Spring 2021 Final Exam Solutions

CS 88

Problem 2: Conceptual Questions

1. Explain how the Place Class in the Ants Vs. SomeBees Project models a Linked List data structure.

The entrance points to the exit, similar to how a linked list's rest attribute points to another linked list.

2. Suppose we have a Restaurant class with the following attributes and methods:

- __init__(self, restaurant_name, menu, available_tables): creates instance attributes restaurant_name, menu, available_tables
- add_item(self, name): adds a new menu item to menu
- cook(self, order): prepares a single order for a customer
- clean_table(self): available_tables increments by 1
- update_price(self, menu_item): updates the price of an item on the menu

Using your knowledge of OOP and inheritance, create an example subclass that would inherit the Restaurant class and explain how you would make use of inheritance. No code needs to be written for this problem, a brief (2-4 sentences) description and justification is sufficient.

We could build a highEndRestaurant subclass which would call the parent method cook(self, order) multiple times within its own buildCourse(self, order) method to create a full course meal. **Other subclass explanations may also be awarded full points based on the student's explanation.**

3. Which data structure would you use to store a family's heritage and why?

Tree data structure, because a family's heritage follows a hierarchical structure.

Problem 3: Inverse WWPD

1. Fill in the blanks for the values of `l` and `n` such that the final expression evaluates to `True`.

```
def mystery(l, n, f):
    if l is Link.empty:
        return n == 0
    return f(n, l.first) and mystery(l.rest, n // 10, f)
```

```
>>> l = _________________
>>> n = _________________
>>> l.first != l.rest.first and mystery(l, n, lambda x, y: x % 10 == y)
True
```

Solution:

Example:

```
l = Link(4, Link(3, Link(1)))
n = 134
```

Any linked list with 2 or more elements and number n will work as long as the digits of n are the elements in the linked list in the reverse order. Additionally, the first two elements of the linked list must be different.

2. a) For this problem, we will be working with the `MusicAlbum` class, where each `MusicAlbum` instance has two attributes, the `album_name` and a `votes_per_song` dictionary that has songs in the album as *keys* and the number of votes per song as *values*.

Though the following `MusicAlbum` class is syntactically correct, there is a bug with the logic in the `most_popular_song_in_album` method. The method is supposed to find the song with the most votes (ties are broken arbitrarily) and though it sometimes works correctly, the logic bug makes it incorrect for some sequence of method calls.

Briefly describe what the issue in the `most_popular_song_in_album` method is (no more than 30 words). Then, come up with a sequence of method calls that would expose the bug in the current code (no more than 5 lines).

```
class MusicAlbum:
    def __init__(self, album_name, songs):
        self.album_name = album_name
        self.votes_per_song = {}
        for song in songs:
```

```
            self.votes_per_song[song] = 0

    def up_vote(self, song):
        assert song in self.votes_per_song
        self.votes_per_song[song] += 1

    def down_vote(self, song):
        assert song in self.votes_per_song
        self.votes_per_song[song] -= 1

    def most_popular_song_in_album(self):
        most_popular_song = ""
        most_popular_votes = 0
        for song in self.votes_per_song:
            if self.votes_per_song[song] >= most_popular_votes:
                most_popular_song = song
                most_popular_votes = self.votes_per_song[song]
        return most_popular_song
```

Solution:

The problem is that `most_popular_votes` is initialized to 0, but it is possible that each song has negative votes. Therefore even the most popular song will not have more than 0 votes, so the `most_popular_song` will stay “”

Example:

```
a = MusicAlbum("A", ["S1", "S2"])
a.down_vote("S1")
a.down_vote("S2")
a.down_vote("S1")
print(a.most_popular_song_in_album())
```

b) What is the runtime of the most_popular_song_in_album method in terms of N, which is the number of songs in the `self.votes_per_song` dictionary?

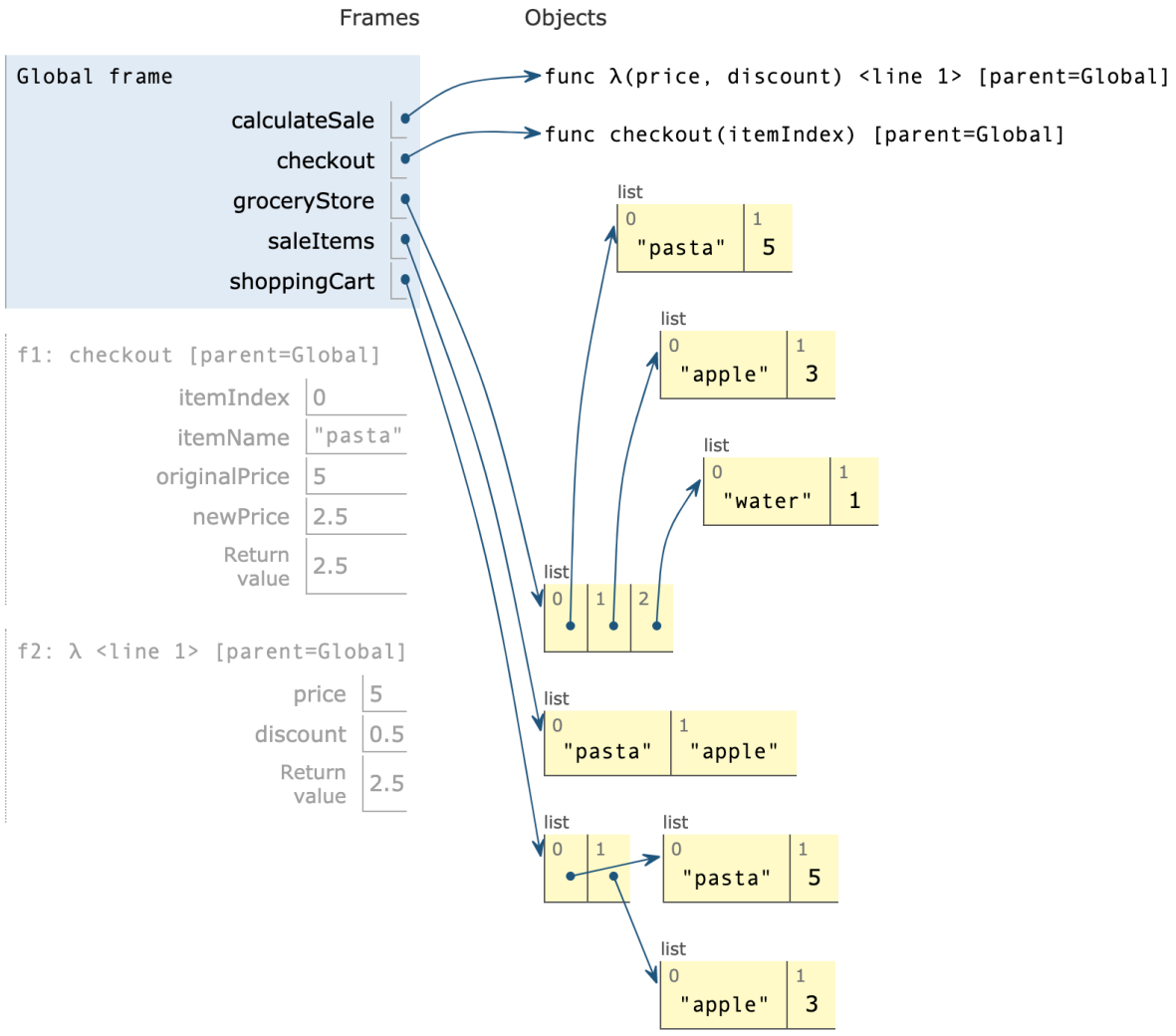
θ(1) θ(log N) θ(N) θ(N^2) θ(2^N)

Solution: θ(N)

Problem 4: Let's go grocery shopping!

Note: Please use: http://tutor.cs61a.org/ to check your environment diagram if needed!

The following environment diagram was generated by a program to completion:



In this series of questions, you'll fill in the blanks of the program that follows so that its execution matches the environment diagram. You may want to fill in the blanks in a different order; feel free to answer the questions in whatever order works for you.

```
calculateSale = ____________________
```

```
                        (a)

def checkout(itemIndex):
      itemName = _____________________
                        (b)
      if itemName in saleItems:
            originalPrice = groceryStore[itemIndex][1]
            newPrice = calculateSale(originalPrice, 0.5)
            return newPrice
      else:
            return groceryStore[itemIndex][1]

groceryStore = [["pasta", 5], ["apple", 3], ["water", 1]]
saleItems = ["pasta", "apple"]
shoppingCart = _____________________
                        (c)
checkout(0)
```

***Hint*: Try solving blank c first before blanks a and b!**

a) Which of these could fill in blank (a)? ***Select one answer from below.***

- O `lambda price, discount: originalPrice * discount`
- O `lambda price, discount : price / discount`
- O `lambda price, discount : price + discount`
- O `lambda price, discount : price * discount`

b) Which of these could fill in blank (b)? ***Check all that apply.***

- ❑ `groceryStore["pasta"][0]`
- ❑ `groceryStore[itemIndex][0]`
- ❑ `groceryStore[0][0]`
- ❑ `groceryStore[-3][0]`

c) Which of these could fill in blank (c)? ***Select one answer from below.***

- O `[groceryStore[0]] + [groceryStore[1]]`
- O `[groceryStore[0][:]] + [groceryStore[1][:]]`
- O `groceryStore[0][:].append(groceryStore[1][:])`
- O `["pasta", 5, "apple", 3]`

Python Tutor Link

```
calculateSale = lambda price, discount: price * discount

def checkout(itemIndex):
```

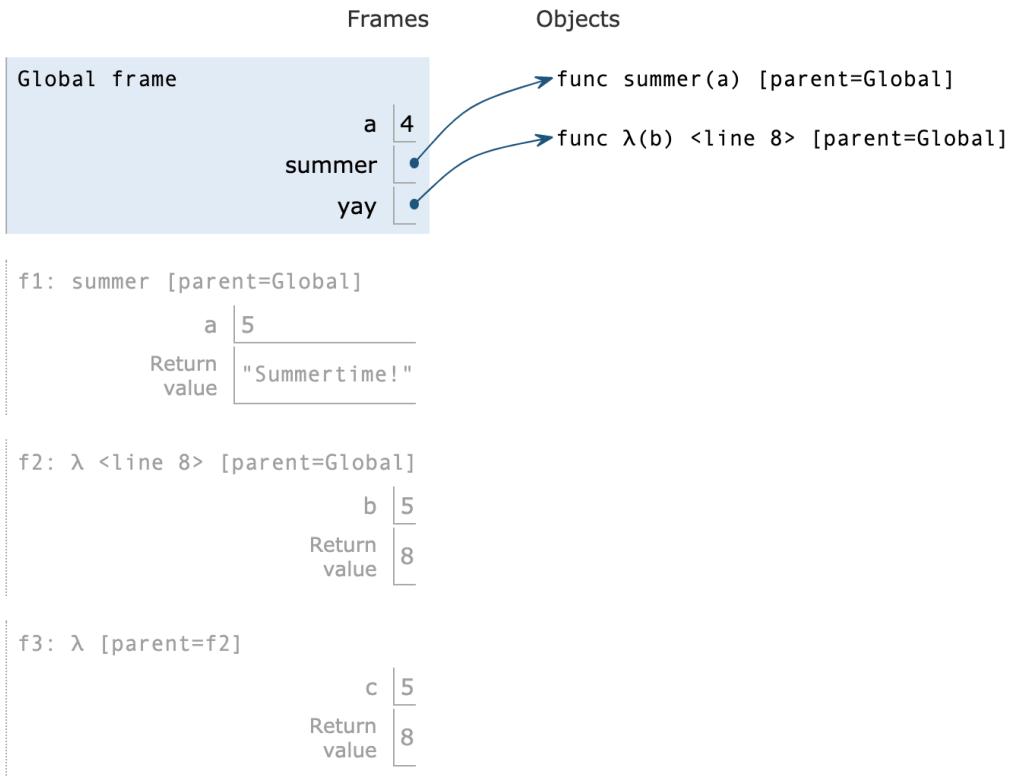
```
        itemName = groceryStore[itemIndex][0]
        if itemName in saleItems:
                originalPrice = groceryStore[itemIndex][1]
                newPrice = calculateSale(originalPrice, 0.5)
                return newPrice
        else:
                return groceryStore[itemIndex][1]

groceryStore = [["pasta", 5], ["apple", 3], ["water", 1]]
saleItems = ["pasta", "apple"]
shoppingCart = [groceryStore[0][:]] + [groceryStore[1][:]]
checkout(0)
```

Problem 5: Summer is finally here!

Please use: http://tutor.cs61a.org/ to check your environment diagram!

The following environment diagram was generated by a program to completion:




```
a = 4
def summer(a):
    if _______________:
                (a)
        return "Summertime!"
    else:
        return "Not yet :("

yay = ________________________
                (b)

summer(_________)
          (c)
```

In this series of questions, you'll fill in the blanks of the program that follows so that its execution matches the environment diagram. You may want to fill in the blanks in a different order; feel free to answer the questions in whatever order works for you.

a) Which of these could fill in blank (a)? ***Check all that apply.***

- ❑ yay(a) == 8
- ❑ yay(a) >= 7
- ❑ yay(a) < 10
- ❑ yay(a) == 7

b) Which of these could fill in blank (b)? ***Select one answer from below.***

- o lambda b: (lambda c: a*2) (b)
- o lambda b: (lambda c : 8)
- o lambda b: (lambda c: 8) (a)
- o lambda b: (lambda c: b*2) (5)

c) Fill in blank (c) and please write out the entire line of code below.

_____summer(5)_____

Python Tutor Link

```
a = 4
def summer(a):
    if yay(a) >= 7:
        return "Summertime!"
    else:
        return "Not yet :("

yay = lambda b: (lambda c: a * 2)(b)
summer(5)
```

Problem 6: Time's up, let's do this!

Create a class class `Clock` which keeps track of hours and minutes. The constructor takes in the hour and minute that the clock should be set to initially. It also has the following two functions

`advance(self)`

- This function should advance the minutes by 1 minute
- If the minute is greater than or equal to 60 then minutes should be set to 0 and hour should be advanced by 1 hour
- If hour is greater than 12 then hour should be set to 0

`sync(self)`

- This function should synchronize all clocks ever created to have the same hour and minute to the instance that this function is called on (see doctests for an example).

Also fill in the `FastClock` class which inherits from Clock. The `advance` method of the `FastClock` should advance the minutes twice as much as a `Clock`. You cannot use more than the lines provided, but feel free to leave some lines blank.

```
class Clock:

    allClocks = []

    def __init__(self, hour, minutes):
        self.hour = hour
        self.minute = minutes
        Clock.allClocks.append(self)

    def advance(self):
        self.minute += 1
        if self.minute >= 60:12
            self.minute = 0
            self.hour += 1
            if self.hour > 12:
                #self.hour = 1 was also accepted
                self.hour = 0

    def sync(self):
        for clock in Clock.allClocks:
```

```
            clock.hour = self.hour
            clock.minute = self.minute

class FastClock(Clock):

    def advance(self):
        super().advance()
        super().advance()
```

Problem 7: Coupons!

Fill in the blanks below to create a coupon tracker that has the functionality in the doctest below. Understanding the doctests will be especially important for this problem.

`coupon_tracker` returns the function `add_coupon` which adds `num_copies` copies of the coupon represented by the string `code` to the dictionary `count`.

Coupons can be added by calling `add_coupon` but when the string "finish" is passed in as `code`, `add_coupon` returns another function that allows coupons to be acquired.

Coupons can be acquired by calling `acquire_coupon` but to successfully acquire a coupon, it must be in the dictionary and have a positive number of copies. Each time a coupon is successfully acquired its number of copies is decreased by 1.

```
def coupon_tracker():
    count = {}
    def acquire_coupon(code):
        if code in count and count[code] > 0:
            count[code] -= 1
            print("Success!")
        else:
            print("Failure.")

    def add_coupon(code, num_copies=0):
        if code == "finish":
            return acquire_coupon
        elif code in count:
            count[code] += num_copies
        else:
            count[code] = num_copies
    return add_coupon
```

Problem 8: List of Links

Your friend is trying to write a function `combine` which takes in a list of linked lists. It concatenates each of the linked lists in sequential order and returns the result. Mutating the linked lists is allowed, and the list that is passed in is guaranteed to have at least one linked list. However, their code is buggy. Describe 3 bugs with the code and the way to fix each of the bugs. After fixing each of the bugs, the code should work as intended, and all of the doctests should output the expected output. You may only provide 3 bugs, if you provide more than 3 bugs then only the first 3 will be considered.

```
def combine(lst):
    """
    >>>lst1 = [Link(1), Link(2), Link(3)]
    >>>combine(lst1)
    Link(1, Link(2, Link(3)))
    >>>lst2 = [Link(1,Link(4, Link(5))), Link(2), Link(3,Link(2))]
    >>>combine(lst2)
    Link(1, Link(4, Link(5, Link(2, Link(3, Link(2))))))
    >>>lst3 = [Link(1)]
    >>>combine(lst3)
    Link(1)
    """
1.    for i in range(len(lst)):
2.        curr = lst[i]
3.        while curr.rest:
4.            curr = curr.rest
5.        curr.rest = lst[i + 1].first
6.    return curr
```

Now write a working solution to `combine`

Solution:

Bug 1: Since we are connecting linked lists by their last link, we do not need to link the last list with anything. So our for loop should be range(len(lst) - 1) otherwise we will get an out of bounds error when we get to lst[i + 1]

Bug 2: We should be connecting the linked lists by the links themselves not the link.first. So the line curr.rest = lst[i + 1].first should just be curr.rest = lst[i + 1]

Bug 3: We should return the first link in the list of linked lists. Currently we are returning the last link since we return curr. So we should instead return lst[0]

```
def combine(lst):
    for i in range(len(lst) - 1):
```

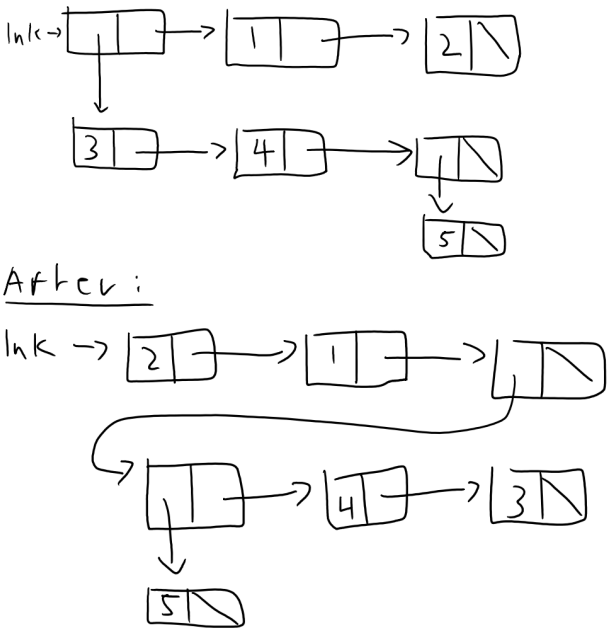
```
        curr = lst[i]
        while curr.rest:
            curr = curr.rest
        curr.rest = lst[i + 1]
    return lst[0]
```

Problem 9: Surprise Holiday!

Paul is just about to submit his test linked list inputs for his cs88 assignment a couple hours before the due date. He checks the calendar one last time, and gasps in horror - he forgot it was opposite day. According to the laws of the universe, he must now submit all his linked lists BUT REVERSED. In this problem you will help Paul write a function so that he can create new linked list inputs to submit in time.

Paul's Linked Lists are not simple linked lists, they are nested, meaning that each linked list can have a first value that is another linked list (and those can have linked lists as their first value and so on). It is your responsibility to write a function that reverses nested linked list, and all nested lists within the list, and returns a new list. Below is a pictorial example.

Hint: isInstance(Link.first, Link) checks whether or not Link.first is of type Link (meaning that it is a linked list). We gave this part in the skeleton as a hint for the function.

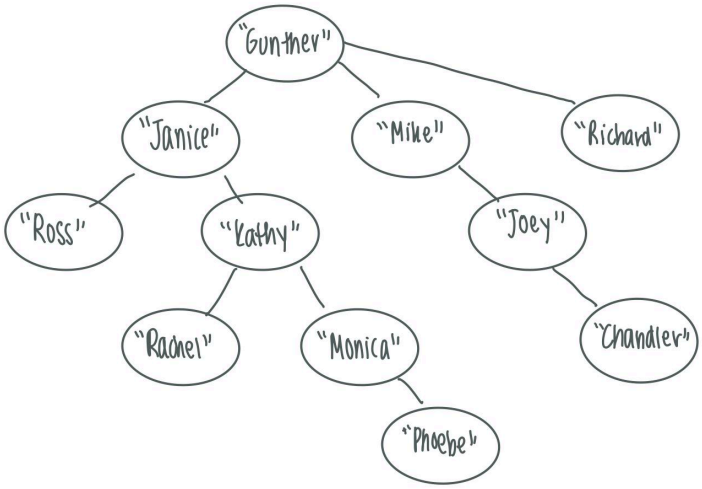



```
def nested_reverse(link):
    new = Link.empty
    while link is not Link.empty:
        if isinstance(link.first, Link):
            new = Link(nested_reverse(link.first), new)
        else:
            new = Link(link.first, new)
        link = link.rest
```

```
return new
```

Problem 10: Working in the IndusTREE

You have been recently hired by Company Z to help the company's recruiting team! Company Z's organizational structure can be represented as a tree (see the image below) with each employee as a branch of their manager. Note that it is possible for a manager to also be an employee, e.g. Kathy reports to Janice, and Rachel and Monica report to Kathy! By this representation, the president is the root node!



a) For your first task, you are asked to consider how many openings managers have on their teams for potential new hires. A manager is anyone that has at least one person reporting to them. At company Z, no manager should have more than k employees immediately reporting to them.

   Given a tree `t` representing company Z and the number `k` representing the max number of employees reporting to one manager, return the number of openings managers have on their teams for new hires across the company.
   You can assume that in the current tree `t`, no manager is managing more than k employees.

   Solution:

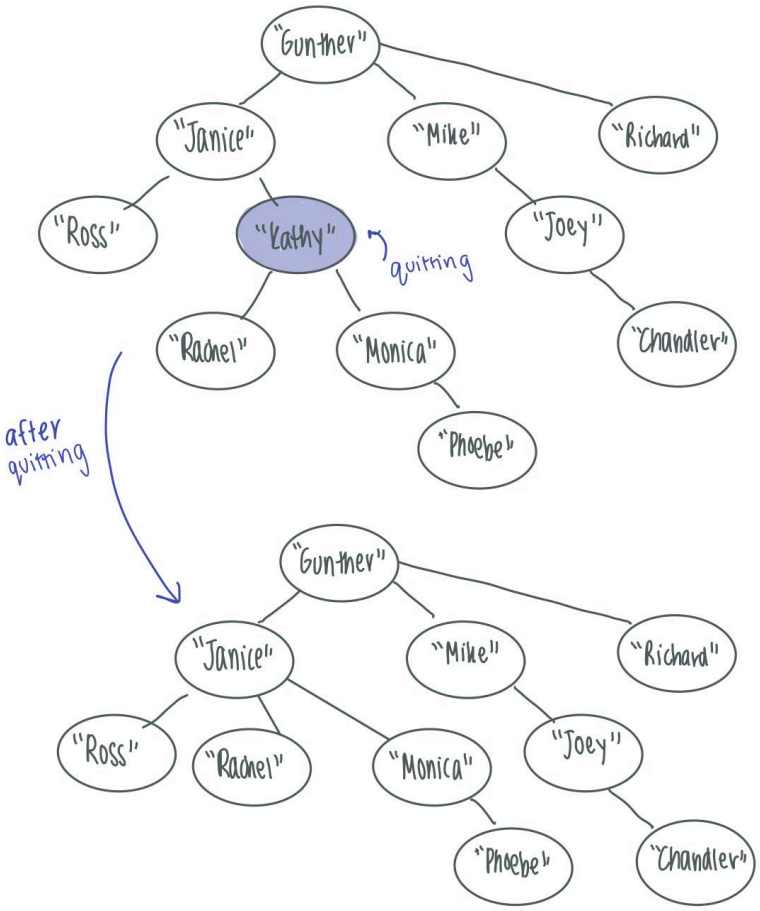
```
def num_openings(t, k):
    if t.is_leaf():
        return 0
    else:
        openings += k - len(t.branches)
        for b in t.branches:
            openings += num_openings(b, k)
        return openings
```

b) For your second task, you have been told that there is an employee quitting the company because they've found a new job opportunity.

Given a tree `t` representing company Z, modify the tree `t` to remove the employee that is leaving, moving everyone who previously reported to that employee as now reporting to that employee's previous manager. For the sake of this problem, you can assume that there are no longer any constraints on how many employees can report to a given manager.

You can assume that the president (or the root node) will never quit the company.

Note: the following image may help better illustrate the doctests.



Solution:

```
def quit(t, employee):
    for b in t.branches:
        if b.entry == employee:
            t.branches.extend(b.branches)
```

```
            t.branches.remove(b)
            return

    for b in t.branches:
        quit(b, employee)
```

Problem 11: Root Path Sums

The year is 2093 and the world has fallen into a post-apocalyptic nuclear winter. Michael Ball prime, a copy of Michael Ball's consciousness put into a robotic exoskeleton, needs to travel away from its shelter to charge himself in nearby cities.

Michael Ball prime starts at the root of a tree, and travels along branches to each node, where he charges himself according to the value (charge points) at that node. But beware, Michael Ball prime's battery can only hold n charge points, and will melt down if given anymore.

Write a procedure to find all paths that sum to exactly n charge points so that he becomes fully charged. In other words, find all the paths in the tree (starting at the root) where the nodes add up to n.

```
def root_path_sums(t, n):
    """
    >>> t = Tree(1, [Tree(2, [Tree(1)]), Tree(3)])
    >>> gen = root_path_sums(t, 4)
    >>> next(gen)
        [1, 2, 1]
    >>> next(gen)
        [1, 3]
    >>> next(gen)
         StopIteration error
    >>> gen2 = root_path_sums(t, 15)
    >>> next(gen2)
         StopIteration error
    """
    def root_path_sums(t, n):
        if t.label == n:
            yield [t.label]
        else:
            for b in t.branches:
                for path in root_path_sums(b, n - t.label):
                    cur_path = [t.label] + path
                    yield cur_path
```

Problem 12: Post Final Dessert!

```
CREATE TABLE ice_cream AS
      SELECT “vanilla” as flavor, “classic” as category UNION
      SELECT “chocolate”, "classic” UNION
      SELECT “strawberry”, “fruits” UNION
      SELECT “mango”, “fruits” UNION
      SELECT “coffee”, “fancy” UNION
      SELECT “mint chocolate chip”, “fancy”;

CREATE TABLE staff AS
      SELECT “Vandana” as name, "coffee" as favorite UNION
      SELECT “Shreya”, "strawberry"  UNION
      SELECT “Sophia”, "mango" UNION
      SELECT “Nick”, "vanilla" UNION
      SELECT “Lukas”, "mango" UNION
      SELECT “Tommy”, "mint chocolate chip" UNION
      SELECT “Kevin”, "strawberry" UNION
      SELECT “Minnie”, "chocolate" UNION
      SELECT “Matt”, "vanilla" UNION
      SELECT "Michael", "coffee" UNION
      SELECT "Gerald", "mango";
```

Use the above tables to write queries below. Keep in mind that not all blanks need to be filled in for each query.

1. Write a SELECT statement which would output the names of people on staff who like ice cream in the “fruits” category.

```
Output:
Shreya
Sophia
Lukas
Kevin
Gerald
```

```
SELECT name
FROM staff, ice_cream
WHERE favorite = flavor AND category = "fruits"
```

2. Write a SELECT statement using the blocks of code given in order to match everyone in the staff table in pairs if they like ice cream in the classic category. Not all the blocks may need to be used and they may be incomplete or incorrect.
Hint: use the output to determine how to handle duplicates

```
Output:
Nick   | Minnie
Nick   | Matt
Minnie | Matt
```

```
SELECT a.name, b.name
FROM staff as a, staff as b, ice_cream as ice_cream_a, ice_cream as
ice_cream_b
WHERE a.favorite = ice_cream_a.flavor AND b.favorite =
ice_cream_b.flavor AND ice_cream_a.category = "classic" AND
ice_cream_b.category = "classic" AND a.name > b.name
```

3. Write a SELECT statement to output how much of each ice cream flavor is needed for everyone to get their favorite flavor in decreasing popularity. If the popularity is tied, break ties alphabetically. Do not include flavors where only one person likes the flavor.

```
Output:
mango               | 3
coffee              | 2
strawberry          | 2
vanilla             | 2
```

```
SELECT favorite, COUNT(*)
FROM staff
GROUP BY favorite
HAVING COUNT(*) > 1
ORDER BY COUNT(*) DESC, favorite
```