1 Implementation of a dictionary with a binary search tree

Variables

tree: binary tree
invariant: *tree* is a binary search tree; the internal nodes of *tree* contain the items of the dictionary; the external nodes of *tree* are empty

Initialization

tree ← tree consisting of a single empty node

Algorithms

size():
  output: size of dictionary
return $\frac{\text{size of } tree - 1}{2}$

isEmpty():
  output: dictionary is empty?
return (size of *tree* = 1)

findElement(*key*):
  input: key to be searched for
  output: element of item with *key* in dictionary; NO-SUCH-KEY if no such item exists
return findElement(*key*, root of *tree*)

findElement(*key*, *node*)
  input: key to be searched for; root of subtree to be searched
  output: element of item with *key* in subtree rooted at *node*; NO-SUCH-KEY if no such item exists
if *node* is leaf **then**
    return NO-SUCH-KEY
else if key of *node* = *key* **then**
    return element of *node*
else if key of *node* > *key* **then**
    return findElement(*key*, left child of *node*)
else (key of *node* < *key*)
    return findElement(*key*, right child of *node*)

insertItem(*key*, *element*)
  input: item to be inserted
postcondition: item (*key*, *element*) has been inserted into dictionary
insertItem(*key*, *element*, root of *tree*)

insertItem(*key*, *element*, *node*)
  input: item to be inserted; root of subtree to be inserted in
  postcondition: item (*key*, *element*) has been inserted into subtree rooted at *node*
if *node* is leaf **then**
    replace *node* with node containing (*key*, *element*) with two empty children
else if key of *node* > *key* **then**
    insertItem(*key*, *element*, left child of *node*)
else (key of *node* ≤ *key*)
    insertItem(*key*, *element*, right child of *node*)

remove(*key*):
  input: key to be searched for
  output: element of item with *key* in dictionary; NO-SUCH-KEY if no such item exists

```
    postcondition: item has been removed from dictionary, if such an item exists
return remove(key, root of tree)

remove(key, node):
    input: key to be searched for; root of subtree to be searched
    output: element of item with key in subtree rooted at node; NO-SUCH-KEY if no such item exists
    postcondition: item has been removed from subtree rooted at node, if such an item exists
if node is leaf then
    return NO-SUCH-KEY
else if key of node > key then
    return remove(key, left child of node)
else if key of node < key then
    return remove(key, right child of node)
else (key of node = key)
    element ← element of node
    if node has no nonempty children then
        replace node with empty leaf
    else if node has only one nonempty child then
        replace node with its nonempty child
    else
        item ← removeMin(right child of node)
        store item in node
    return element

removeMin(node):
    input: root of subtree
    output: item with minimal key in subtree rooted at node
    precondition: node is nonempty
    postcondition: node of item with minimal key has been removed from subtree rooted at node
if node has an empty left child then
    item ← item of node
    replace node with its right child
    return item
else
    return removeMin(left child of node)
```