DATA STRUCTURES

Binary Search Tree

By
Zainab Malik

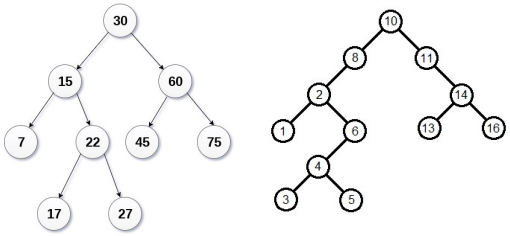
Content

- Binary Search Tree
  - Representation of Binary Tree
    - Array Representation
    - Linked List Representation
- Operations of Binary and Binary Search Trees
  - Insertion(item)
  - Traversing
    - In-order traversal
    - Post-order traversal
    - Pre-order traversal
  - Search(item)
  - FindSuccessor(item)
  - Delete(item)

Binary search Tree (BST)

- A Binary search tree is a tree that satisfies the following properties
  - Every element has the key (content) and no other node has the same key i.e. keys are unique
  - The keys, if any, in the left sub tree of the root are small than the key in the node
  - The keys, if any, in the right sub tree of the root are larger than the key in the node
  - The left and right sub tree of root are also binary search trees

Binary search Tree (BST) - Examples



(a) (b)

Representation of BST

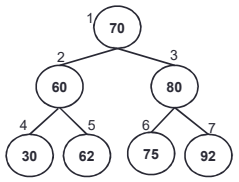
- The binary tree and a binary search tree are represented in an identical manner.
- These can be represented using
  - Linear Array
  - Linked List

Array Representation

- In this representation, each node of tree is assigned a number, as we did in extended binary tree, then each node is stored in the array at the index corresponding to its number.
- A BT/BST of height h requires an array of size $(2^h - 1)$

7

Array Representation

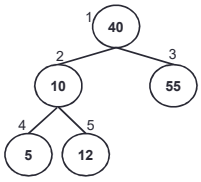


- size of array = $(2^h - 1)$
- h=3
- Size of array = $(2^3 - 1)$
- **Size of array = 7**

1	2	3	4	5	6	7
70	60	80	30	62	75	92

8

Array Representation

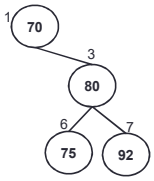


- size of array = $(2^h - 1)$
- h=3
- Size of array = $(2^3 - 1)$
- **Size of array = 7**

1	2	3	4	5	6	7
40	10	55	5	12		

9

Array Representation

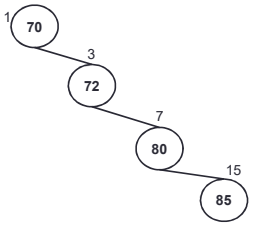


- size of array = $(2^h - 1)$
- h=3
- Size of array = $(2^3 - 1)$
- **Size of array = 7**

1	2	3	4	5	6	7
70		80			75	92

10

Array Representation

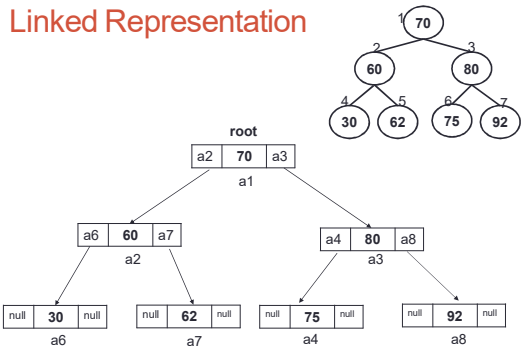


- size of array = $(2^h - 1)$
- h=4
- Size of array = $(2^4 - 1)$
- **Size of array = 15**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
70		72				80								85

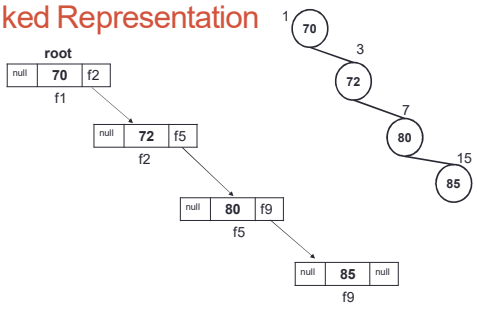
11

Linked Representation



12

Linked Representation



13

Operations of BST

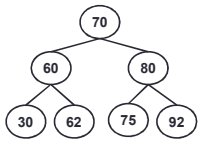
- Insertion
- Traversing
  - Pre-order traversal
  - In-order traversal
  - Post-order traversal
- Search (loc and ploc)
- FindSuccessor: *the smallest value in the RST or the largest value in the LST* ***(sloc, psloc)***
- Deletion

14

Insertion

- Insertion(item):
  - If tree is empty then insert item as root node
  - If item is less than the root node, insert item in the LST of root node
  - If item is greater than the root node, insert item in the RST of root node

To insert item 35:
Compare 35 with root i.e. 70, as 35<70 so move to LST
In LST root is 60, compare 35 with root i.e. 60, as 35<60, so move to its LST
In LST the root is 30, compare 35 with root i.e. 30, as 35 is greater than 30 so move to its RST
As RST is empty add node



15

Insertion

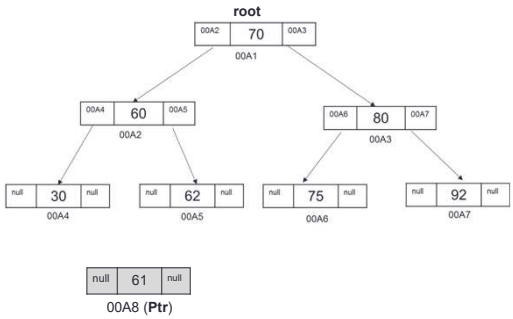
Insert(item):

```
1.  Create a node ptr as (null, item, null) //(left address, info, right address)
2.  If root==null
3.        set root=ptr
4.        return
5.  EndIf
6.  set parentPtr=ptr
7.  set nodePtr=root
8.  Repeat steps 9-14 while (nodePtr!=null)
9.        set parentptr=nodeptr
10.       If (item<nodeptr->info) then
11.          set nodeptr=nodeptr->left
12.       else
13.          set nodeptr=nodeptr->right
14.       EndIf
15. Endwhile
16. if (item< parentptr->info)
17.       set parentptr->left=ptr
18. else
19.       set parentptr->right=ptr
20. Endif
21. return
```

16

Example: insertion (61)

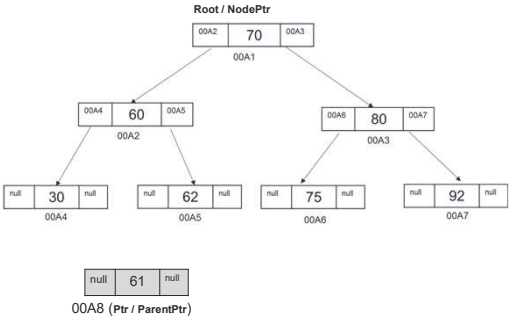
1. Create a node ptr as (null, item, null)



17

Example: insertion (61)

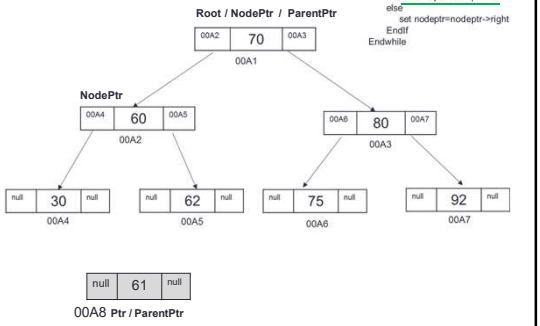
set parentPtr=ptr
set nodePtr=root



18

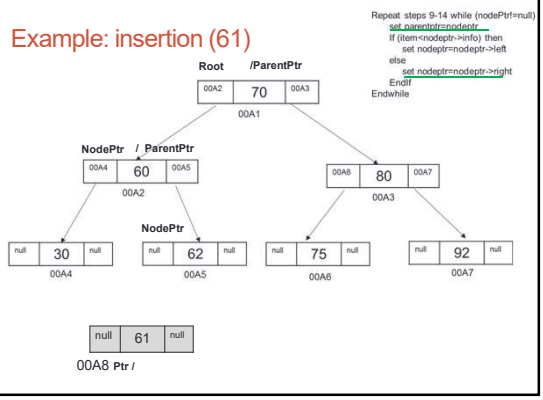
Example: insertion (61)

Repeat steps 9-14 while (nodePtr!=null)
set parentptr=nodeptr
If (item<nodeptr->info) then
set nodeptr=nodeptr->left
else
set nodeptr=nodeptr->right
EndIf
Endwhile



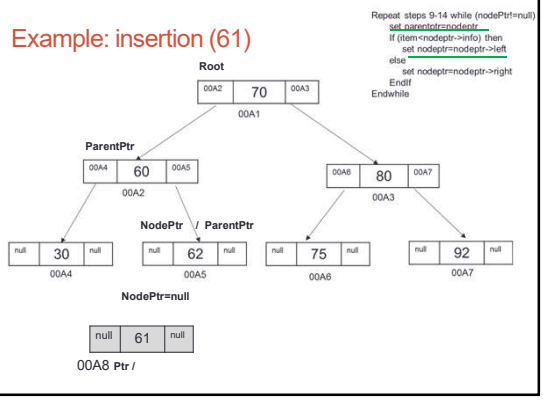
19

Example: insertion (61)



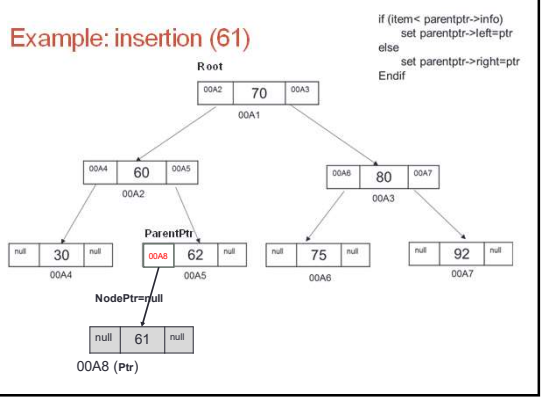
20

Example: insertion (61)



21

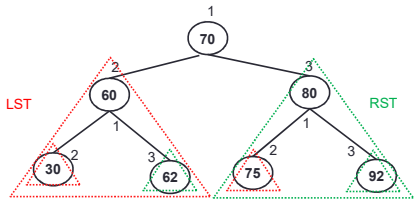
Example: insertion (61)



22

Pre-order traversal

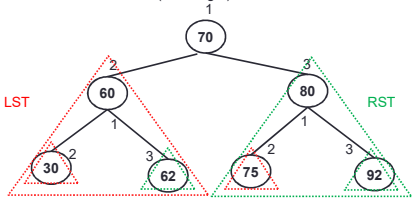
- Pre-order traversal (root)
  - Process the root R
  - Traverse the LST (left sub tree) of R in pre-order
  - Traverse the RST (right sub tree) of R in pre-order



23

Pre-order traversal

- Pre-order-traversal (root)
  - If tree is not empty:
    - Print root->info
    - Pre-order-traversal (root->left)
    - Pre-order-traversal (root->right)



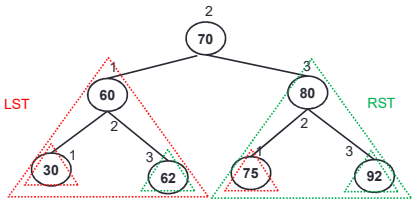
24

- Replace-recursively (*ptr,element)
  - If tree is not empty and ptr info is greater or equal to element:
    - Replace-recursively(ptr->getLeft())
    - Replace-recursively(ptr->getRight ())
    - Ptr->setInfo(0)

25

In-order traversal

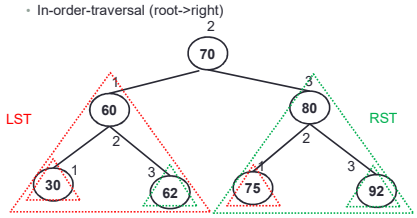
- In-order traversal (root)
  - Traverse the LST (left sub tree) of R as in-order
  - Process the root R
  - Traverse the RST (right sub tree) of R as in-order



26

In-order traversal

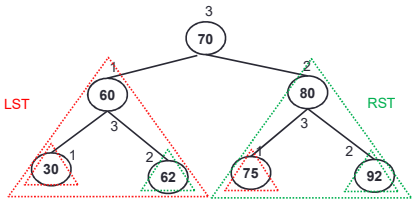
- In-order-traversal (root)
  - If tree is not empty:
    - In-order-traversal (root->left)
    - Print root->info
    - In-order-traversal (root->right)



27

Post-order traversal

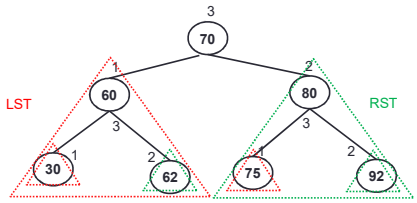
- Post-order traversal (root)
  - Traverse the LST (left sub tree) of R as post-order
  - Traverse the RST (right sub tree) of R as post-order
  - Process the root R



28

Post-order traversal

- Post-order-traversal (root)
  - If tree is not empty:
    - Post-order-traversal (root->left)
    - Post-order-traversal (root->right)
    - Print root->info



29

Search

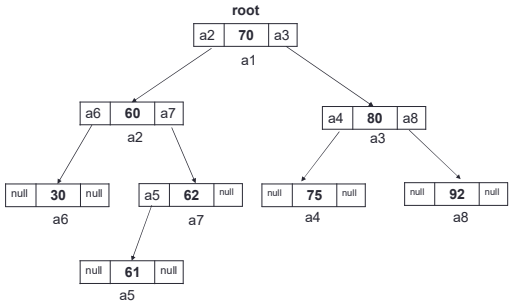
Search(item):

```
1.  Set loc=ploc=Null
2.  If (root == NULL) then
        Display message that tree is empty
        return false
    Endif

3.  Set loc=root
4.  Repeat steps 8-7 while (loc != NULL)
5.      If (item =loc->info) then
            Display message that element is found
            Return true
        Endif
6.      Set ploc=loc
7.      If(item<loc->info)
8.              Set loc=loc->left
        else
            Set loc=loc->right
        Endif
    Endwhile
9.  Display message that element does not exist
9.  Return false
```

30

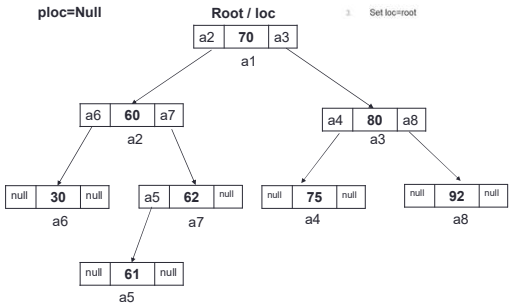
Example: search (61)



31

Example: search (61)

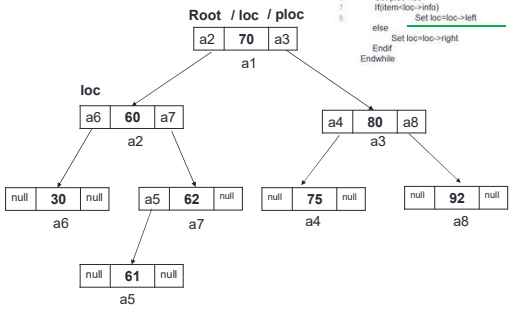
```
1.  Set loc=ploc=Null
2.  If (root == NULL) then
        Display message that tree is empty
        return false
    Endif
3.  Set loc=root
```



32

Example: search (61)

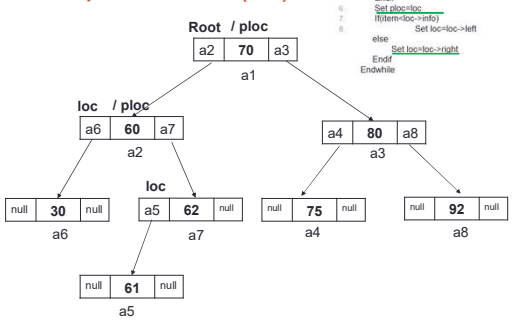
```
4.  Repeat steps 8-7 while (loc != NULL)
5.      If (item =loc->info) then
            Display message that element is found
            Return true
        Endif
6.      Set ploc=loc
7.      If(item<loc->info)
8.              Set loc=loc->left
        else
            Set loc=loc->right
        Endif
    Endwhile
```



33

Example: search (61)

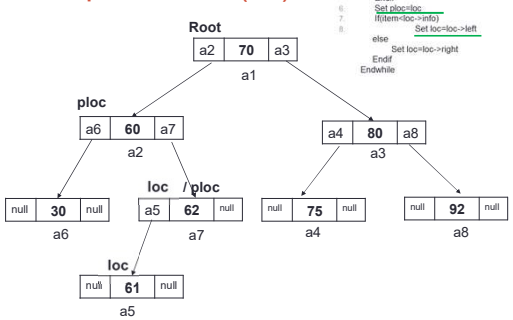
```
4.  Repeat steps 8-7 while (loc != NULL)
5.      If (item =loc->info) then
            Display message that element is found
            Return true
        Endif
6.      Set ploc=loc
7.      If(item<loc->info)
8.              Set loc=loc->left
        else
            Set loc=loc->right
        Endif
    Endwhile
```



34

Example: search (61)

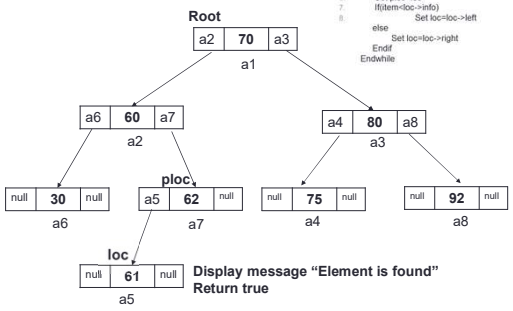
```
4.  Repeat steps 8-7 while (loc != NULL)
5.      If (item =loc->info) then
            Display message that element is found
            Return true
        Endif
6.      Set ploc=loc
7.      If(item<loc->info)
8.              Set loc=loc->left
        else
            Set loc=loc->right
        Endif
    Endwhile
```



35

Example: search (61)

```
4.  Repeat steps 8-7 while (loc != NULL)
5.      If (item =loc->info) then
            Display message that element is found
            return true
        Endif
6.      Set ploc=loc
7.      If(item<loc->info)
8.              Set loc=loc->left
        else
            Set loc=loc->right
        Endif
    Endwhile
```



36

FindSuccessor(sroot):

FindSuccessor(sroot): *the smallest value in the RST or the largest value in the LST*

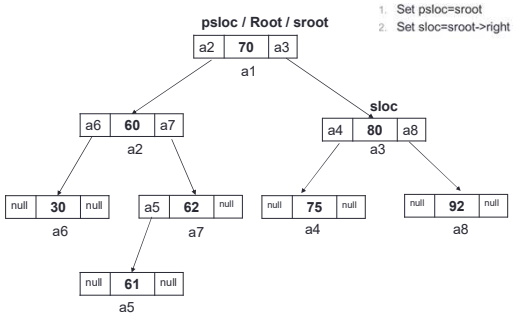
1. Set psloc=sroot
2. Set sloc=sroot->right
3. Repeat steps 4-5 while (sloc->left!= Null)
4. Set psloc = sloc
5. Set sloc = sloc->left

Endwhile

37

Example: FindSuccessor(a1)

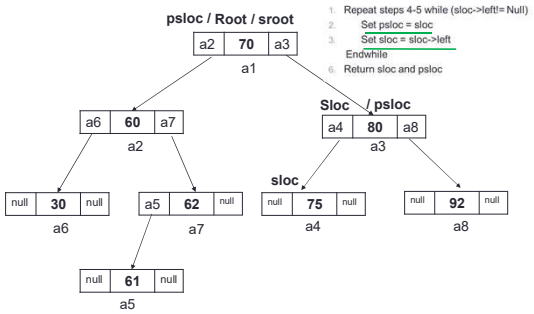
1. Set psloc=sroot
2. Set sloc=sroot->right



38

Example: FindSuccessor(a1)

1. Repeat steps 4-5 while (sloc->left!= Null)
2. Set psloc = sloc
3. Set sloc = sloc->left
   Endwhile
6. Return sloc and psloc



Successor: *the smallest value in the RST is at sloc (**75**)*

39

FindSuccessor(sroot):

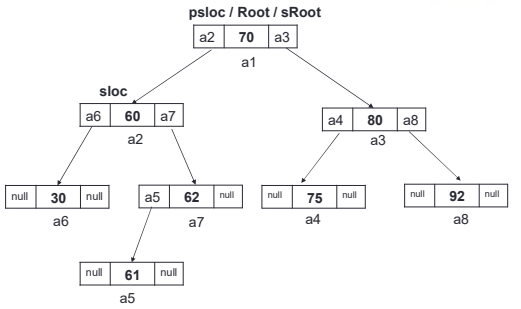
FindSuccessor(sroot): *the smallest value in the RST or the largest value in the LST*

1. Set psloc=sroot
2. Set sloc=sroot->left
3. Repeat steps 4-5 while (sloc->right!= Null)
4. Set psloc = sloc
5. Set sloc = sloc->right
   Endwhile

40

Example: FindSuccessor(a1)

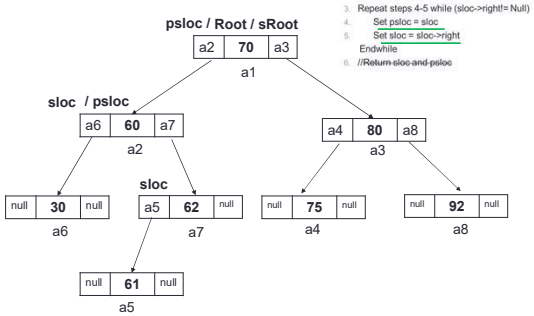
1. Set psloc=sroot
2. Set sloc=sroot->left



41

Example: FindSuccessor(a1)

3. Repeat steps 4-5 while (sloc->right!= Null)
4. Set psloc = sloc
5. Set sloc = sloc->right
   Endwhile
6. //Return sloc and psloc

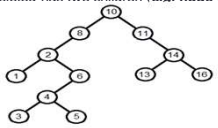


Successor: *the largest value in the LST is at sloc (**62**)*

42

Deletion (item)

- The deletion function/algorithm first search the item to be deleted as loc and keep the record of its parent node in ploc then it deletes/remove a node.
- After node deletion, the remaining tree must also be Binary Search Tree.
  - Depending on the number of children, there can be **three different cases**;
    1. Node to be deleted has no children (***e.g. node 5***)
    2. Node to be deleted has only one child (***e.g. node 11***)
    3. Node to be deleted has two children (***e.g. node 2***)



43

Delete (item)

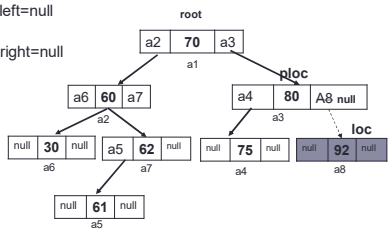
- If BST is empty
  - Display error "Nothing to delete"
  - Return
- else
  - search(item) // will provide Loc & ploc
  - If found:
    - If Case 1 // implementation of case 1
    - If Case 2 // implementation of case 2
    - If Case 3
      - **findSuccesor(loc) // will provide Sloc & psloc**
      - **Case 3 // implementation of case 3**
  - **If not found**
    - **Display message "Item to be deleted not found"**

44

Deletion Case 1: Node to be deleted has no children

Case 1: Node to be deleted has no children

```
If (loc->info < ploc->info)
    Set ploc->left=null
else
    Set ploc->right=null
Endif
```

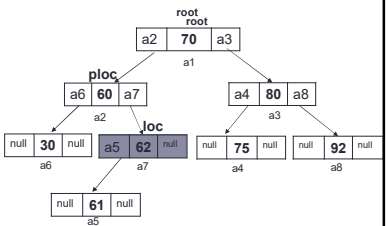


45

Deletion Case 2: Node to be deleted has only one child

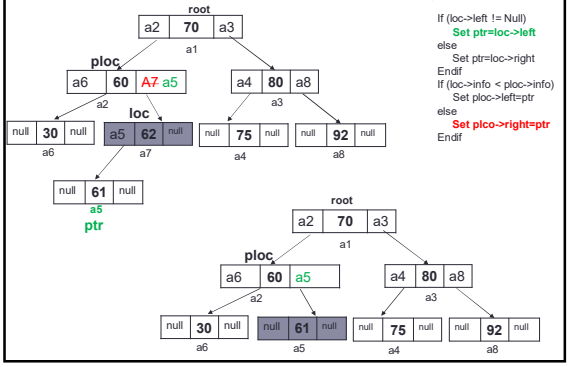
Case 2: Node to be deleted has only one child

```
If (loc->left != Null)
    Set ptr=loc->left
else
    Set ptr=loc->right
Endif
If (loc->info < ploc->info)
    Set plco->left=ptr
else
    Set ploc->right=ptr
Endif
```



46

Deletion Case 2: Node to be deleted has only one child




```
If (loc->left != Null)
    Set ptr=loc->left
else
    Set ptr=loc->right
Endif
If (loc->info < ploc->info)
    Set ploc->left=ptr
else
    Set plco->right=ptr
Endif
```

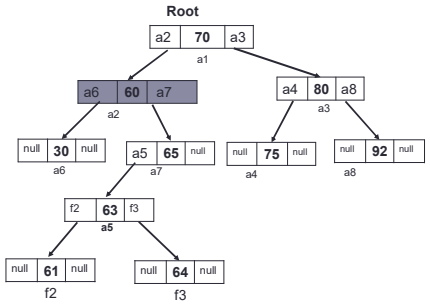
Deletion Case 3: Node to be deleted has two children

*Loc= node to be deleted, ploc= parent of loc // **will get by calling search(item)***
*Sloc= success of the node to be deleted, Psloc= parent of sloc // **will get by calling findSuccessor(item)***

1. **Set loc->info=sloc->info** //replace the value of node(to be deleted) with the value of successor node
2. **If (sloc->left = Null or sloc->right = Null)**
   - **If (psloc->left !=Null)**
     - **Set psloc->left=Null**
   - **else**
     - **Set psloc->right=Null**
   - **Endif**
   - **Else**
     - **If (sloc->left !=Null)**
       - **Set ptr=sloc->left**
     - **else**
       - **Set ptr=sloc->right**
     - **Endif**
     - **If (sloc->info < psloc->info)**
       - **Set psloc->left=ptr**
     - **else**
       - **Set psloc->right=ptr**
     - **Endif**
   - **Endif**
3. **Remove node sloc**

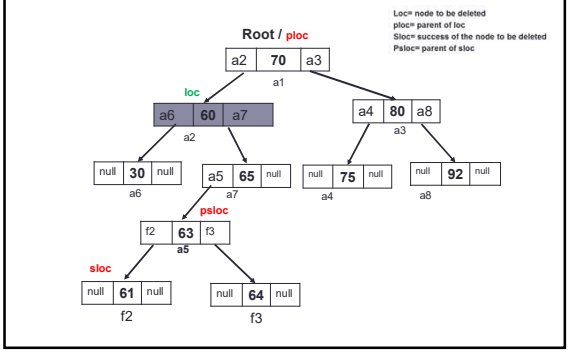
48

Deletion Case 3: Node to be deleted has two children



49

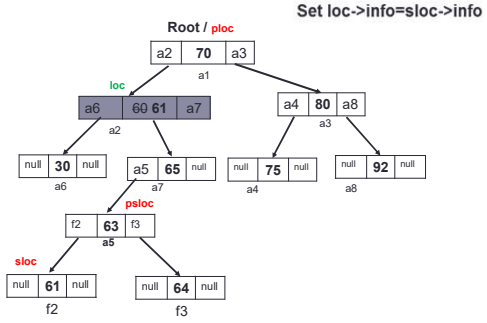
Deletion Case 3: Node to be deleted has two children



50

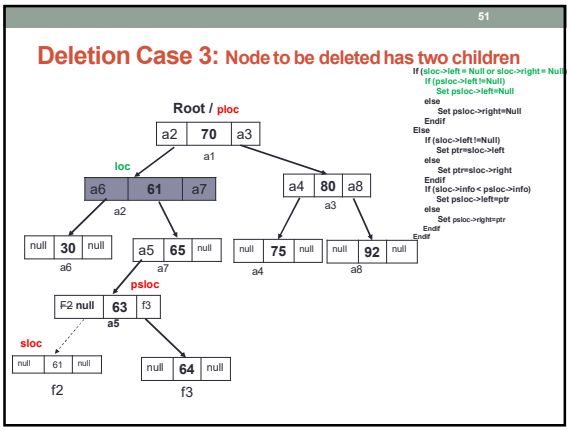
Deletion Case 3: Node to be deleted has two children

Set loc->info=sloc->info



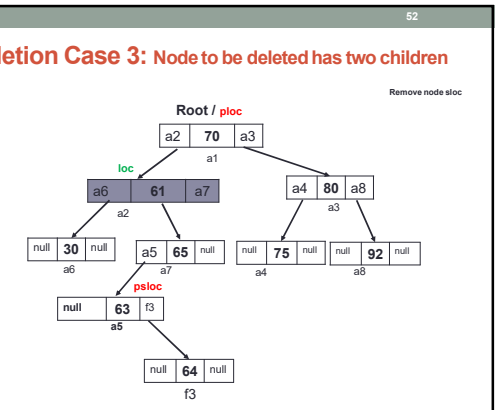
51

Deletion Case 3: Node to be deleted has two children

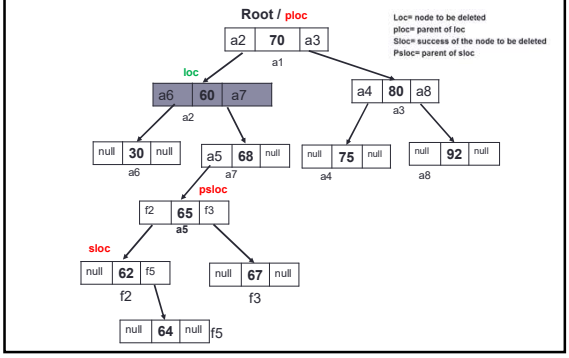


52

Deletion Case 3: Node to be deleted has two children

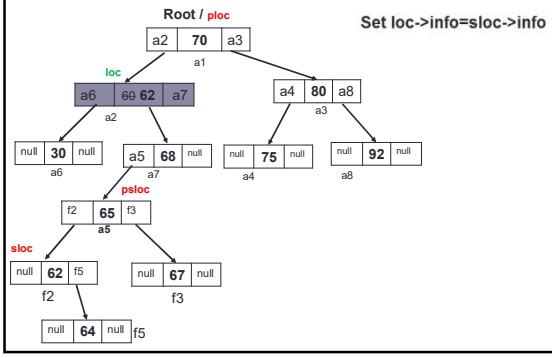


Deletion Case 3: Node to be deleted has two children

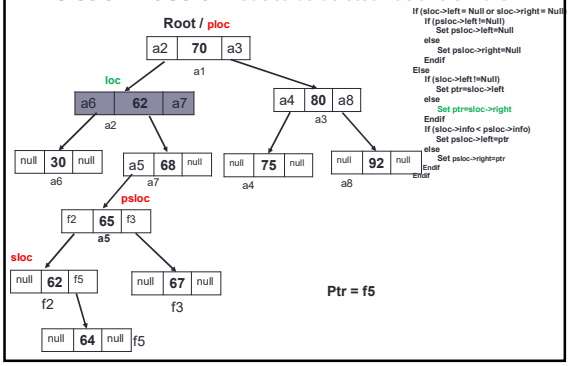


Deletion Case 3: Node to be deleted has two children

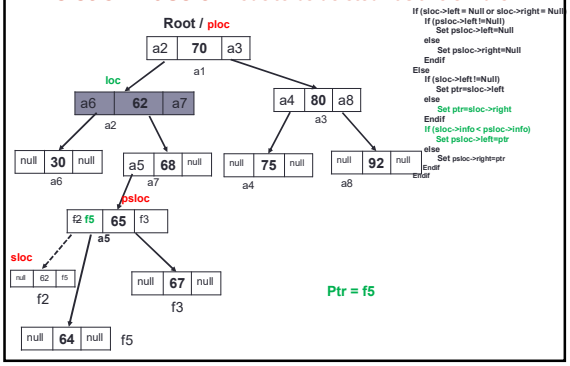
Set loc->info=sloc->info



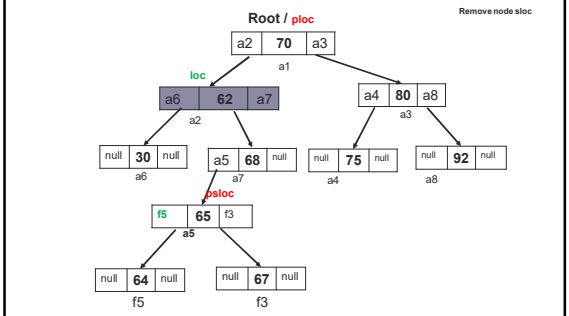
Deletion Case 3: Node to be deleted has two children



Deletion Case 3: Node to be deleted has two children



Deletion Case 3: Node to be deleted has two children



Delete (item)

- If BST is empty
  - Display error “Nothing to delete”
  - Return
- else
  - search(item) // will provide Loc & ploc
  - If Case 1 //entire implementation of case 1
  - If Case 2 //entire implementation of case 2
  - else
    - **findSuccesor(loc) // will provide Sloc & psloc**
    - **Case 3 // implementation of case 3**

Thank You