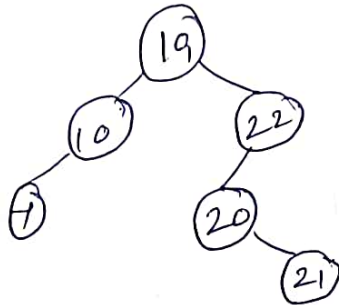


UNIT - II

→ Binary search Tree:

→ Left ^{sub}tree - smaller value than root.

→ Right subtree - larger value than root.



* Difference b/w BT & BST

Basic operations:

insertion, deletion, find, findmin, findmax, makeempty.

struct TreeNode

```
{
  int element;
  searchtree left;
  searchtree right;
};
```

Makeempty:

→ mainly for initialization

→ initialize the first element as one node tree

```
searchtree Makeempty (searchtree T)
```

```
{
  if (T != NULL)
```

```
  {
    Makeempty (T → left);
```

```
    Makeempty (T → right);
```

```
    free (T);
```

```
  }
  return NULL;
```

```
}
```

Insert:


→ Insert the 'x' element into the tree.

→ check with the root node tree.

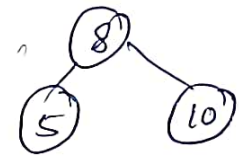
→ if $x < \text{root}$ → insert into ~~left~~ leftside $T \rightarrow \text{left} = \text{NULL}$
 $x > \text{root}$ - " " " " rightside $T \rightarrow \text{right} = \text{NULL}$

eg. 8, 5, 10, 15, 20, 18 Construct BIT

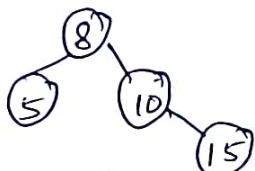
① Insert '8' ② Insert 5
* Compare with 8,
* $5 < 8$ insert left



③ Insert '10'
 * Compare with root '8'
 * $10 > 8$ insert into Right



④ Insert '15'
 * Compare with root '8', $15 > 8$, so traverse right subtree, again compare '10' $15 > 10$, insert 15 in right subtree



⑤ Insert '20' ≤ 18

Routine:

Searchtree Insert (int x, searchtree T)

{ if (T == NULL) *

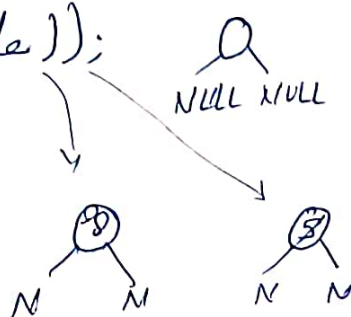
{ T = malloc (sizeof (struct Treenode));

if (T != NULL)

{ T → element = x;

T → left = NULL;

T → Right = NULL;



}
 }

else if ($x < T \rightarrow \text{element}$)

$T \rightarrow \text{left} = \text{insert}(x, T \rightarrow \text{left});$

elseif ($x > T \rightarrow \text{element}$)

$T \rightarrow \text{right} = \text{insert}(x, T \rightarrow \text{right});$

return T;

}

eg:

3, 1, 4, 6, 9, 2, 5, 7

Find operation:

→ check root is NULL, if so return NULL.
→ otherwise check the value 'x' with root node.

if $T \rightarrow \text{data} = x$ return T.

if $x < T \rightarrow \text{data}$ traverse the left of T recursively

if $x > T \rightarrow \text{data}$ " " " " " "

Routine:

int find (int x, searchtree T)

{

if (T == NULL)

return NULL;

if ($x < T \rightarrow \text{element}$)

return find (x, T → left);

elseif ($x > T \rightarrow \text{element}$)

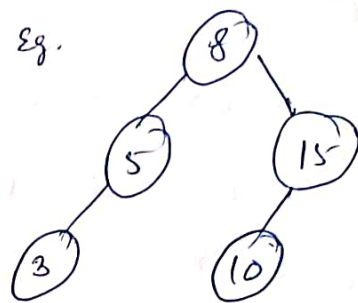
return find (x, T → right);

else

return T;

}

find the value 10 in the following eg.



① find (X, T) ^{10 8}, initialize pointing
Root '8'

if (T == NULL) X

if (X < T) X
10 < 8

else if (10 > 8) ✓

find (10, T → right):

find (10, 8 → right)

return find (10, 15)

② find (X, T) (Recursive call)

if (T == NULL) X

if (10 < 15) ✓

return find (10, T → left)

find (10, 10)

③ find (10, 10) recursive call.
X T

if (T == NULL) X

if (10 < 10) X

if (10 > 10) X

else

return T

10 ✓

findMin:

→ return position of smallest element in the tree.

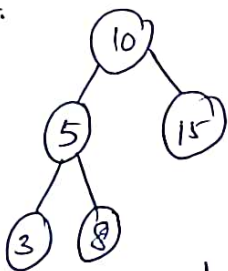
→ start from root, go left,

Recursive alg for findMin:

```
int findMin (searchTree T)
{
    if (T == NULL)
        return NULL;
    else if (T->left == NULL)
        return T;
    else
        return findMin (T->left);
}
```

g

Eg:



findMin (T),
findMin (10)

① findMin (10)
T == NULL X false

T->left == NULL
5 == NULL false

return findMin (T->left),
findMin (5)

② findMin (5)

T == NULL X

T->left == NULL

3 == NULL X

return findMin (3).

③ findMin (3)

T == NULL X

T->left != NULL ✓

return 3.

Non-Recursive alg for findMin.

int findmin (searchtree T)

↳ if (T != NULL)

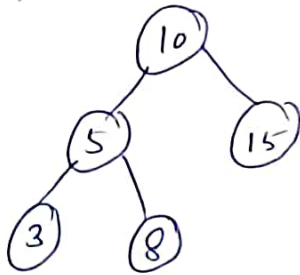
while (T->left != NULL)

T = T->left;

return T;

}

eg:



① (10 != NULL) ✓

while (T->left != NULL)

T = T->left

(T) = 5

T = 5

② 5 != NULL

3 != NULL

T = T->left

T = 3

③ 3 != NULL

while (T->left != NULL)

3->left != NULL ✗

return T

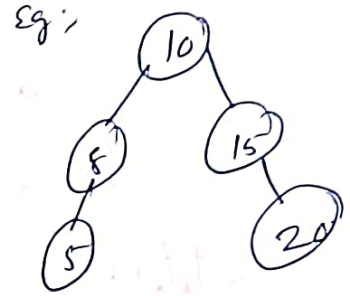
return 3

findmax:

- return position of largest element in the tree.
- start from root, traversal right.

Recursive alg for findmax

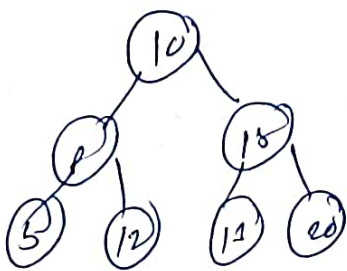
```
int findmax (searchtree T)
{
    if (T == NULL)
        return NULL;
    else if (T->right == NULL)
        return T;
    else
        return findmax (T->right);
}
```



Non Recursive alg:

```
int findmax (searchtree T)
{
    if (T != NULL)
        while (T->right != NULL)
            T = T->right;
        return T;
}
```

Eg:



Deletion in BST:

→ It is a complex operation in BST

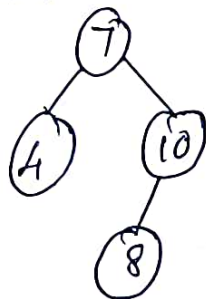
→ For deleting an element consider 3 possibilities

* Node to be deleted is a leaf node, i.e) No children

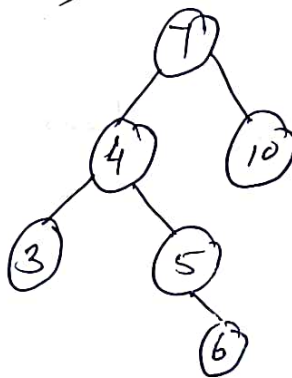
* Node with one child.

* Node with two child.

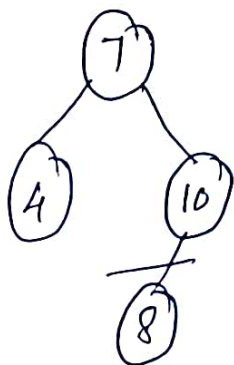
delete(8)



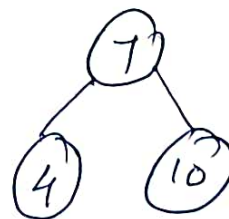
delete(5)



Case 1: Node with no children

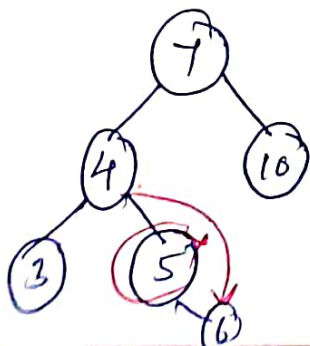


After deletion

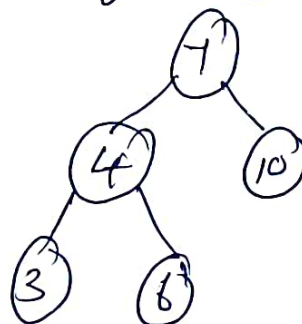


Case 2: Node with one child

→ It can be deleted by adjusting pointer.

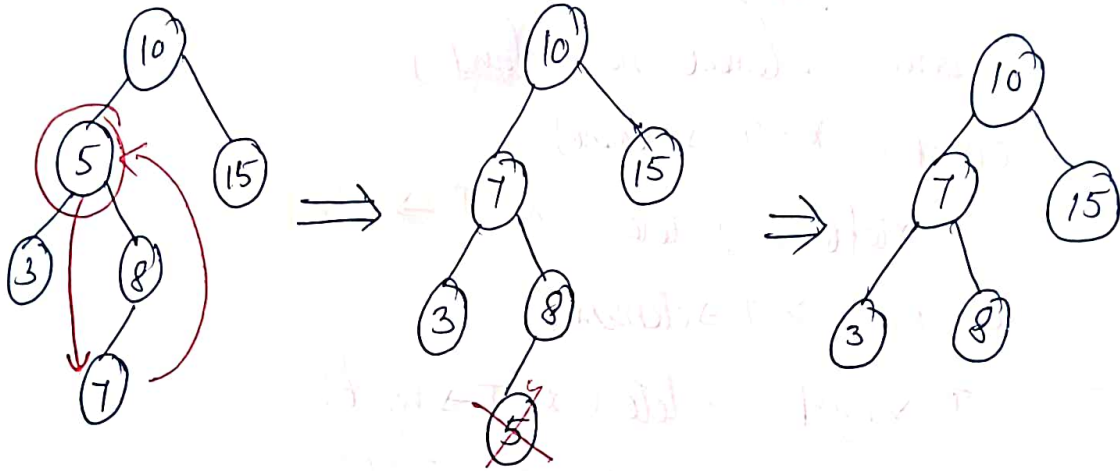


After deletion

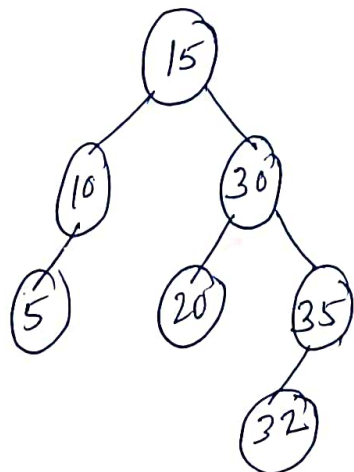
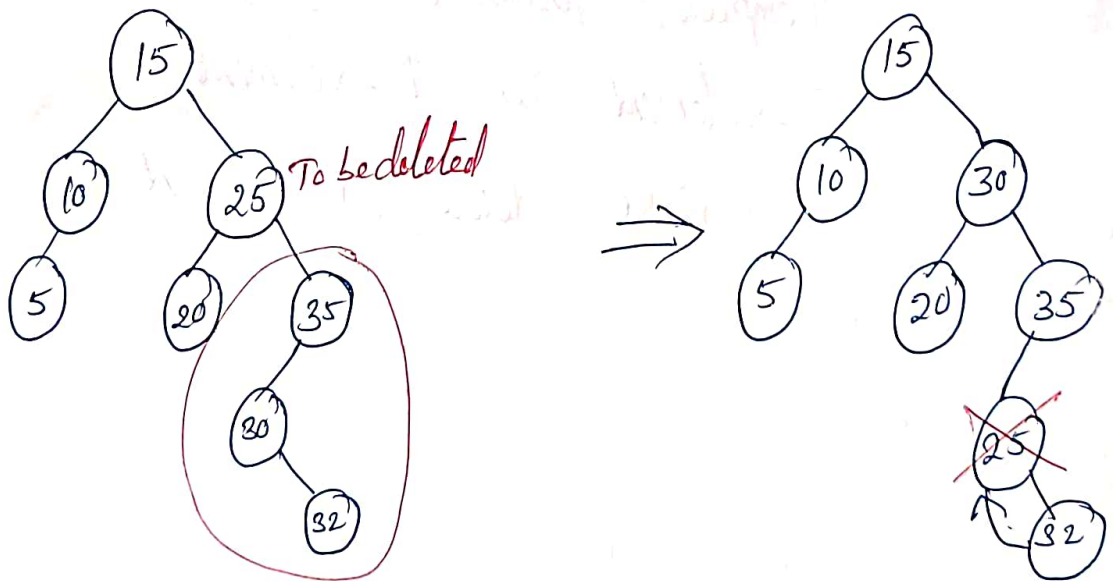


3: Node with two children (Delete 5)

* 1st replace the data of the node to be deleted with its smallest node of the right subtree, and recursively delete the node.



Delete 25:



Swap minimum value in Right subtree

Routine to delete:

Searchtree delete (int x, searchtree T)

{

int tempcell;

if (T == NULL)

error("Element not found");

elseif (x < T->element)

T->left = delete(x, T->left);

elseif (x > T->element)

T->right = delete(x, T->right);

elseif (T->left && T->right)

{

Tempcell = findmin(T->right)

T->element = Tempcell->element;

T->right = delete(Tempcell->element, T->right);

}

else

{

Tempcell = T;

if (T->left == NULL)

T = T->right;

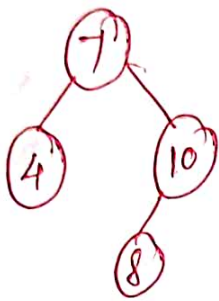
elseif (T->right == NULL)

T = T->left;

free(Tempcell);

}
return T;

}



delete (8)

$x=8$ $T=7$

delete (8, 7)

① $x < T$ $8 < 7$ false
 $8 > 7$ true

$T \rightarrow \text{right} = \text{delete}(x, T \rightarrow \text{right})$
 $= \text{delete}(8, 7 \rightarrow \text{right})$
 $= \text{delete}(8, 10)$

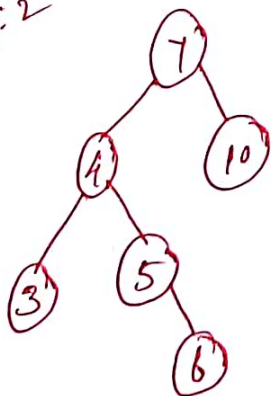
② delete (^x8, ^T10) $8 < 10$ true

$T \rightarrow \text{left} = \text{delete}(x, T \rightarrow \text{left})$
 $= \text{delete}(8, 8)$

③ delete (8, 8) $8 < 8$ X
 $8 > 8$ X

else
 Tempcell = T (8)
 $T = T \rightarrow \text{right}$
 $T \rightarrow \text{right} = \text{NULL}$
 free (tempcell)
 return T

Eg: 2



delete (5)

step 1:

delete (5, T)

delete (5, 7)

$5 < 7$ ✓

$T \rightarrow \text{left} = \text{delete}(5, T \rightarrow \text{left})$
 $= \text{delete}(5, 4)$

step 2:

delete (5, 4)

$5 < 4$ X

5 > 4 True

T → right = delete (5, 4 → right)
= delete (5, 5)

Step 3: delete (5, 5)

5 < 5 X

5 > 5 X

T → left & T → right (false)

bec'z it has only right child

else

Tempcell = 5

if (T → left == NULL) True

T = T → right

T = 6

free (Tempcell)

free (5)

return 6

when call delete function, call step 2, hence T = 4

T → right = 6,

4 → right = 6

