# Dictionary template class

## Documentation

**Author** : Wiktor Łazarski

**Index number** : 281875

**Field of study** : Computer Science

**Faculty** : Electronics and Information Technology, Warsaw University of Technology

# Table of content

# 1.1 *Dictionary* template overview

*Dictionary* class was implemented and compiled by MinGW for Windows compiler in CodeBlocks. Flags that were used during compilation are as follow :

```
mingw32-g++.exe -Wall -fexceptions -O2 -std=c++11
```

C++11 standard is highly recommended for proper use of *Dictionary* class template.

Link to full project : https://github.com/Wiktos/Dictionary

*Dictionary* is a class template implemented as AVLTree, abstract data structure. It supports logarithms complexity of binary search algorithm. Template class code is available in *dictionary.h* file. Implementation of all methods and *DictionaryException* class are available in *dictionary.tpp*.

# 1.2 Template parameters

```
template <typename K, typename V>
class Dictionary
```

| | |
|---|---|
| K | Typename of key by which *Node* will be recognized in *Dictionary*. |
| I | Typename of data stored in particular *Node*. |

# 1.3 Member types

Private member types :

| Member type | Definition |
|---|---|
| `struct Node` | *Node* is a structure that contain Key value, Info value and pointer to left and right *Node* in tree. |
| `Node *root` | *Node* pointer which hold root of *Dictionary*. |

Public member types :

| Member type | Definition |
|---|---|
| `key_type` | Type of key holded in *Node* |
| `value_type` | Type of value holded in *Node* |
| `class DictionaryException` | *DictionaryException* is an exception class. *DictionaryException* exceptions are thrown where member function is called with invalid arguments. |

# 1.4 Member functions

## Public member functions :

### Constructors :

```
Dictionary() noexcept;
```

```
Dictionary(const Dictionary& source);
```

```
Dictionary(Dictionary&& source) noexcept;
```

```
Dictionary(std::initializer_list<std::pair<key_type, value_type>> ls);
```

( All constructor are defined and implemented in header file )

### Operators :

```
Dictionary& operator=(const Dictionary& rhs);
```

```
Dictionary& operator=(Dictionary&& rhs);
```

### Capacity :

```
bool is_empty() const noexcept;
```

```
int height() const noexcept;
```

### Element access :

```
key_type& get_max();
```

```
const key_type& get_max() const;
```

```
key_type& get_min();
```

```
const key_type& get_min() const;
```

```
value_type& get_value(const key_type& key);
```

```
const value_type& get_value(const key_type& key) const;
```

### Modifiers :

```
void insert(const key_type& new_key, const value_type& new_value);
```

```
void clear();
```

```
void remove(const key_type& key);
```

# 1.4 Member functions

**Operations:**

```
void graph(std::ostream& os) const;
void print_inorder(std::ostream& os) const;
void print_preorder(std::ostream& os) const;
void print_postorder(std::ostream& os) const;
bool contain(const key_type& key) const;
```

**Destructor:**

```
~Dictionary();
```

## Private member functions :

**Element access :**

```
key_type& get_max(Node *start) const;
key_type& get_min(Node *start) const;
Node* get_node(const key_type& key, Node *start) const;
```

**Modifiers :**

```
Node* insert(const key_type& key, const value_type& val, Node *start);
void clear(Node *start);
Node* remove(const key_type& key, Node* start);
```

**Operations:**

```
void graph(std::ostream& os, int width, Node *start) const;
void inorder(std::ostream& os, Node *start) const;
void preorder(std::ostream& os, Node *start) const;
void postorder(std::ostream& os, Node *start) const;
bool contain(const key_type& key, Node *start) const;
Node* copy(Node *start);
```

# 1.4 Member functions

### Tree balance

```
Node* lrotation(Node *node) noexcept;
```

```
Node* rrotation(Node *node) noexcept;
```

```
Node* llrotation(Node *node) noexcept;
```

```
Node* lrrotation(Node *node) noexcept;
```

```
Node* rrrotation(Node *node) noexcept;
```

```
Node* rlrotation(Node *node) noexcept;
```

Most of public function simply call private versions. The reason for this solution is that I wanted to hide passing *Node* structure object as an parameter of public function.

# 2.1 Public member functions

## 2.1.1 Standard member functions

| **Default constructor** |
|---|
| `Dictionary() noexcept;` |
| **Parameters :** none |
| **Complexity :** constant $O(1)$ |
| **Exception :** exception safe |
| **Notes :** Assign *root* equals *nullptr*. |

| **Copy constructor** |
|---|
| `Dictionary(const Dictionary& source);` |
| **Parameters :** source – constant reference to *Dictionary* copy pattern |
| **Complexity :** linear $O(2^n)$ |
| **Exception :** *std::bad_alloc* may be thrown |
| **Notes :** Copy constructor call *copy* private method. |

| **Move constructor** |
|---|
| `Dictionary(Dictionary&& source) noexcept;` |
| **Parameters :** source – reference to *Dictionary* that will be moved to current created object |
| **Complexity :** constant $O(1)$ |
| **Exception :** exception safe |
| **Notes :** Set source *root* equals *nullptr*. |

# 2.1 Public member functions

| **Constructor with *std::initializer_list*** |
|---|
| `Dictionary(std::initializer_list<std::pair<key_type, value_type>> ls);` |
| **Parameters :** ls – *std::initializer_list* object containing *std::pair*s of keys and infos that will be stored in *Dictionary* |
| **Complexity :** linear $O(1ogn)$ |
| **Exception :** *std::bad_alloc* may be thrown |
| **Notes :** none |

| **Destructor** |
|---|
| `~Dictionary();` |
| **Parameters :** none |
| **Complexity :** linear $O(2^n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** Call *clear* funtion that delete all allocated memory. |

## 2.1.2 Operators

| **Copy assigment operator =** |
|---|
| `Dictionary& operator=(const Dictionary& rhs);` |
| **Parameters :** rhs – constant reference to *Dictionary* copy pattern |
| **Complexity :** linear $O(2^n)$ |
| **Exception :** *std::bad_alloc* may be thrown |
| **Notes :** Return reference to *this* . |

# 2.1 Public member functions

| **Move assigment operator =** |
|---|
| `Dictionary& operator=(Dictionary&& rhs);` |
| **Parameters :** rhs – reference to *Ring* that will be moved to our object |
| **Complexity :** constant $O(1)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** Set rhs *Dictionary root* as *nullptr*. |

## 2.1.3 Capacity

| **Check if *Dictionary* is empty** |
|---|
| `bool is_empty() const noexcept;` |
| **Parameters :** none |
| **Complexity :** constant $O(1)$ |
| **Exception :** exception safe |
| **Notes :** Check if *root* == *nullptr*. |

| **Return *Dictionary's* height** |
|---|
| `int height() const noexcept;` |
| **Parameters :** none |
| **Complexity :** constant $O(1)$ |
| **Exception :** exception safe |
| **Notes :** Return current value of *root->heigth* or 0 if *root == nullptr*. |

# 2.1 Public member functions

## 2.1.4 Element access

| **Get *Dictionary* object max *Key* value** |
|---|
| `key_type& get_max();` |
| **Parameters :** none |
| **Complexity :** constant $O(\log n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** none |

| **Get *const Dictionary* object max *Key* value** |
|---|
| `const key_type& get_max() const;` |
| **Parameters :** none |
| **Complexity :** constant $O(\log n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** none |

| **Get *Dictionary* object min *Key* value** |
|---|
| `key_type& get_min();` |
| **Parameters :** none |
| **Complexity :** constant $O(\log n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** none |

| **Get *const Dictionary* object min *Key* value** |
|---|
| `const key_type& get_min() const;` |
| **Parameters :** none |
| **Complexity :** constant $O(\log n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** none |

# 2.1 Public member functions

| **Get *Value* of given *Key*** |
|---|
| `value_type& get_value(const key_type& key);` |
| **Parameters :** none |
| **Complexity :** constant $O(\log n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** none |

| **Get *Value* of given *Key* for *const Dictionary* objects** |
|---|
| `const value_type& get_value(const key_type& key) const;` |
| **Parameters :** none |
| **Complexity :** constant $O(\log n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** none |

## 2.1.5 Modifiers

| **Insert new *Node* to the *Dictionary*** |
|---|
| `void insert(const key_type& new_key, const value_type& new_value);` |
| **Parameters :** new_key – element's key that will be stored in new *Node*<br>New_value – element's info that will be stored in new *Node* |
| **Complexity :** constant $O(\log n)$ |
| **Exception :** *DictionaryException* is thrown if *new_key* already exist in *Dictionary*. |
| **Notes :** none |

# 2.1 Public member functions

| **Clear *Dictionary*** |
|---|
| `void clear();` |
| **Parameters :** none |
| **Complexity :** constant $O(2^n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** clearing is pernament |

| **Removing *Node* with a given *key* from *Dictionary*** |
|---|
| `void remove(const key_type& key);` |
| **Parameters :** key – key of *Node* that will be removed |
| **Complexity :** constant $O(\log n)$ |
| **Exception :** *DictionaryException* is thrown if *key* does not exist.<br>*std::bad_alloc* may be thrown |
| **Notes :** none |

## 2.1.6 Operations

| **Graph *Dictionary*** |
|---|
| `void graph(std::ostream& os) const;` |
| **Parameters :** os – output stream which will contain graphed *Dictionary*. |
| **Complexity :** linear $O(2^n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** none |

# 2.1 Public member functions

| **Print *Dictionary* inorder** |
|---|
| `void print_inorder(std::ostream& os) const;` |
| **Parameters :** os – output stream which will contain printed *Dictionary*. |
| **Complexity :** linear $O(2^n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** none |

| **Print *Dictionary* preorder** |
|---|
| `void print_preorder(std::ostream& os) const;` |
| **Parameters :** os – output stream which will contain printed *Dictionary*. |
| **Complexity :** linear $O(2^n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** none |

| **Print *Dictionary* postorder** |
|---|
| `void print_postorder(std::ostream& os) const;` |
| **Parameters :** os – output stream which will contain printed *Dictionary*. |
| **Complexity :** linear $O(2^n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** none |

| **Check if *Dictionary* contain *Node* with a given *key*** |
|---|
| `bool contain(const key_type& key) const;` |
| **Parameters :** key – key that will be looked in *Dictionary*. |
| **Complexity :** linear $O(\log n)$ |
| **Exception :** too deep recursion may terminate your program. |
| **Notes :** none |

# 2.2 Private member functions

All overloaded methods in private section are called in public overloaded methods. They contain the hole buissness login of AVLTree data structure. Those functions are recursive and may terminate your program if recursion will be to deep.

## Tree balance methods :

| **Left rotation** |
|---|
| `Node* lrotation(Node *node) noexcept;` |
| **Parameters :** node – *Node* on which rotation will be performer. |
| **Complexity :** linear $O(1)$ |
| **Exception :** exception safe |
| **Notes :** perform single left rotation on *Node*. |

| **Right rotation** |
|---|
| `Node* rrotation(Node *node) noexcept;` |
| **Parameters :** node – *Node* on which rotation will be performer. |
| **Complexity :** linear $O(1)$ |
| **Exception :** exception safe |
| **Notes :** perform single right rotation on *Node*. |

| **Left-left rotation** |
|---|
| `Node* llrotation(Node *node) noexcept;` |
| **Parameters :** node – *Node* on which rotation will be performer. |
| **Complexity :** linear $O(1)$ |
| **Exception :** exception safe |
| **Notes :** rotation on *Node* which is performed if *Node's bf == -2* and *Node* on the left site has *bf <= 0.* |

# 2.2 Private member functions

| **Left-right rotation** |
|---|
| `Node* lrrotation(Node *node) noexcept;` |
| **Parameters :** node – *Node* on which rotation will be performer. |
| **Complexity :** linear $O(1)$ |
| **Exception :** exception safe |
| **Notes :** rotation on *Node* which is performed if *Node's bf == -2* and *Node* on the left site has *bf > 0.* |

| **Right-right rotation** |
|---|
| `Node* rrrotation(Node *node) noexcept;` |
| **Parameters :** node – *Node* on which rotation will be performer. |
| **Complexity :** linear $O(1)$ |
| **Exception :** exception safe |
| **Notes :** rotation on *Node* which is performed if *Node's bf == 2* and *Node* on the right site has *bf >= 0.* |

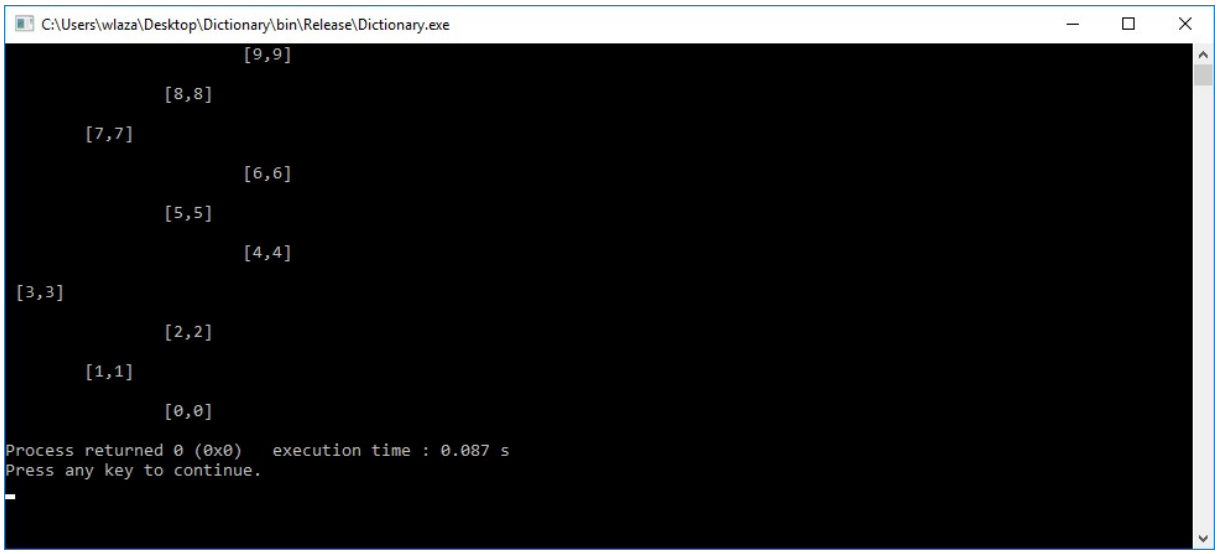
| **Right-left rotation** |
|---|
| `Node* rlrotation(Node *node) noexcept;` |
| **Parameters :** node – *Node* on which rotation will be performer. |
| **Complexity :** linear $O(1)$ |
| **Exception :** exception safe |
| **Notes :** rotation on *Node* which is performed if *Node's bf == 2* and *Node* on the right site has *bf < 0.* |

# 3.1 Organisation of tests

All tests are implemanted and performed in *main.cpp* file. If any error occurs, it will be printed on standard error stream (*std::cerr*). To see if tree keeps proper balance of every *Node* I use *graph* method and simply look at the console output.

Example

*Dictionary<int, int>* graphed after inserting nodes from 0,0 to 9,9 in a row.




Same dictionary after removing *root* (3, 3).