AVL Tree Library Reference Documentation

Authors: Eric Biggers

Overview

The avl_tree library provides an implementation of a nonrecursive, self-balancing binary search tree known as an AVL tree. The library is designed for use in C, offering efficient tree operations while minimizing recursion through an intrusive node structure.

Key Components

Structures

- **struct avl_tree_node**
- Represents a node in an AVL tree, designed to be embedded within a parent data structure for intrusive integration. It contains:
    - left: Pointer to the left child or NULL.
    - right: Pointer to the right child or NULL.
    - parent_balance: Combines the parent pointer with a balance factor, saving memory space and tracking the balance of the subtree.

Macros

- **avl_tree_entry(entry, type, member)**
- Casts an AVL node pointer to the containing data structure type. Useful for accessing the parent data structure.

Inline Functions

- **avl_get_parent(node)**
- Returns the parent of the specified AVL node or NULL if it is the root.

- **avl_tree_node_set_unlinked(node)**
- Marks a node as unlinked from any tree, setting its parent_balance.

- **avl_tree_node_is_unlinked(node)**
- Checks if the node is marked as unlinked.

Functions

- **avl_tree_lookup(root, cmp_ctx, cmp)**
- Searches for an item in the specified tree using a custom comparison function. Returns the matching AVL node or NULL if not found.

- **avl_tree_lookup_node(root, node, cmp)**
- Searches for a node in the tree using a node-to-node comparison function.

- **avl_tree_insert(root_ptr, item, cmp)**
- Inserts an item into the tree. If a duplicate item exists, it returns the existing item node; otherwise, it returns NULL.

- **avl_tree_remove(root_ptr, node)**
- Removes a specified item from the tree and rebalances.

- **avl_tree_first_in_order(root)**
- Returns the smallest node in order.

- **avl_tree_last_in_order(root)**
- Returns the largest node in order.

- **avl_tree_next_in_order(node)**
- Returns the next node in order.

- **avl_tree_prev_in_order(node)**
- Returns the previous node in order.

- **avl_tree_first_in_postorder(root)**
- Begins a postorder traversal with the first node.

- **avl_tree_next_in_postorder(prev, prev_parent)**
- Continues a postorder traversal to the next node.

Macros for Iteration

- **avl_tree_for_each_in_order(...)**
- Macro to iterate over tree nodes in sorted order. Modification of the tree during iteration is not allowed.

- **avl_tree_for_each_in_reverse_order(...)**
- Similar to avl_tree_for_each_in_order, but in reverse order.

- **avl_tree_for_each_in_postorder(...)**
- Iterates over nodes in postorder, allowing node deletion.

Example Usage

Definition of a custom structure with AVL Node:

```
struct int_wrapper {
    int data;
    struct avl_tree_node index_node;
};
```

Inserting into the AVL Tree:

```
bool insert_int(struct avl_tree_node **root_ptr, int data) {
    struct int_wrapper *i = malloc(sizeof(struct int_wrapper));
    i->data = data;
    if (avl_tree_insert(root_ptr, &i->index_node, _avl_cmp_ints)) {
        free(i);
        return false;
    }
    return true;
}
```

Additional Details

- **Rebalancing**: Functions such as avl_tree_rebalance_after_insert ensure that the AVL tree remains balanced after alterations.
- **Traversal**: Use provided functions and macros to traverse the tree efficiently without recursion.