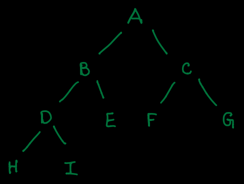
Input:

topAncestor = node A
descendantOne = node E
descendantTwo = node I

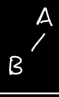
Three inputs, all of which are instances of an AncestralTree class that have an ancestor property pointing to their youngest ancestor



Output:

node B

Youngest common ancestor to the the two descendants
Note: A descendant is considered its own ancestor. e.g:



The youngest common ancestor to nodes A and B is node A

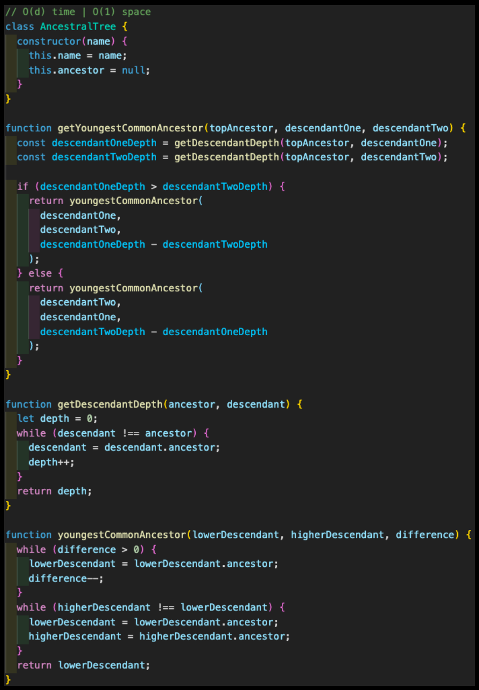
---

1) Get the depths of each descendant relative to the ancestor

2) Bring the deeper descendant to the same level as the higher descendant

3) Just be they're at the same level, doesn't mean they share the same ancestor so iterate up until they have the same ancestor and the return the ancestor

Time: O(d) (where d is the depth of the deepest node) as we'll have to traverse the graph d times at worst
Space: O(1) since we will recurse up iteratively making all other options constant time.



```
// O(d) time | O(1) space
class AncestralTree {
  constructor(name) {
    this.name = name;
    this.ancestor = null;
  }
}

function getYoungestCommonAncestor(topAncestor, descendantOne, descendantTwo) {
  const descendantOneDepth = getDescendantDepth(topAncestor, descendantOne);
  const descendantTwoDepth = getDescendantDepth(topAncestor, descendantTwo);

  if (descendantOneDepth > descendantTwoDepth) {
    return youngestCommonAncestor(
      descendantOne,
      descendantTwo,
      descendantOneDepth - descendantTwoDepth
    );
  } else {
    return youngestCommonAncestor(
      descendantTwo,
      descendantOne,
      descendantTwoDepth - descendantOneDepth
    );
  }
}

function getDescendantDepth(ancestor, descendant) {
  let depth = 0;
  while (descendant !== ancestor) {
    descendant = descendant.ancestor;
    depth++;
  }
  return depth;
}

function youngestCommonAncestor(lowerDescendant, higherDescendant, difference) {
  while (difference > 0) {
    lowerDescendant = lowerDescendant.ancestor;
    difference--;
  }
  while (higherDescendant !== lowerDescendant) {
    lowerDescendant = lowerDescendant.ancestor;
    higherDescendant = higherDescendant.ancestor;
  }
  return lowerDescendant;
}
```