Insertion and Deletion on Binary Search Tree

*Principle:*

The insertion and deletion operation on the Binary Search Tree would cause the structure changes on the Binary Search Tree. We need to modify the structure of Binary Search Tree in order to keep the structure of Binary Search Tree.

*Insertion:*

In the procedure to insert one node into Binary Search Tree, we need to make changes on the tree T and node z.

*struct Node {*

*int key;*

*Node \* left; // The left child node pointer;*

*Node \* right; // The right child node pointer;*

*Node \* p; // The parent node pointer;*

*};*

*Pseudo – Code:*

Tree\_Insertion (Tree T, Node z) {

Node y = T.NIL;

Node x = T.root;

// Locate the place where node z would be inserted.

while (x != T.NIL) {

y = x;

if (z.key > x.key) {

x = x.right;

} else if (z.key < x.key) {

x = x.left;

}

}

// Insert the node z right | left after node y.

z.p = y;

if (y == T.NIL) {

T.root = z;

} else if (z.key < y.key) {

y.left = z;

} else {

y.right = z;

}

}

*Deletion:*

During the deletion procedure, there have three situations of deletion:

1. *First situation:*

The deleted node has no child node, neither left nor right node. Under such situation, we can easily define to delete this node from Tree. There has no need to setup any child node to substitute the current node.

1. *Second situation:*

The deleted node has one child node. If the deleted node has left node, then just substitute the left node from the deleted node;

Or the deleted node has right node, then just substitute the right node from the deleted node.

1. *Third situation:*

The deleted node has two child nodes, both left and right nodes. Under this kind of scenario, what we need to do is to just find the most left node from the sub-tree. And substitute the right node with the most left node. At last, we substitute the most left node with the deleted node.

*Supplement:*

In the pseudo – code, we design the code procedure -「TRASNPLANT」, by using this procedure, we easily define the Node u and the substitution node v to exchange the node v with the node u.

In the code procedure -「Tree\_Deletion」is used to delete the node z from Tree T. We use the three situations described above to finish the following two procedures.

TRANSPLANT (Tree T, Node u, Node v) {

if (u.p == T.NIL) {

T.root = v;

} else if (u == u.p.left) {

u.p.left = v;

} else if (u == u.p.right) {

u.p.right = v;

}

if (v != T.NIL) {

v.p = u.p;

}

}

Tree\_Deletion (Tree T, Node z) {

if (z.left == T.NIL) {

TRANSPLANT (T, z, z.right);

} else if (z.right == T.NIL) {

TRANSPLANT (T, z, z.left);

} else {

// The procedure TREE\_MINIMUM is used to locate the node

// with the minimum key value.

y = TREE\_MINIMUM(z.right);

if (y.right != T.NIL) {

TRANSPLANT (T, y, y.right);

y.right = z.right;

y.p.right = y;

}

TRANSPLANT (T, z, y);

y.left = z.left;

y.left.p = y;

}

}