Design Document for Red Black Tree and AVL Tree

ECE 522

Group 1

**Outline**

* Major Innovations – Additional to the project specifications
* Design Decision Rationale
* System’s limitations
* User manual
* Benchmarking
* Conclusion

**Major Innovations:**

**Red Black Tree:**

1. Preorder traversal of tree

This line of code prints out the pre-order traversal of red-black tree.

Usage:

red\_black\_tree.preorder\_trav\_print();

2. Postorder traversal of tree

The function postorder\_trav\_print() shows the result from post-order traversal of the red-black tree.

Usage:

red\_black\_tree.postorder\_trav\_print();

3. Iterate through the tree.

\*Progress

**Design Decision Rationale:**

1. What does a red-black tree provide that cannot be accomplished with ordinary binary search trees?

An ordinary binary search tree has a worst-case scenario of getting unbalanced which will result in O(N) time for inserting, removing and searching operations on the nodes. It is very easy to get into a trap to create an unbalanced tree and once an ordinary binary search tree gets unbalanced, then it cannot be repaired.

The Red-black tree overcomes limitations of ordinary binary search trees by adding the ability to rebalance the tree after insertion or deletion operation. So, the same operations can be performed in O(log(n)) time. Also, the balancing of tree makes searching much faster than the Binary search trees.

2. Please add a command-line interface (function main) to your crate to allow users to test it.

3. Do you need to apply any kind of error handling in your system (e.g., panic macro, Option<T>, Result<T, E>, etc..)

4. What components do the Red-black tree and AVL tree have in common? Don’t Repeat Yourself! Never, ever repeat yourself – a fundamental idea in programming.

The common components in Red black Tree and AVL tree are given as follows:

* • Creating a new tree (function name: *new*)
* • Tree operations i.e. Rotate Left, Rotate Right. (function names: *rotate\_left, rotate\_right*)
* • Minimum value of the tree. (function name: min)
* • Searching values in the tree. (function names: *find/contains*)
* • Getting the height of the tree. (function name: *height*)
* • Obtaining number of leaves in a tree. (function names: *count\_leaves*, *print\_leaves*)

5. 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?

**System’s Limitations:**

**User Manual:**

**Red Black Tree:**

1. Creating a red black tree.

This line of code initializes an empty red black tree.

Usage:

let mut red\_black\_tree: RBTree<usize, usize> = RBTree::new();

2. Inserting a node into the red-black tree

This function inserts the key and value of a node into the tree. If a node already exists in the tree, it doesn’t allow the duplicate node to be inserted.

Usage:

red\_black\_tree.insert(key of node, value of node).unwrap();

2. Deleting a node from red-black tree

This deletes the node with matching key and values from the red black tree.

Usage:

red\_black\_tree.delete(key of node, value of node).unwrap();

3. Counting the number of leaves in a tree.

\*\*Needs to be checked if it is the right one.

This function prints out the number of leaves in the red black tree.

Usage:

println!(“The number of leaves in the red black tree is: {}, red\_black\_tree.len());

4. Returning height of the tree.

The function get\_height returns the height of the red black tree.

Usage:

println!(“The height of the tree is: {:?}”, red\_black\_tree.get\_height());

5. Printing out in-order traversal of the tree.

This line of code prints out the in-order traversal of red-black tree.

Usage:

red\_black\_tree..inorder\_trav\_print();

6. Checking if the tree is empty.

The function is\_empty() returns a Boolean value indicating emptiness of the red black tree. If there are no nodes in the tree, it returns a ‘true’ value, else it returns ‘false’.

Usage:

println!(“Is the red black tree empty?: {}”, red\_black\_tree.is\_empty());

**Benchmarking:**

\*We need a function to search for the size/10 elements in the tree.

**Conclusion:**