CS 302 Project 7 - Patrick Austin

Generated by Doxygen 1.8.6

Wed Apr 22 2015 16:42:41

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 BinaryNode Class Reference

`#include <intervalTree.h>`

Public Member Functions

- BinaryNode ()
- void setStart (const int &target)
- int getStart () const
- void setEnd (const int &target)
- int getEnd () const
- void setMax (const int &target)
- int getMax () const
- bool isRed () const
- bool isBlack () const
- void setRed ()
- void setBlack ()
- void setLeftChildPtr (BinaryNode *leftPtr)
- BinaryNode * getLeftChildPtr () const
- void setRightChildPtr (BinaryNode *rightPtr)
- BinaryNode * getRightChildPtr () const
- void setParentPtr (BinaryNode *parentNodePtr)
- BinaryNode * getParentPtr () const

3.1.1 Detailed Description

Binary node class for use in an interval tree. Uses a bool to store red/blackness, convention is black == true and red == false. Nodes default to red when created.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BinaryNode::BinaryNode ( )

Binary node default constructor. Sets left/right/parent pointers to null and does not assign data. Node is red by default.

Precondition

None.

Postcondition

Object created with left/right/parent pointers set to null, color red.

Returns

None.

3.1.3 Member Function Documentation

3.1.3.1 int BinaryNode::getEnd ( ) const

Binary node getEnd function. Returns the data stored in the node's end data member.

Precondition

None.

Postcondition

Data stored in end data member of the node returned. Node contents unchanged.

Returns

int The value stored in the node's end data member.

3.1.3.2 BinaryNode * BinaryNode::getLeftChildPtr ( ) const

Binary node getLeftChildPtr function.

Precondition

None.

Postcondition

leftChildPtr of the node returned; node contents unchanged

Returns

BinaryNode* The left child pointer.

3.1.3.3 int BinaryNode::getMax ( ) const

Binary node getMax function. Returns the data stored in the node's max data member.

Precondition

None.

Postcondition

Data stored in max data member of the node returned. Node contents unchanged.

Returns

int The value stored in the node's max data member.

3.1.3.4 BinaryNode * BinaryNode::getParentPtr ( ) const

Binary node getParentPtr function.

Precondition

None.

Postcondition

getParentPtr of the node returned; node contents unchanged

Returns

BinaryNode* The parent child pointer.

3.1.3.5 BinaryNode * BinaryNode::getRightChildPtr ( ) const

Binary node getRightChildPtr function.

Precondition

None.

Postcondition

rightChildPtr of the node returned; node contents unchanged

Returns

BinaryNode* The right child pointer.

3.1.3.6 int BinaryNode::getStart ( ) const

Binary node getStart function. Returns the data stored in the node's start data member.

Precondition

None.

Postcondition

Data stored in start data member of the node returned. Node contents unchanged.

Returns

int The value stored in the node's start data member.

3.1.3.7 bool BinaryNode::isBlack ( ) const

Binary node isBlack function. Returns whether the node is black or not.

Precondition

None. Nodes default to black when created.

Postcondition

Bool returned. Node contents unchanged.

Returns

bool Whether the node is black or not.

3.1.3.8 bool BinaryNode::isRed ( ) const

Binary node isRed function. Returns whether the node is red or not.

Precondition

None. Nodes default to black when created.

Postcondition

Bool returned. Node contents unchanged.

Returns

bool Whether the node is red or not.

3.1.3.9 void BinaryNode::setBlack ( )

Binary node setBlack function. Sets the color of the node to black.

Precondition

None. Nodes default to black when created.

Postcondition

Color value of the node set to black.

Returns

None.

3.1.3.10 void BinaryNode::setEnd (const int & *target*)

Binary node setEnd function. Sets the data stored by the end member of the node to the specified value.

Precondition

None.

Postcondition

End data member holds target value

Parameters

target	The data end should contain.

Returns

None.

3.1.3.11 void BinaryNode::setLeftChildPtr (BinaryNode * *leftPtr*)

Binary node setLeftChildPtr function.

Precondition

None.

Postcondition

leftChildPtr of the node set to specified value.

Parameters

leftPtr	The pointer the left pointer value of the node should store.

Returns

None.

3.1.3.12 void BinaryNode::setMax (const int & *target*)

Binary node setMax function. Sets the data stored by the max member of the node to the specified value.

Precondition

None.

Postcondition

Max data member holds target value

Parameters

target	The data max should contain.

Returns

None.

3.1.3.13 void BinaryNode::setParentPtr (BinaryNode * *parentNodePtr*)

Binary node setParentPtr function.

Precondition

None.

Postcondition

parentPtr of the node set to specified value.

Parameters

parentNodePtr	The pointer the right pointer value of the node should store.

Returns

None.

3.1.3.14 void BinaryNode::setRed ( )

Binary node setRed function. Sets the color of the node to red.

Precondition

None. Nodes default to red when created.

Postcondition

Color value of the node set to red.

Returns

None.

3.1.3.15 void BinaryNode::setRightChildPtr (BinaryNode * *rightPtr*)

Binary node setRightChildPtr function.

Precondition

None.

Postcondition

rightChildPtr of the node set to specified value.

Parameters

rightPtr	The pointer the right pointer value of the node should store.

Returns

None.

3.1.3.16 void BinaryNode::setStart (const int & *target*)

Binary node setStart function. Sets the data stored by the start member of the node to the specified value.

Precondition

None.

Postcondition

Start data member holds target value

Parameters

target	The data start should contain.

Returns

None.

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

- intervalTree.h
- intervalTree.cpp

3.2 IntervalTree Class Reference

`#include <intervalTree.h>`

Public Member Functions

- IntervalTree ()
- ~IntervalTree ()
- bool isEmpty () const
- bool add (const int &s, const int &e)
- bool search (const int &s, const int &e, int &foundS, int &foundE)
- void preorderTraverse () const

Protected Member Functions

- void insertInorder (BinaryNode *z)
- void fixTree (BinaryNode *z)
- void rotateLeft (BinaryNode *x)
- void rotateRight (BinaryNode *x)
- void fixMax (BinaryNode *z)
- int getHeight (BinaryNode *subTreePtr) const
- void preorder (BinaryNode *treePtr) const
- void clear (BinaryNode *treePtr)
- BinaryNode * searchHelper (const int &s, const int &e) const

3.2.1 Detailed Description

Interval tree class. Only add, search, and preorder traversal operations supported. Duplicate start values for intervals not supported. Caveat emptor.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 IntervalTree::IntervalTree ( )

Interval tree default constructor. Creates and sets tree to empty.

Precondition

None.

Postcondition

Empty tree created.

Returns

None.

3.2.2.2 IntervalTree::~IntervalTree ( )

Interval tree destructor. Uses clear helper function to deallocate dynamic memory used for the tree.

Precondition

None.

Postcondition

Dynamic memory used in the tree freed.

Returns

None.

3.2.3 Member Function Documentation

3.2.3.1 bool IntervalTree::add (const int & *s*, const int & *e*)

Interval tree add function. Adds an interval to the tree. Uses insertInorder, fixTree, rotateLeft, rotateRight, fixMax helper functions to maintain sorted and red/black properties in the tree and maintain correct max values in the nodes.

Precondition

s <= e

Postcondition

Interval added to the tree.

Parameters

s	The start of the interval to add.
e	The end of the interval to add.

Returns

bool Returns true, denoting successful add.

3.2.3.2 void IntervalTree::clear (BinaryNode * *treePtr*) [protected]

Interval tree clear helper function. Used by the destructor to deallocate dynamic memory used in the tree. Uses recursion.

Precondition

None.

Postcondition

Dynamic memory deallocated, tree contents destroyed and returned to empty state.

Parameters

treePtr	The tree to visit.

Returns

None.

3.2.3.3 void IntervalTree::fixMax (BinaryNode * z) [protected]

Interval tree fixMax helper function. After a node has been added to the tree and the tree has been re-oriented to maintain red/black rules, adjusts max values in the nodes above it if necessary, denoting the larget value in the subtree. Uses recursion.

Precondition

Node z has been added to the tree and the tree has been fixed to maintain red/black rules.

Postcondition

Max values in the parent, grandparent, etc. nodes of this one adjusted if appropriate.

Parameters

z	The node that has been added.

Returns

None.

3.2.3.4 void IntervalTree::fixTree (BinaryNode * z) [protected]

Interval tree fixTree helper function. Called by insertInorder function to maintain red/black tree properties after a new node has been added.

Precondition

Node z contains valid data and has just been added to a previously valid red/black tree.

Postcondition

Tree color and orientation potentially altered in order to maintain red/black tree rules.

Parameters

z	The node which has been added to the tree, from which the fixing of the tree will begin.

Returns

None.

3.2.3.5 int IntervalTree::getHeight (BinaryNode * *subTreePtr*) const [protected]

Interval tree getHeight helper function. Used in preorder traversal to show the height of the nodes in the tree. Uses recursion. Note that height for a non-null node is the taller of its subtrees plus one. IE a node with height 3 can have a height 2 child and a height 1 child, etc.

Precondition

None.

Postcondition

Height of this subtree returned.

Parameters

subTreePtr	The subtree to check.

Returns

int The height of the subtree.

3.2.3.6 void IntervalTree::insertInorder (BinaryNode * *z*) [protected]

Interval tree insertInorder helper function. Called by add function to add nodes to the tree.

Precondition

Node z contains valid data (no duplicate start values allowed), has null children.

Postcondition

Node added to the interval tree with sorted property maintained and max values updated.

Parameters

z	The node to be added.

Returns

None.

3.2.3.7 bool IntervalTree::isEmpty ( ) const

Interval tree isEmpty function. Returns true if empty, false if not.

Precondition

None.

Postcondition

bool denoting emptiness returned; tree contents unchanged.

Returns

bool True if empty, false if not empty.

3.2.3.8 void IntervalTree::preorder (BinaryNode * *treePtr*) const [protected]

Interval tree preorder helper function. Used by public preorder traversal function. Uses recursion. Displays start, end, max, color, and height for each node in preorder order (root, left, right).

Precondition

None.

Postcondition

Tree contents unchanged; values of each node printed to console.

Parameters

treePtr	The tree to visit.

Returns

None.

3.2.3.9 void IntervalTree::preorderTraverse ( ) const

Interval tree preorder traversal function. Uses preorder helper function. Visits the nodes in the tree in preorder order (root, left, right) and prints the start, end, max, color, and height of each node to console.

Precondition

None.

Postcondition

Values of the nodes printed to console; tree contents unchanged.

Returns

None.

3.2.3.10 void IntervalTree::rotateLeft (BinaryNode * *x*) [protected]

Interval tree rotateLeft helper function. Used by fixTree function in some situations in order to re-orient the tree so that red/black rules can be maintained.

Precondition

Node x is in the previously valid red/black tree.

Postcondition

x rotated left in the tree- data maintained, just re-oriented.

Parameters

x	The node to rotate left.

Returns

None.

3.2.3.11 void IntervalTree::rotateRight (BinaryNode * *x*) [protected]

Interval tree rotateRight helper function. Used by fixTree function in some situations in order to re-orient the tree so that red/black rules can be maintained.

Precondition

Node x is in the previously valid red/black tree.

Postcondition

x rotated right in the tree- data maintained, just re-oriented.

Parameters

x	The node to rotate right.

Returns

None.

3.2.3.12 bool IntervalTree::search (const int & *s,* const int & *e,* int & *foundS,* int & *foundE*)

Interval tree search function. Searches for an interval overlapping with the specified interval within the interval tree. Uses searchHelper helper function to find an overlapping node if there is one.

Precondition

s <= e

Postcondition

If an overlapping interval has been found, returns true and updates foundS and foundE variables to contain the start and end of the interval found. If no overlapping interval found, returns false and does not update foundS and foundE from the provided values. Tree contents are unchanged by search process.

Parameters

s	The start of the interval to search for.
e	The end of the interval to search for.
foundS	If the search is successful, this value will be updated to contain the start of the found interval. If search is unsuccessful, this value is not updated.

foundE	If the search is successful, this value will be updated to contain the end of the found interval. If search is unsuccessful, this value is not updated.

Returns

bool True if an overlapping interval was found, false if not.

3.2.3.13 BinaryNode * IntervalTree::searchHelper (const int & *s*, const int & *e*) const [protected]

Interval tree searchHelper helper function. Used by the public search method to find a node that overlaps with the interval specified, if there is one.

Precondition

s <= e

Postcondition

Node returned if found; else nullptr returned; tree contents unchanged.

Parameters

s	The start of the interval to search for.
e	The end of the interval to search for.

Returns

BinaryNode* The overlapping node that was found if found, null otherwise.

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

- intervalTree.h
- intervalTree.cpp

Chapter 4

File Documentation

4.1 intervalTree.cpp File Reference

```
#include "intervalTree.h"
#include <iostream>
```

4.1.1 Detailed Description

CS 302 Project 7 - scheduling with red black trees, class implementation

Author

Patrick Austin

Date

4/29/2015

4.2 intervalTree.h File Reference

Classes

- class BinaryNode
- class IntervalTree

4.2.1 Detailed Description

CS 302 Project 7 - scheduling with red black trees, class specification

Author

Patrick Austin

Date

4/29/2015

4.3 main.cpp File Reference

```
#include "intervalTree.h"
#include <iostream>
#include <fstream>
```

Functions

- void readin (IntervalTree &target)
- int main ()

4.3.1 Detailed Description

CS 302 Project 7 - scheduling with red black trees, main program for scheduling

Author

Patrick Austin

Date

4/29/2015

4.3.2 Function Documentation

4.3.2.1 int main ( )

4.3.2.2 void readin (IntervalTree & *target*)

Readin function. Reads correctly formatted data from "data.txt" into target interval tree. Reads 10 intervals by default.

Precondition

Datafile is correctly formatted.

Postcondition

Intervals read into the interval tree.

Parameters

target	The interval tree into which the values should be inserted.

Returns

None.

Index