COL226: Programming Languages
II semester 2021-22

Assignment: Integer Binary (Search) Trees in Prolog

This is an assignment meant to get you to learn to define various things related to binary trees axiomatically through rules in Prolog. It is also a way of axiomatising a data-structure, while encouraging sound and efficient programming through axioms and rules inductively.

We define an integer binary tree in Prolog as follows:

```
ibt(empty).
ibt(node(N, L, R)):- integer(N), ibt(L), ibt(R).
```

In the sequel parameters of predicates have adhere to certain types. The names of the parameters of the parameter variables in the predicates have been chosen as follows.

- `N` is an integer.
- `S` is a string of printable characters.
- `L` is a list of integers.
- `BT`, `LBT` and `RBT` are all integer binary trees.
- `BST`, `BST1` and `BST2` are all integer binary search trees.

Now define the following predicates in Prolog.

1. `size(BT, N)`. `N` is the number of integer-labelled nodes in `BT`.
2. `height(BT, N)`. `N` is the height of `BT` with `empty` having a height of 0.
3. `preorder(BT, L)`. `L` is the preorder traversal of `BT`.
4. `inorder(BT, L)`. `L` is the inorder traversal of `BT`.
5. `postorder(BT, L)`. `L` is the postorder traversal of `BT`.
6. `trPreorder(BT, L)`. `L` is a *tail-recursive* preorder traversal of `BT`.
7. `trInorder(BT, L)`. `L` is a *tail-recursive* inorder traversal of `BT`.
8. `trPostorder(BT, L)`. `L` is a *tail-recursive* postorder traversal of `BT`.
9. `eulerTour(BT, L)`. `L` is the Euler tour of `BT`. See figure 1 for an example of an Euler tour.
10. `preET(BT, L)`. `L` is the preorder traversal of `BT` extracted from the Euler tour.
11. `inET(BT, L)`. `L` is the inorder traversal of `BT` extracted from the Euler tour.
12. `postET(BT, L)`. `L` is the postorder traversal of `BT` extracted from the Euler tour.
13. `toString(BT, S)`. `S` is the string representing a *linear* rendering of a nonempty binary tree `BT` as a 3-tuple of the form "`(N, LBT, RBT)`" where `N` is the label of the root and `LBT` and `RBT` are the left and right subtrees respectively. An empty tree is rendered as "`()`".
14. `isBalanced(BT)`. Whether `BT` is balanced.

15. `isBST(BT)`. Whether `BT` is a binary search tree under the usual $<$ ordering on integers.
16. `makeBST(L, BST)`. `BST` is a *balanced* binary search tree whose nodes have labels from `L`.
17. `lookup(N, BST)`. Determine whether there is a node labelled `N` in `BST`.
18. `insert(N, BST1, BST2)`. *Insert* a new node labelled `N` in `BST1` to yield `BST2` provided there is no node labelled `N` already in `BST1`.
19. `delete(N, BST1, BST2)`. Delete a node labelled `N` (if it is present) from `BST1` to yield `BST2`.

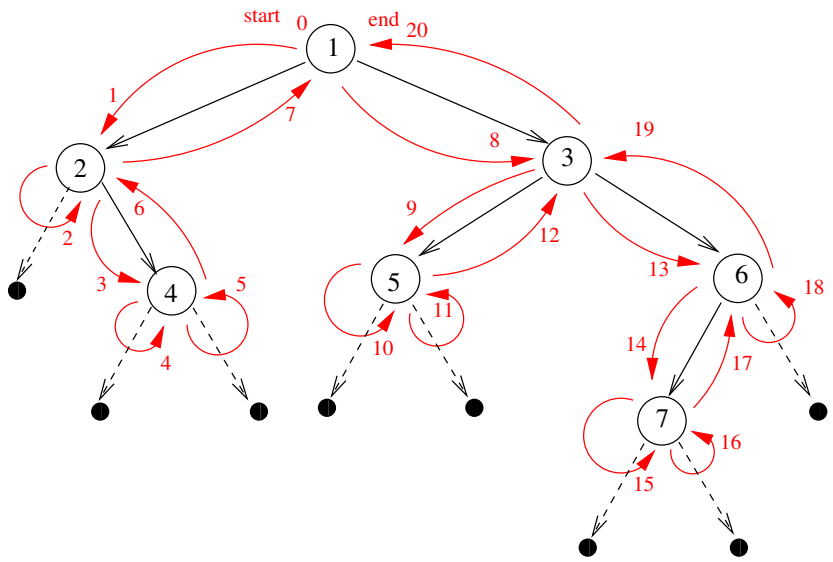

Figure 1: Euler tour: Starting from the root make a tour around the whole tree in a counter-clockwise fashion so that the nodes being visited are always on the left as you walk around the tree visiting each node.

Note

1. You are *not* allowed to change any of the names or types given in the specification/signature. You are not even allowed to change upper-case letters to lower-case letters or vice-versa.
2. The evaluator may use automatic scripts to evaluate the assignments, especially when the number of submissions is large.
3. You may define any new auxiliary functions/predicates if you like in your code besides those mentioned in the specification.
4. Your program should implement the given specifications/signature.
5. You need to think of the *most efficient way* of implementing the various functions given in the specification/signature so that the function results satisfy their definitions and properties.
6. The evaluator may look at your source code before evaluating it, you must explain your algorithms in the form of comments, so that the evaluator can understand what you have implemented.
7. Do *not* add any more decorations or functions or user-interfaces in order to impress the evaluator of the program. Nobody is going to be impressed by it.

8. There is a serious penalty for code similarity (similarity goes much deeper than variable names, indentation and line numbering). If it is felt that there is too much similarity in the code between any two persons, then both are going to be penalized equally. So please set permissions on your directories, so that others have no access to your programs.