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:**

**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*)

**System’s Limitations:**

**User Manual:**

**Benchmarking:**

**Conclusion:**