


(Refer Slide Time: 12:59)

find(v)

Recursive	Iterative
<pre>function find(t,v) if (t == NIL) return(False) if (t.value == v) return(True) if (v < t.value) return(find(t.left,v)) else return(find(t.right,v))</pre>	<pre>function find(t,v) while (t != NIL) { if (t.value == v) return(True) if (v < t.value) t = t.left else t = t.right } return(False)</pre>

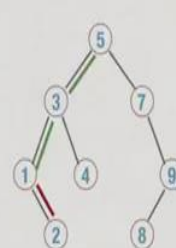



Now, like binary search you can do iterative version of this, so you start at the root and then so long as it is not nil you trust on the path. So, the current values we return true; otherwise, you walked on to the left are you actually start to the root and you kind of trace a path you can values are that find. And then when you reach are node you say yes, on the other hand if you reach a point where you cannot going further, if you run out of nodes, if you come all the way down and then you say there is no extension, where I can find it then we will safe also. So, this is the simple recursive and iterative version of find.

(Refer Slide Time: 13:34)

Minimum

- Left most node in the tree

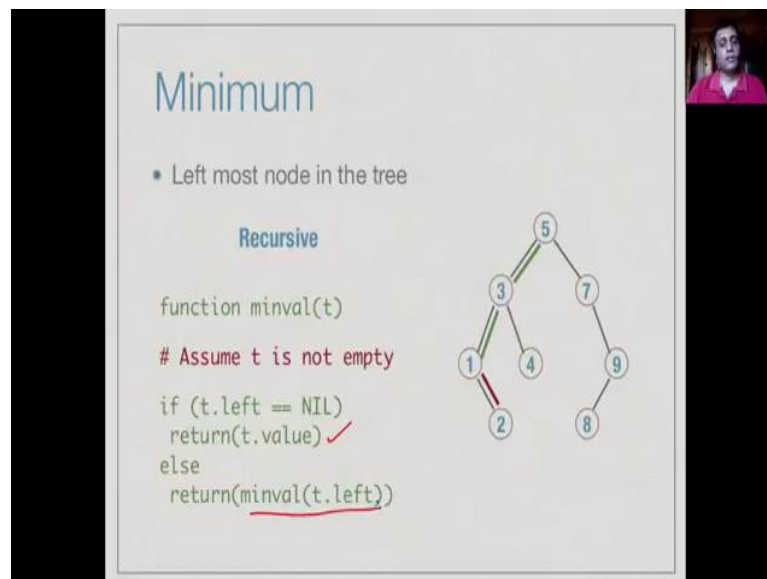




So, the next two operations insert we want to do your finding the minimum and the maximum. So, the minimum node in the tree is the left most node, in other wards it is the

node that you reach if you go and as far as you can follow only left edges. So, in this case from 5 for go left I go to 3, from 3 if I go to left I go to 1 and then I cannot go left there is something below one it is on the right, but never the less in cannot go left one must be the minimum element on the tree, because everything that is smaller than something will be 2 is left, so if I cannot go left is nothing smaller than 1. So, this is the reason why the left most value in the tree will be the minimum.

(Refer Slide Time: 14:14)



The slide is titled "Minimum" and contains the following content:

- Left most node in the tree

Recursive

```
function minval(t)
# Assume t is not empty
if (t.left == NIL)
return(t.value) ✓
else
return(minval(t.left))
```

The diagram shows a binary tree with root 5. Node 5 has a left child 3 and a right child 7. Node 3 has a left child 1 and a right child 4. Node 1 has a right child 2. Node 7 has a right child 9. Node 9 has a left child 8. The path from 5 to 3 to 1 to 2 is highlighted with green lines, and the edge from 1 to 2 is highlighted with a red line. A small video inset of a person is visible in the top right corner of the slide.

So, we can easily find it recursively, now we will typically use this assuming the trees not empty, it simplifies a little bit of coding. So, assuming the trees not empty we do not have to check any special condition and given a error when it is not empty, when it is empty. So, we assume it is not empty, so if we can go left we can we do, so if the t dot left is nil then this is the minimum value and we return otherwise, we recursively search for the minimum value on the left.

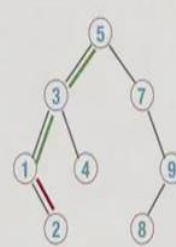
(Refer Slide Time: 14:47)

Minimum

- Left most node in the tree

Iterative

```
function minval(t)  
# Assume t is not empty  
while (t.left != NIL)  
    t = t.left  
return(t.value)✓
```



And again there is the very simple iterative version, we start with the current thing and we keep going left as long as we can. So, long as we do not it will nil and when reach this point why t dot left is nil, we come out with this loop and you return the value at this point, so here from since we would start with 5 t dot left is 3, t dot left is 1, t dot left is nil. So, this at this point we stop, so t is pointing to 1, so therefore t dot value is 1 and this is the value by return, so we can find the minimum.

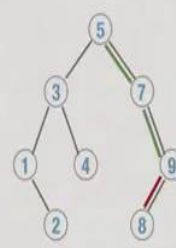
(Refer Slide Time: 15:14)

Maximum

- Right most node in the tree

Recursive

```
function maxval(t)  
# Assume t is not empty  
if (t.right == NIL)  
    return(t.value)  
else  
    return(maxval(t.right))
```



So, symmetrically the maximum is the right most value on the tree, as you go right you go bigger, if you cannot go right any more this is nothing which is bigger than that node. So, here from 5 we can go to 7 to 9 and 9 is the biggest value, because a next node which

all though it is below 9 in the tree it is through the left and therefore, smaller than 9. So, we have a symmetric recursive solution maxval, we checks the right if the right is nil then we are at the maximum we return the value if the right is not nil, then we recursively search the right for the maximum.

(Refer Slide Time: 15:46)

Maximum

- Right most node in the tree

Iterative

```
function maxval(t)
# Assume t is not empty
while (t.right != NIL)
t = t.right
return(t.value)
```

Find
Minimum
Maximum
Pred
Succ

And again iterative version of the same, we just follow chase point as to the right. So, as long as right t dot right is not nil, we move from t to t dot right and when we cannot move any more here of the maximum value, so we return that value. So, this point we have done find minimum and maximum in a search tree. So, these three operations we have done, now if you remember we have do predecessor and successor.

(Refer Slide Time: 16:20)

Successor

- succ(x) is what inorder(t) prints after x
- If x has a right subtree, min(right subtree)

t

So, let us first look at successor, so recall that the successor of x in the list, supposed to be the next value, the next smallest value after x the list. So, if we print it out in sorted order, it will be the value that would appear to the right of x and the in order traversal of a tree prints out the values in sorted order. So, it is effectively what in order would print immediately after x . So, we know that the way that in order works, it prints the left sub tree then it prints x , then it prints right sub tree.

So, if you want to one that comes immediately after x , it will be the first value here. So, the smallest value in the right sub tree, in other words the minimum of the right sub tree and we already know how to compute the minimum of a tree, but they could be a possibility that we want to successor of a value with does not have a right sub tree, then what we do.

(Refer Slide Time: 17:16)

Successor

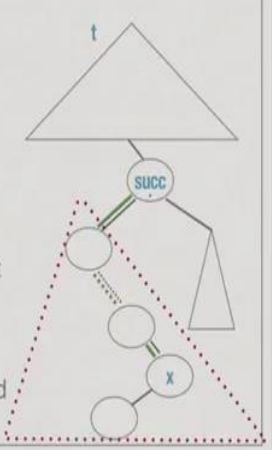
- $\text{succ}(x)$ is what $\text{inorder}(t)$ prints after x
- If x has a right subtree, $\text{min}(\text{right subtree})$
- if x has no right subtree
- x is max of the subtree it belongs to
- walk up to find where this subtree is connected

So, the first observation is that if a value has no right sub tree, then it must be the maximum of the sub tree to which it belongs. So, in this case we look at the value x , it has no right sub trees, so it is the maximum of some sub tree.

(Refer Slide Time: 17:29)

Successor

- $\text{succ}(x)$ is what $\text{inorder}(t)$ prints after x
- If x has a right subtree, $\text{min}(\text{right subtree})$
- if x has no right subtree
 - x is max of the subtree it belongs to
 - walk up to find where this subtree is connected



So, we have to first identify this sub tree, so how do we identify the sub tree, will be wake up and find out how this sub tree is connected. So, we get to x there following is sequence of right pointers. So, we follow those right pointers backwards and when we follow a left pointer, we know that we have come out of this sub tree. So, therefore that node must be the root of the sub tree below is this is the left sub tree, therefore that is the next value.

So, this entire red, block sub tree will be printed out and sorted order ending with x , after this the root will be printed thus which is this node which we have label as successor. So, therefore, that is the node that we want took a designate in the success for x .

(Refer Slide Time: 18:10)

Successor

```
function succ(t)
  if (t.right != NIL)
    return(minval(t.right))

  y = t.parent
  while (y != NIL and t == y.right)
    t = y
    y = y.parent
  return(y)
```

The diagram illustrates a binary search tree with nodes 1 through 9. The root is 5. Node 5 has left child 3 and right child 7. Node 3 has left child 1 and right child 4. Node 1 has right child 2. Node 7 has left child 9 and right child 8. A small inset diagram shows a node t as the right child of its parent y, with arrows indicating the movement of t and y up the tree during the successor search process.

So, here is a simple pseudo code for this, so we want to find the successor of node t. Now, if this node has a right pointer, then we just return the minimum value of the right sub tree, on the other hand if it does not have a right, then we need to go up and find the first place where the part turns right. So, we start with t and then we compute it is parent which we call y now. So, long as this direction is right, we keep moving up's we go to one more parent and we move t up's, so that is what happening in this loop here.

So, we move t and y up and then at some point we have move t and y will go this way. So, it will be y dot left and therefore, this now this y is the successor of this original node t and then there is one special case where we have reach the top and we do not ever turn right and that is case where we as you as started from the maximum value overall in the tree, in which case we reach in then it has no successive of we will just return nil.

(Refer Slide Time: 19:19)

Successor

```
function succ(t)
if (t.right != NIL)
    return(minval(t.right))

y = t.parent
while (y != NIL and t == y.right)
    t = y
    y = y.parent
return(y)
```

A binary search tree diagram with root 5. Node 5 has left child 3 and right child 7. Node 3 has left child 1 and right child 4. Node 1 has right child 2. Node 7 has right child 9. Node 9 has left child 8. A blue circle highlights the subtree rooted at 9 (nodes 9 and 8). A vertical red line is placed between the code and the tree diagram.

So, let us look at this particular thing, so for 3 for incidents the right sub tree exists 4 and this smaller is value there is 4. Therefore, there is a success of for 1 the smallest values 2, so that is the successor for 7 the minimum value in this sub tree is 8. So, 8 is a successor or 7, so these all come out of the basic case for you have a right sub tree.

(Refer Slide Time: 19:42)

Successor

```
function succ(t)
if (t.right != NIL)
    return(minval(t.right))

y = t.parent
while (y != NIL and t == y.right)
    t = y
    y = y.parent
return(y)
```

The same binary search tree diagram as in the previous slide. Blue arrows indicate the path for finding the successor of node 2. The path starts at node 2, goes up to its parent node 1, then up to its parent node 3, and finally up to its parent node 5. A vertical double yellow line is placed between the code and the tree diagram.

On the other hand, if you do not have a right sub tree then for instance you started 2, then you wake up and that they where you turn right, you find that 3 is the successive or from 4 you wake up and you very term right you find that 5 is successive. Similarly, for I 8 you immediately turn right, so 9 is successive and finally, for 9 you will come in and you will ((Refer time: 20:02)) this nil case. So, 9 will say that there is no successive, because

it is the large possible value.

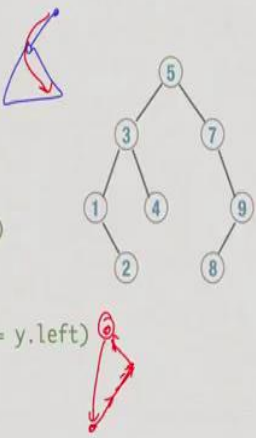
(Refer Slide Time: 20:09)

Predecessor

- Symmetric

```
function pred(t)
if (t.left != NIL)
    return(maxval(t.left))

y = t.parent
while (y != NIL and t == y.left)
    t = y
    y = y.parent
return(y)
```



So, the situation for the predecessor is symmetric, so if we have a left sub tree, then the predecessor is the maximum value in this. So, we go down left and then we go all the way right and if we do not have a left sub tree, then it means that this is already the minimum value in it is sub trees. So, we walk back up following these sequence of left pointers, until we turn the other way and then this node we have turn is predecessors.

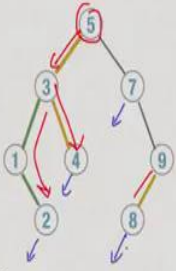
(Refer Slide Time: 20:41)

Predecessor

- Symmetric

```
function pred(t)
if (t.left != NIL)
    return(maxval(t.left))

y = t.parent
while (y != NIL and t == y.left)
    t = y
    y = y.parent
return(y)
```



So, for example if you look at these values which have left sub trees, so for 5 we go down the left sub tree and we find the right most value which is 4. So, 4 is the

predecessor are 5, likewise 2 is the predecessor of 3 and 8 is the predecessor of 9, now we have this other value. So, which we cannot to left for example, we cannot go left it 2, we cannot go left it 4, we cannot go left it 7, cannot go left it 8.

(Refer Slide Time: 21:09)

Predecessor

- Symmetric

```

function pred(t)
if (t.left != NIL)
    return(maxval(t.left))

y = t.parent

while (y != NIL and t == y.left)
    t = y
    y = y.parent

return(y)
    
```

So, what do we do in these cases we started 2 for example, and then we try to go up and where we turn right there we find. So, we come here and we find that one is a predecessor simulate from 4 if you go and so we a goal is to go right and then where we turn left if find predecessor. So, immediately turn left of these three nodes and then 8 we go right and where we turn left we find the predecessor. So, this is how the predecessor works it is exactly symmetric to the successive function.

(Refer Slide Time: 21:46)

Insert

- Try to find v
- If it is not present, add it where the search fails

Insert 21