教程一：[js数据结构和算法 二叉树](https://segmentfault.com/a/1190000000740261)

二叉树

二叉树中每一个节点都是一个对象，每一个数据节点都有三个指针，分别是指向父母、左孩子和右孩子的指针。每一个节点都是通过指针相互连接的。相连指针的关系都是父子关系。

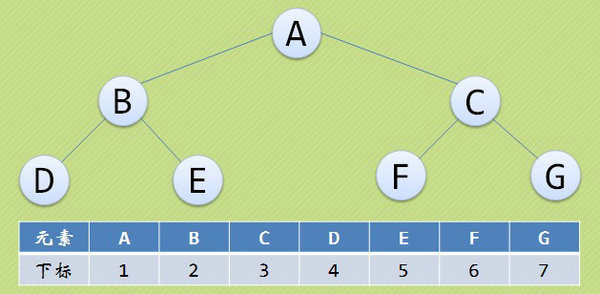
二叉树的性质

二叉树的性质一：在二叉树的第i层上至多有2^(i-1)个结点(i>=1)

二叉树的性质二：深度为k的二叉树至多有2^k-1个结点(k>=1)

二叉树的顺序存储结构

二叉树的顺序存储结构就是用一维数组存储二叉树中的各个结点，并且结点的存储位置能体现结点之间的逻辑关系。



## 二叉链表

二叉树的存储按照国际惯例来说一般也是采用链式存储结构的。

二叉树每个结点最多有两个孩子，所以为它设计一个数据域和两个指针域是比较自然的想法，我们称这样的链表叫做二叉链表。



二叉树的遍历

二叉树的遍历(traversing binary tree)是指从根结点出发，按照某种次序依次访问二叉树中所有结点，使得每个结点被访问一次且仅被访问一次。

二叉树的遍历有三种方式，如下：

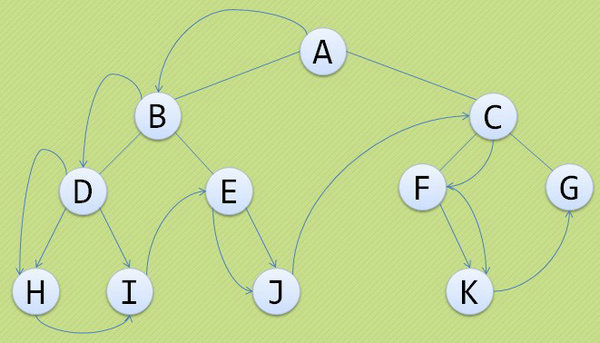
（1）前序遍历（DLR），首先访问根结点，然后遍历左子树，最后遍历右子树。简记根-左-右。

（2）中序遍历（LDR），首先遍历左子树，然后访问根结点，最后遍历右子树。简记左-根-右。

（3）后序遍历（LRD），首先遍历左子树，然后遍历右子树，最后访问根结点。简记左-右-根。

前序遍历：

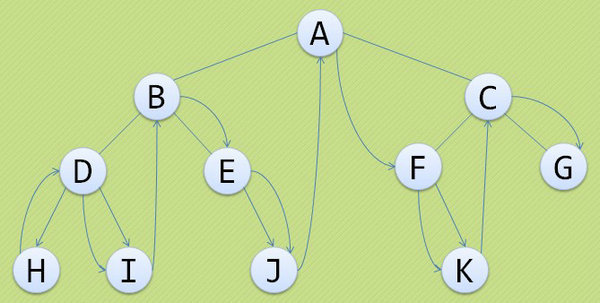
若二叉树为空，则空操作返回，否则先访问根结点，然后前序遍历左子树，再前序遍历右子树。



遍历的顺序为：A B D H I E J C F K G

中序遍历：

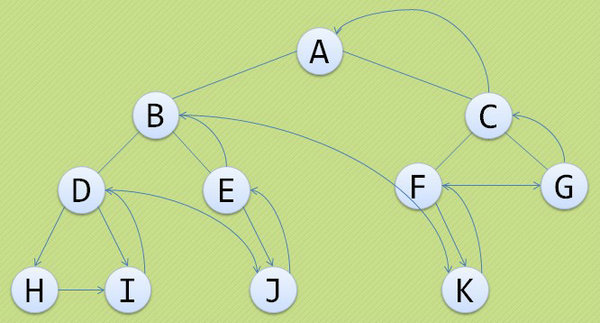
若树为空，则空操作返回，否则从根结点开始（注意并不是先访问根结点），中序遍历根结点的左子树，然后是访问根结点，最后中序遍历右子树。



遍历的顺序为：H D I B E J A F K C G

后序遍历：

若树为空，则空操作返回，否则从左到右先叶子后结点的方式遍历访问左右子树，最后访问根结点。



遍历的顺序为：H I D J E B K F G C A

教程二：[Data Structures With JavaScript: Tree](http://code.tutsplus.com/articles/data-structures-with-javascript-tree--cms-23393)

 two different methods of tree traversal: Depth-First Search (DFS) and Breadth-First Search (BFS)

#### Node

* data stores a value.
* parent points to a node's parent.
* children points to the next node in the list.

#### Tree

* \_root points to the root node of a tree.
* traverseDF(callback) traverses nodes of a tree with DFS.
* traverseBF(callback) traverses nodes of a tree with BFS.
* contains(data, traversal) searches for a node in a tree.
* add(data, toData, traverse) adds a node to a tree.
* remove(child, parent) removes a node in a tree.

#### Properties of a Node

For our implementation, we will first define a function named Node and then a constructor named Tree.

|  |
| --- |
| * function Node(data) { * this.data = data; * this.parent = null; * this.children = []; * } |

Every instance of Node contains three properties: data, parent, and children. The first property holds data associated with a node. The second property points to one node. The third property points to many children nodes.

#### Properties of a Tree

Now, let's define our constructor for Tree, which includes the Node constructor in its definition:

|  |  |
| --- | --- |
| 1  2  3  4 | function Tree(data) {      var node = new Node(data);      this.\_root = node;  } |

Tree contains two lines of code. The first line creates a new instance of Node; the second line assigns node as the root of a tree.

var tree = new Tree('CEO');

// {data: 'CEO', parent: null, children: []}

tree.\_root;

#### Methods of a Tree

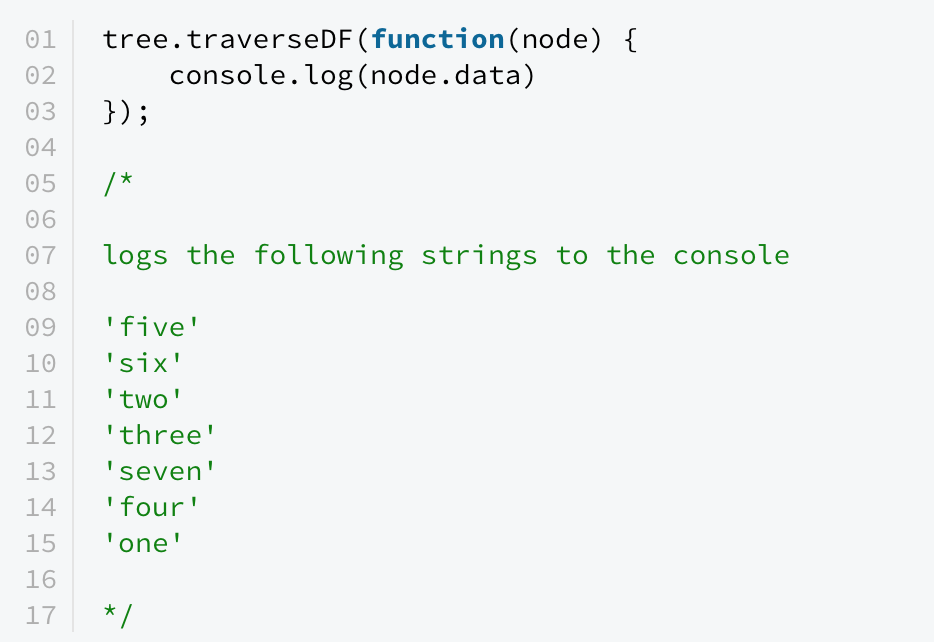
Next, we will create the following five methods:

**Tree**

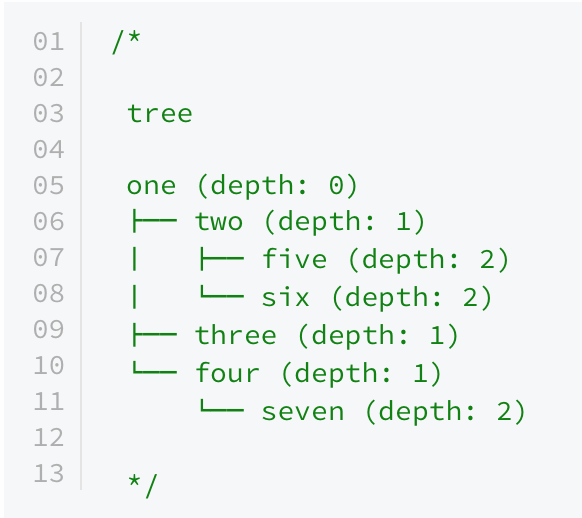
1. traverseDF(callback)
2. traverseBF(callback)
3. contains(data, traversal)
4. add(child, parent)
5. remove(node, parent)

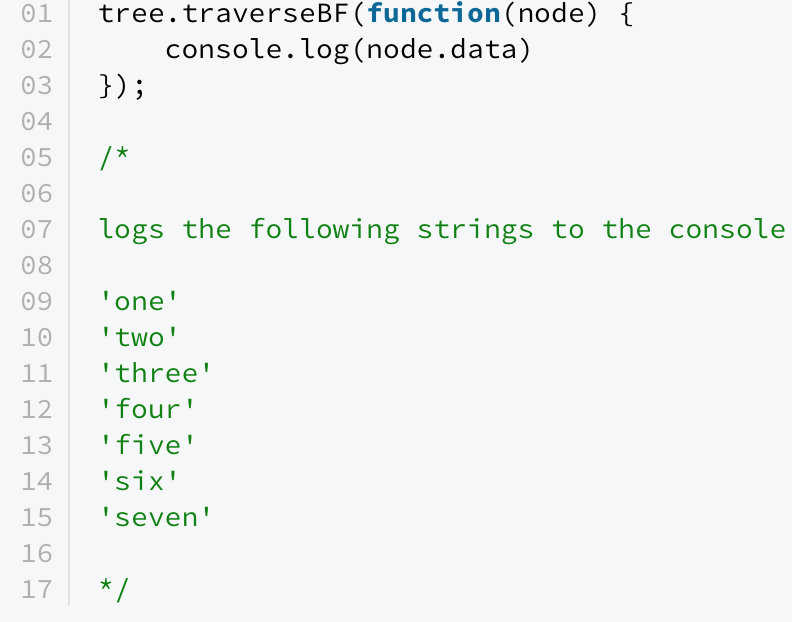
**1 of 5: traverseDF(callback)**





**2 of 5: traverseBF(callback)**

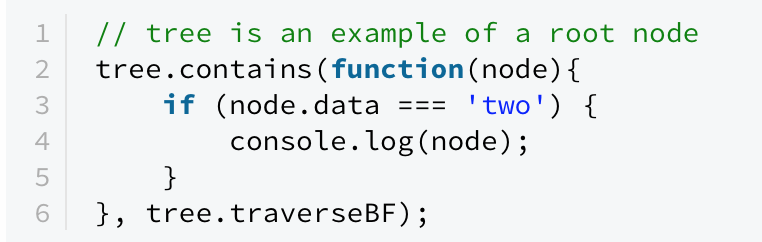




**3 of 5: contains(callback, traversal)**

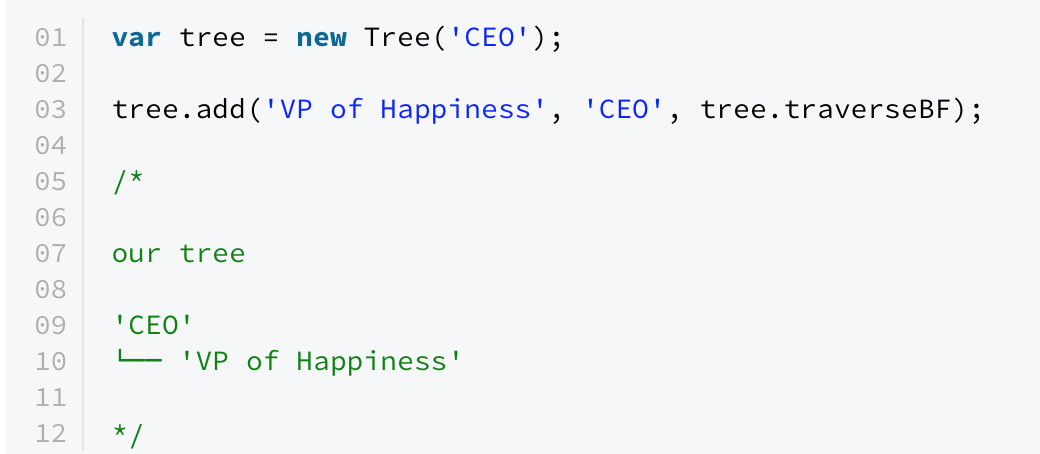
a method that will allow us to search for a particular value in our tree

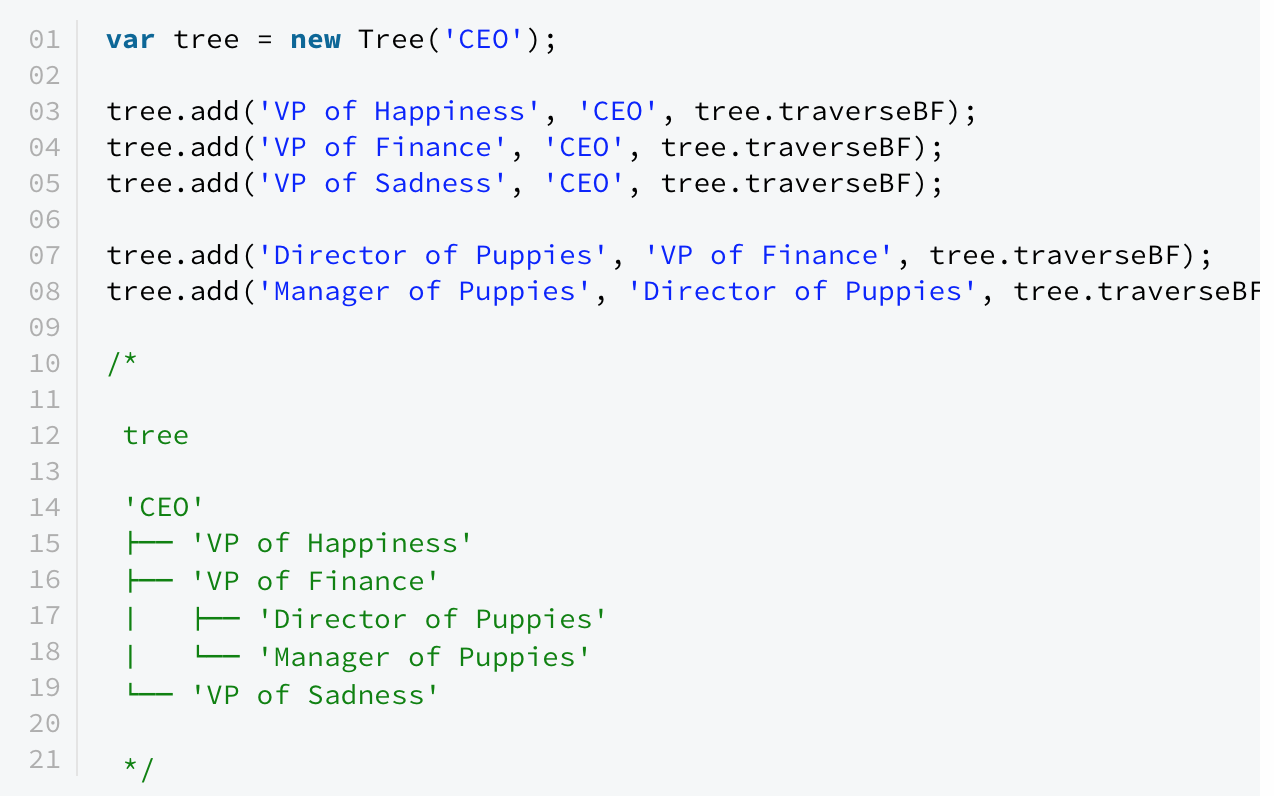
two arguments: the data to search and the type of traversal.



**4 of 5: add(data, toData, traversal)**

 a method that will enable us to add a node to a specific node.





**5 of 5: remove(data, fromData, traversal)**

this method will remove a node and all of its children

# JavaScript prototype 属性

prototype 属性使您有能力向对象添加属性和方法。

语法：object.prototype.name=value



教程三：Computer science in javascript: Binary search tree

Binary search tree，BST，二叉搜索树

The unique part of a binary search tree, however, is that the nodes are ordered based on the value they contain: any values that are part of a node’s left subtree are always less than the node’s value and any values in the right subtree are always greater than the node’s value.

某个节点的左子树中的所有value一定比这个节点小，其右子树节点的所有value一定比这个节点大。