

FINAL REVIEW 12

DATA C88C

August 1, 2024

0.1 Environment Diagram

1. (Fall 2012) Draw the environment diagram.

```
def box(a) :  
    def box(b) :  
        def box(c) :  
            a.append(c)  
            return (a, b)  
        return box  
    gift = box(1)  
    return (gift(2), gift(3))  
box([4])
```

0.2 OOP

1. (Summer 2015 Final) The TAs are building a social networking website called CS61A+. The TAs plan to represent the network in a class called `Network` that supports the following method:

- `add_friend(user1, user2)` adds `user1` and `user2` to each other's friends lists. If `user1` or `user2` are not in the `Network`, add them to the dictionary of friends.

Help the TAs implement these two methods to make their social networking website popular!

```
class Network:
```

```
    """
```

```
    >>> cs61a_plus = Network()
    >>> cs61a_plus.add_friend('Robert', 'Jeffrey')
    >>> cs61a_plus.friends['Robert']
    ['Jeffrey']
    >>> cs61a_plus.friends['Jeffrey']
    ['Robert']
    >>> cs61a_plus.add_friend('Jessica', 'Robert')
    >>> cs61a_plus.friends['Robert']
    ['Jeffrey', 'Jessica']
    """
```

```
def __init__(self):
```

```
    self.friends = {}          # Maps users to a list of their friends
```

```
def add_friend(self, user1, user2):
```

```
    if _____:
```

```
        _____
```

```
    if _____:
```

```
        _____
```

```
        _____
```

```
        _____
```

CS61A+ turns out to be unpopular. To attract more users, the TAs want to implement a feature that checks if two users have at most n degrees of separation. Consider the following CS61A+ Network:

```
self.friends = {
    'Robert': ['Jeffrey', 'Jessica'],
    'Jeffrey': ['Robert', 'Jessica', 'Yulin'],
    'Jessica': ['Robert', 'Jeffrey', 'Yulin'],
    'Yulin': ['Jeffrey', 'Jessica'],
    'Albert': []
}
```

- There is 1 degree of separation between Robert and Jeffrey, because they are direct friends.
- There are 2 degrees of separation between Robert and Yulin (Robert → Jessica → Yulin)
- The degree of separation between Albert and anyone else is undefined, since Albert has no friends.

class Network:

Code from previous question

```
def degrees(self, user1, user2, n):
    """In these doctests, assume cs61a_plus is a Network with the
    dictionary of friends described in the example.

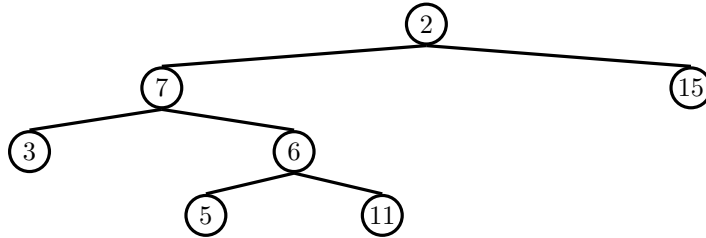
    >>> cs61a_plus.degrees('Robert', 'Yulin', 2)    # Exactly 2 degrees
    True
    >>> cs61a_plus.degrees('Robert', 'Jessica', 2)  # Less than 2
           degrees
    True
    >>> cs61a_plus.degrees('Yulin', 'Robert', 1)    # More than 1
           degree
    False
    >>> cs61a_plus.degrees('Albert', 'Jessica', 10) # No friends!
    False
    """
    if _____:
        return True
    elif _____:
        return False
    for friend in _____:
        if _____:
            return True
    return _____
```

0.3 Trees

1. Write a function that takes in a tree and a value x and returns a list containing the nodes along the path required to get from the root of the tree to a node containing x .

If x is not present in the tree, return `None`. Assume that the entries of the tree are unique.

For the following tree, `find_path(t, 5)` should return `[2, 7, 6, 5]`



```

def find_path(tree, x):
    """
    >>> t = Tree(2, [Tree(7, [Tree(3), Tree(6, [Tree(5), Tree(11)])]),
    Tree(15)])
    >>> t.value
    2
    >>> find_path(t, 5)
    [2, 7, 6, 5]
    >>> find_path(t, 10) # returns None
    """
  
```

```

if _____:
  
```

```

    return _____
  
```

```

_____:
  
```

```

    path = _____
  
```

```

    if _____:
  
```

```

        return _____
  
```

0.4 Recursion

1. (Fall 2013) Fill in the blanks in the implementation of `paths`, which takes as input two positive integers `x` and `y`. It returns the number of ways of reaching `y` from `x` by repeatedly incrementing or doubling. For instance, we can reach 9 from 3 by incrementing to 4, doubling to 8, then incrementing again to 9.

```
def inc(x):
    return x + 1

def double(x):
    return x * 2

def paths(x, y):
    """Return the number of ways to reach y from x by repeated
    incrementing or doubling.
    >>> paths(3, 5) # inc(inc(3))
    1
    >>> paths(3, 6) # double(3), inc(inc(inc(3)))
    2
    >>> paths(3, 9) # E.g. inc(double(inc(3)))
    3
    >>> paths(3, 3) # No calls is a valid path
    1
    """
    if x > y:
        return _____

    elif x == y:
        return _____

    else:
        return _____
```

0.5 Linked List

1. You are trying to communicate with your project partner but Evil Eve is attempting to intercept your messages. Write a function `insert_secret` that takes a `LinkedList`, `lnk`, and a string `secret` and mutates `lnk` by interleaving the secret message into the linked list. Look at the doctests for details. (You can assume that the length of the secret message is less than or equal to the length of the linked list)

```
def insert_secret(lnk, secret):
    """
    >>> two_link = Link(1, Link(2))
    >>> insert_secret(two_link, 'f')
    >>> two_link
    Link(1, Link('f', Link(2)))
    >>> surprise = Link(8, Link(8))
    >>> insert_secret(surprise, 'CS')
    >>> surprise
    Link(8, Link('C', Link(8, Link('S'))))
    """
```

0.6 Iterator/Generator

1. Write a generator function `generate_subsets` that returns all subsets of the positive integers from 1 to n . Each call to this generator's **next** method will return a list of subsets of the set $[1, 2, \dots, n]$, where n is the number of previous calls to `next`.

```
def generate_subsets():
    """
    >>> subsets = generate_subsets()
    >>> for _ in range(3):
    ...     print(next(subsets))
    ...
    [[]]
    [[], [1]]
    [[], [1], [2], [1, 2]]
    """
```

0.7 SQL

1. Write a query that outputs all divisions for which there is more than one employee, and all pairs of employees within that division have a salary less than 100,000.

Reminder: we are using a table named `records` that stores information about the employees at a small company¹. Each of the eight rows represents an employee.

| Name | Division | Title | Salary | Supervisor |
|-----------------|-----------------|--------------------|---------------|-------------------|
| Ben Bitdiddle | Computer | Wizard | 60000 | Oliver Warbucks |
| Alyssa P Hacker | Computer | Programmer | 40000 | Ben Bitdiddle |
| Cy D Fect | Computer | Programmer | 35000 | Ben Bitdiddle |
| Lem E Tweakit | Computer | Technician | 25000 | Ben Bitdiddle |
| Louis Reasoner | Computer | Programmer Trainee | 30000 | Alyssa P Hacker |
| Oliver Warbucks | Administration | Big Wheel | 150000 | Oliver Warbucks |
| Eben Scrooge | Accounting | Chief Accountant | 75000 | Oliver Warbucks |
| Robert Cratchet | Accounting | Scrivener | 18000 | Eben Scrooge |

¹Example adapted from Structure and Interpretation of Computer Programs