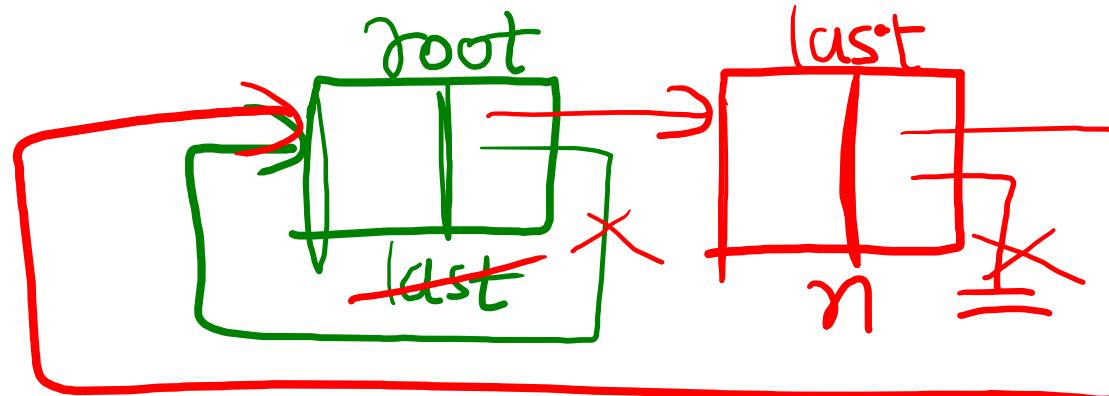
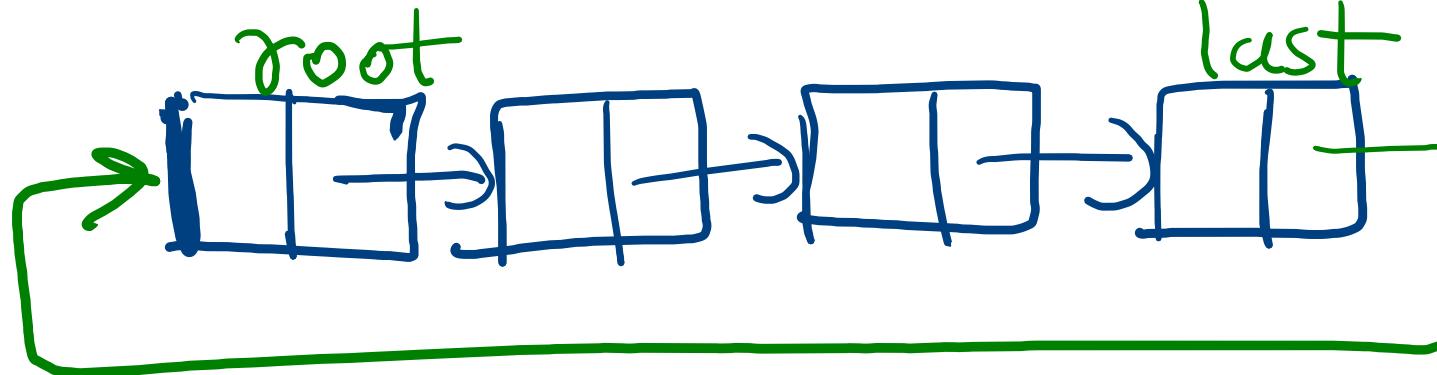
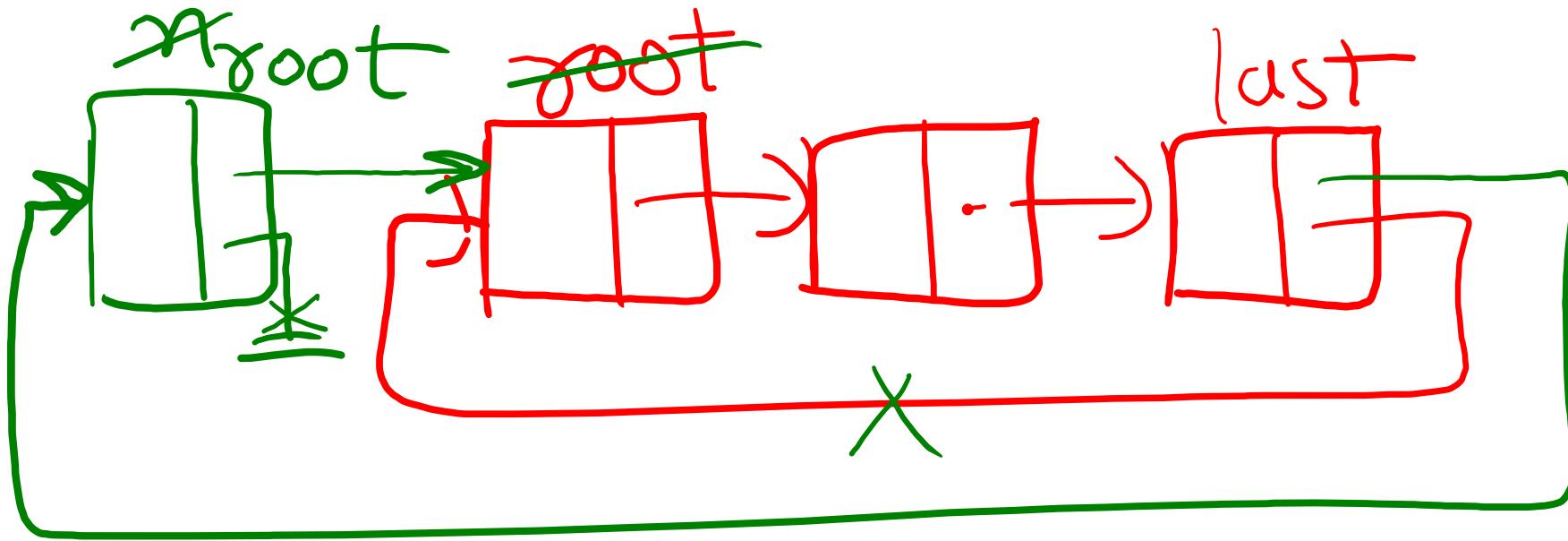


Circular





DataStructures_MET_2022 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

...va StackDemo.java QueueLinearDemo.java CircularQueueDemo.java PriorityQueueDemo.java LinkedListLinear.java DynamicStackDemo.java DynamicQueueDemo.java CircularLinkedListDemo.java

Source History Navigator

```
21 {  
22     Node n=new Node(data);  
23     if(root==null)  
24     {  
25         root=last=n;  
26         last.next=root; } }  
27 else  
28 {  
29     n.next=root; //1  
30     root=n; //2  
31     last.next=root; //3 } }
```

Diagram illustrating the insertion of a new node 'n' with data '1000' into a circular linked list. The list initially has one node 'root' with data '5000'. After insertion, the list becomes circular with three nodes: 'root' (5000), 'n' (1000), and 'last' (2000). Red annotations show the state before and after insertion, with arrows indicating pointer changes.



DataStructures_MET_2022 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

StackDemo.java QueueLinearDemo.java CircularQueueDemo.java PriorityQueueDemo.java LinkedListLinear.java DynamicStackDemo.java DynamicQueueDemo.java CircularLinkedListDemo.java

Source History

35 void deleteLeft()
36 {
37 if(root==null)
38 {
39 System.out.println("List Empty");
40 }
41 else
42 {
43 Node t=root; //1
44 root=root.next; //2
45 System.out.println("Deleted:"+t.data);
46 }
47 }
48 void insertRight(int data)

Diagram illustrating the deletion of the leftmost node from a circular linked list. It shows three nodes: 'root' (the head), 't' (the current node being processed), and 'last' (the previous node). A red box highlights the code for handling an empty list. A blue box highlights the assignment of 'root' to its next node. Red arrows show the connections between nodes and the flow of pointers. A blue arrow points to the assignment of 'root'. A red 'X' marks a point where a pointer would normally be, indicating it's null.



Search

ENG
IN09:20
28-11-2022

DataStructures_MET_2022 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

StackDemo.java QueueLinearDemo.java CircularQueueDemo.java PriorityQueueDemo.java LinkedListLinear.java DynamicStackDemo.java DynamicQueueDemo.java CircularLinkedListDemo.java

Source History

35 void deleteLeft()
36 {
37 if(root==null)
38 {
39 System.out.println("List Empty");
40 }
41 else
42 {
43 Node t=root;//1
44 root=root.next;//2
45 last.next=root;//3
46 System.out.println("Deleted:"+t.data);
47 }
48 }

Diagram illustrating the deletion of the leftmost node from a circular linked list:

The diagram shows a circular linked list with three nodes. The head of the list is labeled "root". The data value of the first node is "1000". A pointer from the "root" label points to the first node. The last node in the list is labeled "last" with a data value of "1000". A pointer from the "last" label points back to the first node. A variable "t" is shown pointing to the first node.

Output

Search

45:28 09:21

ENG IN 28-11-2022

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

StackDemo.java QueueLinearDemo.java CircularQueueDemo.java PriorityQueueDemo.java LinkedListLinear.java DynamicStackDemo.java DynamicQueueDemo.java CircularLinkedListDemo.java

Source History

```

41
42
43
44
45
46
47
48
49
50
51
52
53
54

```

else

```

41
42
43
44
45
46
47
48
49
50
51
52
53
54

```

{

Node t=root; //1

if(root.next==root)

```

41
42
43
44
45
46
47
48
49
50
51
52
53
54

```

{

root=last=null; //single node deletion

}

else

```

41
42
43
44
45
46
47
48
49
50
51
52
53
54

```

{

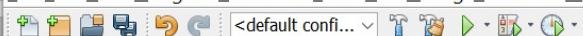
root=root.next; //2

last.next=root; //3

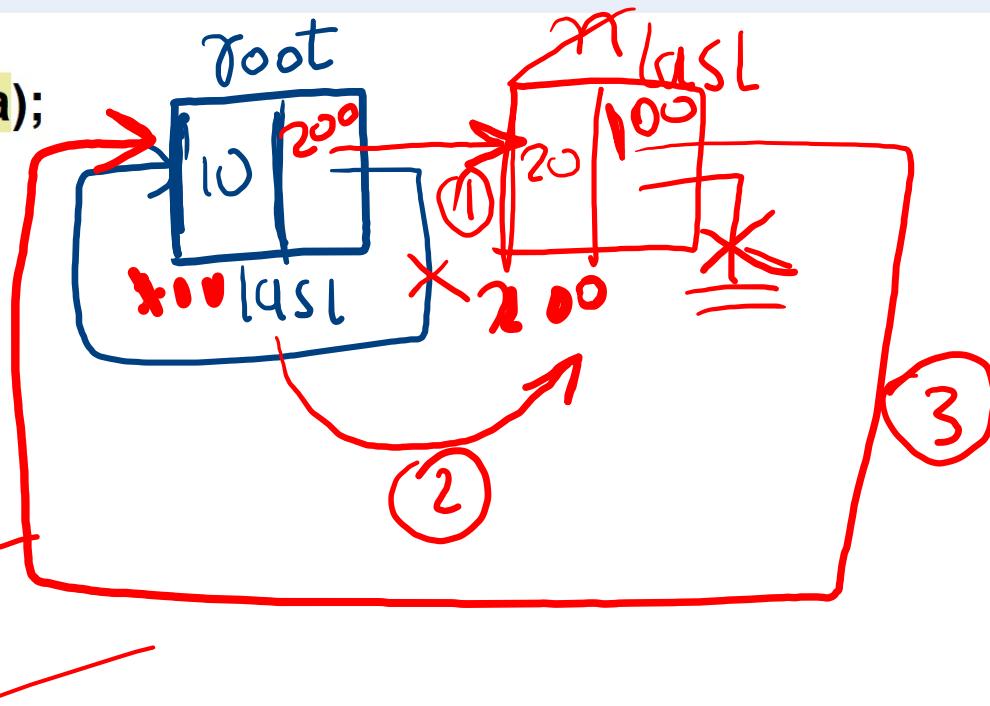
}

System.out.println("Deleted:"+t.data);

Diagram illustrating the deletion of a single node from a circular linked list. It shows three nodes: **root**, **t**, and **last**. The **root** pointer is set to **null**, and the **last** pointer is also set to **null**, indicating a single-node deletion. The **next** pointer of the previous node (**t**) is updated to point to the next node (**last**).



```
56 void insertRight(int data)
57 {
58     Node n=new Node(data);
59     if(root==null)
60     {
61         root=last=n;
62         last.next=root;
63     }
64     else
65     {
66         last.next=n;//1
67         last=n;//2 ✓
68         last.next=root;//3
69     }
}
```



DataStructures_MET_2022 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

StackDemo.java QueueLinearDemo.java CircularQueueDemo.java PriorityQueueDemo.java LinkedListLinear.java DynamicStackDemo.java DynamicQueueDemo.java CircularLinkedListDemo.java

Source History

if(root.next==root)
{
 root=last=null;
}
else
{
 while(t!=last)//2
 {
 t2=t;
 t=t.next;
 }
 last=t2;
 last.next=root;//ref root
}

t/t2

root/last - null

root last last
t t2 t2

DataStructures_MET_2022 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Source History Navigator Projects Files Services

```
83     root=last=null;
84 }
85 else
86 {
87     while(t!=last)//2
88     {
89         t2=t;
90         t=t.next;
91     }
92     last=t2;
93     last.next=root;//re ref root
94 }
95 System.out.println("Deleted:"+t.data);
96 }
```

Diagram illustrating the deletion of the second node (data 200) from a circular linked list. The list consists of four nodes with data values 100, 200, 300, and 400. The list is circular, with each node's next pointer pointing to the next node in sequence. Red annotations show the state before deletion:

- The first node (100) has its original next pointer to the second node (200) crossed out.
- A new next pointer is drawn from the first node (100) to the third node (300).
- The variable *t* is annotated with a red arrow pointing to the second node (200).
- The variable *t2* is annotated with a red arrow pointing to the first node (100).
- The variable *root* is annotated with a red arrow pointing to the first node (100).
- The variable *last* is annotated with a red arrow pointing to the fourth node (400).

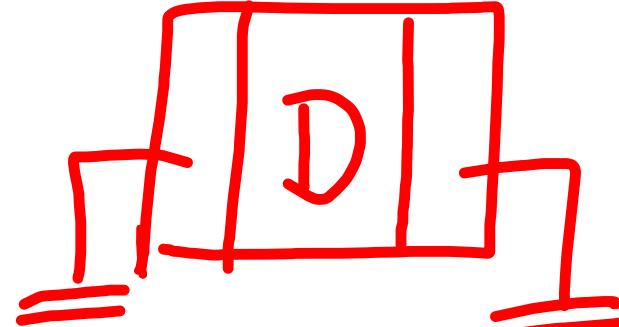
Output window:

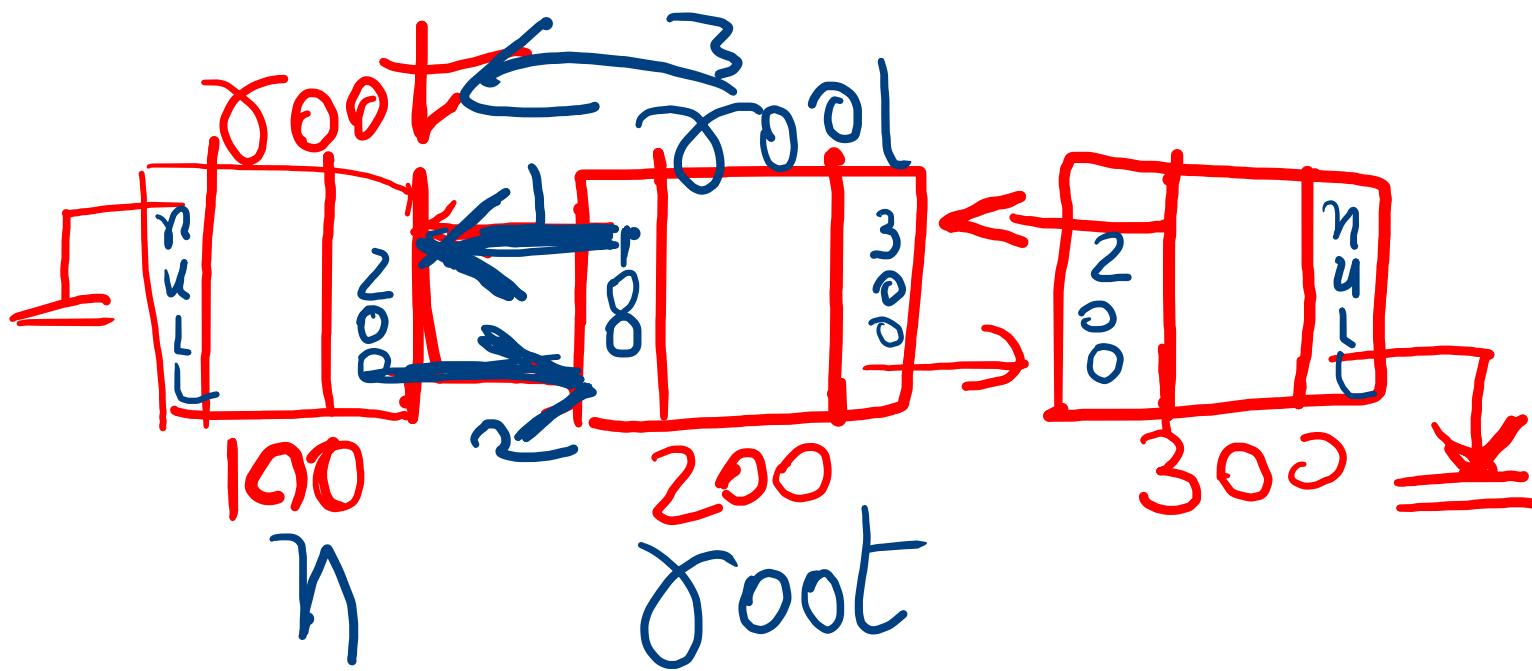
Search bar: Search (Ctrl+I)

System.out.println("Deleted:"+t.data);

Bottom status bar: ENG IN 10:14 28-11-2022

```
...va CircularQueueDemo.java x PriorityQueueDemo.java x LinkedListLinear.java x DynamicStackDemo.java x DynamicQueueDemo.java x CircularLinkedListDemo.java x DoublyLinkedListDemo.java x
Source History Navigator Projects Files Services
10 */
11 class Dnode
12 {
13     int data;
14     Dnode left,right;
15     Dnode(int data)
16     {
17         this.data=data;
18         left=right=null;
19     }
20 }
21 public class DoublyLinkedListDemo {
22
23 }
```



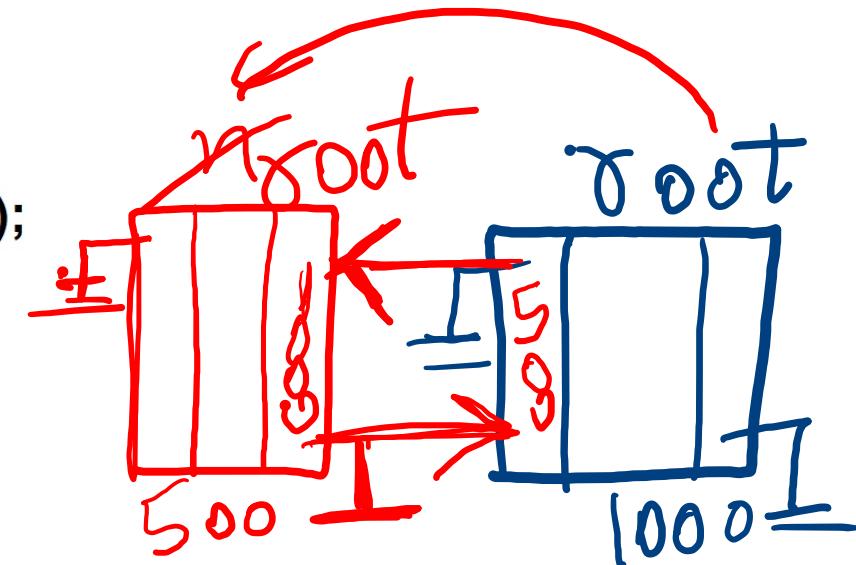


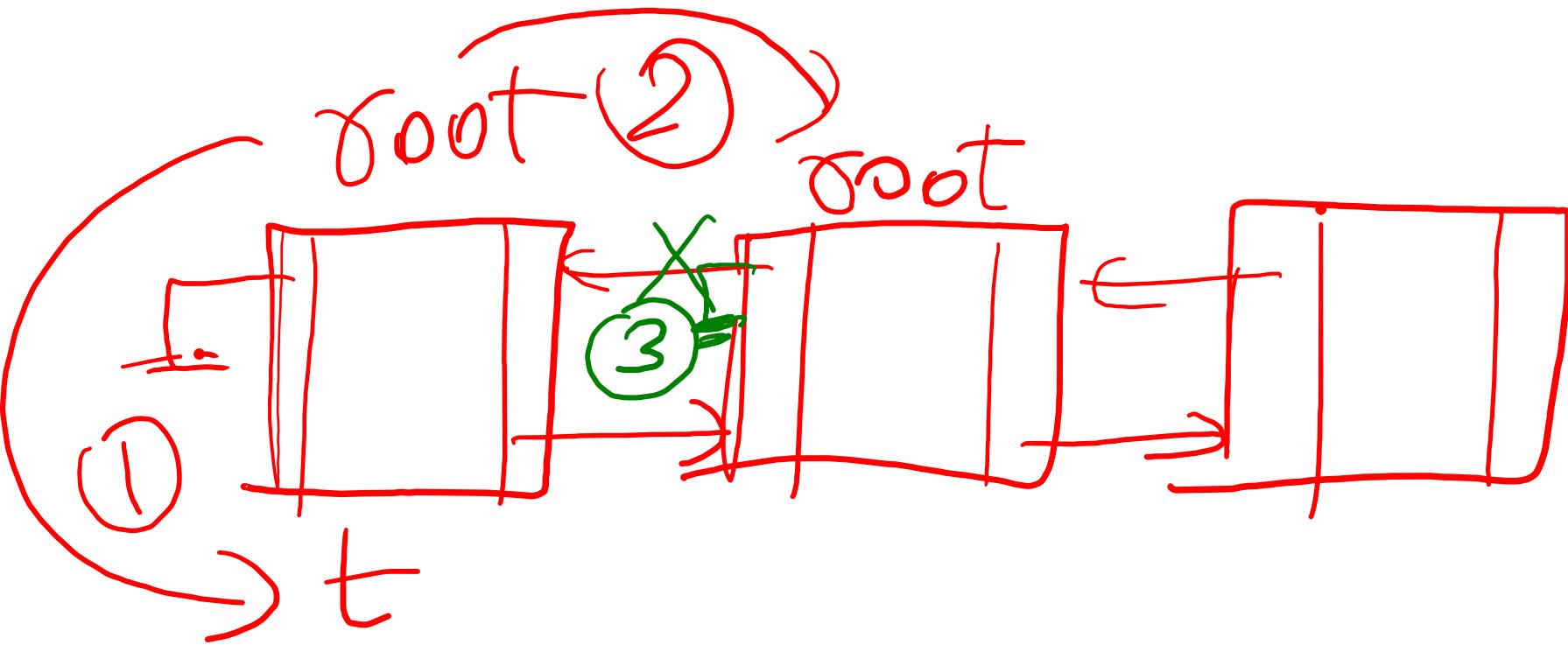
...va CircularQueueDemo.java x PriorityQueueDemo.java x LinkedListLinear.java x DynamicStackDemo.java x DynamicQueueDemo.java x CircularLinkedListDemo.java x DoublyLinkedListDemo.java

Source History

Navigation

26
27
28
29
30
31
32
33
34
35
36
37
38
39





DataStructures_MET_2022 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

CircularQueueDemo.java PriorityQueueDemo.java LinkedListLinear.java DynamicStackDemo.java DynamicQueueDemo.java CircularLinkedListDemo.java DoublyLinkedListDemo.java

Source History Navigator Projects Files Services

```
void deleteLeft()
{
    if(root==null)
        System.out.println("Empty list");
    else
    {
        Dnode t=root; //1 ✓
        root=root.right; //2
        root.left=null; //3 ✓
        System.out.println("Deleted:"+t.data);
    }
}
```

Diagram illustrating the deletion of the leftmost node from a doubly linked list. The list consists of five nodes. A red arrow points to the second node from the left, labeled 'root'. A green circle labeled '1' is around the assignment 'root=root.right;'. A green circle labeled '2' is around the assignment 'root.left=null;'. A green circle labeled '3' is around the printed output 'Deleted:' + t.data;'. A red X is drawn over the original 'root' pointer in the list structure.

DataStructures_MET_2022 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Airplane mode off

```
void insertRight(int data)
{
    Dnode n=new Dnode(data);
    if(root==null)
        root=n;
    else
    {
        n.right=root; //1
        root.left=n; //2
        root=n; //3
    }
}
```

Diagram illustrating the insertion of a new node `n` with data 4 into a doubly linked list. The list initially has nodes with data 1, 2, and 3. Node 1 is the root. Red arrows show the connections: from `root` to 1, from 1 to 2, from 2 to 3, and from 3 to null. A green arrow points from the left of node 3 to node 4, indicating its insertion. Handwritten labels `root` above node 1 and `t` below node 2 are also present.



11:04

28-11-2022

DataStructures_MET_2022 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

CircularQueueDemo.java PriorityQueueDemo.java LinkedListLinear.java DynamicStackDemo.java DynamicQueueDemo.java CircularLinkedListDemo.java DoublyLinkedListDemo.java

Source History

73 }
74 else
75 {
76 Node t,t2;
77 t=t2=root;
78 while(t.next!=null)//2
79 {
80 t2=t;
81 t=t.next;
82 }
83 t2.next=null;//break link
84 System.out.println("Deleted:"+t.data);
85 }
86 }

Diagram illustrating the deletion of a node from a Doubly Linked List. The list contains nodes with data values 200, 800, and 1200. The head pointer 'root' points to the first node (200). The node with data 800 is highlighted in green. A red circle labeled '①' encircles the first node (200). A green circle labeled '③' encircles the node with data 800. A green circle labeled '②' encircles the node with data 1200. Yellow arrows indicate the movement of pointers: 't' moves from the first node to the second node; 't2' moves from the first node to the second node; and 't.next' moves from the second node to the third node. Red lines show the original connections between nodes. Handwritten annotations include 'root' above the first node, '200' above the first node, '800' above the second node, '1200' above the third node, and 't', 't2', and 't.next' pointing to their respective variables in the code.

Output

Search

77.2 11:08

ENG IN 28-11-2022

DataStructures_MET_2022 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Source History Navigator Projects Files Services

73 }
74 else
75 {
76 Dnode t,t2;
77 t=t2=root; ✓
78 while(t.right!=null)//2
79 {
80 t=t.right;
81 t2=t.left; ✓
82 t2.right=null;//break link
83 System.out.println("Deleted:"+t.data);
84 }
85 }
86 }

Diagram illustrating the deletion of a node from a Doubly Linked List. The list consists of three nodes: root (data 1000), t2 (data 1200), and t (data 1800). The diagram shows the pointer manipulation: t2's right pointer is set to null, and t's left pointer is set to null. Handwritten annotations include green boxes around the nodes and labels 'root', 't2', and 't' above them. Yellow arrows point to the 'right' and 'left' pointers being modified.

Output

Search

File Explorer

Task List

Properties

Navigator

Projects

Files

Services

Output

84 85 86

11:14 28-11-2022

11:08 28-11-2022

DataStructures_MET_2022 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

CircularQueueDemo.java PriorityQueueDemo.java LinkedListLinear.java DynamicStackDemo.java DynamicQueueDemo.java CircularLinkedListDemo.java DoublyLinkedListDemo.java

Source History Navigator Projects Files Services

```
88 void printList()
89 {
90     if(root==null)
91         System.out.println("List Empty");
92     else
93     {
94         Dnode t;
95         t=root;
96         while(t!=null)//2
97         {
98             System.out.println(t.data);
99             t=t.right;
100        }
101    }
```

Diagram illustrating a Doubly Linked List. The root node contains the value 10. Red arrows show the connections between nodes: a double-headed arrow between 10 and 20, and single-headed arrows pointing left from 20 to 10 and right from 30 to 20. The nodes are labeled 10, 20, 30, and 40.

DataStructures_MET_2022 - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

CircularQueueDemo.java PriorityQueueDemo.java LinkedListLinear.java DynamicStackDemo.java DynamicQueueDemo.java CircularLinkedListDemo.java DoublyLinkedListDemo.java

Source History

109
110 Dnode t;
111 t=root;
112 while(t.right!=null)//stop at last
113 { System.out.println(t.data);
114 t=t.right; }
115 while(t!=null)//stop when null
116 { System.out.println(t.data);
117 t=t.left; }
118 }
119 }
120 }
121 }
122 }

Dnode t;

t=root;

while(t.right!=null)//stop at last

{

System.out.println(t.data);

t=t.right;

}

while(t!=null)//stop when null

{

System.out.println(t.data);

t=t.left;

}

}

}

119 120 121 122

root

5 10 20

100 200 300

t t t t

20 10 5

Diagram illustrating the traversal of a Doubly Linked List. The list consists of three nodes with data values 5, 10, and 20. The nodes are represented as rectangles divided into four quadrants: top-left (data), top-right (right pointer), bottom-left (left pointer), and bottom-right (garbage). The list is circular, with each node's right pointer pointing to the next node and its left pointer pointing back to the previous node. The variable 't' is used to traverse the list. The first traversal (lines 113-114) starts at the root node (5) and moves to the right node (10), then to the right node (20). The second traversal (lines 118-119) starts at the right node (20) and moves to the left node (10), then to the left node (5). Handwritten annotations include 'root' above the first node, 't' with arrows pointing to each node during both traversals, and the data values 100, 200, and 300 written near the nodes.

Output

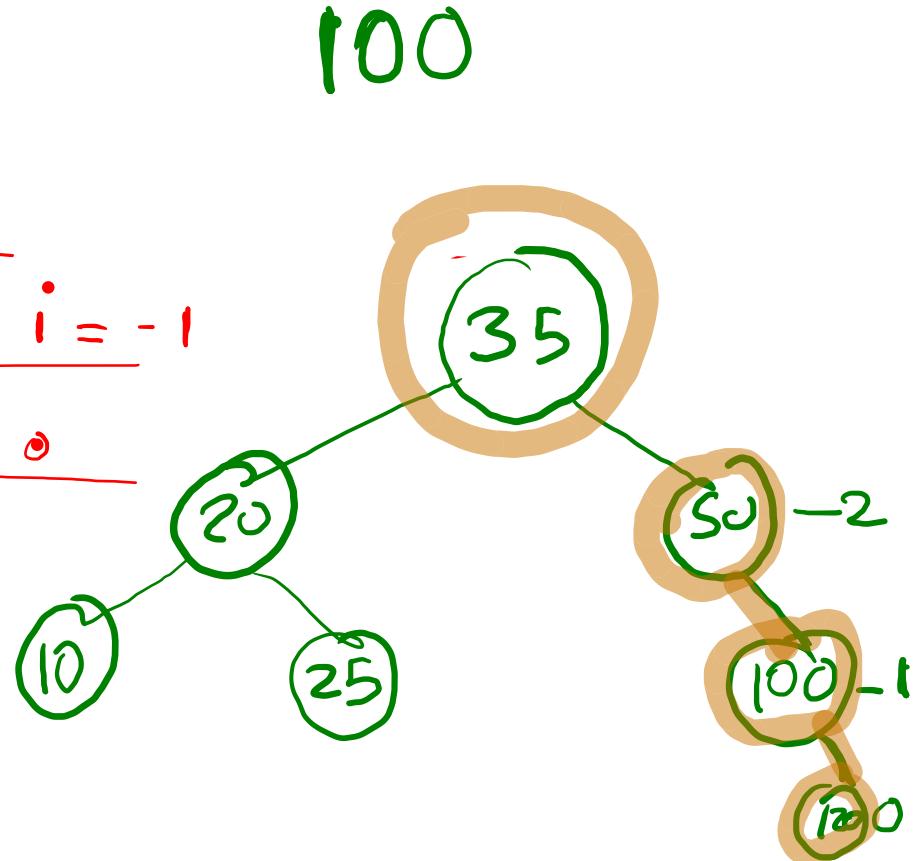
Search

119:23

ENG IN

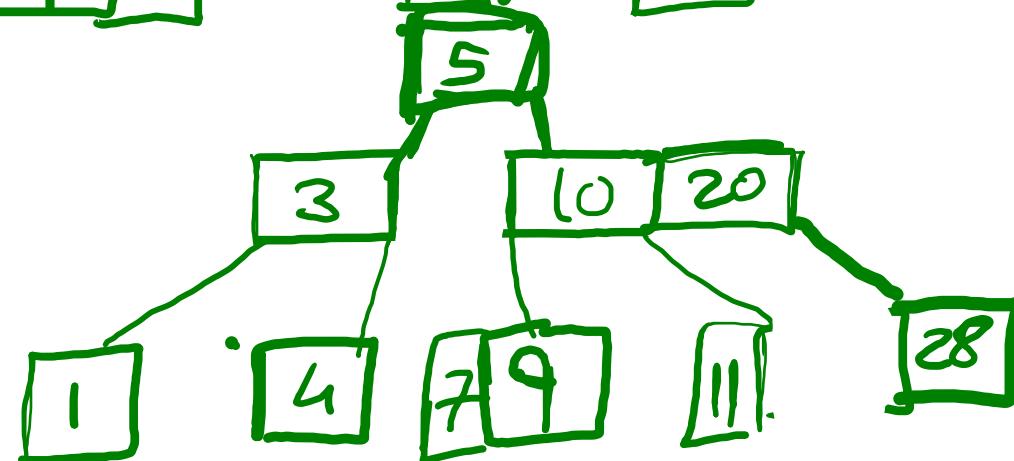
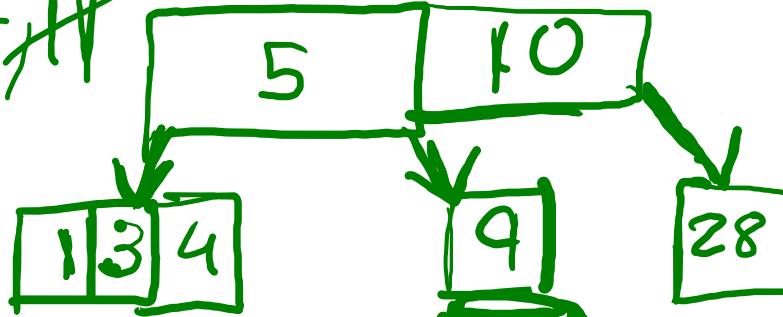
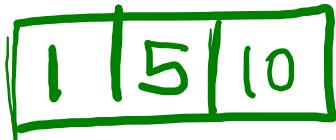
28-11-2022

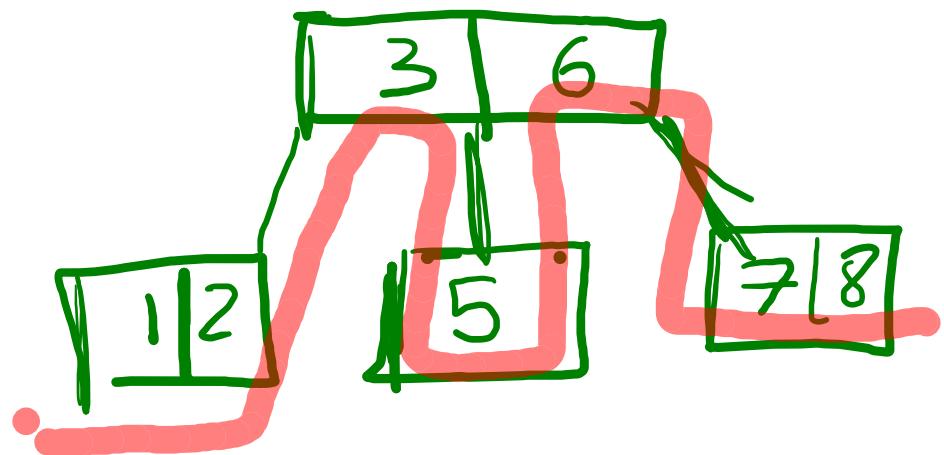
$$\begin{array}{r}
 1 \quad 1 \\
 20 \quad 1 - 3 = -2 \\
 \hline
 0 \quad 10 \quad 1 \\
 35 \quad 1 - 2 = -1 \\
 \hline
 0 \quad 25 \quad 0 \\
 50 \quad 0 - i = -1 \\
 \hline
 0 \quad 100 \quad 0
 \end{array}$$



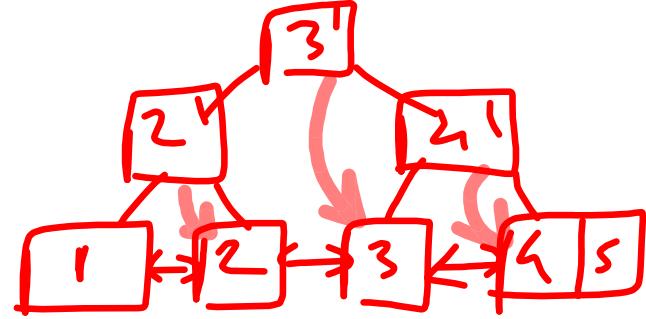
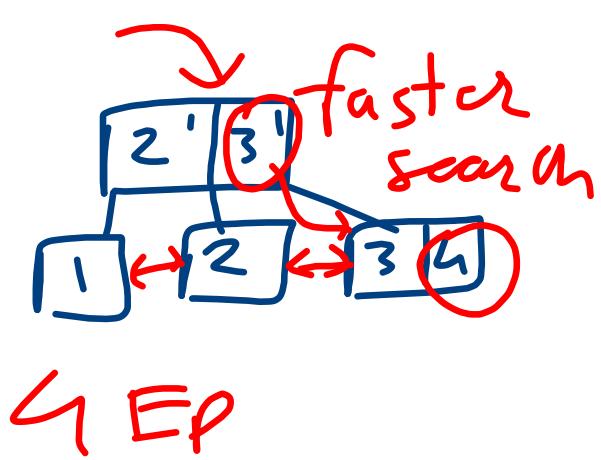
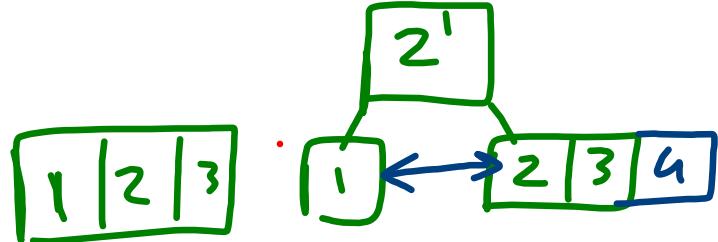
M=3 min=1 max=2 split=3

3, 10, 1, 3, 9, 28, 4, 11
20





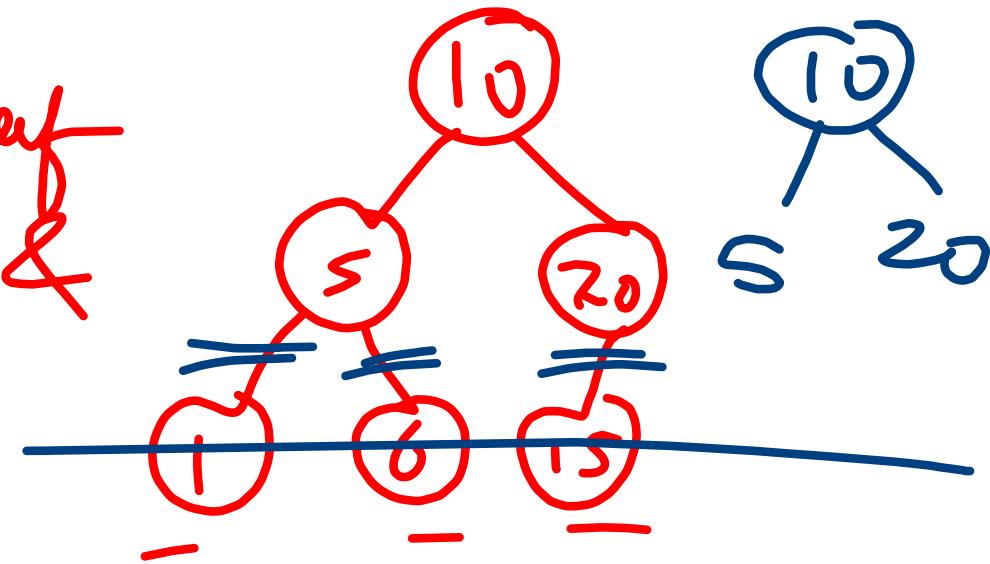
$M = 3$



Deletion

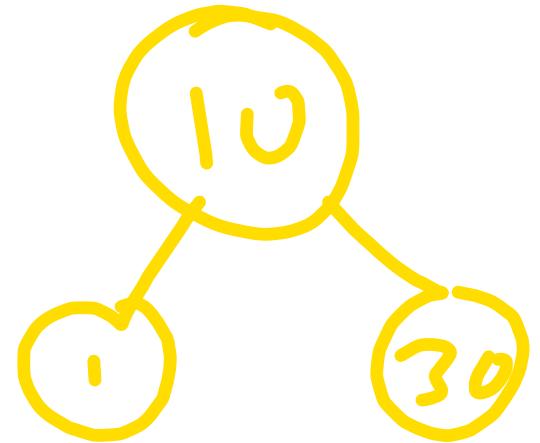
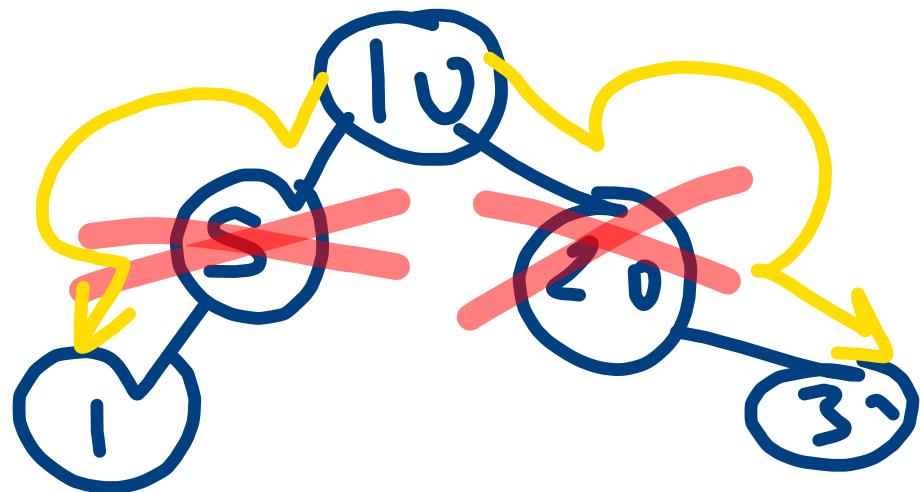
Case-1: Deletion of leaf

Sol: Cut parent's link &
remove node



Case 2: Parent with single
Child

Sol: Child takes over Parent



Case 3: Del with both sub

tree.

Sol: Inorder next takes over

5, 10, 15, 20, 30

