# BLACK MAMBA
# AN ADVANCED DATA STRUCTURES LIBRARY

BENNING, LUKE

## 1. Structure List

• General : Union Find, Fenwick Tree
• Trees : AVL (Adelson-Velsky and Landis) Tree, Red Black Tree, Splay Tree, Treap

## 2. General

Black Mamba contains multiple data structures that do not fit into any specific category of data structures, such as lists, heaps or trees. Currently there are two general structures, Union Find and Fenwick Trees.

• Union Find - Used for implementing minimum spanning tree algorithms such as Kruskal's and Prim's.

• Fenwick Tree (Binary Indexed Tree) - Used for tracking and updating prefix sums.

## 3. Trees

Black Mamba contains a multitude of self-balancing / self-adjusting binary search trees (BST's).

3.1. **Binary Search Trees.** The four supported BST's are AVL, Red-Black, Splay and Treap. Each tree implements the interface MambaTree, which details the core operations that the trees support. The abstraction of core operations to an interface was done to support the following two objectives:

• Allow access to the node structure of the tree such that functions can be written to traverse the tree and examine the key-value pairs to compute things such as tree height, tree width etc.

• Prevent illegal modification of the tree structure to ensure the integrity of the data structure and correctness of the supported operations. While the trees may be traversed through the interface operations, the nodes themselves may not be modified directly. Tree data may only be modified by executing the core operations from the root node of the tree.

These choices are different from the two implementation extremes, where either an implementation exposes the node structure and trusts the user to not break references, violate BST property etc., or completely abstracts the nodes into an inner private class of a tree. The former choice satisfies objective one but violates objective two. Likewise the second choice satisfies objective two but violates objective one. With the chosen implementation, both objectives are satisfied.

Each of the four BST's perform well in certain contexts, and the tree implementation should be chosen based on the application. The following is a brief overview of the situations where each tree excels.

• Splay trees are commonly seen in online environments where they can take advantage of locality of information. Every key-value pair that is inserted or looked up will be propagated up to the tree root via a series of tree rotations. If a key-value pair is looked up multiple times in succession, then after the first lookup the key-value pair will be located at/close to the root, and be quickly found in subsequent look-ups.

• AVL trees are used in situations where the number of look-ups significantly exceeds the number of insertions or deletions. They maintain the balanced property of BST's, which is that the maximum difference in length between two paths from the root node to a leaf is one. Maintaining this property induces overhead, but results in a strictly balanced tree allowing for fast look-ups.

• Red-Black trees are used when the number of insertions and deletions is greater than the number of look-ups. They are generally not as balanced as AVL trees and maintain a more loose requirement on the tree structure. This makes insertions and deletions less costly, however look-ups might be slower than AVL trees due to the more imbalanced structure.

• Treaps work well in general contexts since they in expectation form a roughly balanced tree due to randomized priorities. If no prior knowledge of the frequency of operations is available, then choosing Treaps can be a reasonable choice.

3.2. **Specialized Trees.**