Ask a Question

Introduction to Tree Depth-First Search

Let’s go over the Tree Depth-first Search pattern, its real-world applications, and some problems we can solve with it.

We'll cover the following...

About the pattern

A tree is a graph that contains the following properties:

- It is undirected.
- It is acyclic.
- It is a connected graph where any two vertices are connected by exactly one path.
- Its nodes can contain values of any data type.

The following key features set trees apart from other data structures, such as arrays or linked lists:

- They organize data in a hierarchical manner with a root node at the top and child nodes branching out from it.
- They are nonlinear, which means that the elements in a tree are not arranged sequentially but rather in a branching structure.

- The time complexity for search and insert operations in trees is typically $O(\log n)$, where n is the number of elements in the tree. In contrast, the time complexity for search and insert operations in arrays and linked lists can be $O(n)$, where n is the number of elements in the array or list.
- There are multiple ways to traverse them.

A naive approach to exploring the tree would be to revisit the already traversed nodes. More specifically, we start at the root node and traverse a particular branch all the way to the leaf node. Then, we start at the root node again and explore another branch. In this way, we'll revisit many nodes over and over. This would take our time complexity to $O(n^2)$ in the worst case.

Tree depth-first search is another traversal method that explores the tree by traveling as far as possible along a branch before exploring the other branches. Because it visits each node of the tree exactly once, it guarantees a traversal in $O(n)$ time.

There are three main methods to traverse a tree with the tree depth-first search pattern—preorder, inorder, and postorder.

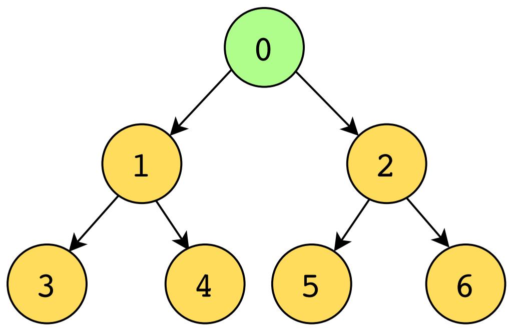
Preorder traversal

We start at the root node and visit it first. Then, we recursively traverse the left subtree, followed by the right subtree. This means we explore nodes in the order of root, left, and right. It's like starting at the top of a tree and moving down, exploring branches as we go. Here are the steps required to implement this algorithm:

1. **Traverse the left subtree:** We recursively traverse the left subtree, starting from the root node. We visit each node in the traversal. If the left child node doesn't exist, we move to step 2.
2. **Traverse the right subtree:** We repeat step 1 starting from the right child node of the current node. If the right child node doesn't exist, we visit the current node.
3. We repeat steps 1–2 until all nodes in the tree have been visited.

The following illustration shows an example of preorder traversal:

Preorder

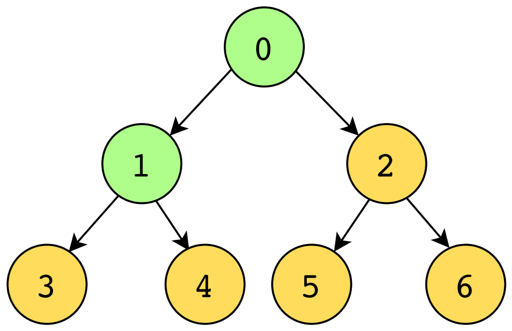


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 0

1 of 11

Preorder

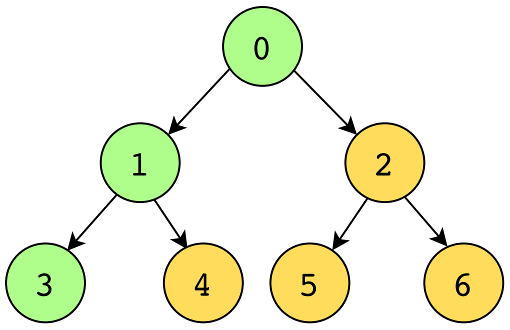


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 0, 1

2 of 11

Preorder

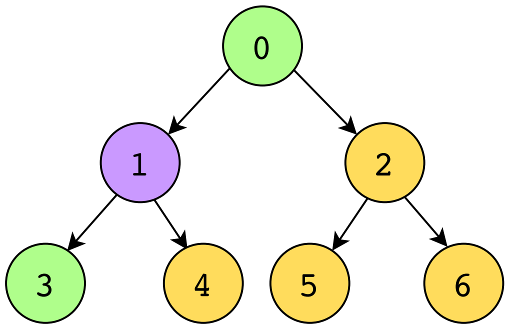


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 0, 1, 3

3 of 11

Preorder

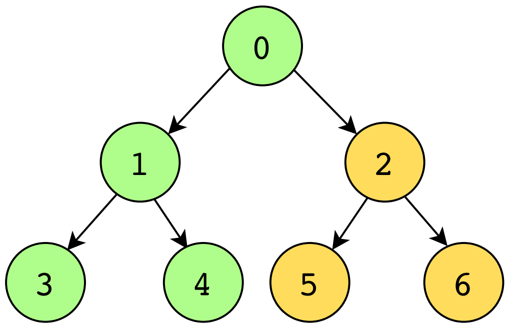


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 0, 1, 3

4 of 11

Preorder

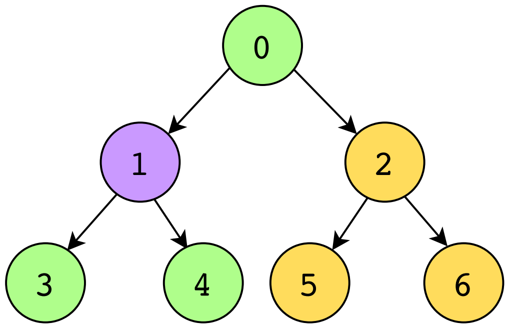


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 0, 1, 3, 4

5 of 11

Preorder



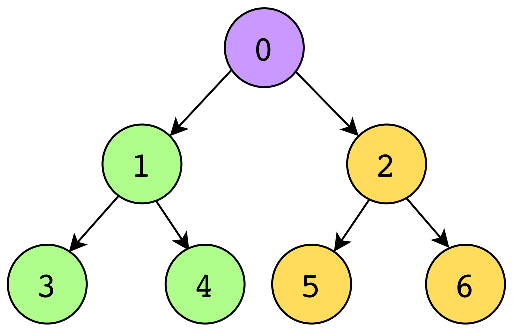
Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 0, 1, 3, 4

6 of 11

Preorder

Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

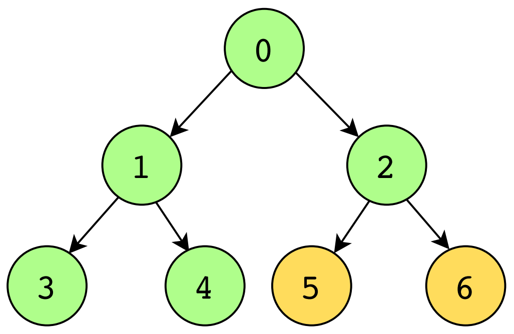


Output: 0, 1, 3, 4

7 of 11

Preorder

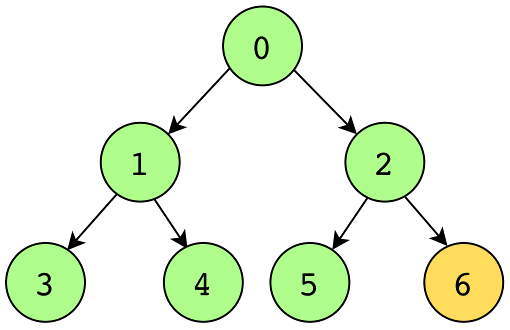
Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited



Output: 0, 1, 3, 4, 2

8 of 11

Preorder

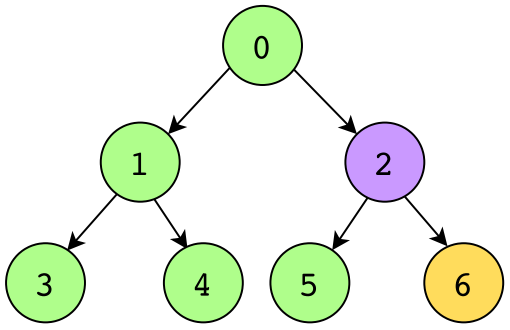


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 0, 1, 3, 4, 2, 5

9 of 11

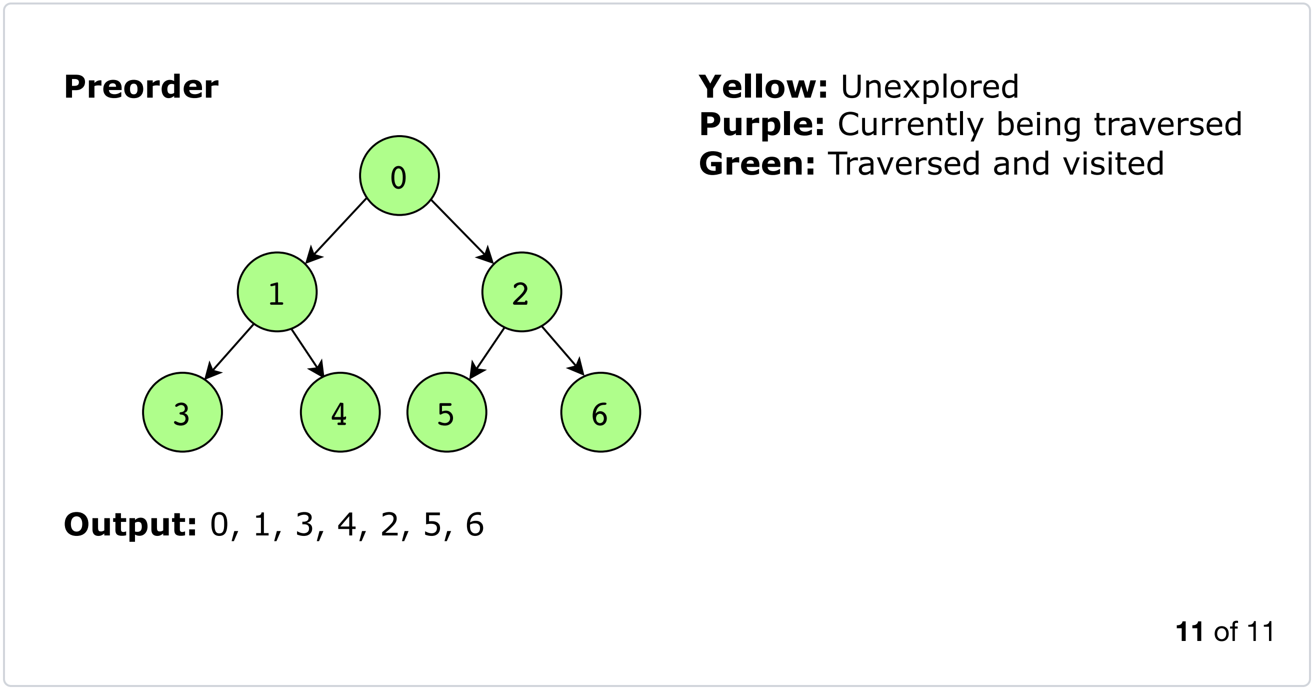
Preorder



Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 0, 1, 3, 4, 2, 5

10 of 11


Preorder
Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited
0
1
2
3
4
5
6
Output: 0, 1, 3, 4, 2, 5, 6
11 of 11

> **Note:** The meaning of *visit* depends on the application. If we are calculating the size in bytes of the directory structure on a disk, for example, this may involve accumulating the size of the files in a particular directory.

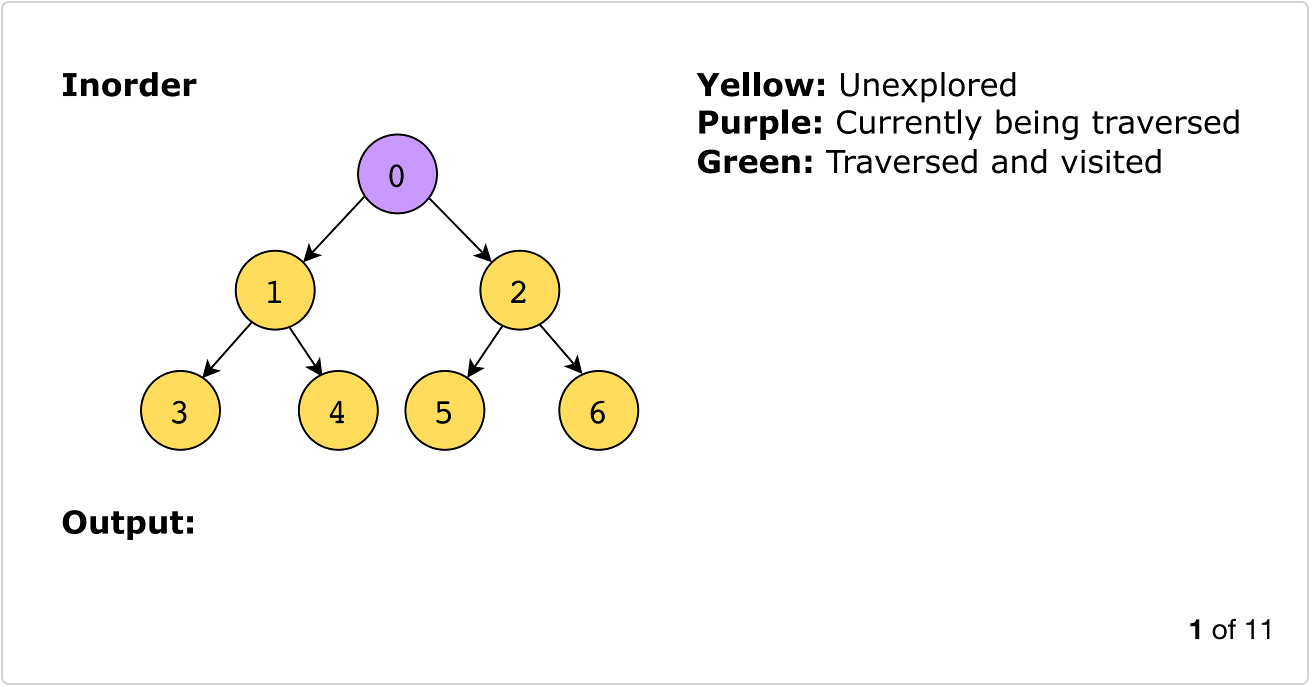
Inorder traversal

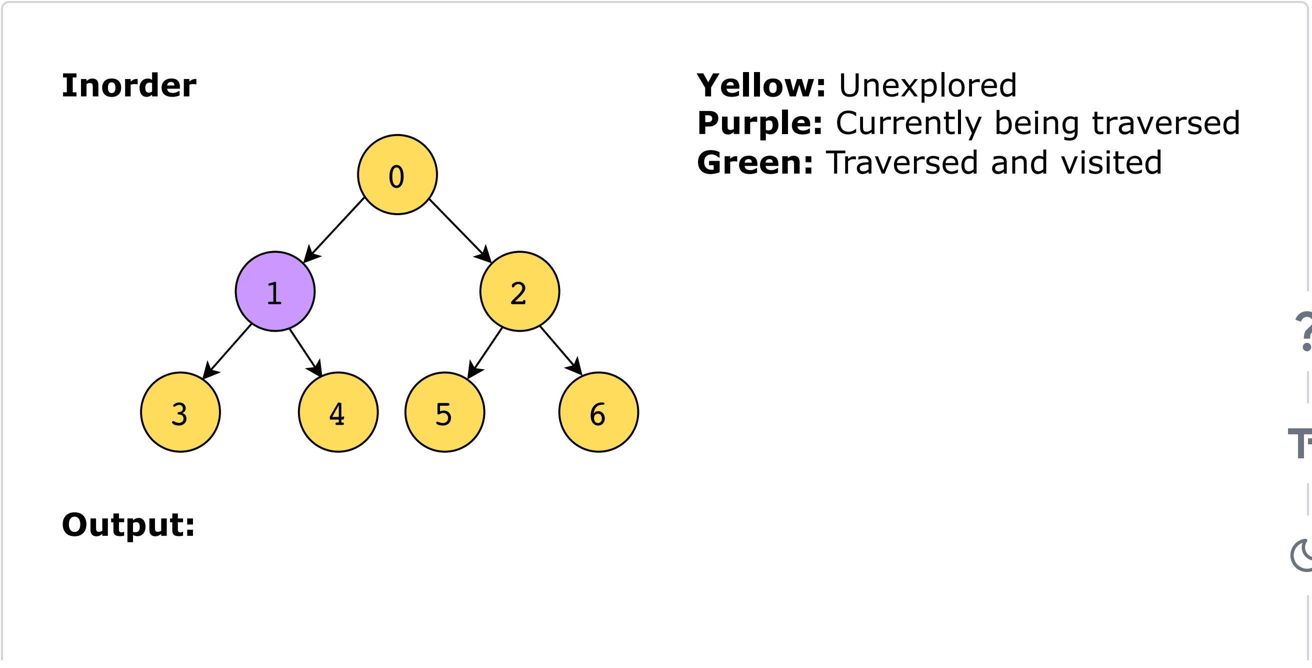
Starting from the root, we first traverse the left subtree, then visit the current node, and finally explore the right subtree. This order means we'll visit nodes in ascending order if the tree represents numbers. It's like exploring a tree from the bottom-left corner to the right, moving up gradually. Here are the steps required to implement this algorithm:

1. **Traverse the left subtree:** We recursively traverse the left subtree, starting from the root node. If the left child node doesn't exist, we visit the current node.

2. **Traverse the parent node:** After visiting the current node in step 1, we visit its parent node.
3. **Traverse the right subtree:** We repeat step 1 starting from the right child node. If the right child node doesn't exist, we visit the current node.
4. We repeat steps 1–3 until all nodes in the tree have been visited.

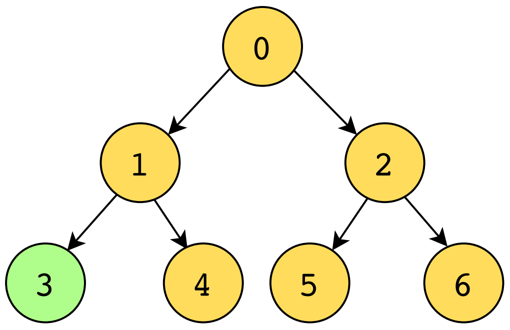
The following illustration shows an example of inorder traversal:


Inorder
Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited
0
1
2
3
4
5
6
Output:
1 of 11


Inorder
Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited
0
1
2
3
4
5
6
Output:

2 of 11

Inorder

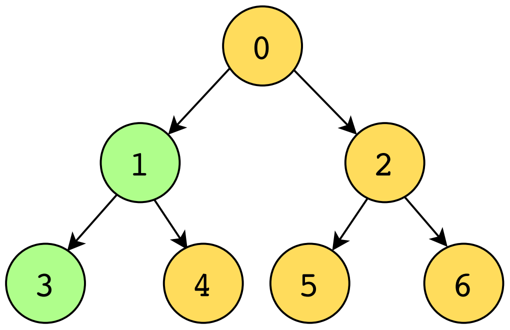


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3

3 of 11

Inorder

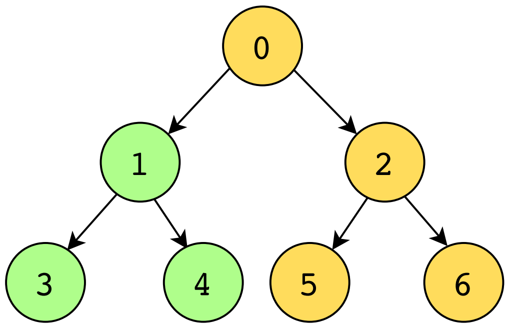


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 1

4 of 11

Inorder

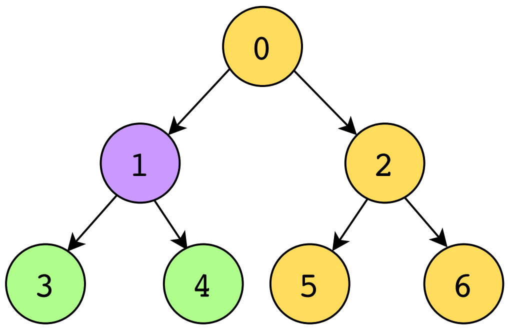


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 1, 4

5 of 11

Inorder

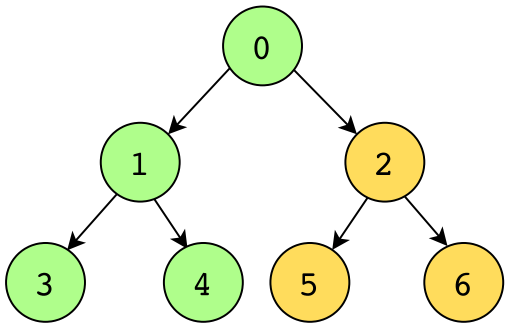


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 1, 4

6 of 11

Inorder

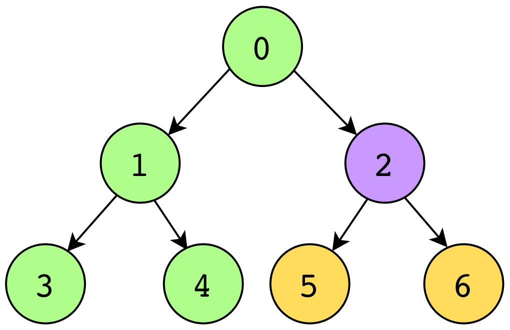


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 1, 4, 0

7 of 11

Inorder

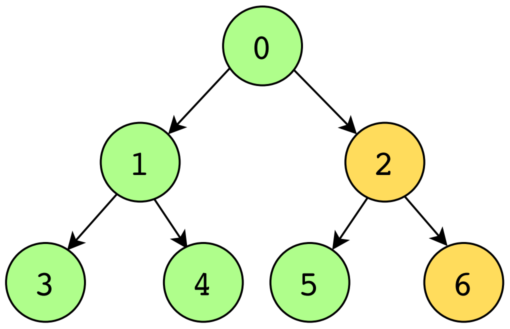


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 1, 4, 0

8 of 11

Inorder

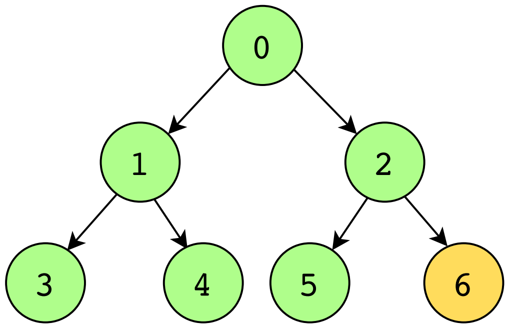


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 1, 4, 0, 5

9 of 11

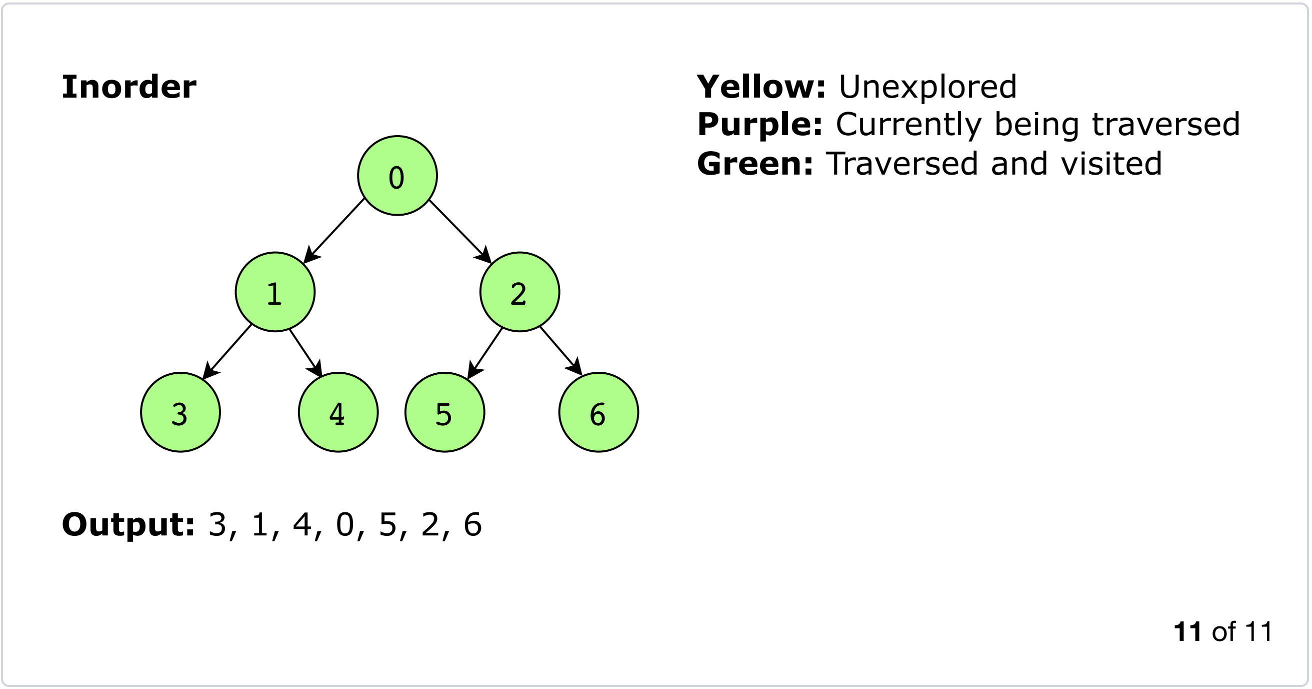
Inorder



Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 1, 4, 0, 5, 2

10 of 11

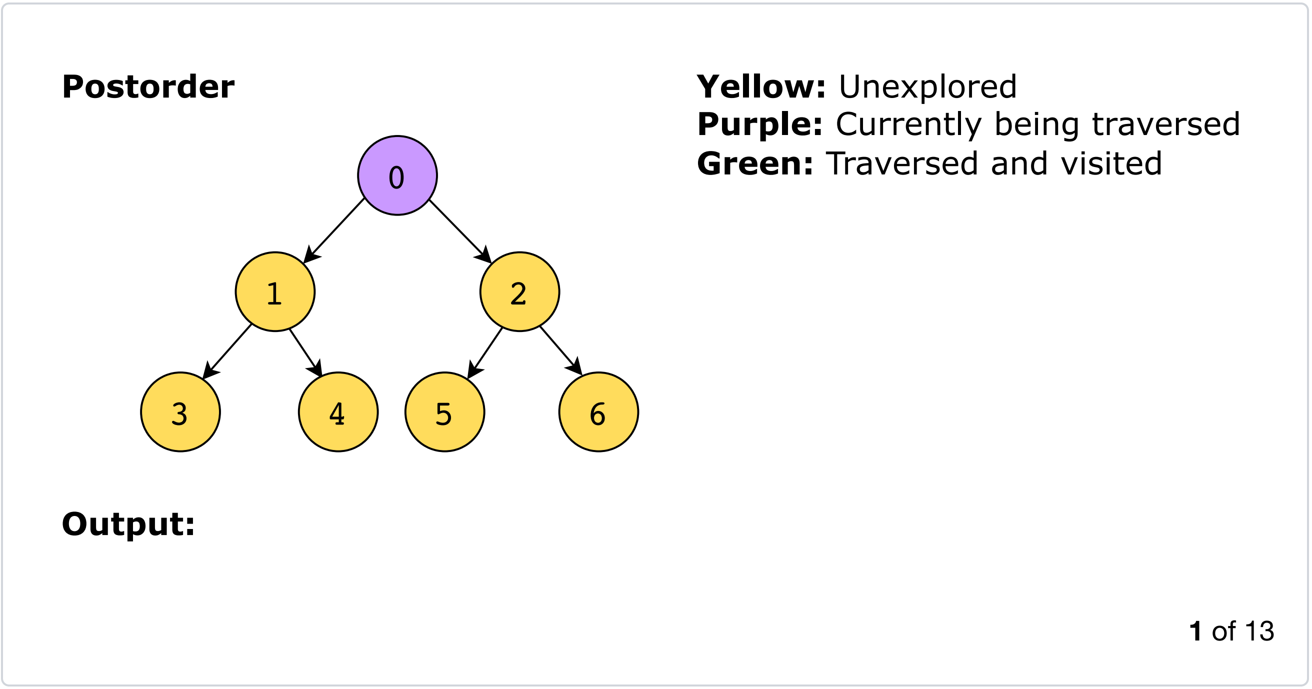

Inorder
Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited
0
1
2
3
4
5
6
Output: 3, 1, 4, 0, 5, 2, 6
11 of 11

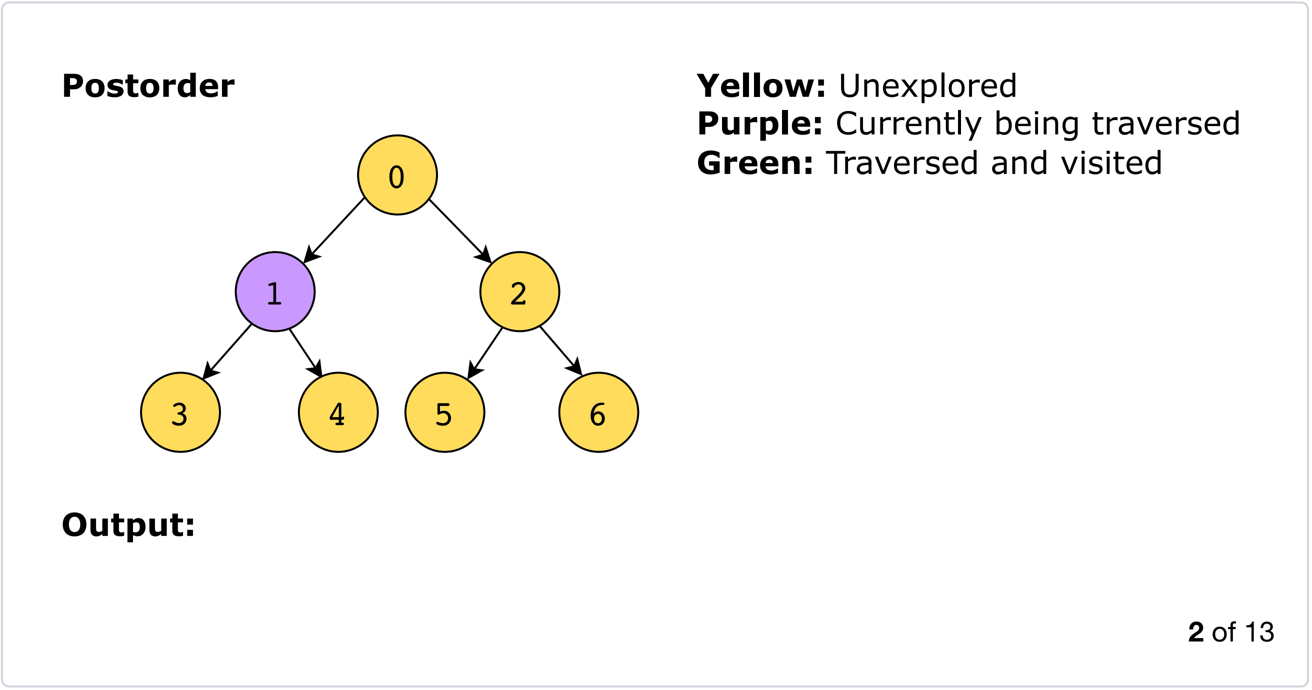
Postorder traversal

Starting from the root, we first traverse the left subtree, then the right subtree, and finally visit the current node. This order means we explore nodes from the bottom up, going from the leaves toward the root. It’s like examining a tree from its outermost branches inward, reaching the trunk last. Here are the steps required to implement this algorithm:

1. **Traverse the left subtree:** We recursively traverse the left subtree, starting from the root node. If the left child node doesn’t exist, we visit the current node and move to step 2.
2. **Traverse the right subtree:** We repeat step 1 starting from the right child node. If the right child node doesn’t exist, we visit the current node.
3. **Traverse the parent node:** After visiting the current node in step 2, we visit its parent node.
4. We repeat steps 1–3 until all nodes in the tree have been visited.

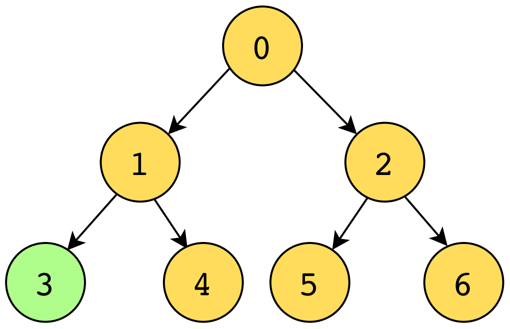
The following illustration shows an example of postorder traversal:


Postorder
0
1
2
3
4
5
6
Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited
Output:
1 of 13


Postorder
0
1
2
3
4
5
6
Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited
Output:
2 of 13

Postorder

Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

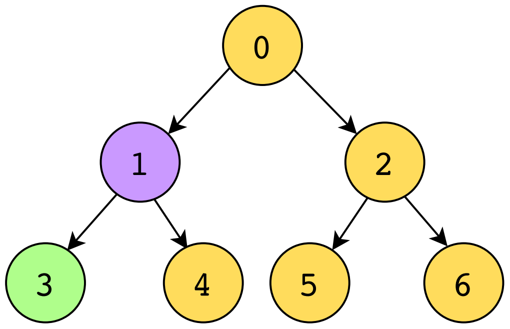


Output: 3

3 of 13

Postorder

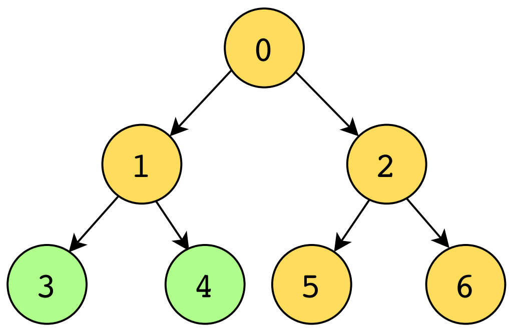
Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited



Output: 3

4 of 13

Postorder

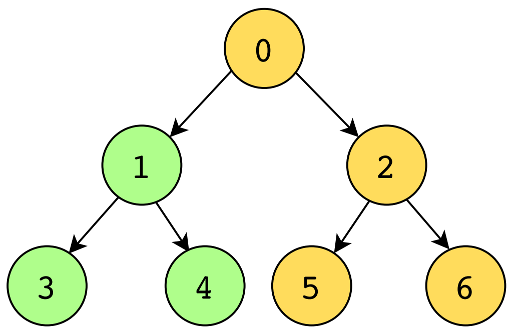


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 4

5 of 13

Postorder

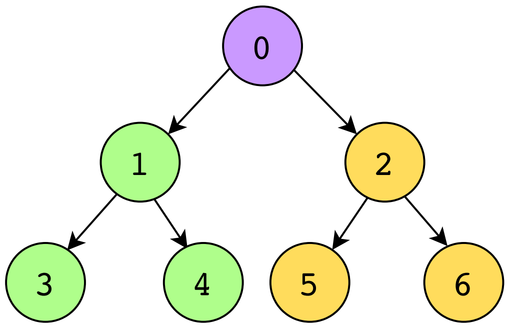


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 4, 1

6 of 13

Postorder

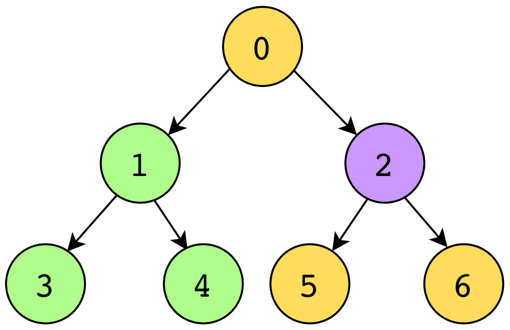


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 4, 1

7 of 13

Postorder

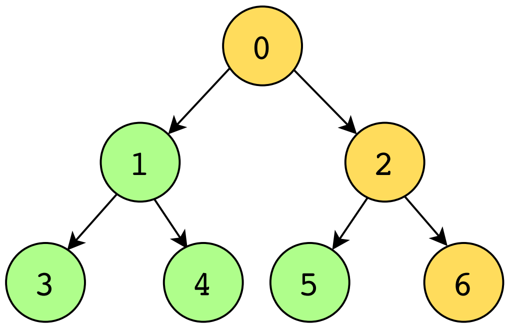


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 4, 1

8 of 13

Postorder

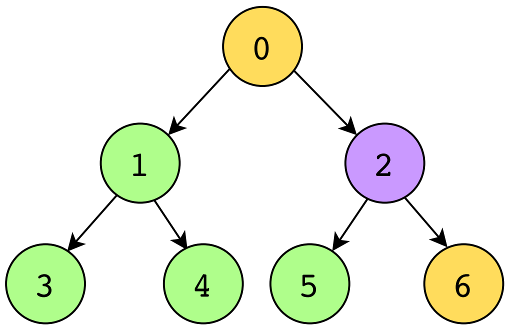


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 4, 1, 5

9 of 13

Postorder

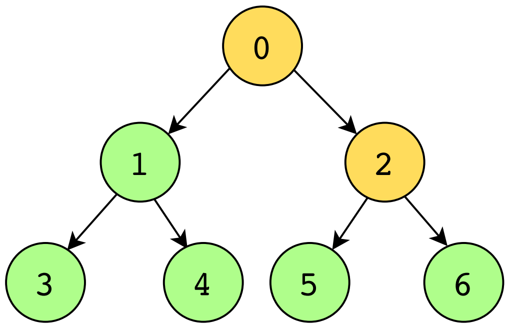


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 4, 1, 5

10 of 13

Postorder

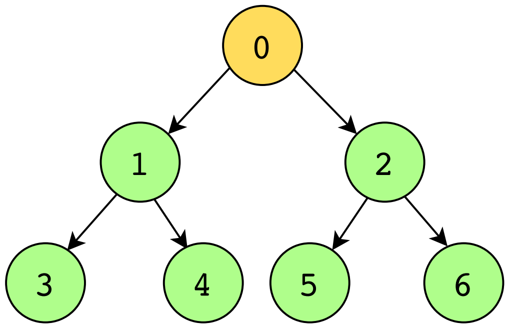


Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 4, 1, 5, 6

11 of 13

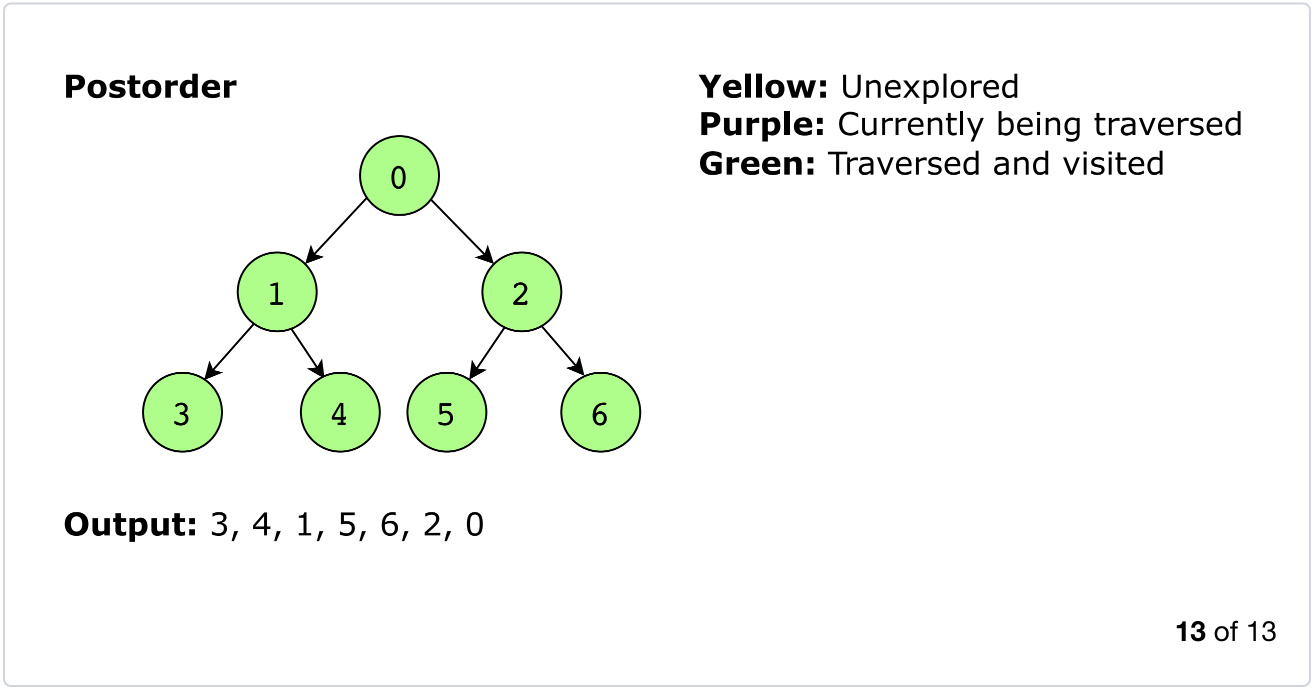
Postorder



Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited

Output: 3, 4, 1, 5, 6, 2

12 of 13


Postorder
Yellow: Unexplored
Purple: Currently being traversed
Green: Traversed and visited
0
1
2
3
4
5
6
Output: 3, 4, 1, 5, 6, 2, 0
13 of 13

Note: While preorder, inorder, and postorder traversals are commonly discussed in the context of binary trees, they can also be generalized to nonbinary trees with any number of children per node, also known as m-ary trees. In such trees, the traversal methods can still be applied by defining an order in which to visit nodes.

Examples

The following examples illustrate some problems that can be solved with this approach:

1. **Path sum:** Determine if a tree contains a root-to-leaf path, such that adding up all the values along the path equals the specified target sum.

Preorder traversal is used to explore each potential path from the root to a leaf node, summing up the node values along the way. Upon reaching a leaf