AVL tree

Generated by Doxygen 1.9.3

Chapter 1

Binary Tree

1.1 Description

This program developed in C++17 allows using an Abstract Data Type such as binary search trees (BST) and binary balanced search trees (AVL), where you can perform basic operations and see the content of the tree by console. command line (CLI), for more information the documentation is in the /doc directory

1.2 Usage

To compile the program, the *make* command is used in the main directory and the executable of the program is located in the /bin/main directory.

1.3 Author

Fabrizzio Daniell Perilli Martín – alu0101138589@ull.edu.es

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

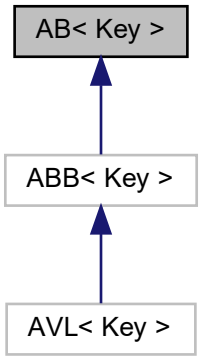
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 AB< Key > Class Template Reference

Inheritance diagram for AB< Key >:



Public Member Functions

- AB ()

  Construct a new AB<Key>::AB object.
- virtual ~AB ()

  Destroy the AB<Key>::AB object.
- bool isEmpty (NodeB< Key > *)

  Check if node is empty or not.
- bool isLeafNode (NodeB< Key > *)

  Check if node is leaf.
- int size () const

  This method return the size of tree.
- int height () const

Return the height tree.

- bool isEquilibrium () const

  This method check if the tree is equilibrium.
- virtual void insert (const Key &)=0
- virtual bool search (const Key &) const =0
- void **inorden** () const

Protected Attributes

- NodeB< Key > * **root_**

Friends

- template<class T >
  std::ostream & **operator**<< (std::ostream &, const AB< T > &)

5.1.1 Constructor & Destructor Documentation

5.1.1.1 AB()

```
template<class Key >
AB< Key >::AB
```

Construct a new AB<Key>::AB object.

Template Parameters

Key	

5.1.1.2 ~AB()

```
template<class Key >
AB< Key >::~AB  [virtual]
```

Destroy the AB<Key>::AB object.

Template Parameters

Key	

5.1.2 Member Function Documentation

5.1.2.1 height()

```
template<class Key >
int AB< Key >::height  [inline]
```

Return the height tree.

Template Parameters

Key	

Returns

const int

5.1.2.2 insert()

```
template<class Key >
virtual void AB< Key >::insert (
            const Key &  )  [pure virtual]
```

Implemented in AVL< Key >.

5.1.2.3 isEmpty()

```
template<class Key >
bool AB< Key >::isEmpty (
            NodeB< Key > * node )
```

Check if node is empty or not.

Template Parameters

Key	

Parameters

node	

Returns

true

false

5.1.2.4 isEquilibrium()

```
template<class Key >
bool AB< Key >::isEquilibrium  [inline]
```

This method check if the tree is equilibrium.

Template Parameters

Key	

Returns

true

false

5.1.2.5 isLeafNode()

```
template<class Key >
bool AB< Key >::isLeafNode (
            NodeB< Key > * node )
```

Check if node is leaf.

Template Parameters

Key	

Parameters

node	

Returns

true

false

5.1.2.6 search()

```
template<class Key >
virtual bool AB< Key >::search (
             const Key &  ) const  [pure virtual]
```

Implemented in ABB< Key >.

5.1.2.7 size()

```
template<class Key >
int AB< Key >::size  [inline]
```

This method return the size of tree.

Template Parameters

Key	

Returns

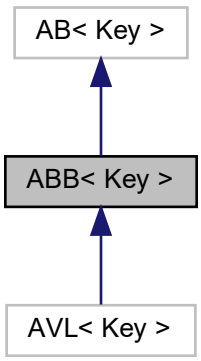
const int

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

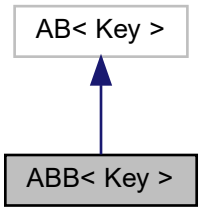
- include/AB.h

5.2 ABB< Key > Class Template Reference

Inheritance diagram for ABB< Key >:



Collaboration diagram for ABB< Key >:



Public Member Functions

- ABB ()

  Construct a new ABB<Key>::ABB object.
- ~ABB ()

  Destroy the ABB<Key>::ABB object.
- void insert (const Key &) override
- virtual bool search (const Key &) const override

  Search a node in the branch tree.

Additional Inherited Members

5.2.1 Constructor & Destructor Documentation

5.2.1.1 ABB()

```
template<class Key >
ABB< Key >::ABB
```

Construct a new ABB<Key>::ABB object.

Template Parameters

Key	

5.2.1.2 ~ABB()

```
template<class Key >
```

ABB< Key >::~ABB

Destroy the ABB<Key>::ABB object.

Template Parameters

Key	

5.2.2 Member Function Documentation

5.2.2.1 insert()

```
template<class Key >
void ABB< Key >::insert (
            const Key & key )  [override], [virtual]
```

Implements AB< Key >.

Reimplemented in AVL< Key >.

5.2.2.2 search()

```
template<class Key >
bool ABB< Key >::search (
            const Key & key ) const  [override], [virtual]
```

Search a node in the branch tree.

Template Parameters

Key	

Parameters

key	

Returns

true

false

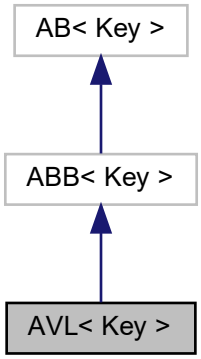
Implements AB< Key >.

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

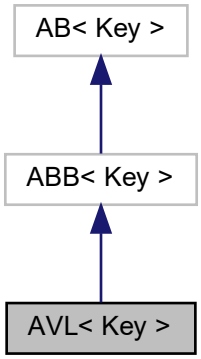
- include/ABB.h

5.3 AVL< Key > Class Template Reference

Inheritance diagram for AVL< Key >:



Collaboration diagram for AVL< Key >:



Public Member Functions

- AVL ()
  Construct a new AVL<Key>::AVL object.
- ~AVL ()
  Destroy the AVL<Key>::AVL object.
- NodeAVL< Key > * getRootAVL () const
  Return the root of the branch tree.
- NodeAVL< Key > *& getRootAVL ()
  Return the root of the branch tree.
- void insert (const Key &k) override
  Insert a node in the branch tree.

Additional Inherited Members

5.3.1 Constructor & Destructor Documentation

5.3.1.1 AVL()

```
template<class Key >
AVL< Key >::AVL
```

Construct a new AVL<Key>::AVL object.

Template Parameters

Key	

5.3.1.2 ~AVL()

```
template<class Key >
AVL< Key >::~AVL
```

Destroy the AVL<Key>::AVL object.

Template Parameters

Key	

5.3.2 Member Function Documentation

5.3.2.1 getRootAVL() [1/2]

```
template<class Key >
NodeAVL< Key > *& AVL< Key >::getRootAVL  [inline]
```

Return the root of the branch tree.

Template Parameters

Key	

Returns

NodeAVL<Key>*

5.3.2.2 getRootAVL() [2/2]

```
template<class Key >
NodeAVL< Key > * AVL< Key >::getRootAVL  [inline]
```

Return the root of the branch tree.

Template Parameters

Key	

Returns

NodeAVL<Key>*

5.3.2.3 insert()

```
template<class Key >
void AVL< Key >::insert (
            const Key & key )  [override], [virtual]
```

Insert a node in the branch tree.

Template Parameters

Key	

Parameters

node	
key	

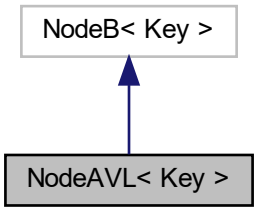
Reimplemented from ABB< Key >.

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

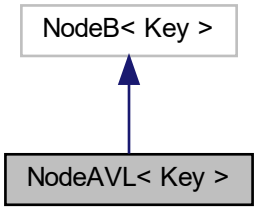
- include/AVL.h

5.4 NodeAVL< Key > Class Template Reference

Inheritance diagram for NodeAVL< Key >:



Collaboration diagram for NodeAVL< Key >:



Public Member Functions

- NodeAVL (const Key &, NodeAVL< Key > *=NULL, NodeAVL< Key > *=NULL)

  Construct a new Node AVL< Key>:: Node AVL object.
- ~NodeAVL ()

  Destroy the Node AVL< Key>:: Node AVL object.
- int getBalanceFactor () const

  Return the balance factor of node.
- int & getBalanceFactor ()

  Return the balance factor of node.
- NodeAVL< Key > * getLeftAVL () const

  Return the left node.
- NodeAVL< Key > *& getLeftAVL ()

  Return the left node.
- NodeAVL< Key > * getRightAVL () const

  Return the right node.
- NodeAVL< Key > *& getRightAVL ()

  Return the right node.

Additional Inherited Members

5.4.1 Constructor & Destructor Documentation

5.4.1.1 NodeAVL()

```
template<class Key >
NodeAVL< Key >::NodeAVL (
            const Key & data,
            NodeAVL< Key > * left = NULL,
            NodeAVL< Key > * right = NULL )
```

Construct a new Node AVL< Key>:: Node AVL object.

Template Parameters

Key	

Parameters

data	
left	
right	

5.4.1.2 ~NodeAVL()

```
template<class Key >
NodeAVL< Key >::~NodeAVL
```

Destroy the Node AVL< Key>:: Node AVL object.

Template Parameters

Key	

5.4.2 Member Function Documentation

5.4.2.1 getBalanceFactor() [1/2]

```
template<class Key >
int & NodeAVL< Key >::getBalanceFactor  [inline]
```

Return the balance factor of node.

Template Parameters

Key	

Returns

int

5.4.2.2 getBalanceFactor() [2/2]

```
template<class Key >
int NodeAVL< Key >::getBalanceFactor  [inline]
```

Return the balance factor of node.

Template Parameters

Key	

Returns

int

5.4.2.3 getLeftAVL() [1/2]

```
template<class Key >
NodeAVL< Key > *& NodeAVL< Key >::getLeftAVL  [inline]
```

Return the left node.

Template Parameters

Key	

Returns

NodeAVL<Key>*

5.4.2.4 getLeftAVL() [2/2]

```
template<class Key >
NodeAVL< Key > * NodeAVL< Key >::getLeftAVL  [inline]
```

Return the left node.

Template Parameters

Key	

Returns

NodeAVL<Key>*

5.4.2.5 getRightAVL() [1/2]

```
template<class Key >
NodeAVL< Key > *& NodeAVL< Key >::getRightAVL  [inline]
```

Return the right node.

Template Parameters

Key	

Returns

NodeAVL<Key>*

5.4.2.6 getRightAVL() [2/2]

```
template<class Key >
NodeAVL< Key > * NodeAVL< Key >::getRightAVL  [inline]
```

Return the right node.

Template Parameters

Key	

Returns

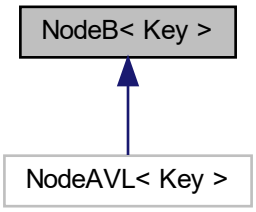
NodeAVL<Key>*

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

- include/NodeAVL.h

5.5 NodeB< Key > Class Template Reference

Inheritance diagram for NodeB< Key >:



Public Member Functions

- NodeB (const Key &, NodeB< Key > *=NULL, NodeB< Key > *=NULL)
  Construct a new Node B< Key>:: Node B object.
- virtual ~NodeB ()
  Destroy the Node B< Key>:: Node B object.
- Key getData () const
  Return the dato of node.
- NodeB< Key > * getLeft () const
  Return the left node.
- NodeB< Key > *& getLeft ()
  Return the left node.
- NodeB< Key > * getRight () const
  Return the right node.
- NodeB< Key > *& getRight ()
  Return the right node.

Protected Attributes

- NodeB< Key > * **left_**
- NodeB< Key > * **right_**

Friends

- template<class T >
  std::ostream & **operator**<< (std::ostream &, const NodeB< T > &)

5.5.1 Constructor & Destructor Documentation

5.5.1.1 NodeB()

```
template<class Key >
NodeB< Key >::NodeB (
            const Key & data,
            NodeB< Key > * left = NULL,
            NodeB< Key > * right = NULL )
```

Construct a new Node B< Key>:: Node B object.

Template Parameters

Key	

Parameters

data	
left	
right	

5.5.1.2 ~NodeB()

```
template<class Key >
NodeB< Key >::~NodeB  [virtual]
```

Destroy the Node B< Key>:: Node B object.

Template Parameters

Key	

5.5.2 Member Function Documentation

5.5.2.1 getData()

```
template<class Key >
Key NodeB< Key >::getData  [inline]
```

Return the dato of node.

Template Parameters

Key	

Returns

Key

5.5.2.2 getLeft() [1/2]

```
template<class Key >
NodeB< Key > *& NodeB< Key >::getLeft  [inline]
```

Return the left node.

Template Parameters

Key	

Returns

NodeB<Key>*&

5.5.2.3 getLeft() [2/2]

```
template<class Key >
NodeB< Key > * NodeB< Key >::getLeft  [inline]
```

Return the left node.

Template Parameters

Key	

Returns

NodeB<Key>*

5.5.2.4 getRight() [1/2]

```
template<class Key >
NodeB< Key > *& NodeB< Key >::getRight  [inline]
```

Return the right node.

Template Parameters

Key	

Returns

NodeB<Key>*&

5.5.2.5 getRight() [2/2]

```
template<class Key >
NodeB< Key > * NodeB< Key >::getRight  [inline]
```

Return the right node.

Template Parameters

Key	

Returns

NodeB<Key>*

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

- include/NodeB.h

Chapter 6

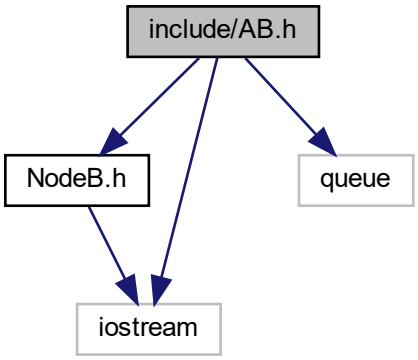
File Documentation

6.1 include/AB.h File Reference

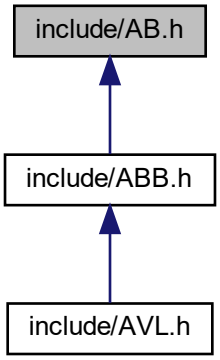
This class is Binary Tree.

```
#include "NodeB.h"
#include <iostream>
#include <queue>
```

Include dependency graph for AB.h:



This graph shows which files directly or indirectly include this file:



Classes

- class AB< Key >

Functions

- template<class Key >
  std::ostream & operator<< (std::ostream &os, const AB< Key > &ab_tree)

  This method print the tree.

6.1.1 Detailed Description

This class is Binary Tree.

Author

Fabrizzio Daniell Perilli Martín alu0101138589@ull.edu.es

Version

0.1

Date

2023-04-23

Copyright

Copyright (c) 2023

6.1.2 Function Documentation

6.1.2.1 operator<<()

```
template<class Key >
std::ostream & operator<< (
             std::ostream & os,
             const AB< Key > & ab_tree )
```

This method print the tree.

Template Parameters

Key	

Parameters

os	
ab_tree	

Returns

std::ostream&

6.2 AB.h

Go to the documentation of this file.

```
1
11 #pragma once
12 #include "NodeB.h"
13 #include <iostream>
14 #include <queue>
15
16 template <class Key>
17 class AB
18 {
19 private:
20   int branchSize_(NodeB<Key> *) const;
21   int heightN_(NodeB<Key> *) const;
22   bool isBranchEquilibrium_(NodeB<Key> *) const;
23   void prune_(NodeB<Key> *&);
24   void route_(NodeB<Key> *) const;
25
26 protected:
27   NodeB<Key> *root_;
28
29 public:
30   AB();
31   virtual ~AB();
32
33   bool isEmpty(NodeB<Key> *);
34   bool isLeafNode(NodeB<Key> *);
35   inline int size() const;
36   inline int height() const;
37   inline bool isEquilibrium() const;
38
39   virtual void insert(const Key &) = 0;
40   virtual bool search(const Key &) const = 0;
41   void inorden() const;
```

```
42
43   template <class T>
44   friend std::ostream &operator«(std::ostream &, const AB<T> &);
45 };
46
52 template <class Key>
53 AB<Key>::AB() : root_(NULL) {}
54
60 template <class Key>
61 AB<Key>::~AB()
62 {
63   prune_(root_);
64 }
65
72 template <class Key>
73 void AB<Key>::prune_(NodeB<Key> *&node)
74 {
75   if (node == NULL)
76     return;
77   prune_(node->getLeft());
78   prune_(node->getRight());
79   delete node;
80   node = NULL;
81 }
82
91 template <class Key>
92 bool AB<Key>::isEmpty(NodeB<Key> *node)
93 {
94   return node == NULL;
95 }
96
105 template <class Key>
106 bool AB<Key>::isLeafNode(NodeB<Key> *node)
107 {
108   return !node->getLeft() && !node->getRight();
109 }
110
118 template <class Key>
119 int AB<Key>::branchSize_(NodeB<Key> *node) const
120 {
121   if (node == NULL)
122     return 0;
123   return (1 + branchSize_(node->getLeft) + branchSize_(node->getRight()));
124 }
125
132 template <class Key>
133 int AB<Key>::size() const
134 {
135   return branchSize_(root_);
136 }
137
145 template <class Key>
146 int AB<Key>::heightN_(NodeB<Key> *node) const
147 {
148   if (node == NULL)
149     return 0;
150   int height_left = heightN_(node->getLeft());
151   int height_right = heightN_(node->getRight());
152
153   if (height_right > height_left)
154     return ++height_right;
155   else
156     return ++height_left;
157 }
158
165 template <class Key>
166 int AB<Key>::height() const
167 {
168   return heightN_(root_);
169 }
170
179 template <class Key>
180 bool AB<Key>::isBranchEquilibrium_(NodeB<Key> *node) const
181 {
182   if (node == NULL)
183     return true;
184   const int eq = branchSize_(node->getLeft()) - branchSize_(node->getRight());
185   switch (eq)
186   {
187   case -1:
188   case 0:
189   case 1:
190     return isBranchEquilibrium_(node->getLeft()) && isBranchEquilibrium_(node->getRight());
191   default:
192     return false;
193   }
194 }
```

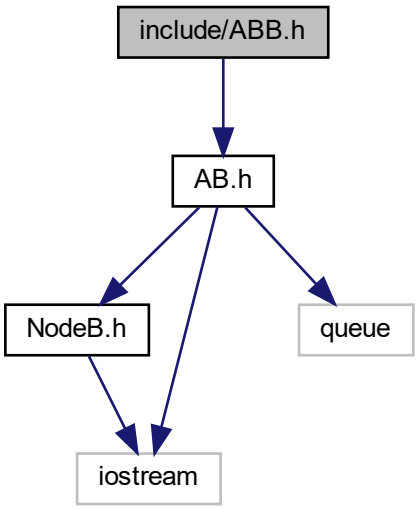
```
195
203 template <class Key>
204 bool AB<Key>::isEquilibrium() const
205 {
206   return isBranchEquilibrium_(root_);
207 }
208
209 template <class Key>
210 void AB<Key>::inorden() const
211 {
212   route_(root_);
213 }
214
215 template <class Key>
216 void AB<Key>::route_(NodeB<Key> *node) const
217 {
218   if (node == NULL)
219     return;
220   route_(node->getLeft());
221   std::cout « node->getData() « " ";
222   route_(node->getRight());
223 }
224
233 template <class Key>
234 std::ostream &operator«(std::ostream &os, const AB<Key> &ab_tree)
235 {
236   std::queue<std::pair<NodeB<Key> *, int» queue;
237   queue.push(std::make_pair(ab_tree.root_, 0));
238   int current_level = 0;
239
240   std::cout « "Level " « current_level « ": ";
241   while (!queue.empty())
242   {
243     std::pair<NodeB<Key> *, int> node_current = queue.front();
244     queue.pop();
245
246     if (node_current.second > current_level)
247     {
248       current_level++;
249       os « "\nLevel " « current_level « ": ";
250     }
251
252     if (node_current.first != NULL)
253     {
254       os « "[" « *node_current.first « "] ";
255       queue.push(std::make_pair(node_current.first->getLeft(), current_level + 1));
256       queue.push(std::make_pair(node_current.first->getRight(), current_level + 1));
257     }
258     else
259     {
260       os « "[.] ";
261     }
262   }
263
264   return os;
265 }
```

6.3 include/ABB.h File Reference

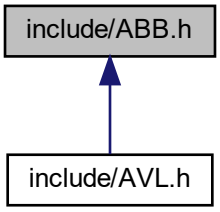
This class content ABB Binary tree search is derivated class of AB.

```
#include "AB.h"
```

Include dependency graph for ABB.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ABB< Key >

6.3.1 Detailed Description

This class content ABB Binary tree search is derivated class of AB.

Author

Fabrizzio Daniell Perilli Martín alu0101138589@ull.edu.es

Version

0.1

Date

2023-04-25

6.4 ABB.h

Go to the documentation of this file.

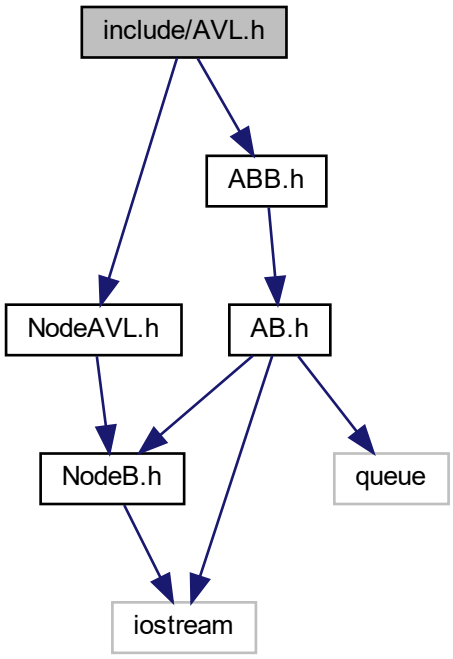
```
1
12 #pragma once
13 #include "AB.h"
14
15 template <class Key>
16 class ABB : public AB<Key>
17 {
18 private:
19   void insertBranch_(NodeB<Key> *&, const Key &);
20   bool searchBranch_(NodeB<Key> *, const Key &) const;
21
22 public:
23   ABB();
24   ~ABB();
25   void insert(const Key &) override;
26   virtual bool search(const Key &) const override;
27 };
28
34 template <class Key>
35 ABB<Key>::ABB() : AB<Key>() {}
36
44 template <class Key>
45 void ABB<Key>::insertBranch_(NodeB<Key> *&node, const Key &key)
46 {
47   if (node == NULL)
48     node = new NodeB<Key>(key);
49   else if (key < node->getData())
50     insertBranch_(node->getLeft(), key);
51   else
52     insertBranch_(node->getRight(), key);
53 }
54
60 template <class Key>
61 ABB<Key>::~ABB() {}
62
63 template <class Key>
64 void ABB<Key>::insert(const Key &key)
65 {
66   insertBranch_(this->root_, key);
67 }
68
78 template <class Key>
79 bool ABB<Key>::searchBranch_(NodeB<Key> *node, const Key &key) const
80 {
81   if (node == NULL)
82     return false;
83   else if (key == node->getData())
84     return true;
85   else if (key < node->getData())
86     return searchBranch_(node->getLeft(), key);
87   else
88     return searchBranch_(node->getRight(), key);
89 }
90
99 template <class Key>
100 bool ABB<Key>::search(const Key &key) const
101 {
102   return searchBranch_(this->root_, key);
103 }
```

6.5 include/AVL.h File Reference

This class content AVL Binary tree search is derivated class of ABB.

```
#include "ABB.h"
#include "NodeAVL.h"
```

Include dependency graph for AVL.h:



Classes

- class AVL< Key >

6.5.1 Detailed Description

This class content AVL Binary tree search is derivated class of ABB.

Author

Fabrizzio Daniell Perilli Martín alu0101138589@ull.edu.es

Version

0.1

Date

2023-04-30

Copyright

6.6 AVL.h

Go to the documentation of this file.

```
1
11 #include "ABB.h"
12 #include "NodeAVL.h"
13
14 template <class Key>
15 class AVL : public ABB<Key>
16 {
17 private:
18   void rotationLL_(NodeAVL<Key> *&);
19   void rotationRR_(NodeAVL<Key> *&);
20   void rotationLR_(NodeAVL<Key> *&);
21   void rotationRL_(NodeAVL<Key> *&);
22
23   void insertBalanceRight_(NodeAVL<Key> *&, bool &);
24   void insertBalanceLeft_(NodeAVL<Key> *&, bool &);
25   void insertBalance_(NodeAVL<Key> *&, NodeAVL<Key> *, bool &);
26
27 public:
28   AVL();
29   ~AVL();
30   inline NodeAVL<Key> *getRootAVL() const;
31   inline NodeAVL<Key> *&getRootAVL();
32
33   void insert(const Key &k) override;
34 };
35
41 template <class Key>
42 AVL<Key>::AVL() : ABB<Key>() {}
43
49 template <class Key>
50 AVL<Key>::~AVL() {}
51
58 template <class Key>
59 NodeAVL<Key> *AVL<Key>::getRootAVL() const
60 {
61   return reinterpret_cast<NodeAVL<Key> *>(this->root_);
62 }
63
70 template <class Key>
71 NodeAVL<Key> *&AVL<Key>::getRootAVL()
72 {
73   return reinterpret_cast<NodeAVL<Key> *&>(this->root_);
74 }
75
83 template <class Key>
84 void AVL<Key>::insert(const Key &key)
85 {
86
87   bool grow = false;
88   insertBalance_(getRootAVL(), new NodeAVL<Key>(key), grow);
89 }
90
97 template <class Key>
98 void AVL<Key>::rotationLL_(NodeAVL<Key> *&node)
99 {
100   #ifdef TRACE
101   std::cout « "\nUnbalance" « std::endl;
102   std::cout « *this « std::endl;
103   std::cout « "\nRotation Left-Left in [" « node->getData() « "]" « std::endl;
104   #endif
105
106   NodeAVL<Key> *node1 = node->getLeftAVL();
107   node->getLeftAVL() = node1->getRightAVL();
108   node1->getRightAVL() = node;
109
110   if (node1->getBalanceFactor() == 1)
111   {
112     node->getBalanceFactor() = 0;
113     node1->getBalanceFactor() = 0;
114   }
115   else
116   {
117     node->getBalanceFactor() = 1;
118     node1->getBalanceFactor() = -1;
119   }
120   node = node1;
121 }
122
129 template <class Key>
130 void AVL<Key>::rotationRR_(NodeAVL<Key> *&node)
131 {
132   #ifdef TRACE
```

```
133   std::cout « "\nUnbalance" « std::endl;
134   std::cout « *this « std::endl;
135   std::cout « "\nRotation Right-Right in [" « node->getData() « "]" « std::endl;
136   #endif
137
138   NodeAVL<Key> *node1 = node->getRightAVL();
139   node->getRightAVL() = node1->getLeftAVL();
140   node1->getLeftAVL() = node;
141
142   if (node1->getBalanceFactor() == -1)
143   {
144     node->getBalanceFactor() = 0;
145     node1->getBalanceFactor() = 0;
146   }
147   else
148   {
149     node->getBalanceFactor() = -1;
150     node1->getBalanceFactor() = 1;
151   }
152   node = node1;
153 }
154
161 template <class Key>
162 void AVL<Key>::rotationLR_(NodeAVL<Key> *&node)
163 {
164   #ifdef TRACE
165   std::cout « "\nUnbalance" « std::endl;
166   std::cout « *this « std::endl;
167   std::cout « "\nRotation Left-Right in [" « node->getData() « "]" « std::endl;
168   #endif
169
170   NodeAVL<Key> *node1 = node->getLeftAVL();
171   NodeAVL<Key> *node2 = node1->getRightAVL();
172   node->getLeftAVL() = node2->getRightAVL();
173   node2->getRightAVL() = node;
174   node1->getRightAVL() = node2->getLeftAVL();
175   node2->getLeftAVL() = node1;
176
177   if (node2->getBalanceFactor() == -1)
178   {
179     node1->getBalanceFactor() = 1;
180   }
181   else
182   {
183     node1->getBalanceFactor() = 0;
184   }
185   if (node2->getBalanceFactor() == 1)
186   {
187     node->getBalanceFactor() = -1;
188   }
189   else
190   {
191     node->getBalanceFactor() = 0;
192   }
193   node2->getBalanceFactor() = 0;
194   node = node2;
195 }
196
203 template <class Key>
204 void AVL<Key>::rotationRL_(NodeAVL<Key> *&node)
205 {
206   #ifdef TRACE
207   std::cout « "\nUnbalance" « std::endl;
208   std::cout « *this « std::endl;
209   std::cout « "\nRotation Right-Left in [" « node->getData() « "]" « std::endl;
210   #endif
211
212   NodeAVL<Key> *node1 = node->getRightAVL();
213   NodeAVL<Key> *node2 = node1->getLeftAVL();
214   node->getRightAVL() = node2->getLeftAVL();
215   node2->getLeftAVL() = node;
216   node1->getLeftAVL() = node2->getRightAVL();
217   node2->getRightAVL() = node1;
218
219   if (node2->getBalanceFactor() == 1)
220   {
221     node1->getBalanceFactor() = -1;
222   }
223   else
224   {
225     node1->getBalanceFactor() = 0;
226   }
227   if (node2->getBalanceFactor() == -1)
228   {
229     node->getBalanceFactor() = 1;
230   }
231   else
```

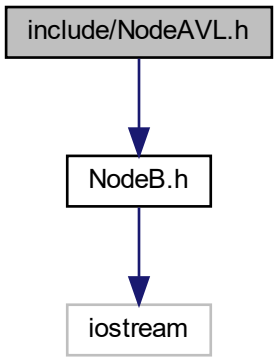
```
232   {
233     node->getBalanceFactor() = 0;
234   }
235   node2->getBalanceFactor() = 0;
236   node = node2;
237 }
238
246 template <class Key>
247 void AVL<Key>::insertBalanceLeft_(NodeAVL<Key> *&node, bool &grow)
248 {
249   switch (node->getBalanceFactor())
250   {
251   case -1:
252     node->getBalanceFactor() = 0;
253     grow = false;
254     break;
255   case 0:
256     node->getBalanceFactor() = 1;
257     break;
258   case 1:
259     NodeAVL<Key> *node1 = node->getLeftAVL();
260     if (node1->getBalanceFactor() == 1)
261       rotationLL_(node);
262     else
263       rotationLR_(node);
264     grow = false;
265     break;
266   }
267 }
268
276 template <class Key>
277 void AVL<Key>::insertBalanceRight_(NodeAVL<Key> *&node, bool &grow)
278 {
279   switch (node->getBalanceFactor())
280   {
281   case 1:
282     node->getBalanceFactor() = 0;
283     grow = false;
284     break;
285   case 0:
286     node->getBalanceFactor() = -1;
287     break;
288   case -1:
289     NodeAVL<Key> *node1 = node->getRightAVL();
290     if (node1->getBalanceFactor() == -1)
291       rotationRR_(node);
292     else
293       rotationRL_(node);
294     grow = false;
295     break;
296   }
297 }
298
307 template <class Key>
308 void AVL<Key>::insertBalance_(NodeAVL<Key> *&node, NodeAVL<Key> *newNode, bool &grow)
309 {
310   if (node == NULL)
311   {
312     node = newNode;
313     grow = true;
314   }
315   if (node->getData() == newNode->getData())
316     return;
317
318   if (node->getData() > newNode->getData())
319   {
320     insertBalance_(node->getLeftAVL(), newNode, grow);
321     if (grow)
322       insertBalanceLeft_(node, grow);
323   }
324   else
325   {
326     insertBalance_(node->getRightAVL(), newNode, grow);
327     if (grow)
328       insertBalanceRight_(node, grow);
329   }
330 }
```

6.7 include/NodeAVL.h File Reference

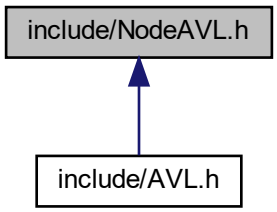
This class content Node AVL Binary tree search is derivated class of NodeB.

```
#include "NodeB.h"
```

Include dependency graph for NodeAVL.h:



This graph shows which files directly or indirectly include this file:



Classes

- class NodeAVL< Key >

6.7.1 Detailed Description

This class content Node AVL Binary tree search is derivated class of NodeB.

Author

Fabrizzio Daniell Perilli Martín alu0101138589@ull.edu.es

Version

0.1

Date

2023-04-30

Copyright

6.8 NodeAVL.h

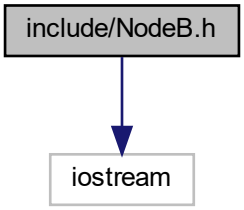
Go to the documentation of this file.

```
1
12 #include "NodeB.h"
13
14 template <class Key>
15 class NodeAVL : public NodeB<Key>
16 {
17 private:
18   int balance_factor_;
19
20 public:
21   NodeAVL(const Key &, NodeAVL<Key> * = NULL, NodeAVL<Key> * = NULL);
22   ~NodeAVL();
23
24   inline int getBalanceFactor() const;
25   inline int &getBalanceFactor();
26
27   inline NodeAVL<Key> *getLeftAVL() const;
28   inline NodeAVL<Key> *&getLeftAVL();
29   inline NodeAVL<Key> *getRightAVL() const;
30   inline NodeAVL<Key> *&getRightAVL();
31 };
32
41 template <class Key>
42 NodeAVL<Key>::NodeAVL(const Key &data, NodeAVL<Key> *left, NodeAVL<Key> *right) : NodeB<Key>(data, left,
       right), balance_factor_(0) {}
43
49 template <class Key>
50 NodeAVL<Key>::~NodeAVL() {}
51
58 template <class Key>
59 int NodeAVL<Key>::getBalanceFactor() const
60 {
61   return balance_factor_;
62 }
63
70 template <class Key>
71 int &NodeAVL<Key>::getBalanceFactor()
72 {
73   return balance_factor_;
74 }
75
82 template <class Key>
83 NodeAVL<Key> *NodeAVL<Key>::getLeftAVL() const
84 {
85   return reinterpret_cast<NodeAVL<Key> *>(this->left_);
86 }
87
94 template <class Key>
95 NodeAVL<Key> *&NodeAVL<Key>::getLeftAVL()
96 {
97   return reinterpret_cast<NodeAVL<Key> *&>(this->left_);
98 }
99
106 template <class Key>
107 NodeAVL<Key> *NodeAVL<Key>::getRightAVL() const
108 {
109   return reinterpret_cast<NodeAVL<Key> *>(this->right_);
110 }
111
118 template <class Key>
119 NodeAVL<Key> *&NodeAVL<Key>::getRightAVL()
120 {
121   return reinterpret_cast<NodeAVL<Key> *&>(this->right_);
122 }
```

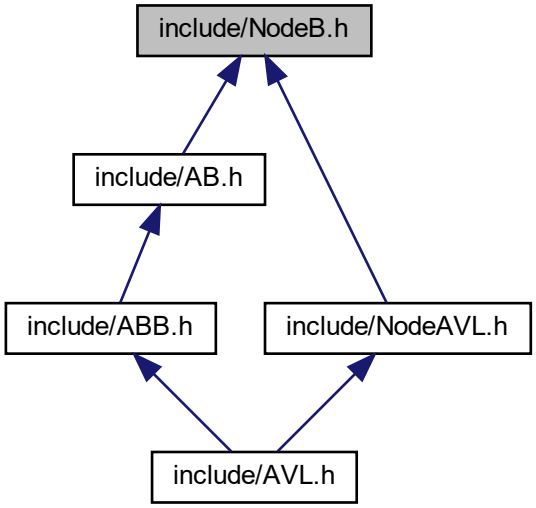
6.9 include/NodeB.h File Reference

This class contains the node of binary tree.

`#include <iostream>`
Include dependency graph for NodeB.h:



This graph shows which files directly or indirectly include this file:



Classes

- class NodeB< Key >

Functions

- template<class Key >
  std::ostream & **operator**<< (std::ostream &os, const NodeB< Key > &node)

6.9.1 Detailed Description

This class contains the node of binary tree.

Author

Fabrizzio Daniell Perilli Martín alu0101138589@ull.edu.es

Version

0.1

Date

2023-04-23

Copyright

Copyright (c) 2023

6.10 NodeB.h

Go to the documentation of this file.

```
1
11 #pragma once
12 #include <iostream>
13
14 template <class Key>
15 class NodeB
16 {
17 private:
18   Key data_;
19
20 protected:
21   NodeB<Key> *left_;
22   NodeB<Key> *right_;
23
24 public:
25   NodeB(const Key &, NodeB<Key> * = NULL, NodeB<Key> * = NULL);
26   virtual ~NodeB();
27
28   inline Key getData() const;
29
30   inline NodeB<Key> *getLeft() const;
31   inline NodeB<Key> *&getLeft();
32
33   inline NodeB<Key> *getRight() const;
34   inline NodeB<Key> *&getRight();
35
36   template <class T>
37   friend std::ostream &operator<<(std::ostream &, const NodeB<T> &);
38 };
39
48 template <class Key>
49 NodeB<Key>::NodeB(const Key &data, NodeB<Key> *left, NodeB<Key> *right) : data_(data), left_(left),
       right_(right) {}
50
56 template <class Key>
57 NodeB<Key>::~NodeB() {}
58
65 template <class Key>
66 Key NodeB<Key>::getData() const
67 {
68   return data_;
69 }
70
77 template <class Key>
78 NodeB<Key> *NodeB<Key>::getLeft() const
```

```
79 {
80   return left_;
81 }
82
89 template <class Key>
90 NodeB<Key> *&NodeB<Key>::getLeft()
91 {
92   return left_;
93 }
94
101 template <class Key>
102 NodeB<Key> *NodeB<Key>::getRight() const
103 {
104   return right_;
105 }
106
113 template <class Key>
114 NodeB<Key> *&NodeB<Key>::getRight()
115 {
116   return right_;
117 }
118
119 template <class Key>
120 std::ostream &operator«(std::ostream &os, const NodeB<Key> &node)
121 {
122   os « node.data_;
123   return os;
124 }
```

Index