Assignment 4 MTL458

AVL Tree

Implementation:

In our implementation, we have decided on the following approach: We allow as many read operations as concurrently as needed (contains, in order), and we allow only one write operation at a time (insert, delete). Also, whenever we encounter a delete operation, we ensure that all previously encountered inserts are finished.

We have ensured that no deadlocks occur and there are no race conditions. We have also ensured that there is no starvation.

Comparison:

Comparing the behaviour of a concurrent AVL tree implementation with a traditional (non-concurrent) AVL tree is essential to understand the advantages and trade-offs of each approach. We discuss these below:

1. Non-Determinant Behaviour

Due to concurrency in operations, the structure of the AVL Tree might change drastically, and the in-order and pre-order traversals might be vastly different. This is expected but sometimes not encouraged.

2. Performance Metrics:

	Concurrent AVL Tree	**Traditional AVL Tree**
Insertion performance	may experience increased overhead due to synchronization, especially in high-contention scenarios. Performance may be affected by the need for locks or atomic operations.	doesn't have the added overhead of synchronization. It typically performs well in insertion operations.
Deletion performance	may experience additional overhead during deletion. The synchronization mechanisms can impact performance.	may perform deletion operations more efficiently, as it doesn't have to deal with synchronization concerns.
Search(contains) performance	Both implementations are likely to have similar performance in search operations, as no write operations are involved, and synchronization overhead is minimal.	
In-order Traversal performance	the performance may vary depending on the synchronization mechanisms used. Contention among threads may lead to suboptimal performance.	can typically perform in-order traversal efficiently as it doesn't have to deal with concurrency-related challenges.
Overall performance	Multiple concurrent reads might result in a good overall performance, but overhead introduced might counter it.	-

3. Memory usage:

Concurrent AVL Tree	Traditional AVL Tree
Node Overhead: tree may have additional memory overhead per node due to synchronization mechanisms. This overhead can include locks, flags, or other data required to manage concurrent access. The memory used per node will depend on the synchronization method chosen.	**Node Overhead:** trees have a smaller memory overhead per node because they don't require synchronization mechanisms. They only contain the essential components, such as key, value, and pointers to children.
Additional Data Structures: might uses additional data structures to manage concurrent operations, such as thread-specific data, contention management structures, or atomic variables for synchronization. These data structures can consume extra memory.	**Simplicity:** simplicity can lead to less memory overhead. There are no additional data structures or atomic variables to manage concurrent access.
Node Payload: stores the same data payload (keys, values, etc.) as the traditional AVL tree, so the memory usage related to the payload itself will be the same.	

Examples:

1.

Input	Output	Performance
insert 3 insert 4 insert 2 delete 2 contains 3 in order exit	 yes 3 4 3 4	Time with concurrency: 0.015 s Time without concurrency: 0.016s

2.

Input	Output	Performance
insert 5 insert 10 insert 15 insert 25 insert 30 insert 14 insert 23 insert 6 insert 24 insert 30 insert 3 in order delete 23 contains 23 in order insert 23 delete 36 insert 37 in order insert 36 delete 37 contains 37 in order exit	3 5 6 10 14 15 23 24 25 30 no 3 5 6 10 14 15 24 25 30 3 5 6 10 14 15 23 24 25 30 37 no 3 5 6 10 14 15 23 24 25 30 36 15 10 5 3 6 14 25 24 23 36 30	Time with concurrency: 0.020 s Time without concurrency: 0.018s

Thus, we can see that there is marginal difference in this case for AVL Trees implemented using concurrency, limited to testing conditions.

Suggested Improvements: We might consider implementing locks per node rather than a lock for the whole tree. However, as always, it also adds further complexity at the cost of concurrency.