MIDTERM 2 REVIEW SESSION Solutions

COMPUTER SCIENCE MENTORS 61A

October 23, 2022

1 Recursion

1. **Skipping Around** (Su21 MT2 Q2b)

   A skip list is defined as a sublist of a list such that each element in the sublist is non adjacent in the original list. For original list `[5, 6, 8, 2]`, the lists `[5, 8]`, `[5, 2]`, `[6, 2]`, `[5]`, `[6]`, `[8]`, `[2]`, `[]` are all skip lists of the original list. The empty list is always a skip list of any list.

   Given a list `int_lst` of unique integers, return a list of all unique skip lists of `int_lst` where each skip list contains integers in strictly increasing order. The order in which the skip lists are returned does not matter.

```
def list_skipper(int_list):
    """
    >>> list_skipper([5,6,8,2])
    [[5, 8], [5], [6], [8], [2], []]
    >>> list_skipper([1,2,3,4,5])
    [[1, 3, 5], [1, 3], [1, 4], [1, 5], [1], [2, 4], [2, 5], [2], [3, 5],
       [3], [4], [5], []]
    >>> list_skipper([])
    [[]]
    """
    if len(int_lst) == 0:

        return _____________________________

    with_first = _____________________________

    without_first = _____________________________

    with_first = [ __________ for x in with_first if x == []

        or _____________]

    return with_first + without_first
```

Curated by Gene Pan, Raman Varma, Frank Liu, Nicholas Trong-Nhan Nguyen, Elisa Kim, Riya Singhal, Ryan Campbell, Issac Chen, Vy Ho, Jaewon Dong, and Jonathan Chen in association with Gabe Classon, Aditya Balasubramanian, and Maya Romero

```
def list_skipper(int_list):
    if len(int_lst) == 0:
        return [[]]
    with_first = list_skipper(int_lst[2:])
    without_first = list_skipper(int_lst[1:])
    with_first = [ [int_lst[0]] + x for x in with_first if x == [] or
       x[0] > int_lst[0]]
    return with_first + without_first
```

2. **maxkd** (Su20 MT1 Q3)

```
def maxkd(meteor, k):
    """
    Given a number `meteor`, finds the largest number of length `k` or
       fewer,
    composed of digits from `meteor`, in order.

    >>> maxkd(1234, 1)
    4
    >>> maxkd(32749, 2)
    79
    >>> maxkd(1917, 2)
    97
    >>> maxkd(32749, 18)
    32749
    """
    if ____________________________________________:

        return ____________________________________________

    a = ____________________________________________

    b = ____________________________________________

    return ____________________________________________
```

```
def maxkd(meteor, k):
    """
    Given a number `meteor`, finds the largest number of length `k` or
       fewer,
    composed of digits from `meteor`, in order.

    >>> maxkd(1234, 1)
    4
    >>> maxkd(32749, 2)
    79
    >>> maxkd(1917, 2)
    97
    >>> maxkd(32749, 18)
    32749
```

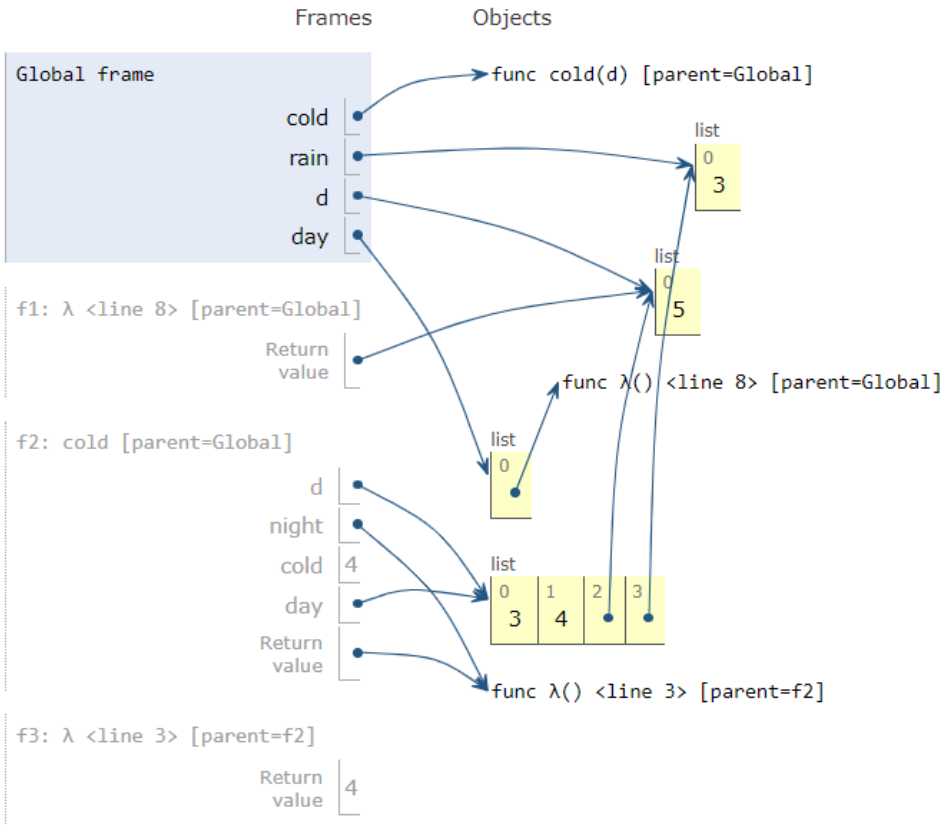
```
"""
if meteor == 0 or k == 0:
    return 0
a = maxkd(meteor // 10, k - 1) * 10 + meteor % 10
b = maxkd(meteor // 10, k)
return max(a, b)
```

1. **Protect the Environment** (Fa19 Final Q2)

   Draw the environment diagram that results from running the following code.

```
def cold(d):
    day = rain[:1]
    night = lambda: len(day)
    cold = rain.pop()
    day = d
    return night
rain = [3, 4]
d, day = [5], [lambda: d]
cold(rain + [day[0](), rain])()
```

   https://tinyurl.com/ms67cb5c



2. **Wait 'til You See What's in Store** (Su21 MT Q5a)

   Implement `memory_store`, a function that will return two functions: `add_val` and `times_seen`. When called on a number `var1`, `times_seen` will return the number of times `add_val` has been called on that particular value `var1`.

```
def memory_store():
    """
    >>> add_val, times_seen = memory_store()
    >>> add_val(4)
    >>> for _ in range(3):
    ...     add_val(3)
    >>> times_seen(3)
    3
    >>> times_seen(4)
    1
    >>> times_seen(2)
    0
    >>> add_val(3)
    >>> times_seen(3)
    4
    """
    memory = {}
    def add_val(var1):
        if var1 in memory:

            ____________________________________________
        else:

            ____________________________________________

    def times_seen(var1):

        return ____________________________________________
    return add_val, times_seen


def memory_store():
    """
    >>> add_val, times_seen = memory_store()
    >>> add_val(4)
    >>> for _ in range(3):
    ...     add_val(3)
    >>> times_seen(3)
    3
    >>> times_seen(4)
    1
    >>> times_seen(2)
    0
    >>> add_val(3)
    >>> times_seen(3)
    4
```

```
"""
memory = {}
def add_val(var1):
    if var1 in memory:
        memory[var1] = memory[var1] + 1
    else:
        memory[var1] = 1
def times_seen(var1):
    return memory.get(var1, 0)
return add_val, times_seen
```

3 Efficiency

1. What is the runtime of the following function?

```
def mystery(n):
    for i in range(10000):
        print(n)
```

Constant, because we run 10,000 operations no matter the size of `n`

2. What is the runtime of the following function?

```
def mystery(n):
    a = 0
    for i in range(n):
        for j in range(i, n):
            a += 1
```

Quadratic because we are doing $\sim n \cdot n$ operations.

3. What is the runtime of the following function?

```
def mystery(n):
    i = 1
    while n:
        i = i * 3
        n = n // 3
    return i
```

Logarithmic because n must triple to increase the number of operations by 1.

4 Iterators and Generators

1. **Yield Fibonacci!** (Fa20 MT2 Q2a)

   Implement `fibs`, a generator function that takes a one-argument pure function `f` and yields all Fibonacci numbers `x` for which `f(x)` returns a true value.

   The Fibonacci numbers begin with `0` and then `1`. Each subsequent Fibonacci number is the sum of the previous two. Yield the Fibonacci numbers in order.

```
def fibs(f):
    """Yield all Fibonacci numbers x for which f(x) is a true value.
    >>> odds = fibs(lambda x: x % 2 == 1)
    >>> [next(odds) for i in range(10)]
    [1, 1, 3, 5, 13, 21, 55, 89, 233, 377]
    >>> bigs = fibs(lambda x: x > 20)
    >>> [next(bigs) for i in range(10)]
    [21, 34, 55, 89, 144, 233, 377, 610, 987, 1597]
    >>> evens = fibs(lambda x: x % 2 == 0)
    >>> [next(evens) for i in range(10)]
    [0, 2, 8, 34, 144, 610, 2584, 10946, 46368, 196418]
    """

    n, m = 0, 1

    while _____________________________________________:

        if _____________________________________________:

            _____________________________________________

        _____________________________________________
```

```
def fibs(f):
    """Yield all Fibonacci numbers x for which f(x) is a true value.
    >>> odds = fibs(lambda x: x % 2 == 1)
    >>> [next(odds) for i in range(10)]
    [1, 1, 3, 5, 13, 21, 55, 89, 233, 377]
    >>> bigs = fibs(lambda x: x > 20)
    >>> [next(bigs) for i in range(10)]
    [21, 34, 55, 89, 144, 233, 377, 610, 987, 1597]
    >>> evens = fibs(lambda x: x % 2 == 0)
    >>> [next(evens) for i in range(10)]
    [0, 2, 8, 34, 144, 610, 2584, 10946, 46368, 196418]
    """

    n, m = 0, 1
    while True:
        if f(n):
            yield n
        n, m = m, n + m
```

2. **All Links** (Su21 Final Q6a)

Implement a generator function `all_links`, which takes in nums, a list of equal-length lists. `all_links` should yield all linked lists `s` that can be constructed such that the first element of `s` is the first element of one of the lists in `nums`, the second element of `s` is the second element of one of the lists in `nums`, and so on. Lists can be yielded in any order. You can assume that `nums` is a non-empty list.

For example, for the second doctest `all_links([[0, 2], [1, 3]])`, there are four total linked lists we should yield. `Link(0, Link(2))` is generated from using the first element from the first list and the second element from the first list. `Link(0, Link(3))` is generated with the first element from the first list and the second element from the second list. `Link(1, Link(2))` and `Link(1, Link(3))` get the first element from the second list and the second element from the first and second lists respectively.

```
def all_links(nums):
    """
    >>> list(all_links([[0], [1], [2]]))
    [Link(0), Link(1), Link(2)]
    >>> list(all_links([[0, 2], [1, 3]]))
    [Link(0, Link(2)), Link(0, Link(3)), Link(1, Link(2)), Link(1,
       Link(3))]
    """

    if len(nums[0]) == 0:

        ____________________________________________

    else:
        rests = [_____________________________________________ for x in nums]
        for first in [x[0] for x in nums]:

            for item in _____________________________________________:

                ____________________________________________
```

```
def all_links(nums):
    """
    >>> list(all_links([[0], [1], [2]]))
    [Link(0), Link(1), Link(2)]
    >>> list(all_links([[0, 2], [1, 3]]))
    [Link(0, Link(2)), Link(0, Link(3)), Link(1, Link(2)), Link(1,
       Link(3))]
    """

    if len(nums[0]) == 0:
        yield Link.empty
    else:
        rests = [x[1:] for x in nums]
        for first in [x[0] for x in nums]:
            for item in all_links(rests):
```

```
            yield Link(first, item)
```

5 Object-oriented Programming

1. **To-Do Lists** (Fa19 Final Q5)

   Implement the `TodoList` and `Todo` classes. When a `Todo` is complete, it is removed from all the `TodoList` instances to which it was ever added. Track both the number of completed `Todo` instances in each list and overall so that printing a `TodoList` instance matches the behavior of the doctests below. Assume the complete method of a `Todo` instance is never invoked more than once.

```
class TodoList:
    """A to-do list that tracks the number of completed items in the list
       and overall.

    >>> a, b = TodoList(), TodoList()
    >>> a.add(Todo('Laundry'))
    >>> t = Todo('Shopping')
    >>> a.add(t)
    >>> b.add(t)
    >>> print(a)
    Remaining: ['Laundry', 'Shopping'] ; Completed in list: 0 ; Completed
       overall: 0
    >>> print(b)
    Remaining: ['Shopping'] ; Completed in list: 0 ; Completed overall: 0
    >>> t.complete()
    >>> print(a)
    Remaining: ['Laundry'] ; Completed in list: 1 ; Completed overall: 1
    >>> print(b)
    Remaining: [] ; Completed in list: 1 ; Completed overall: 1
    >>> Todo('Homework').complete()
    >>> print(a)
    Remaining: ['Laundry'] ; Completed in list: 1 ; Completed overall: 2
    """

    def __init__(self):
        self.items, self.complete = [], 0
    def add(self, item):
        self.items.append(item)

        ____________________________________________________________
    def remove(self, item):

        ___________________________________________ += 1

        self.items.remove(_____________________________________________________)
    def __str__(self):

        return ('Remaining: ' +
            str(_____________________________________________________) +
             ' ; Completed in list: ' + str(self.complete) +

             ' ; Completed overall: ' + str(_____________________________________))
class Todo:
    done = 0
    def __init__(self, task):
        self.task, self.lists = task, []
    def complete(self):

        ___________________________________________ += 1
        for t in self.lists:
        t.remove(self)
```

```
class TodoList:
    """A to-do list that tracks the number of completed items in the list
       and overall.

    >>> a, b = TodoList(), TodoList()
    >>> a.add(Todo('Laundry'))
    >>> t = Todo('Shopping')
    >>> a.add(t)
    >>> b.add(t)
    >>> print(a)
    Remaining: ['Laundry', 'Shopping'] ; Completed in list: 0 ; Completed
       overall: 0
    >>> print(b)
    Remaining: ['Shopping'] ; Completed in list: 0 ; Completed overall: 0
    >>> t.complete()
    >>> print(a)
    Remaining: ['Laundry'] ; Completed in list: 1 ; Completed overall: 1
    >>> print(b)
    Remaining: [] ; Completed in list: 1 ; Completed overall: 1
    >>> Todo('Homework').complete()
    >>> print(a)
    Remaining: ['Laundry'] ; Completed in list: 1 ; Completed overall: 2
    """

    def __init__(self):
        self.items, self.complete = [], 0
    def add(self, item):
        self.items.append(item)
        items.lists.append(self)
    def remove(self, item):
        self.complete += 1
        self.items.remove(item)
    def __str__(self):
        return ('Remaining: ' + str([t.task for t in self.items]) +
            ' ; Completed in list: ' + str(self.complete) +
            ' ; Completed overall: ' + str(Todo.done))
class Todo:
    done = 0
    def __init__(self, task):
        self.task, self.lists = task, []
    def complete(self):
        Todo.done += 1
        for t in self.lists:
            t.remove(self)
```

2. **Midterm Elections** (Fa18 MT2 Q5a)

Implement the `Poll` class and the `tally` function, which takes a choice `c` and returns a list describing the number of votes for `c`. This list contains pairs, each with a name and the number of times `vote` was called on that choice at the `Poll` with that name. Pairs can be in any order. Assume all `Poll` instances have distinct names. Hint: the dictionary `get(key, default)` method (MT 2 guide, page 1 top-right) returns the value for a key if it appears in the dictionary and default otherwise.

```
class Poll:
    s = []
    def __init__(self, n):

        self.name = ____________________________________________
        self.votes = {}


        ____________________________________________________
    def vote(self, choice):

        self._____________________________ = ____________________________________
    def tally(c):
        """Tally all votes for a choice c as a list of (poll name, vote
           count) pairs.
        >>> a, b, c = Poll('A'), Poll('B'), Poll('C')
        >>> c.vote('dog')
        >>> a.vote('dog')
        >>> a.vote('cat')
        >>> b.vote('cat')
        >>> a.vote('dog')
        >>> tally('dog')
        [('A', 2), ('C', 1)]
        >>> tally('cat')
        [('A', 1), ('B', 1)]
        """

        return ____________________________________________________
```

```
class Poll:
    s = []
    def __init__(self, n):
        self.name = N
        self.votes = {}
        Poll.s.append(self)
    def vote(self, choice):
        self.votes[choice] = self.votes.get(choice, 0) + 1
    def tally(c):
        """Tally all votes for a choice c as a list of (poll name, vote
           count) pairs.
        >>> a, b, c = Poll('A'), Poll('B'), Poll('C')
        >>> c.vote('dog')
        >>> a.vote('dog')
        >>> a.vote('cat')
```

```
>>> b.vote('cat')
>>> a.vote('dog')
>>> tally('dog')
[('A', 2), ('C', 1)]
>>> tally('cat')
[('A', 1), ('B', 1)]
"""

return [(p.name, p.votes[c]) for p in Poll.s if c in p.votes]
```

1. **Filter Index** (Fa20 MT2 Q1)

   Definition. For a linked list `s`, the index of an element is the number of times `rest` appears in the smallest dot expression containing only `s`, `rest`, and `first` that evaluates to that element. For example, in the linked list `s = Link(5, Link(7, Link(9, Link(11))))`,

   - The index of 5 `(s.first)` is `0`.
   - The index of 7 `(s.rest.first)` is `1`.
   - The index of 11 `(s.rest.rest.rest.first)` is `3`.

   Implement `filter_index`, a function that takes a one-argument pure function `f` and a `Link` instance `s`. It returns a `Link` containing all elements of `s` that have an index `i` for which `f(i)` returns a true value.

   Assume that `s` is a finite linked list of numbers that contains no repeated elements. The `Link` class appears on Page 2 (left column) of the Midterm 2 Study Guide.

```
def filter_index(f, s):
    """Return a Link containing the elements of Link s that have an index
       i for
    which f(i) is a true value.

    >>> powers = Link(1, Link(2, Link(4, Link(8, Link(16, Link(32))))))
    >>> filter_index(lambda x: x < 4, powers)
    Link(1, Link(2, Link(4, Link(8))))
    >>> filter_index(lambda x: x % 2 == 1, powers)
    Link(2, Link(8, Link(32)))
    """

    def helper(i, s):
        if s is Link.empty:
            return s

        filtered_rest = ____________________________________________

    if ____________________________________________:

        return ____________________________________________
    else:
        return filtered_rest

    return ____________________________________________
```

```
def filter_index(f, s):
    """Return a Link containing the elements of Link s that have an index
       i for
    which f(i) is a true value.

    >>> powers = Link(1, Link(2, Link(4, Link(8, Link(16, Link(32))))))
```

```
    >>> filter_index(lambda x: x < 4, powers)
    Link(1, Link(2, Link(4, Link(8))))
    >>> filter_index(lambda x: x % 2 == 1, powers)
    Link(2, Link(8, Link(32)))
    """

    def helper(i, s):
        if s is Link.empty:
            return s
        filtered_rest = helper(i + 1, s.rest)
    if f(i):
        return Link(s.first, filtered_rest)
    else:
        return filtered_rest
    return helper(0, s)
```

2. **Combine Two** (Unknown source)

   Implement `combine_two`, which takes in `lnk`, a linked list of integers, and a two-argument function `fn`. Return a new linked list with every two elements from `lnk` combined with `fn`.

```
def combine_two(lnk, fn):
    >>> lnk1 = Link(1, Link(2, Link(3)))
    >>> combine_two(lnk1, add)
    Link(3, Link(3))
    >>> link2 = Link(4, lnk1)
    >>> combine_two(lnk2, mul)
    Link(4, Link(6))

    if _____________________________________________________:

        return _____________________________________________

    elif _____________________________________________________:

        return _____________________________________________

    combined = _____________________________________________

    return _____________________________________________________
```

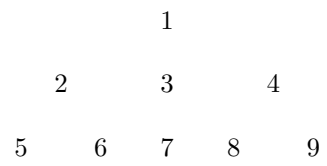
```
def combine_two(lnk, fn):
    >>> lnk1 = Link(1, Link(2, Link(3)))
    >>> combine_two(lnk1, add)
    Link(3, Link(3))
    >>> link2 = Link(4, lnk1)
    >>> combine_two(lnk2, mul)
    Link(4, Link(6))

    if lnk is Link.empty:
        return Link.empty
```

```
    elif lnk.rest is Link.empty:
        return Link(lnk.first)
    combined = fn(lnk.first, lnk.rest.first)
    return Link(combined, combine_two(lnk.rest.rest, fn))
```

7 Trees

1. **Level-Headed Trees** (Fa17 Final Q5a)

   A *level-order traversal* of a tree, `T`, traverses the root of `T` (level 0), then the roots of all the branches of `T` (level 1) left to right, then all the roots of the branches of the nodes traversed in level 1, (level 2) and so forth. Thus, a level-order traversal of the tree

   visits nodes with labels 1, 2, 3, 4, 5, 6, 7, 8, 9 in that order.

   Fill in the following generator function to yield the labels of a given tree in level order. All trees are of the class `Tree`,1 defined on page 2 of the Midterm 2 Study Guide. The strategy is to use a helper function that yields nodes at one level, and then to call this function with increasing levels until a level does not yield any labels. You may not need all the lines.

```
def level_order(tree):
    """Generate all labels of tree in level order."""
    def one_level(tree, k):
        """Generate the labels of tree at level k."""

        if ____________________________________________:

            ____________________________________________
        else:
            for child in ___________________________________:

                ____________________________________________
    level, count = 0, True
    while count:
        count = 0
        ____________________________________________

        for label in ___________________________________:

            ____________________________________________

        ____________________________________________
```

```
def level_order(tree):
    """Generate all labels of tree in level order."""
    def one_level(tree, k):
        """Generate the labels of tree at level k."""

        if k == 0:
            yield tree.label
        else:
            for child in tree.branches:
                yield from one_level(child, k-1)
    level, count = 0, True
    while count:
        count = 0
        for label in one_level(tree, level):
            count += 1
            yield label
        level += 1
```

2. **Prune Tree** (Su17 Final Q5)

Implement `prune_tree` which takes in a `Tree t` and an integer `total` and mutates `t` so that the sum of each root-to-leaf path is at most `total`. Assume values are positive numbers and `t.root` $\leq$ `total`.

```
class Tree:
    """A mutable tree data type containing a root value and a list of
       branches."""

    def __init__(self, root, branches=[]):
        self.root = root
        self.branches = list(branches)
    def is_leaf(self):
        return not self.branches

def prune_tree(t, total):
    """Destructively prune the tree t so that the sum of each path from
       root-to-leaf is less
    than or equal to total. All values are positive numbers and t.root <=
       total.

    >>> t1 = Tree(1, [Tree(2, [Tree(2, [Tree(1)]),
                               Tree(3),
                               Tree(4)]),
                   Tree(3, [Tree(2), Tree(1, [Tree(5), Tree(1)])]),
                   Tree(6, [Tree(2)])])
    >>> prune_tree(t1, 6)
    >>> print_tree(t1)
    1
        2
            2
                1
            3
        3
            2
            1
                1
    """
t.branches = _____________________________________________

_____________________________________________________________

    _____________________________________________________________
```

```
class Tree:
    """A mutable tree data type containing a root value and a list of
       branches."""

    def __init__(self, root, branches=[]):
        self.root = root
        self.branches = list(branches)
```

```
    def is_leaf(self):
        return not self.branches

def prune_tree(t, total):
"""Destructively prune the tree t so that the sum of each path from
   root-to-leaf is less
than or equal to total. All values are positive numbers and t.root <=
   total.

>>> t1 = Tree(1, [Tree(2, [Tree(2, [Tree(1)]),
                           Tree(3),
                           Tree(4)]),
               Tree(3, [Tree(2), Tree(1, [Tree(5), Tree(1)])]),
               Tree(6, [Tree(2)])])
>>> prune_tree(t1, 6)
>>> print_tree(t1)
1
    2
        2
            1
        3
    3
        2
        1
            1
"""
t.branches = [b for b in t.branches if t.label + b.label <= total]
for b in t.branches;
    prune_tree(b, total - t.label)
```