Is It A Red-Black Tree

Li Yunfan, 3200102555

November 15, 2023

1 Introduction

Red-Black Trees (RBTs) are a type of self-balancing binary search tree, widely used in computer science due to their ability to maintain balanced height during insertions and deletions. This characteristic allows for operations such as search, insert, and delete to occur in logarithmic time, making RBTs highly efficient for associative arrays and implementing other abstract data types.

The objective of this project is to devise a program in C that can verify whether a given binary tree satisfies the properties of a Red-Black Tree.

2 Algorithm Specification

The algorithm employed in this project is designed to validate a binary tree against the properties of a Red-Black Tree (RBT). An RBT is a binary search tree where each node has an additional color attribute, red or black. The color attribute is used to ensure the tree remains approximately balanced, such that it guarantees searching in logarithmic time. The properties that define a valid RBT are as follows:

1. Each node is either red or black.
2. The root node is black.
3. All leaves (NIL nodes) are black.
4. If a node is red, then both of its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

To verify these properties, the algorithm examines the preorder traversal of the tree. Preorder traversal is a method of tree traversal that starts at the root and explores the nodes from the left to the right. Thus, for a given node, it processes the current node before its child nodes. In the context of this problem, the preorder sequence is provided as input, with negative values indicating red nodes and positive values indicating black nodes.

The core function, 'judge', operates recursively and performs several checks at each node:

- It verifies that the root node is black by checking the color of the first node in the traversal sequence.
- It checks whether the current node is a leaf and, if so, confirms it is black.
- For non-leaf nodes, it ensures that if a node is red, its immediate children are black.
- It recursively checks the left and right subtrees, maintaining a count of black nodes on each path and comparing these counts to ensure they are equal.

The pseudocode provided in the subsequent section outlines these steps in detail. Upon invoking the 'judge' function with the root node, it will traverse the tree in preorder, applying these rules. If at any point a violation is detected, the function will return a value indicating the tree is not a valid RBT. If the traversal is completed without detecting any violations, the tree is declared a valid RBT.

Negative values in the input represent red nodes, while positive values represent black nodes.

2.1 Pseudocode

```
function isRBT(node):
  if node is root and red:
    return false
  if node is leaf and red:
    return false
  if node is red and has red child:
    return false
  lcount = isRBT(node.left)
  rcount = isRBT(node.right)
  if lcount != rcount or lcount == 0 or rcount == 0:
    return false
  return lcount + (node is black)
end function
```

3 Testing Results

The implemented algorithm was subjected to a series of tests using predefined binary trees. The testing aimed to assess the accuracy of the `judge` function across various tree structures.

The results are as follows:

Test Case	Preorder Sequence	Result
1	7 -2 1 5 -4 -11 8 14 -15	Yes
2	11 -2 1 -7 5 -4 8 14 -15	No
3	10 -7 5 -6 8 15 -11 17	No

4 Analysis and Comments

The `judge` function exhibits a time complexity of O(n), as it must visit every node in the tree to validate its color and the properties of its children. The space complexity is also O(n) due to the recursion stack that could, in the worst case, contain all nodes if the tree degenerates into a linked list.

The algorithm performed effectively with the given test cases, although it could be optimized further by implementing tail recursion or iterative approaches to reduce the stack space usage.

5 Appendix: Source Code (in C)

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define MAX_NODES 30

int pre[MAX_NODES];

// Function to take absolute value of an integer
int abs_val(int value) { return value < 0 ? -value : value; }

// Function to judge if the subtree is a red-black tree
int judge(int l, int r) {
  // Root is red
  if (l == 0 && pre[l] < 0)
    return 0;
  // Leaf
  if (l == r) {
    if (pre[l] > 0)
      return 1;
    else
      return 0;
  }
  if (l > r)
    return 0;
  int i = l + 1;
  while (i <= r && abs_val(pre[i]) < abs_val(pre[l]))
    i++;
  // Node is red and its children not all black
  if ((pre[l] < 0) && (pre[l + 1] < 0 || pre[i] < 0))
    return 0;
  int lnum = judge(l + 1, i - 1);
  int rnum = judge(i, r);
  if (lnum == rnum)
    return lnum + (pre[l] > 0 ? 1 : 0);
  else
    return 0;
}

int main() {
  int k, n;
```

```
  scanf("%d", &k);
  for (int i = 0; i < k; i++) {
    scanf("%d", &n);
    for (int j = 0; j < n; j++)
      scanf("%d", &pre[j]);
    if (judge(0, n - 1))
      printf("Yes\n");
    else
      printf("No\n");
  }
  return 0;
}
```

6 Declaration

I hereby declare that all the work done in this project is of my independent effort.