

(/cs/) (https://www.baeldung.com/cs/)

(https://freestar.com/?
n_campaign=branding&utm_medium=banner&utm_source=baeldung.com&utm_content=baeldung_leaderboard
atf)

Finding the In-Order Successor of a Node

Last modified: January 1, 2022

| by Milos Simic (https://www.baeldung.com/cs/author/milossimic)

Graph Traversal (https://www.baeldung.com/cs/category/algorithms/graph-traversal)

Trees (https://www.baeldung.com/cs/category/graph-theory/trees)

If you have a few years of experience in Computer Science or research, and you're interested in sharing that experience with the community, have a look at our **Contribution Guidelines** (</cs/contribution-guidelines>).

1. Introduction

In this tutorial, we'll show three ways to find the in-order successor of a node in a binary search tree (BST) (</cs/binary-search-trees>). In such a tree, each node is \geq that the nodes in its left sub-tree and $<$ than the nodes in its right sub-tree.

Nodes can be numbers, strings, tuples, in general, objects that we can compare to one another.

2. What Is the In-Order Successor of a Node?

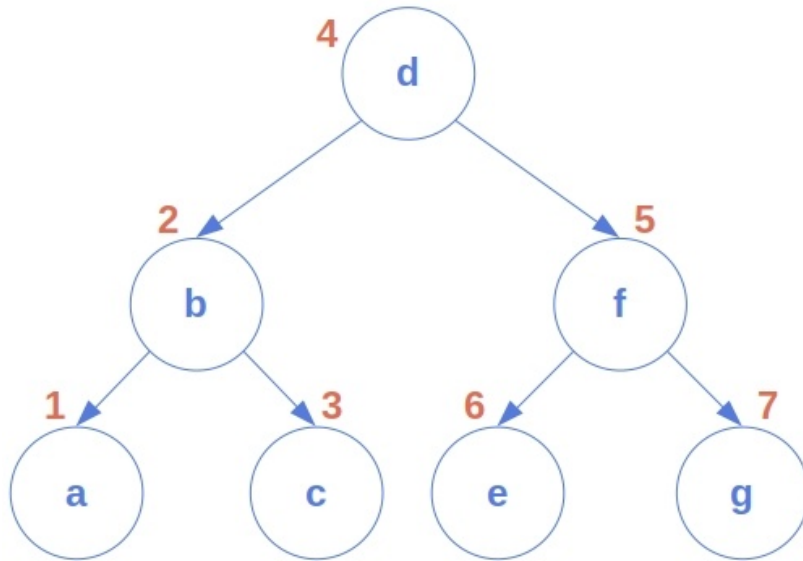
In-order traversal (</cs/tree-traversal-time-complexity>) is a tree traversal technique we recursively define as follows:

Algorithm 1: Identifying the Root of a Tree in the Graph

Data: *node* - a node in a tree**function** IN-ORDER-TRAVERSE(*node*) : **if** *node* \neq *NONE* **then** IN-ORDER-TRAVERSE(*node.left*) VISIT(*node*) IN-ORDER-TRAVERSE(*node.right*) **end****end**

Applied to a BST, it visits the nodes in the non-decreasing order, as in the example of the following tree:

(<https://freestar.com/?>



The order in which the nodes are visited: a b c d e f g

Our goal is to find the immediate successor of a given node x in the in-order traversal of a tree. The successor is the smallest node of all those greater than x in the tree.

2.1. Where to Find the In-Order Successor?

If we take a look at the above image, we'll see a pattern.

If a node x has a right child, its successor is the minimal node of its right sub-tree. In other words, the node's successor is the leftmost descendant of its right child in this case. Why? Because, after visiting x , the traversal procedure will process the right sub-tree of x , and the first to visit there is its leftmost leaf.


If x has no right child, its in-order successor is located above it in the tree, among its ancestors. While unfolding the recursive calls, the in-order traversal function will first visit the node whose left child was the most recent input. **So, the x 's successor is the parent of the x 's youngest ancestor that is a left child.**

3. Finding the Successor in an Out-Tree

Usually, we implement the nodes as structures with three attributes. One is the node's content: the object placed at that place in the tree hierarchy. The other two are pointers to the node's left and right children.

Conceptually, **this implementation corresponds to an out-tree**

([https://en.wikipedia.org/wiki/Arborescence_\(graph_theory\)](https://en.wikipedia.org/wiki/Arborescence_(graph_theory))): a tree with edges oriented away from its root.



(https://freestar.com/?utm_campaign=branding&utm_medium=&utm_source=baeldung.com&utm_content=baeldung_leaderboard_mid_2)

3.1. Algorithm

To find a node's in-order successor in an out-tree, we should first locate the node, memorizing the parents of the left children along the way. Once we find it, we check if it has the right child. If that's the case, we return the right sub-tree's leftmost leaf. Otherwise, we return the most recently memorized parent above the node:

Algorithm 2: Finding the Successor in an Out-tree

Data: x - the node whose successor we search for, $root$ - the root of the out-tree to search

Result: The in-order successor of x in the tree, or $NONE$ if x isn't there.

$parent \leftarrow NONE$

$node \leftarrow root$

while $node \neq NONE$ and $node.content \neq x$ **do**

if $x < node.content$ **then**

$parent \leftarrow node$

$node \leftarrow node.left$

else

$node \leftarrow node.right$

end

end

if $node = NONE$ **then**

return $NONE$

end

if $node.right \neq NONE$ **then**

$successor \leftarrow node.right$

while $successor.left \neq NONE$ **do**

$successor \leftarrow successor.left$

end

return $successor$

else

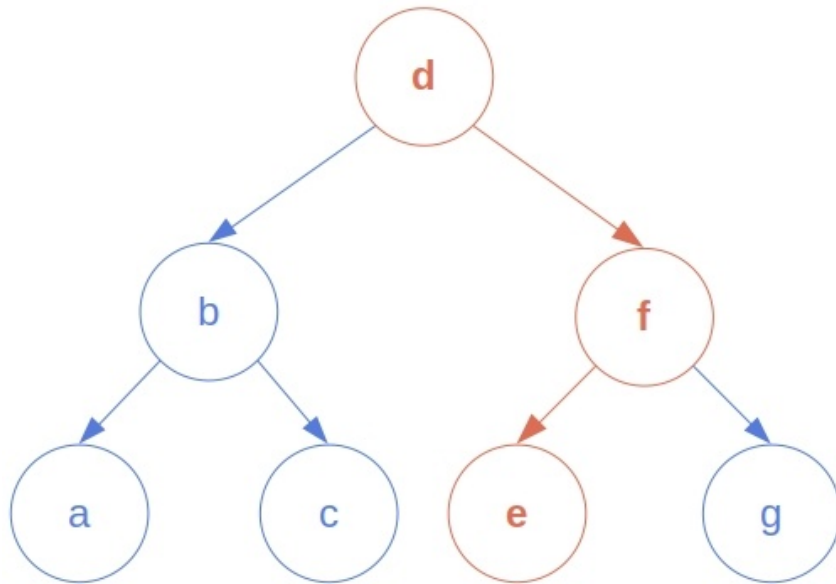
return $parent$

end

If x is the maximal node, it has no successor. So, in that case, we return $NONE$.

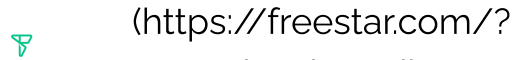
3.2. Complexity Analysis

Let h be the height of the whole BST (/cs/height-balanced-tree). Let $d(y, z)$ be the length of the path from y to z in the tree. If x has no right child, we visit $d(\text{root}, x) \leq h$ nodes to find x . Afterward, we output the successor without further processing. If x does have the right child, we find the successor upon reaching a leaf. The depth of any leaf in a tree is bounded from above by the tree's height. So, **we don't make more than $\Theta(h)$ steps in the worst case:**



The nodes visited while searching for the successor of **d**.

If the tree is balanced (/cs/self-balancing-bts), it will hold that $h = \Theta(\log n)$, so our algorithm will run in logarithmic time $O(\log n)$, where n is the number of nodes in the tree. However, non-balanced trees may be degenerate. Their height is $n - 1$ in the worst case, so the algorithm will be linear for such inputs.



4. Finding the Successor in a Bidirectional Implementation of BST

Usually, the nodes contain pointers only to their children. However, **we sometimes store the pointers to parents as well**. The reason is that it simplifies many tree operations. Even though we take more memory this way, the overhead is negligible most of the time.

4.1. Algorithm

In such a tree, we don't have to keep track of the most recent parent with a left child while searching for x . Instead, if $x.right$ doesn't exist, we follow the to-parent pointers until we find the one we seek:

Algorithm 3: Finding the Successor in an Out-tree

Data: x - the node whose successor we search for, $root$ - the root of the bidirectional tree to search

Result: The in-order successor of x in the tree, or $NONE$ if x isn't there.

$node \leftarrow root$

while $node \neq NONE$ and $node.content \neq x$ **do**

if $x < node.content$ **then**

$node \leftarrow node.left$

else

$node \leftarrow node.right$

end

end

if $node = NONE$ **then**

return $NONE$

end

if $node.right \neq NONE$ **then**

$successor \leftarrow node.right$

while $successor.left \neq NONE$ **do**

$successor \leftarrow successor.left$

end

return $successor$

else

while $node.parent \neq NONE$ and $node.parent.left \neq node$ **do**

$node \leftarrow node.parent$

end

return $node.parent$

end

The complexity of this approach is the same as that of the algorithm for out-trees.

5. In-Order Search

The third and final way we'll present is the in-order search. **Since we're looking for the immediate successor of x , we can run the in-order traversal until visiting a node greater than x .** Since the in-order procedure visits the nodes in the non-descending order, we can be sure that the first visited node greater than x is its successor.

Here's the pseudocode:

Algorithm 4: In-Order Search

Data: x - the node whose successor we search for, $node$ - the node we're currently traversing

function IN-ORDER-SEARCH($node, x$) :

if $node \neq NONE$ **then**

$successor \leftarrow$ IN-ORDER-SEARCH($node.left$)

if $successor \neq NONE$ **then**

return $successor$

else

if $node.content > x$ **then**

return $node.content$

else

return IN-ORDER-SEARCH($node.right$)

end

end

end

end

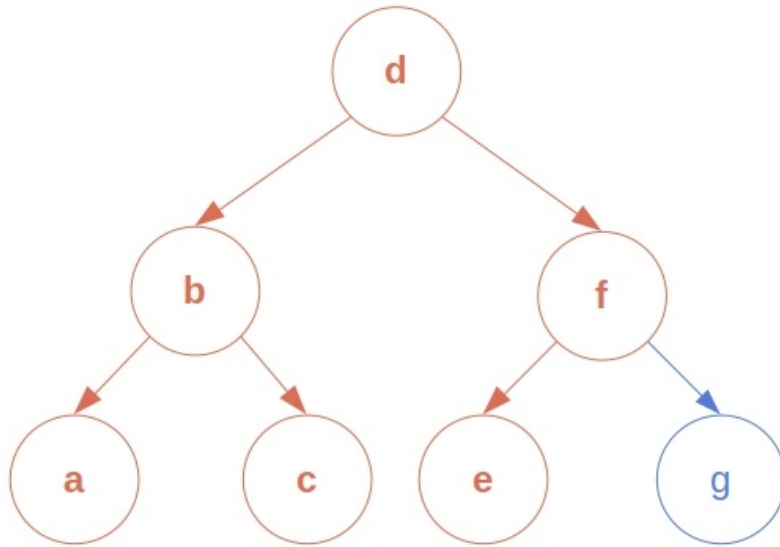
5.1. Complexity Analysis

This method traverses $\geq \text{rank}(x)$ nodes, where $\text{rank}(x) \in \{1, 2, \dots, n\}$ is the rank of x among the nodes in the tree:



(<https://freestar.com/?>

utm_campaign=branding&utm_medium=banner&utm_source=baeldung.com&utm_content=baeldung_incontent



The nodes visited while searching for the successor of d.

In the worst case, x will be the greatest node, so $rank(x) = n$. Therefore, **the in-order search visits all the n nodes and runs in $O(n)$ time in the worst-case scenario.**

6. Discussion

Which algorithm to choose? The first two are unaffected by the node's rank, while the third approach's complexity doesn't depend on the tree's height.

At first glance, it may seem that the first two algorithms are better. After all, if a tree is balanced, both are of logarithmic time complexity, whereas the in-order approach is linear in any case. However, building and maintaining a balanced tree isn't a simple job. For example, if we often update the tree but mostly ask for the successor of a low-rank node, the in-order search will run faster in practice. The reason is that the updates won't require rebalancing the tree.

What about duplicate entries? The first two algorithms locate the closest-to-the root node that contains x . All the other nodes equal to it are in its left sub-tree, which we don't process at all. So, the returned successor is going to be the first greater than x . That is also the case with the third approach.

7. Conclusion

In this article, we presented three ways to find the in-order successor of a node. One is a simple extension of the in-order traversal algorithm. The other use of the fact is that a node's successor is either the leftmost descendant of its right child or the parent of its youngest ancestor that is a left child.

If you have a few years of experience in Computer Science or research, and you're interested in sharing that experience with the community, have a look at our **Contribution Guidelines** (</cs/contribution-guidelines>).

Comments are closed on this article!



(<https://freestar.com/>?)

CATEGORIES

ALGORITHMS (/CS/CATEGORY/ALGORITHMS)

ARTIFICIAL INTELLIGENCE (/CS/CATEGORY/AI)

CORE CONCEPTS (/CS/CATEGORY/CORE-CONCEPTS)

DATA STRUCTURES (/CS/CATEGORY/DATA-STRUCTURES)

GRAPH THEORY (/CS/CATEGORY/GRAPH-THEORY)

LATEX (/CS/CATEGORY/LATEX)

NETWORKING (/CS/CATEGORY/NETWORKING)

SECURITY (/CS/CATEGORY/SECURITY)

SERIES

[DRAWING CHARTS IN LATEX \(/CS/CATEGORY/SERIES\)](#)

ABOUT

[ABOUT BAELDUNG \(HTTPS://WWW.BAELDUNG.COM/ABOUT\)](#)

[THE FULL ARCHIVE \(/CS/FULL_ARCHIVE\)](#)

[WRITE FOR BAELDUNG \(/CS/CONTRIBUTION-GUIDELINES\)](#)

[EDITORS \(HTTPS://WWW.BAELDUNG.COM/EDITORS\)](#)

[TERMS OF SERVICE \(HTTPS://WWW.BAELDUNG.COM/TERMS-OF-SERVICE\)](#)

[PRIVACY POLICY \(HTTPS://WWW.BAELDUNG.COM/PRIVACY-POLICY\)](#)

[COMPANY INFO \(HTTPS://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO\)](#)

[CONTACT \(/CONTACT\)](#)