Binary search tree project

Generated by Doxygen 1.8.17

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 documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 _iterator< node_t, pair_type > Class Template Reference

Public Types

- using **value_type** = pair_type
- using **reference** = value_type &
- using **pointer** = value_type *
- using **iterator_category** = std::forward_iterator_tag
- using **difference_type** = std::ptrdiff_t

Public Member Functions

- _iterator (node_t *n) noexcept

  Constructs a new _iterator setting current to the input pointer.
- ~_iterator () noexcept=default

  Destroys the _iterator object.
- _iterator (const _iterator &i) noexcept

  Copy constructor.
- _iterator & operator= (const _iterator &i) noexcept

  Copy assignment.
- reference operator* () const noexcept

  Dereference operator.
- _iterator & operator++ () noexcept

  Pre-increment operator. It is used to traverse the tree from left to right.
- bool operator== (const _iterator &a) const noexcept

  Boolean equality operator.
- bool operator!= (const _iterator &a) const noexcept

  Boolean inequality operator.

3.1.1 Constructor & Destructor Documentation

3.1.1.1 _iterator() [1/2]

```
template<typename node_t , typename pair_type >
_iterator< node_t, pair_type >::_iterator (
            node_t * n )  [inline], [explicit], [noexcept]
```

Constructs a new _iterator setting current to the input pointer.

Template Parameters

n	input pointer to a node.

3.1.1.2 _iterator() [2/2]

```
template<typename node_t , typename pair_type >
_iterator< node_t, pair_type >::_iterator (
            const _iterator< node_t, pair_type > & i )  [inline], [noexcept]
```

Copy constructor.

Template Parameters

i	const lvalue reference to _iterator.

3.1.2 Member Function Documentation

3.1.2.1 operator"!=()

```
template<typename node_t , typename pair_type >
bool _iterator< node_t, pair_type >::operator!= (
            const _iterator< node_t, pair_type > & a ) const  [inline], [noexcept]
```

Boolean inequality operator.

Template Parameters

a	const lvalue reference to _iterator.

Returns

bool true if the iterators point to different nodes.

3.1.2.2 operator*()

```
template<typename node_t , typename pair_type >
reference _iterator< node_t, pair_type >::operator* ( ) const  [inline], [noexcept]
```

Dereference operator.

Returns

reference to the value stored in the node pointed by the iterator.

3.1.2.3 operator++()

```
template<typename node_t , typename pair_type >
_iterator& _iterator< node_t, pair_type >::operator++ ( )  [inline], [noexcept]
```

Pre-increment operator. It is used to traverse the tree from left to right.

Returns

_iterator& reference to _iterator.

3.1.2.4 operator=()

```
template<typename node_t , typename pair_type >
_iterator& _iterator< node_t, pair_type >::operator= (
            const _iterator< node_t, pair_type > & i )  [inline], [noexcept]
```

Copy assignment.

Template Parameters

i	const lvalue reference to _iterator.

3.1.2.5 operator==()

```
template<typename node_t , typename pair_type >
bool _iterator< node_t, pair_type >::operator== (
            const _iterator< node_t, pair_type > & a ) const  [inline], [noexcept]
```

Boolean equality operator.

Template Parameters

a	const lvalue reference to _iterator.

Returns

bool true if the iterators point to the same node.

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

- iterator.hpp

3.2 bst< key_type, value_type, cmp_op > Class Template Reference

Public Member Functions

- bst (cmp_op x) noexcept

  Constructs a new bst object with a comparison operator of type cmp_op.
- bst () noexcept=default

  Constructs a new bst object.
- ~bst () noexcept=default

  Destroys the bst object.
- bst (const bst &b)

  Copy constructor. Creates a deep copy of a bst tree calling the copy constructor of node.
- bst & operator= (const bst &b)

  Copy assignment.
- bst (bst &&b) noexcept

  Move constructor.
- bst & operator= (bst &&b) noexcept

  Move assignment.
- void clear () noexcept

  This function empties out the tree, releasing the memory occupied by the nodes. root is set to nullptr.
- iterator begin () noexcept

  Overloaded function that returns an iterator pointing to the leftmost node of the tree.
- iterator end () noexcept

  Overloaded function that returns an iterator pointing to one node after the rightmost one.
- const_iterator begin () const noexcept

  Overloaded function that returns a const iterator pointing to the leftmost node of the tree.
- const_iterator end () const noexcept

  Overloaded function that returns a const iterator pointing to one node after the rightmost one.
- const_iterator cbegin () const noexcept

  This function returns a const iterator pointing to the leftmost node of the tree.
- const_iterator cend () const noexcept

  This function returns a const iterator pointing to one node after the rightmost one.
- std::pair< iterator, bool > insert (const pair_type &x)

  Overloaded function that inserts a new node with the given pair, calling _insert.
- std::pair< iterator, bool > insert (pair_type &&x)

  Overloaded function that inserts a new node with the given pair, calling _insert using std::move.
- template<class... Args>
  std::pair< iterator, bool > emplace (Args &&... args)

  This function inserts a new node both with a std::pair <key,value> and with a key and a value.
- iterator find (const key_type &x)

  Overloaded function that searches for a node in the tree, given a key.
- const_iterator find (const key_type &x) const

  Overloaded function that searches for a node in the tree, given a key.
- value_type & operator[ ] (const key_type &x)

Overloaded operator that looks for a key to return the corresponding value. If the key is not present in the tree, it inserts a node with that key and a default value.

- value_type & operator[ ] (key_type &&x)

  Overloaded operator that looks for a key to return the corresponding value. If the key is not present in the tree, it inserts a node with that key and a default value.

- std::size_t size () const noexcept

  This function counts the number of nodes in the tree.

- std::vector< std::pair< key_type, value_type > > vectorize () const

  This function stores in a vector the pairs key,value stored in nodes of the tree.

- void balance ()

  This function balances the tree by calling vectorize, reorder, clear and _insert. At first, it stores the nodes of the tree in a vector calling vectorize; then, the vector is reordered calling reorder. Finally, after calling clear to empty out the tree, the pairs are reinserted calling _insert.

- void erase (const key_type &x)

  This function erases the node with the key given as input (if present) from the tree. At first, it calls _find to have a pointer to the node that has to be erased; then, it calls leftmost in order to know where the possible left child of the node to erase has to be attached. There are three possible cases: the node to erase is the root, it is a left child or it is a right child. In all these cases, the ownership of the node that has to be erased is released and two subcases can arise: if the node that is being erased has a right child, it substitutes the parent and the possible left child is attached to the leftmost node of the right subtree, otherwise the left child substitues the parent. Finally, the pointer to the node that is being erased is used to delete the node.

- bool unbalanced () const noexcept

  This function checks if the tree is unbalanced by calling the function unbalanced_node of struct node.

Friends

- std::ostream & operator<< (std::ostream &os, const bst &x) noexcept

  Friend operator that prints the tree in order (from left to right) using const iterators.

3.2.1 Constructor & Destructor Documentation

3.2.1.1 bst() [1/3]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
bst< key_type, value_type, cmp_op >::bst (
            cmp_op x )  [inline], [explicit], [noexcept]
```

Constructs a new bst object with a comparison operator of type cmp_op.

Template Parameters

x	object of type cmp_op.

3.2.1.2 bst() [2/3]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
```

```
bst< key_type, value_type, cmp_op >::bst (
            const bst< key_type, value_type, cmp_op > & b )  [inline]
```

Copy constructor. Creates a deep copy of a bst tree calling the copy constructor of node.

Template Parameters

b	const lvalue reference to the tree to be copied.

3.2.1.3 bst() [3/3]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
bst< key_type, value_type, cmp_op >::bst (
            bst< key_type, value_type, cmp_op > && b )  [inline], [noexcept]
```

Move constructor.

Template Parameters

b	rvalue reference to the tree to be moved.

3.2.2 Member Function Documentation

3.2.2.1 begin() [1/2]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
const_iterator bst< key_type, value_type, cmp_op >::begin ( ) const  [inline], [noexcept]
```

Overloaded function that returns a const iterator pointing to the leftmost node of the tree.

Returns

const iterator pointing to the leftmost node.

3.2.2.2 begin() [2/2]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
iterator bst< key_type, value_type, cmp_op >::begin ( )  [inline], [noexcept]
```

Overloaded function that returns an iterator pointing to the leftmost node of the tree.

Returns

iterator pointing to the leftmost node.

3.2.2.3 cbegin()

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
const_iterator bst< key_type, value_type, cmp_op >::cbegin ( ) const  [inline], [noexcept]
```

This function returns a const iterator pointing to the leftmost node of the tree.

Returns

const iterator pointing to the leftmost node.

3.2.2.4 cend()

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
const_iterator bst< key_type, value_type, cmp_op >::cend ( ) const  [inline], [noexcept]
```

This function returns a const iterator pointing to one node after the rightmost one.

Returns

const iterator pointing to one node after the rightmost one.

3.2.2.5 emplace()

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
template<class...  Args>
std::pair<iterator, bool> bst< key_type, value_type, cmp_op >::emplace (
            Args &&...  args )  [inline]
```

This function inserts a new node both with a std::pair <key,value> and with a key and a value.

Template Parameters

args	a std::pair or key and value.

Returns

std::pair<iterator,bool> returned by _insert.

3.2.2.6 end() [1/2]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
const_iterator bst< key_type, value_type, cmp_op >::end ( ) const  [inline], [noexcept]
```

Overloaded function that returns a const iterator pointing to one node after the rightmost one.

Returns

const iterator pointing to one node after the rightmost one.

3.2.2.7 end() [2/2]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
iterator bst< key_type, value_type, cmp_op >::end ( )  [inline], [noexcept]
```

Overloaded function that returns an iterator pointing to one node after the rightmost one.

Returns

iterator pointing to one node after the rightmost one.

3.2.2.8 erase()

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
void bst< key_type, value_type, cmp_op >::erase (
            const key_type & x )  [inline]
```

This function erases the node with the key given as input (if present) from the tree. At first, it calls _find to have a pointer to the node that has to be erased; then, it calls leftmost in order to know where the possible left child of the node to erase has to be attached. There are three possible cases: the node to erase is the root, it is a left child or it is a right child. In all these cases, the ownership of the node that has to be erased is released and two subcases can arise: if the node that is being erased has a right child, it substitutes the parent and the possible left child is attached to the leftmost node of the right subtree, otherwise the left child substitues the parent. Finally, the pointer to the node that is being erased is used to delete the node.

Template Parameters

x	const lvalue reference to key.

3.2.2.9 find() [1/2]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
iterator bst< key_type, value_type, cmp_op >::find (
            const key_type & x )  [inline]
```

Overloaded function that searches for a node in the tree, given a key.

Template Parameters

x	const lvalue reference to the key to look for.

Returns

iterator pointing to the node containing the key or to nullptr if the key is not found.

3.2.2.10 find() [2/2]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
const_iterator bst< key_type, value_type, cmp_op >::find (
            const key_type & x ) const  [inline]
```

Overloaded function that searches for a node in the tree, given a key.

Template Parameters

x	const lvalue reference to the key to look for.

Returns

const iterator pointing to the node containing the key or to nullptr if the key is not found.

3.2.2.11 insert() [1/2]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
std::pair<iterator, bool> bst< key_type, value_type, cmp_op >::insert (
            const pair_type & x )  [inline]
```

Overloaded function that inserts a new node with the given pair, calling _insert.

Template Parameters

x	const lvalue reference to the pair to be inserted in the tree.

Returns

std::pair<iterator,bool> returned by _insert.

3.2.2.12 insert() [2/2]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
std::pair<iterator, bool> bst< key_type, value_type, cmp_op >::insert (
            pair_type && x )  [inline]
```

Overloaded function that inserts a new node with the given pair, calling _insert using std::move.

Template Parameters

x	rvalue reference to the pair to be inserted in the tree.

Returns

std::pair<iterator,bool> returned by _insert.

3.2.2.13 operator=() [1/2]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
bst& bst< key_type, value_type, cmp_op >::operator= (
            bst< key_type, value_type, cmp_op > && b )  [inline], [noexcept]
```

Move assignment.

Template Parameters

b	rvalue reference to the tree to be moved.

Returns

bst&.

3.2.2.14 operator=() [2/2]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
bst& bst< key_type, value_type, cmp_op >::operator= (
            const bst< key_type, value_type, cmp_op > & b )  [inline]
```

Copy assignment.

Template Parameters

b	const lvalue reference to the tree to be copied.

Returns

bst&.

3.2.2.15 operator[]() [1/2]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
value_type& bst< key_type, value_type, cmp_op >::operator[] (
            const key_type & x )  [inline]
```

Overloaded operator that looks for a key to return the corresponding value. If the key is not present in the tree, it inserts a node with that key and a default value.

Template Parameters

x	const lvalue reference to key.

Returns

value_type& value mapped by the key.

3.2.2.16 operator[]() [2/2]

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
value_type& bst< key_type, value_type, cmp_op >::operator[] (
            key_type && x )  [inline]
```

Overloaded operator that looks for a key to return the corresponding value. If the key is not present in the tree, it inserts a node with that key and a default value.

Template Parameters

x	rvalue reference to key.

Returns

value_type& value mapped by the key.

3.2.2.17 size()

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
std::size_t bst< key_type, value_type, cmp_op >::size ( ) const  [inline], [noexcept]
```

This function counts the number of nodes in the tree.

Returns

std::size_t number of nodes in the tree.

3.2.2.18 unbalanced()

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
bool bst< key_type, value_type, cmp_op >::unbalanced ( ) const  [inline], [noexcept]
```

This function checks if the tree is unbalanced by calling the function unbalanced_node of struct node.

Returns

bool true if the tree is unbalanced.

3.2.2.19 vectorize()

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
std::vector<std::pair<key_type,value_type> > bst< key_type, value_type, cmp_op >::vectorize
( ) const  [inline]
```

This function stores in a vector the pairs key,value stored in nodes of the tree.

Returns

std::vector<std::pair<key_type,value_type>> vector containing the pairs stored in the nodes of the tree.

3.2.3 Friends And Related Function Documentation

3.2.3.1 operator<<

```
template<typename key_type , typename value_type , typename cmp_op = std::less<key_type>>
std::ostream& operator<< (
            std::ostream & os,
            const bst< key_type, value_type, cmp_op > & x )  [friend]
```

Friend operator that prints the tree in order (from left to right) using const iterators.

Parameters

os	output stream object.

Template Parameters

x	const lvalue reference to binary search tree.

Returns

std::ostream& output stream object.

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

- bst.hpp

3.3 node< T > Struct Template Reference

Public Types

- using **value_type** = T

Public Member Functions

- node (const T &elem, node *p=nullptr) noexcept

  Constructs a node initializing left and right children to nullptr, parent to the input pointer to node, element to the input data.
- node (T &&elem, node *p=nullptr) noexcept

  Constructs a node initializing left and right children to nullptr, parent to the input pointer to node, element to the input data using std::move.
- node (const node &n, node *p=nullptr)

  Copy constructor. It calls itself recursively.
- ~node () noexcept=default

  Destroys the node object.
- std::pair< int, int > num_nodes () const noexcept

  This function counts the number of right and left descendants of a node.
- std::pair< bool, const node * > unbalanced_node () const noexcept

  This function checks if the subtree on whose root it is invoked is unbalanced and calls itself recursively on children nodes.

Public Attributes

- std::unique_ptr< node > right

  Unique pointer to the right child.
- std::unique_ptr< node > left

  Unique pointer to the left child.
- node * parent

  Raw pointer to the parent node.
- T element

  Member variable of type T.

3.3.1 Constructor & Destructor Documentation

3.3.1.1 node() [1/3]

```
template<typename T >
node< T >::node (
            const T & elem,
            node< T > * p = nullptr )  [inline], [noexcept]
```

Constructs a node initializing left and right children to nullptr, parent to the input pointer to node, element to the input data.

Template Parameters

elem	const lvalue reference to the type of element.
p	pointer to parent node.

3.3.1.2 node() [2/3]

```
template<typename T >
node< T >::node (
            T && elem,
            node< T > * p = nullptr )  [inline], [noexcept]
```

Constructs a node initializing left and right children to nullptr, parent to the input pointer to node, element to the input data using std::move.

Template Parameters

elem	rvalue reference to the type of element.
p	pointer to parent node.

3.3.1.3 node() [3/3]

```
template<typename T >
node< T >::node (
            const node< T > & n,
            node< T > * p = nullptr )  [inline]
```

Copy constructor. It calls itself recursively.

Template Parameters

elem	const lvalue reference to node to be copied.
p	pointer to parent node.

3.3.2 Member Function Documentation

3.3.2.1 num_nodes()

```
template<typename T >
std::pair<int, int> node< T >::num_nodes ( ) const  [inline], [noexcept]
```

This function counts the number of right and left descendants of a node.

Returns

std::pair<int,int> number of right and left descendants.

3.3.2.2 unbalanced_node()

```
template<typename T >
std::pair<bool,const node*> node< T >::unbalanced_node ( ) const  [inline], [noexcept]
```

This function checks if the subtree on whose root it is invoked is unbalanced and calls itself recursively on children nodes.

Returns

std::pair<bool,const node*> true,pointer to the node where unbalance is found or false,nullptr otherwise.

The documentation for this struct was generated from the following file:

- node.hpp

Chapter 4

File Documentation

4.1 bst.hpp File Reference

Header containing bst class implementation.

```
#include <utility>
#include <memory>
#include "node.hpp"
#include "iterator.hpp"
#include <vector>
```

Classes

- class bst< key_type, value_type, cmp_op >

Functions

- template<typename T >
  void reorder (std::vector< T > &v, std::vector< T > &median)

  A utility for the function balance(): given a vector ordered in some way, this function builds another vector containing "moving" median values. Given a vector, the function stores in another vector the median element of the previous vector and removes from the vector the median value. Then, it calls itself recursively on right and left subvectors, until the size of the vector is equal to one.

4.1.1 Detailed Description

Header containing bst class implementation.

Author

Lorenzo Basile

Arianna Tasciotti

Date

February 2020

4.1.2 Function Documentation

4.1.2.1 reorder()

```
template<typename T >
void reorder (
             std::vector< T > & v,
             std::vector< T > & median )
```

A utility for the function balance(): given a vector ordered in some way, this function builds another vector containing "moving" median values. Given a vector, the function stores in another vector the median element of the previous vector and removes from the vector the median value. Then, it calls itself recursively on right and left subvectors, until the size of the vector is equal to one.

Template Parameters

v	lvalue reference to std::vector<T> input vector.
median	lvalue reference to std::vector<T> vector in which the previous is reordered.

4.2 iterator.hpp File Reference

Header containing _iterator class implementation.

```
#include "node.hpp"
```

Classes

- class _iterator< node_t, pair_type >

4.2.1 Detailed Description

Header containing _iterator class implementation.

Author

Lorenzo Basile

Arianna Tasciotti

Date

February 2020

4.3 node.hpp File Reference

Header containing node struct implementation.

```
#include <memory>
#include <utility>
```

Classes

- struct node< T >

4.3.1 Detailed Description

Header containing node struct implementation.

Author

Lorenzo Basile

Arianna Tasciotti

Date

February 2020