**Balanced Search Tree (Left Leaning Red Black BST)**

Theoretical analysis:

Space complexity:

The space complexity of a red-black tree is O(n), where n is the number of nodes in the tree. This is because each node in the tree requires a fixed amount of memory for storing its key and pointers to its left and right children. As the number of nodes in the tree increases, the amount of memory required to store the tree also increases linearly.

In average case, where each element inserted is mostly distinct, space required for n insertions is close to but slightly smaller than 1 n nodes. However, in worst case where every element is distinct, so the space required is strictly 1 n nodes.

Time complexity:

For a left leaning red black binary search tree, no two red links are in a row, and the root of the tree is black. Every red node has a black left children and a black right children. In the worst case, the longest possible path alternates between black and red links, so the length is equal to 2 \* number of black links. Since each node has at most one red child, there can be at most n/2 red nodes in the tree. In this situation, there are n/2 black nodes in the tree. The longest path is 2 log2(n/2), which simplifies to 2 log(n). The time complexity for both search and insert is proportional to the height of the tree, so in the worst case, the time complexity is 2 log(n).

In average case, distribution of nodes affects the height of the tree. In general, the more balanced the distribution of nodes is within the tree, the closer the average height will be to the minimum height. The minimum height is achieved where all the nodes are black, and that height is log2(n+1). Thus, the average time complexity is close to 1 log(n).

Experimental analysis:

Real data:

Total insertion time for test 1 is 1.5243637999999464

Total search time for test 1 is 0.001167500000065047

Total insertion time for test 2 is 3.7367073000000346

Total search time for test 2 is 0.001385700000014367

Total insertion time for test 3 is 39.49727419999999

Total search time for test 3 is 0.001562600000056591

( comparisons for time between different algorithms need to be added)

Synthetic data:

The aspects that we take into consideration are:

Strings are inserted in ascending, descending or random order.

Strings inserted have a high number of duplicates, or may include None.

The length of the strings inserted.

The number of strings inserted.

Whether the search values are in the inserting set.

When considering one particular aspect, we keep all others at a default setting, which is: inserting 100,000 strings with length 5 to 10 characters each in a random order with no or few duplicates and no None. Then measure the total insertion time and total time for searching every element in the set.

We generate string by putting together randomly chosen ASCII characters.

For number of strings inserted:

We generate set of random strings with an increasing number, ranging from 5 to 1,000,000. The aim for this test is to check if the time complexity grows as expected, and to compare the performances for 4 different algorithms.

For left leaning red black binary search tree, insertion and search time increases as the number of strings inserted. However, the growth rate is greater than log(n), which is what we expect in theoretical analysis. Instead, the time complexity is closer to O(n).

For orders which strings are inserted in:

We test the performance of algorithms when inserting strings in ascending order, descending order and random order. Merge sort is used to sort the synthetic data. We did this because inserting ordered elements is a worse case for some algorithms, for example, binary search tree.

For left leaning red black binary search tree, insertion is faster at ascending order, random order, and slowest at descending order, and search is faster at descending, ascending and slowest at random order.

For length of strings inserted:

We change the length of each string inserted, from 3 to 8, to 40 to 45. We want to test if the length of strings has an impact on the algorithms’ performances because comparisons of longer strings may take longer.

For left leaning red black binary search tree, the insertion and search time fluctuates, indicating no obvious impact.

For duplicates in inserted strings:

We set the duplicate proportion to range from 0, 0.2, 0.4 ... to 1. For each proportion, for example 0.2, it means, 80% of the strings are distinct, where 10% repeats themselves, resulting in 100% strings. Randomly generated strings are assumed to have no or very few duplicates themselves. With more duplicates, the time taken is expect to be shorter, since no same element needs to be inserted, and the size of the tree would be smaller.

For left leaning red black binary search tree, both insertion and search time fluctuate at first, and decrease dramatically at duplicate proportion greater than 0.8. This indicates that when there are plenty of duplicates, the algorithm has a better performance.

For number of None:

We replace number of strings to be inserted into None, since there could be empty strings. The number ranges from 0 to 40000, which results in 40% of the strings being empty. The effect of empty strings is similar to that of duplicates, because empty strings do not need to be inserted. The size of the tree can be smaller.

For left leaning red black binary search tree, as number of None increases, insertion and search time generally decreases, which is the same as what we expect.

For searching values absent in the set:

We set the absent number ranging from 0 to 100,000. In the final case, all search values are absent in the set. We did this to find out if the algorithms takes longer time to search for values that are not in the set. For the searching set, we use newly generated strings to represent the strings absent in the list, and extract from the inserting set as the present part. However, there is a small possibility that our random generator would give us a string that is present in the set, while we assume it is absent. But this chance is really small, so it can be regarded as negligible.

For left leaning red black binary search tree, as absent number increases, the search time generally increases, showing that it does take longer for the tree to search for values that are not in the set.