ECE 522 Project 2 Documentation
Group 4

A few design questions:

1. What does a red-black tree provide that cannot be accomplished with ordinary binary search trees?
Solution. Unlike ordinary binary search trees, Red Black Trees are self-balancing trees meaning that it makes sure that the tree becomes neither left heavy or right heavy. This is done by recoloring and rotating the tree whenever the given Red Black tree is modified. This leads to faster search, insertion and deletion operations than ordinary binary trees which can become heavier one side. This can be inferred by the worst-case time complexities of Red Black trees and the standard binary search trees as described in the tables below:

Operation	Red Black Trees	Binary Search Trees
Search	O(log n)	O(n)
Insert	O(log n)	O(n)
Delete	O(log n)	O(n)

2. Do you need to apply any kind of error handling in your system (e.g., panic macro, Option<T>, Result<T, E>, etc..)
Solution. Option<T> was used to unwrap the values and the 'if let some' was used to handle null values without encountering any errors. Also, '.expect()' was used in specific cases like creating a file to provide a proper message in case of a failure. In a few cases, we used 'is_some()' and 'is_none()' methods to check for presence or absence of values.

3. What components do the Red-black tree and AVL tree have in common?
Solution. A few common components of Red Black Trees and AVL trees are:

- Performing the rotation operation to ensure a balanced height in left and right subtrees.
- The algorithmic methods to count the leaf nodes, print the trees, get the height of the trees and do the in-order traversal are the same in case of the two trees.

4. How do we construct our design to “allow it to be efficiently and effectively extended”? For example, could your code be reused to build a 2-3-4 tree or B tree?

Solution. Ideally, this can be done by using factory pattern.

Benchmarking Questions:

1- Which data structure is more efficient?
Solution: For Insertion Red-Black Trees and for searching AVL Trees.

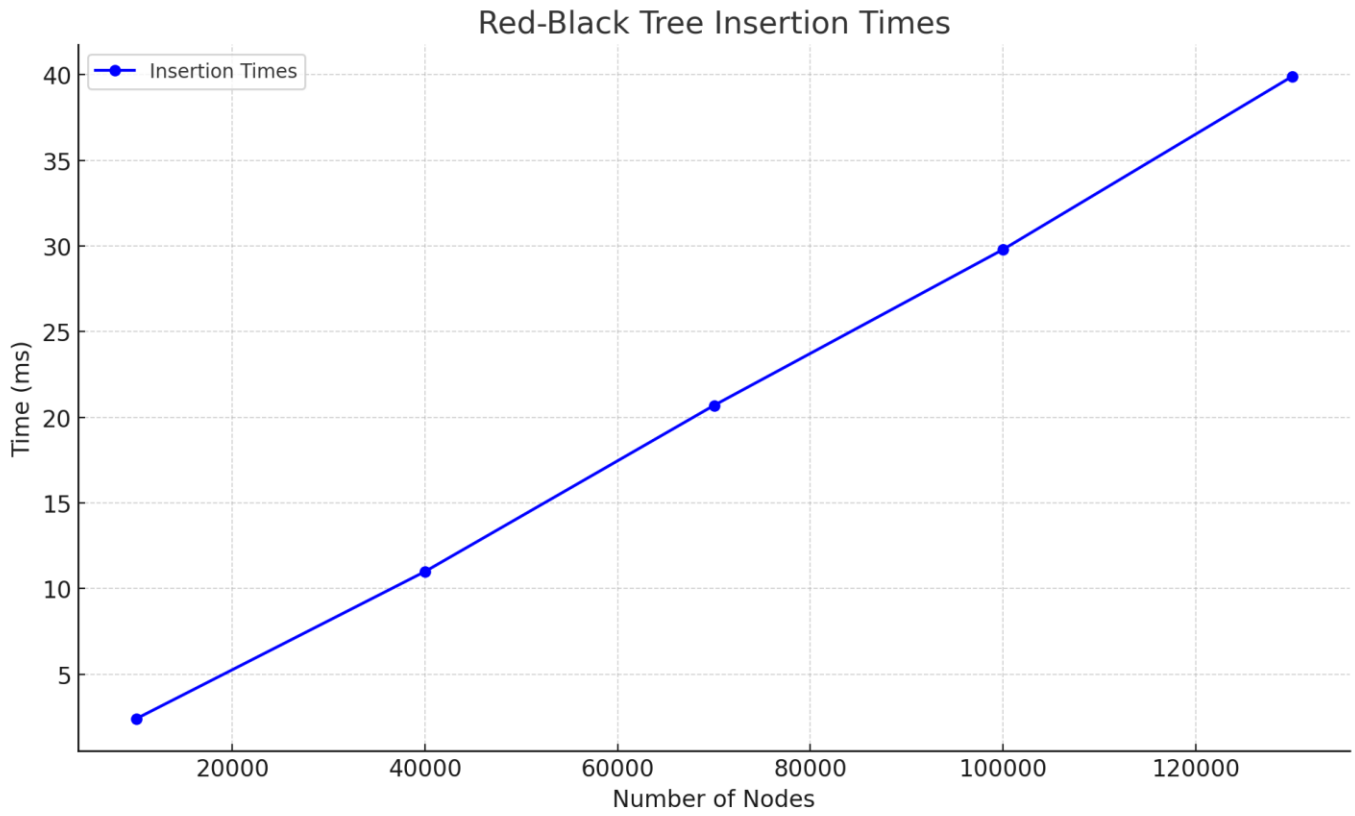
2- Do you think we need to accommodate other test cases?
Solution: One of the tests that can be further accommodated in benchmarking is using the ‘rand’ crate which uses gaussian distribution to generate values and which almost model’s real-world cases rather than just testing the worst-case scenarios every time. This will check for all rotation cases and locate all kinds of borrow errors that might be present in our application.

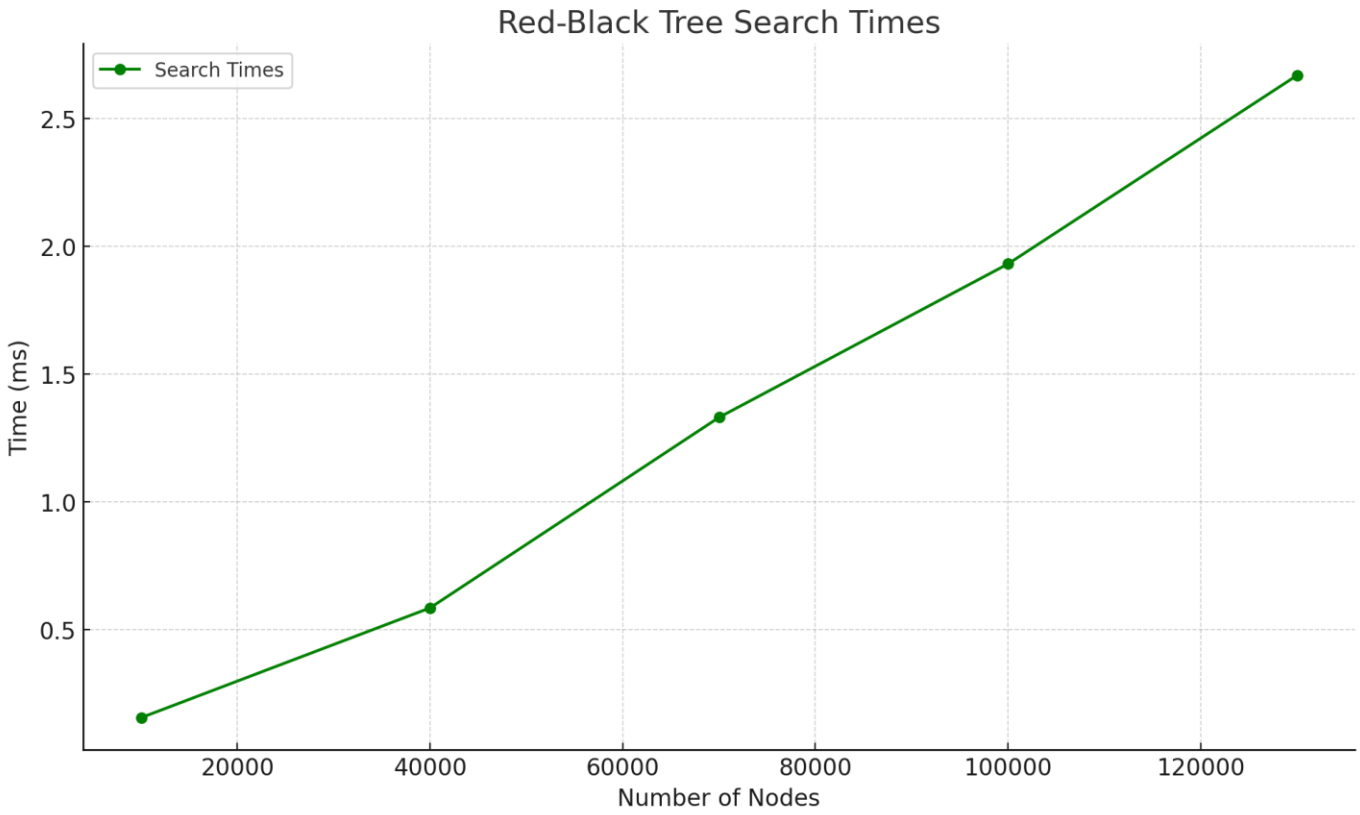
3- Do you think we need to include additional data structures in the benchmarking to perform as the baseline (i.e., binary search tree)?
Solution: Since binary search trees typically perform worse than both Red-Black Trees and AVL trees, including it as a baseline would not necessarily provide any useful information.

We have plotted the graphs for benchmarking for red-black and AVL trees insertion and searching operation separately and a comparison graph as well to compare the performances.

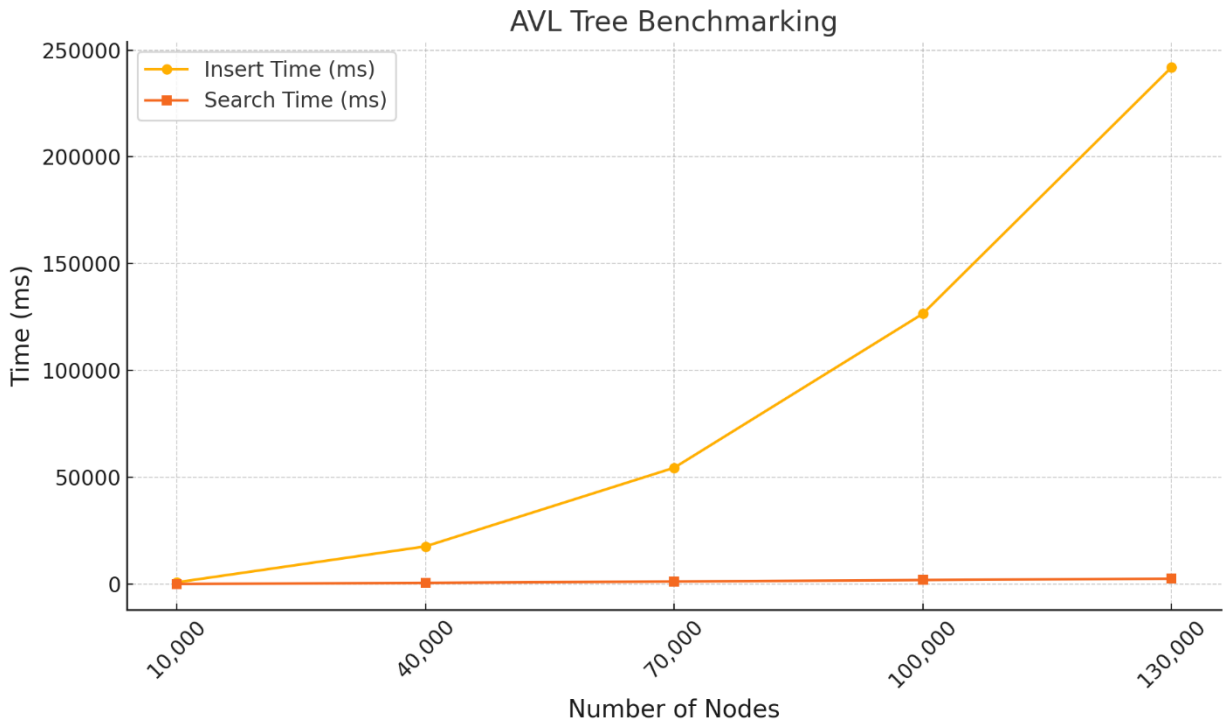
Note: All the benchmarking files are there in their respective directories. For example, for AVL its avl_bench.txt and for Red-Black trees its redblack_bench.txt.

BENCHMARKING RESULT FOR RED-BLACK TREES:





Benchmarking for AVL Trees:



Project Specifications:

1. Objective: The main objective of this project was to create implementations for Red Black Trees and AVL (Adelson-Velsky and Landis) trees and also perform benchmarking in order to determine which trees have better performance in insertion and search time.
2. Functional Requirements: For both of these trees the user can perform the following functions:
    - Inserting a Node
    - Deleting a Node
    - Counting the number of Leaves
    - Returning the height of the trees
    - Printing In-order traversal of the trees
    - Checking if the trees are empty
    - Printing the trees (Showing color and structure in case of Red Black Trees and Structure in case of AVL Trees).
3. Technical Specifications: All the implementations have been made in Rust by utilizing Rust smart pointers (Rc, RefCell and Weak). Command Line interface has also been integrated in our implementations.
4. Deliverables: The deliverables of this project are described as follows:
    - A copy of the source code of the three libraries (utility method to print trees, Red Black tree implementation and AVL tree implementation).

- A document outlining: Project specifications, major innovations, Rationale behind design decisions, a list of known errors, faults, defects and missing functionalities, a User Manual.
- A 2-minute video highlighting the new system,

User manual:

1. **Brief Background of Red-Black and AVL Trees:**
    a. **Red-Black Trees:** These are a type of self-balancing binary search tree where each node is either red or black. These trees were invented in 1978 by Leonidas J. Guibas and Robert Sedgewick. Properties followed by Red Black Trees are as follows:
        - Every node is either red or black.
        - The root node is always black
        - All None nodes are considered black.
        - A red node cannot have a red child.
        - Every path from root to its leaves has the same number of black nodes.

    Whenever a red black tree is modified it is rotated and recolored in order to maintain the above stated properties of Red Black Trees.

    b. **AVL Trees:** These are a type of self-balancing binary search tree which were invented by Georgy Adelson-Velsky and Evgenii Landis in 1962. An AVL tree keeps the tree balanced. It does this by ensuring that the height difference between the left and right subtrees of any node is at most 1. This height difference is called the balance factor
    Balance Factor = Height of Left Subtree - Height of Right Subtree

    For AVL trees, the balance factor must always be 1, 0, or -1. Whenever an AVL tree is modified, it is rotated to ensure the above stated values of the Balance Factor.

2. **Usage Details:**
    - **Step 1:** Download the .zip files.
    - **Step 2:** Extract all the files.
    - **Step 3:** Install the Graphviz library (https://graphviz.org/download/)
    - **Step 4:** Go to the project main directory

```
PS C:\Users\prabh> cd Project2
PS C:\Users\prabh\Project2> ls


    Directory: C:\Users\prabh\Project2


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2024-12-09   2:09 PM                .idea
d-----        2024-12-09   2:28 PM                src
d-----        2024-12-09   2:30 PM                target
-a----        2024-12-09   2:05 PM              9 .gitignore
-a----        2024-12-09   4:37 PM            690 avl.dot
-a----        2024-12-09   4:37 PM          21363 avl.png
-a----        2024-12-09   2:30 PM          18578 Cargo.lock
-a----        2024-12-09   2:28 PM            318 Cargo.toml
-a----        2024-12-09   4:40 PM            800 rbt.dot
-a----        2024-12-09   4:40 PM          22027 rbt.png
-a----        2024-12-09   3:21 PM          21610 rbt1.png
-a----        2024-12-09   2:05 PM             12 README.md
```

Fig. 1 Project main directory structure

- **Step 5:** Run the following command:
  cargo run -- --tree-type <Option>

  Your option should be one of the following: RedBlackTree or AVL_Tree

```
PS C:\Users\prabh\Project2> cargo run -- --tree-type RedBlackTree
   Compiling Project2 v0.1.0 (C:\Users\prabh\Project2)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.62s
     Running `target\debug\Project2.exe --tree-type RedBlackTree`
RBT chosen
Enter 1 for Insertion operation
Enter 2 for Deletion operation
Enter 3 for Number of leaves in the Tree
Enter 4 for Height of Tree
Enter 5 for In-order traversal of the Tree
Enter 6 to check if the tree is empty
Enter 7 to print the tree showing its structure
```

Fig 2. Sample CLI command to run Red Black Trees

- **Step 6:** Select the function you want to perform from the given list. For example: - Inserting 10, 20, 30, 40, 50 in Red Black Trees: -

```
RBT chosen
Enter 1 for Insertion operation
Enter 2 for Deletion operation
Enter 3 for Number of leaves in the Tree
Enter 4 for Height of Tree
Enter 5 for In-order traversal of the Tree
Enter 6 to check if the tree is empty
Enter 7 to print the tree showing its structure
1
Enter a comma-separated list of numbers:
10,20,30,40,50,60
Parsed numbers: [10, 20, 30, 40, 50, 60]
10 was inserted in the tree
20 was inserted in the tree
30 was inserted in the tree
40 was inserted in the tree
50 was inserted in the tree
60 was inserted in the tree
Enter 1 for Insertion operation
Enter 2 for Deletion operation
```

Fig. 3 An example to show the insertion operation in Red Black Trees

Then printing the Red Black Tree created in the above code:

```
Enter 1 for Insertion operation
Enter 2 for Deletion operation
Enter 3 for Number of leaves in the Tree
Enter 4 for Height of Tree
Enter 5 for In-order traversal of the Tree
Enter 6 to check if the tree is empty
Enter 7 to print the tree showing its structure
7
Status: exit code: 0
stdout:
stderr:
Successfully generated the png file. File path: ./rbt.png
Enter 1 for Insertion operation
Enter 2 for Deletion operation
Enter 3 for Number of leaves in the Tree
Enter 4 for Height of Tree
Enter 5 for In-order traversal of the Tree
Enter 6 to check if the tree is empty
Enter 7 to print the tree showing its structure
```

Fig 4. An example to show the printing operation in Red Black Trees

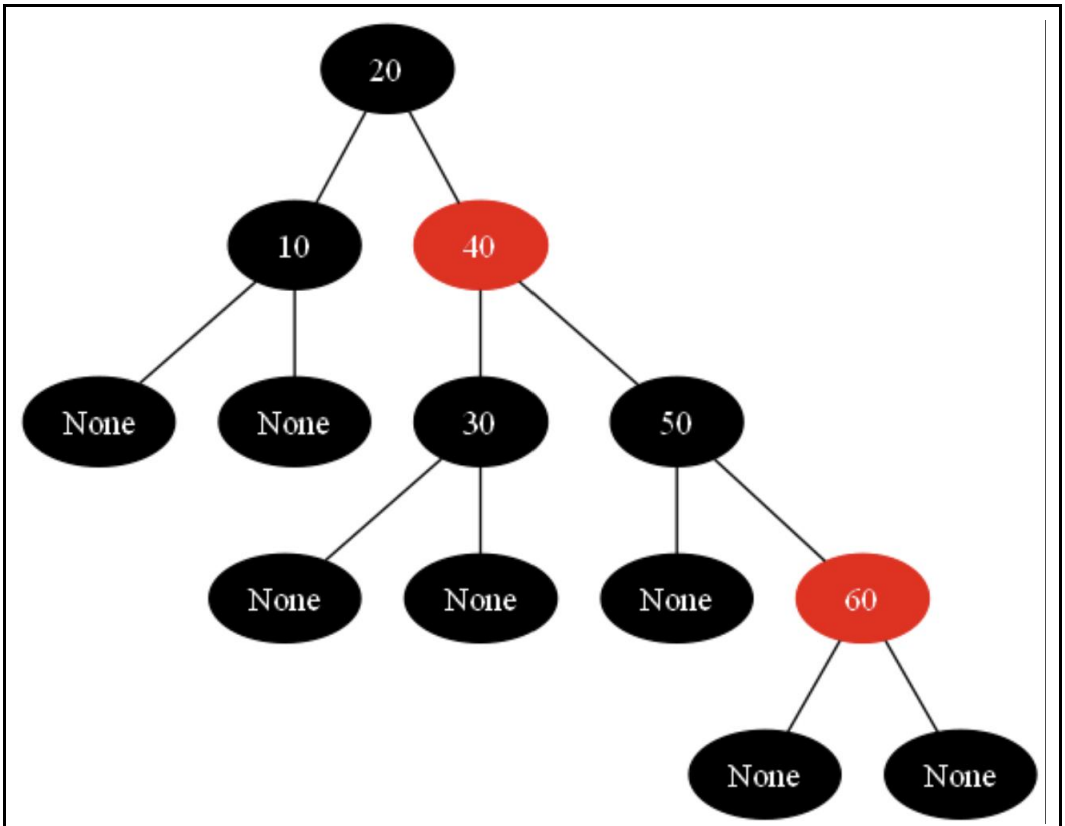


Fig 5. The image generated by our program

The generated image can be located at this path: -
For Red-Black Trees: ./rbt.png
For AVL Trees: ./avl.png

- **Step 7:** To exit the process, enter '0'.

Major innovations:

- Created utility to create graphs of any kind of tree.
- Common functions for both the trees except for Insert and delete
- Weak References were used in order to avoid infinite cycles which were being caused by the use of strong references.

A list of known errors, faults, defects, missing functionality, etc.

- Since we are using RC instead of ARC, the implementations work only in single threaded applications.
- In the case of AVL trees, in certain cases intermittent deletion errors due to borrow checking were encountered.
- In the case of Red Black trees, in certain cases intermittent insertion errors due to borrow checkering were encountered.
- Automated tests and unit tests were not included.

References:

- https://en.wikipedia.org/wiki/Red%E2%80%93black_tree
- https://en.wikipedia.org/wiki/AVL_tree
- https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Raw Outputs from Benchmarking for Reference:

AVL Trees:

```
running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s


running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

avl_insert_10000        time:   [932.14 ms 936.09 ms 940.09 ms]
                        change: [-12.620% -10.658% -8.6831%] (p = 0.00 < 0.05)
                        Performance has improved.
Found 2 outliers among 100 measurements (2.00%)
  2 (2.00%) high mild

avl_search_10000        time:   [143.39 ┬╡s 144.88 ┬╡s 146.25 ┬╡s]
                        change: [+19.270% +23.446% +26.723%] (p = 0.00 < 0.05)
                        Performance has regressed.
Found 7 outliers among 100 measurements (7.00%)
  3 (3.00%) low severe
  4 (4.00%) low mild

avl_insert_40000        time:   [17.704 s 17.755 s 17.811 s]
Found 6 outliers among 100 measurements (6.00%)
  5 (5.00%) high mild
  1 (1.00%) high severe

avl_search_40000        time:   [670.98 ┬╡s 685.56 ┬╡s 699.02 ┬╡s]
Found 14 outliers among 100 measurements (14.00%)
  1 (1.00%) low severe
  12 (12.00%) low mild
  1 (1.00%) high mild
```

avl_insert_70000 time: [54.425 s 54.555 s 54.708 s]
Found 3 outliers among 100 measurements (3.00%)
1 (1.00%) high mild
2 (2.00%) high severe

avl_search_70000 time: [1.2792 ms 1.3001 ms 1.3198 ms]
Found 2 outliers among 100 measurements (2.00%)
1 (1.00%) low mild
1 (1.00%) high mild

avl_insert_100000 time: [125.71 s 126.63 s 127.57 s]

avl_search_100000 time: [2.0011 ms 2.0104 ms 2.0192 ms]
Found 6 outliers among 100 measurements (6.00%)
2 (2.00%) low severe
3 (3.00%) low mild
1 (1.00%) high mild

avl_insert_130000 time: [219.49 s 241.96 s 281.83 s]
Found 7 outliers among 100 measurements (7.00%)
6 (6.00%) high mild
1 (1.00%) high severe

avl_search_130000 time: [2.6211 ms 2.6359 ms 2.6507 ms]
Found 5 outliers among 100 measurements (5.00%)
2 (2.00%) low mild
3 (3.00%) high mild

Red-Black Trees:

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

rbt_insert_10000 time: [2.3749 ms 2.3997 ms 2.4260 ms]
change: [-3.0727% -0.6865% +1.5765%] (p = 0.57 > 0.05)

No change in performance detected.
Found 10 outliers among 100 measurements (10.00%)
  8 (8.00%) high mild
  2 (2.00%) high severe

rbt_search_10000   time: [152.17 ┬╡s 153.63 ┬╡s 154.99 ┬╡s]
                        change: [-0.6133% +0.6222% +1.7233%] (p = 0.29 > 0.05)
                        No change in performance detected.
Found 3 outliers among 100 measurements (3.00%)
  3 (3.00%) low mild

rbt_insert_40000   time: [10.901 ms 11.019 ms 11.155 ms]
                        change: [-11.223% -8.0946% -4.7739%] (p = 0.00 < 0.05)
                        Performance has improved.
Found 4 outliers among 100 measurements (4.00%)
  2 (2.00%) high mild
  2 (2.00%) high severe

rbt_search_40000   time: [574.37 ┬╡s 583.59 ┬╡s 593.05 ┬╡s]
                        change: [-20.410% -18.641% -16.717%] (p = 0.00 < 0.05)
                        Performance has improved.
Found 6 outliers among 100 measurements (6.00%)
  5 (5.00%) high mild
  1 (1.00%) high severe

rbt_insert_70000   time: [20.466 ms 20.696 ms 20.951 ms]
                        change: [+2.7790% +4.2491% +5.8279%] (p = 0.00 < 0.05)
                        Performance has regressed.
Found 4 outliers among 100 measurements (4.00%)
  3 (3.00%) high mild
  1 (1.00%) high severe

rbt_search_70000   time: [1.3149 ms 1.3297 ms 1.3423 ms]
                        change: [+0.1789% +1.5673% +2.9804%] (p = 0.02 < 0.05)
                        Change within noise threshold.
Found 12 outliers among 100 measurements (12.00%)
  6 (6.00%) low severe
  5 (5.00%) low mild
  1 (1.00%) high mild

rbt_insert_100000   time: [29.423 ms 29.749 ms 30.123 ms]
                        change: [+1.8950% +3.4504% +5.1701%] (p = 0.00 < 0.05)
                        Performance has regressed.
Found 6 outliers among 100 measurements (6.00%)

```
  2 (2.00%) high mild
  4 (4.00%) high severe

rbt_search_100000       time:   [1.9053 ms 1.9307 ms 1.9519 ms]
                        change: [-5.8562% -4.5965% -3.5390%] (p = 0.00 < 0.05)
                        Performance has improved.
Found 8 outliers among 100 measurements (8.00%)
  8 (8.00%) low mild

rbt_insert_130000       time:   [39.511 ms 39.861 ms 40.241 ms]
                        change: [+4.5971% +6.0116% +7.3212%] (p = 0.00 < 0.05)
                        Performance has regressed.
Found 8 outliers among 100 measurements (8.00%)
  7 (7.00%) high mild
  1 (1.00%) high severe

rbt_search_130000       time:   [2.6563 ms 2.6690 ms 2.6802 ms]
                        change: [-3.7263% -1.4947% +0.3146%] (p = 0.18 > 0.05)
                        No change in performance detected.
Found 9 outliers among 100 measurements (9.00%)
  6 (6.00%) low severe
  3 (3.00%) low mild
```