Binary Search Tree

[def]

It must satisfy the binary-search-tree property:

1. Let x, y and p be a node in binary search tree. If x is the left child of p and y is the right child of p, then .

[visit]

1. Inorder tree walk
2. Preorder tree walk
3. Postorder tree walk

Let L is the left child of P and R is the right child of P.

The visit are as follows for any non-leaf node.

1. Inorder tree walk: LPR
2. Preorder tree walk: PLR
3. Postorder tree walk: LRP

[pseudo code]

(1)inorder:

inorder(x)

{

if (x!= NIL) { inorder(x.left); print(x); inorder(x.right);}

}

(2)preorder:

preorder(x)

{

if(x!=NIL){print(x); preorder(x.left); preorder(x.right);}

}

(3)postorder:

postorder(x)

{

if(x!=NIL){postorder(x.left); postorder(x.right);print(x);}

}

[analysis]

If x is the root of an n-node subtree, then the call inorder(x) takes time.

[pf]

Let’s take two part.

Suppose T(n) donate the time taken by inorder(x).

1. Ω(n):

When we call inorder(x) for x is root node, we must spend the most time. In this case, it has to visit all node.

Hence, this part is proven.

1. O(n):

For x is leaf node, call inorder(x) must spend less time. In this case, it only spends a small time to check if x is not NIL. We can write

For x is not leaf node, call inorder(x) must visit left child and right child. In this case, it spends

Let’s substitute

We have

Hence this part is proven.

Proved. YA!!!

[operation]

(1)