PDS Tutorial Questions

Tutorial/Prep 10

Oct 6, 2024

Information

- Since conceptual/theory questions for similar topics (including BSTs) were given in Quiz-2 practice worksheet, this document comprises only programming questions for CS2710 (as a preparation for Lab 10 on BSTs).

---

1. Review the two recursive implementations of BST insert function seen in class (based on pass by reference or value of pointer to the current Tree Node). Similarly, there are also two recursive implementations of BST delete/remove function, with the pass by reference implementation shown below.

   But the implementation shown below has a bug in one line. Identify the bug and correct it; and write the rest of the class to get a compilable and executable code that can insert and remove elements into the BST.

```
/**
 * Method to remove from a subtree.
 * x is the item to remove.
 * t is the node that roots the subtree.
 * Set the new root of the subtree.
 *
 * Called as BST::remove(key, root)
 */
template <typename Comparable>
void BST::remove( const Comparable & x, BinaryNode * & t )
{
    if( t == NULL )
        return;   // Item not found; do nothing
    if( x < t->element )
        remove( x, t->left );
    else if( t->element < x )
        remove( x, t->right );
    else if( t->left != NULL && t->right != NULL ) // Two children
    {
        t->element = findMin( t->right )->element;
        remove( x, t->right );
    }
    else
    {
        BinaryNode *oldNode = t;
        t = ( t->left != NULL ) ? t->left : t->right;
        delete oldNode;
```

```
        }
    }
```

2. Find the Lowest Common Ancestor (LCA) of two distinct values *key*1 and *key*2 stored in a Binary Search Tree (BST). The desired LCA is the deepest node in the BST that is the ancestor of both *key*1 and *key*2 nodes. Your code is expected to leverage the BST property of each node.

3. You have seen in class that the minimum number of nodes in an AVL tree of height h, denoted $S(h)$, follows the recursion: $S(h) = S(h-1) + S(h-2) + 1$. The challenge now is to actually construct such an AVL tree of height h that contains the fewest number of nodes possible. Since there are multiple such AVL trees possible, to break ties, at each node, you can construct a left-heavy AVL tree (i.e., height of the node's left subtree is $h-1$ and the right subtree is $h-2$).

   Write a function that constructs this AVL tree. You can either construct this AVL tree recursively, or reuse your BST insert code and just use repeated calls to BST::insert using an appropriate insertion order. You can label the keys as $1, 2, \ldots, n$, where $n = S(h)$ denotes the minimal number of nodes.

4. Given a height h, write a function that will output an insertion order for a binary search tree (BST) that will lead to a perfectly balanced BST containing keys from 1 to $2^{(h+1)} - 1$.

5. Given a set of n integers, write a function that outputs an insertion order that will lead to the most balanced Binary Search Tree (BST). Construct this BST using the specified insertion order and print its postorder traversal.