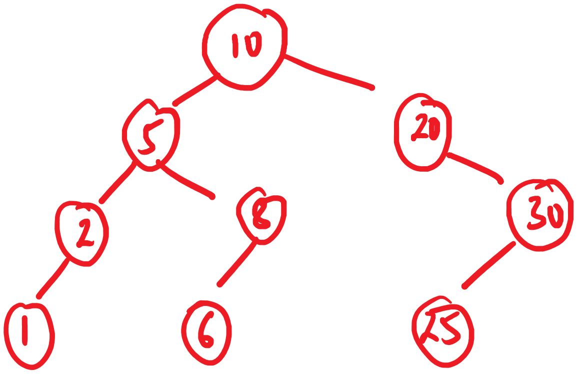
Lecture 7-8

Wednesday, October 13, 2021 8:00 AM

Binary Search Tree :-

It is a binary tree such that for all nodes, 'i', then

$$LST(i) < data(i) < RST(i)$$



At max we require height of BST to search a given element

Searching:

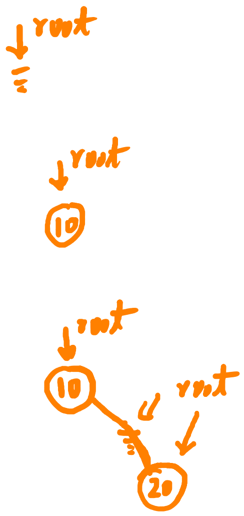
$\Theta(\log_2 n)$

$\Omega(1)$ ← root node is to be searched

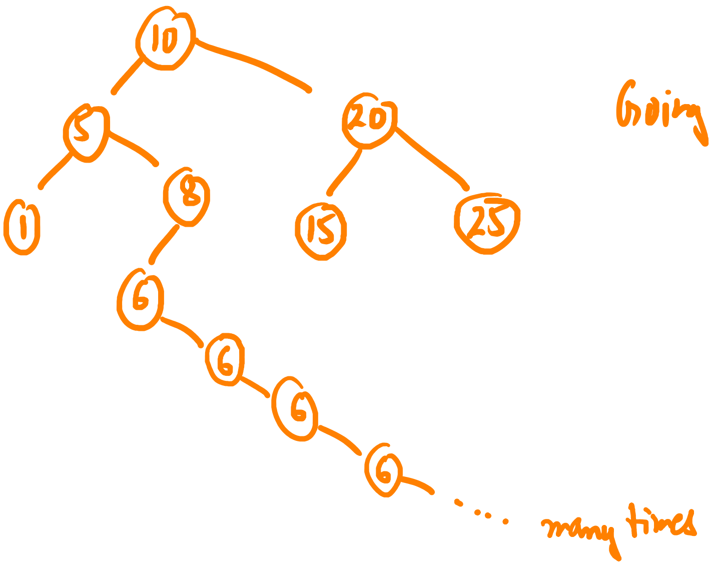
$O(n)$ ← skew binary tree

BST Insertion:-

```
BTNODE * insertion(BTNODE *root, int x)
{
    if (root == NULL)
    {
        root = (BTNODE *) malloc (sizeof (BTNODE));
        root -> data = x;
        root -> lchild = NULL;
        root -> rchild = NULL;
    }
    elseif (x < root -> data)
        root -> lchild = insertion (root -> lchild, x);
    else
        root -> rchild = insertion (root -> rchild, x);
    return root;
}
```



Duplicate data : Either in LST or RST



Going skew

Increase space

lchild	data	freq	rchild

BST Deletion:-

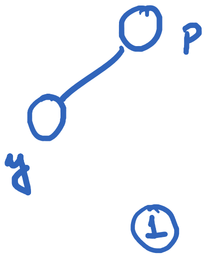
3 kinds of nodes

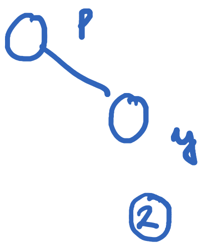
0 degree node

1 degree node

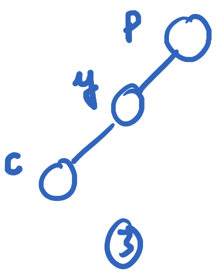
2 degree node

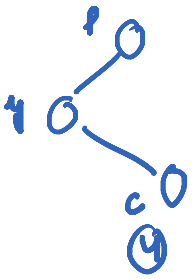
(i) 0 degree node (Delete y)

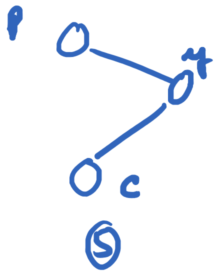


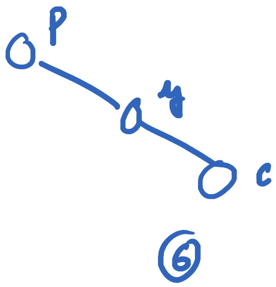


(ii) 1 degree node (Delete y)







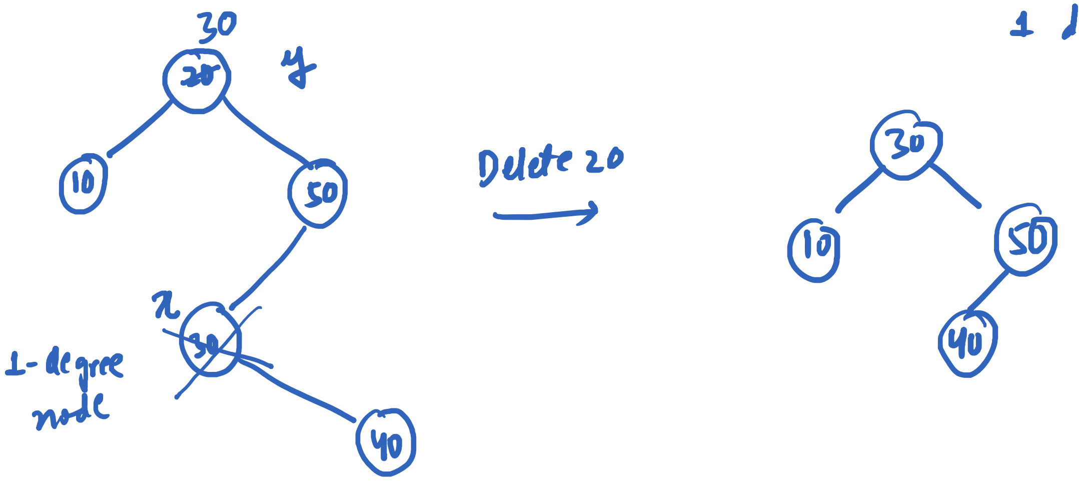


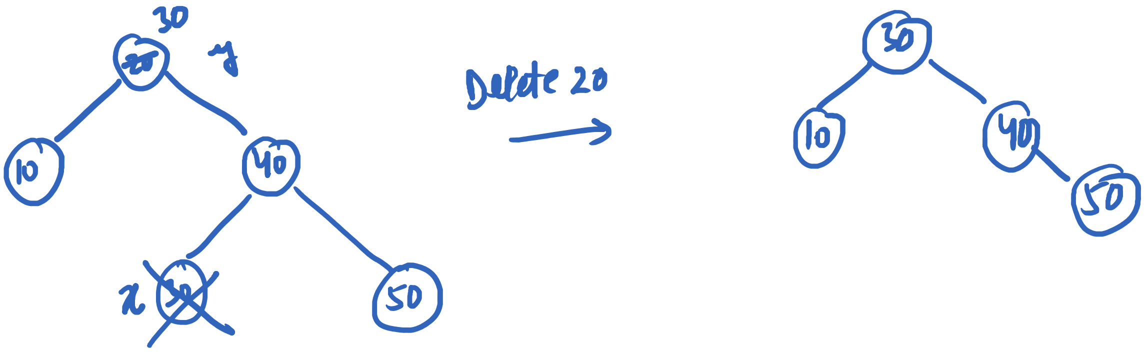
(iii) 2 degree node (delete y)

Replace y with its inorder successor node's data 'z'

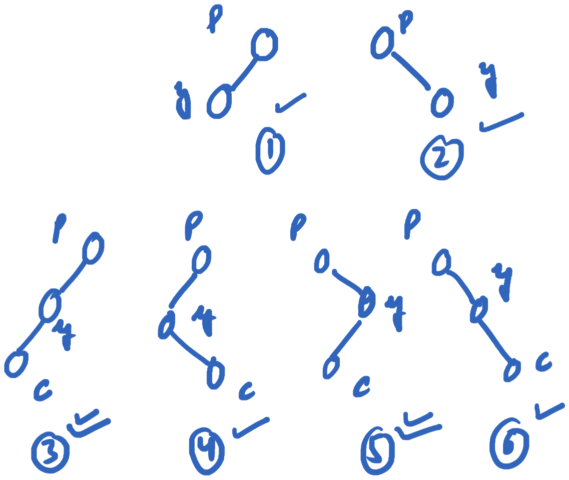
Now delete the inorder successor node

will always be either a 0 degree node or a 1 degree node

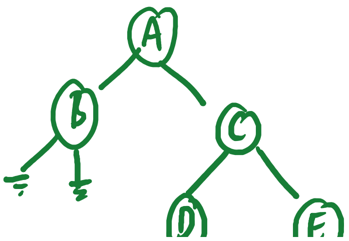




```
BTNODE *deletion (BTNODE *root, int y)
{
    BTNODE *temp;
    if (root == NULL)
    {
        Printf("Either the BST is empty or y doesnot exists in BST");
          return NULL;
    }
    elseif (y < root->data)
          root->lchild = deletion(root->lchild, y);
    elseif (y > root->data)
          root->rchild = deletion(root->rchild, y);
    else
else
{
    if (root -> lchild == NULL)
    {
        temp = root -> rchild;
        free(root);
        return temp;
    }
    elseif (root -> rchild == NULL)
    {
        temp = root -> lchild;
        free(root);
        return temp;
    }
    else
    {
        temp = root -> rchild;
        while (temp -> lchild != NULL)
            temp = temp -> lchild;
        root -> data = temp -> data;
        root -> rchild = deletion(root -> rchild, temp -> data);
    }
}
return root;
}
```



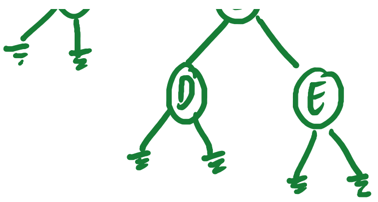
Threaded Binary Tree:-



Few drawbacks of Binary Tree

(i) More than 50% pointers are NULL

(ii) Traversal requires a stack of size equivalent to height of the binary tree



to height of the binary tree

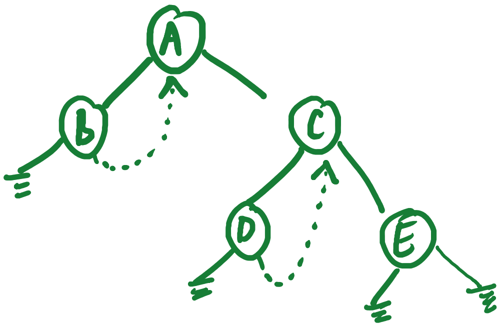
TBT

Left TBT (Inorder predecessor)

Right TBT (Inorder successor)

Left-Right TBT

Right In TBT :-

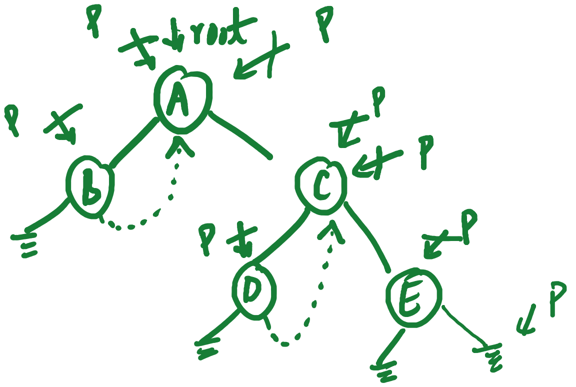


—— edge

-.-.-.- thread

lchild	data	rthread	rchild

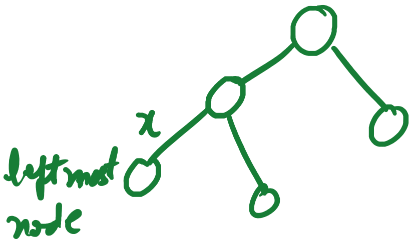
$$\text{rthread} = \begin{cases} 0 & \text{edge} \\ 1 & \text{thread} \end{cases}$$

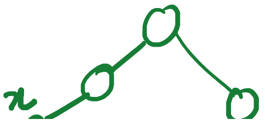


BTNODE * p = root ;

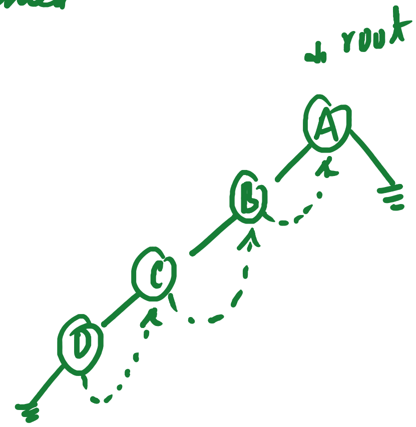
B A D C E

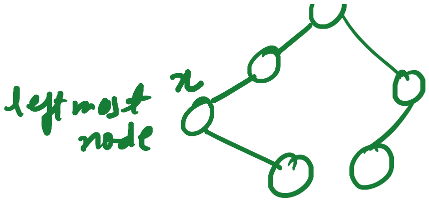
```
P = root;
while (P != NULL)
{
P = move to the leftmost node
  Display P
  if (P→ rthread == 1)
       Display P→ rchild
  P = P→ rchild
```





```
    p = p → rchild
}
```





x
leftmost node