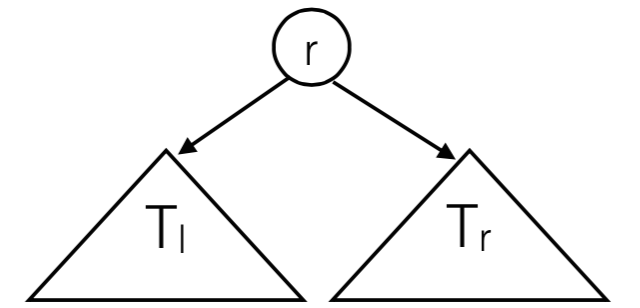
**Binary Search Tress**

A binary search tree consist of a root node, and at most two non-empty sub-tress connected by a direct edge to the root; no value will appear more than one in the binary search tree. We want to be able to find items in O (log N).

* For every node n with value x
  + Value of nodes in left sub-tree of n are smaller than x
  + Value of nodes in right sub-tree of n are larger than x



**BST Operations**

**Contains**

To find a value V into tree T,

* If the tree T is empty, the lookup failed. If V is at the root, you found it. If V < T.root, look for V in T.left, otherwise look in T.right.

private boolean contains(Integer x, BinaryNode t) {

if (t == null)

return false;

if (x < t.data)

return contains(x, t.left);

else if(t.data < x)

return contains(x, t.right);

else

return true;

}

**Insertion**

To insert a value V into tree T,

* If the tree T is empty, give it a new root node with V, otherwise if V < T.root, insert V into T.left, otherwise insert V into T.right.

private BinaryNode insert( Integer x, BinaryNode t ) {

if (t == null)

return new BinaryNode(x, null, null);

if (x < t.data)

t.left = insert(x, t.left);

else if (t.data < x)

t.right = insert(x, t.right);

return t;   
}

**findMin**

private BinaryNode findMin(BinaryNode t) {

if (t == null)

return null;

else if (t.left == null)

return t;

return findMin(t.left);

}

* findMax is similar, just traverse the right tree instead

**Remove**

Use contains(x) to find x; if x is a node that is a leaf, simply just remove it. If the node containing x has a single child, attach that child to the parent of the node containing x.

If the node containing x has two children, childleft and childright:

* Find the smallest node, n, in the sub-tree of childright
* Replace x with value of n.
* Recursively remove n

private BinaryNode remove(Integer x, BinaryNode t){

if(t == null)

return t; // Item not found; do nothing

if (x < t.data)

t.left = remove(x, t.left);

else if(t.data < x)

t.right = remove(x, t.right);

else //found x

if (t.left != null && t.right != null) { // 2 children

t.element = findMin(t.right).element;

t.right = remove(t.element, t.right);

} else

if (t.left != null) // 1 or 0 children.

return t.left;

else

return t.right;

}

**BST Heights**

In the worst case, the tree would not branch, and would thus have a height of N; in the best case, the height would be log N.

**AVL Trees**

An AVL tree is a binary search tree in which the following condition holds after each operation:

* For each node, the heights of the left and right sub-trees differ by at most 1.

To maintain this balance, after each insertion, we find the lowest node that violates the balance condition (if any such node does); then we perform a rotation to re-balance the tree.

**Single Rotation**

**Left Rotation**

a

\

b

\

c

We must perform a rotation here, rooted at a. To achieve this, the following should be done:

* ‘b’ will be the new root
* The left child of ‘b’ becomes ‘a’s right child (in this case, it is null)
* ‘a’ becomes the left child of ‘b’

b

/ \

a c

**Right Rotation**

c

/

b

/

a

We must perform a rotation here, rooted at a. To achieve this, the following should be done:

* ‘b’ will be the new root
* The right child of ‘b’ becomes ‘c’s left child (in this case, it is null)
* ‘c’ becomes the right child of ‘b’

b

/ \

a c

**Double Rotation**

*\*\*REFER TO LINAN’S GODLY 1337 helicopter shortcut/trick\*\*\**

*^quite possibly the most amazing thing I learned last term*