

**PROJECT DOCUMENT**  
**WITH USER MANUAL**

# **AVL TREE AND RED BLACK TREE DATA STRUCTURE**

**AUTHORS: SHUO WANG, ZHAOYI WANG, ZIHAO WANG**

**December 7, 2021**



# Command Line Manual

## Command Line Manual

Part 1: Brief Overview

Part 2: AVL Tree

Part 3: Red-black Tree

---

## Part 1: Brief Overview

---

The three command-line instructions you can run:

- (1) `cargo run avl`: Go to AVL tree interface
- (2) `cargo run rb`: Go to Red-Black tree interface
- (3) `cargo run prebuild`: Run pre-build AVL and RB tree examples

---

## Part 2: AVL Tree

---

1. Enter the following code into the terminal:

```
cargo run avl
```

2. The output is the following:

```
===== AVL HELP MANUAL =====  
0 - Exit  
1 - Insert: insert a node/some nodes to the avl tree
```

```
2 - Delete: delete a node/some nodes from the avl tree
3 - Leaves: count the number of leaves in this avl tree
4 - Height: check the height of this avl tree
5 - In Order Traversal: print the in-order traversal of the avl
tree
6 - Pre Order Traversal: print the pre-order traversal of the
avl tree
7 - Post Order Traversal: print the post-order traversal of the
avl tree
8 - Empty Or Not: check it is empty or not
9 - Print: print this tree
10 - Update: Update the value of a specific node (replace A
with B)
11 - Exist Or Not: Check whether a value exists
12 - Validate: Check whether it is a balanced tree
13 - Total Number: Total number of elements
=====
Please input your choice:
```

3. When you choose 0, the output is the following:

```
Thank you! Hope to see you again!
```

4. When you choose 1, the output is the following:

```
Please input what kind of value you want to add. Separate by
one whitespace.
e.g. 1 2 3 4 5
```

When you input: 1 2 3 4 5(normal situation), the output is the following:

```
Insert [1, 2, 3, 4, 5] successfully.
```

When you input: 1 a 2(wrong situation), the output is the following:

```
Please replace 'a' with a integer!  
INSERT FAILED: Node(1) already exists!  
INSERT FAILED: Node(2) already exists!
```

5. When you choose 2, the output is the following:

```
Current tree contains [1, 2, 3, 4, 5]  
Please input what kind of value you want to delete. Separate by  
one whitespace.  
e.g. 1 2 3 4 5
```

When you input: 2 3(existed), the output is the following:

```
Node(2) delete successfully.  
Node(3) delete successfully.
```

When you input: 7 8(non-existed), the output is the following:

```
DELETE FAILED: No such node(7) to delete  
DELETE FAILED: No such node(8) to delete
```

6. When you choose 3, the output is the following:

```
Number of leaves: 2
```

7. When you choose 4, the output is the following:

```
Height of tree: 2
```

8. When you choose 5, the output is the following:

```
In Order Traverse: [1, 4, 5]
```

9. When you choose 6, the output is the following:





17. When you input characters(wrong situation), the output is the following:

```
Wrong input! Input should be a number from 0-13, please try again...
```

---

## Part 3: Red-black Tree

---

1. Enter the following code into the terminal:

```
cargo run rb
```

2. The output is the following:

```
===== RBTree HELP MANUAL =====
0 - Exit
1 - Insert: insert a node/some nodes to the avl tree
2 - Delete: delete a node/some nodes from the avl tree
3 - Leaves: count the number of leaves in this avl tree
4 - Height: check the height of this avl tree
5 - In Order Traversal: print the in-order traversal of the avl tree
6 - Pre Order Traversal: print the pre-order traversal of the avl tree
7 - Post Order Traversal: print the post-order traversal of the avl tree
8 - Empty Or Not: check it is empty or not
9 - Print: print this tree
10 - Update: Update the value of a specific node (replace A with B)
11 - Exist Or Not: Check whether a value exists
12 - Total Number: Total number of elements
=====
Please input your choice:
```

3. When you choose 0, the output is the following:

Thank you! Hope to see you again!

4. When you choose 1, the output is the following:

Please input what kind of value you want to add. Separate by one whitespace.

e.g. 1 2 3 4 5

When you input: 1 2 3 4 5 (normal situation), the output is the following:

Insert 1 successfully.  
Insert 2 successfully.  
Insert 3 successfully.  
Insert 4 successfully.  
Insert 5 successfully.

When you input: 1 a 2 (wrong situation), the output is the following:

Please replace 'a' with a no-sign integer!  
INSERT FAILED: Node(1) already exists!  
INSERT FAILED: Node(2) already exists!

5. When you choose 2, the output is the following:

Current tree contains [1, 2, 3, 4, 5]

Please input what kind of value you want to delete. Separate by one whitespace.

e.g. 1 2 3 4 5

When you input: 2 3(existed), the output is the following:

Delete 2 successfully  
Delete 3 successfully

When you input: 9(non-existed), the output is the following:







When you input 9(non-existed), the output is the following:

```
Does 9 exist? false
```

15. When you choose 12, the output is the following:

```
This RBTREE has a total of 3 elements.
```

16. When you input characters(wrong situation), the output is the following:

```
Wrong input! Input should be a number from 0-13, please try  
again...
```

---

Document Author: Shuo Wang

---



# AVL Tree Design Document

This AVL tree is part of a tree data structure project by Zhaoyi Wang (Mike), Zihao Wang (Bluce) and Shuo Wang (Tina). The goal of this project is to write AVL tree and Red Black tree using `Rust` language. If you are looking for the project repository, please click [here](#).

---

## Table of Contents (Click to jump)

### AVL Tree Design Document

#### Part 1: Major Innovations

##### Brief Overview

##### Why We Add Those Features?

#### Part 2: Current Shortcomings

#### Part 3: User Manual

##### Quick Start

##### Public Interface

#### Part 4: Performance Evaluation



---

## Part 1: Major Innovations

---

### Brief Overview

Except the following basic features marked by , we add some additional features to this tree marked by .

-  Insert / Delete / Count Leaves / Calculate Height / In-order Traversal / Check Empty / Print Tree
-  Pre-order Traversal / Post-order Traversal / Check Element Existence / Update

## Why We Add Those Features?

For **Pre-order** and **Post-order** traversal, from the data structure point of view, they are the three basic traversal methods along with the in-order traversal. They are very common not only in tree structures, but also in linked lists. From a practical point of view, preorder traversal can be used to implement a computer's file directory structure. For post-order traversal, it can be used to count the memory size occupied by each folder. In addition, post-order traversal can be used as a reference when cleaning up system processes. For example, to clean up a child process before cleaning up the main process.

As for **Check Element Existence** and **Update Node**, in the case of data structures, their purpose is to store data. Then, for a carrier that can store data, it must have four basic functions, that is, add, delete, update and check. For example, MySQL database. That's why we added these two features.

As for the **Validate Tree** function, this feature is designed for users to quickly check if a tree they create is a balanced binary tree. In this way, they do not need to reproduce the code logic with paper and pen or write their own test cases.

For **Total Number of Elements**, it is meant to give the user a macro summary and help the user better understand how many elements are currently stored.

---

## Part 2: Current Shortcomings

---

✗ Up to this stage, if users want to test in the command line, then their input can only be of numeric type (e.g., `1,2,3,4,5`). This version does not support users using character or string types in the command line interface for now (e.g., `'a','b','c'`). *However, users can import the file and use the interface we defined while writing their code and this will not have any restrictions.*

---

# Part 3: User Manual

---

This AVL Tree implementation based on `Box<>` and `Option<>`.

A tree type defined like the following:

```
pub type AVLTreeNode<T> = Option<Box<TreeNode<T>>>;
```

## Quick Start

Before we start, copy the `AVL.rs` file to your root path ( `~/project_name/src` ), and add the following line to `main.rs`.

```
mod AVL;  
use crate::AVL::{AVLTree, AVLTreeNode};
```

First, we need to define an empty tree, and make sure it is mutable.

```
let mut avl_tree: AVLTreeNode<_> = AVLTree::generate_empty_tree();
```

Second, we can add some value to this tree. Here I will use for loop.

```
for i in vec![5, 3, 1, 4, 2] {  
    avl_tree.insert_node(i);  
}
```

Then, print the tree and validate it.

```
avl_tree.print_tree_diagram();  
println!("Balanced Tree? {}", avl_tree.validate_tree());
```

Next, we can calculate the leaves and count the height of the tree.

```
println!("Number of leaves: {}", avl_tree.number_of_leaves());
println!("Height of tree: {}", avl_tree.height_of_tree());
```

Also, if you need, you can do three types of traversal to your tree.

```
println!("In Order Traverse: {:?}", avl_tree.in_order_traverse());
println!("Pre Order Traverse: {:?}", avl_tree.pre_order_traverse());
println!("Post Order Traverse: {:?}",
avl_tree.post_order_traverse());
```

Don't forget to check whether it is empty if you are not sure about whether the above operation was successful.

```
if avl_tree.is_tree_empty() {
    println!("Tree is Empty")
} else {
    println!("Tree is not empty!")
}
```

After that, we can delete some node.

```
avl_tree.delete_node(1);
```

You can get the feedback at the same time, if you want.

```
let res = avl_tree.delete_node(2);
println!("The deleted Node(2) contains {:?}", res);
```

To check the result of the `insert_node()` and `delete_node()` operation, you can do the following to check the existence of a specific node.

```
for i in vec![4, 6, 5] {
    // Check the existence of Node(4), Node(5) and Node (6)
    println!("Does {} exist? {}", i, avl_tree.exist_or_not(i));
}
```

By the way, you can update a node just like the way you want to update a info in your database.

```
avl_tree.update_node(1, 2); // 1 is the OLD one, 2 is the NEW one
```

Finally, let us do in-order traversal again and print the final tree.

```
println!("In Order Traverse: {:?}", avl_tree.in_order_traverse());
avl_tree.print_tree_diagram();
```

And see how many elements we have now.

```
println!("This AVL tree has a total of {} elements.",
avl_tree.total_number_elements());
```

## Public Interface

```
fn insert_node(&mut self, val: T);
// insert a node
fn delete_node(&mut self, val: T) -> Self;
// delete a node
fn validate_tree(&self) -> bool;
// balanced or not?
fn is_tree_empty(&self) -> bool;
// empty or not?
fn height_of_tree(&self) -> i32;
// get the height of this tree
fn number_of_leaves(&self) -> i32;
// how many leaves
```



```

fn in_order_traverse(&mut self) -> Vec<T>;
// In-order traverse
fn pre_order_traverse(&mut self) -> Vec<T>;
// Pre-order traverse
fn post_order_traverse(&mut self) -> Vec<T>;
// Post-order traverse
fn print_tree_diagram(&mut self);
// Nicely print the tree
fn exist_or_not(&self, val: T) -> bool;
// Check whether a value exists
fn generate_empty_tree() -> Self;
// generate a new empty tree
fn update_node(&mut self, old: T, new: T);
// update a node
fn total_number_elements(&mut self) -> i32;
// count total number of elements

```

---

## Part 4: Performance Evaluation

---

The evaluation criteria are as follows:

```

for tree_size in (10000, 40000, 70000, 100000, 130000):
    Create a empty tree;
    Values with tree_size are inserted into the tree;
    A search is conducted for the (tree_size/10) lowest values.

```

This evaluation was done on the following computer configurations, and *the results may vary from computer to computer*.

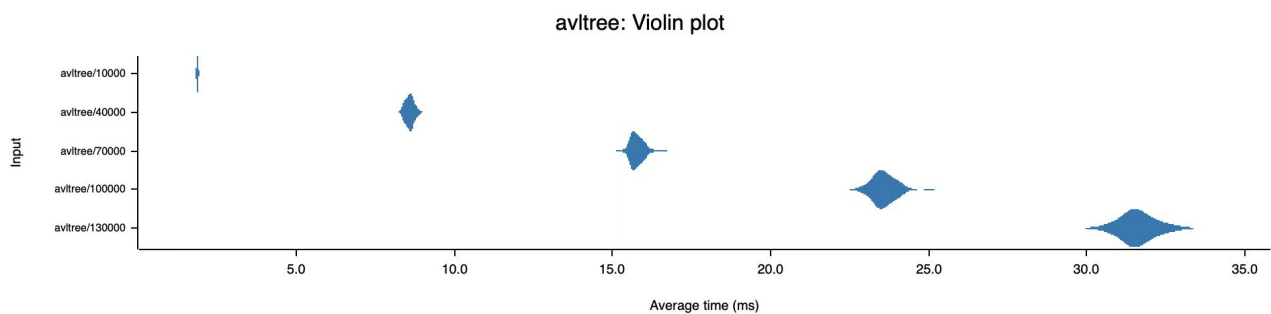
AMD Ryzen 5 3600 6-Core CPU / 16G DDR4 3200MHz Memory

We use `Criterion` crate to perform the benchmark. For more information, please click [here](#).

Size	AVL Tree	Binary Search Tree
10000	1.1143 ms	98.981 ms
40000	4.6040 ms	<i>Stack Overflow</i>
70000	8.2933 ms	<i>Stack Overflow</i>
100000	11.893 ms	<i>Stack Overflow</i>
130000	15.480 ms	<i>Stack Overflow</i>

## avltree

### Violin Plot



This chart shows the relationship between function/parameter and iteration time. The thickness of the shaded region indicates the probability that a measurement of the given function/parameter would take a particular length of time.

😊 Hope you enjoy ! Project made by ❤️

Document Author: Zhaoyi Wang

Project Author: Shuo Wang, Zhaoyi Wang, Zihao Wang

Copyright © 2021 All Authors All rights reserved



# RBTree Design Document

This Red-black tree is part of a tree data structure project by Zhaoyi Wang (Mike), Zihao Wang (Bluce) and Shuo Wang (Tina). The goal of this project is to write AVL tree and Red Black tree using **Rust** language. If you are looking for the project repository, please click [here](#).

---

## RBTree Design Document

Part 1: Major Innovations

Brief Overview

Why We Add Those Features?

Part 2: Current Shortcomings

Part 3: Functions

Part 4: User Manual

Part 5: Performance Evaluation



---

## Part 1: Major Innovations

---

### Brief Overview

Except the following basic features marked by , we add some additional features to this tree marked by .

-  Insert / Delete / Count Leaves / Calculate Height / In-order Traversal / Check Empty / Print Tree
-  Pre-order Traversal / Post-order Traversal / Check Element Existence / Update Node/Total number of elements in a tree

## Why We Add Those Features?

For **Pre-order** and **Post-order** traversal, from the data structure point of view, they are the three basic traversal methods along with the in-order traversal. They are very common not only in tree structures, but also in linked lists. From a practical point of view, preorder traversal can be used to implement a computer's file directory structure. For post-order traversal, it can be used to count the memory size occupied by each folder. In addition, post-order traversal can be used as a reference when cleaning up system processes. For example, to clean up a child process before cleaning up the main process.

As for **Check Element Existence** and **Update Node**, in the case of data structures, their purpose is to store data. Then, for a carrier that can store data, it must have four basic functions, that is, add, delete, update and check. For example, MySQL database. That's why we added these two features.

---

## Part 2: Current Shortcomings

---

✗ Up to this stage, if users want to test in the command line, then their input can only be of numeric type (e.g., `1,2,3,4,5`). This version does not support users using character or string types in the command line interface for now (e.g., `'a','b','c'`). *However, users can import the file and use the interface we defined while writing their code and this will not have any restrictions.*

---

## Part 3: Functions

---

The functions of Red-black are as follow:

```
pub fn insert_node(&mut self, val: u32) -> bool;
```

Test whether the node is inserted successfully.

```
pub fn get_number_leaves(&self) -> u32;
```

Return the number of leaves in a tree.

```
pub fn get_height(&self) -> u32;
```

Return the height of a tree.

```
pub fn is_empty(&self) -> bool;
```

Test whether the tree is empty.

```
pub fn exist_or_not(&mut self, val: u32) -> bool;
```

Test whether the value exists in the tree.

```
pub fn update_node(&mut self, old_val: u32, new_val: u32);
```

Update the tree using new value to replace old value.

```
pub fn delete(&mut self, val: u32) -> Result<(), String>;
```

Delete the node with value in the tree, return the delete result either success(Ok) or failure (Err).

```
pub fn print_in_order_traversal(&self) -> Vec<u32>;
```

Return the vector based on in-order traversal.

```
pub fn print_pre_order_traversal(&self) -> Vec<u32>;
```

Return the vector based on pre-order traversal.

```
pub fn print_post_order_traversal(&self) -> Vec<u32>;
```

Return the vector based on post-order traversal.

```
pub fn print_tree(&self)
```

Print the tree in format<L/R> ::

```
pub fn total_number_elements(&self) -> i32
```

Print total number of elements in a tree.

---

## Part 4: User Manual

---

First, we need to define an empty tree and insert some values into this tree.

```
let mut rb_tree = RBTREE::RBTREE::new();
for i in vec![1,2,3,4,5,6] {
    rb_tree.insert_node(i);
}
```

Then, we can print the tree, print the number of leaves, print the height and print the tree based on in-order, pre-order or post-order traversal.

```
//print the tree
rb_tree.print_tree();
//print the number of leaves of the tree
println!("Number of leaves: {}", rb_tree.get_number_leaves());
//print the height of the tree
println!("Height of tree: {}", rb_tree.get_height());
//print pre/in/post reversal
println!("In Order Traverse: {:?}",
rb_tree.print_in_order_traversal());
println!("Pre Order Traverse: {:?}",
rb_tree.print_pre_order_traversal());
println!("Post Order Traverse:
{:?}",rb_tree.print_post_order_traversal());
```

Don't forget to check whether it is empty when you not sure about your tree.

```
if rb_tree.is_empty() { println!("Tree is Empty") } else { println!
("Tree is not empty!") }
```

After that, we can delete some node.

```
rb_tree.delete(3);
rb_tree.delete(4);
rb_tree.delete(5);
```

To check the result of the `insert_node()` and `delete_node()` operation, you can do the following to check the existence of a specific node.

```
for i in vec![1,2,3,4,5,6] {
    println!("Does {} exist? {}", i, rb_tree.exist_or_not(i));
}
```

By the way, you can update a node just like the way you want to update a info in your database.

```
rb_tree.update_node(2, 3);
```

---

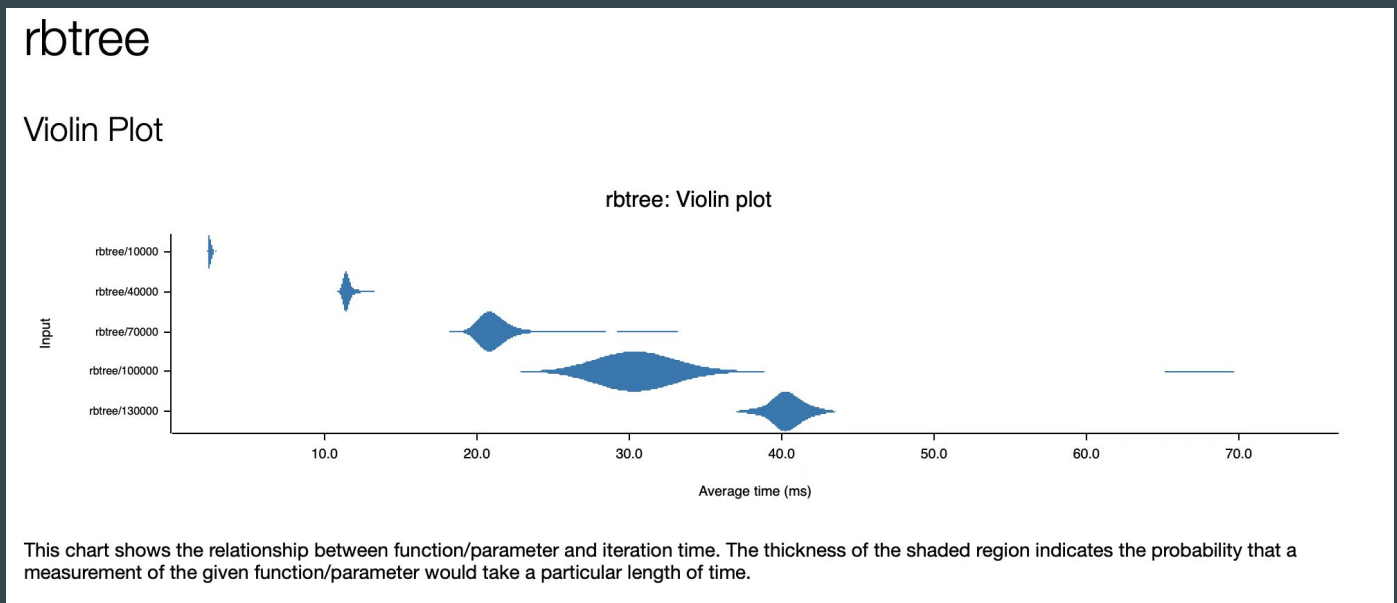
## Part 5: Performance Evaluation

---

We use `criterion` crate to perform the benchmark. For more information, please click [here](#).

Size	RBTree	AVL Tree	Binary Search Tree
10000	2.7116 ms	1.1143ms	98.981 ms
40000	11.917 ms	4.6040 ms	Stack Overflow
70000	24.365 ms	8.2933 ms	Stack Overflow
100000	33.931 ms	11.893 ms	Stack Overflow
130000	39.982 ms	15.480 ms	Stack Overflow

This result shows that a Red-black tree and a AVL tree can make a significant improvment in searching and inserting when comparing with binary search tree.



Hope you enjoy ! 😊 Project made by ❤️

Document Author: Shuo Wang

Project Author: Shuo Wang, Zhaoyi Wang, Zihao Wang

Copyright © 2021 All Authors All rights reserved