COMP 251: Algorithms and Data Structures - Proofs Assignment

Liam Scalzulli
liam.scalzulli@mail.mcgill.ca

March 13, 2023

Complexity Proof

Claim: Inserting a node x into a red-black tree takes O(log n) time.

Presentation

Proof. To prove that inserting a node x into a red-black tree takes $O(logn)$ time, we need to show that the number of operations performed during the insertion operation is proportional to the height of the tree, which is $O(logn)$ as shown in the correctness proof.

The insertion operation in a red-black tree can be divided into three main steps:

1. **Binary search tree insertion**: This step involves finding the correct position for the new node x in the tree based on its key value, and inserting it as a regular binary search tree node.
2. **Coloring the new node red**: The new node is colored red to preserve the red-black tree properties.
3. **Fixing the red-black tree properties**: If the insertion of the new node causes a violation of the red-black tree properties, then one or more rotations and/or color flips are performed to restore the properties.

Let h be the height of the tree before the insertion operation, and let h' be the height of the tree after the operation. Since step 1 involves a standard binary search tree insertion, the number of comparisons performed during this step is proportional to the height of the tree, i.e., $O(h)$. Therefore, the worst-case time complexity of step 1 is $O(h)$, which is $O(logn)$ in the average case for a balanced tree.

Step 2 involves a constant number of operations, namely setting the color of the new node to red. Therefore, the time complexity of this step is $O(1)$.

Step 3 involves fixing any violations of the red-black tree properties that may have occurred due to the insertion of the new node. The number of operations required to fix these violations is proportional to the height of the subtree affected by the violation. Since the subtree height is at most h+1, the worst-case time complexity of step 3 is $O(h+1)$, which is $O(logn)$ in the average case for a balanced tree.

Therefore, the total worst-case time complexity of inserting a node into a red-black tree is $O(logn)$. This is because step 1 takes $O(logn)$, step 2 takes $O(1)$, and step 3 takes $O(logn)$. Therefore, the total number of operations is proportional to the height of the tree, which is $O(logn)$. □

Source: Slides

Summary

Algorithm

Code explanation

Real world example

Correctness Proof

Claim: Red-Black Trees [CLRS 308]: A red-black tree with n internal nodes has height at most 2 log(n + 1).

Presentation

Proof. We start by showing that the subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal nodes. We prove this claim by induction on the height of x. If the height of x is 0, then x must be a leaf (T.nil), and the subtree rooted at x indeed contains at least $2^{bh(x)} - 1 = 2^0 - 1 = 0$ internal nodes. For the inductive step, consider a node x that has positive height and is an internal node with two children. Each child has a black-height of either $bh(x)$ or $bhx - 1$, depending on whether its color is red or black, respectively. Since the height of a child of x is less than the height of x itself, we can apply the inductive hypothesis to conclude that each child has at least $2^{bh(x)-1} - 1$ internal nodes. Thus, the subtree rooted at x contains at least $(2^{bh(x)-1} - 1) + (2^{bh(x)-1} - 1) + 1 = 2^{bh(x)} - 1$ internal nodes, which proves the claim. □

Source: CLRS

Summary

Algorithm

Code explanation

Real world example