Zadanie 5 – Vyhľadávanie v dynamických množinách

Dmytro Shapovalov

I have implemented three data structures designed for efficient searching in dynamic sets:

Weight-Balanced Binary Search Tree (WBT) – a binary search tree that maintains balance based on the size (weight) of its subtrees. The **weight** of a node refers to the number of nodes (or size) in the subtree rooted at that node. The tree maintains balance by ensuring that the ratio of weights between left and right subtrees remains within certain limits "(rightWeight * 5 + 2 < leftWeight * 2)" and "(leftWeight * 5 + 2 < rightWeight * 2)".

Red-Black Tree (RBT) – a self-balancing binary search tree with all nodes either red or black in color. Red-Black tree maintains balance based on 5 properties:

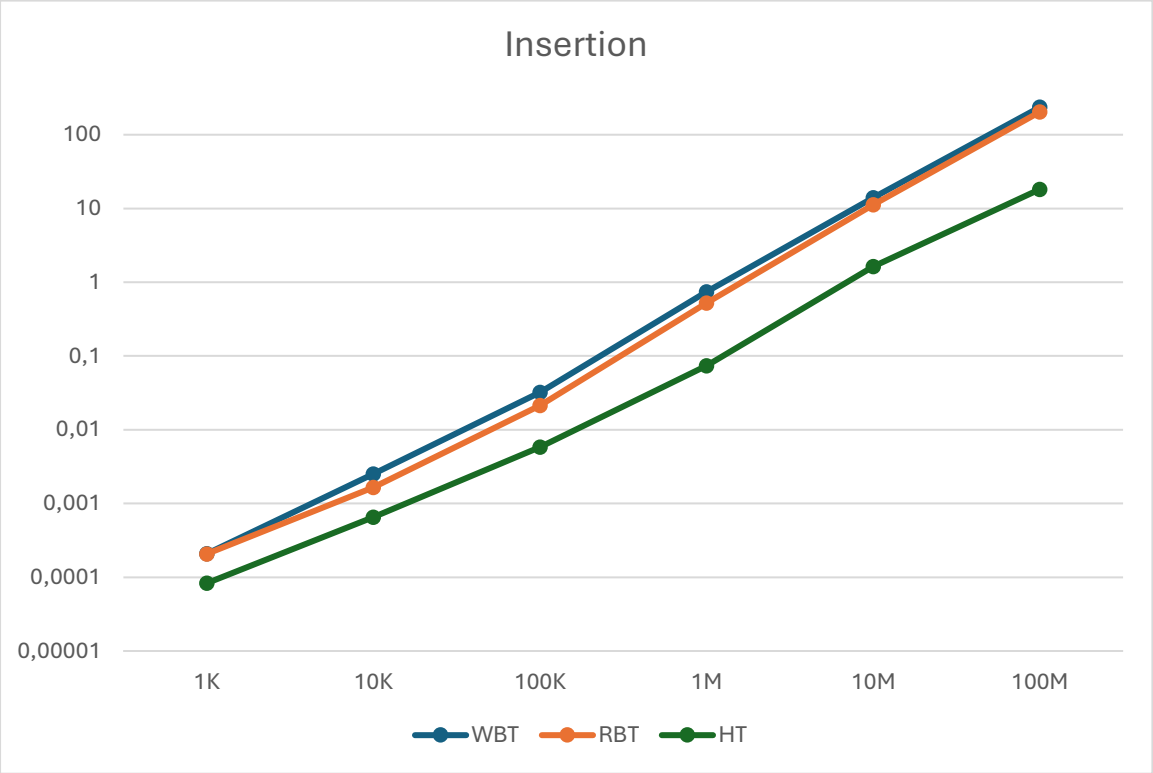
1. Each node is either red or black.
2. The root is always black.
3. All leaves (NIL nodes) are black.
4. Red nodes cannot have red children.
5. Every path from a node to its descendant leaves contains the same number of black nodes.

Hash Table with Open Addressing and Double Hashing – a hash-based structure where collisions are resolved using a secondary hash function. The table dynamically resizes when the load factor exceeds a certain threshold. In open addressing, when a collision occurs, the algorithm searches for the next free position within the table. Double hashing uses two hash functions to reduce collision probability:

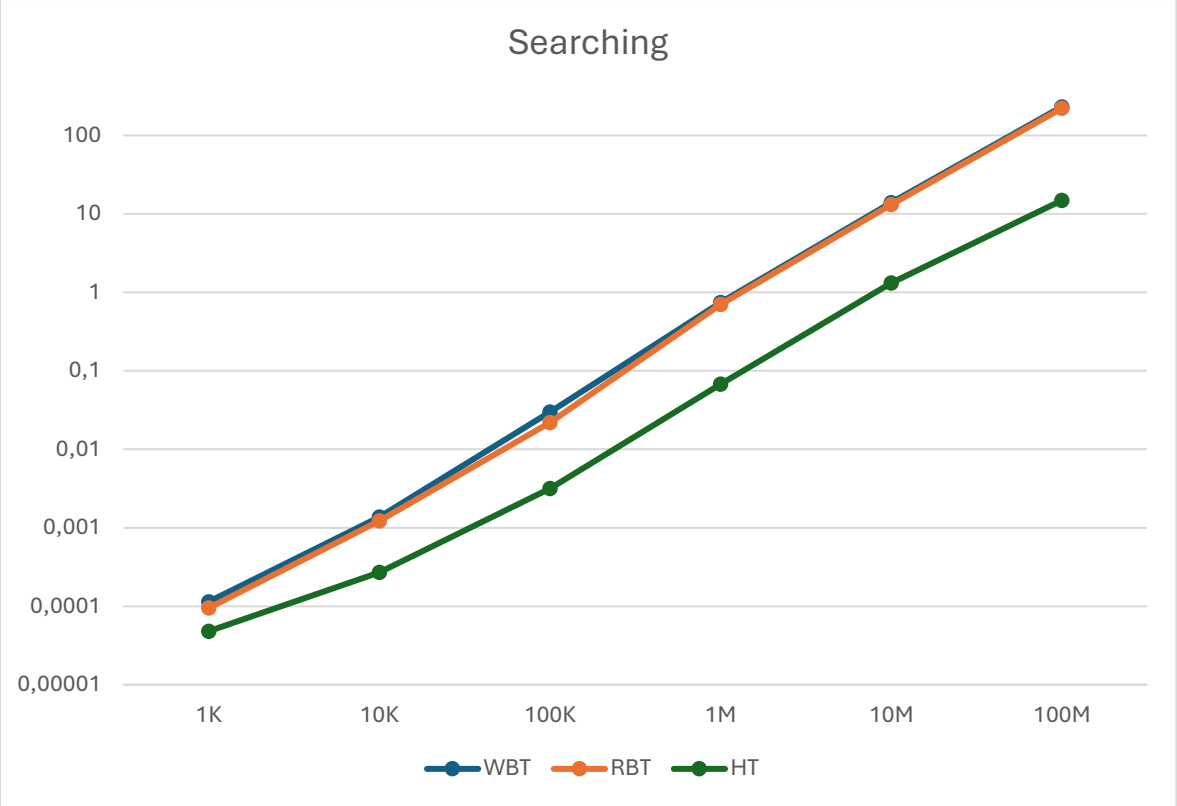
1. First Hash Function: id % HT->size;
2. Second Hash Function: HT->prime - (id % HT->prime)
   *prime is nearest to Hash Table size prime number

Testing:

I did two kinds of tests. In the first one, first N insert operations are performed, then N search operations are performed and finally N delete operations are performed. In the second, N random operations are performed. These can be insert, search and delete.



	WBT	RBT	HT
1K	0,000208	0,000207	0,000083
10K	0,002515	0,001659	0,000655
100K	0,032092	0,021239	0,005803
1M	0,744447	0,521769	0,07341
10M	14,052308	11,246472	1,638722
100M	235,774927	203,515463	18,20154

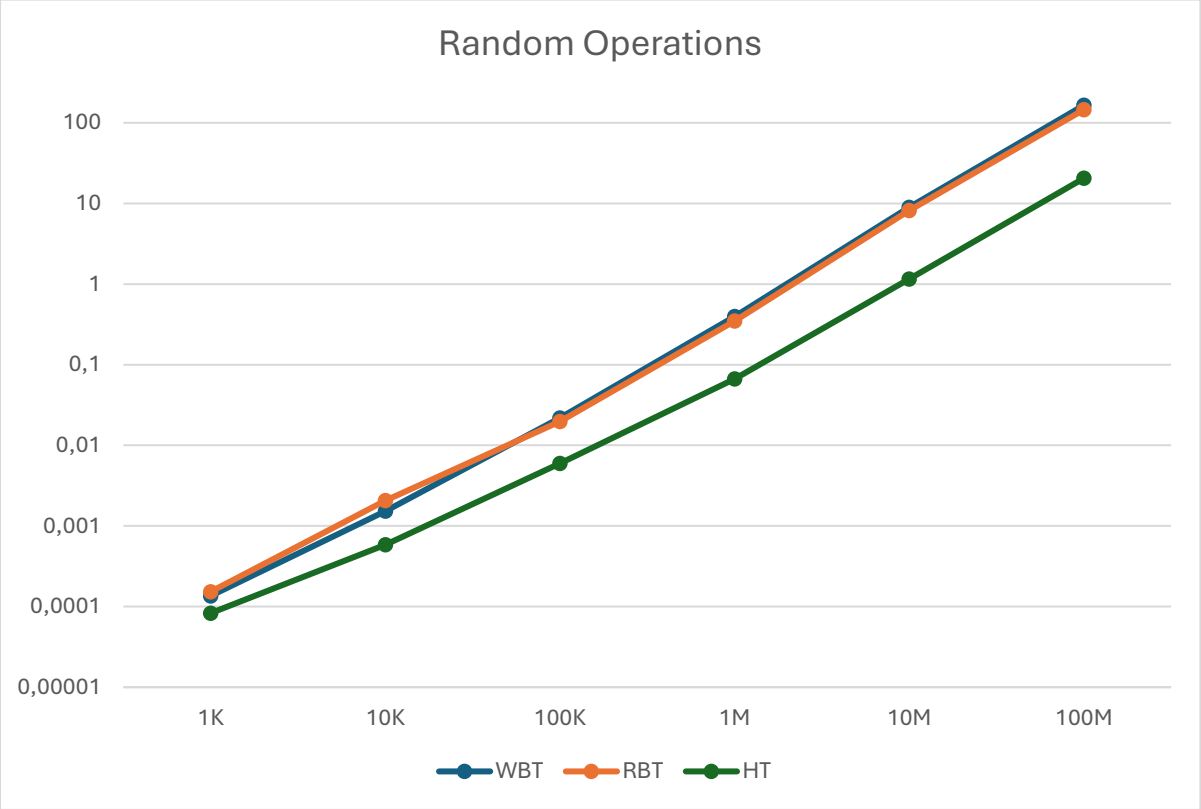


	WBT	RBT	HT
1K	0,000114	0,000095	0,000048
10K	0,001365	0,001229	0,000271
100K	0,029939	0,021939	0,003152
1M	0,738458	0,699242	0,067299
10M	13,933675	13,11571	1,31469
100M	231,867793	221,759477	14,858911

	WBT	RBT	HT
1K	0,000134	0,000181	0,00005
10K	0,002139	0,001665	0,000276
100K	0,022569	0,022484	0,004191
1M	0,684377	0,708802	0,063811
10M	13,880621	13,24474	1,26021
100M	232,665061	228,477019	15,363152

Individual functions test: a comparative analysis of HashTable (HT), Red-Black Tree (RBT), and Weight-Balanced Tree (WBT) across insertion, search, and deletion operations (1000 to 100 million operations), HashTable demonstrated superior speed in all scenarios due to its average **O(1) time complexity**, enabling near-instantaneous access even at large scales. For example, at 1 million operations, HT completed insertions in 0.1s, while RBT and WBT required 1s. This efficiency stems from HT's direct key-to-index mapping and absence of balancing overhead, though its performance slightly degrades beyond 10 million operations due to collision resolution and rehashing.

Red-Black Tree (RBT) and Weight-Balanced Tree (WBT) demonstrate **comparable performance** across insertion, deletion, and search operations, with execution times aligning closely in practical tests. Both structures operate within the **O(log n)** complexity class, resulting in similar scalability and efficiency for most workloads. Their use may be justified in cases where data ordering or strict logarithmic access time guarantees are required, but from the point of view of pure speed Hash Table is the optimal choice.



	WBT	**RBT**	**HT**
1K	0,000134	0,000181	0,00005
10K	0,002139	0,001665	0,000276
100K	0,022569	0,022484	0,004191
1M	0,684377	0,708802	0,063811
10M	13,880621	13,24474	1,26021
100M	232,665061	228,477019	15,363152

Random Operations test: HashTable (HT) consistently outperforms RedBlackTree (RBT) and WeightBalancedTree (WBT) across all tested scales (1000–100M operations) due to its average **O(1)** time complexity. For small to medium datasets (1000–1M operations), HT is fastest, though collisions and rehashing slightly slow it at large scales (10M–100M) making it ideal for big static data sets. RedBlackTree and WeightBalancedTree provide predictable **O(log n)** performance, making it ideal for dynamic datasets requiring frequent updates and sorted data. But RedBlackTree a little bit faster. WeightBalancedTree is the slowest, especially beyond 10M and 100M operations, due to high balancing costs.

Conclusion:

Hash Table is the fastest and easiest to implement data structure among the tested ones. It is well suited for tasks where storage order is not important and performance is paramount. Red-Black Tree (RBT) and Weight-Balanced Tree (WBT) have their own important advantages: Guarantee logarithmic worst-case execution times, Maintain data ordering, which is important for range queries, Do not depend on the choice of hash function and avoid collision problems.

For maximum speed and simplicity, choose Hash Table. For tasks where structured, sorted and predictable behavior is needed - RBT or WBT are better, and you can choose between them based on your implementation preferences, as their behavior in the tests was almost the same.