CS61A NOTE3 Recursion

Recursion

Still follows the 3 steps of recursion:

1. Base Case(s)
2. Reducing the problem `fib(n - 1), fib(n - 2)`
3. Connecting it together (+)

Common combiners: +, -, /, *; max, min, etc.

lab10 Q1: Subsequences

A subsequence of a sequence `S` is a subset of elements from `S`, in the same order they appear in `S`. Consider the list `[1, 2, 3]`. Here are a few of it's subsequences `[]`, `[1, 3]`, `[2]`, and `[1, 2, 3]`.

Write a function that takes in a list and returns all possible subsequences of that list. The subsequences should be returned as a list of lists, where each nested list is a subsequence of the original input.

In order to accomplish this, you might first want to write a function `insert_into_all` that takes an item and a list of lists, adds the item to the beginning of each nested list, and returns the resulting list.

纯文本

```
def insert_into_all(item, nested_list):
    """Return a new list consisting of all the lists in nested_list,
    but with item added to the front of each. You can assume that
     nested_list is a list of lists.

    >>> nl = [[], [1, 2], [3]]
    >>> insert_into_all(0, nl)
    [[0], [0, 1, 2], [0, 3]]
    """
    "*** YOUR CODE HERE ***"
    return [[item]+i for i in nested_list]

def subseqs(s):
    """Return a nested list (a list of lists) of all subsequences of S.
```

```
    The subsequences can appear in any order. You can assume S is a list.

    >>> seqs = subseqs([1, 2, 3])
    >>> sorted(seqs)
    [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]]
    >>> subseqs([])
    [[]]
    """
    if s==[]:
        return [[]]
    else:
        k=subseqs(s[1:])
        return k+insert_into_all(s[0],k)
```

lab10 Q2: Non-Decreasing Subsequences

Just like the last question, we want to write a function that takes a list and returns a list of lists, where each individual list is a subsequence of the original input.

This time we have another condition: we only want the subsequences for which consecutive elements are *nondecreasing*. For example, `[1, 3, 2]` is a subsequence of `[1, 3, 2, 4]`, but since 2 < 3, this subsequence would *not* be included in our result.

You may assume that the list passed in as `s` contains only nonnegative elements.

Fill in the blanks to complete the implementation of the `non_decrease_subseqs` function. You may assume that the input list contains no negative elements.You may use the provided helper function `insert_into_all`, which takes in an `item` and a list of lists and inserts the `item` to the front of each list.

纯文本

```
def non_decrease_subseqs(s):
    """Assuming that S is a list, return a nested list of all subsequences
    of S (a list of lists) for which the elements of the subsequence
    are strictly nondecreasing. The subsequences can appear in any order.

    >>> seqs = non_decrease_subseqs([1, 3, 2])
    >>> sorted(seqs)
    [[], [1], [1, 2], [1, 3], [2], [3]]
```

```
    >>> non_decrease_subseqs([])
    [[]]
    >>> seqs2 = non_decrease_subseqs([1, 1, 2])
    >>> sorted(seqs2)
    [[], [1], [1], [1, 1], [1, 1, 2], [1, 2], [1, 2], [2]]
    """

    def subseq_helper(s, prev):
        if not s:
            return [[]]
        elif s[0] < prev: #没有s[0]
            return subseq_helper(s[1:], prev)
            #prev最小能达到的数,且只能用自己不能用father
        else:
            a = subseq_helper(s[1:], s[0])
            b = subseq_helper(s[1:], prev)
            return insert_into_all(s[0], a) + b
    return subseq_helper(s, 0)
```

disc Q4 : Is Prime 用 recursion 判断质数

You will need a helper function! Remember helper functions are nested functions that are useful if you need to keep track of more variables than the given parameters, or if you need to change the value of the input.需要 helper，用来跟踪给定参数的更多变量或需要改变输入的值

纯文本

```
def is_prime(n): #递归要传入新的i,并不改变自身的n,重新def函数
    def helper(i):
        if i>(n**0.5): #用了一点数学
            return True #先排除最后情况
        elif n%i==0: #过程中随时淘汰
            return False
        return helper(i+1) #如果前两个不跳出的话，去看下一个，千万别return True
    return helper(2) #什么时候return后不加数字什么时候加？
```

disc Q5: Recursive Hailstone 用递归

纯文本

```
def hailstone(n):  #n自身改变并打印
    print(n)
    if n==1:
        return 1
    elif n%2==0:
```

```
        return hailstone(n//2)
    else:
        return hailstone(n*3+1)
```

disc Q6: Merge Numbers 用递归以单减 merge 数字

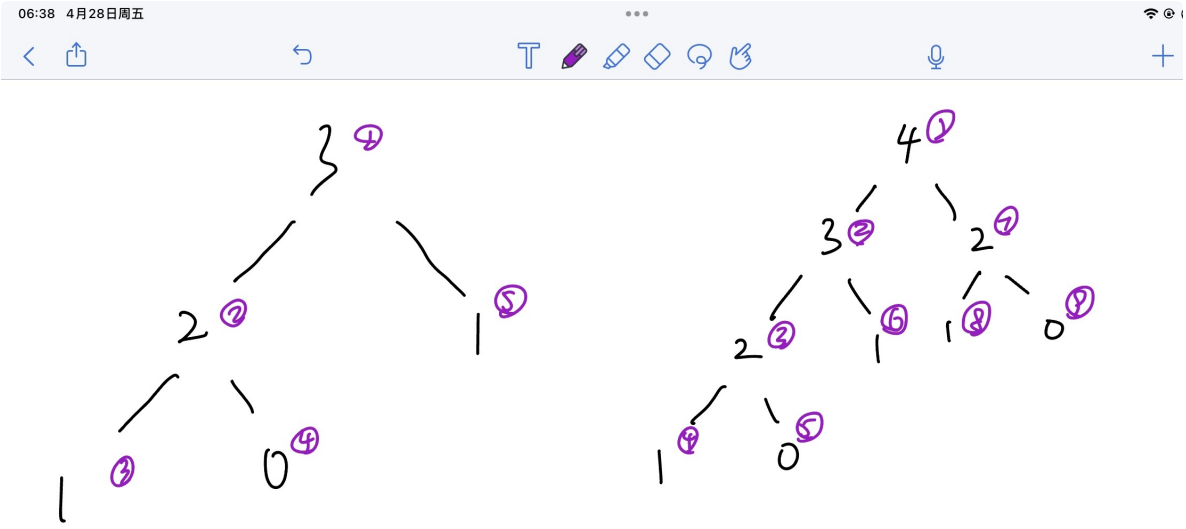
纯文本

```
def merge(n1, n2):
    """ Merges two numbers by digit in decreasing order
    >>> merge(31, 42)
    4321
    >>> merge(21, 0)
    21
    >>> merge (21, 31)
    3211
    """
    if n1==0:
        return n2
    elif n2==0:
        return n1
    elif n1%10 < n2%10:
        return merge(n1//10,n2)*10+n1%10
    elif n1%10 >= n2%10:
        return merge(n1,n2//10)*10+n2%10
```

Tree Recursion

A tree recursive function is a recursive function that makes more than one call to itself, resulting in a tree-like series of calls.

lab4 Q1: WWPD: Squared Virahanka Fibonacci



纯文本

```
>>> def virfib_sq(n):
...     print(n)
...     if n <= 1:
...         return n
...     return (virfib_sq(n - 1) + virfib_sq(n - 2)) ** 2
>>> r0 = virfib_sq(0)
0

>>> r3 = virfib_sq(3)
3
2
1
0
1

>>> r3
4

>>> (r1 + r2) ** 2
4

>>> r4 = virfib_sq(4)
4
3
2
1
0
1
```

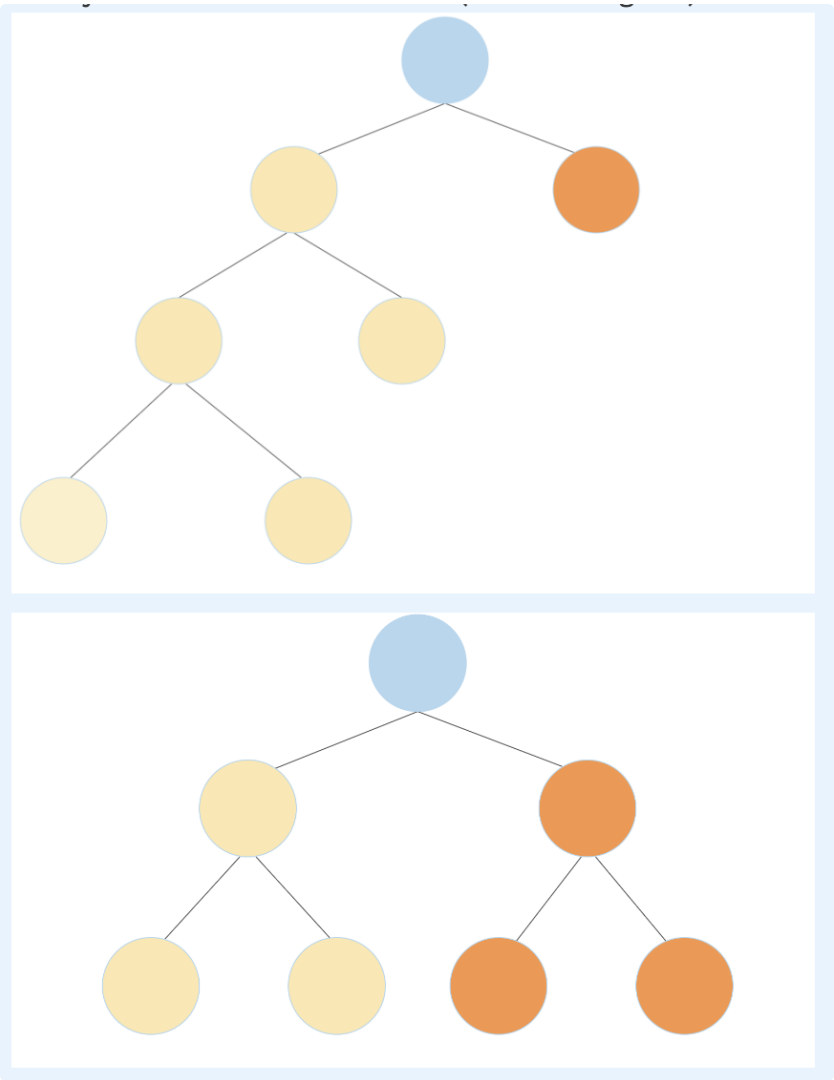
```
2
1
0

>>> r4
25
```

lab10 Q3: Number of Trees

A **full binary tree** is a tree where each node has either 2 branches or 0 branches, but never 1 branch.

Write a function which returns the number of unique full binary tree structures that have exactly n leaves. See the doctests for visualizations of the possible full binary tree sturctures that have 1, 2, and 3 leaves.

Hint: A full binary tree can be constructed by connecting two smaller full binary trees to a root node. If the two smaller full binary trees have `a` and `b` leaves, the new full binary tree will have `a + b` leaves. For example, as shown in the first diagram below, a full binary tree with 4 leaves can be constructed by connecting a full binary tree that has three leaves (yellow) with a full binary tree that has one leaf (orange). A full binary tree with 4 leaves can also be constructed by connecting two full binary trees with 2 leaves each (second diagram)两个叶子 a 和 b 的子树加一个顶点就是新的要的 a+b 叶子树



```
def num_trees(n):
    """Returns the number of unique full binary trees with exactly n leaves

    1   2       3       3    ...
    *   *       *       *
       / \     / \     / \
      *   *   *   *   *   *
             / \         / \
            *   *       *   *

    >>> num_trees(1)
    1
    >>> num_trees(2)
    1
    >>> num_trees(3)
```

```
    2
    >>> num_trees(8)
    429

    """
    "*** YOUR CODE HERE ***"
    if n==1:
        return 1
    else:
        #return sum([num_trees(i)*num_trees(n-i) for i in range(1,n)])
        return sum([num_trees(i) * num_trees(n - i) for i in range(1, n)])
        #会重复 return num_trees(n-1)*n，根据hint提醒
```

disc Q1: Count Stair Ways

纯文本

```
def count_stair_ways(n):
    if n==1:
        return 1
    elif n==2:
        return 2
    else:
        return count_stair_ways(n-1)+count_stair_ways(n-2)
```

disc Q2:Count K 极其特殊的例子

Counts the number of paths up a flight of n stairs when taking up to 最多且包括 and including k steps at a time.计算每次上 n 个楼梯的路径数，每次最多包括 k 个步骤。

纯文本

```
def count_k(n, k):
    """>>> count_k(3, 3) # 3, 2 + 1, 1 + 2, 1 + 1 + 1
    4"""
    if n == 0 or n==1:
        return 1
    else:
        total = 0
        for i in range(1,k+1):
            if n-i >= 0:
                total += count_k(n-i, k) #以第一步分类
        return total
```

hw3 Q1: Num Eights

Write a recursive function `num_eights` that takes a positive integer `n` and returns the number of times the digit 8 appears in `n`.

纯文本

```
def num_eights(n):
    #返回8的个数
    if n% 10 == 8:
        return 1+num_eights(n//10)
    elif n<10 and n != 8: #其实可以不用加n != 8,注意顺序
        return 0
    else:
        return num_eights(n//10)
```

hw3 Q2: Ping-pong

The ping-pong sequence counts up starting from 1 and is always either counting up or counting down. At element `k`, the direction switches if `k` is a multiple of 8 or contains the digit 8 如果是 8 的倍数或者包含 8 则转换方向，index 恒加 1. The first 30 elements of the ping-pong sequence are listed below, with direction swaps marked using brackets at the 8th, 16th, 18th, 24th, and 28th elements:

纯文本

```
def pingpong(n):
    #Return the nth element of the ping-pong sequence.
    #复杂，helper里用k追踪方向
    #此外只能正着递归，倒着不知方向，另一个角度证实helper(result,1,1)

    def help(result,i,k): #i追踪位置，k代表加减，result得出结果
        if i == n:
            return result
        elif i % 8 ==0 or num_eights(i) > 0: #k转变方向，else里下一个k不用变
            return help(result - k, i + 1, -k) #return指向下一个result，因此上
        else:
            return help(result + k, i + 1, k)
    return help(1,1,1)
```

hw3 Q3: Count Coins

Write a **recursive** function `count_coins` that takes a positive integer `change` and returns the number of ways to make change for `change` using coins.

可用 next_larger_coin：1 到 5，5 到 10，10 到 25；

next_smaller_coin：5 到 1，10 到 5，25 到 10，略去此处代码

- 15 1-cent coins
- 10 1-cent, 1 5-cent coins
- 5 1-cent, 2 5-cent coins
- 5 1-cent, 1 10-cent coins
- 3 5-cent coins
- 1 5-cent, 1 10-cent coin

纯文本

```
def count_coins(change):
    #法一
    def helper(change,largest):
        if change==0:
            return 1 #⚠️
        elif change<0:
            return 0
        elif largest==None:
            return 0
        with_largest=helper(change-largest,largest) #要不是前面-，最大不必须
        without_largest=helper(change,next_smaller_coin(largest)) #next大不
        return with_largest+without_largest
    return helper(change,25)

    #法二
    def helper(change,smallest):
        if change<0:
             return 0
        elif change==0:
             return 1
        elif next_larger_coin(smallest)==None:
             return 0
        with_smallest=helper(change-smallest,smallest)
        without_smallest=helper(change,next_larger_coin(smallest))
        return with_smallest+without_smallest
    return helper(change,1)
```

Lab 4 Q2: Summation

Write a recursive implementation of `summation`, which takes a positive integer `n` and a function `term`. It applies `term` to every number from `1` to `n` including `n` and returns the sum.

```
def summation(n, term):
    if n==1:
        return term(n)
    else:
        return term(n)+summation(n-1, term)
```

Lab4 Q3: Insect Combinatorics

Consider an insect in an *M* by *N* grid. The insect starts at the bottom left corner, *(1, 1)*, and wants to end up at the top right corner, *(M, N)*. The insect is only capable of moving right or up. Write a function `paths` that takes a grid length and width and returns the number of different paths the insect can take from the start to the goal. (There is a closed-form solution to this problem, but try to answer it procedurally using recursion.)

```
def paths(m, n):
    if m==1 or n==1:
        return 1
    else:
        return paths(m-1, n)+paths(m, n-1)
```

Lab4 Q5: Pascal's Triangle

Pascal's triangle gives the coefficients of a binomial expansion; if you expand the expression `(a + b) ** n`, all coefficients will be found on the `n`th row of the triangle, and the coefficient of the `i`th term will be at the `i`th column.帕斯卡三角

```
>>> pascal(0, 5)  # Empty entry; outside
```

```
def pascal(row, column):
    if row<column: #超出
        return 0
    elif row==column or column==0: #边缘是1
        return 1
    else:
        return pascal(row-1,column-1)+pascal(row-1,column)
```

Lab4 Q6: Double Eights

Write a recursive function 是否有双 8

纯文本

```
def double_eights(n):
    if n%100==88:
        return True
    elif n<188:
        return False
    else:
        return double_eights(n//10)
    #
```