1 Binary search tree implementation of a dictionary

Variables

tree: binary tree
inv: tree is a binary search tree containing the items of the dictionary

Algorithms

size()
out: size of dictionary
return (size of tree)

isEmpty()
out: dictionary is empty?
return (tree is empty?)

findElement(key)
in: key to be searched for
out: element of item with key in dictionary; NO-SUCH-KEY if no such item exists

```
if tree is empty
    return NO-SUCH-KEY
else
    return findElement(key, root of tree)
```

findElement(key, node)
in: key to be searched for; root of subtree to be searched
out: element of item with key in subtree rooted at node; NO-SUCH-KEY if no such item exists

```
if node is leaf
    if key of node = key
        return element of node
    else
        return NO-SUCH-KEY
else
    if key of node = key
        return element of node
    else if key of node > key
        if node has left child
            return findElement(key, left child of node)
        else
            return NO-SUCH-KEY
    else (key of node < key)
        if node has right child
            return findElement(key, right child of node)
        else
            return NO-SUCH-KEY
```

insertItem(key, element)
in: item to be inserted
post: item (key, element) has been inserted into dictionary

```
if tree is empty
    let node containing (key, element) be the root of tree
else
    return insertItem(key, element, root of tree)

insertItem(key, element, node)
in: item to be inserted; root of subtree to be inserted in
post: item (key, element) has been inserted into subtree rooted at node
if node is leaf
    if key of node ≥ key
        add node containing (key, element) as left child of node
    else
        add node containing (key, element) as right child of node
else
    if key of node ≥ key
        if node has left child
            insertItem(key, element, left child of node)
        else
            add node containing (key, element) as left child of node
    else (key of node < key)
        if node has right child
            insertItem(key, element, right child of node)
        else
            add node containing (key, element) as right child of node

remove(key)
in: key to be searched for
out: element of item with key in dictionary; NO-SUCH-KEY if no such item exists
post: item has been removed from dictionary
if tree is empty
    return NO-SUCH-KEY
else
    return remove(key, root of tree)

remove(key, node)
in: key to be searched for; root of subtree to be searched
out: element of item with key in subtree rooted at node; NO-SUCH-KEY if no such item exists
post: item has been removed from subtree rooted at node
if node is leaf
    if key of node = key
        return element of node
        remove node
    else
        return NO-SUCH-KEY
else
    if key of node > key
        if node has left child
            return remove(key, left child of node)
        else
```

```
            return NO-SUCH-KEY
    else if key of node < key
        if node has right child
            return remove(key, right child of node)
        else
            return NO-SUCH-KEY
    else (key of node = key)
        if node has only one child
            return element of node
            replace node by its child
        else
            let item = removeMin(right child of node)
            return element of node
            store item in node
```

removeMin(node)
in: root of subtree
out: item with minimal key in subtree rooted at node
post: node of item with minimal key has been removed from subtree rooted at node

```
if node is leaf
    return item of node
    remove node
else
    if node has left child
        removeMin(left child of node)
    else
        return item of node
        replace node by its right child
```