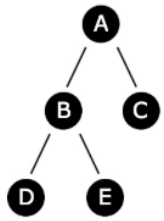
Tree traversal is the process of visiting each node in a tree, such as a binary tree or binary search tree, exactly once. There are several effective traversal algorithms which we will cover below.

All of the algorithms below will implement Node objects we create, which were covered in a previous algorithm on linked lists. Although, we will be slightly changing the code for the nodes. The tree we will be operating on looks like the following:



And we can assume the tree is properly constructed via the following code which sets up nodes and links them to their proper child nodes:

```
function Node(data) {
  this.data = data;
  this.left = null;
  this.right = null;
}

// create nodes
var root = new Node('A');
var n1 = new Node('B');
var n2 = new Node('C');
var n3 = new Node('D');
var n4 = new Node('E');

// setup children
root.left = n1;
root.right = n2;
n1.left = n3;
n1.right = n4;
```

Level-order

A level-order traversal on a tree performs the following steps starting from the root:

> 1) Add the root to a queue.
> 2) Pop the last node from the queue, and return its value.
> 3) Add all children of popped node to queue, and continue from step 2 until queue is empty.

For the tree above, performing a level-order traversal would output the node values in the following order:

A, B, C, D, E

```
function level_order(root, nodes) {
    var queue = [root];
    while (queue.length > 0) {
        // front of queue is at element 0 and we push elements to back of queue
        var n = queue.shift();
        nodes.push(n.data);
        if (n.left !== null) { queue.push(n.left); }
        if (n.right !== null) { queue.push(n.right); }
    }
    return nodes;
}

level_order(root, []); // => [ A, B, C, D, E ]
```