# Binary search Tree

### Binary Search Tree

Definition:

1. An empty tree(a tree that has no nodes) is a Binary Search Tree.

2. The left(right) sub-tree of a Binary Search Tree is a Binary Search Tree. The root of a Binary Search Tree is not less(greater) than the root of its left(right) sub-tree if its left sub-tree is not empty.

### Left(Right) Most Child

Denotations: the Left(Right) Most Child of a node ***N*** is denoted as lmc(***N***)(rmc(***N***)).

Definition:

1. If node ***N*** doesn't have a left(right) child, then lmc(***N***)(rmc(***N***)) = ***N***.

2. Else the lmc(***N***)(rmc(***N***)) = lmc(***N.left***)(rmc(***N.right***))

### Left(Right) Most Parent

Denotations: the Left(Right) Most Parent of a node ***N*** is denoted as lmp(***N***)(rmp(***N***)).

Definition:

1. If node ***N*** is not the right(left) child of its parent, then lmp(***N***)(rmp(***N***)) = ***N***.

2. Else the lmp(***N***)(rmp(***N***)) = lmp(***N.parent***)(rmp(***N.parent***)).

### Successor of a node

Denotations: the Successor of a node ***N*** is denoted as suc(***N)***.

Definition:

1. If ***N*** has a right child, then the suc(***N***)=lmc(***N.right***).

2. Else if lmp(***N***) is the left child of a node ***M***, suc(***N***)=***M***.

3. Else ***N*** don't have a successor.

### Predecessor of a node

Denotations: the Successor of a node ***N*** is denoted as pre(***N)***.

Definition:

1. If ***N*** has a left child, then the pre(***N***)=rmc(***N.left***).

2. Else if rmp(***N***) is the right child of a node ***M***, pre(***N***)=***M***.

3. Else ***N*** don't have a predecessor.

### Minimum(Maximum) Node

Denotations: the Minimum(Maximum) of a node ***N*** is denoted as min(***N)***(max(***N***)).

Definition:

The Minimum(Maximum node) of a node ***N*** *in a* Binary Search Tree is a descendent(including ***N***) of that is not greater(less) than any other descendents of ***N***.

### Proposition 1

A node in a Binary Search Tree is not less(greater) than any node in its left(right) sub-tree.

### Proposition 2

The lmc(***N***)(rmc(***N***))is a minimum(maximum) node of a node ***N***.

### **Proposition 3**

Assume that ***M*** and ***N*** are two nodes in a Binary Search Tree. If ***M*** is greater than min(***N***)*and* is less than max(***N***), then ***M*** is a descendent of ***N***.

### Proposition 4

If suc(***N***)(pre(***N***)) exists, then it is the smallest(greatest) node that is not less(greater) than ***N***.

***proof:***

First we prove that suc(***N***) is not less than ***N***.

If ***N*** has a right child, then suc(***N***) = lmc(***N.right***), that is suc(***N***) is in the right sub-tree of ***N***. By **Proposition 1**, suc(***N***) is not less than ***N***.

If ***N*** doesn't have a right child then lmp(***N***) is the left child of suc(***N***), which means that ***N*** resides in the left sub-tree of suc(***N***). By **Proposition 1**, suc(***N***) is not less than ***N***.

In either of the two cases suc(***N***) is not less than ***N***.

Now we prove that suc() is the smallest(greatest) node that is not less(greater) than ***N***. Assume that node ***M*** is greater than ***N*** and is less than suc(***N***).

If ***N*** has a right child, suc(***N***) is in the right sub-tree of ***N***. We could conclude from our assumption, **Proposition 3** and **Proposition 1** that ***M*** resides in the right sub-tree of ***N***. But ***M*** is less than suc(***N***) and suc(***N***)=lmc(***N.right***), by **Proposition 2** this leads to a contradiction.

If ***N*** doesn't have a right child then lmp(***N***) is the left child of suc(***N***). We could conclude from our assumption, **Proposition 3** and **Proposition 1** that ***M*** resides in the left sub-tree of suc(***N***).That is ***M*** is a descendent of lmp(***N***). But ***M*** is greater than ***N*** and by **Proposition 5** *and* **Proposition 2**, the largest node in of lmp(***N***) is ***N***, this leads to a contradiction.

Both of the two cases would lead to contradictions. That is there exists no such node that is greater than ***N*** and is less than suc(***N***).

## **Proposition 5**

If a node does not have a right(left) child, then rmc(lmp(***N***))(lmc(rmp(***N***))) = ***N***.

### ***Proposition 6***

Assume that every node in a tree is distinct, then the smallest(greatest) node that is not less(greater) than ***N***, if such a node exists, is suc(***N***)(pre(***N***)).

***proof:***

Since every node in a tree is distinct, there must have at most one node such that it is the smallest(greatest) node that is not less(greater) than ***N***. And by **Proposition 4**,this node must be the suc(***N***) if it exists.

### Proposition 7

If node ***N*** has a right(letft) child, then suc(***N***)(pre(***N***)) does not have a left(right) child.

***proof:***

If node N has a right(left)child, then suc(***N***)(pre(***N***))= lmc(***N.right***)(rmc(***N.left***)), by the definition of “lmc”(“rmc”), suc(***N***)(pre(***N***)) does not have a left(right)child.

### Proposition 8

If ***N*** is a leaf node and ***P*** is its parent, then ***P*** is either the smallest node that is not less than ***N***, or ***P*** is the largest node not greater than ***N***.

**proof:**

If ***N*** is the left child of ***P***. Since ***N*** is a leaf, it doesn't have right child, and lmp(***N***) = ***N***.

suc(***N***)=***P***. By **Proposition 4**, ***P*** is the smallest node that is not less than ***N***.

If ***N*** is the right child of ***P***. Since ***N*** is a leaf, it doesn't have left child, and rmp(***N***) = ***N***.

suc(***N***)=***P***. By **Proposition 4**, ***P*** is the largest node not greater than ***N***.