Ask a Question

Introduction to Tree Breadth-First Search

Let’s go over the Tree Breadth-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:

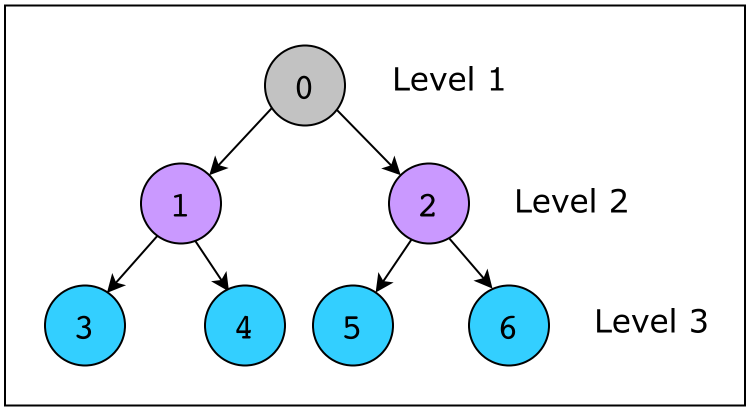
- They organize data in a hierarchical manner with a root node at the top and child nodes branching out from it.
- They are nonlinear, which means that the elements in a tree are not arranged sequentially but rather in a branching structure.

- The time complexity for the 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 the 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 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 breadth-first search (BFS) is another traversal method that explores the tree level by level, starting from the root node. Nodes that are directly connected to the root node by an edge are considered to be at level 1, nodes connected to those at level 1 are at level 2, and so forth.

Here's an example of the level-by-level representation of nodes in a tree:

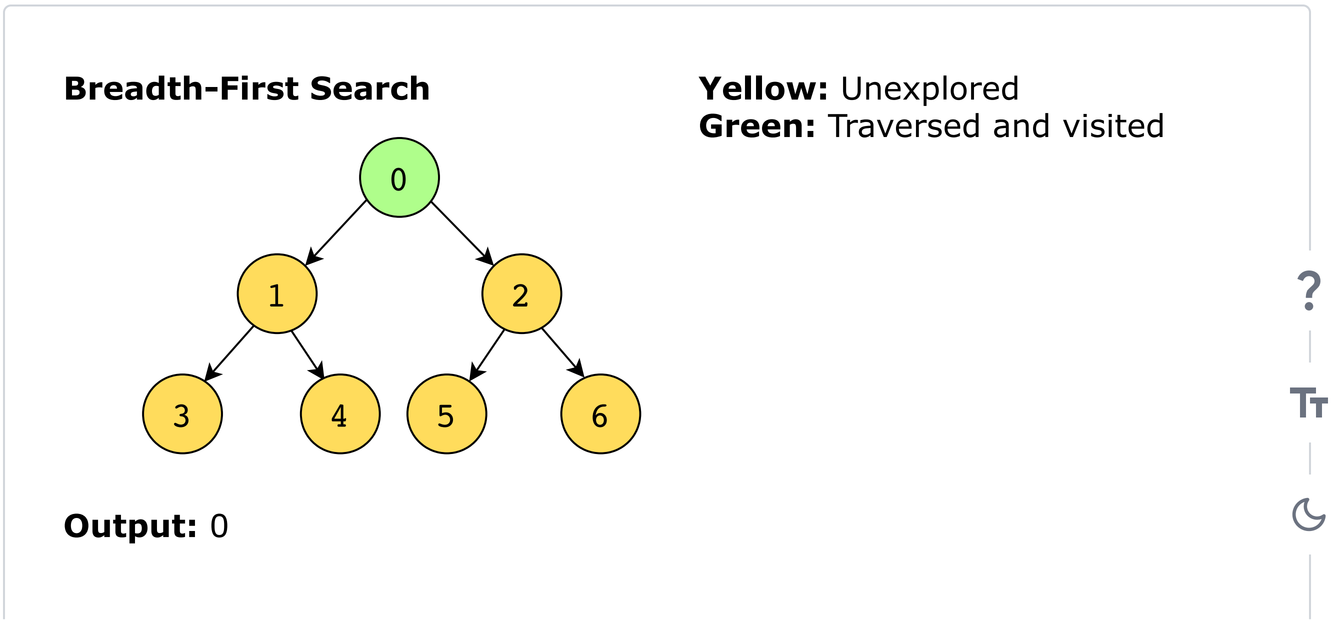


Because this traversal method visits each node of the tree exactly once guarantees a traversal in $O(n)$ time.

Here are the key features of tree breadth-first search:

1. **Initialization:** BFS begins its search from the root node of the tree, serving as the starting point.
2. **Exploring adjacent nodes:** At each step, BFS examines all nodes at the current level before moving on to nodes at the next level. It explores adjacent nodes, ensuring that nodes closer to the root are visited before deeper nodes.
3. **Traversal strategy:** BFS explores nodes layer by layer rather than depth-first. This ensures that all nodes at a particular level are visited before moving to the next level.
4. **Node discovery:** As BFS discovers new nodes, it processes them according to the search requirements. For instance, if BFS is searching for a specific node, it stops when the target node is found. Otherwise, it continues exploring nodes and enqueues them for further processing.
5. **Stopping condition:** BFS terminates when either the desired node is found (if searching for a specific node) or when all nodes have been explored.

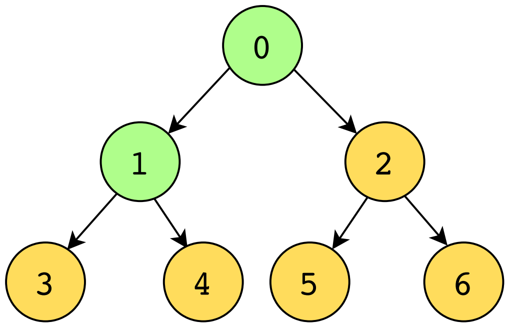
The following illustration shows an example of how breadth-first search traverses the nodes of a tree:



1 of 7

Breadth-First Search

Yellow: Unexplored
Green: Traversed and visited

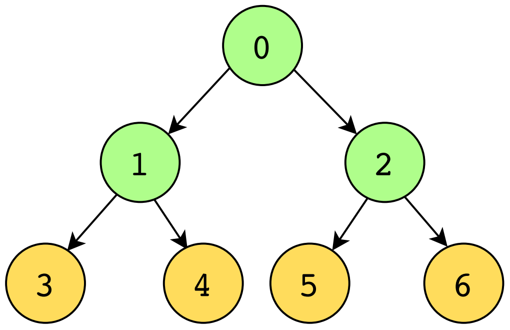


Output: 0, 1

2 of 7

Breadth-First Search

Yellow: Unexplored
Green: Traversed and visited

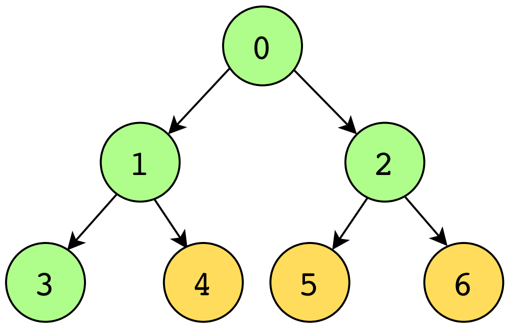


Output: 0, 1, 2

3 of 7

Breadth-First Search

Yellow: Unexplored
Green: Traversed and visited

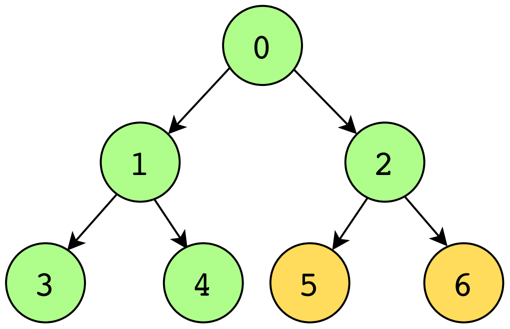


Output: 0, 1, 2, 3

4 of 7

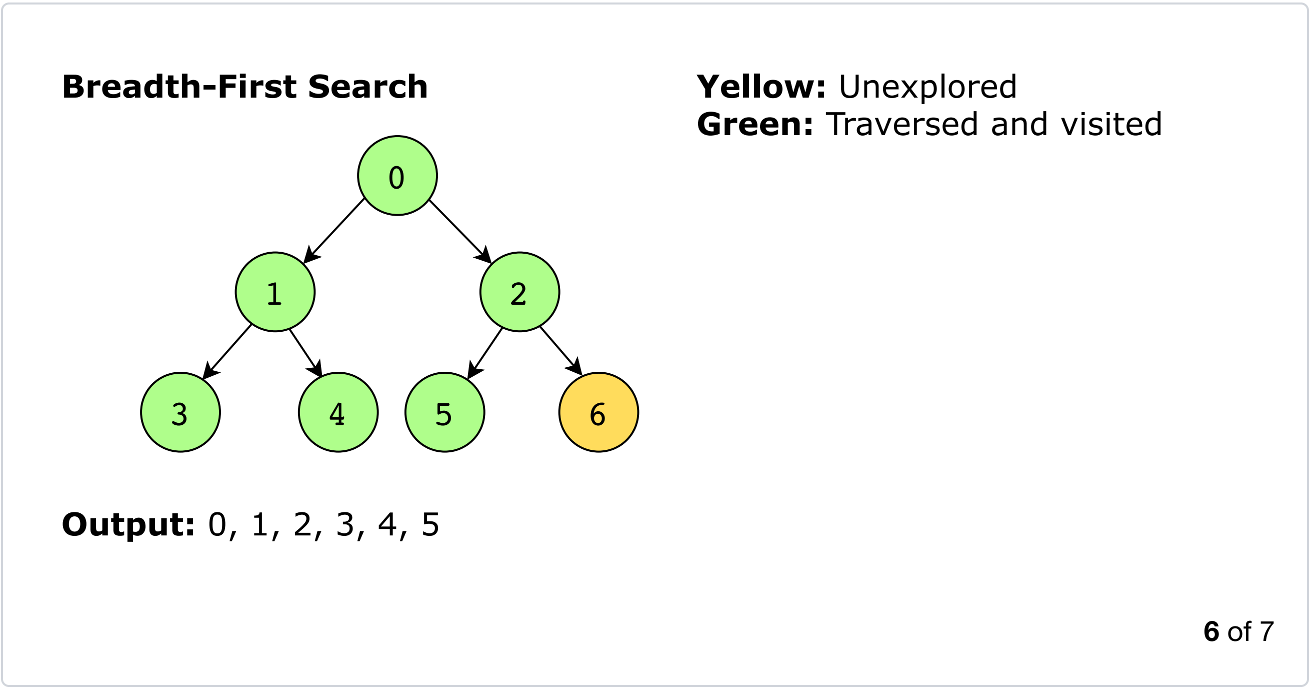
Breadth-First Search

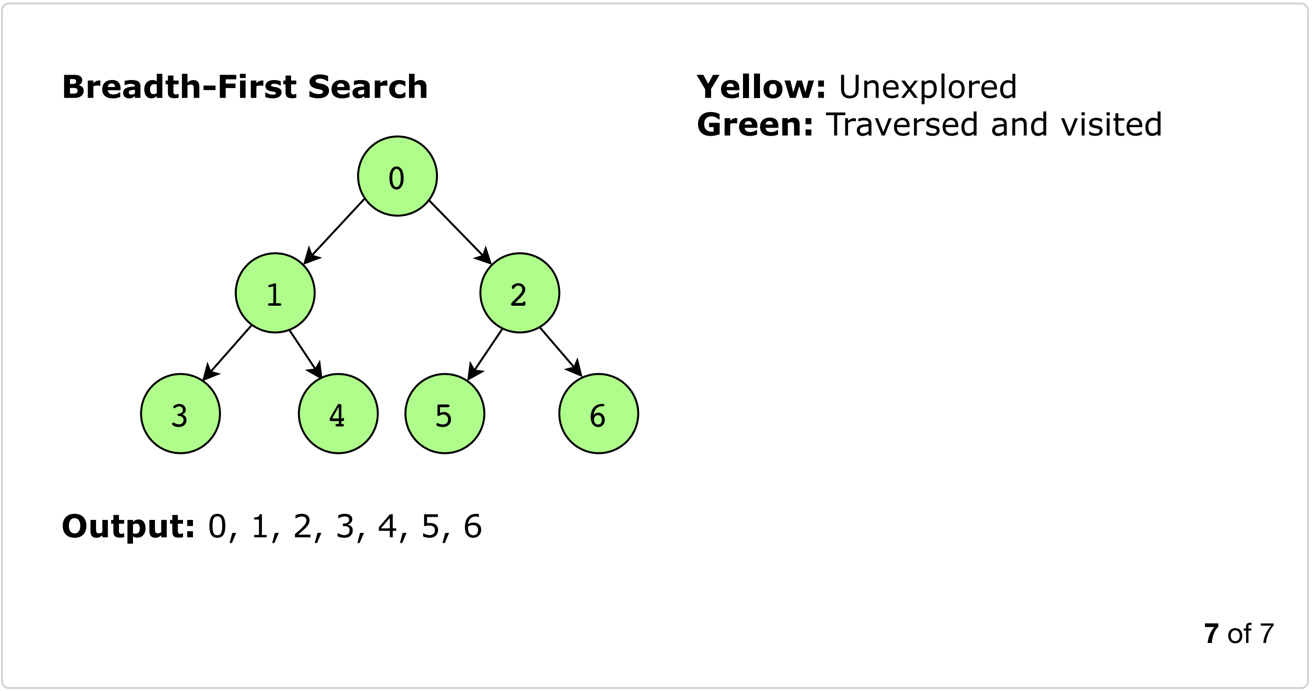
Yellow: Unexplored
Green: Traversed and visited



Output: 0, 1, 2, 3, 4

5 of 7





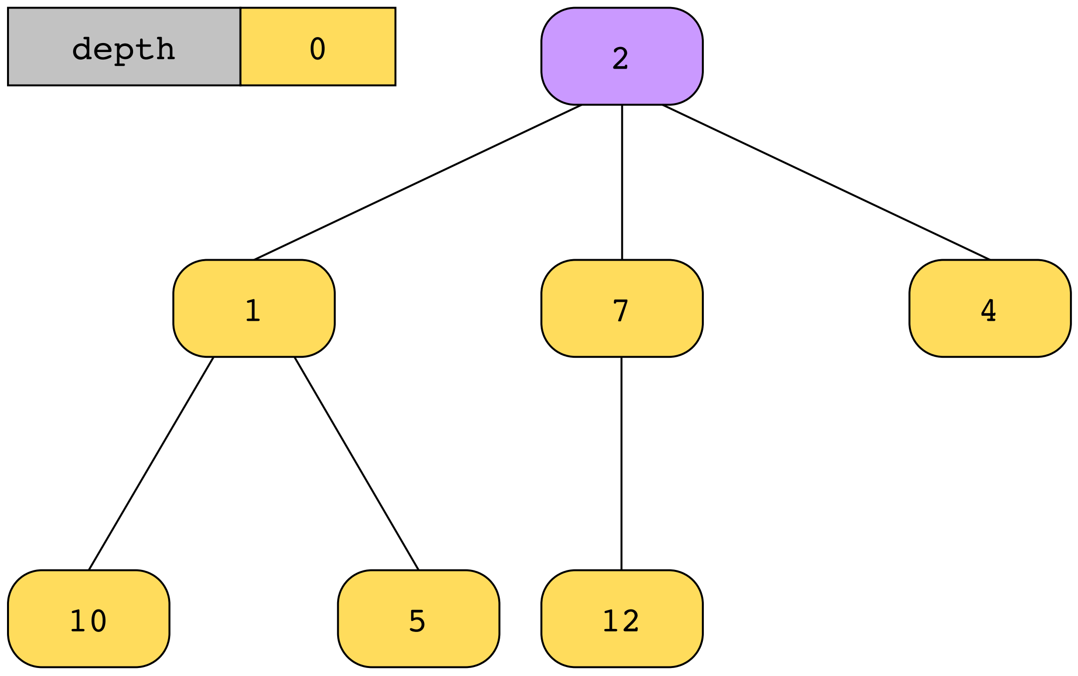
Note: In BFS, it's important to understand that traversal within a level doesn't strictly adhere to a left-to-right direction. While the algorithm processes nodes level by level, it doesn't enforce a specific order within each level. This means that BFS may visit nodes within the same level in any order, depending on the implementation of the algorithm.

Examples

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

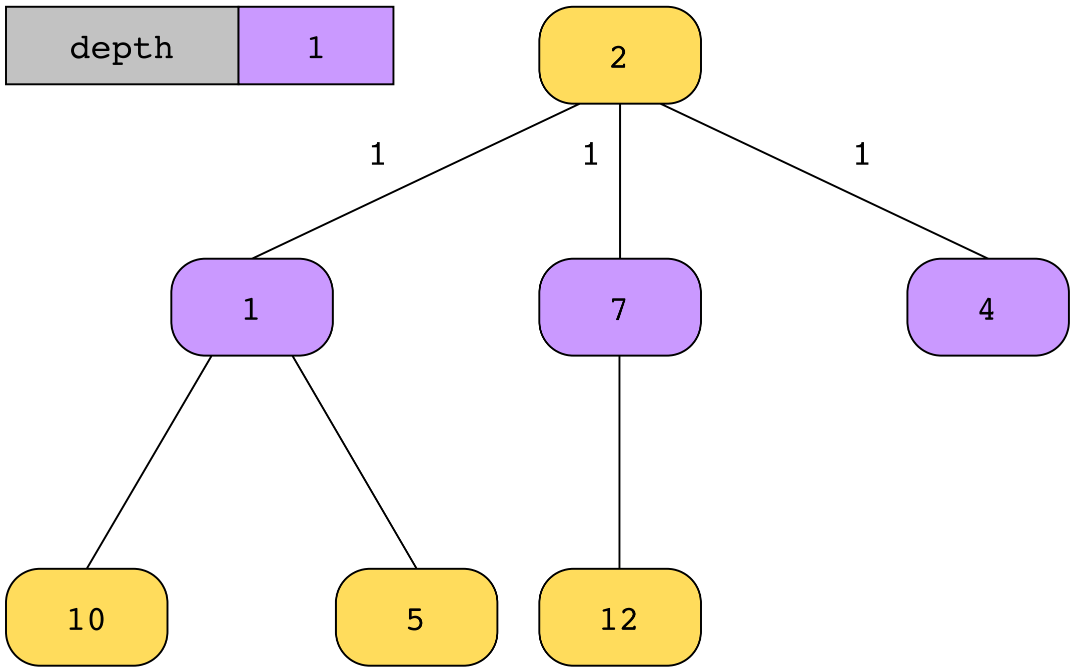
1. **Minimum depth of a tree:** Find the shortest distance from the root node to the nearest leaf node.

We will start the BFS traversal from the root node. We keep track of the depth as we traverse down the tree.



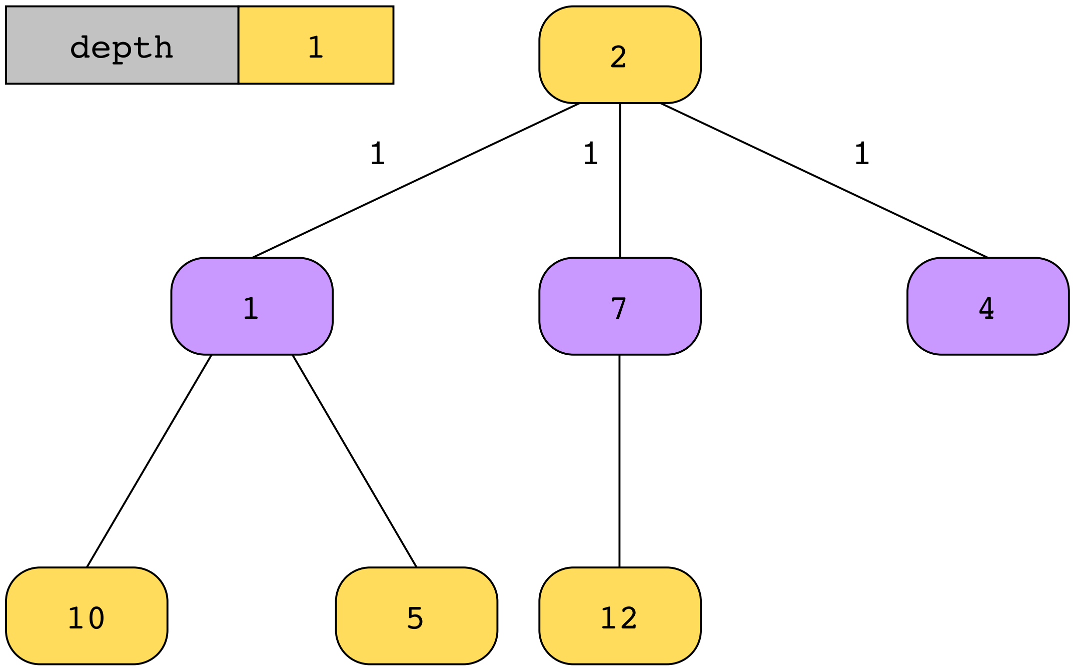
1 of 4

At each level, we will iterate over all the children of the current node, incrementing the depth.



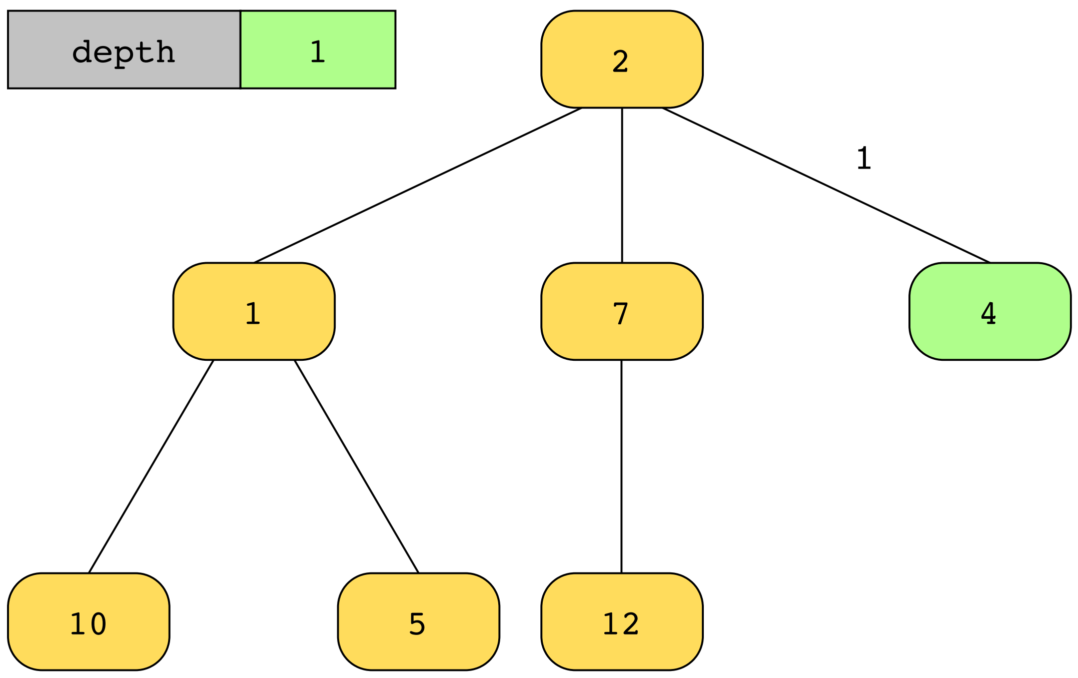
2 of 4

Next, we check if the current level of nodes contain a leaf node.



3 of 4

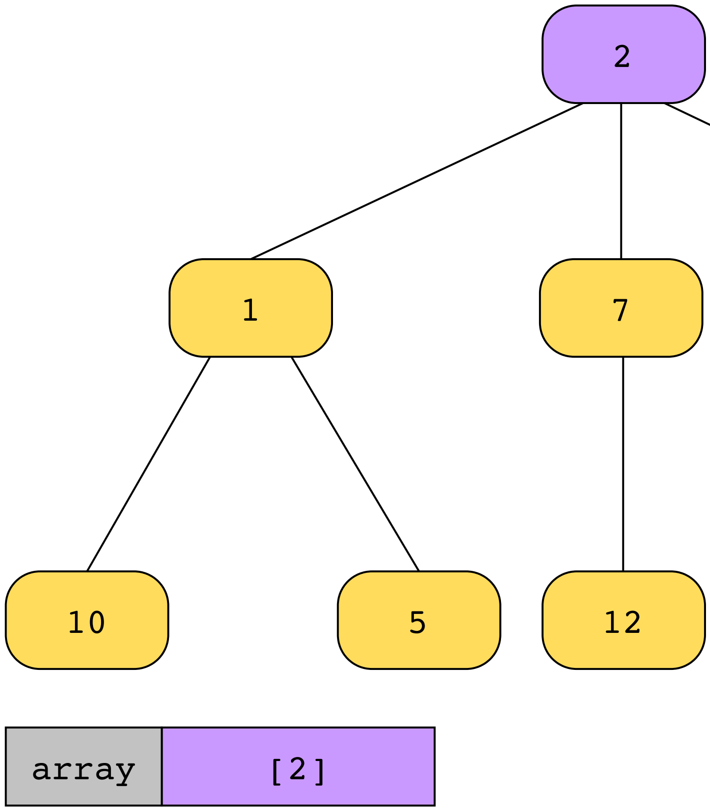
Because node 4 is a leaf node, we will not traverse the tree further because we have found the minimum depth from the root to node 4.



4 of 4

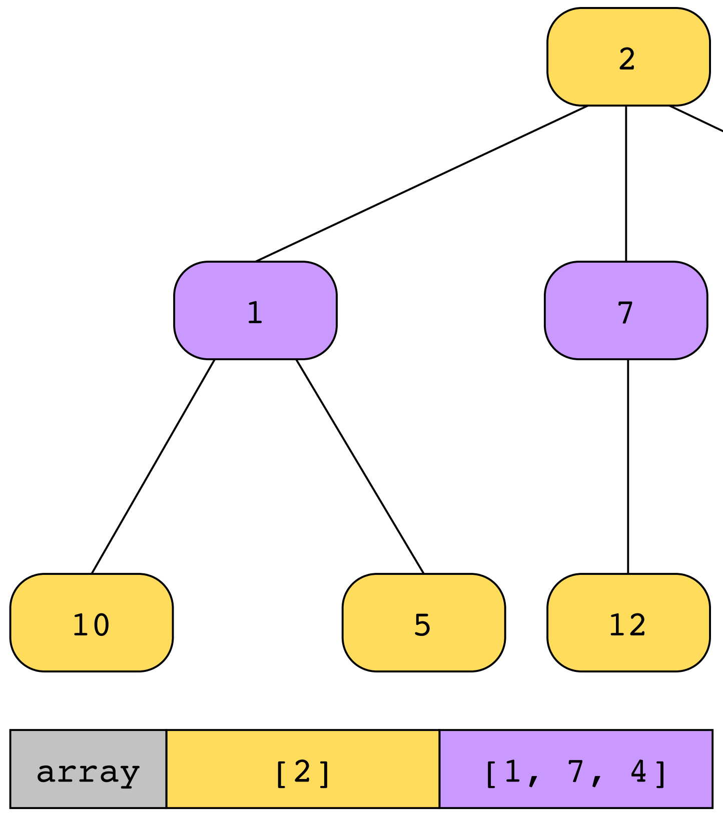
2. **Bottom-up level order traversal:** Given a tree, return the bottom-up, left-to-right level order traversal of its nodes' values.

We will start the BFS traversal from the root node. We add the list of current nodes in each level to an array.



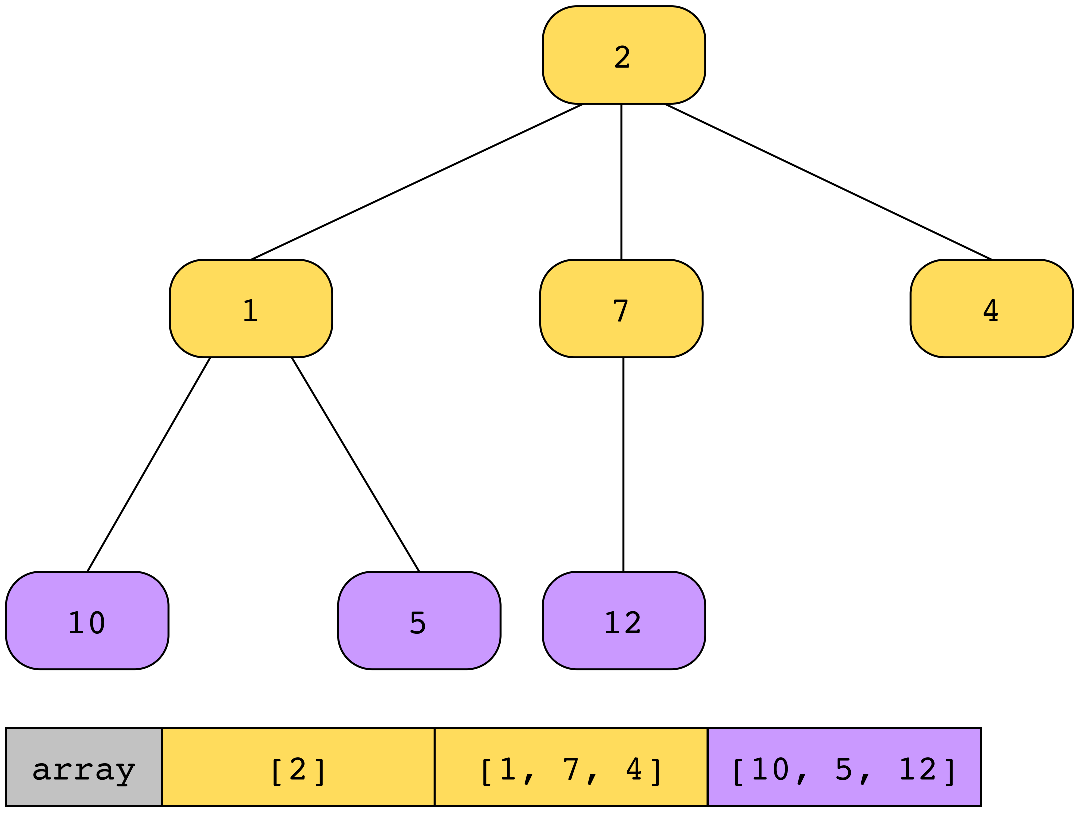
1 of 5

We will start the BFS traversal from the root node. We add the current nodes in each level to an array.



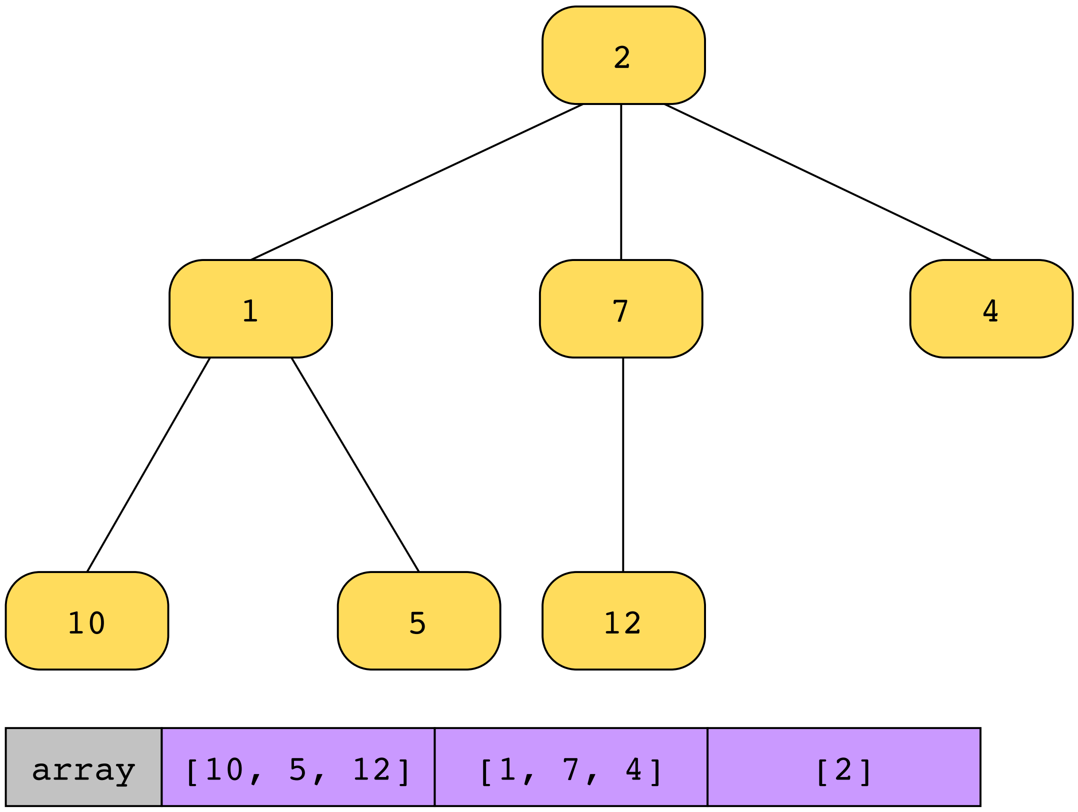
2 of 5

We will start the BFS traversal from the root node. We add the current nodes in each level to an array.



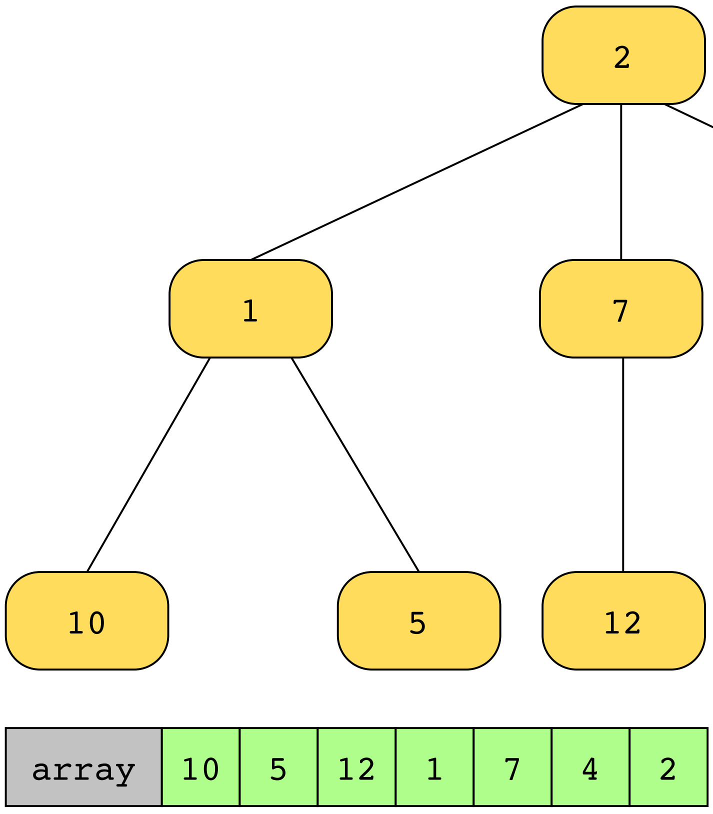
3 of 5

After all the nodes have been traversed, we reverse the array, so that it contains the nodes in the desired bottom-up order.



4 of 5

We flatten the array and return it.



5 of 5

Does your problem match this pattern?

Yes, if either of these conditions is fulfilled:

- **Tree data structure:** The input data is in the form of a tree, or the cost of transforming it into a tree is low.
- **Not a wide tree:** If the tree being searched is very wide. In such scenarios, the time complexity of breadth-first search may become prohibitive.

- **Level-by-level traversal:** The solution dictates traversing the tree one level at a time, for example, to find the level order traversal of the nodes of a tree or a variant of this ordering.
- **Solution near the root:** We have reason to believe that the solution is near the root of the tree. Instead, if the solution is near the leaves of the tree, BFS may not be efficient because it exhaustively explores nodes level by level. In such cases, depth-first search (DFS) would be more suitable for traversing deep into the tree.

Real-world problems

Many problems in the real world use the tree breadth-first search pattern. Let’s look at some examples.

- **File system analysis:** In file system analysis, the directory structure is commonly represented as a tree data structure. Each directory is a node in the tree, and each file is a leaf node. The tree’s root represents the starting directory. BFS can be used to traverse this tree, helping to analyze file dependencies or find the shortest path to a specific file.
- **Version control systems:** BFS can be utilized in version control systems (e.g., Git) to traverse the file system tree and identify changes made to files and directories. This information is crucial for tracking revisions, managing branches, and merging changes.
- **Genealogy and evolutionary trees:** In biology, BFS can be used to analyze genealogy and evolutionary trees. By starting from a specific organism or species and traversing ancestral connections in a breadth-first manner, BFS can help reconstruct evolutionary histories, analyze genetic relationships or study species diversification patterns.