Experiment Organization

Experimentally, both black box tree implementations are identical with minor algorithmic differences. Within the scope of organizational structure, after instantiation of a Tree model, provided trials loop for a value of ‘n’ within the specified ‘*srand’* seed sequence. The motivation behind individual Tree trials resides mainly in the sake of simplicity, though mutual exclusivity in experimentation also increases timing accuracy.

Sequentially in order, we begin with the Initialize phase where a given Tree implementation builds up an array of data-points. Secondly, we begin an *‘srand’* loop for 5 different seeds, and within each of these seed iterations there is also a nested *‘for’* loop for each provided value of ‘n’. The meat of the experimentation resides within the ‘n’ for loop iterations, mainly an insertion loop that injects values into the Tree structure for a given number of ‘n’ data-points. After each inner for loop trial the Tree is re-instantiated, and when an entire implementation test completes the Tree structure is removed from memory. This process repeats once for the leftist tree, and then once for the skew tree.

After, the raw insertion phase, a new trial is run where operations are performed on the tree in a random fashion. The structure of these test directly mirrors the test above, however, an if condition and a random variable determine whether to insert a value or perform a delete operation.

Data generation

The data generated from this experiment stems from carefully places timing statements according to the following:

* Individual Timing counts are isolated to a single while loop of random insertions
* This timing count includes random number generation
* Tree structures are re-instantiated between iterations (trivial time penalty)
* Total execution time is recorded (Mostly, for posterities sake)
* Computation environments were isolated during program execution for accuracy (meaning the only non-system process running was the experimentation code)

Summary of results (CPU timing)

Testing Environments:

|  |  |  |
| --- | --- | --- |
| Home Workstation | Work Desktop | Cycle Server |
| Intel 4th Gen. i7-4690k @ 4.0GHz | Intel 2nd Gen. i7-2600 @ 3.40GHz | Intel Xeon X5650 @ 2.67GHz |
| 16 GB DDR3 – 2400MHz | 8 GB DDR3 – 1333 MHz | ~ 23 GB – (Unauthorized Info) |
| Windows 10 (Linux Mint VM) | Ubuntu 14.04 LTS | Fedora 22 (Server) |
| 494.5286 Seconds Total | 651.9222 Seconds Total | 675.9946 Seconds Total |

For collected data see *‘Raw Data with Averages’*.

Observation and conclusion

Both *Leftist* and *Skew* concatenated queue’s exhibit pseudo linear behavior for low operation counts. However, as the number of operations increases, the behavior of both structures starts to deviate from the linear trend presumably due increasing numbers of swap and rank calculations. For low operation values, the raw insertion times for both data structures is nearly identical. Though the leftist heap clearly starts to outperform the skew heap as insertion count increases. Presumably, the continual swapping might bog down the skew tree implementation with large data sets. However, the random operations test resulted in no such difference in trends.

Raw Data with Averages

Leftist Raw Insertions



Skew Raw Insertions



Leftist Raw Insertions



Skew Random Operations



Graphical Results