Recursion, Tree Recursion

Computer Science Mentors 61A

February 13–February 17, 2022

Tree Recursion vs Recursion

In most recursive problems we've seen so far, the solution function contains only one call to itself. However, some problems will require multiple recursive calls – we colloquially call this type of recursion "tree recursion," since the propagation of function frames reminds us of the branches of a tree. "Tree recursive" or not, these problems are still solved the same way as those requiring a single function call: a base case, the recursive leap of faith on a subproblem, and solving the original problem with the solution to our subproblems. The difference? We simply may need to use multiple subproblems to solve our original problem.

Tree recursion will often be needed when solving counting problems (how many ways are there of doing something?) and optimization problems (what is the maximum or minimum number of ways of doing something?), but remember there are all sorts of problems that may need multiple recursive calls! Always come back to the recursive leap of faith.

Two rules that are often useful in solving counting problems:

1. If there are x *ways* of doing something and y *ways* of doing another thing, there are xy *ways* of doing **both** at the same time.
2. If there are x *ways* of doing one thing and y *ways* of doing another, but we can't do both things at the same time, there are $x + y$ ways of doing either the first thing **or** the second thing.

Created by Alyssa Smith, Esther Shen, Maya Romero, Natalie Wei, Sandhya Ganesan, Kaelyn Huang, Cecilia Aiko, Aurelia Wang

1. The *Gibonacci sequence* is a recursively defined sequence of integers; we denote the nth Gibonacci number g_n. The first three terms of the sequence are $g_0 = 0, g_1 = 1, g_2 = 2$. For $n \geq 3$, g_n is defined as the sum of the previous three terms in the sequence.

   Complete the function `gib`, which takes in an integer `n` and returns the nth Gibonacci number, g_n. Also, identify the three parts of recursive function design as they are used in your solution.

```
def gib(n):
    """
    >>> gib(0)
    0
    >>> gib(1)
    1
    >>> gib(2)
    2
    >>> gib(3) # gib(2) + gib(1) + gib(0) = 3
    3
    >>> gib(4) # gib(3) + gib(2) + gib(1) = 6
    6
    """
    if ______________________________:

        return ______________________________

    return ______________________________
```

2. Including the original call, how many calls are made to `gib` when you evaluate `gib(5)`?

3. Implement a recursive `fizzbuzz`.

```
def fizzbuzz(n):
    """Prints the numbers from 1 to n. If the number is
       divisible by 3, it
    instead prints 'fizz'. If the number is divisible by 5,
       it instead prints
    'buzz'. If the number is divisible by both, it prints
       'fizzbuzz'.

    >>> fizzbuzz(15)
    1
    2
    fizz
    4
    buzz
    fizz
    7
    8
    fizz
    buzz
    11
    fizz
    13
    14
    fizzbuzz
    """
```

4. Write a function `selective_sum`, which takes in an integer `n` and a predicate function `cond`. `selective_sum` returns the sum of all positive integers up to `n` for which `cond(n)` is true.

```
def selective_sum(n, cond):
    """
    >>> is_odd = lambda x: x % 2 == 1
    >>> selective_sum(5, is_odd) # 5 + 3 + 1 = 9
    9
    >>> bigger_than_10 = lambda x: x > 10
    >>> selective_sum(13, bigger_than_10) # 13 + 12 + 11 = 36
    36
    >>> selective_sum(-1, is_odd) # no positive integers <= 1
    0
    """
    if ______________________________________________:

        return ______________________________________________

    if ______________________________________________:

        return ______________________________________________

    return ______________________________________________
```

5. In an alternate universe, 61A software is not that good (inconceivable!). Tyler is in charge of assigning students to discussion sections, but sections.cs61a.org only knows how to list sections with either `m` or `n` number of students (the two most popular sizes). Given a `total` number of students, can Tyler create sections with only sizes of either `m` or `n` and not have any leftover students? Return `True` if he can and `False` otherwise.

```
def fit_sections(total, n, m):
    """
    >>> fit_sections(1, 3, 5)
    False
    >>> fit_sections(5, 3, 5) # 0 * 3 + 1 * 5 = 5
    True
    >>> fit_sections(11, 3, 5) # 2 * 3 + 1 * 5 = 11
    True
    >>> fit_sections(61, 11, 15) # can't express 61 as a
       * 11 + b * 15
    False
    """
    if ________________________________________________:

        return True

    elif
       _________________________________________________:

        return False

    return ____________________________________________
```

6. Mario needs to get from one end of a level to the other, but there are deadly Piranha plants in his way! Mario only moves forward and can either *step* (move forward one space) or *jump* (move forward two spaces) from each position. A level is represented as a series of ones and zeros, with zeros denoting the location of Piranha plants. Mario can step on ones but not on zeros. How many different ways can Mario traverse a level without stepping or jumping into a Piranha plant? Assume that every level begins with a 1 (where Mario starts) and ends with a 1 (where Mario must end up).

   Hint: Does it matter whether Mario goes from left to right or right to left? Which one is easier to check?

```
def mario_number(level):
    """
    >>> mario_number(10101)
    1
    >>> mario_number(11101)
    2
    >>> mario_number(100101)
    0
    """
    if ________________________:

        _______________________

    elif ______________________:

        _______________________

    else:

        _____________________________________________________
```

7. Fill in `collapse`, which takes in a non-negative integer `n` and returns the number resulting from removing all digits that are equal to an adjacent digit, i.e. the number has no adjacent digits that are the same.

```
def collapse(n):
    """
    >>> collapse(12234441)
    12341
    >>> collapse(11200000013333)
    12013
    """
    rest, last = n // 10, n % 10

    if ______________________________________:

        _________________________________________

    elif ____________________________________:

        _________________________________________
    else:

        _________________________________________
```