Challenge II: DLL to BST

Description

This challenge is to verify an in-place algorithm to convert a sorted doubly-linked list (DLL) into a balanced binary search tree (BST). The algorithm runs in linear time, and proceeds by first making a pass over the list to calculate its length, say n. It then recursively constructs a BST out of the first $n/2$ nodes, makes this the left subtree of the middle node, and then recursively constructs the right subtree using the remaining nodes.

Assume that each node has 3 fields, `data`, `prev`, and `next`, and when constructing the BST, we use the `prev` field to link a node to its left child and the `next` field to link a node to its right child.

The code for this algorithm is given in Figure 1.

Verification Tasks

1. Prove that this algorithm converts an input list into a tree.
2. Prove that the algorithm is memory-safe.
3. Prove that if the input list is sorted then the resulting tree is a BST.
4. Prove that the resulting BST is balanced.
5. Prove that the algorithm terminates.
6. (Optional) Prove the above for an iterative version of `size`.

```
// Ref is the type of nodes used for both list and tree, and has these fields:
field data: Int
field prev: Ref // Also used as left subtree pointer
field next: Ref // Also used as right subtree pointer

method size(head: Ref) returns (count: Int) {
  if (head != null) {
    count := size(head.next)
    count := count + 1
  } else {
    count := 0
  }
}

method dll_to_bst(head: Ref) returns (root: Ref) {
  var n: Int
  var right: Ref
  n := size(head)
  root, right := dll_to_bst_rec(head, n)
}

// Converts a sorted DLL into a balanced BST
//   head: Pointer to doubly linked list
//   n: number of nodes of list to convert to tree
method dll_to_bst_rec(head: Ref, n: Int) returns (root: Ref, right: Ref) {
  if (n > 0) {
    // Recursively construct the left subtree
    var left: Ref
    left, root := dll_to_bst_rec(head, n/2)
    // [head, root) is a tree rooted at left, [root, ...] is a list

    // Set pointer to left subtree
    root.prev := left

    // Recursively construct the right subtree
    // size(right subtree) = n - size(left subtree) - 1 (for root)
    var temp: Ref
    temp, right := dll_to_bst_rec(root.next, n-n/2-1)
    // [head, root) is a tree rooted at left, [root.next, right) is tree at temp

    // Set pointer to right subtree
    root.next := temp
    // [head, right) is a tree rooted at root
  } else {
    root := null
    right := head
  }
}
```

Figure 1: Code for the algorithm that converts a sorted DLL into a balanced BST.