CSE 674 Advanced Data Structures

More on Balanced Binary Search Trees

Andrew C. Lee

EECS Dept, Syracuse U.

Contents

We will discuss

1. The implementation of the main operations of a Binary Search Trees
2. AVL Trees
3. Splay Trees

Binary Search Trees and its variants

Binary Search Trees

1. Worst case performance of all the basic operations are $O(h)$, where h is the height of the tree
2. keep the key in order when we implement the basic operations
3. try to make additional efforts to maintain *balance*
   Two examples: AVL trees and Splay trees

Binary Search Trees

the three major operations are:

1. Insertion
2. Deletion
3. Search

We will review the implementation of these operations

The contains (Search) operation

Discussions

```
/**
 * Internal method to test if an item is in a subtree.
 * x is item to search for.
 * t is the node that roots the subtree.
 */
bool contains( const Comparable & x, BinaryNode *t ) const
{
    if( t == nullptr )
        return false;
    else if( x < t->element )
        return contains( x, t->left );
    else if( t->element < x )
        return contains( x, t->right );
    else
        return true;    // Match
}
```

Figure 4.18 contains operation for binary search trees

The insert operation

Let's try the following:
`insert (x,p)` // what is `x` ? what is `p`? what should the function return ?

1. What if `p` is null // empty tree ... work needs to be done
2. What if `p` is not null but `x < p-> data`
3. What if `p` is not null but `x > p-> data`
4. You reach this step when you detect a duplicate. decide what will you do with them (keep a count ?)

The delete operation I

It may be better if we have a findmin function

`findmin(p)` // p cannot be null here

1. `while (p->left !=null) p=p->left;`
2. `return p`

The delete operation II

Assuming that we have `findmin`, `delete` may look like

`delete (x,p)` // what should it return ?

1. `if (p is null)` // fell out of the tree, x does not exist in the tree
2. `else` // the main work is here
3. return p;

The delete operation III

The main work for the `delete` operation may look like the following:

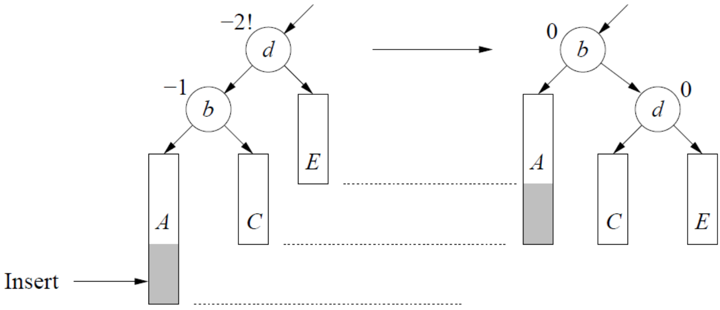
```
if (x < p.data)                              // x in left subtree
    p.left = delete(x, p.left);
else if (x > p.data)                         // x in right subtree
    p.right = delete(x, p.right);
                                             // x here, either child empty?
else if (p.left == null || p.right == null) {
    BinaryNode repl;                         // get replacement
    if (p.left == null) repl = p.right;
    if (p.right == null) repl = p.left;
    return repl;
}
else {                                       // both children present
    p.data = findMin(p.right).data;          // copy replacement
    p.right = delete(p.data, p.right);       // now delete the replacement
}
```

AVL Trees

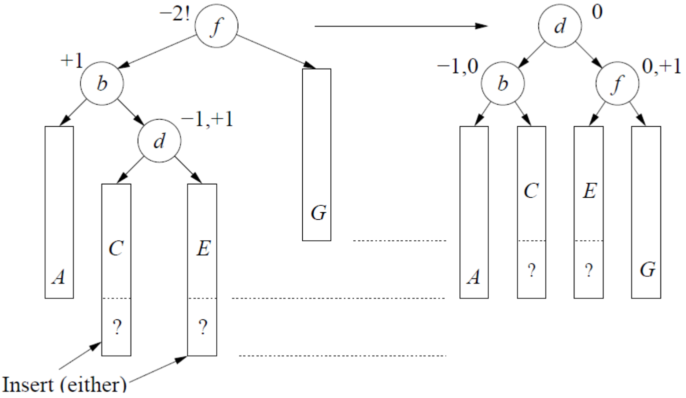
To understand the working of AVL trees, please note the following:

1. An AVL tree needs to maintain the following *invariant*:
   AVL Balance Condition: For every node in the tree, the height of the left subtree and the right subtree differ by at most 1.
2. The node structure needs to keep track of the heights of the subtrees (e.g. keep a field `balance` which must be either -1, 0 or 1 to satisfy the AVL condition)
3. After we perform an `insert` or `delete` operation, we need to check if we need to `re-balance` the resulting tree
4. When attempting to balance the tree, suitable types of rotations needs to be used (How to decide that ?)

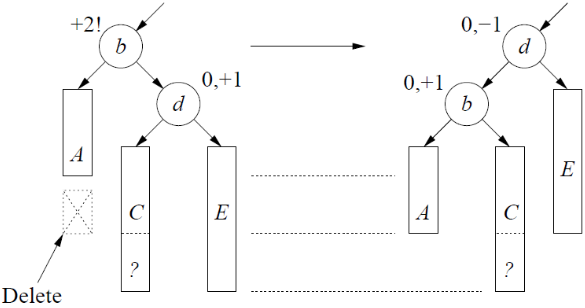
AVL trees: insertion I



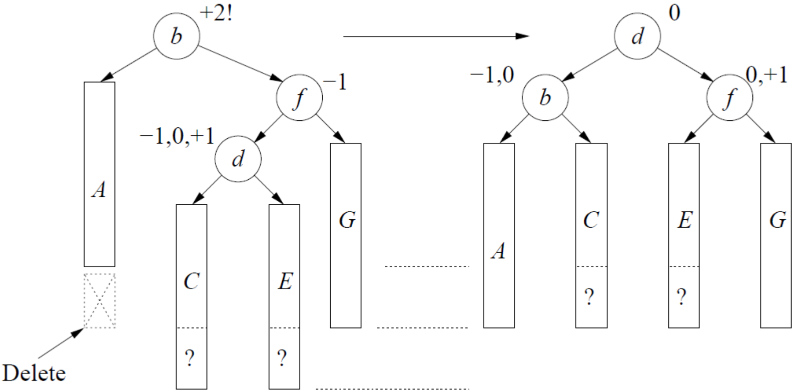
AVL trees: insertion II



AVL trees: deletion I



AVL trees: deletion II



Splay Trees

1. Simpler than AVL trees
2. Guarantee
   after applying M consecutive tree operations from the empty tree, the worst case is $O(M \lg n)$ time
   What is the amortized time per operation ?
3. Do not need to maintain height information at each node
4. Use `Splay` operation

The Splay Operations

The main points are:

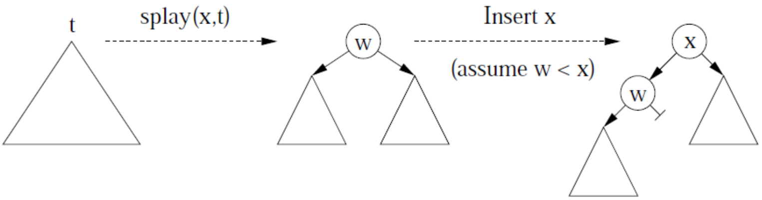
1. Similar to Rotations
2. zig and zag operations
3. zig-zag and zig-zig operations
4. goes from bottom up

Note: We will use diagrams to illistrate these operations

Splay Trees: The dictionary operations

1. Search for an element x from the tree t is simply splay (x, t) and check the root after the operation
2. Insert an element x to the tree t
   Note: splay(x, t) will move the inorder successor or predessor to the root and we insert x from the resulting tree.
3. Delete and element x to the tree t
   Question: How to make use of splay in this case?

Splay Trees (Insert)



Splay Trees (Delete)