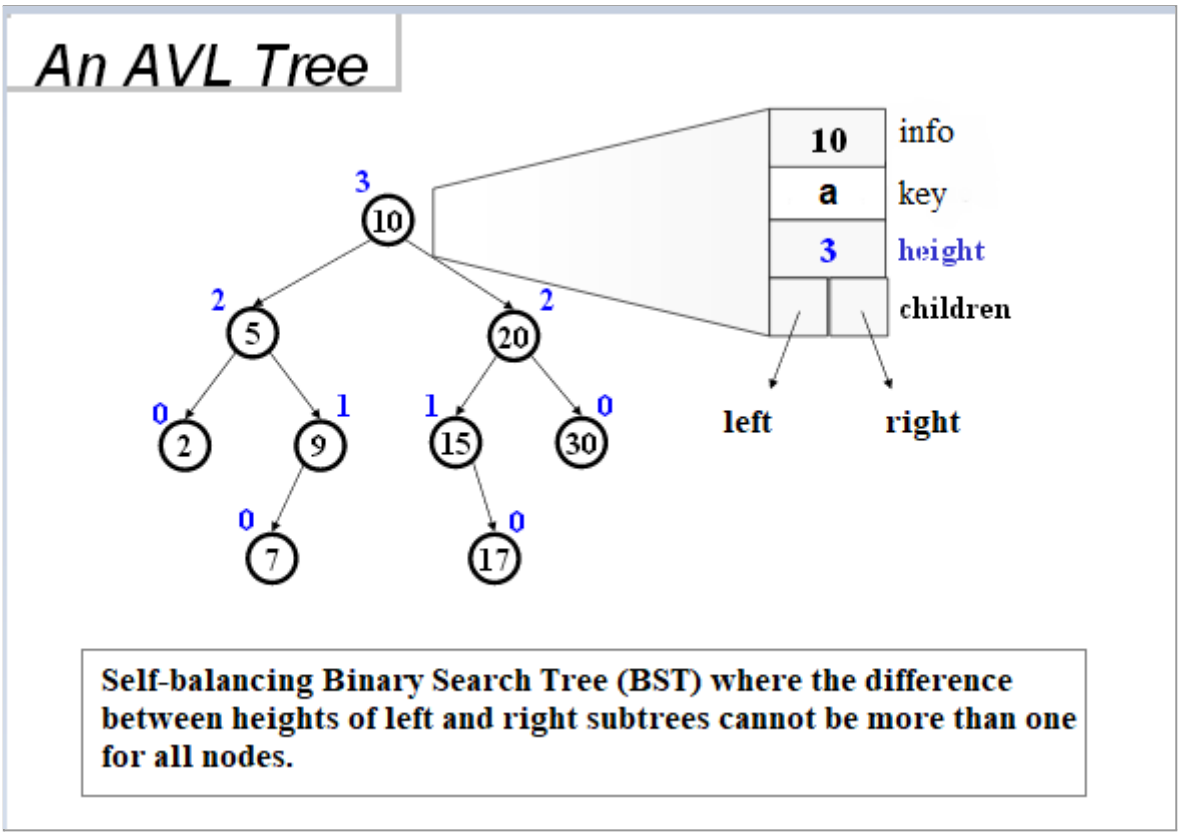
EADS-Lab 3

AVL Tree



Student: Xián García Nogueira K-6077

Lecturer: Turlej Dariusz

16-05-2021

Index:

Description:

This project consists of a ALV tree implemented in c++ composed by nodes that store key and info in alphabetical order.
The source files are the **Dictionary.ccp** and **ring.cpp.** They must be included in the in the desired code being both of them in the project folder as follows:

“#include "Dictionary.cpp"

“#include “ring.cpp"

Also, the header files **Dictionary.h** and **ring.h** with the corresponding definitions.
Then in the **main.cpp** are the tests ,one function for testing and the 2 externals functions for counter and listing.

For initialization a new Dictionary Tree this is the code:

Dictionary<Key, Info> name;

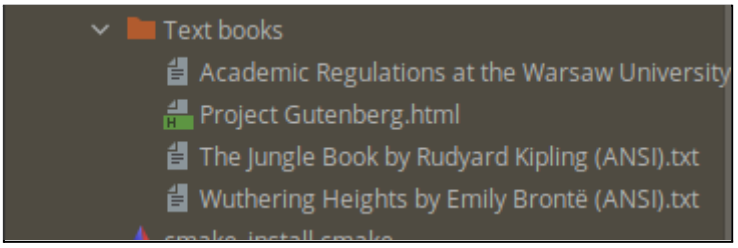
Aim of the Lab:

The aim of the Lab is to implement a full Dictionary Tree with 2 external functions:

- **Counter** function creates an AVL tree and puts each word of the file introduced as a route in each node key of the tree in alphabetical order and sets the info of each node as the count of occurrences of each word.
- **Listing** function creates a doubly linked ring and fill it with the nodes of the ALV tree received in order of first priority the info of the node and when is the same info node, in alphabetical order of the node keys.
- Then **testing** all the methods.

Recommended steps:

In order to execute the program you must include in the build folder, the books that are in the Teams channel, inside the folder “Text books”.



Methods:

Dictionary();
Constructor for the Dictionary

~Dictionary();
Deconstructor for the Dictionary Tree

Dictionary (const Dictionary<Key, Info>&);
Copy constructor, that create a Dictionary Tree initializing it with a deep copy of another Dictionary Tree

Dictionary<Key, Info>& operator= (const Dictionary<Key, Info>&copy);
Assign a copy of a Dictionary Tree to an existing Dictionary Tree.

Node<Key,Info>* copyHelper(const Node<Key,Info> *toCopy);
Method to help copy the Dictionary Tree.

void clear();
Method to help clear the Dictionary Tree.

bool search(const Key &where);
Search for the key in the actual AVL tree.

int height() const;
Returns the height of the actual AVL tree.

void inorder(void callback(const Key &where)) const;
Recursive function that prints the actual AVL tree in Inorder.

void inorderTraversal() const;
Function that prints the actual AVL tree in Inorder.

bool insert(const Key &newKey);
Insert in the actual AVL tree.

bool isEmpty() const;
Returns if the actual Dictionary tree is empty.

int leaves() const;

Returns the leaves of the actual Dictionary tree

int length() const;
Returns the length of the actual Dictionary tree

void preorder(void callback(const Key &where)) const;
Recursive function that prints the actual AVL tree in preorder.

void preOrder2It(Node<Key,Info>* node) const;
Recursive function that prints the key and the info of the actual AVL tree in preorder .

void preOrder2() const;
Function that prints the key and the info of the actual AVL tree in preorder .

void inorder2It(Node<Key,Info>* node) const;
Recursive function that prints the key and the info of the actual AVL tree in inorder .

void inorder2() const;
Function that prints the key and the info of the actual AVL tree in inorder .

void printGraphIt(Node<Key,Info>* node,int ident,vector<int> indexVector)const;
Recursive function that prints the key and the info of the actual AVL tree in graphical form.

void printGraph() const;
Function that prints the key and the info of the actual AVL tree in graphical form.

void preorderTraversal() const;
Function that prints the actual AVL tree.

bool remove(const Key &what);
Function that removes one element by the key in the actual AVL tree.

Node<Key,Info>* getRoot() const;
Returns the root of the actual Dictionary tree

Info getInfo(const Key &where);
Returns the info of the actual Node.

void preORet(Node<Key,Info> *c);
Function that prints the actual AVL tree.

bool operator==(const Dictionary<Key,Info> &other);
Comparison operator

bool operator!=(const Dictionary<Key,Info> &other);
Inverse comparison operator

private methods:

Node<Key, Info> *balance(Node<Key, Info> *&tree);
Balances the AVL tree, if necessary.

Node<Key, Info> *balanceFromLeft(Node<Key, Info> *&tree);
Balances a subtree from the left

Node<Key, Info> *balanceFromRight(Node<Key, Info> *&tree);
Balances a subtree from the right

void clear(Node<Key, Info> *&tree);
Clears a subtree of all its contents

Node<Key, Info> *deleteNode(Node<Key, Info> *&tree);
Deletes a node from a subtree

int difference(const Node<Key, Info> *tree) const;
Calculates the difference between the heights of the left and right subtrees rooted at the given node

Node<Key, Info> *find(Node<Key, Info> *tree, const Key &where) const;
Finds a value in a subtree

Node<Key, Info> *getmax(Node<Key, Info> *&tree);
Finds the node with the maximum value in a subtree

int height(const Node<Key, Info> *tree) const;
Calculates the height of the subtree rooted at the given node

void inorder(Node<Key, Info> *tree, void callback(const Key &where)) const;
Traverses a subtree in inorder and processes the value

Node<Key, Info> *insertIntoAVL(Node<Key, Info> *&tree, Node<Key, Info> *&newNode,bool &isTaller);
Inserts a new node into a subtree

int leaves(const Node<Key, Info> *tree) const;
Counts the number of leaves in a subtree

void preorder(Node<Key, Info> *tree, void callback(const Key &where)) const;
Traverses a subtree in preorder

static void print(const Key &where);
Prints the value to the terminal

Node<Key, Info> *remove(Node<Key, Info> *&tree, const Key &where, bool &flag);
Function that removes one element by the key.

Node<Key, Info> *removemax(Node<Key, Info> *&node);
Removes the node with the maximum value in a subtree

Node<Key, Info> *getNode(Node<Key, Info> *tree, const Key &where);
Returns an item from a subtree

Node<Key, Info> *rotateLeft(Node<Key, Info> *&tree);
Performs a left rotation on a subtree

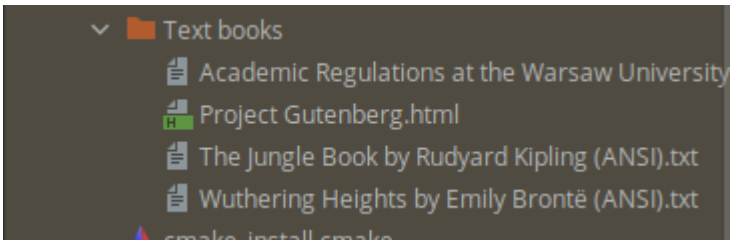
Node<Key, Info> *rotateLeftRight(Node<Key, Info> *&tree);
Performs a left-right rotation on a subtree

Node<Key, Info> *rotateRight(Node<Key, Info> *&tree);
Performs a right rotation on a subtree

Node<Key, Info> *rotateRightLeft(Node<Key, Info> *&tree);
Performs a right-left rotation on a subtree

Testing:

The testing methods are implemented in the main.cpp.
To test all the given methods, you must include in the build folder, the books that are in the Teams channel, inside the folder “Text books” and uncomment in the main, the line 29.



To test with an easier form, you may uncomment the line 30. and the all go automatically

Constructors tests:

Test the constructors and operators, this is the output that checks that all tested are correct.

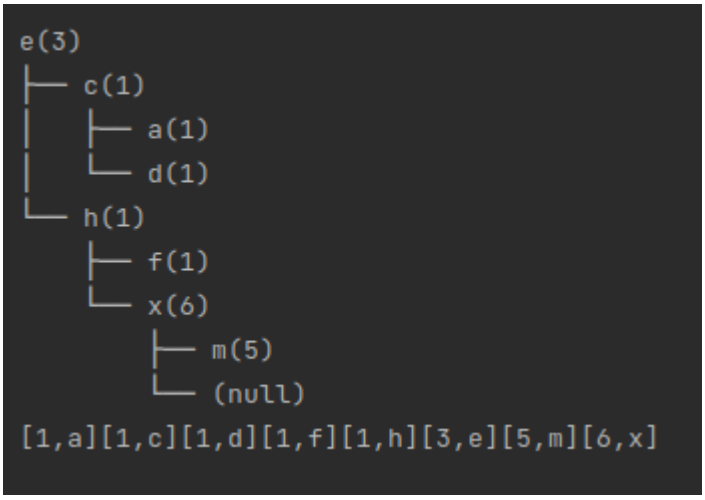
Output:

```
Empty tree: PASSED

Now testing the copy constructor.
-- First with an initializing declaration: Dictionary<int> copy = intDictionary;
-- Inorder traversal of copy:
2

-- Now by passing intDictionary to a value parameter:

--Check that original Dictionary hasn't been changed.
-- Inorder traversal of original:
2

Now testing the assignment constructor with the statement:

-- Inorder traversal of intDictionary:
2

-- Inorder traversal of anotherDictionary:
2

-- Inorder traversal of copy:
2
 Now testing self-assignment with: copy = copy;


-- Inorder traversal of copy:
2
```

Insert tests:

Test the insert, the graph form and the ring created with the external funcition, to know that is correct .

Output:



This is an example execution of the insertTests that you can try. I used this test to know that the insert, inorder,preorder, remove, and search methods are correct.

```
Constructing empty Dictionary
Dictionary is empty
Inorder Traversal of Dictionary:

Now insert a bunch of integers into the Dictionary.
Try items not in the Dictionary and some that are in it:
Item to insert (-999 to stop): 5
Item to insert (-999 to stop): 4
Item to insert (-999 to stop): 5
Item to insert (-999 to stop): 1
Item to insert (-999 to stop): -999
Dictionary is not empty
Inorder Traversal of Dictionary:
1 4 5


Now testing the search() operation.
Try both items in the Dictionary and some not in it:
Item to find (-999 to stop): 4
Found
Item to find (-999 to stop): 0
Not found
Item to find (-999 to stop): -999

Now testing the remove() operation.
Try both items in the Dictionary and some not in it:
Item to remove (-999 to stop): 1
Item to remove (-999 to stop): -999

Inorder Traversal of Dictionary:
4 5
[4,1] [5,2] 4 5

Inorder Traversal of Dictionary:
4 5
Preorder Traversal of Dictionary:
4 5
Postorder Traversal of Dictionary:
4 5
```

External counter and listing functions tests:

I created funcions like

-testRingWithBook();
-testDiccionaryAllBooks();

To prove that the functions work properly. But is hard to insert a good screenshot that prove it, so I attach a screenshot with the firsts and last elements in the ring, for the book "*Academic Regulations at the Warsaw University of Technology (ANSI).txt*"

First words on the Ring:

```
[1,Academy][1,Administrative][1,Affairs][1,After][1,Agreements][1,Attendance][1,Award][1,Bachelors]
```

Last words on the Ring:

```
[202,study][204,be][233,shall][242,in][243,and][262,to][269,a][523,of][1086,the]
```

Last words in the Right bottom of the tree:

First words in the Left top of the tree: