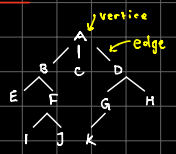
Depth - First Search

Input: graph =



Given a Node class that has a name and an array of optional children nodes. When put together, nodes form an acyclic tree-like structure.

Input: An empty array (fn should traverse the tree using Depth-first-search)

Output: Stores all of the nodes' names in the input array and returns it

Output:

["A", "B", "E", "F", "I", "J", "C", "D", "G", "K", "H"]

O(v+e) time; O(v) space

```
// O(v+e) time | O(v) space where v are the vertices (nodes)
// and e are the edges (node connections)
class Node {
  constructor(name) {
    this.name = name;
    this.children = [];
  }

  addChild(name) {
    this.children.push(new Node(name));
    return this;
  }

                          array that'll return the DFS order
  depthFirstSearch(array) {
  ① array.push(this.name)
    for (let i = 0; i < this.children.length; i++) {
  ②   this.children[i].depthFirstSearch(array)
    }
    return array
  }
}
```

Every Node has a name and arrray of children nodes

① We're calling DFS on the Node that has a name property so we append it to the array

② We iterate through the children Nodes and call DFS on each of them, passing in our array as an argument

Time depends on the vertices and edges. We are traversing every Node (vertex) to add it's name to the array. We also iterate through the children nodes of each vertex we're at and call the DFS function. The for loop at a node depends on the Edges of that node thus O(v+E)

Space depends on two things: the call stack and the array we're returning

1. The call stack would grow to the deepest level of the tree (due to recursive calls on the call stack) which is O(v) in the worst case (one branch A-B-C-D..)
2. The array in our case is also O(v) since we store all the vertices (nodes)