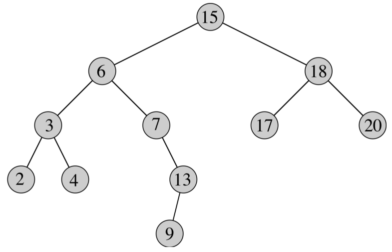


Binary Search Trees 3
Lecture 14

COSC 242 – Algorithms and Data Structures

Today’s outline

1. Assignment
2. Delete
3. Minimum and maximum
4. Successor and predecessor
5. Delete algorithm
6. Delete proof

Today’s outline

1. Assignment
2. Delete
3. Minimum and maximum
4. Successor and predecessor
5. Delete algorithm
6. Delete proof

Assignment

The assignment has been released. Details can be found on Blackboard, under Assessment (sidebar). The same information is also on the 242 department web page.

Due date: 2020-09-14, at 4pm.

Groups of 3

You can choose your own group, but if you don't tell Iain by end of day on Wednesday 19th, you will be assigned group members.

We will group people who have completed similar levels of internal assessment.

Today’s outline

1. Assignment
2. Delete
3. Minimum and maximum
4. Successor and predecessor
5. Delete algorithm
6. Delete proof

Delete

So far we’ve looked at insertion, searching, and traversal of a binary search tree.

Today we’re going to look at delete. This is a more complex operation that requires a few helper operations:

- Minimum and maximum
- Successor and predecessor

We’ll look at these helper operations first, then return to look at delete.

Why more complex?

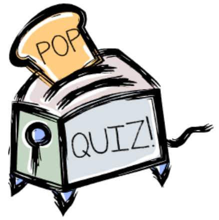
Think about the operations we’ve looked at so far.

How does delete differ from search and traversal? How does it differ from insert?

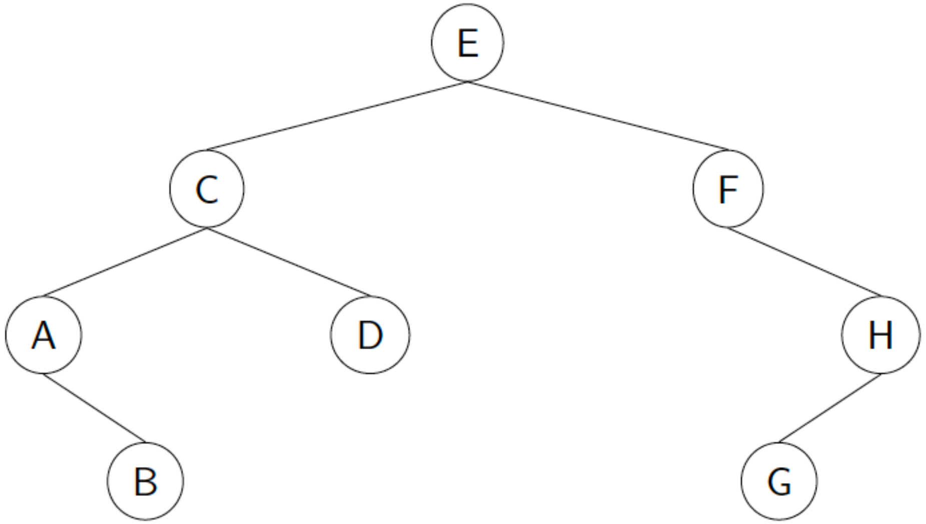
Today’s outline

1. Assignment
2. Delete
3. Minimum and maximum
4. Successor and predecessor
5. Delete algorithm
6. Delete proof

Pop quiz 1



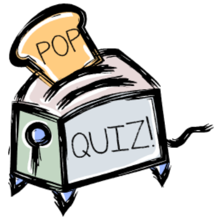
Where will you always find the minimum value in a BST?



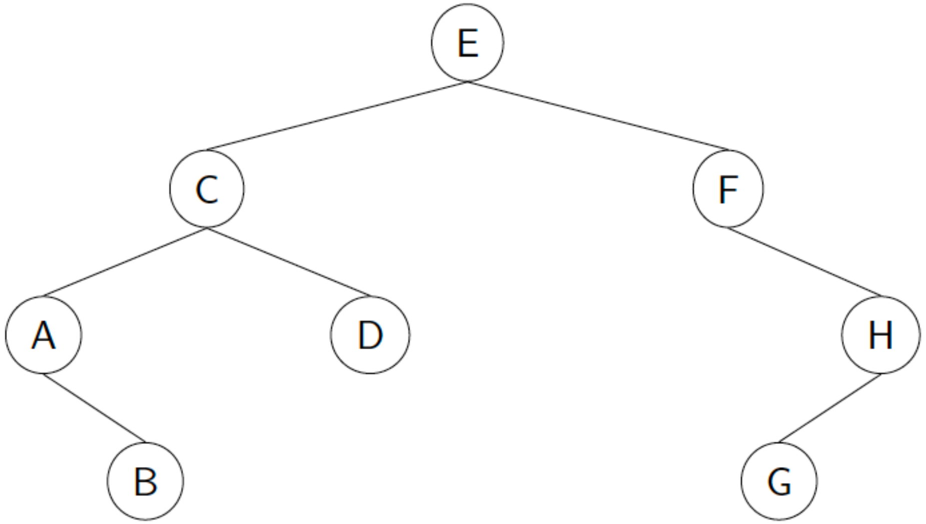
Minimum


```
function BST_find_min(T)
    if T == NIL then
        return Not Found
    else if T→left == NIL then
        return T→key
    else
        return BST_find_min(T→left)
    end if
end function
```

Pop quiz 2



Where will you always find the maximum value in a BST?



Class challenge 1

Sketch out the pseudocode to find the maximum element in a BST.

Observations on min, max

Both minimum and maximum run in O(h) time on a tree of height h.

As with search, the sequence of nodes encountered forms a simple path downward from the root.

Today’s outline

1. Assignment
2. Delete
3. Minimum and maximum
4. Successor and predecessor
5. Delete algorithm
6. Delete proof

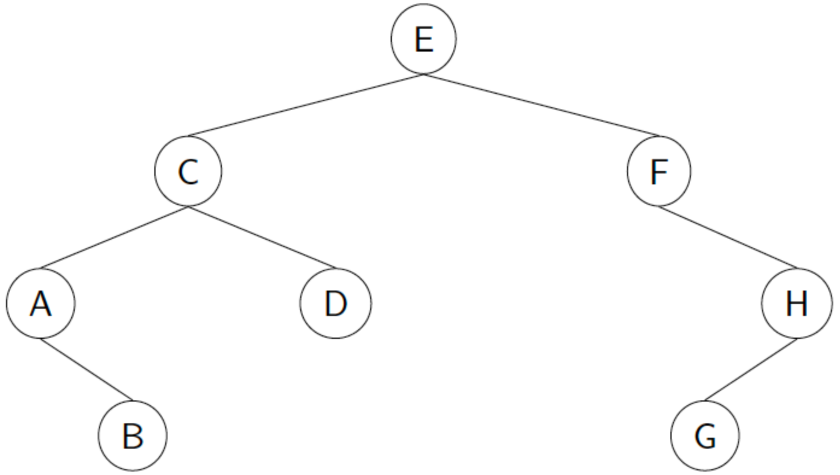
Successor

Given a node *T* in a binary tree, sometimes we need to find its successor in sorted order determined by an inorder traversal.

If all keys are distinct, the successor of node *T* is the node with the smallest key greater than T→key.

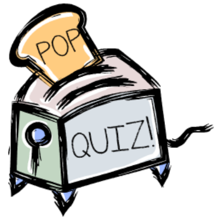
The structure of a binary tree allows us to determine the successor of a node without ever comparing keys.

Successor



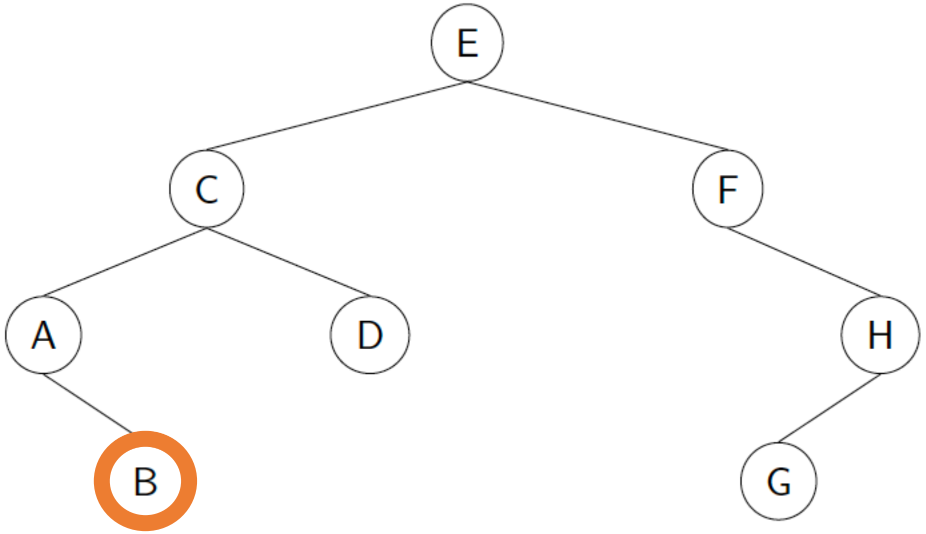

```
function BST_Successor(T)
    if T→right ≠ NIL then
        return BST_find_min(T→right)
    else
        parent = T→parent
        while parent ≠ NIL and T→key > parent→key do
            parent = parent→parent
        end while
        return parent
    end if
end function
```

Pop quiz 3

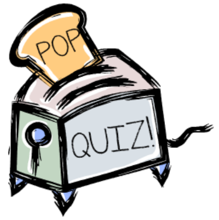


Who is B’s successor? What path does it follow to get there?

```
function BST_Successor(T)
    if T→right ≠ NIL then
        return BST_find_min(T→right)
    else
        parent = T→parent
        while parent ≠ NIL and T→key > parent→key do
            parent = parent→parent
        end while
        return parent
    end if
end function
```



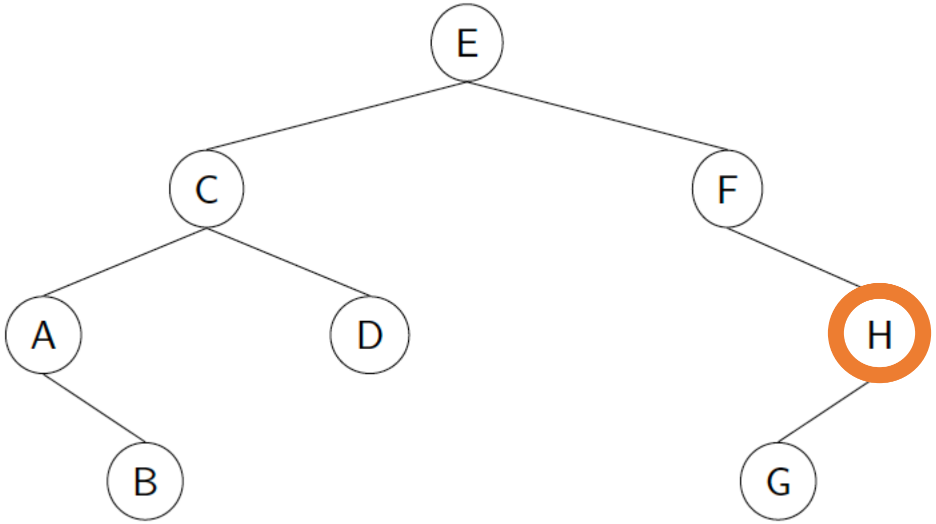
Pop quiz 4



Who is H's successor?

What path does it follow to get there?

```
function BST_Successor(T)
    if T→right ≠ NIL then
        return BST_find_min(T→right)
    else
        parent = T→parent
        while parent ≠ NIL and T→key > parent→key do
            parent = parent→parent
        end while
        return parent
    end if
end function
```



Predecessor

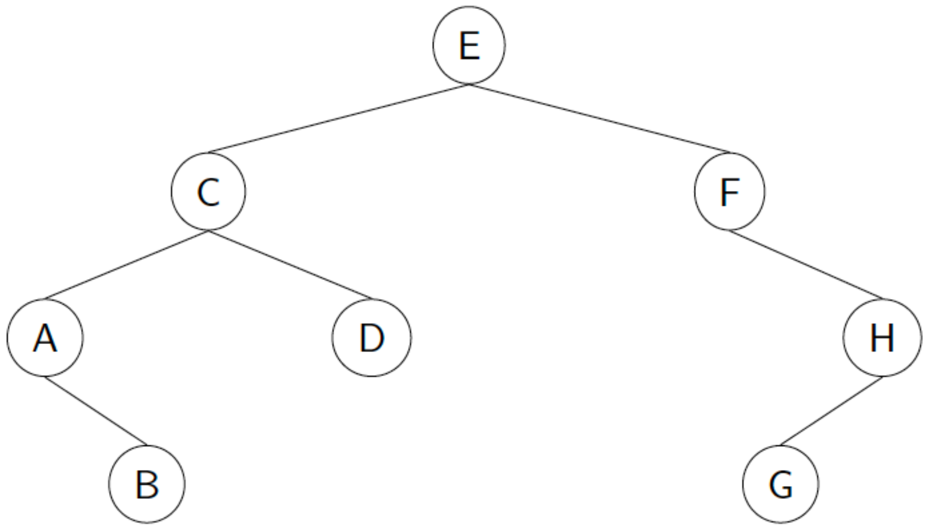
If all keys are distinct, the predecessor of node *T* is the node with the smallest key less than T→key.

Pseudocode for BST_Predecessor is left as a tutorial exercise.

Today’s outline

1. Assignment
2. Delete
3. Minimum and maximum
4. Successor and predecessor
5. Delete algorithm
6. Delete proof

Delete



Let's consider our example BST again.

What do we do if we want to:

1. delete a node with no children? E.g. D?
2. delete a node with one child? E.g. A?
3. delete a node with two children? E.g. C? or E?

Delete

In the labs, you will develop a delete based on key values (which basically incorporates a search into delete).

Here we are going to assume the node to delete is already found.

Delete


```
procedure BST_delete(T)
    if T→left == NIL and T→right == NIL then
        T→parent→[left or right] = NIL
        delete T
    else if T has one child then // splice out T
        T→parent→[left or right] = T→[left or right]
        delete T
    else if T has two children then
        BST_replace_with_successor(T)
    end if
end procedure
```

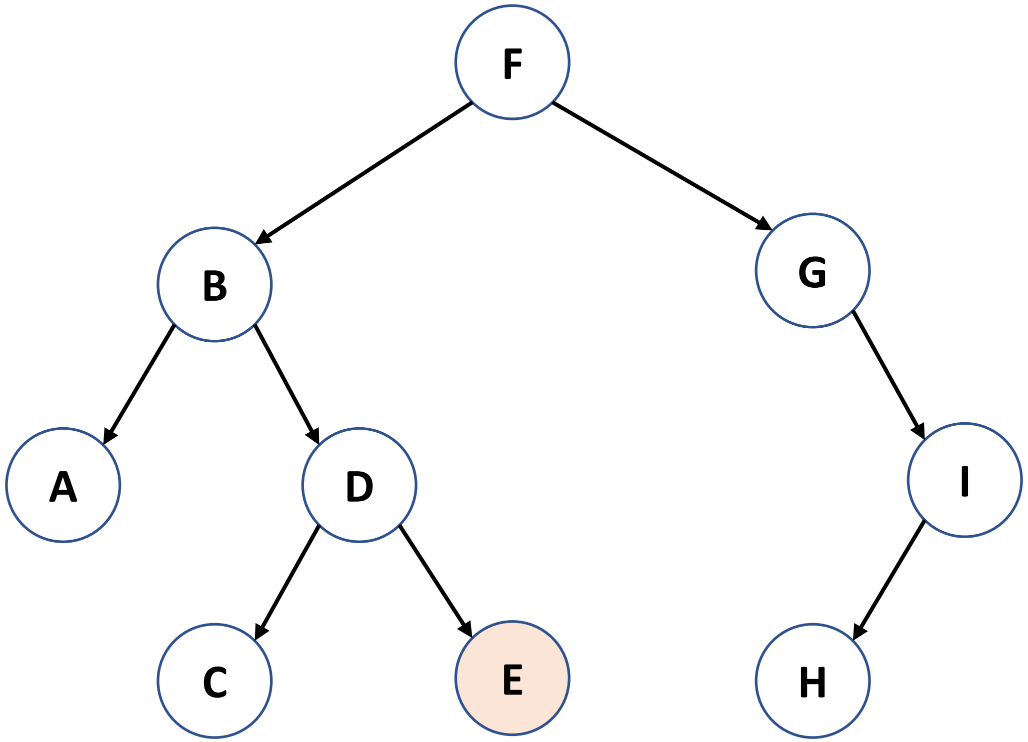
Splicing out

When T has two children, then we can replace T (or rather its contents) by its successor (or predecessor), and recursively delete the successor:

```
procedure BST_replace_with_succesor(T)
     successor = BST_Successor(T)
     successor_key = successor→key
     BST_delete(successor)
     T→key = successor_key
end procedure
```

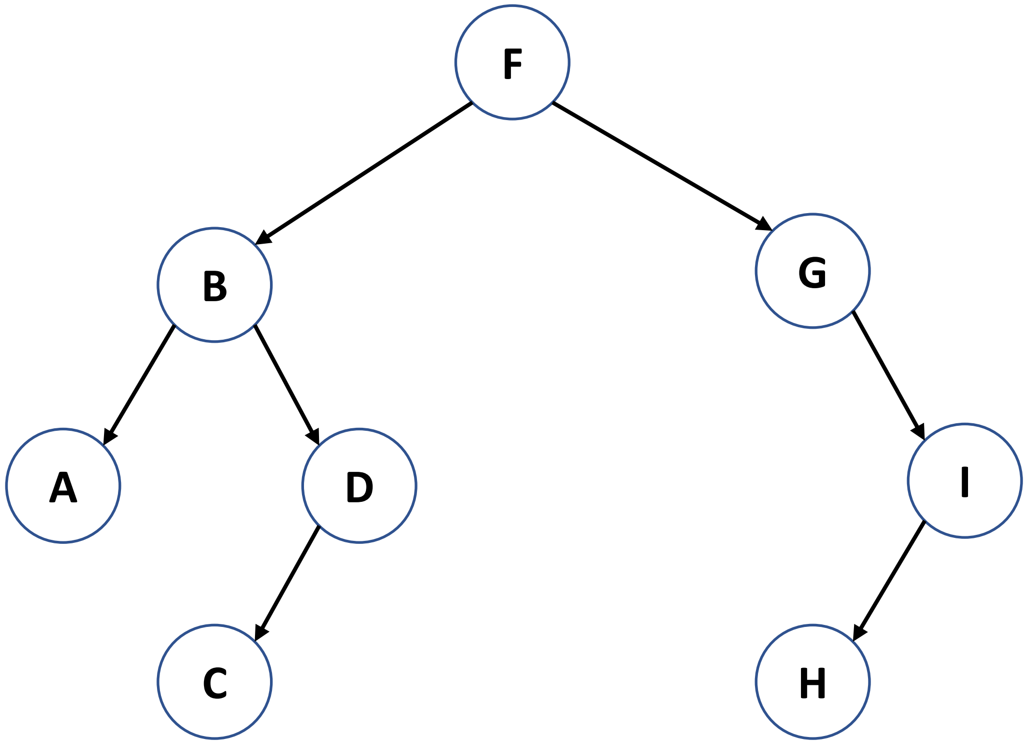
Delete ‘E’ (no children)

```
procedure BST_delete(T)
    if T→left == NIL and T→right == NIL then
        T→parent→[left or right] = NIL
        delete T
    else if T has one child then // splice out T
        T→parent→[left or right] = T→[left or right]
        delete T
    else if T has two children then
        BST_replace_with_successor(T)
    end if
end procedure
```



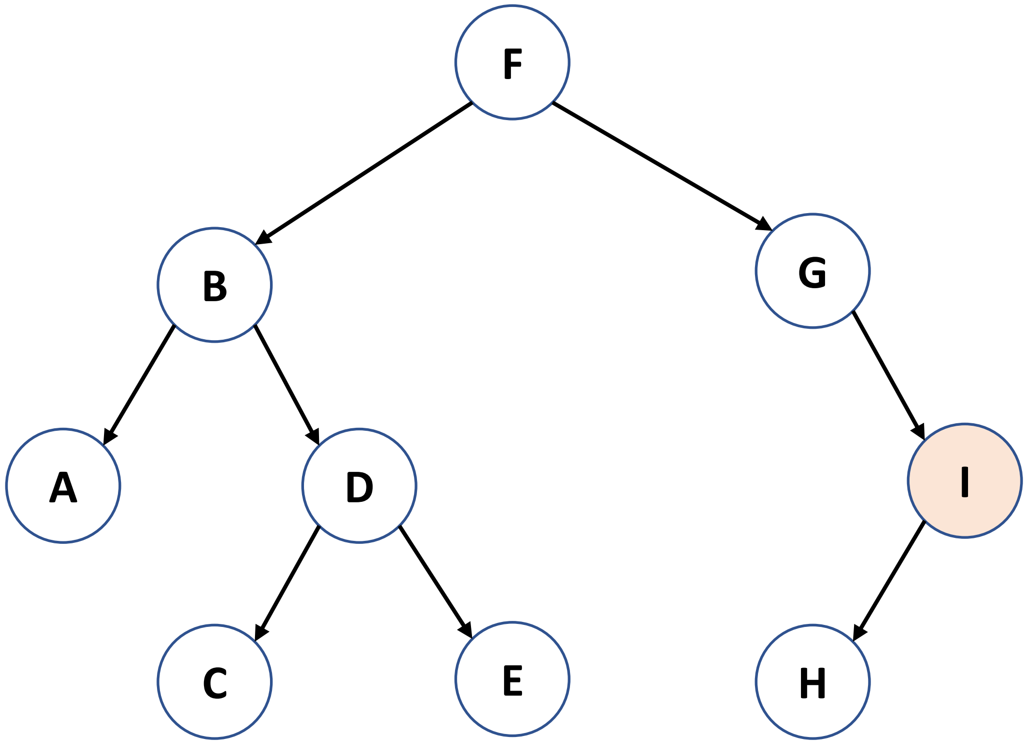
Delete ‘E’ (no children)

```
procedure BST_delete(T)
    if T→left == NIL and T→right == NIL then
        T→parent→[left or right] = NIL
        delete T
    else if T has one child then // splice out T
        T→parent→[left or right] = T→[left or right]
        delete T
    else if T has two children then
        BST_replace_with_successor(T)
    end if
end procedure
```



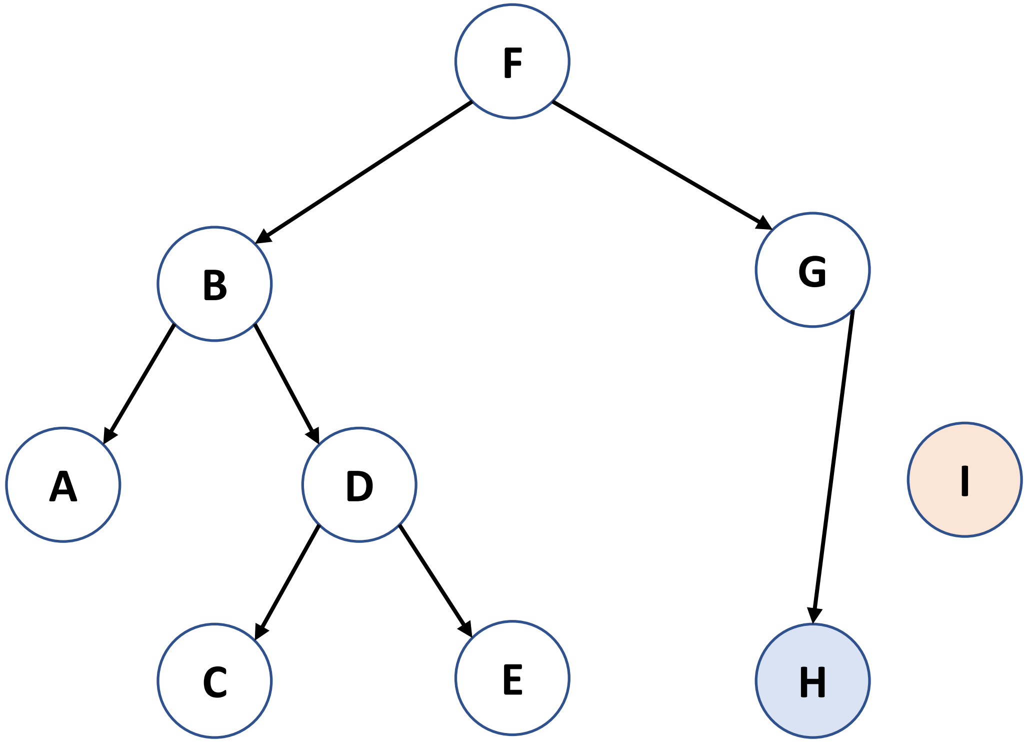
Delete ‘I’ (1 child)

```
procedure BST_delete(T)
    if T→left == NIL and T→right == NIL then
        T→parent→[left or right] = NIL
        delete T
    else if T has one child then // splice out T
        T→parent→[left or right] = T→[left or right]
        delete T
    else if T has two children then
        BST_replace_with_successor(T)
    end if
end procedure
```



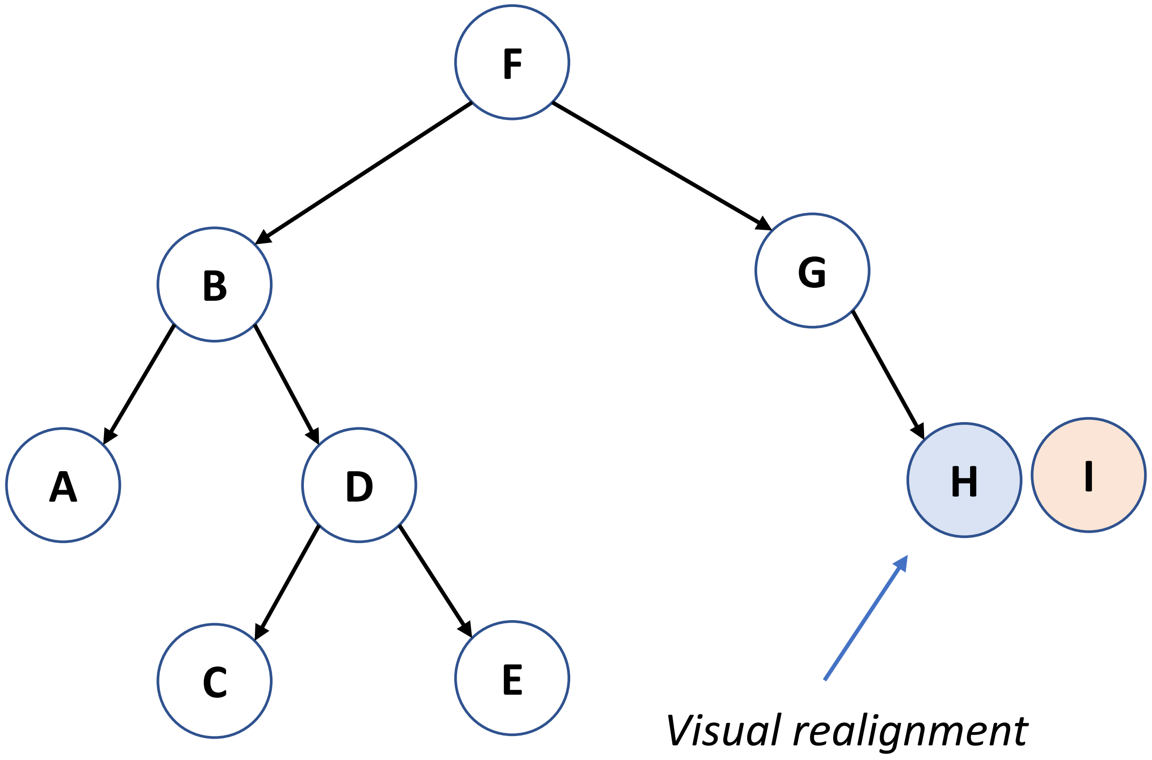
Delete ‘I’ (1 child)

```
procedure BST_delete(T)
    if T→left == NIL and T→right == NIL then
        T→parent→[left or right] = NIL
        delete T
    else if T has one child then // splice out T
        T→parent→[left or right] = T→[left or right]
        delete T
    else if T has two children then
        BST_replace_with_successor(T)
    end if
end procedure
```



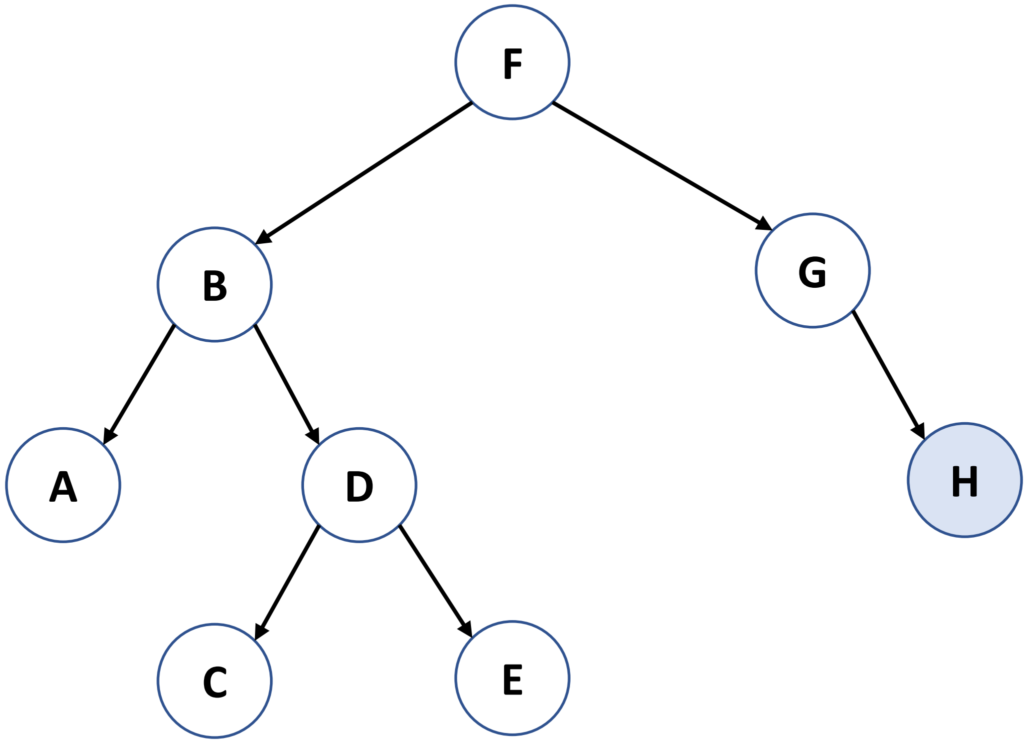
Delete ‘I’ (1 child)

```
procedure BST_delete(T)
    if T→left == NIL and T→right == NIL then
        T→parent→[left or right] = NIL
        delete T
    else if T has one child then // splice out T
        T→parent→[left or right] = T→[left or right]
        delete T
    else if T has two children then
        BST_replace_with_successor(T)
    end if
end procedure
```



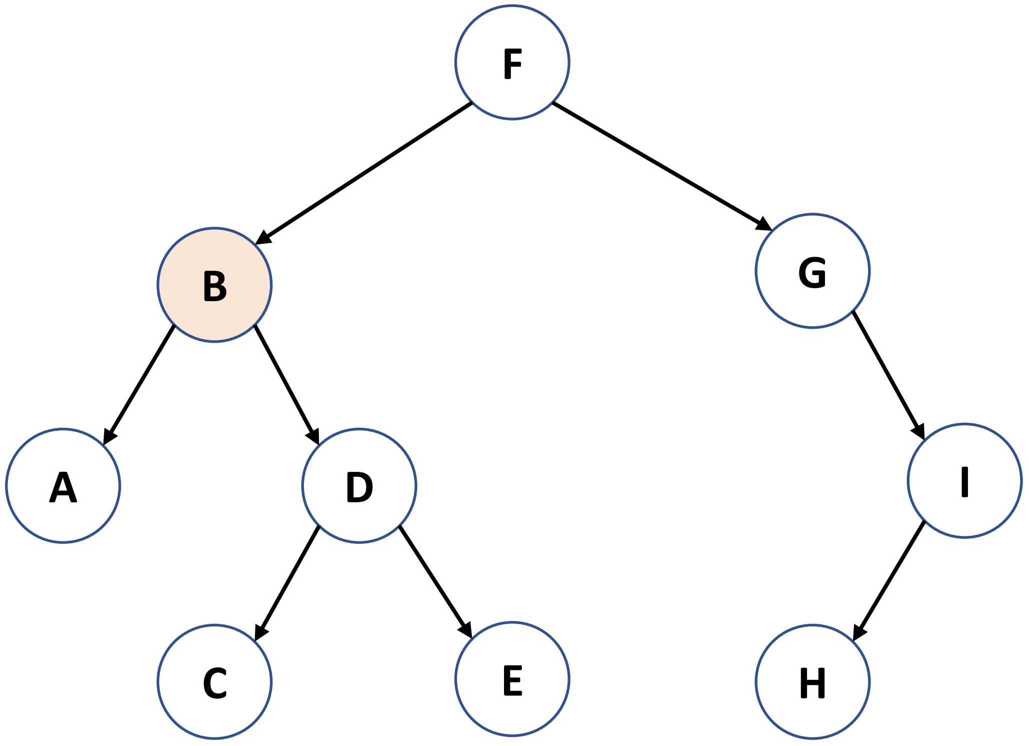
Delete ‘I’ (1 child)

```
procedure BST_delete(T)
    if T→left == NIL and T→right == NIL then
        T→parent→[left or right] = NIL
        delete T
    else if T has one child then // splice out T
        T→parent→[left or right] = T→[left or right]
        delete T
    else if T has two children then
        BST_replace_with_successor(T)
    end if
end procedure
```



Delete ‘**B**’ (2 children)

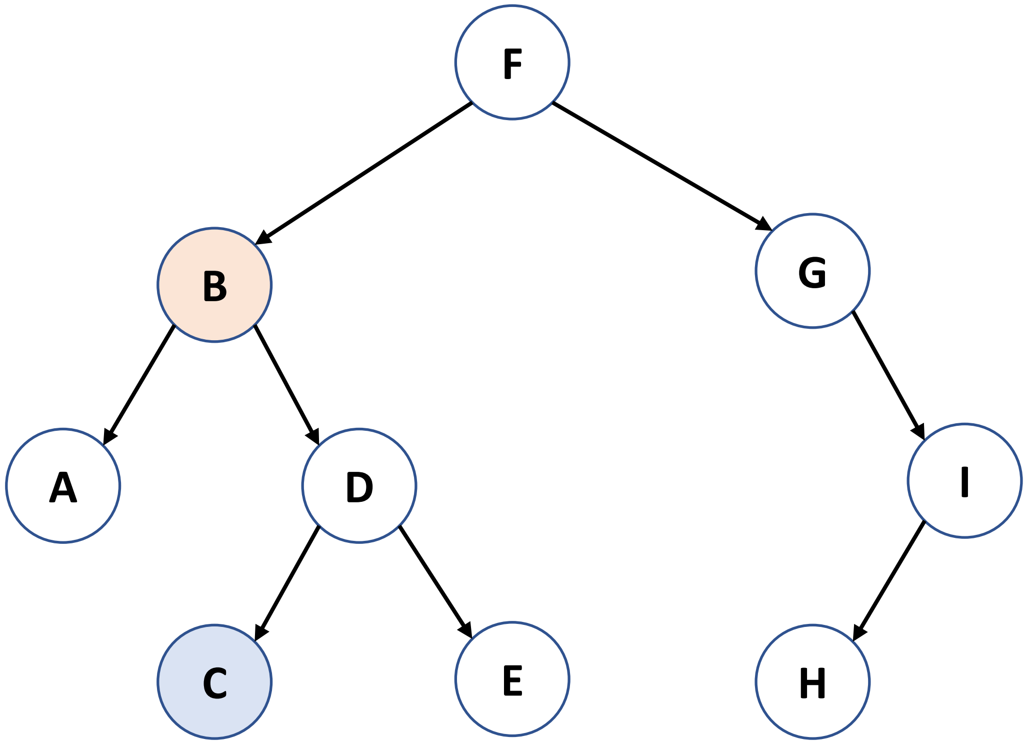
```
procedure BST_delete(T)
    if T→left == NIL and T→right == NIL then
        T→parent→[left or right] = NIL
        delete T
    else if T has one child then // splice out T
        T→parent→[left or right] = T→[left or right]
        delete T
    else if T has two children then
        BST_replace_with_successor(T)
    end if
end procedure
```




```
procedure BST_replace_with_succesor(T)
    successor = BST_Successor(T)
    successor_key = successor→key
    BST_delete(successor)
    T→key = successor_key
end procedure
```

Delete ‘B’ (2 children)

```
procedure BST_delete(T)
    if T→left == NIL and T→right == NIL then
        T→parent→[left or right] = NIL
        delete T
    else if T has one child then // splice out T
        T→parent→[left or right] = T→[left or right]
        delete T
    else if T has two children then
        BST_replace_with_successor(T)
    end if
end procedure
```

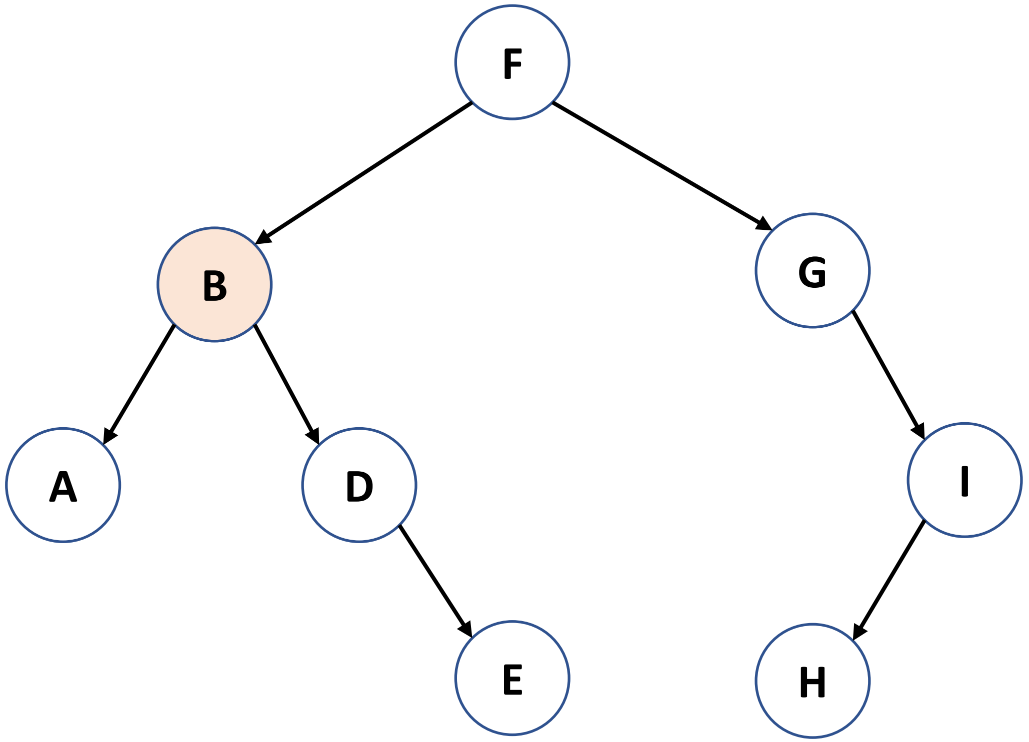



```
procedure BST_replace_with_succesor(T)
    successor = BST_Successor(T)
    successor_key = successor→key
    BST_delete(successor)
    T→key = successor_key
end procedure
```

Delete ‘**B**’ (2 children)

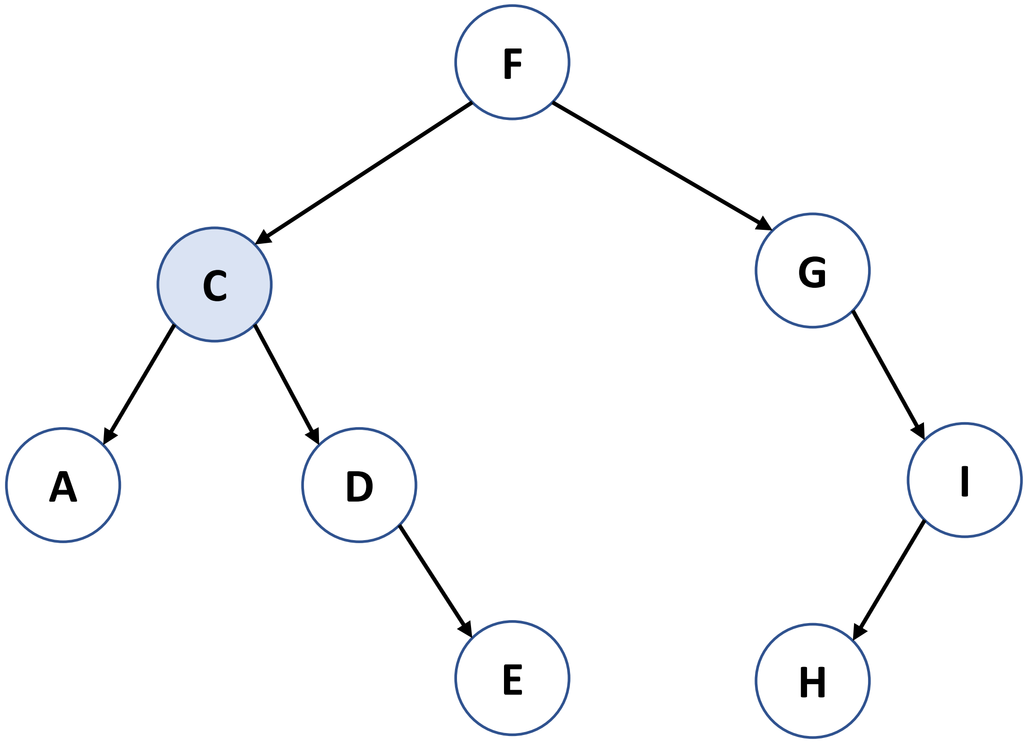
```
procedure BST_delete(T)
    if T→left == NIL and T→right == NIL then
        T→parent→[left or right] = NIL
        delete T
    else if T has one child then // splice out T
        T→parent→[left or right] = T→[left or right]
        delete T
    else if T has two children then
        BST_replace_with_successor(T)
    end if
end procedure
```

```
procedure BST_replace_with_succesor(T)
    successor = BST_Successor(T)
    successor_key = successor→key
    BST_delete(successor)
    T→key = successor_key
end procedure
```



Delete ‘B’ (2 children)

```
procedure BST_delete(T)
    if T→left == NIL and T→right == NIL then
        T→parent→[left or right] = NIL
        delete T
    else if T has one child then // splice out T
        T→parent→[left or right] = T→[left or right]
        delete T
    else if T has two children then
        BST_replace_with_successor(T)
    end if
end procedure
```




```
procedure BST_replace_with_succesor(T)
    successor = BST_Successor(T)
    successor_key = successor→key
    BST_delete(successor)
    T→key = successor_key
end procedure
```

Today’s outline

1. Assignment
2. Delete
3. Minimum and maximum
4. Successor and predecessor
5. Delete algorithm
6. Delete proof

Lemma

If T has two children, then T's successor must be a right descendent of T.

Proof

If T has two children then T's successor must be a descendant in the right subtree of T:

- If T is a right descendant of some node, A, then T is greater than A, and therefore A cannot be a successor;
- If T is a left descendant of some node, A, then T's right descendants lie between T and A and therefore A cannot be a successor;
- Therefore T's successor must be a descendant of T;
- All of T's left descendants are less than T, therefore T's successor must be a right descendant. ■

What's a lemma?

Lemma

If T has two children, then T’s successor has no left child.

Proof

- From the lemma in the previous slide, we know that T's successor is a right descendant.
- All of T's right descendants are greater than T.
- By definition, T's successor is the smallest value in T's right subtree.
- Let's assume that T's successor has a left child. If the successor has a left child, then that child's value is smaller than the successor, but since it is in T's right subtree, it must be greater than T. In which case we have found a node that is greater than T, but smaller than the successor, which is a contradiction.
- Therefore we conclude that T's successor has no left child. ■

Theorem

The algorithm BST_delete will always terminate.

Proof

- The only loop in BST_delete is via a call to BST_replace_with_successor which subsequently calls BST_delete with T's successor (call T's successor S).
- BST_replace_with_successor is only ever called when T has two children.
- S has at most one child by the lemma on the previous page.
- Therefore, when BST_delete is called with S, one of the first two branches of the if statement are taken and neither of them include a loop.
- Therefore BST_delete will always terminate. ■

Suggested reading

Iterative versions of minimum, maximum, and successor are given in section 12.2.

Delete is discussed in section 12.3, and although similar in spirit, is quite different to the delete algorithm discussed here. It is also different to the less complex deletion method used in earlier editions of the textbook (1^{st} and 2^{nd}).

Suggested reading

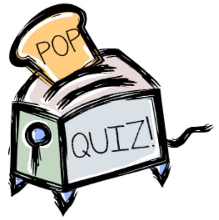
The two lemmas and theorems at the end of the lecture are not discussed in the textbook or anywhere that faculty are aware of.

The particular knowledge is not that important, but the idea of proving something about an algorithm is.

It is also important to note that not all proofs contain equations!

Solutions

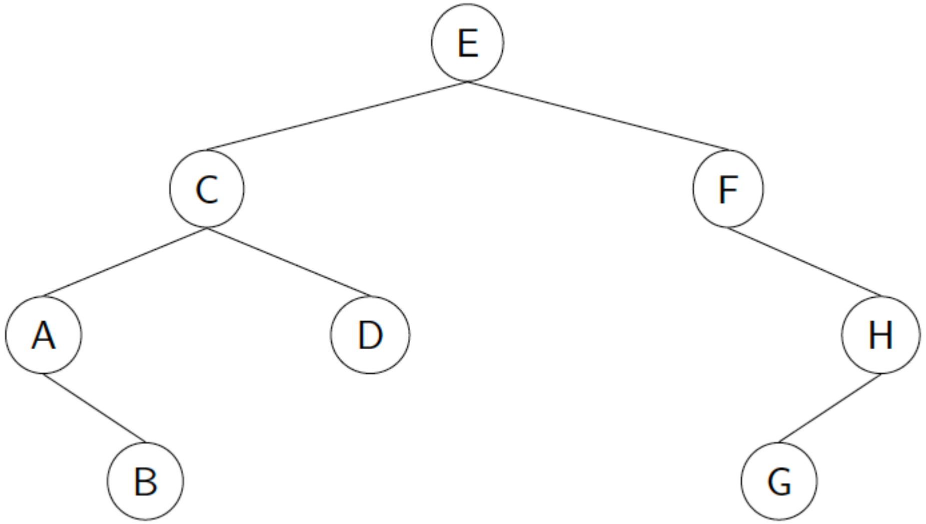
Pop quiz 1



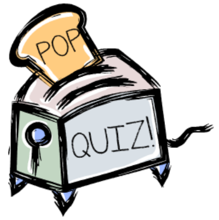
Where will you always find the minimum value in a BST?

Answer

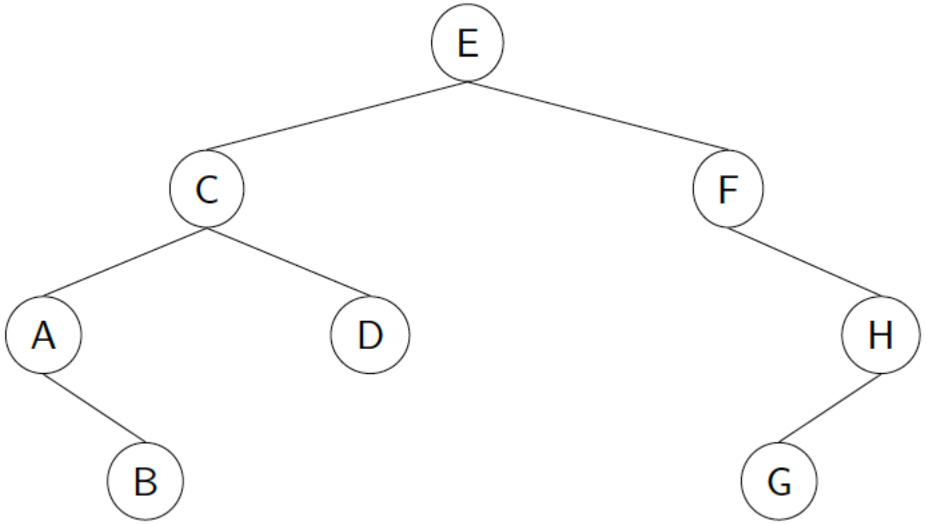
The left-most node.



Pop quiz 2



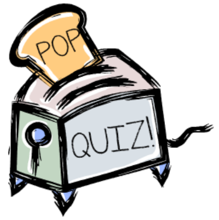
Where will you always find the maximum value in a BST?



Answer

The right-most node.

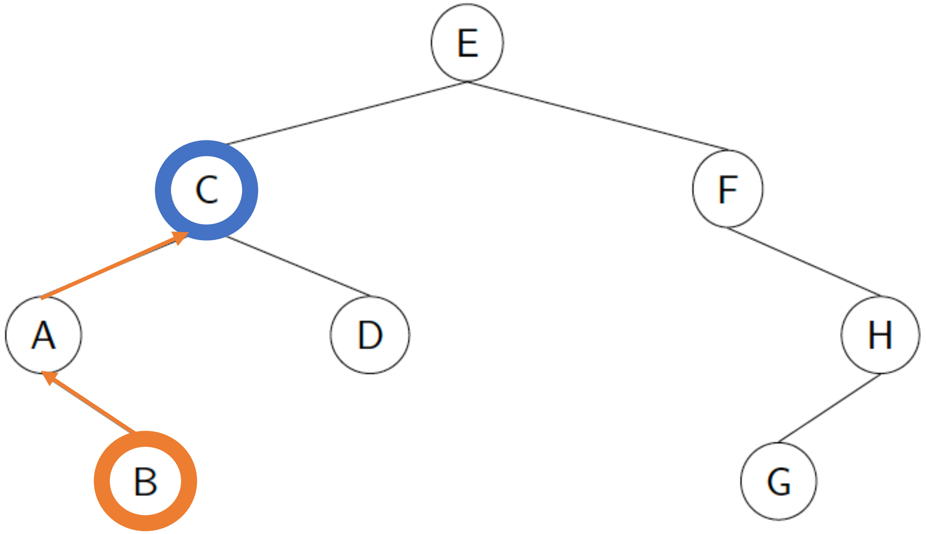
Pop quiz 3



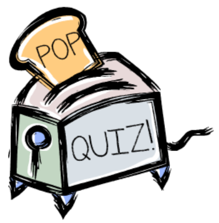
Who is B's successor? What path does it follow to get there?

Answer

C



Pop quiz 4

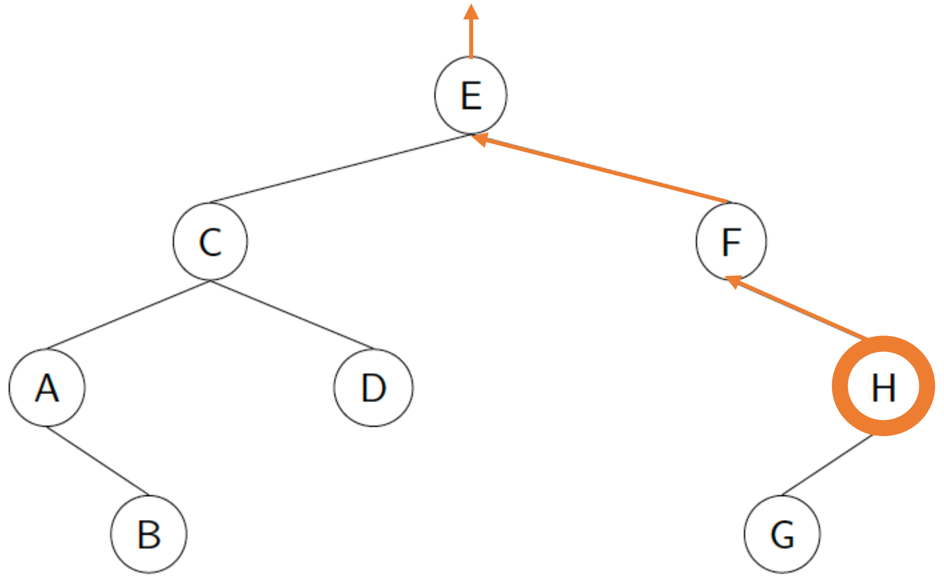


Who is H’s successor?
What path does it follow to get there?

Answer

NIL

```
        else
            parent = T→parent
            while parent ≠ NIL and T→key > parent→key do
                parent = parent→parent
            end while
            return parent
```



Class challenge 1

```
function BST_find_max(T)
    if T == NIL then
        return Not Found
    else if T→right == NIL then
        return T→key
    else
        return BST_find_max(T→right)
    end if
end function
```

Image attributions

[This Photo](#) by Unknown Author is licensed under [CC BY](#)

Disclaimer: Images and attribution text provided by PowerPoint search. The author has no connection with, nor endorses, the attributed parties and/or websites listed above.