CS 61A Fall 2020

Structure and Interpretation of Computer Programs

Quiz 4 Solutions

INSTRUCTIONS

- Please review this worksheet before the exam prep session. Coming prepared will help greatly, as the TA will be live solving without allocating much time for individual work.
- Either Sean or Derek will be on video live solving these questions. The other TA will be answering questions in the chat. It is in your best interest to come prepared with **specific** questions.
- This is not graded, and you do not need to turn this in to anyone.
- Fall 2020 students: the boxes below are an artifact from more typical semesters to simulate exam environments. Obviously this doesn't apply to this semester's exams, but we just kept the fields to keep our materials looking professional :) Feel free to ignore them.
- For multiple choice questions, fill in each option or choice completely.
  - ☐ means mark **all options** that apply
  - ◯ means mark a **single choice**

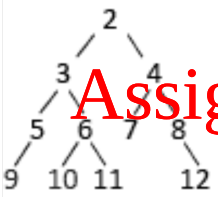
Last name	
First name	
Student ID number	
CalCentral email (_@berkeley.edu)	
Discussion Section	___ ___ ___
All the work on this exam is my own. **(please sign)**	

1. **Stonks**

(a) A quant firm has enlisted your help in maximizing their trade profits. You know ahead of time how much profit you will make by executing each trade. These values are stored in a tree (i.e., every node's value corresponds to the profit you earn by making a certain trade). However, if you decide to trade a stock represented by some node S, you are not allowed to execute trades that have an edge directly connected to S (i.e., if you earn profit S, you cannot earn the profit from the parent or children of S). For example, in the following tree (see below), if I choose to execute the trade corresponding to the node 4, then I earn \$4, but I cannot trade the stock corresponding to the 2, the 7, or the 8, since all 3 of these are connected to 4.
Return the maximum profit you can make based on these constraints. The maximum you can make from the tree below is \$52 (3 + 9 + 10 + 11 + 7 + 12). Note that none of these selected profits have direct edges to each other.

FYI: assume you have access to `left` and `right`, which are functions that select the left and right subtrees of a given tree. They return `None` if the requested subtree does not exist. Also assume each node has at most 2 children.

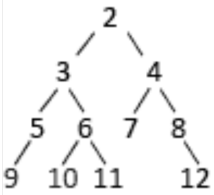
Challenge (out of scope for 61A): solve this in one post-order traversal **without** memoization.




```
def profit(tree):
    return helper(tree, True)

def helper(tree, z):
    if tree is None:
        return 0
    if is_leaf(tree):
        return label(tree) * z
    if z:
        x = helper(left(tree), False) + helper(right(tree), False) + label(tree)
        y = helper(left(tree), True) + helper(right(tree), True)
        return max(x, y)
    else:
        return helper(left(tree), True) + helper(right(tree), True)
```

2. **Don't Leaf Me**

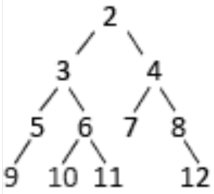
(a) The leaves in our trees are looking for love and would like to mingle with each other. However, they can only contact other leaves that are within a distance of `k`, where `k` is the fewest number of edges that connect one leaf to another. Return the number of leaf pairs in the tree that are separated by $\leq k$ edges. For example, for $k = 4$ in the following tree, the function should return 4, since there are 4 pairs that are within k edges of each other (10 and 11, 7 and 12, 9 and 10, 9 and 11). **FYI:** assume you have access to `left` and `right`, which are functions that select the left and right subtrees of a given tree. They return `None` if the requested subtree does not exist. Also assume each node has at most 2 children.




```
def pairs(tree, k):

    def helper(tree, k):

        if tree is None:

            return [{}, 0]

        if is_leaf(tree):

            return [{1: 1}, 0]

        left, total_left = helper(left(tree), k)

        right, total_right = helper(right(tree), k)

        total = total_left + total_right

        for depth in left:

            for i in range(k - depth + 1):

                total += left[depth] * right.get(i, 0)

        ret = {}

        for key in left:

            ret[key + 1] = left[key]

        for key in right:

            ret[key + 1] = ret.get(key + 1, 0) + right[key]

        return [ret, total]

    return helper(tree, k)[1]
```

3. **I Am Your Father**

(a) The lowest common ancestor of two nodes `p` and `q` is the lowest node in the tree from which both `p` and `q` can be reached. For example, in the following tree, the lowest common ancestor of 10 and 11 is 6. The lowest common ancestor for 9 and 6 is 3. The lowest common ancestor of 4 and 12 is 4. Assume all values in the tree are unique, and return the value associated with the lowest common ancestor of `p` and `q`. **FYI:** assume you have access to `left` and `right`, which are functions that select the left and right subtrees of a given tree. They return `None` if the requested subtree doesn't exist. Also assume each node has at most 2 children.




```
def lowestCommonAncestor(tree, p, q):
    L = helper(left(tree), p, q)
    if L and L[1]:
        return L[0]
    R = helper(right(tree), p, q)
    if R and R[1]:
        return R[0]
    return label(tree)

def helper(tree, p, q):
    if tree is None:
        return
    if is_leaf(tree) and (label(tree) in [p, q]):
        return [label(tree), False]
    L, R = helper(left(tree), p, q), helper(right(tree), p, q)
    if label(tree) in [p, q]:
        return [label(tree), not (L is None and R is None)]
    if L and (L[1] or not R):
        return L
    if L and R:
        return (label(tree), True)
    if (not L) and R:
        return R
```


I wrote a rap, a little easter egg for people who actually read these solutions...

–To the 8 mile soundtrack–
Look
If you had
One line
One piece of code
To seize every point you ever wanted
One lambda
Would you capture it
Or just let it slip

His palms are sweaty
Knees weak, fingers are heavy
There's tests failing on his computer already, mom's spaghetti code
He's nervous, but on the surface he looks calm and ready
To drop tables, but he keeps on forgettin'
What code he wrote down, the anNOUNCEments so loud
He opens VSCode, but the code won't come out
Tests failing (how?!), every build broken now
The clock's run out
Pencils down
Time's out over blow

Revert back to stability
Oh there goes validity
Oh there goes sanity
He choked
He so bad
But he won't
Give up that easy
No, he knows his whole Slack is these blokes
He knows that but he's broke

You better lose yourself in the bug fix
The moment
You own it
You better never let it go
You only got one shot to feel like you've written good code.
This opportunity comes once in a lifetime yo!