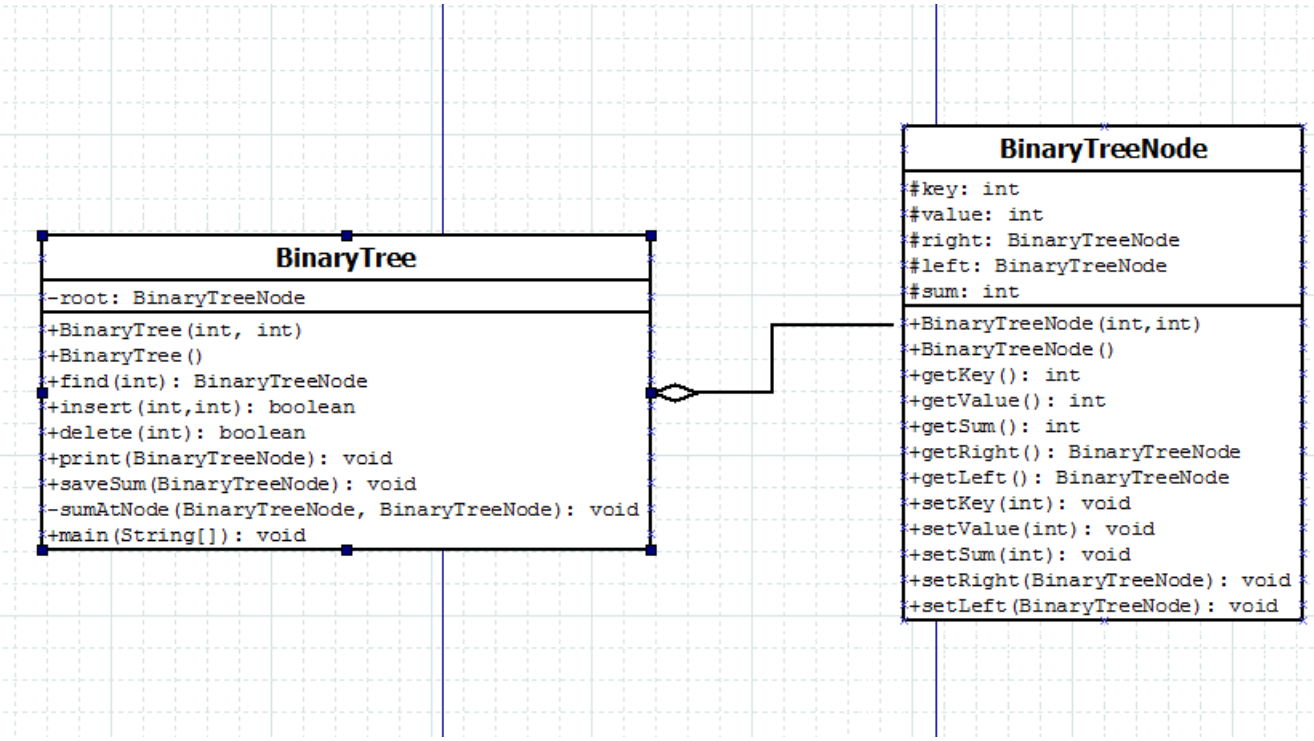
Aufgabenblatt 5
Binary Search Tree Algorithmus

Cao, Thi Huyen(Lilli)

October 18, 2016

Contents

1 UML



2 API

Anbei sind Javadoc in Pdf Dateien von 2 Klassen: BinaryTreeNode.java und BinaryTree.java

aufgabenblatt5

Class BinaryTreeNode

java.lang.Object
    aufgabenblatt5.BinaryTreeNode

```
public class BinaryTreeNode
extends java.lang.Object
```

This class describes a node in a binary search tree, which contains data, an extra information (which is the summary of all values, key of which are smaller than key of node in the tree) and 2 pointers pointing to other nodes or NULL

Author:

cao

Constructor Summary

Constructors

Constructor and Description
`BinaryTreeNode()`
`BinaryTreeNode(int key, int value)`

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
int	`getKey()`
BinaryTreeNode	`getLeft()`
BinaryTreeNode	`getRight()`
int	`getSum()`
int	`getValue()`
void	`setKey(int key)`
void	`setLeft(BinaryTreeNode left)`
void	`setRight(BinaryTreeNode right)`
void	`setSum(int sum)`
void	`setValue(int value)`

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

BinaryTreeNode

```
public BinaryTreeNode(int key,
                      int value)
```

BinaryTreeNode

```
public BinaryTreeNode()
```

Method Detail

getKey

```
public int getKey()
```

setKey

```
public void setKey(int key)
```

getValue

```
public int getValue()
```

setValue

```
public void setValue(int value)
```

getRight

```
public BinaryTreeNode getRight()
```

setRight

```
public void setRight(BinaryTreeNode right)
```

getLeft

```
public BinaryTreeNode getLeft()
```

setLeft

```
public void setLeft(BinaryTreeNode left)
```

getSum

```
public int getSum()
```

setSum

```
public void setSum(int sum)
```

aufgabenblatt5

Class BinaryTree

java.lang.Object
    aufgabenblatt5.BinaryTree

```
public class BinaryTree
extends java.lang.Object
```

This class describes a special binary tree. A binary tree has nodes containing key, value, summary of all values, keys of which are smaller than the current node, and two references to other nodes or NULL, one on the left and one on the right

Author:

cao

Constructor Summary

Constructors

Constructor and Description
`BinaryTree()`
`BinaryTree(int key, int value)`

Method Summary

All Methods Static Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
boolean	`delete(int key)` This method deletes the node with the key in the tree and change the references between related nodes.
`BinaryTreeNode`	`find(int key)` This method searches for a node with an input key in a binary search tree.
boolean	`insert(int key, int value)` This method inserts a node with key in a binary search tree.
static void	`main(java.lang.String[] args)` main function: quick test
void	`print(BinaryTreeNode root)` Output through console: all nodes with key and value in the undertree, the root of which is the input node
void	`saveSum(BinaryTreeNode underRoot)` This method writes in every node the summary of all values of nodes (which are in the undertree with input root), the keys of which are smaller

which are in the undertree with input root), the keys of which are smaller than its key in the tree

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

BinaryTree

```
public BinaryTree(int key,
                  int value)
```

BinaryTree

```
public BinaryTree()
```

Method Detail

find

```
public BinaryTreeNode find(int key)
                   throws java.lang.NullPointerException
```

This method searches for a node with an input key in a binary search tree. Return the node if the node with the key is found, otherwise throw NullPointerException("key isnt in tree")

Parameters:

key -

Returns:

found node/ Exception

Throws:

java.lang.NullPointerException

insert

```
public boolean insert(int key,
                      int value)
```

This method inserts a node with key in a binary search tree. If the key is already in the tree, return false (which means for this data structure is no duplicates of key allowed), otherwise return true(the node with key is successfully inserted)

Parameters:

key -

Returns:

`true/false`

delete

```
public boolean delete(int key)
```

This method deletes the node with the key in the tree and change the references between related nodes. Return true if the key is successfully deleted, otherwise return false if the key is not found.

Parameters:

`key -`

Returns:

`true/false`

print

```
public void print(BinaryTreeNode root)
```

Output through console: all nodes with key and value in the undertree, the root of which is the input node

saveSum

```
public void saveSum(BinaryTreeNode underRoot)
```

This method writes in every node the summary of all values of nodes (which are in the undertree with input root), the keys of which are smaller than its key in the tree

Parameters:

`underRoot -`

main

```
public static void main(java.lang.String[] args)
```

main function: quick test

Parameters:

`args -`

3 Summenberechnung Funktion

3.1 Semantik

IDEE: Jeder Knoten in dem Binären Suchbaum enthält eine extra Information, die die Summe aller Werte von allen Knoten ist, dessen Schlüssel kleiner als Schlüssel von dem Knoten selbst sind. Diese Summe lässt sich durch 2 Funktionen realisiert.
1. Hilffunktion: sumAtNode(BinaryTreeNode, BinaryTreeNode): Diese Funktion berechnet die Summe aller Werte von allen Knoten in dem Unterbaum mit dem eingegebenen Würzel,dessen Schlüssel kleiner als der eingegebene Knoten sind und speichert es in dem eingegeben Knoten.
2. sumSave(BinaryNodeTree): Diese Funktion ruft Hilffunktion sumAtNode(BinaryTreeNode, BinaryTreeNode) mit dem Würzel von dem Baum und somit wird in allen Knoten in dem Baum die Summe von allen Werten von allen Knoten, dessen Schllüssel kleiner als Schlüssel von dem Knoten selbst gespeichert.

3.2 Aufwandsanalyse

⋄Einfache Ansatz: Durchsuchen den ganzen Baum und vergleichen den Schlüssel von allen Knoten mit m und M und somit bilden die Summe aller Werte
$\sum_i a_i; m \leq a_i \leq M$
→ Komplexiät: T=O(n)
beim neuen Eingaben von m und M oder sogar nur von m oder M muss den ganzen Baum wieder durchgesucht werden
⋄Ansatz mit extra Information: Da wird die Funktion saveSum einmal ausgeführt und somit wird summe aller Werte, dessen Schlüssel kleiner als Schlüssel von dem aktuellen Knoten gespeichert. Für die Summenberechnung ist es nur notwendig, die Knoten mit den Schlüssel m und M durchzusuchen und die Summe davon abzufragen.

$$\text{Sum} = \text{sum(M)} - \text{sum(n)} + \text{value(M)} + \text{value(n)}$$

→ Komplexität für die Summenberechnung: T= O(1)
→ Komplexität für das Suchen: Worstcase O(Höhe von Baum), Bestcase O(1) (Würzel)

END