Computer Science 2530
April 9, 2020

Happy Thursday, April 9.

Today we look at important data structure, binary search trees. I will use abbreviation BST for binary search tree.

Binary search trees

A BST can be used to store a set of values where you can

1. check whether a particular value is in the set;
2. add a value to the set;
3. remove a value from the set.

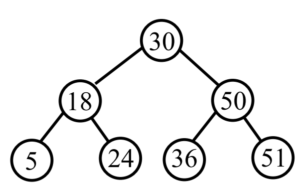
For simplicity, our BSTs will always store sets of integers. It is easy to modify them to store sets of any type of values.

A BST is a binary tree with an ordering requirement. If a node v in a BST contains item k, then every node in the left subtree of v contains an item that is $< k$ and every node in the right subtree of v contains an item that is $> k$. No item can occur more than once in a binary search tree.

Page **40A** in the notes describes BSTs and shows examples.

Lookup in a binary search tree

A BST represents the set of all of its items. For example, BST



Tree t_0

represents set $\{5, 18, 24, 30, 36, 50, 51\}$. Lets call the above BST t_0.

The ordering requirement makes it easy to test whether a value is in a BST. For example, imagine checking whether 24 is in t_0. Comparing 24 to 30 shows that 24 must be in the left subtree, if it is in t_0 at all. So you move to the left subtree. At each node v, you simply compare the value x that you are searching for with the item k that is in node v.

1. If $k = x$, you can stop; obviously, x occurs in the tree rooted at v. For example, if you want to know whether 30 occurs in t_0, you stop immediately because 30 occurs in the root of t_0.

2. If $x < k$, then x can only occur in the left subtree of v, if it occurs at all. So you search the left subtree.

3. If $x > k$, then x can only occur in the right subtree of v, if it occurs at all. So you search the right subtree.

There is one more important case. Recall that a NULL pointer is an empty tree. It represents an empty set. When asked whether x occurs in an empty tree, you always answer no. That will cause a search for 42 in t_0 to return false.

Page **40B** in the notes describes function member(x, t), which returns true if (and only if) x occurs in BST t. Read about that.

Insertion into a binary search tree

Insertion into a BST is really simple. Here are the rules for inserting x into a BST t.

1. If you are asked to insert x into an empty tree, replace the empty tree by a node that contains x.

2. If you are asked to insert x into a tree whose root contains x, do nothing, since x is already there, and a BST is not allowed to contain any value more than once.

3. If you are asked to insert x into a tree t whose root contains item k where $x < k$, insert x into the left subtree of t.

4. If you are asked to insert x into a tree t whose root contains item k where $x > k$, insert x into the right subtree of t.

Important fact

After inserting x into a tree that did not already contain x, you will always find x in a leaf.

Page **40B** of the notes gives a definition of insert(x, t), which inserts x into BST t. (Insert is a destructive function; it changes t.) Here is the definition of insert.

```
//===========================================
//                insert
//===========================================
// insert(x,T) inserts x (destructively) into
// binary search tree T. If x is already a
// member of T, insert does nothing.
//===========================================

void insert(int x, Node*& T)
{
  if(T == NULL)
  {
    T = new Node(x, NULL, NULL);
  }
  else if(x < T->item)
  {
    insert(x, T->left);
  }
  else if(x > T->item)
  {
    insert(x, T->right);
  }
}
```

Here are some observations about the definition of insert.

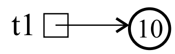
1. Notice that T is a pointer that is passed by reference. Any change to variable T will change the variable that is passed to insert. For example, after

   ```
   Node* t1 = NULL;
   ```

   variable $t1$ holds a null pointer. Then, after doing

```
insert(10, t1);
```

$t1$ looks looks like this.



Clearly, insert has changed the pointer stored in $t1$.

2. There is no else case at the end of the cases. If $x = t$->item, then insert(x, t) does nothing.

Exercises

Do the exercises at the bottom of page **40B**. You should find them very easy.

Removing the smallest value

The smallest value in a BST is always found by starting at the root and moving as far as possible to the left. That is, you follow left pointers until you encounter a NULL pointer.

Page **40C** describes function removeSmallest(t), which

1. removes the smallest value from nonempty BST t, and
2. returns the value that was removed.

Removing the smallest value is an important tool to help remove a given value. Read **40C** to see how removeSmallest works.