codecademy

Learn Trees

Wide and deep trees

There are two ways to describe the shape of a tree. Trees can be *wide*, meaning that each node has many children. And trees can be *deep*, meaning that there are many parent-child connections with few siblings per node. Trees can be both *wide* and *deep* at the same time.

Binary search tree

In a *binary search tree*, parent nodes can have a maximum of two children. These children are called the “left child” and the “right child”. A binary search tree requires that the values stored by the left child are **less** than the value of the parent, and the values stored by the right child are **greater** than that of the parent.

Nodes as parents

Trees in computer science are often talked about similarly to family trees. A tree node that references one or more other nodes is called a “parent”.
A tree node can be a “parent” and a “child” simultaneously, because they are not exclusive. For instance, a node ‘b’ can be the child of node ‘a’, while being the parent to nodes ‘d’ and ‘e’. However, a child can only have one parent, while a parent can have multiple children.

Trees are composed of nodes

Trees are a data structure composed of nodes used for storing hierarchical data.
Each tree node typically stores a value and references to its child nodes.

Tree nodes children

A tree node contains a value, and can also include references to one or more additional tree nodes which are known as “children”.

Python TreeNode class

A `TreeNode` is a data structure that represents one entry of a tree, which is composed of multiple of such nodes.

The topmost node of a tree is called the “root”, and each node (with the exception of the root node) is associated with one parent node. Likewise, each node can have an arbitrary number of child nodes.

An implementation of a `TreeNode` class in Python should have functions to add nodes, remove nodes, and traverse nodes within the tree.

```
class TreeNode:
  def __init__(self, value):
    self.value = value # data
    self.children = [] # references to
other nodes

  def add_child(self, child_node):
    # creates parent-child relationship
    print("Adding " + child_node.value)
    self.children.append(child_node)

  def remove_child(self, child_node):
    # removes parent-child relationship
    print("Removing " +
child_node.value + " from " +
self.value)
    self.children = [child for child in
self.children
                     if child is not
child_node]

  def traverse(self):
    # moves through each node
referenced from self downwards
    nodes_to_visit = [self]
    while len(nodes_to_visit) > 0:
      current_node =
nodes_to_visit.pop()
      print(current_node.value)
      nodes_to_visit +=
current_node.children
```