C++ Final Exercise

Generated by Doxygen 1.8.16

Chapter 1

C++ Exam: Binary Tree

Jesus Espinoza and Federico Barone

For this exercise we were asked to implement a binary tree class.
`Main.cpp` contains the implementation of the class as well as a few simple tests to check the different methods of the class and a benchmark of its performance (how the find method performs).
Full `doxygen` documentation of the code is available in the documentation folder. (Check `index.html`).
Running either the tests or the benchmark can be done through the `Makefile` . To run the test execute the following command:

```
$ make test
```

To run the benchmark and plot the results execute the following commands:

```
$ make benchmark
$ make plot
```

In order to check for memory leaks with `valgrind`, run:

```
$ make debug
```

Finally, In order to clean up the directory from binaries, generated benchmark data and images, run:

```
$ make clean
```

Figure 1 shows the results of the benchmark. We compared the performance of the following data structures:

- A linked list (actually it is our binary tree with nodes inserted in ascending order).
- A balanced binary tree (the result of applying the the function `balance()` to the previous structure).
- A std::map from the standard library.

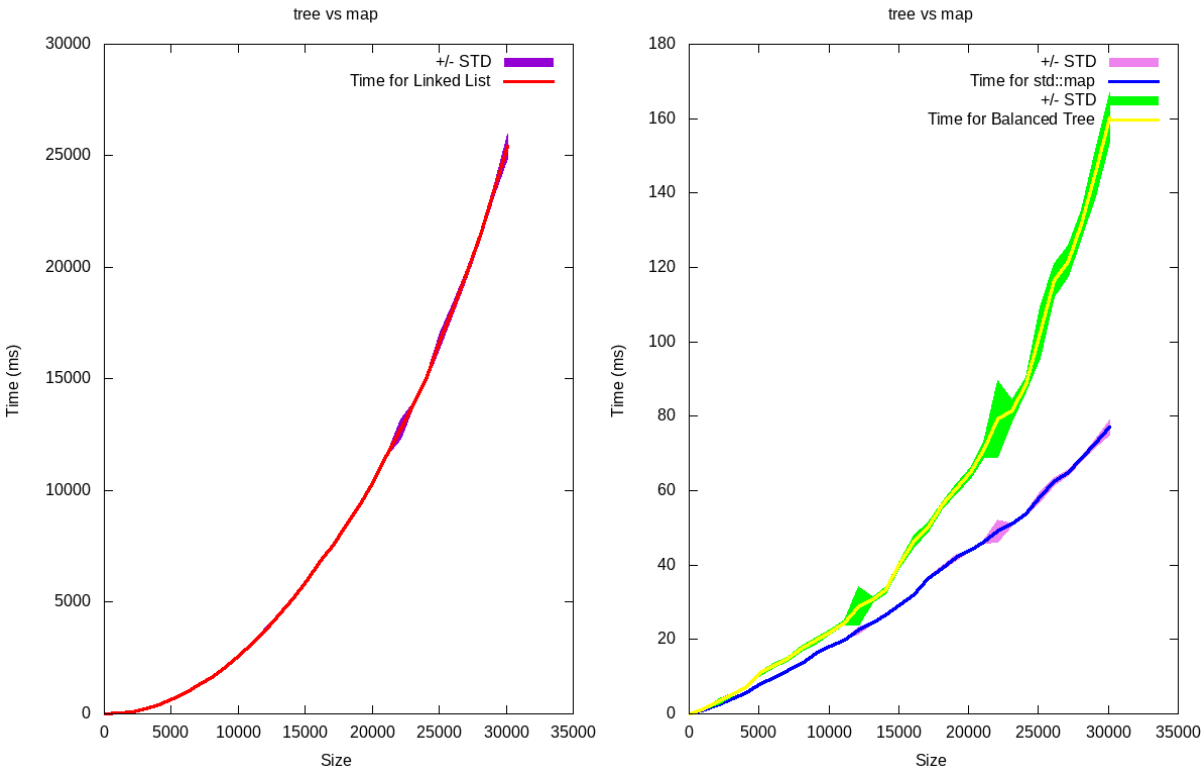


Figure 1.1 Figure 1: Benchmark results for the three different data types

The figure shows the search times for the 3 structures, depending on the number of nodes (The number of nodes is varied from 100 to 30000). All structures contain the same data nodes and all look for the same random keys (the wall time is used as the seed of random numbers, so the keys to search are always different in each run). To reduce the statistical errors associated with time fluctuations, two levels of repetitions are used: a first series within the program that adds up the required search times for a set of common keys for the three structures inside a for-loop, and a second series of repetitions made by a bash script `runner.sh` that repeats the measurements of the previous times for different sizes of the structures. The appropriate calculations of averages and standard deviations are carried out by that same script. As can be seen, the search times of the linked list are considerably higher.

The difference in time between the balanced binary tree and std :: map are similar for low numbers of nodes, however std :: map ends up having a clearly superior performance for larger structures.

Lastly, some comments about the implementation of two important functions of the class: `insert()` and `find()`. The core of both functions is another function, `pos_find(TK& key)`, which works as follows. It takes a key as an argument and returns a pointer to the node which matches that key. If the key is not present in the `BinaryTree` then it returns a pointer to the node directly above this missing key. By doing this, it simplifies the implementation of `insert()` and `find()`, which with almost all the work done for them, they just need a simple if condition to return what they are suppose to.

Extra comment: Although optional, we implemented the `operator[]` in both of its versions. Both of them work, but for the const version we decided to throw an exception if the key you are looking is not in the tree. We are not sure if this is the best implementation.

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

Class Documentation

4.1 BinaryTree< TK, TV > Class Template Reference

A binary tree class to store data.

Classes

- class ConstIterator
- class Iterator
- struct Node

  A struct that contains the data of each node in the BinaryTree.

Public Member Functions

- BinaryTree ()
- BinaryTree (const BinaryTree &)
- BinaryTree & operator= (const BinaryTree &v)
- BinaryTree (BinaryTree &&bt) noexcept
- BinaryTree & operator= (BinaryTree &&bt) noexcept
- const TV & operator[ ] (const TK &key) const
- TV & operator[ ] (const TK &key)
- Iterator begin ()
- Iterator **end** ()
- ConstIterator begin () const
- ConstIterator **end** () const
- ConstIterator cbegin ()
- ConstIterator **cend** ()
- ConstIterator cbegin () const
- ConstIterator **cend** () const
- Node * goLeft () const
- Node * insert (const std::pair< TK, TV > &pair)
- Iterator find (const TK &key) const
- Node * pos_find (const TK &key) const
- void clear ()
- std::size_t checkSize () const
- void copy_node (const Node *np)
- void balance (BinaryTree &balanceTree, Iterator begin, std::size_t locSize)
- void **dummy_func** (Iterator it)

Private Attributes

- std::unique_ptr< Node > root
- std::size_t treeSize

Friends

- template<class otk , class otv >
  std::ostream & **operator**<< (std::ostream &, const BinaryTree< otk, otv > &)

4.1.1 Detailed Description

template<class TK, class TV>
class BinaryTree< TK, TV >

A binary tree class to store data.
The class is templated in both the Key and the Value. At the moment it only handles key types which have operations: ==, <, > defined.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 BinaryTree() [1/3]

```
template<class TK , class TV >
BinaryTree< TK, TV >::BinaryTree ( )  [inline]
```

Default constructor. Constructs and empty binary tree. treeSize is set to zero.

4.1.2.2 BinaryTree() [2/3]

```
template<class TK , class TV >
BinaryTree< TK, TV >::BinaryTree (
            const BinaryTree< TK, TV > & bt )
```

Copy constructor.

4.1.2.3 BinaryTree() [3/3]

```
template<class TK , class TV >
BinaryTree< TK, TV >::BinaryTree (
            BinaryTree< TK, TV > && bt )  [noexcept]
```

Move constructor.

4.1.3 Member Function Documentation

4.1.3.1 balance()

```
template<class TK , class TV >
void BinaryTree< TK, TV >::balance (
            BinaryTree< TK, TV > & balanceTree,
            Iterator begin,
            std::size_t locSize )
```

balance the tree The balance is not in place.

4.1.3.2 begin() [1/2]

```
template<class TK , class TV >
Iterator BinaryTree< TK, TV >::begin ( )  [inline]
```

Used when the user wants to change the value of a Tree using iterators.

4.1.3.3 begin() [2/2]

```
template<class TK , class TV >
ConstIterator BinaryTree< TK, TV >::begin ( ) const  [inline]
```

Used so that the user cannot change the state of a Tree using a reference call to a const Tree.

4.1.3.4 cbegin() [1/2]

```
template<class TK , class TV >
ConstIterator BinaryTree< TK, TV >::cbegin ( )  [inline]
```

Used if we don´t want to change the state of a tree, but we are not using a const Tree.

4.1.3.5 cbegin() [2/2]

```
template<class TK , class TV >
ConstIterator BinaryTree< TK, TV >::cbegin ( ) const  [inline]
```

To use if the user calls cbegin on a const Tree. (we allow the user to be lazy and not think about const Tree vs. noConst Tree)

4.1.3.6 checkSize()

```
template<class TK , class TV >
std::size_t BinaryTree< TK, TV >::checkSize ( ) const  [inline]
```

Retrieve tree size

4.1.3.7 clear()

```
template<class TK , class TV >
void BinaryTree< TK, TV >::clear ( )  [inline]
```

Remove ALL nodes from tree

4.1.3.8 copy_node()

```
template<class TK , class TV >
void BinaryTree< TK, TV >::copy_node (
            const Node * np )
```

auxilary function of the copy constructor

4.1.3.9 find()

```
template<class TK , class TV >
BinaryTree< TK, TV >::Iterator BinaryTree< TK, TV >::find (
            const TK & key ) const
```

Find if a given key exists in the tree. Returns an Iterator to the Node.

4.1.3.10 goLeft()

```
template<class TK , class TV >
Node* BinaryTree< TK, TV >::goLeft ( ) const  [inline]
```

Auxilary function to begin() / end() / cbegin() / cend() It finds the left most element of the tree (smaller key).

4.1.3.11 insert()

```
template<class TK , class TV >
BinaryTree< TK, TV >::Node * BinaryTree< TK, TV >::insert (
            const std::pair< TK, TV > & pair )
```

Insert new node to the BinaryTree.

4.1.3.12 operator=() [1/2]

```
template<class TK , class TV >
BinaryTree< TK, TV > & BinaryTree< TK, TV >::operator= (
            const BinaryTree< TK, TV > & v )
```

Copy assignment.

4.1.3.13 operator=() [2/2]

```
template<class TK , class TV >
BinaryTree< TK, TV > & BinaryTree< TK, TV >::operator= (
            BinaryTree< TK, TV > && bt )  [noexcept]
```

Move assignment.

4.1.3.14 operator[]() [1/2]

```
template<class TK , class TV >
const TV & BinaryTree< TK, TV >::operator[] (
            const TK & key ) const
```

Returns a const reference to the value associated with the key. If the key is not present, throws an exception.

4.1.3.15 operator[]() [2/2]

```
template<class TK , class TV >
TV & BinaryTree< TK, TV >::operator[] (
            const TK & key )
```

Returns a reference to the value associated with the key. If the key is not present, it inserts the key with TTV{}.

4.1.3.16 pos_find()

```
template<class TK , class TV >
BinaryTree< TK, TV >::Node * BinaryTree< TK, TV >::pos_find (
            const TK & key ) const
```

auxiliary function for find() and insert()

4.1.4 Member Data Documentation

4.1.4.1 root

```
template<class TK , class TV >
std::unique_ptr<Node> BinaryTree< TK, TV >::root  [private]
```

pointer to root node.

4.1.4.2 treeSize

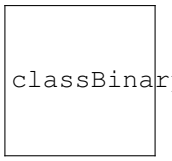
```
template<class TK , class TV >
std::size_t BinaryTree< TK, TV >::treeSize  [private]
```

stores the numbers of nodes in the list.
The documentation for this class was generated from the following file:

- /home/fede/Documents/mhpc/P1.3_exam/c++/main.cc

4.2 BinaryTree< TK, TV >::ConstIterator Class Reference

Inheritance diagram for BinaryTree< TK, TV >::ConstIterator:



Public Types

- using **parent** = BinaryTree< TK, TV >::Iterator

Public Member Functions

- const std::pair< TK, TV > & operator * () const
- **Iterator** (Node *n)

4.2.1 Member Function Documentation

4.2.1.1 operator *()

```
template<class TK , class TV >
const std::pair<TK,TV>& BinaryTree< TK, TV >::ConstIterator::operator * ( ) const  [inline]
```

Returns a const std::pair keyVal of the node by reference
The documentation for this class was generated from the following file:

- /home/fede/Documents/mhpc/P1.3_exam/c++/main.cc

4.3 BinaryTree< TK, TV >::Iterator Class Reference

Inheritance diagram for BinaryTree< TK, TV >::Iterator:

Public Member Functions

- **Iterator** (Node *n)
- std::pair< TK, TV > & operator * () const
- bool **operator==** (const Iterator &other)
- bool **operator!=** (const Iterator &other)
- Iterator & operator++ ()

Private Types

- using **Node** = BinaryTree< TK, TV >::Node

Private Attributes

- Node * **current**

4.3.1 Member Function Documentation

4.3.1.1 operator ∗()

```
template<class TK , class TV >
std::pair<TK,TV>& BinaryTree< TK, TV >::Iterator::operator * ( ) const  [inline]
```

Returns the std::pair keyVal of the node by reference

4.3.1.2 operator++()

```
template<class TK , class TV >
Iterator& BinaryTree< TK, TV >::Iterator::operator++ ( )  [inline]
```

It points the iterator to the next node in the tree asuming ascending key order.
The documentation for this class was generated from the following file:

- /home/fede/Documents/mhpc/P1.3_exam/c++/main.cc

4.4 BinaryTree< TK, TV >::Node Struct Reference

A struct that contains the data of each node in the BinaryTree.

Public Member Functions

- Node (const TK &key, const TV &val, Node ∗s, Node ∗l, Node ∗r)
- Node (const std::pair< TK, TV > kV, Node ∗s, Node ∗l, Node ∗r)
- Node ()=default
- ∼Node ()
- void **print_node** ()

Public Attributes

- std::pair< TK, TV > keyVal
- Node ∗ ppNode
- std::unique_ptr< Node > left
- std::unique_ptr< Node > right

4.4.1 Detailed Description

template<class TK, class TV>
struct BinaryTree< TK, TV >::Node

A struct that contains the data of each node in the BinaryTree.
We choose a struct so that elements its members are visible from the container class.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Node() [1/3]

```
template<class TK , class TV >
BinaryTree< TK, TV >::Node::Node (
            const TK & key,
            const TV & val,
            Node * s,
            Node * l,
            Node * r )  [inline]
```

Constructor that takes key and value separately.

4.4.2.2 Node() [2/3]

```
template<class TK , class TV >
BinaryTree< TK, TV >::Node::Node (
            const std::pair< TK, TV > kV,
            Node * s,
            Node * l,
            Node * r )  [inline]
```

Constructor that takes key and value as std::pair

4.4.2.3 Node() [3/3]

```
template<class TK , class TV >
BinaryTree< TK, TV >::Node::Node ( )  [default]
```

Default Constructor

4.4.2.4 ~Node()

```
template<class TK , class TV >
BinaryTree< TK, TV >::Node::~Node ( )  [inline]
```

Default Destructor

4.4.3 Member Data Documentation

4.4.3.1 keyVal

```
template<class TK , class TV >
std::pair<TK,TV> BinaryTree< TK, TV >::Node::keyVal
```

stores the key and value of the node.

4.4.3.2 left

```
template<class TK , class TV >
std::unique_ptr<Node> BinaryTree< TK, TV >::Node::left
```

Unique pointer to the left node from *this node

4.4.3.3 ppNode

```
template<class TK , class TV >
Node* BinaryTree< TK, TV >::Node::ppNode
```

Pointer to the proper parent of the node. This is, the node which key is next in the tree (Increasing key order)

4.4.3.4 right

```
template<class TK , class TV >
std::unique_ptr<Node> BinaryTree< TK, TV >::Node::right
```

Unique pointer to the right node from *this node

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

- /home/fede/Documents/mhpc/P1.3_exam/c++/main.cc