PA_07 - Red Black Tree

Generated by Doxygen 1.8.6

Mon Dec 12 2016 21:45:27

Contents

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

BinaryNode	**2**
RedBlackTree	**2**

2 File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

BinaryNode.h	**??**
RedBlackTree.cpp **Implementation file for RedBlackTree class**	**14**
RedBlackTree.h **Definition file for RedBlackTree class**	**15**

3 Class Documentation

3.1 BinaryNode Class Reference

Public Member Functions

- **BinaryNode** (int)
- **BinaryNode** (int, BinaryNode ∗, BinaryNode ∗)
- void **setItem** (int)
- int **getItem** () const
- bool **isLeaf** () const
- int **getNumOfChildren** () const
- BinaryNode ∗ **getParent** () const
- BinaryNode ∗ **getLeftChildPtr** () const
- BinaryNode ∗ **getRightChildPtr** () const
- Color **getColor** () const
- void **setParent** (BinaryNode ∗)
- void **setLeftChildPtr** (BinaryNode ∗)
- void **setRightChildPtr** (BinaryNode ∗)
- void **setColor** (Color)

Private Attributes

- int **data**
- BinaryNode ∗ **parent**
- BinaryNode ∗ **leftChildPtr**
- BinaryNode ∗ **rightChildPtr**
- Color **color**

The documentation for this class was generated from the following files:

- BinaryNode.h
- BinaryNode.cpp

3.2 RedBlackTree Class Reference

Public Member Functions

- RedBlackTree ()

  Constructor for class RedBlackTree.
- RedBlackTree (int)

  Constructor for class RedBlackTree.
- ∼RedBlackTree ()

  Destructor for class RedBlackTree.
- int getHeightHelper (BinaryNode ∗) const

  Gets the height of the tree.
- void destroyTree (BinaryNode ∗)

  Destroys the tree.
- void preorder (void visit(int &), BinaryNode ∗) const

  Traverses the tree using preorder.
- void inorder (void visit(int &), BinaryNode ∗) const

  Traverses the tree using inorder.
- void postorder (void visit(int &), BinaryNode ∗) const

  Traverses the tree using postorder.
- bool isEmpty () const

Checks if tree is empty.
- int getHeight () const

  Calls recursive getHeightHelper function.
- int getRootData () const

  gets the data stored in rootPtr
- BinaryNode ∗ getRootNode () const

  Returns rootPtr.
- void setRootData (int)

  Sets the data stored in rootPtr to the parameter.
- void insert (int)

  Inserts a new node into the red black tree.
- void insertFix (BinaryNode ∗)

  rebuilds the red black tree based on the new input
- void leftRotate (BinaryNode ∗)

  Rotates subTree to the left.
- void rightRotate (BinaryNode ∗)

  Rotates subTree to the right.
- void clear ()

  Calls recursive destroyTree function.
- void preorderTraverse (void visit(int &)) const

  Calls recursive preorder function.
- void inorderTraverse (void visit(int &)) const

  Calls recursive inorder function.
- void postorderTraverse (void visit(int &)) const

  Calls recursive postorder function.

Private Attributes

- BinaryNode ∗ **rootPtr**

3.2.1 Constructor & Destructor Documentation

3.2.1.1 RedBlackTree::RedBlackTree ( )

Constructor for class RedBlackTree.

Able to construct a RedBlackTree object

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.1.2 RedBlackTree::RedBlackTree (int *a*)

Constructor for class RedBlackTree.

Able to construct a RedBlackTree object with given parameters

Precondition

None

Postcondition

None

None

Parameters

a	- rootPtr will contain this value as an item

Exceptions

None	

Note

: None

3.2.1.3 RedBlackTree::∼RedBlackTree ( )

Destructor for class RedBlackTree.

Able to destruct a RedBlackTree object

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2 Member Function Documentation

3.2.2.1 void RedBlackTree::clear ( )

Calls recursive destroyTree function.

Calls destroyTree function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.2 void RedBlackTree::destroyTree (BinaryNode ∗ *subTreePtr*)

Destroys the tree.

Recursively destroys the tree from the bottom up

Precondition

None

Postcondition

None

Algorithm

Recursively calls destroyTree until the bottom of it is reached, then begins to delete the nodes from the bottom up and set them to nullptr

Parameters

subTreePtr	- the node that will be deleted after the bottom-most nodes are

Exceptions

None	

Note

: None

3.2.2.3 int RedBlackTree::getHeight ( ) const

Calls recursive getHeightHelper function.

Calls getHeightHelper function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.4 int RedBlackTree::getHeightHelper (BinaryNode ∗ *subTreePtr*) const

Gets the height of the tree.

Recursively gets the height of the tree

Precondition

None

Postcondition

None

Algorithm

Returns 0 if there is no node, returns a 1 + the max between a recursive call to the left side of the subtree and a recursive call to the right side of the subtree. This will eventually get you how many layers the tree has in total, which is the height of the tree

Parameters

subTreePtr	- used as the root of the subTree, will call its left side or right side depending on which one has more layers

Exceptions

None	

Note

: None

3.2.2.5 int RedBlackTree::getRootData ( ) const

gets the data stored in rootPtr

returns the data being stored in rootPtr

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.6 BinaryNode ∗ RedBlackTree::getRootNode ( ) const

Returns rootPtr.

None

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.7 void RedBlackTree::inorder (void *visitint &,* BinaryNode ∗ *treePtr*) const

Traverses the tree using inorder.

Traverses the tree recursively, calling visit after it traverses the left side of the tree and before it traverses the right side

Precondition

None

Postcondition

None

Algorithm

Gets the item in treePtr and calls visit using that item in the correct order

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.8 void RedBlackTree::inorderTraverse (void *visitint &*) const

Calls recursive inorder function.

Calls inorder function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.9 void RedBlackTree::insert (int *newData*)

Inserts a new node into the red black tree.

Inserts a new node with specified data into the red black tree using a nonrecursive algorithm

Precondition

newNodePtr is allocated with newData

Postcondition

newNodePtr is inserted into the red black

Algorithm

Node pointers y and x traverse the tree using newData to correctly decide which branch to take until the correct branch is null. y acts as the parent and x acts as the child throughout the algorithm. Once the correct place is found, y becomes newNodePtr's parent and then depending on what value newData is, then newNodePtr becomes y's right or left child.

Parameters

newData	is the data that will be the new node's item

Exceptions

None	

Note

: None

3.2.2.10 void RedBlackTree::insertFix (BinaryNode ∗ *z*)

rebuilds the red black tree based on the new input

rebuilds the red black tree based on the new input and follows the specifications that a red black tree must have

Precondition

uncle gets the new node's left or right uncle, depending on where the new node currently is placed
grandparent gets the new node's parent's parent

Postcondition

uncle and grandparent are set to black or red depending on the rest of the tree

Algorithm

First, checks if the new node, z, is the root pointer. Sets to black if it is and returns, nothing else needs to be done. Else checks if z is equal to null, its parent is equal to null, its grandparent is equal to null, or if its parent's color is equal to black. If any of those are true, nothing needs to be done. If they are all false, then the loop begins. First the loop decides what side the uncle is on based on whether z's parent is a right or left child. Then the loop checks if the uncle is red (and !null) and if it is, then it simply sets the parent and uncle to black, and the grandparent to red, then it sets z to the grandparent so the loop can continue if it has to, this is CASE 1. If the uncle is not red, then it checks if z is a right or left child. If it is in the correct position then the subtree gets rotated around z's parent, CASE 2, and then the subtree gets rotated around the grandparent and z's parent is set to black and z's grandparent is set to red, CASE 3, otherwise it's just case 3. The end of the algorithm is just the root being set to black just in case it got changed.

Parameters

z,the	node that's being inserted

Exceptions

None	

Note

: None

3.2.2.11 bool RedBlackTree::isEmpty ( ) const

Checks if tree is empty.

Returns true if height is 0 and false if it's not

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.12 void RedBlackTree::leftRotate (**BinaryNode** ∗ *x*)

Rotates subTree to the left.

Rotates subTree to the left around pivot x

Precondition

None

Postcondition

None

Algorithm

y gets x's right child. If y's left child is not equal to null, then y's left child becomes x's parent. y's parent then becomes x's parent. If x's parent is equal to null, then x is at the root, so y gets the rootPtr. Otherwise if x is on the left then it gets y's left child and if it's on the right then it gets y's right child. Finally, x becomes y's left child and y becomes x's parent.

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.13 void RedBlackTree::postorder (void *visitint &,* **BinaryNode** ∗ *treePtr*) const

Traverses the tree using postorder.

Traverses the tree recursively, calling visit after it traverses the rest of the tree

Precondition

None

Postcondition

None

Algorithm

Gets the item in treePtr and calls visit using that item in the correct order

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.14 void RedBlackTree::postorderTraverse (void *visitint &*) const

Calls recursive postorder function.

Calls postorder function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.15 void RedBlackTree::preorder (void *visitint &,* **BinaryNode** ∗ *treePtr*) const

Traverses the tree using preorder.

Traverses the tree recursively, calling visit before it traverses the rest of the tree

Precondition

None

Postcondition

None

Algorithm

Gets the item in treePtr and calls visit using that item in the correct order

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.16 void RedBlackTree::preorderTraverse (void *visitint &*) const

Calls recursive preorder function.

Calls preorder function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.17 void RedBlackTree::rightRotate (BinaryNode ∗ *x*)

Rotates subTree to the right.

Rotates subTree to the right around pivot x

Precondition

None

Postcondition

None

Algorithm

y gets x's left child. If y's right child is not equal to null, then y's right child becomes x's parent. y's parent then becomes x's parent. If x's parent is equal to null, then x is at the root, so y gets the rootPtr. Otherwise if x is on the left then it gets y's left child and if it's on the right then it gets y's right child. Finally, x becomes y's right child and y becomes x's parent.

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.18 void RedBlackTree::setRootData (int *data*)

Sets the data stored in rootPtr to the parameter.

None

Precondition

None

Postcondition

None

None

Parameters

data	- rootPtr's item gets set to this integer

Exceptions

None	

Note

: None

The documentation for this class was generated from the following files:

- RedBlackTree.h
- RedBlackTree.cpp

4 File Documentation

4.1 RedBlackTree.cpp File Reference

Implementation file for RedBlackTree class.

```
#include "RedBlackTree.h"
```

4.1.1 Detailed Description

Implementation file for RedBlackTree class.

Author

Alex Kastanek

Implements all member methods of the RedBlackTree class

Version

1.00 C.S. Student (15 November 2016) Initial development and testing of RedBlackTree class

Note

Requires RedBlackTree.h
None

4.2 RedBlackTree.h File Reference

Definition file for RedBlackTree class.

```
#include <iostream>
#include <fstream>
#include <cstddef>
#include "BinaryNode.h"
```

Classes

- class RedBlackTree

4.2.1 Detailed Description

Definition file for RedBlackTree class.

Author

Alex Kastanek

Specifies all member methods of the RedBlackTree class

Version

1.00 C.S. Student (15 November 2016) Initial development and testing of RedBlackTree class

Note

None

Index