**CS 590 - Assignment 3**

**Submitted By: Shweta Madhale**

**CWID: 20015921**

**Abstract**:

A binary search tree is a data structure that can support multiple dynamic-set operations like search, insert, remove, etc. Such a binary search tree was implemented with few functionalities using C++ for different inputs. The binary tree works well with smaller heights but is significantly inefficient for larger heights. So, the red-black trees which are balanced are used to improve efficiency. The red-black tree was also implemented for different inputs. A random generator was used to create various keys up to the given input size. These keys were inserted in both trees in random, sorted, and inverse sorted. The tree implementation included insertion (for unique nodes), removal of nodes, traversing nodes, converting a tree to an array in ascending order, finding the black height, performing various insertion cases for RBT, and performing different rotations for RBT. Also, counter for various mentioned operations. The runtime performance for sorting and computing black height for these trees was observed for input sizes = = 50000; 100000; 250000; 500000; 1000000; 2500000; 5000000. A scatterplot was obtained for these performances. An analysis of both trees’ performance was done.

**Observations:**

1. **Performance and counter for BST**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Binary Search Tree | | | | | | |
| n | Random | | Sorted | | Inverse Sorted | |
| Sorting Runtime (ms) | Duplicate Nodes | Sorting Runtime (ms) | Duplicate Nodes | Sorting Runtime (ms) | Duplicate Nodes |
| 50,000 | 25.6 | 0.8 | 4348.2 | 0 | 4203.8 | 0 |
| 100,000 | 51.4 | 2 | 17164.2 | 0 | 18483.6 | 0 |
| 250,000 | 143 | 13 | - | - | - | - |
| 500,000 | 378.2 | 59.2 | - | - | - | - |
| 1,000,000 | 1047.4 | 233.2 | - | - | - | - |
| 2,500,000 | 3595 | 1446.2 | - | - | - | - |
| 5,000,000 | 8805.2 | 5780.8 | - | - | - | - |

1. **Performance for random inputs for RBT**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | Random Input | | | | | | | | |
| Sorting Runtime | Duplicate Nodes | Case 1 Insertion | Case 2 Insertion | Case 3 Insertion | Left Rotation | Right Rotation | Runtime (Black Height) | Black Height |
| 50000 | 25.8 | 0.6 | 25692.4 | 9699.6 | 19402.2 | 14526 | 14575.8 | 1.2 | 19 |
| 100000 | 51.8 | 2 | 51364.8 | 19435.8 | 38916.6 | 29213.4 | 81254.6 | 6 | 20.4 |
| 250000 | 158.6 | 14.2 | 128368.8 | 48747.2 | 97233 | 72949.6 | 73039.6 | 19.8 | 22 |
| 500000 | 386.4 | 54.8 | 256711 | 97027 | 543482 | 145634.2 | 406582 | 53.2 | 23 |
| 1000000 | 1057 | 235.8 | 513207.6 | 193873.4 | 1088205 | 291126.2 | 290952.2 | 120 | 24.6 |
| 2500000 | 3603.6 | 1475.8 | 1283300.2 | 484921.8 | 970801.2 | 727947.2 | 727729.8 | 337.4 | 26 |
| 5000000 | 7923 | 5794.6 | 2564106.4 | 969564.4 | 1939730.2 | 1454626.2 | 1454668.4 | 684.4 | 27 |
|  |  |  |  |  |  |  |  |  |  |

1. **Performance for sorted inputs for RBT**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | Sorted Input | | | | | | | | |
| Sorting Runtime | Duplicate Nodes | Case 1 Insertion | Case 2 Insertion | Case 3 Insertion | Left Rotation | Right Rotation | Runtime (Black Height) | Black Height |
| 50000 | 20.6 | 0 | 49966 | 0 | 49971 | 49971 | 0 | 3 | 29 |
| 100000 | 37.4 | 0 | 99964 | 0 | 99969 | 99969 | 0 | 3.2 | 31 |
| 250000 | 81.8 | 0 | 249961 | 0 | 249967 | 249967 | 0 | 7.4 | 33 |
| 500000 | 156.6 | 0 | 499959 | 0 | 499965 | 499965 | 0 | 17.6 | 35 |
| 1000000 | 293.6 | 0 | 999957 | 0 | 999963 | 999963 | 0 | 35 | 37 |
| 2500000 | 779.2 | 0 | 2499952 | 0 | 2499960 | 2499960 | 0 | 90.6 | 40 |
| 5000000 | 1616.4 | 0 | 4999950 | 0 | 4999958 | 4999958 | 0 | 177 | 42 |

1. **Performance for inverse sorted inputs for RBT**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | Random Input | | | | | | | | |
| Sorting Runtime | Duplicate Nodes | Case 1 Insertion | Case 2 Insertion | Case 3 Insertion | Left Rotation | Right Rotation | Runtime (Black Height) | Black Height |
| 50000 | 22 | 0 | 49966 | 0 | 49971 | 0 | 49971 | 2.8 | 29 |
| 100000 | 44.2 | 0 | 99964 | 0 | 99969 | 0 | 99969 | 2.2 | 31 |
| 250000 | 81.2 | 0 | 249961 | 0 | 249967 | 0 | 249967 | 6.8 | 33 |
| 500000 | 150.4 | 0 | 499959 | 0 | 499965 | 0 | 499965 | 18 | 35 |
| 1000000 | 293.6 | 0 | 999957 | 0 | 999963 | 0 | 999963 | 33.4 | 37 |
| 2500000 | 781.8 | 0 | 2499952 | 0 | 2499960 | 0 | 2499960 | 87 | 40 |
| 5000000 | 1602.2 | 0 | 4999950 | 0 | 4999958 | 0 | 4999958 | 175.8 | 42 |

**Discussion:**

**Chart, line chart

Description automatically generated**

**Chart, line chart

Description automatically generated**

The plots of the runtime vs the input size parameters are shown above. From the plots above, the BST sorting performance becomes significantly worse with an increase in input size for the sorted and inverse sorted inputs. There is not a big difference in the BST and RBT performance for the random input, but this might change with increasing input size. The RBT performed significantly better than BST, especially for the sorted and inverse sorted inputs. This verifies that the RBT’s performance is much better than BST for greater heights.

For the RBT, with an increase in input, the counters increase rapidly, the black height increases much more slowly. For sorted input, there is no insertion with case 2, nor is there a right rotation. Whereas, for the inverse sorted input, there is no case 2 insertion and there is no left rotation.

**Analysis:**

1. Sorting performance of BST and RBT is comparable for random inputs (RBT is slightly better)
2. The sorting performance of RBT is extremely more efficient than BST for sorted and inverse sorted inputs.
3. RBT is more efficient than BST, especially for increasing input size.
4. The black height of RBT increases gradually with the rapid increase in the input size.
5. For sorted input, no case 2 insertions are done, and there is no right rotation.
6. For inverse sorted input, no case 2 insertions are done, and there is no left rotation.

**Conclusion:**

1. After implementing BST and RBT with their set operations, the performance of each of these was observed.
2. BST runtime is slightly worse than BST for random input.
3. BST runtime is extremely bas than BST for sorted and inverse sorted input.
4. RBT is significantly more efficient than BST.
5. BST has best case runtime of O(log(N)) and worst case of O(N), whereas, RBT has best and worst-case runtime of O(log(N))
6. Black height of RBT increases gradually with increasing input size.