Esmeralda Jardine

2391491

Search

Firstly the base case is set - if there is no node found after recursive searching then it will reach a null pointer which sets the node to NULL, this is what is returned.

Starting from the root, the function compares the toFind value with the current node's value. If the value matches, the function returns the current node. If toFind is smaller, the search traverses the left subtree, otherwise it traverses the right subtree. This function appears to implement the desired behaviour correctly.

Generation

The createTree function initializes a new binary search tree with a single root. It allocates memory for the root and sets its left and right children to NULL. The function returns NULL if the allocation is not successful.

The destroyTree function deallocates all memory associated with the tree. It uses post-order traversal to recursively free each (left then right) subtree before freeing the current node.

Maintenance

The insert function identifies places for a new element by recursively navigating the tree. Elements smaller than the current node move the next comparison to the left child, otherwise it moves to the right child. If the child is NULL, a new node is allocated and the value is updated with the elements value. Duplicate values cannot be added as the return statement in the base case covers this.

The delete function traverses the tree to find the node with the specified value (elem), it keeps track of the parent node and whether the current node is a left child. If the node cannot be found, the function returns without making any changes.

1. Leaf Nodes: The node is removed, and its parent's left and right pointers are set to NULL. Function also handles
2. Nodes with One Child: The child replaces the node in the tree structure and the updates the parent's pointer point to the child.
3. Nodes with Two Children: the function finds the smallest node in the right subtree and uses it to replace the value of the node to be deleted, it then deletes the in-order successor node.

Memory Management

The functions rely on malloc for dynamic memory allocation and free for deallocation. Each function that uses malloc has a condition to check if the allocation was successful before performing the next task to prevent segmentation faults. Dangling pointers are avoided by setting the parent of freed pointers to null. In createTree and insert, memory is allocated using sizeof(), preventing memory wastage. The destroyTree function ensures no memory leaks occur by freeing in a post-order fashion.

Issues

The code does not handle root deletion very well - it fails to update the root pointer in the calling context, which could lead to undefined behaviour and segmentation faults. Memory leaks can happen when the root is replaced by one child as the original root is not properly freed. Separating root-specific logic and using a double pointer (node_t **) for the root could potentially address these issues.