CA208 Prolog Assignment 2 – Implementing AVL Trees

Ailbhe Byrne

19424402

I acknowledge the DCU academic integrity policy.

Predicates

bt(\_, \_, \_).

* This is a predicate for a binary tree. It takes 3 parameters: root, left child and right child, where all the items in the left child are smaller than the root and all the items in the right child are bigger than the root.

addnode(X, T1, T2).

* This predicate adds an item to a binary tree. This is true if adding X to T1 gives T2. It adds the item in the right place numerically but doesn’t rebalance the tree to maintain an AVL tree. This predicate came from the CA208 prolog notes.

height(T, H).

* This gets the height of a binary tree, it is true if H is the height of tree T. It works recursively by going through the tree and adding 1 for each level. The biggest number gotten from going through the levels of the tree is the height.

bfactor(T, B).

* This predicate gets the balancing factor of a tree. This is true if B is the balancing factor of the tree T. The balancing factor is the height of the left subtree – the height of the right subtree.

ll(T). rr(T). rl(T). lr(T).

* These predicates match each kind of tree imbalance that can happen to a binary tree. If a tree has a left-left imbalance, the balancing factor of the tree will be 2 and the balancing factor of the left child will be either 1 or 2. If a tree has a right-right imbalance, the balancing factor of the tree will be -2 and the balancing factor of the right child will be either -1 or -2. If a tree has a right-left imbalance, the balancing factor of the tree will be -2 and the balancing factor of the right child will be either 1 or 2. If a tree has a left-right imbalance, the balancing factor of the tree will be 2 and the balancing factor of the left child will be either -1 or -2.

ll rr rl lr

5 2 2 5

/ \ \ /

3 4 5 2

/ \ / \

1 5 3 4

ll\_rebalance(T1, T2). rr\_rebalance(T1, T2). rl\_rebalance(T1, T2). lr\_rebalance(T1, T2).

* These predicates are used to rebalance a tree after it comes unbalanced when adding an item. This is true if T1 becomes rebalanced by changing to T2. This predicate maintains an AVL tree when adding items to the tree.

insert(X, T1, T2).

* This predicate adds an item to an AVL tree and then rebalances the tree after the item is added to maintain an AVL tree. It uses addnode to first add the item in the correct spot numerically, then checks for each kind of imbalance and if it matches the imbalance, it applies the correct rebalance. This is true if adding X to the AVL tree T1 results in the AVL tree T2.

display(T).

* This predicate prints the tree. It is always true. It works recursively by going through the left subtree, printing the item and then going through the right subtree. This is in-order traversal. It also gets the height of the tree for printing each item.

printel(X, H).

* This predicate prints an item in a tree. It prints a tab 3 times the height of the item, prints the item and then prints a new line. The tree is read from right to left.

Text

Description automatically generated with medium confidence