Luiss
Libera Università Internazionale degli Studi Sociali Guido Carli

Algorithms A.Y. 2022/2023

Lab – Binary Search Trees

Irene Finocchi, Flavio Giorgi, Bardh Prenkaj
finocchi@luiss.it, fgiorgi@luiss.it, bprenkaj@luiss.it©

17 February 2023

courtesy of: *Andrea Coletta*

LUISS

Dipartimento di Impresa e Management

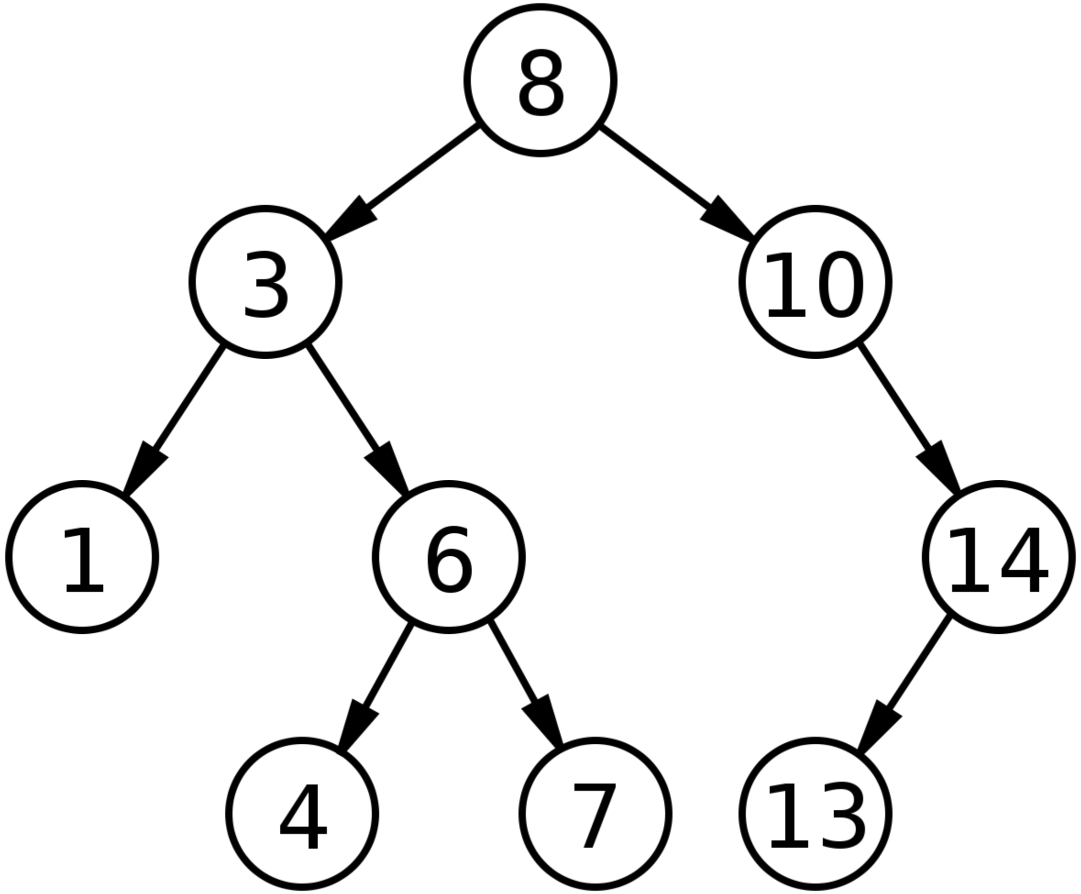
Binary Search Tree

A ***binary search tree*** is a binary tree that satisfies three properties:

- Each node v is associated with a key $key(v)$
- All the keys in the **left** subtree of v are **smaller** than $key(v)$
- All the keys in the **right** subtree of v are **grater** than $key(v)$

Binary Search Tree

A *binary search tree* example



Binary Search Tree

We can perform many operations on binary search trees (BST):

- Search
- Insertion
- Deletion
- …

Binary Search Tree

We can perform many operations on binary search trees (BST):

- Search **DONE**
- Insertion **DONE**
- Deletion **TODO**
- …

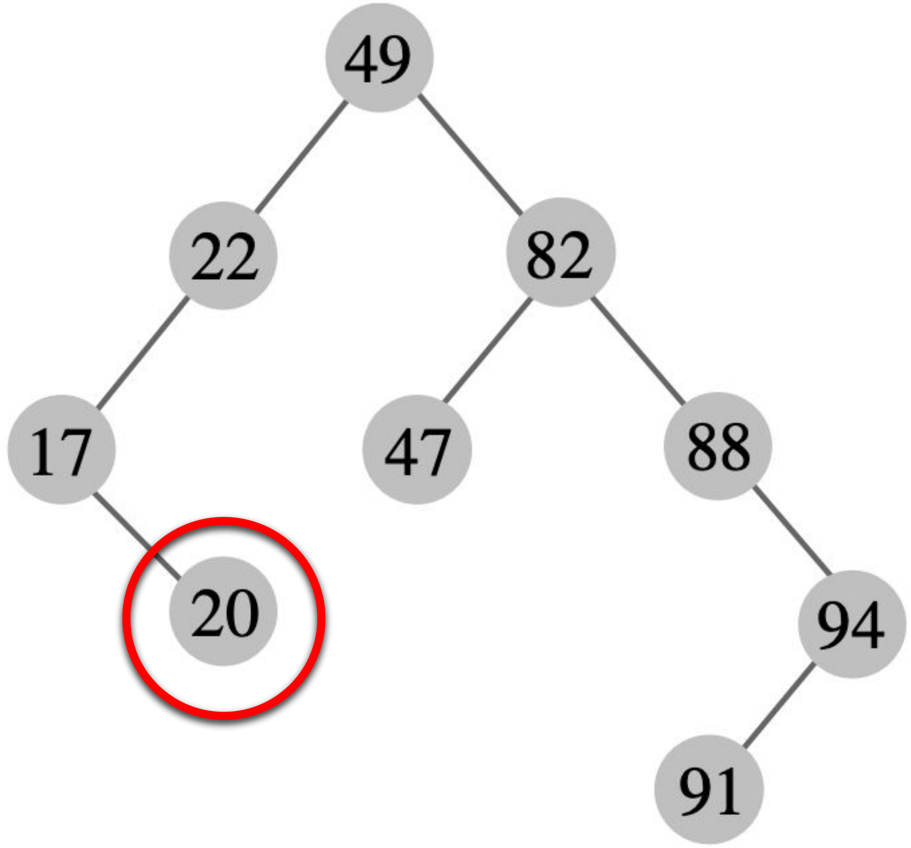
Binary Search Tree - Element Deletion

To delete an element from a BST we have 3 different cases:

Binary Search Tree - Element Deletion

To delete an element from a BST we have 3 different cases:

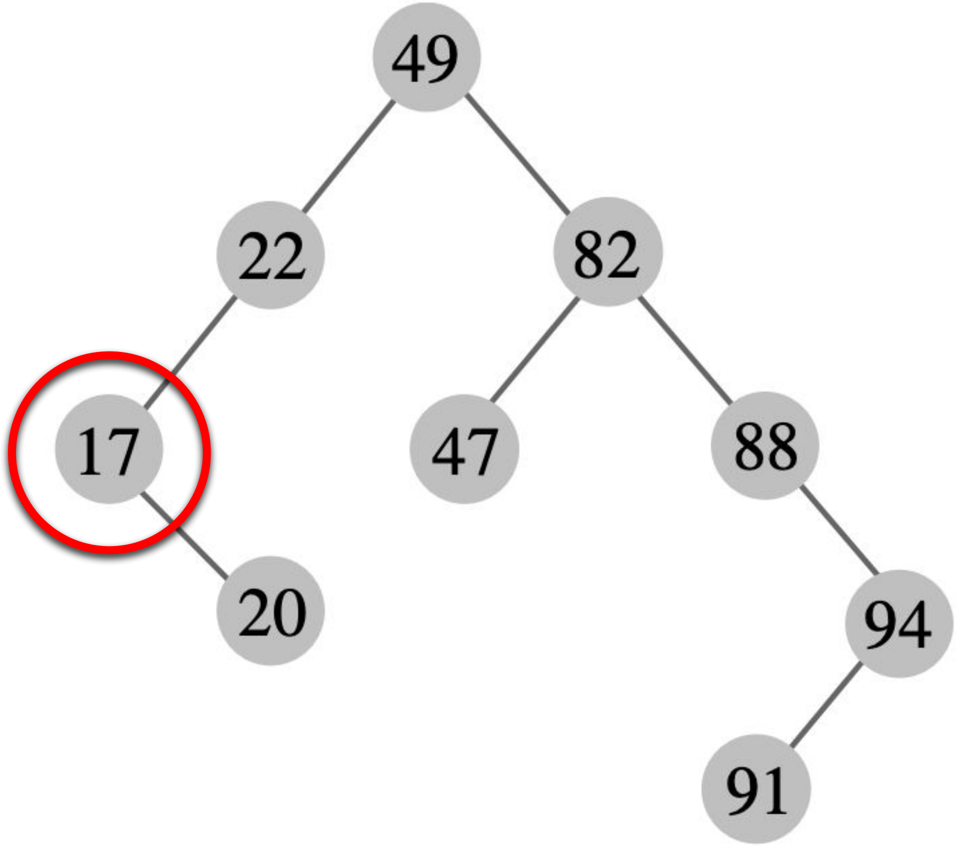
1. The node to delete is a leaf



Binary Search Tree - Element Deletion

To delete an element from a BST we have 3 different cases:

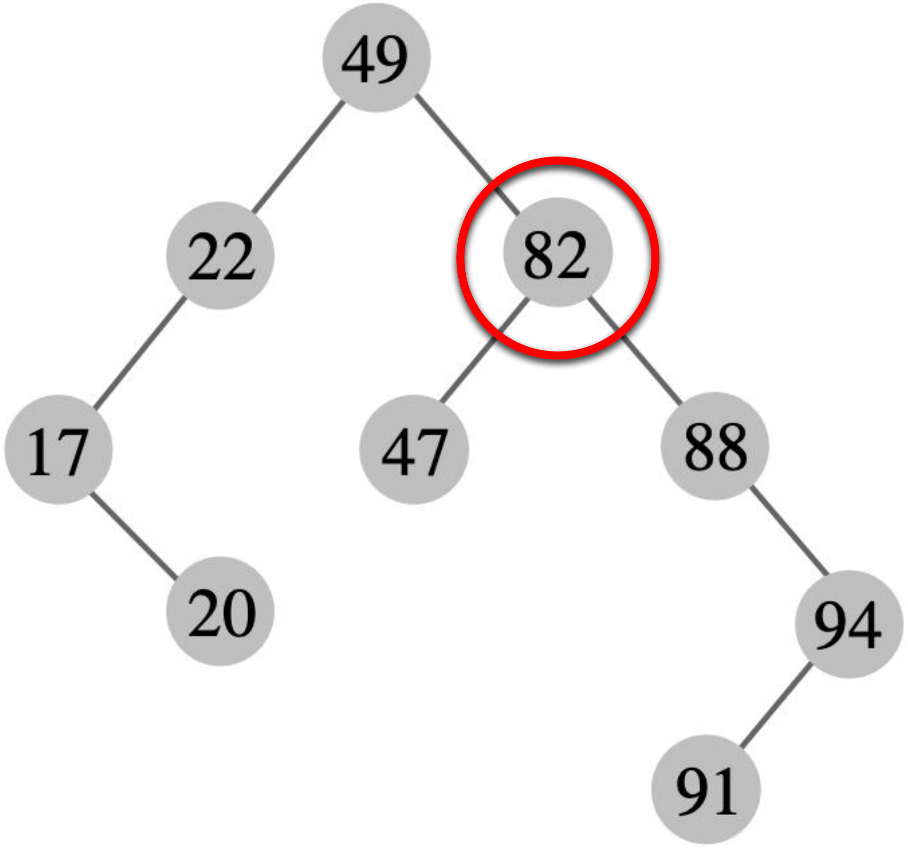
1. The node to delete is a leaf
2. The node to delete has just 1 child



Binary Search Tree - Element Deletion

To delete an element from a BST we have 3 different cases:

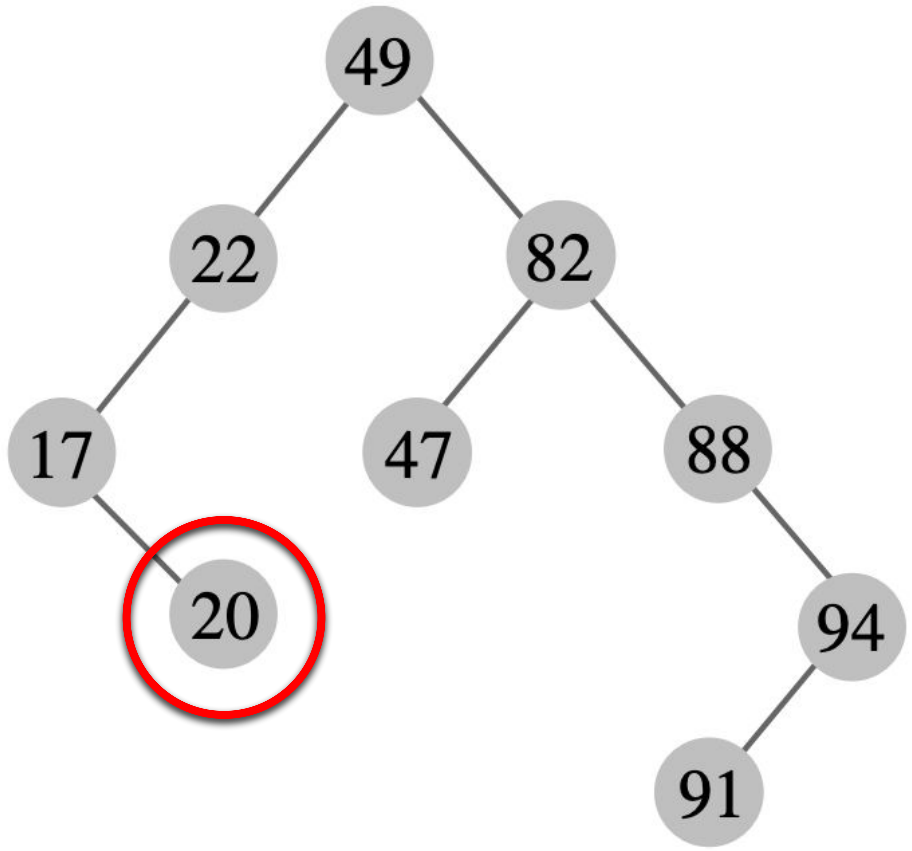
1. The node to delete is a leaf
2. The node to delete has just 1 child
3. The node to delete has 2 children



Binary Search Tree - Element Deletion

Case 1: The node to delete is a leaf

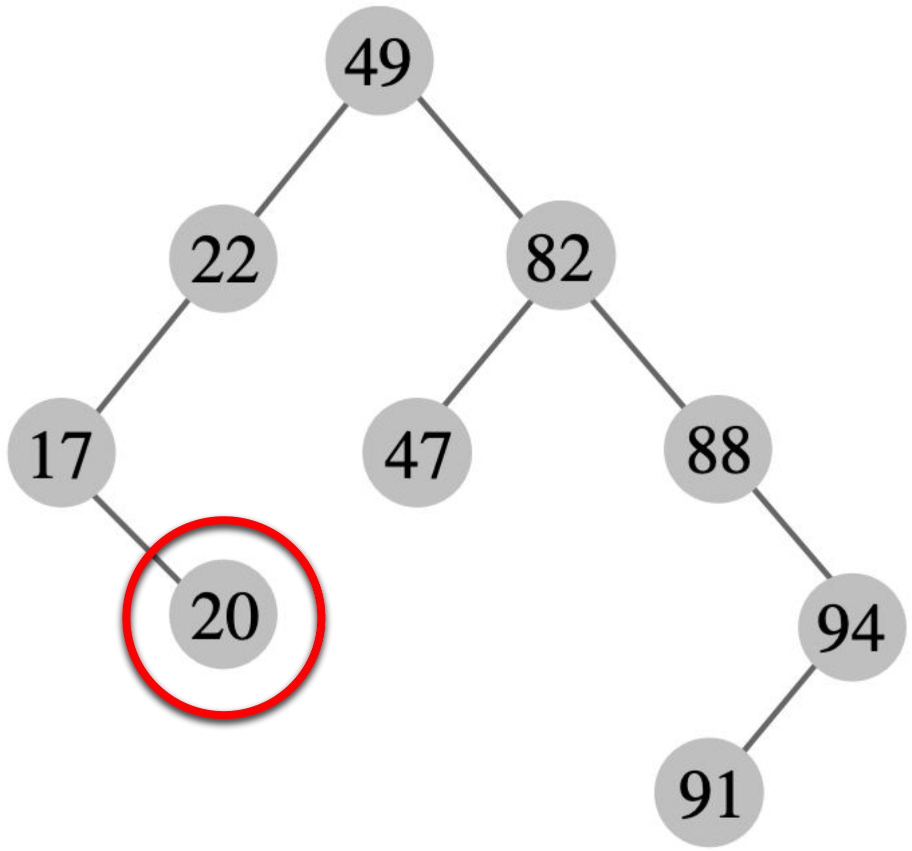
Any guess?



Binary Search Tree - Element Deletion

Case 1: The node to delete is a leaf

We can just delete the node! **Super Easy**

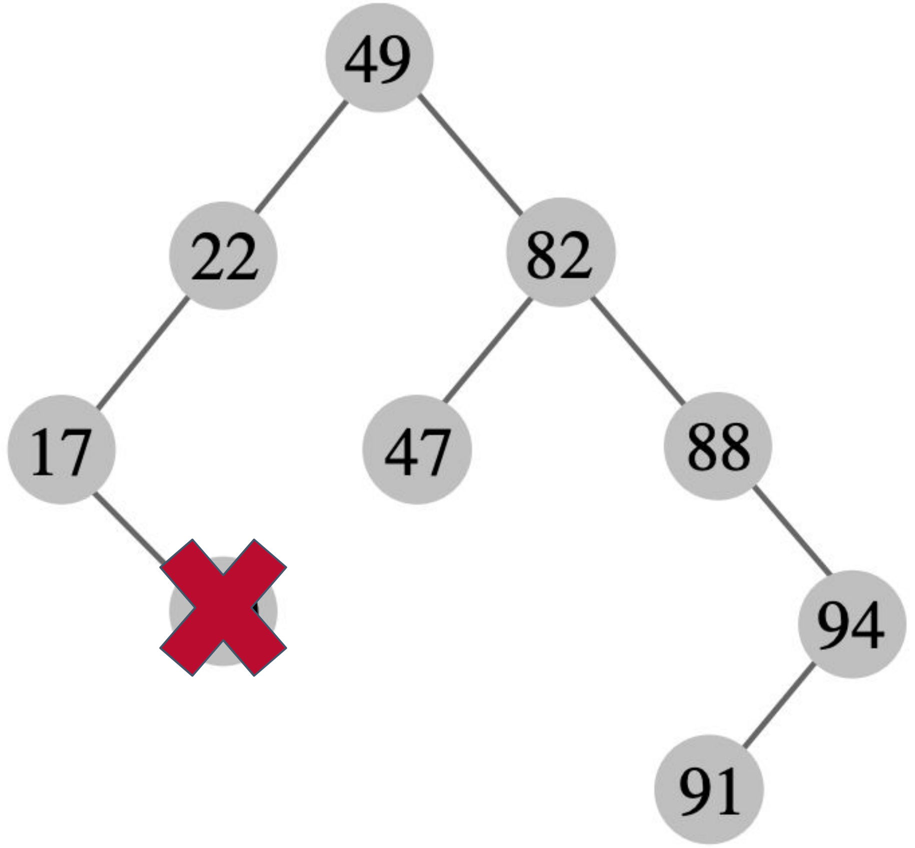


LUISS

Binary Search Tree - Element Deletion

Case 1: The node to delete is a leaf

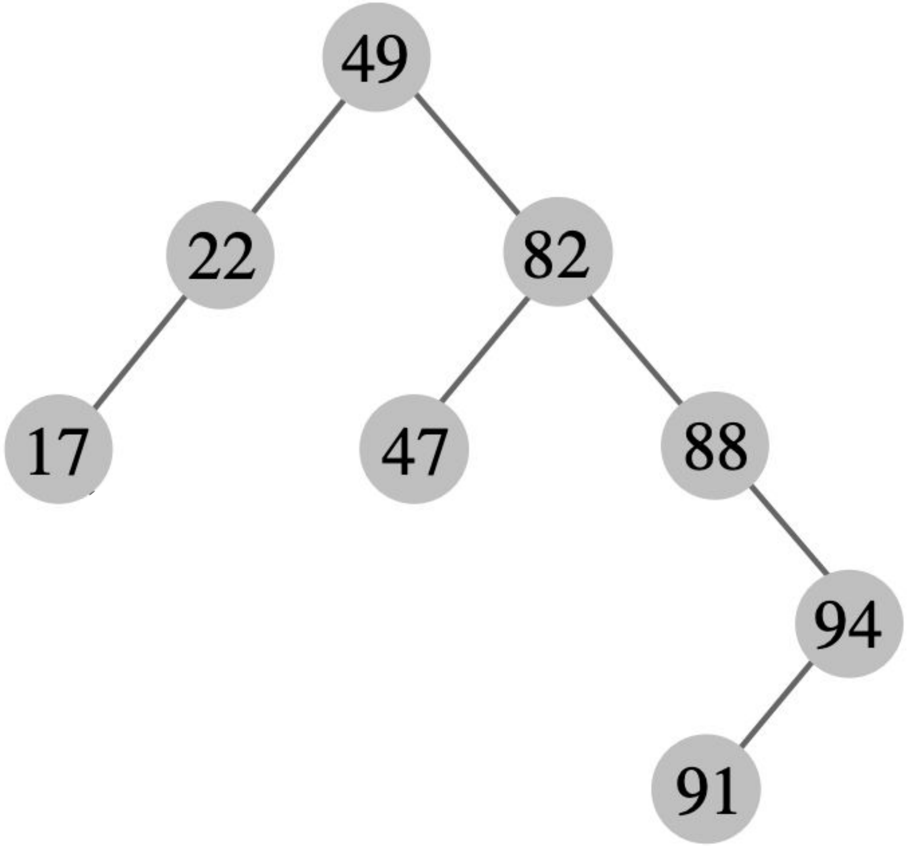
We can just delete the node! **Super Easy**



Binary Search Tree - Element Deletion

Case 1: The node to delete is a leaf

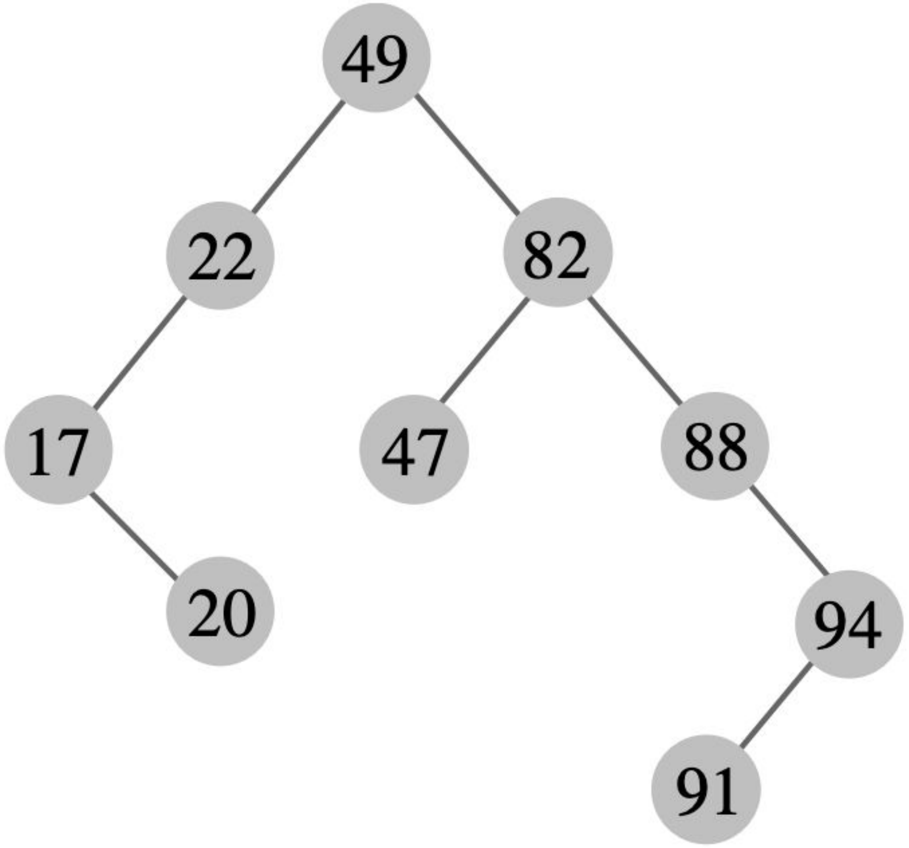
We can just delete the node! **Super Easy**



Binary Search Tree - Element Deletion

Case 2: The node to delete has just one child

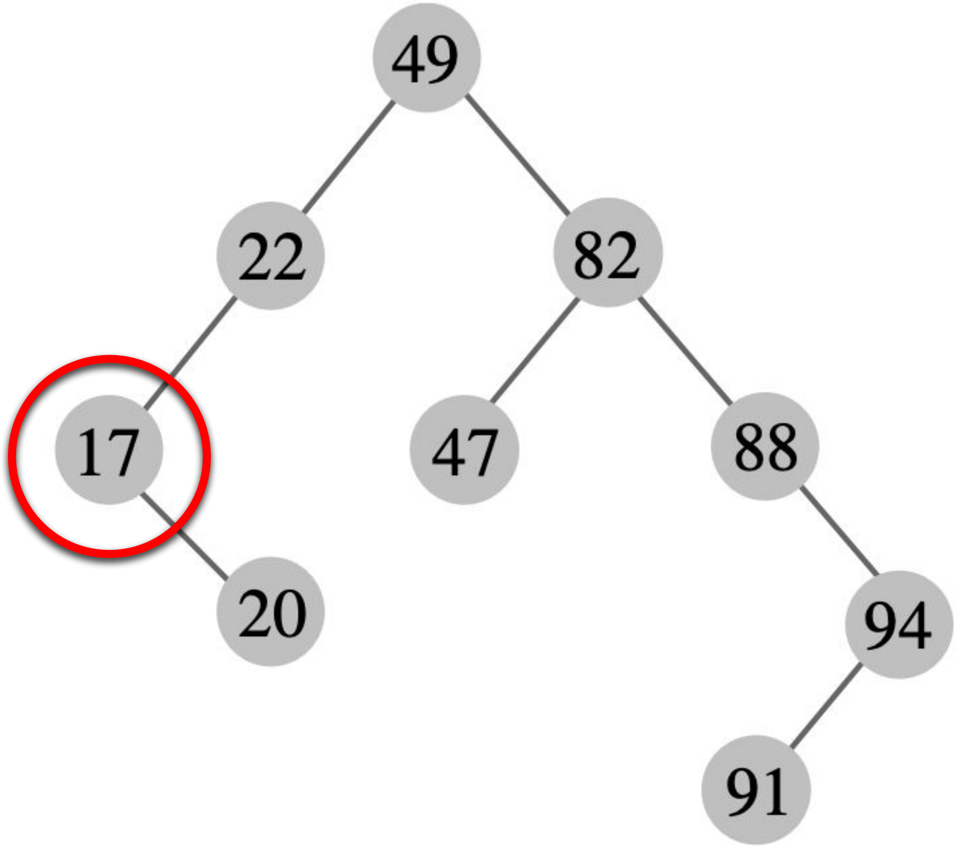
Any guess?



Binary Search Tree - Element Deletion

Case 2: The node to delete has just one child

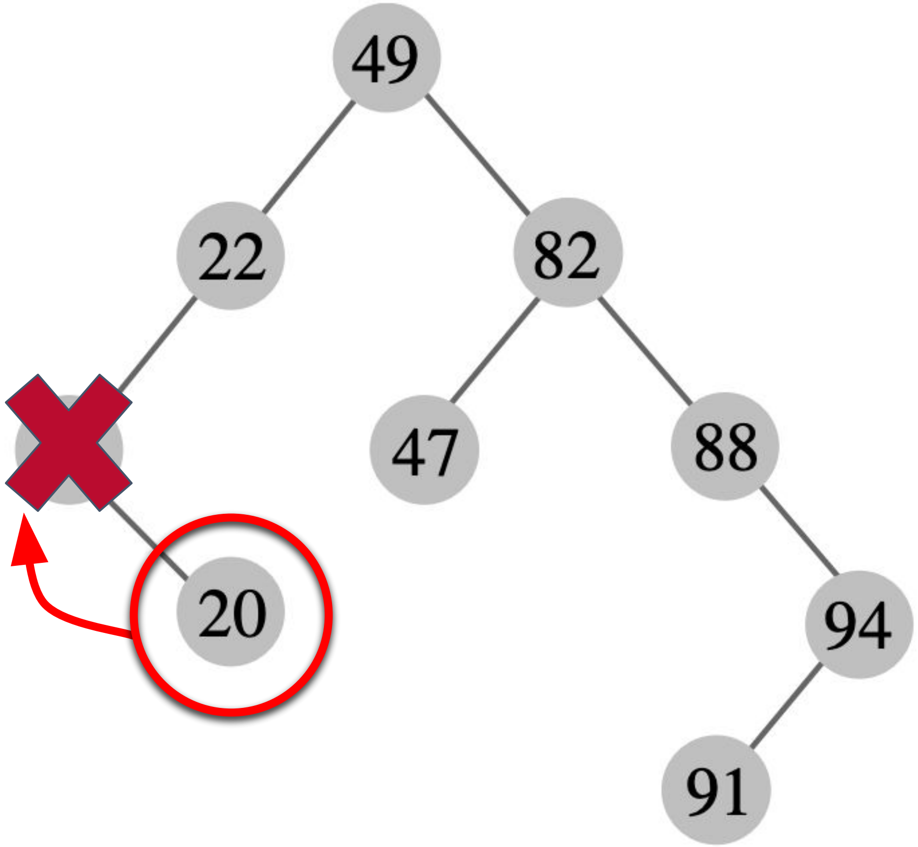
We can delete the node and put the child in the same place of the parent



Binary Search Tree - Element Deletion

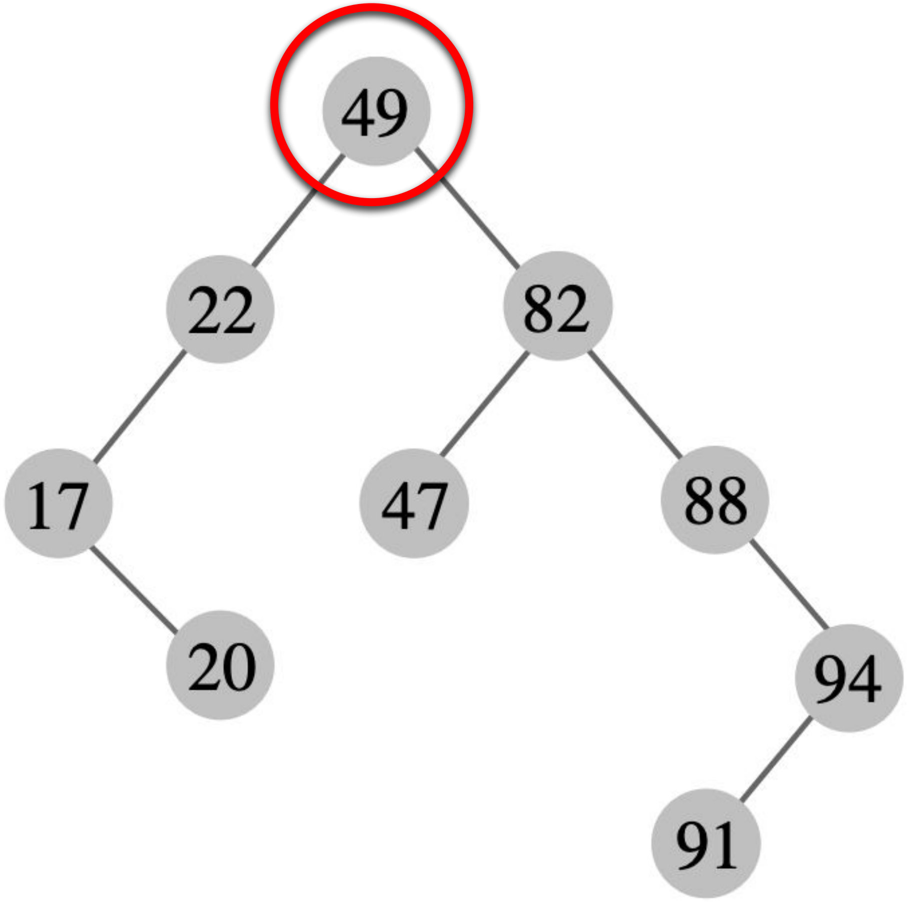
Case 2: The node to delete has just one child

We can delete the node and put the child in the same place of the parent **Easy!**



Binary Search Tree - Element Deletion

Case 3: The node to delete has two children.



Binary Search Tree - Element Deletion

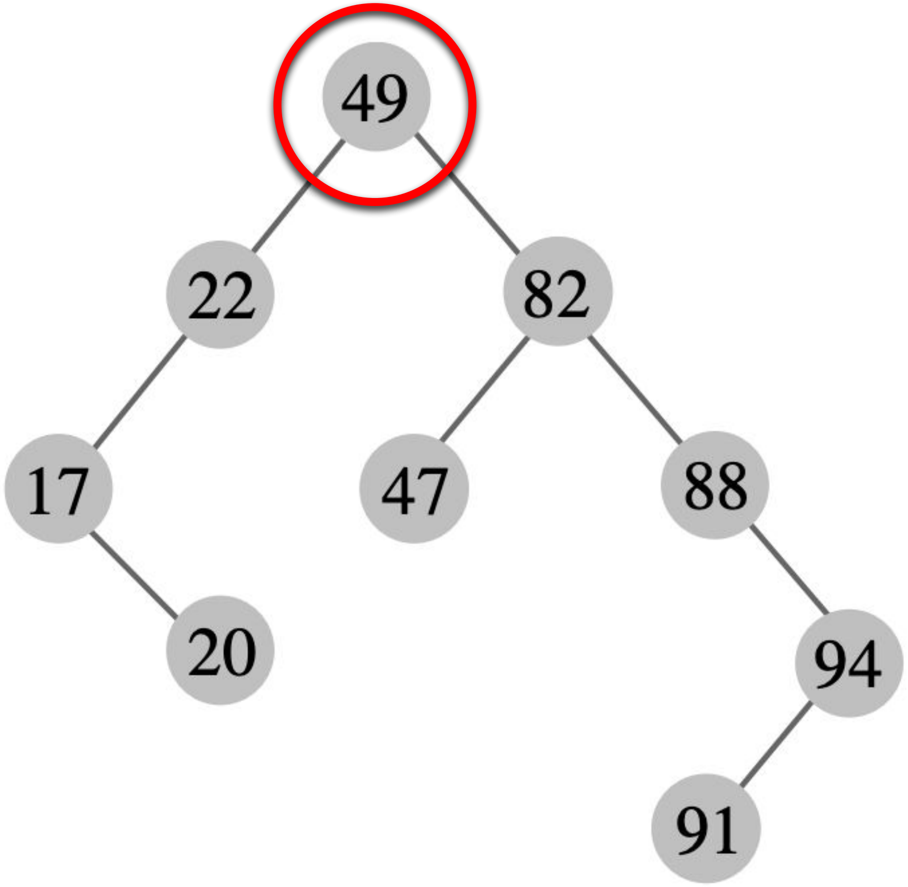
Case 3: The node to delete has two children.

It is slightly more complex compare to others.

The node to delete is replaced with its **in-order successor** (or predecessor).

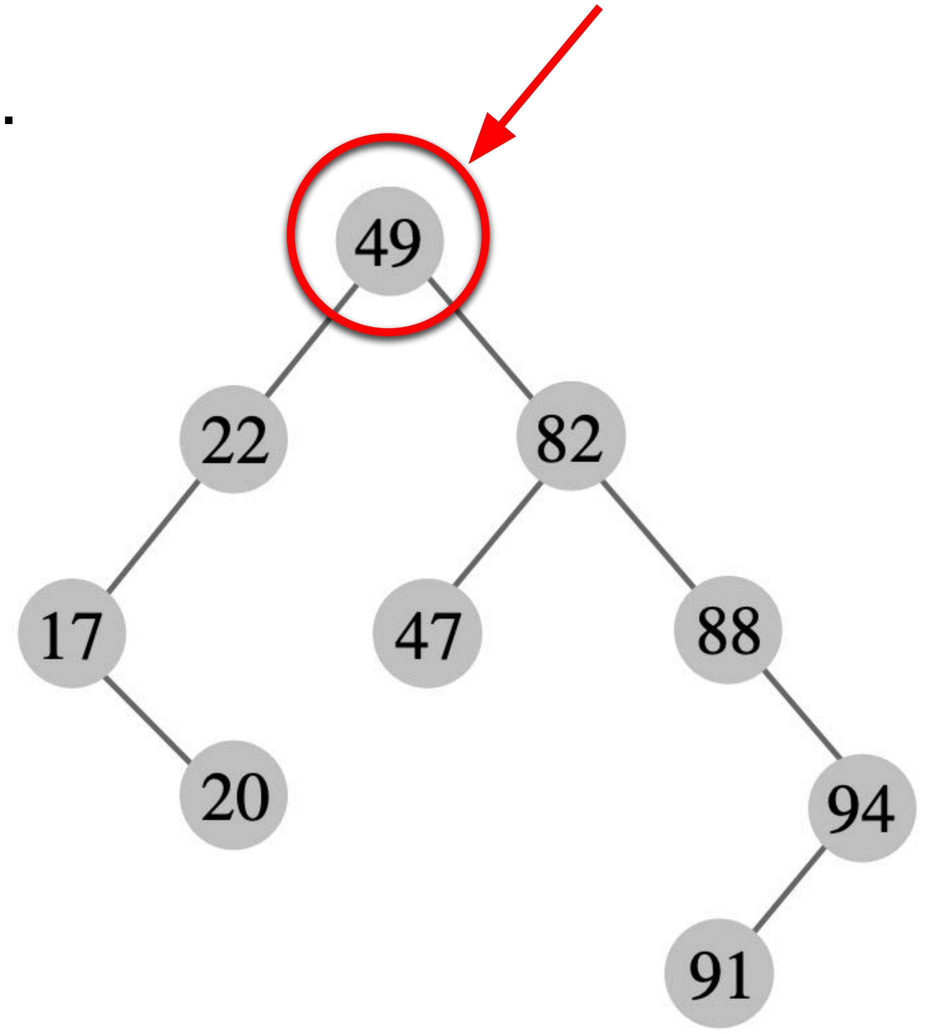
Binary Search Tree - Element Deletion

Case 3: The node to delete has two children.



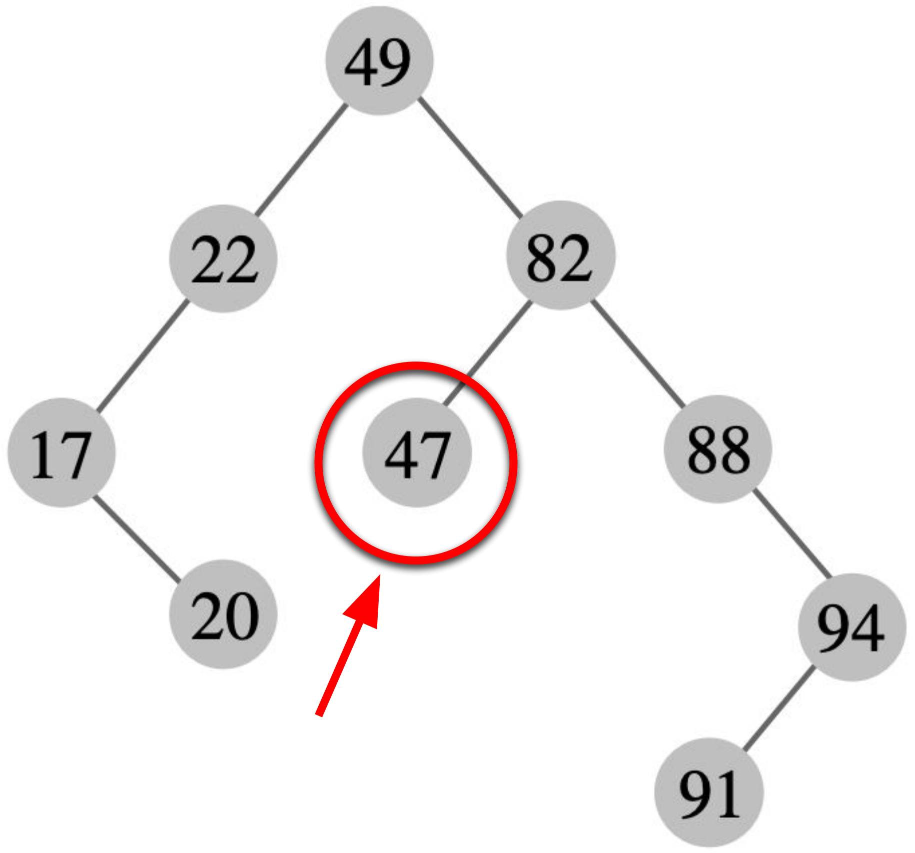
Binary Search Tree - Element Deletion

Case 3: The node to delete has two children.



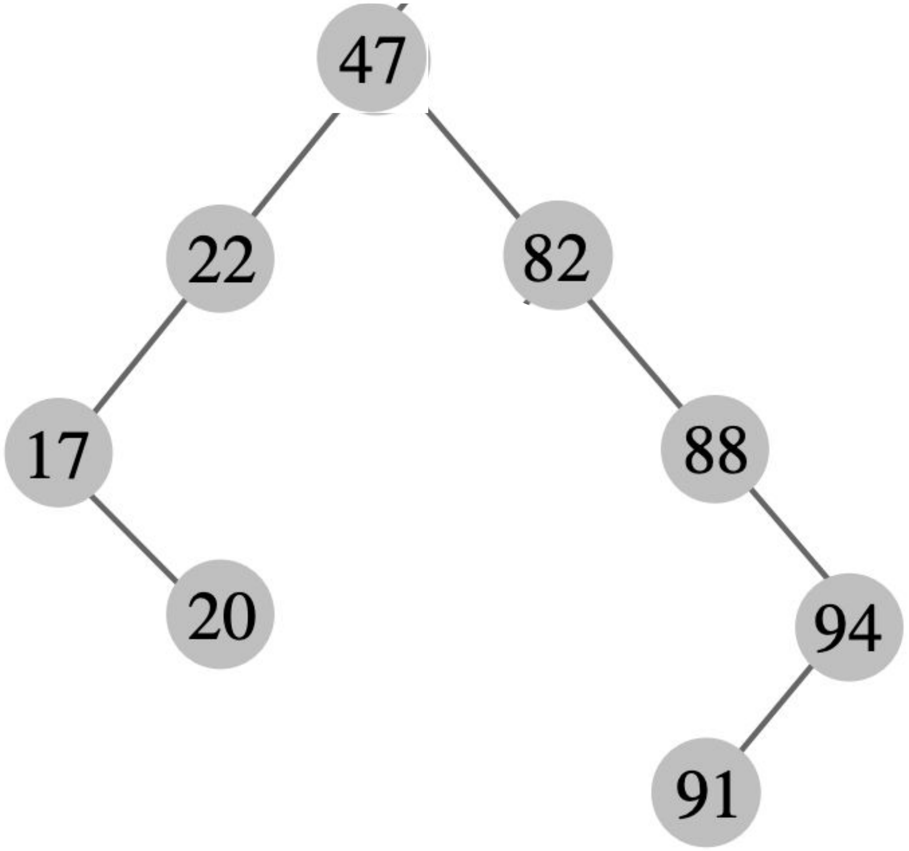
Binary Search Tree - Element Deletion

Case 3: The node to delete has two children.



Binary Search Tree - Element Deletion

Case 3: The node to delete has two children.



Binary Search Tree - What they are used for?

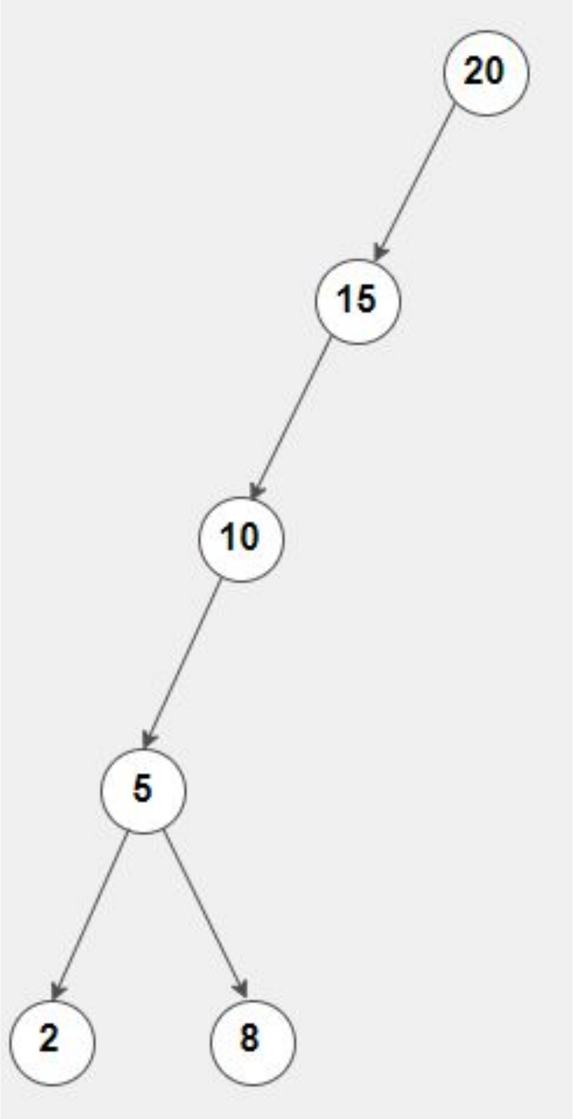
- BSTs are used for indexing

Binary Search Tree - What they are used for?

- BSTs are used for indexing

Be aware that a BST can become **unbalanced**

What is the problem?

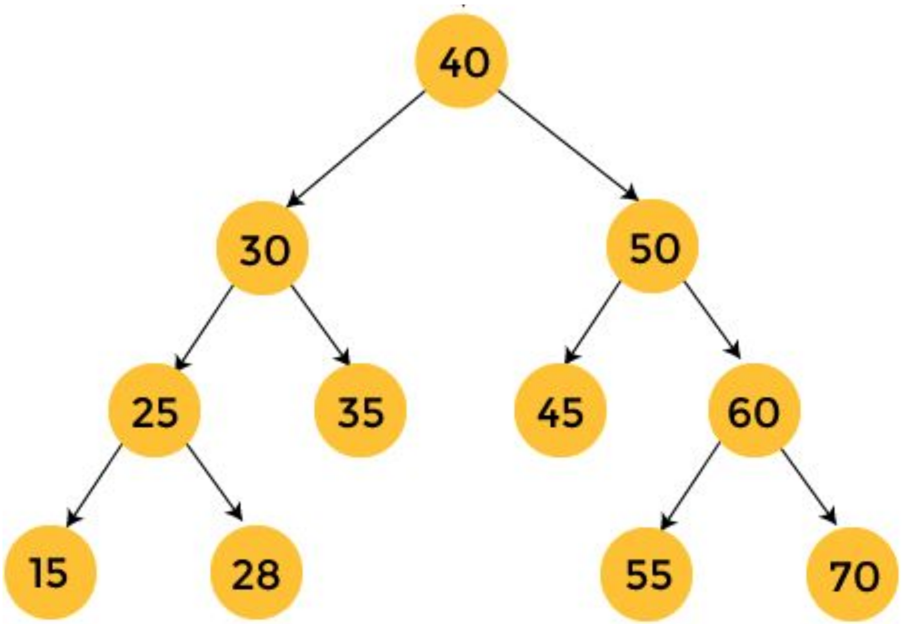


Binary Search Tree - What they are used for?

- BSTs are used for indexing
- TreeMap and TreeSet data structures in java are internally implemented using self-balancing BSTs to avoid unbalanced cases

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



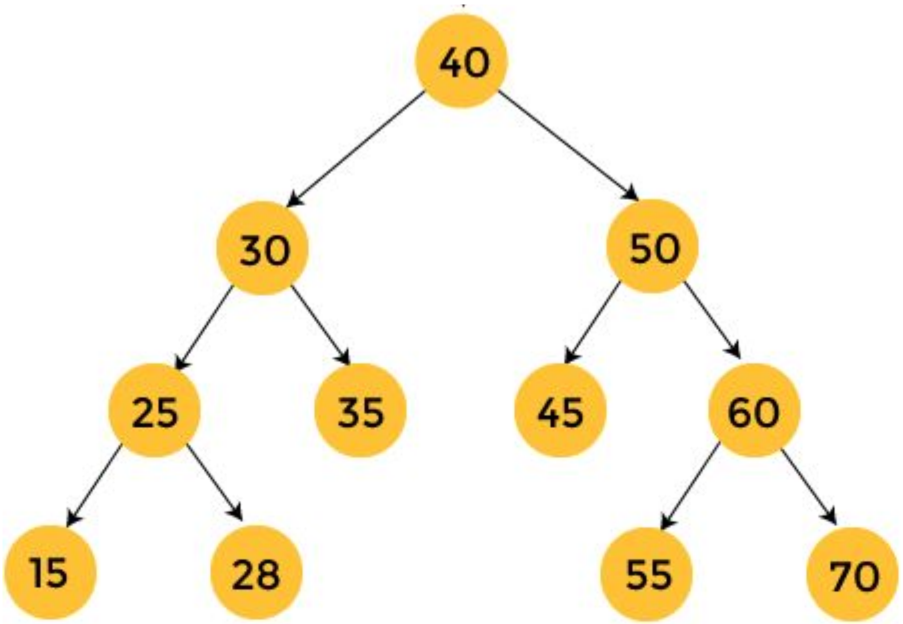
LUISS

Binary Search Tree - Exercises

Just to recap, a post-order visit is done as follow:

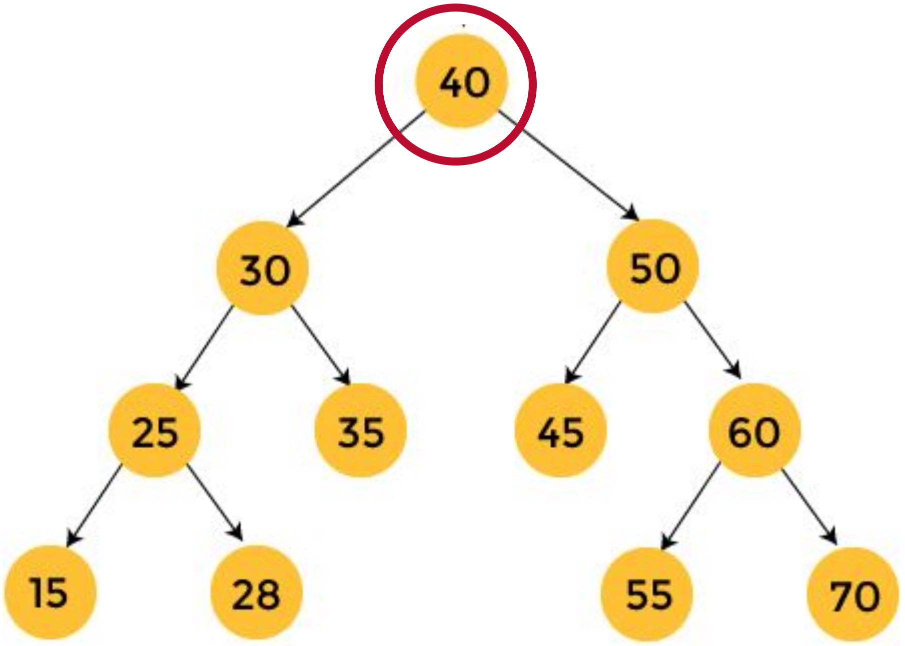
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```



Binary Search Tree - Exercises

We start exploring 40 and call again the function on the left node



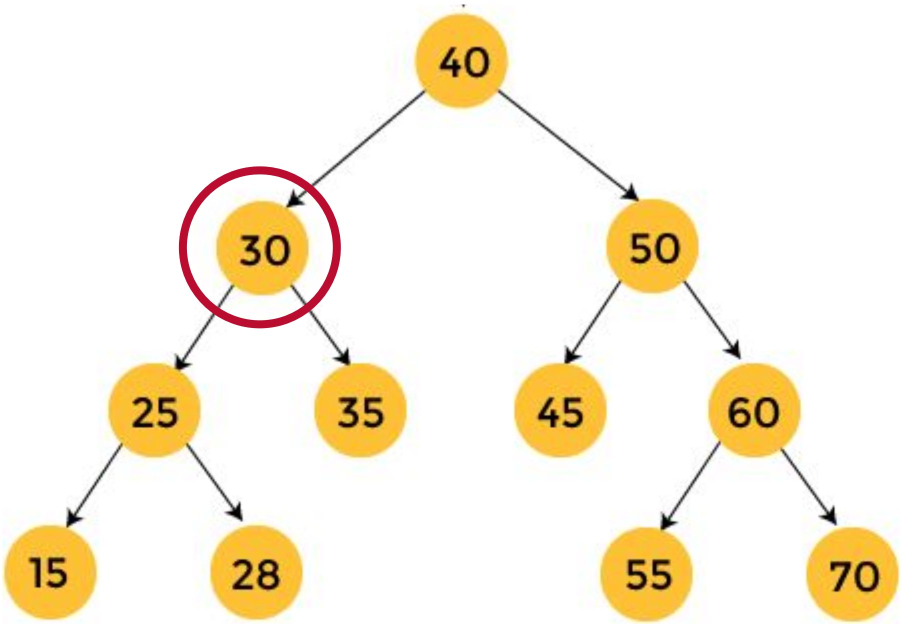
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

LUISS

Binary Search Tree - Exercises

We start exploring 30 and call again the function on the left node

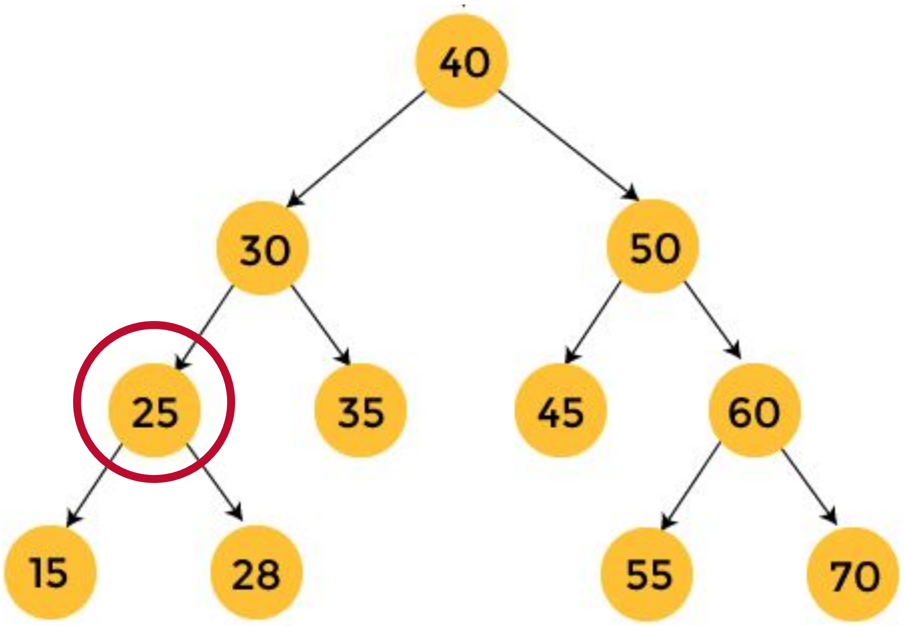


Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Binary Search Tree - Exercises

We start exploring 25 and call again the function on the left node



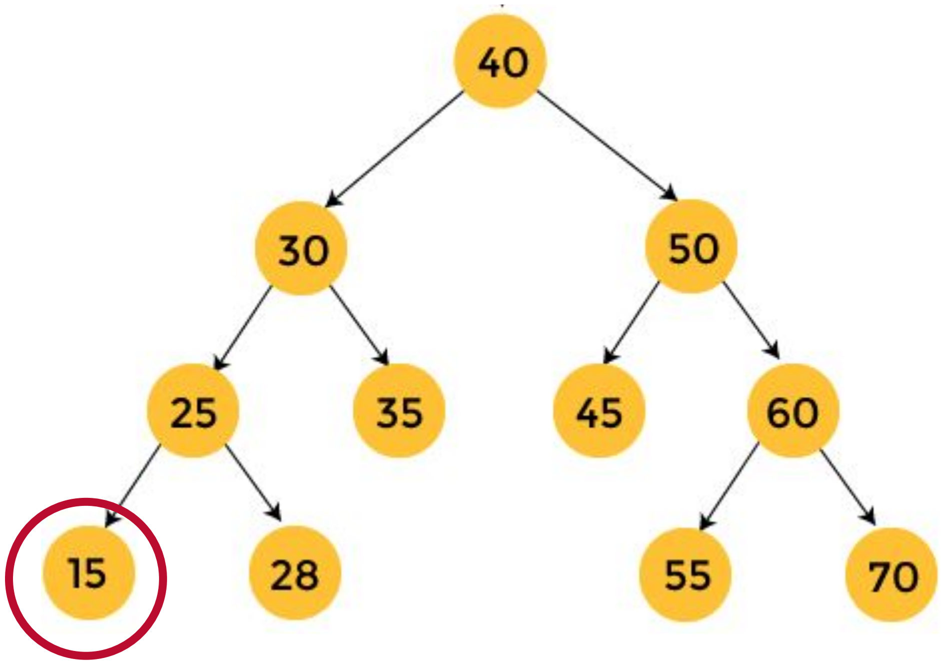
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Binary Search Tree - Exercises

We start exploring 15 and call again the function on the left node… But it has no children!

So we can mark it as visited!



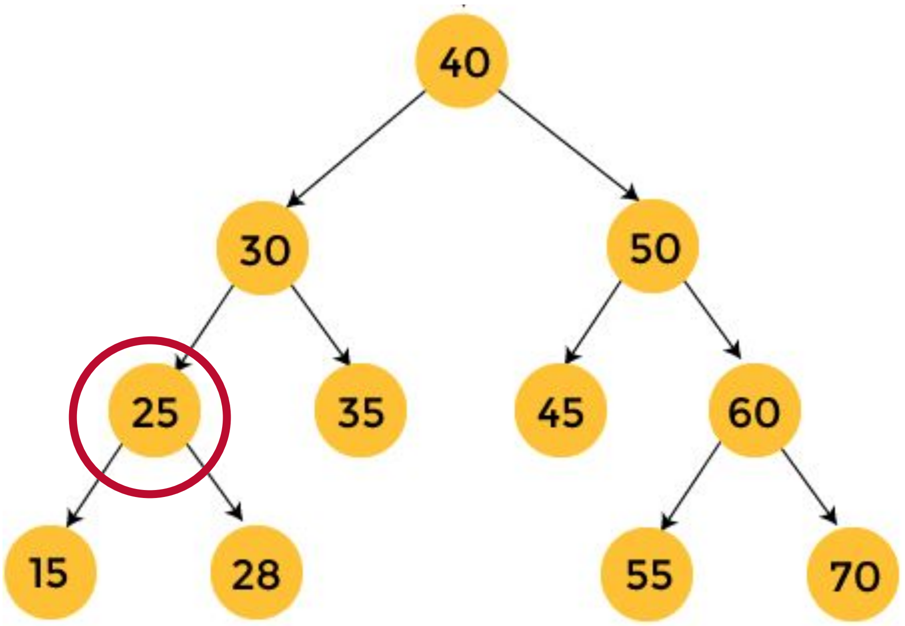
Nodes visited: 15

Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Binary Search Tree - Exercises

We return to 25 and call again the function on the right node



Nodes visited: 15

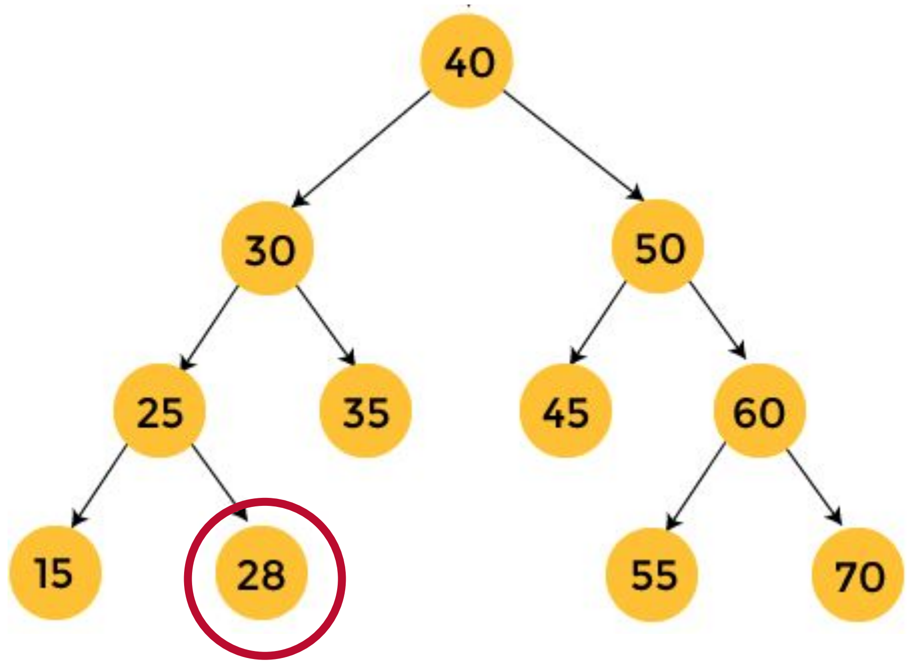
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Binary Search Tree - Exercises

We start exploring 28 and call again the function on the left node… But it has no children!

So we can mark it as visited!



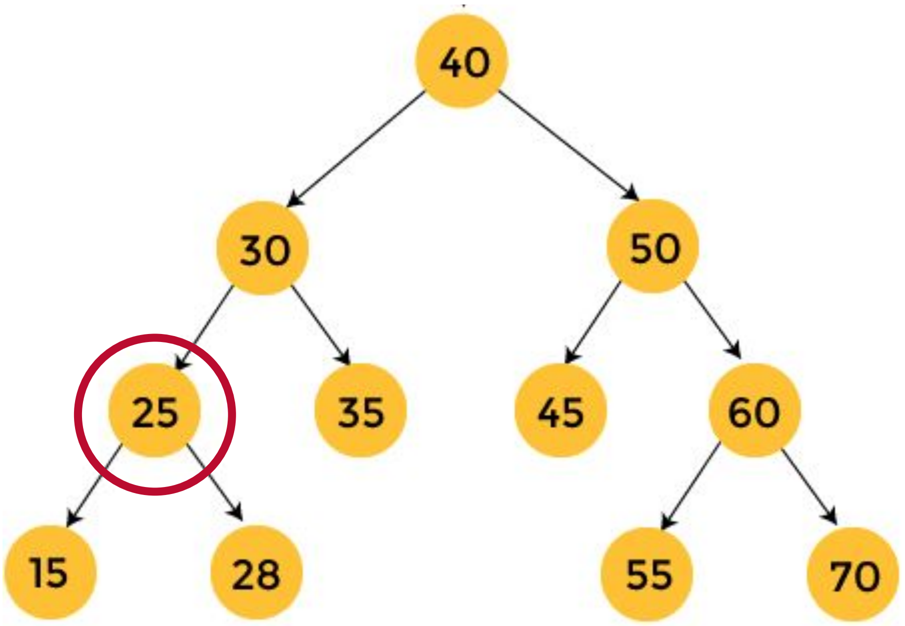
Nodes visited: 15, 28

Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Binary Search Tree - Exercises

We return to 25



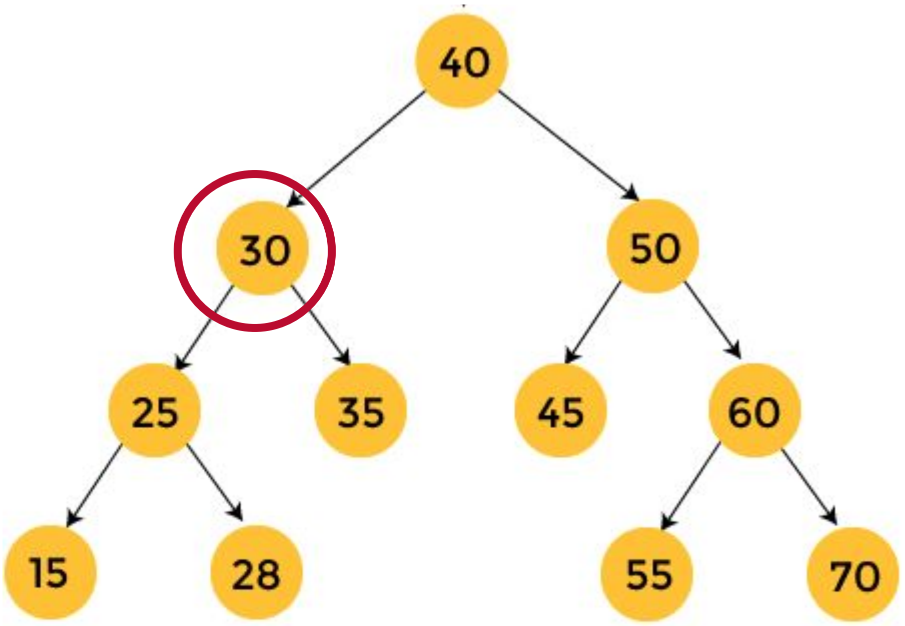
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: 15, 28

Binary Search Tree - Exercises

We return to 25 but we finished the exploration so we mark it as visited and we can return to 30



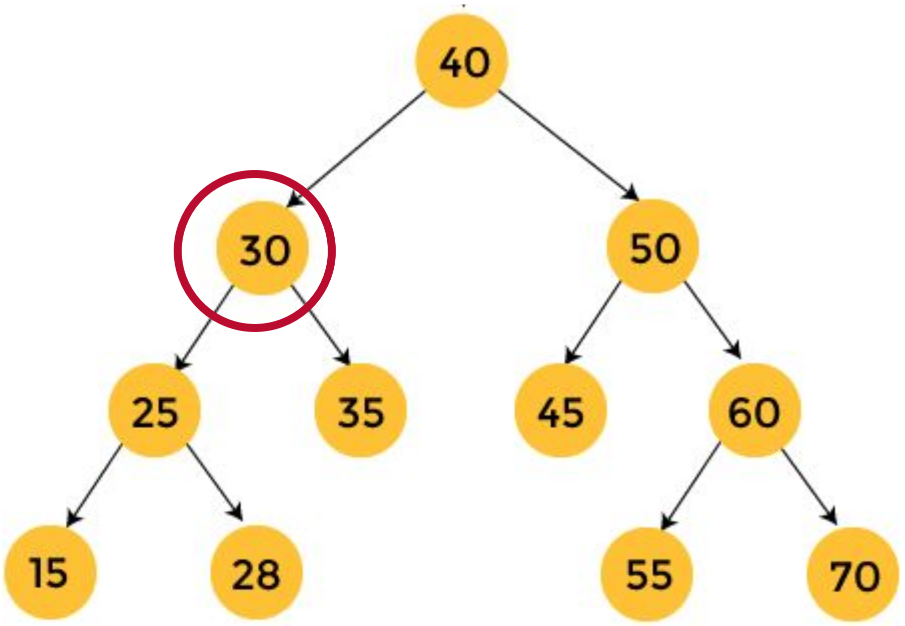
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: 15, 28, 25

Binary Search Tree - Exercises

Then we call again the function on the right node, namely, 35



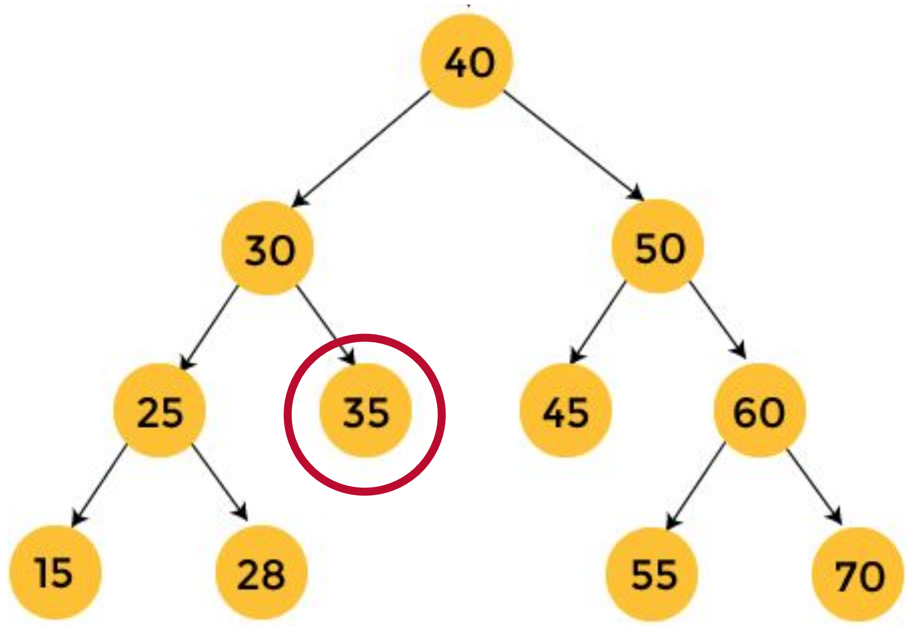
Nodes visited: 15, 28, 25

Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Binary Search Tree - Exercises

Then we call again the function on the right node, namely, 35 and since it has no children we can mark it as visited



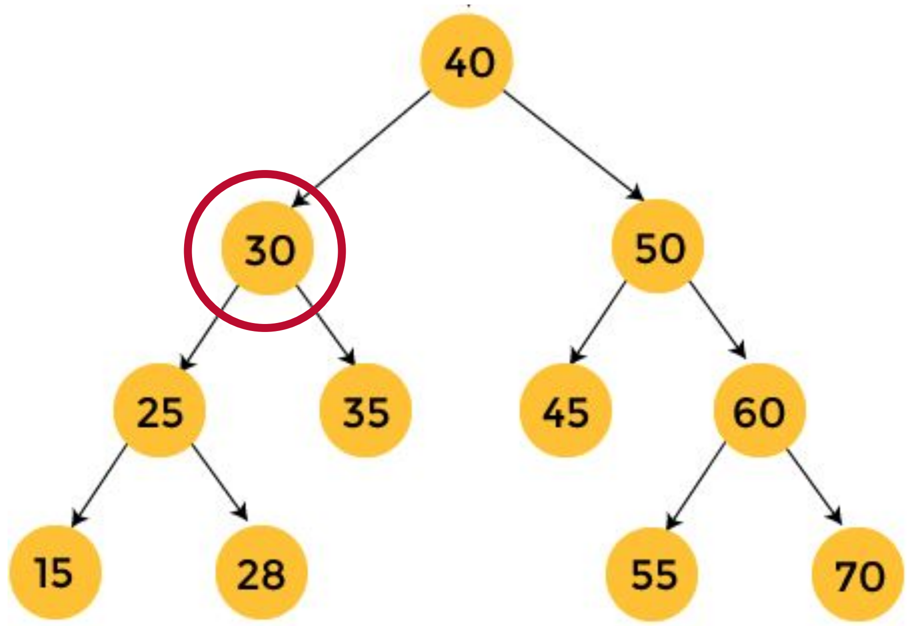
Nodes visited: 15, 28, 25, 35

Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Binary Search Tree - Exercises

We return to 30 and since the exploration is done we mark it as visited



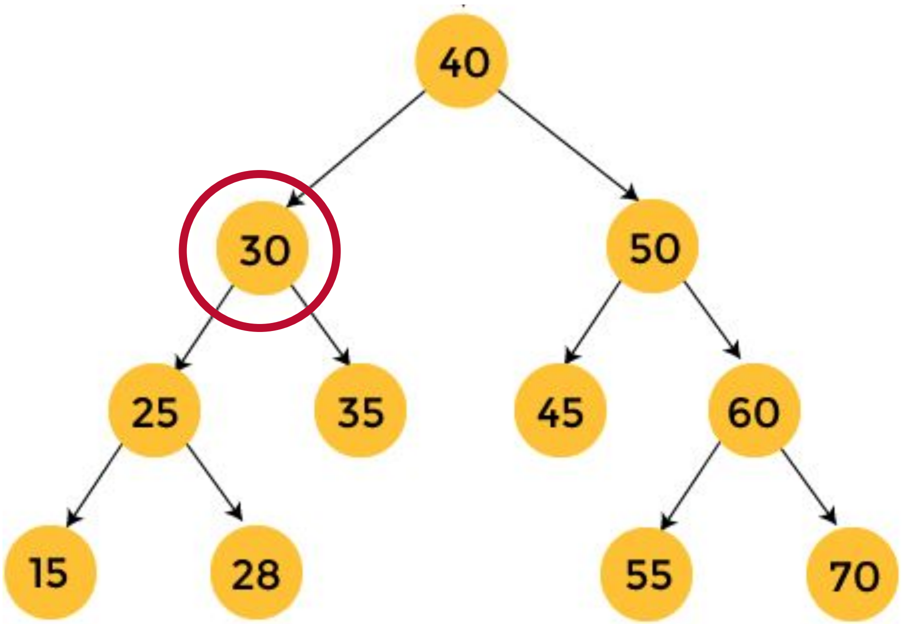
Nodes visited: 15, 28, 25, 35, 30

Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Binary Search Tree - Exercises

Now try to finish the exercise by yourself!



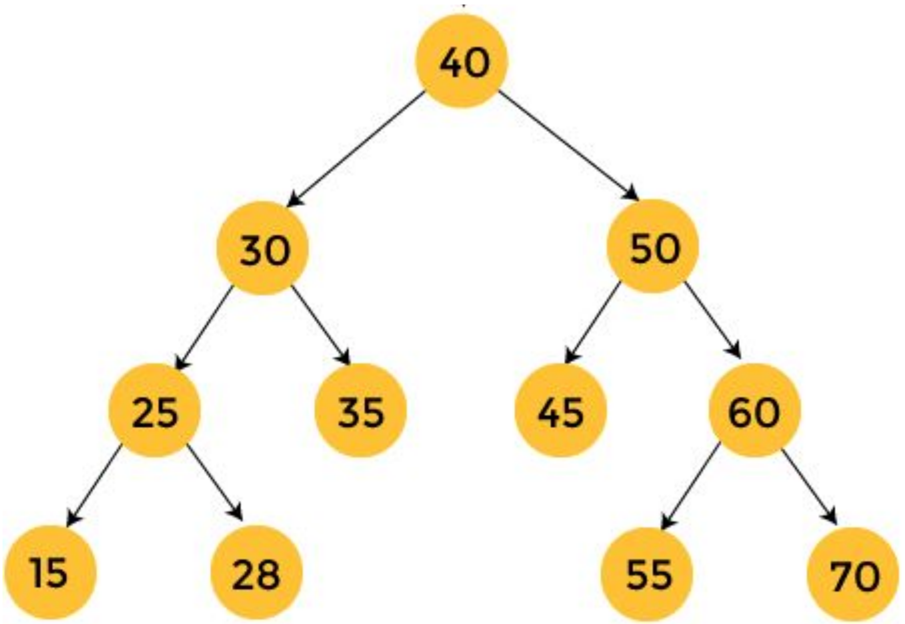
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: 15, 28, 25, 35, 30

Binary Search Tree - Exercises

Solution!



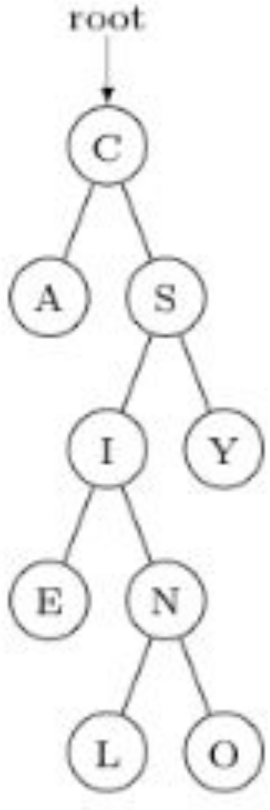
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: 15, 28, 25, 35, 30, 45, 55, 70, 60, 50, 40

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right

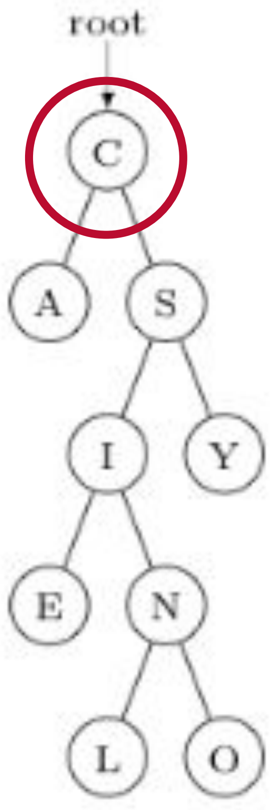


Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right

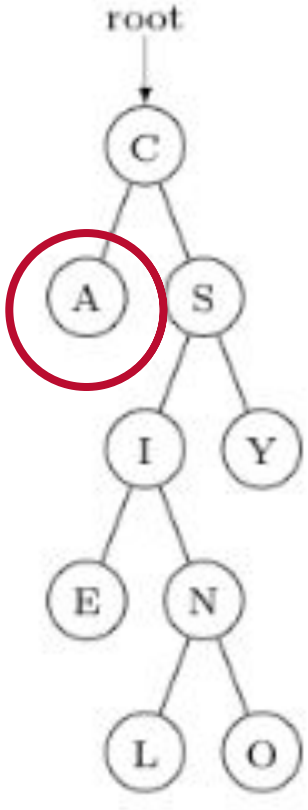


Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



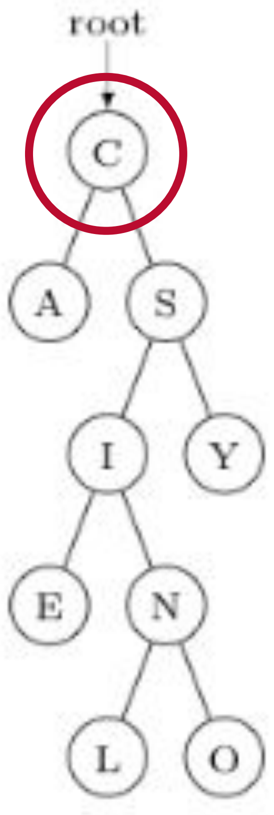
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



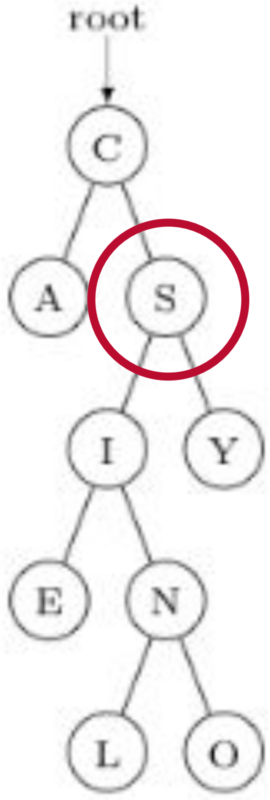
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



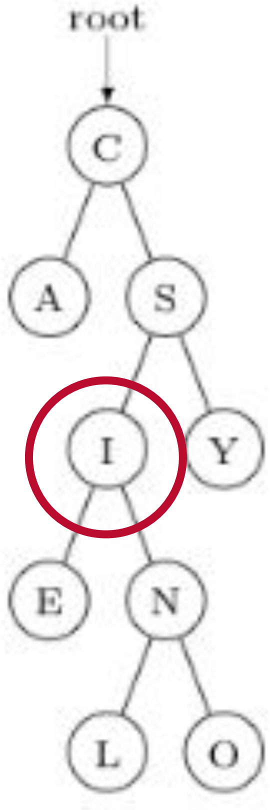
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



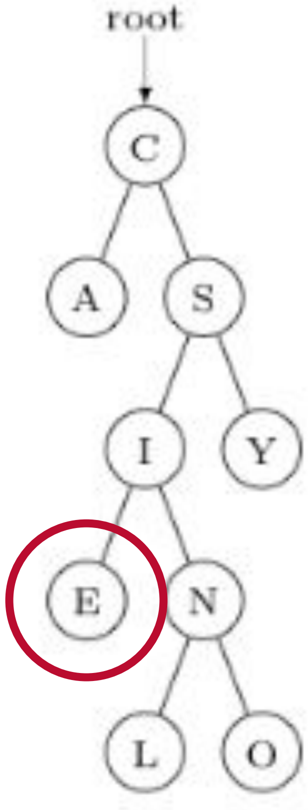
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



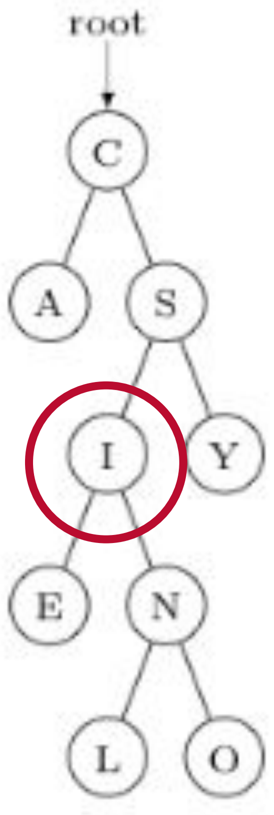
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A, E

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



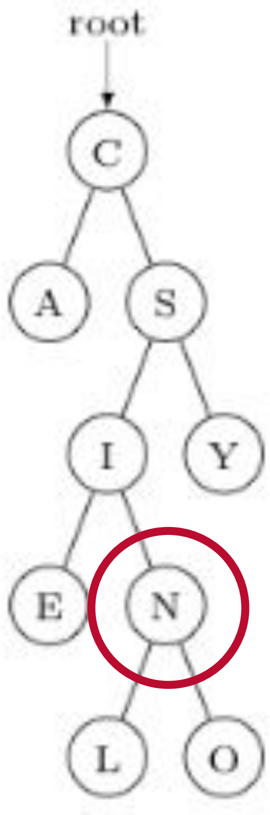
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A, E

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



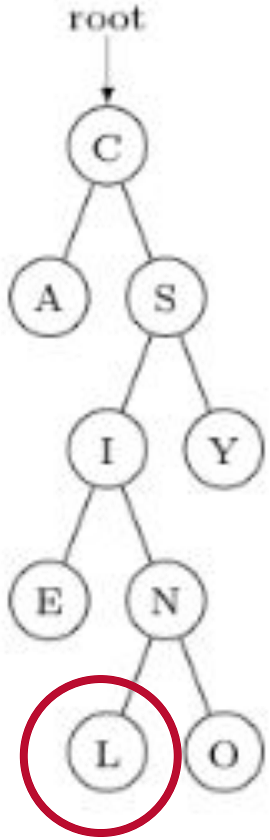
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A, E

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



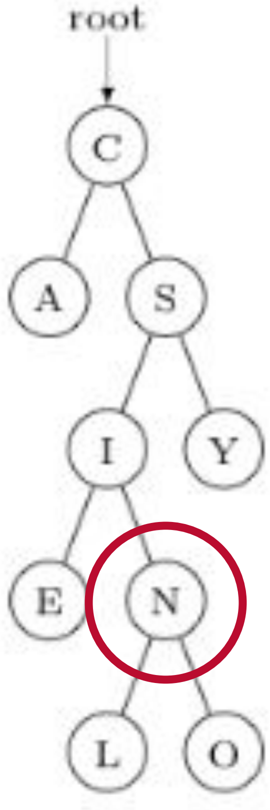
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A, E, L

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



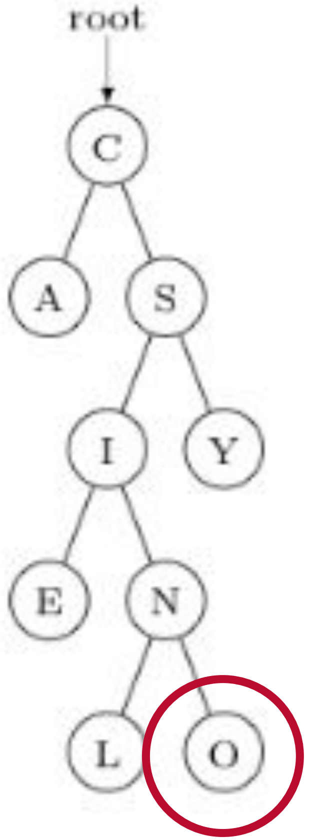
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A, E, L

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



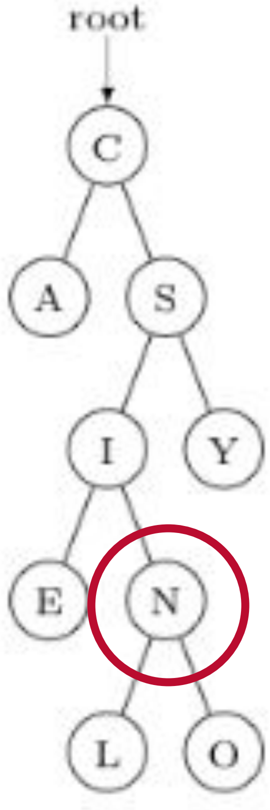
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A, E, L, O

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



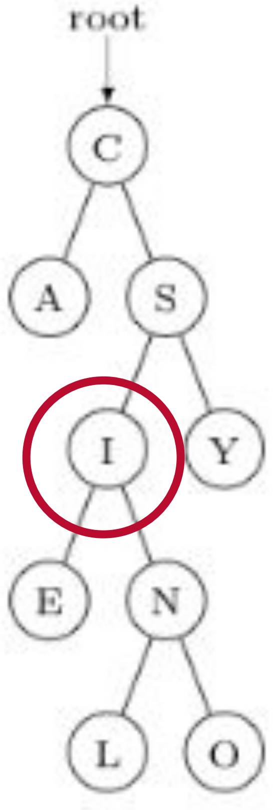
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A, E, L, O, N

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



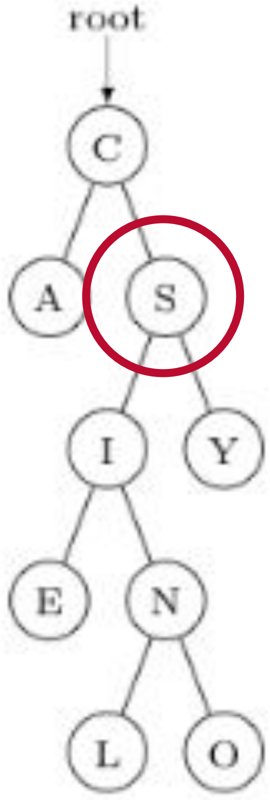
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A, E, L, O, N, I

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



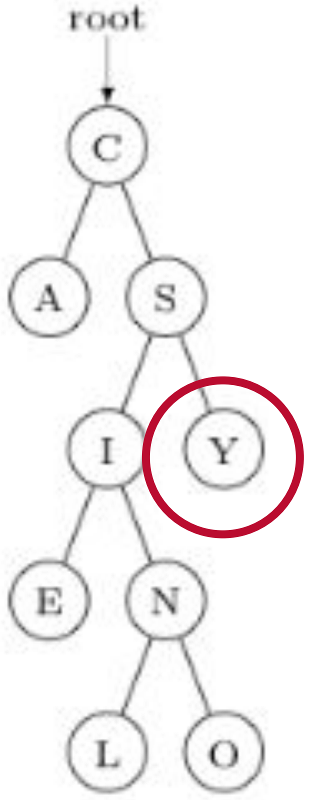
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A, E, L, O, N, I

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



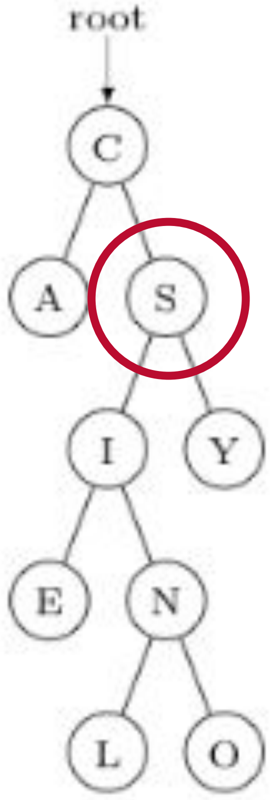
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A, E, L, O, N, I, Y

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



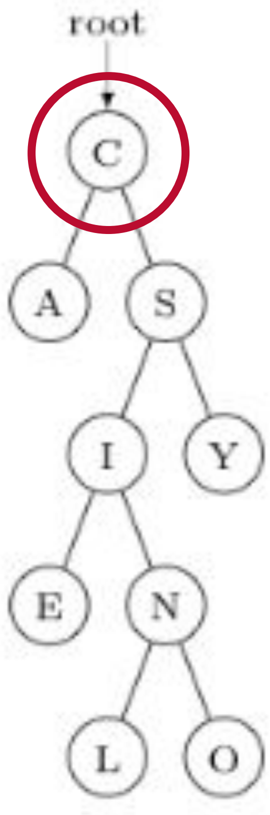
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A, E, L, O, N, I, Y, S

Binary Search Tree - Exercises

Perform a **post-order visit** of the tree on the right



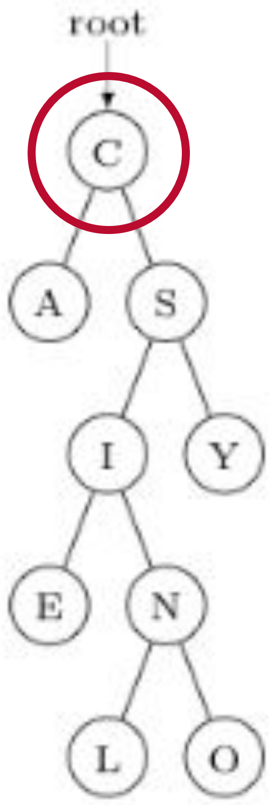
Algorithm Postorder(tree)

```
    Postorder(left->subtree)
    Postorder(right->subtree)
    Visit the root
```

Nodes visited: A, E, L, O, N, I, Y, S, C

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



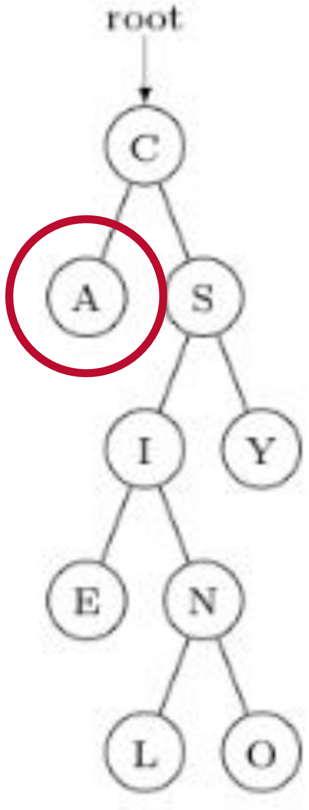
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited:

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



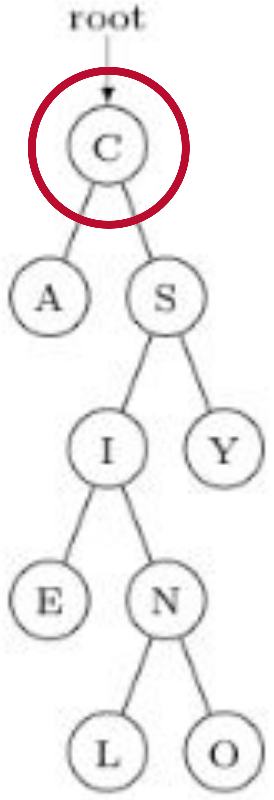
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



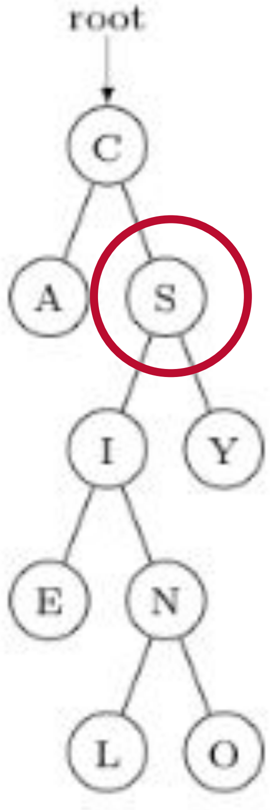
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



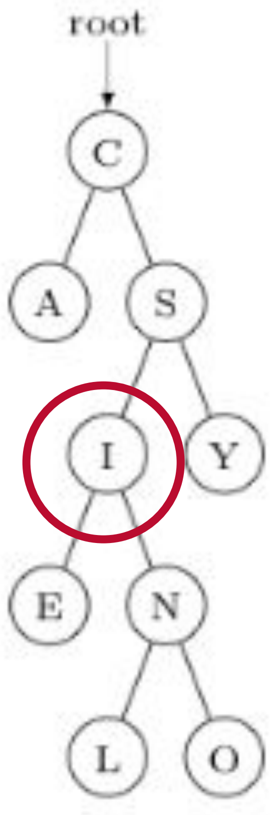
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



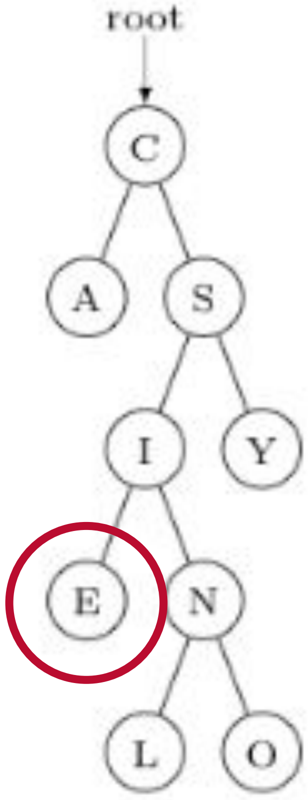
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



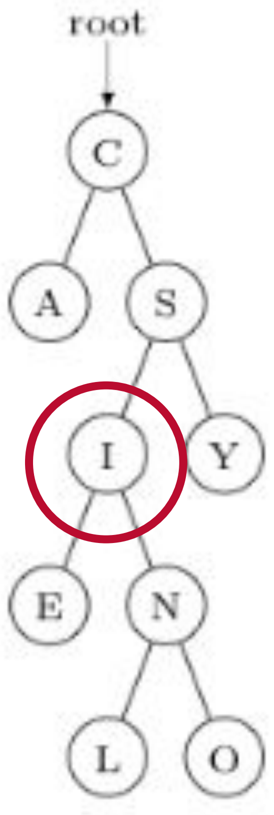
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C, E

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



Algorithm Inorder(tree)

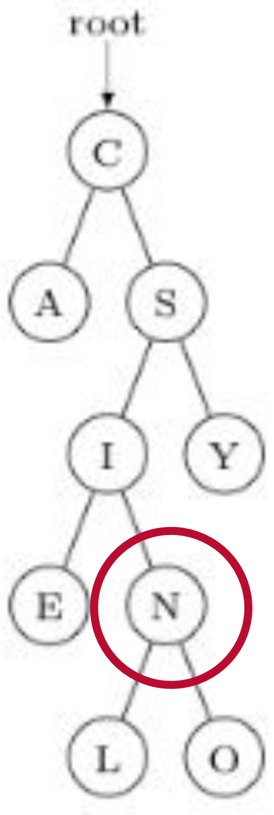
```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C, E, I

LUISS

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



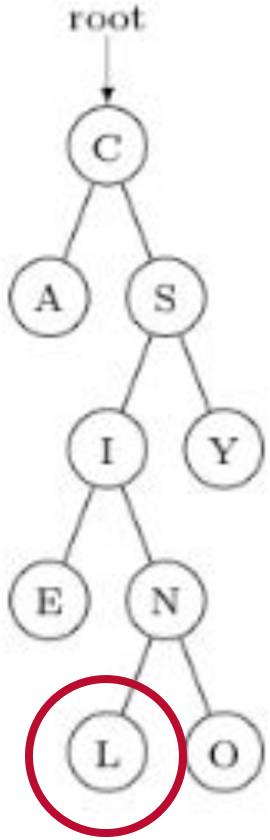
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C, E, I

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



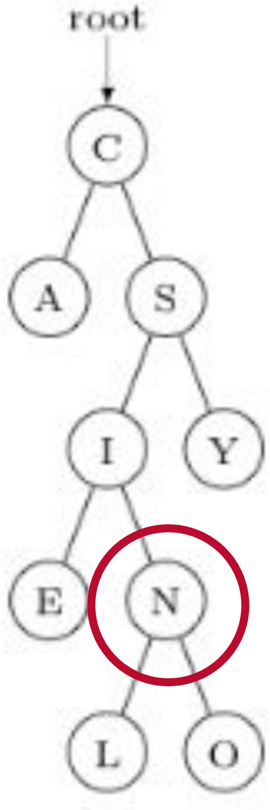
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C, E, I, L

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



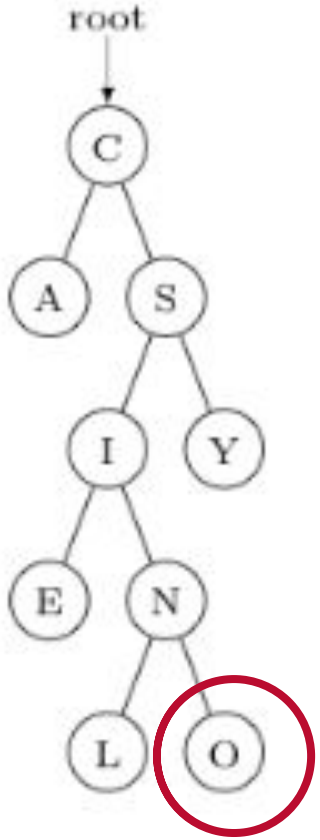
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C, E, I, L, N

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



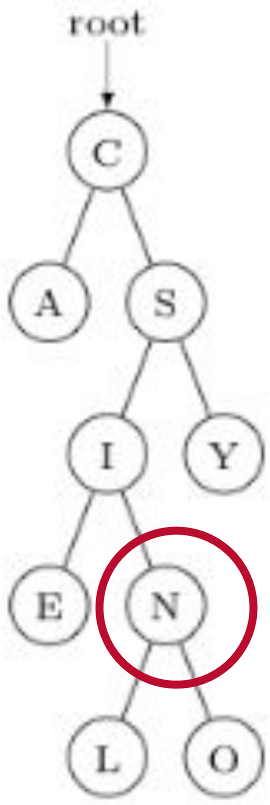
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C, E, I, L, N, O

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



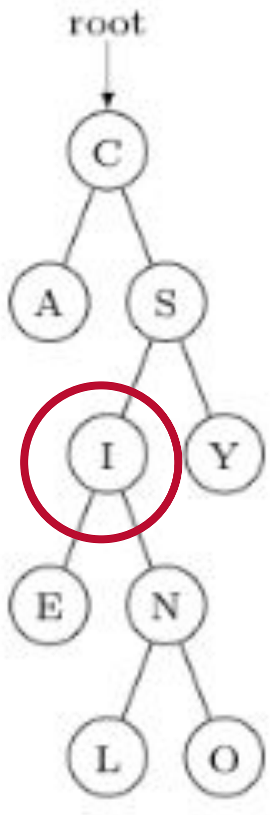
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C, E, I, L, N, O

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



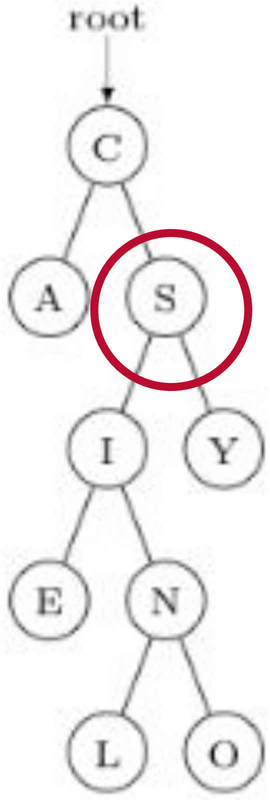
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C, E, I, L, N, O

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



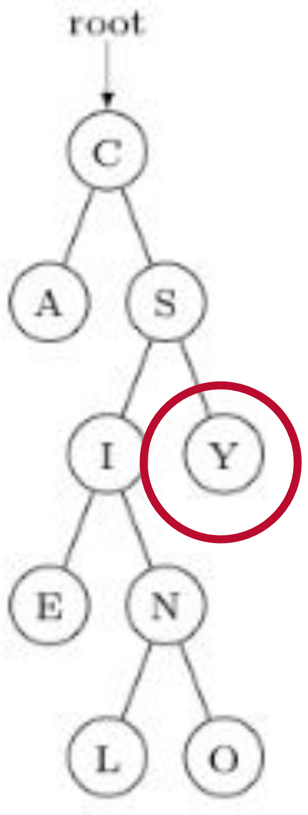
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C, E, I, L, N, O, S

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



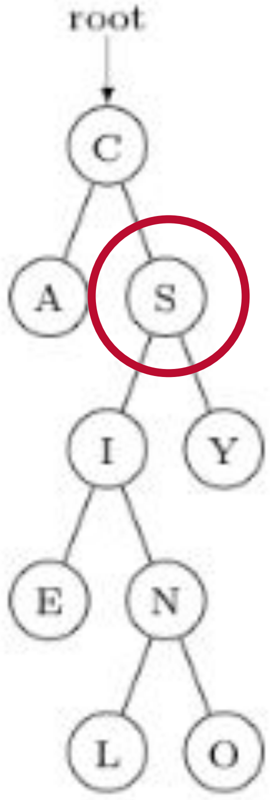
Algorithm Inorder(tree)

```
Inorder(left->subtree)
Visit the root
Inorder(right->subtree)
```

Nodes visited: A, C, E, I, L, N, O, S, Y

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right



Algorithm Inorder(tree)

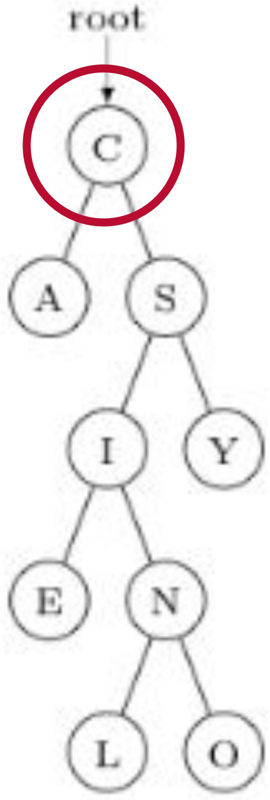
```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C, E, I, L, N, O, S, Y

Binary Search Tree - Exercises

Perform a **in-order visit** of the tree on the right

We found the solution!



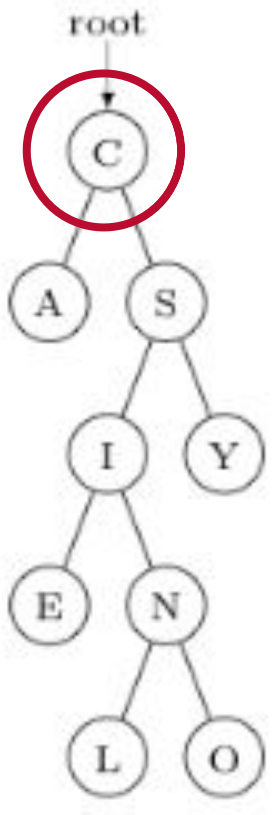
Algorithm Inorder(tree)

```
    Inorder(left->subtree)
    Visit the root
    Inorder(right->subtree)
```

Nodes visited: A, C, E, I, L, N, O, S, Y

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



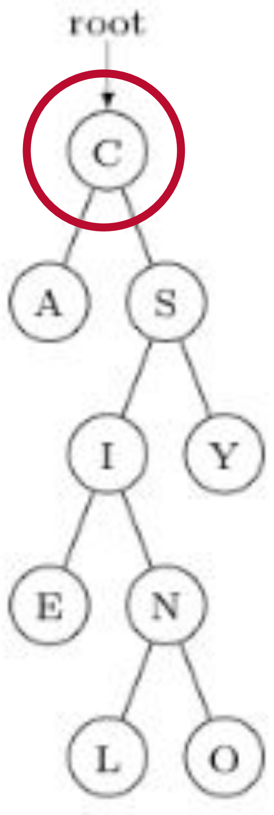
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited:

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



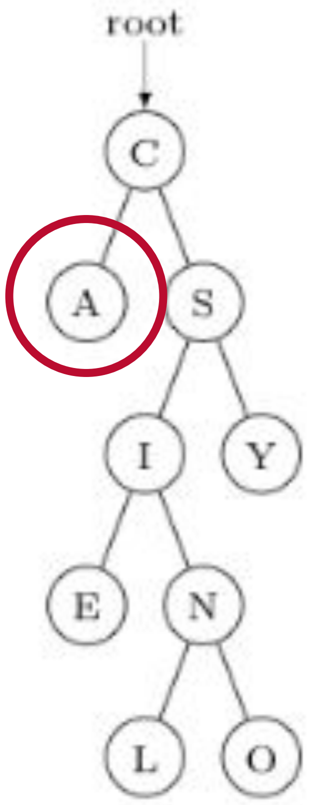
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited: C

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



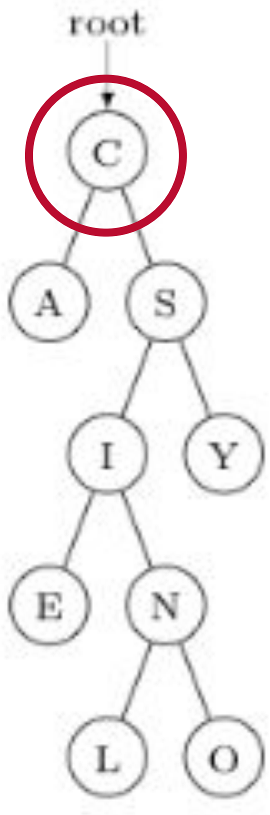
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited: C, A

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



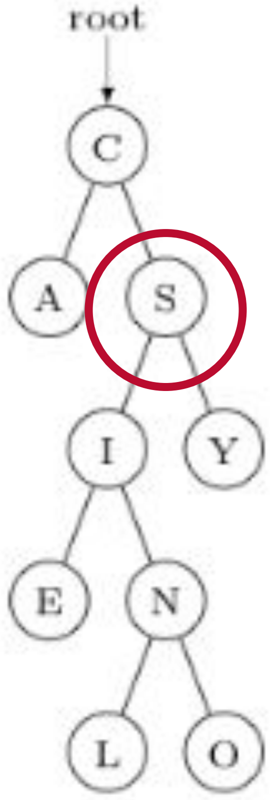
Algorithm Preorder(tree)

Visit the root
Preorder(left->subtree)
Preorder(right->subtree)

Nodes visited: C, A

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



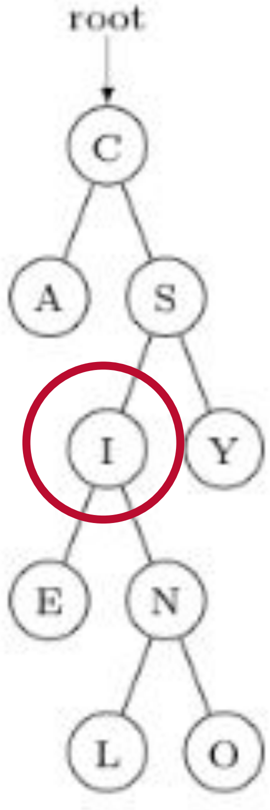
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited: C, A, S

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



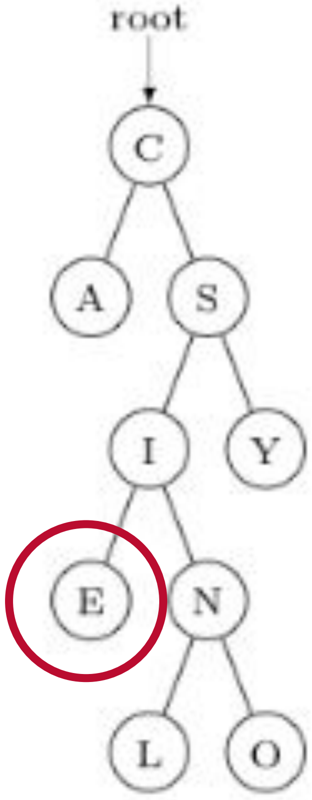
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited: C, A, S, I

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



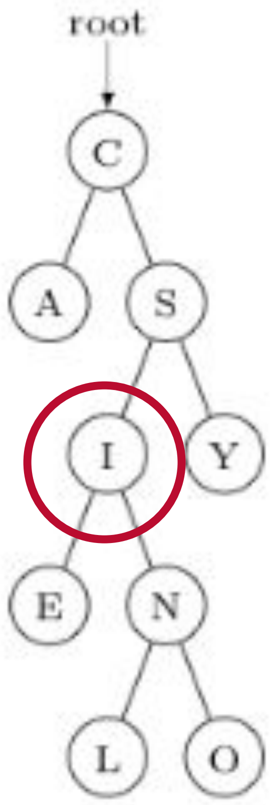
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited: C, A, S, I, E

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



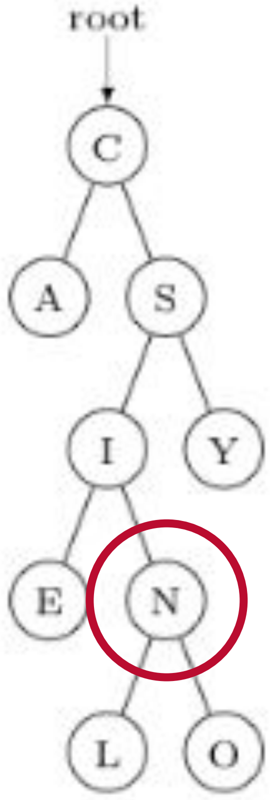
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited: C, A, S, I, E

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



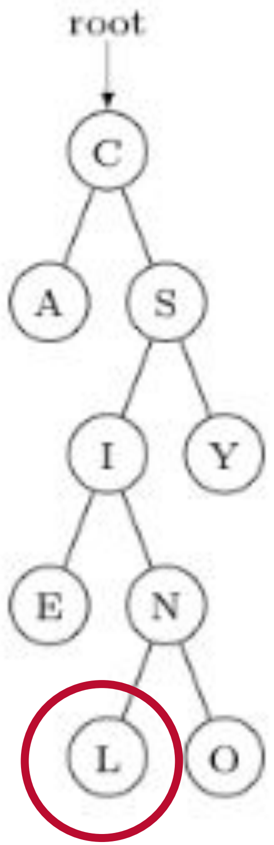
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited: C, A, S, I, E, N

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



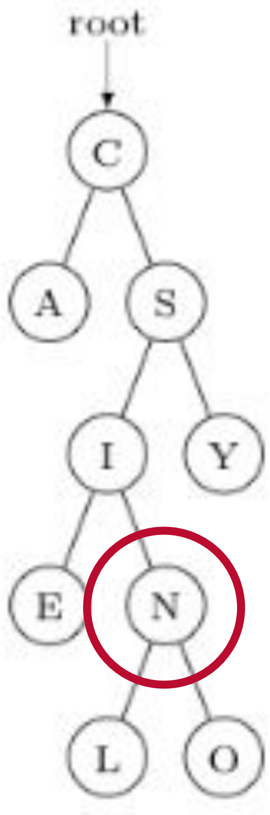
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited: C, A, S, I, E, N, L

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



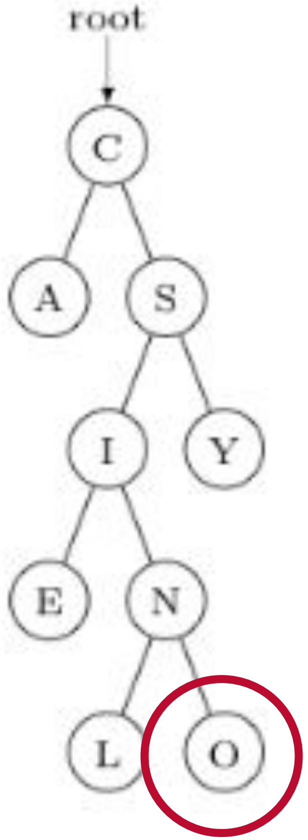
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited: C, A, S, I, E, N, L

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



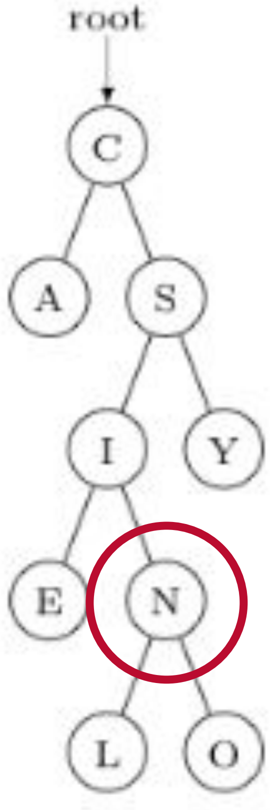
Algorithm Preorder(tree)

Visit the root
Preorder(left->subtree)
Preorder(right->subtree)

Nodes visited: C, A, S, I, E, N, L, O

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



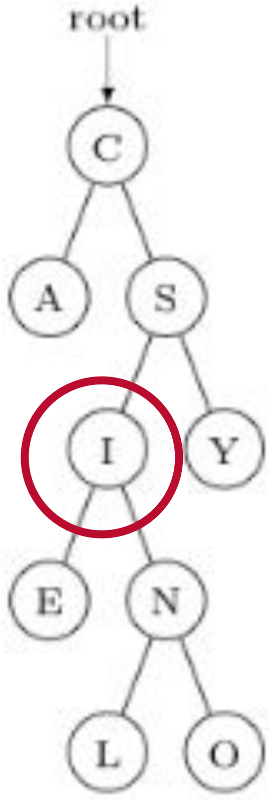
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited: C, A, S, I, E, N, L, O

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



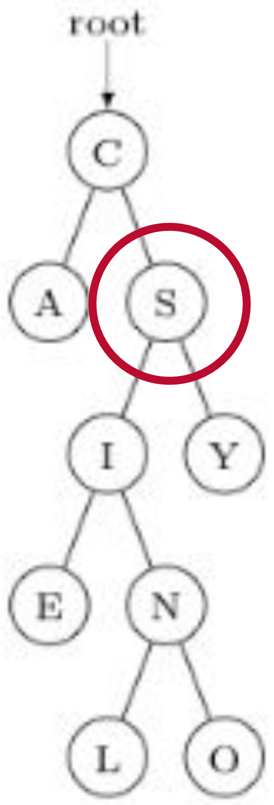
Algorithm Preorder(tree)

Visit the root
Preorder(left->subtree)
Preorder(right->subtree)

Nodes visited: C, A, S, I, E, N, L, O

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



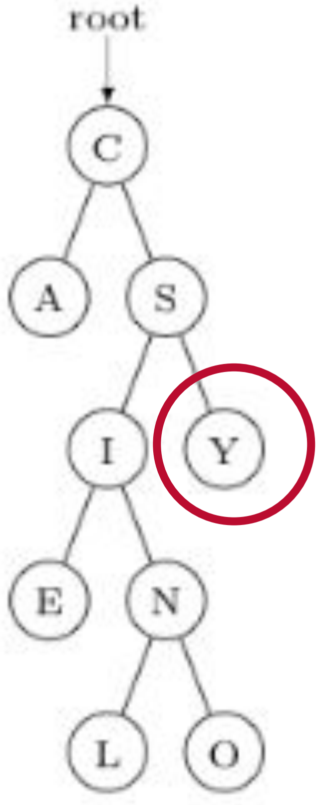
Algorithm Preorder(tree)

```
Visit the root
Preorder(left->subtree)
Preorder(right->subtree)
```

Nodes visited: C, A, S, I, E, N, L, O

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



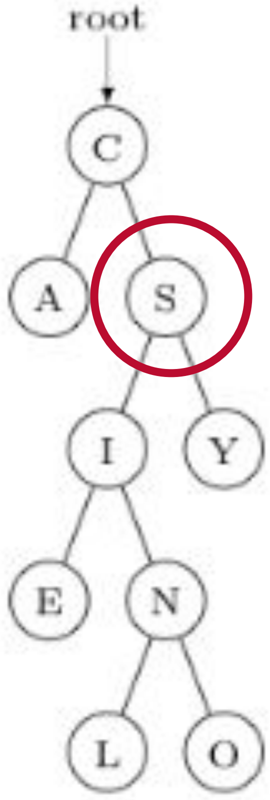
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited: C, A, S, I, E, N, L, O, Y

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



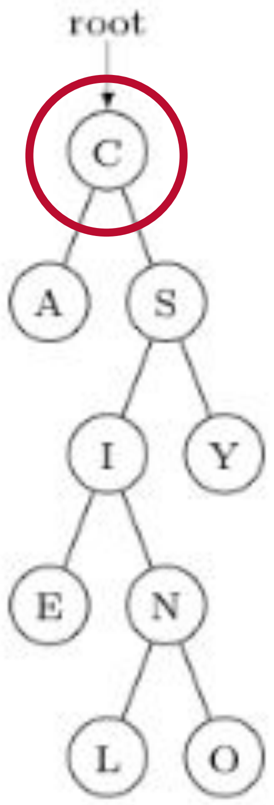
Algorithm Preorder(tree)

```
    Visit the root
    Preorder(left->subtree)
    Preorder(right->subtree)
```

Nodes visited: C, A, S, I, E, N, L, O, Y

Binary Search Tree - Exercises

Perform a **pre-order visit** of the tree on the right



Algorithm Preorder(tree)

```
Visit the root
Preorder(left->subtree)
Preorder(right->subtree)
```

Nodes visited: C, A, S, I, E, N, L, O, Y