T> t);`

`t`가 가리키는 노드를 root로 하는 subtree의 height를 return 하는 재귀함수이다. 어떤 subtree의 height는 root의 `leftChild`의 height와 `rightChild`의 height를 비교하여 큰

height에 1 더한 값이 뒀을 이용한다.

프로그램 테스트

lab.in,lab.out: insert와 delete 모든 내용 구현 검사 사용

labins.in, labins.out: Insert only 구현 검사 시 사용