# Use Breadth First Search in a Binary Search Tree

Here we will introduce another tree traversal method: breadth-first search. In contrast to the depth-first search methods from the last challenge, breadth-first search explores all the nodes in a given level within a tree before continuing on to the next level. Typically, queues are utilized as helper data structures in the design of breadth-first search algorithms.

In this method, we start by adding the root node to a queue. Then we begin a loop where we dequeue the first item in the queue, add it to a new array, and then inspect both its child subtrees. If its children are not null, they are each enqueued. This process continues until the queue is empty.

Let's create a breadth-first search method in our tree called `levelOrder`. This method should return an array containing the values of all the tree nodes, explored in a breadth-first manner. Be sure to return the values in the array, not the nodes themselves. A level should be traversed from left to right. Next, let's write a similar method called `reverseLevelOrder` which performs the same search but in the reverse direction (right to left) at each level.

---

| | |
|---|---|
| | The `BinarySearchTree` data structure should exist. |
| | The binary search tree should have a method called `levelOrder`. |
| | The binary search tree should have a method called `reverseLevelOrder`. |
| | The binary search tree should have a method called `postorder`. |
| | The `levelOrder` method should return an array of the tree node values explored in level order. |

| | |
|---|---|
| | The `reverseLevelOrder` method should return an array of the tree node values explored in reverse level order. |
| | The `levelOrder` method should return `null` for an empty tree |
| | The `reverseLevelOrder` method should return `null` for an empty tree |