**AVL Tree Assignment**

Student Number: 19452724

Acknowledgement of DCU Academic Integrity Policy:

I, Katie Clancy, acknowledge and accept the terms and conditions of the DCU Academic Integrity Policy.

All work submitted by me in this assignment, except where indicated, is solely my own.

I have read and understood the plagiarism policy found at <https://www.computing.dcu.ie/policies/plagarism-statement> along with what has been recommended in the assignment guidelines.

*Name*: Katie Clancy

*Date*: 18/04/2021

*References*

* ic.unicamp.br

ninety-nine prolog problems

working with prolog lists – rotating a list

\*\*used to assist in understanding and implementing list rotation\*\*

<https://www.ic.unicamp.br/~meidanis/courses/mc336/problemas-prolog/p19.prolog>

* computer-programming-forum.com

AVL Trees – improving an implementation of:

A program for constructing and searching a tree

\*\*used to assist in understanding and implementing the display of a binary tree\*\*

<http://computer-programming-forum.com/55-prolog/05a8f78b7e533897.htm>

*Code Application*

Firstly, I defined my tree predicates, for the empty binary tree and the non-empty binary tree.

For the non-empty tree, I ensured the elements of the left subtree remain less than or equal to the root node, and for the right subtree the elements will remain greater than the root node.

After defining the tree predicates, I implemented the insert predicates.

I defined insert(I, T1, T2) as true by inserting I into the tree T1, making a new tree T2.

I then implemented height checking to check for the height of the tree which will determine if it is balanced or not.

Firstly, the tree is checked to see if it is empty.

Then, the tree is checked for an element count of 1.

After both of these checks are bypassed, the height can be determined by getting the height of either the right or left branches and adding the root node.

If the tree is unbalanced, I implemented rotations to rebalance it, both for left and right subtrees.

The rotations move my either N places to the left or right.

Finally, I implemented display, which prints the binary tree.

*Code*:

avl.pl

# binary tree insert and display

# tree predicates

check\_tree(nil).

check\_tree(tree(\_, LeftN, RightN)) :- check\_tree(leftN), check\_tree(RightN).

empty = nil.

bt(tree(\_, nil, nil)).

bt(tree(N, LeftN, nil)) :- LeftN <= N.

bt(tree(N, nil, RightN)) :- N < RightN.

bt(tree(\_, LeftN, RightN)) :- bt(LeftN), bt(rightN).

bt(tree(N, LeftN, RightN)) :- LeftN <= N, N < RightN.

# insert predicates

insert(I, empty, bt(I, empty, empty)).

insert(I, bt(I, T1, T2), bt(I, T1, T2)) :- !.

insert(I, bt(N, T1, T2), bt(N, LeftN, T2)) :- I < N, !, insert(I, T1, LeftN).

insert(I, bt(N, T1, T2), bt(N, L, RightN)) :- insert(I, T2, RightN).

# checking height for balance

high(A, B, A) :- A > B, !.

high(\_, B, B).

height(empty, 0).

height(bt(N, empty, empty), 1).

height(bt(\_, T1, T2),Ht):- height(T1, HT1), height(T2, HT2), high(HT1, HT2, HT3), Ht is HT3 + 1.

# if unbalanced

# rotate N places to Left

rotate(T1, N, T2) :- N >= 0, length(T1, NT1), N1 is N mod NT1, rotate\_to\_left(T1, N, T2).

rotate(T1, N, T2) :- N < 0, length(T1, NT1), N1 is NT1 + (N mod NT1), rotate\_to\_left(T1, N1, T2).

rotate\_to\_left(T, 0, T).

rotate\_to\_left(T1, N, T2) :- N > 0, spl(T1, N, S1, S2), append(S2, S1, T2).

# rotate N places to Right

rotate(T1, N, T2) :- N >= 0, length(T2, NT2), N2 is N mod NT2, rotate\_to\_right(N, T1, T2).

rotate(T1, N, T2) :- N < 0, length(T2, NT2), N2 is NT2 + (N mod NT2), rotate\_to\_right(N1, T1, T2).

rotate\_to\_right(T, 0, T).

rotate\_to\_right(N, T1, T2) :- N > 0, spl(T2, N, S2, S1), append(S1, S2, T1).

# print the tree

display(tree) :- displays(tree, 0).

displays(nil / 0, \_).

displays(bt(LeftN, I, RightN) /\_, Indent) :- Indnt is Indent + 2, displays(RightN, Indnt), tab(Indent), write(I), nl, displays(LeftN, Indnt).