Tree Recursion

Announcements

Recursion Review

Effective Recursion

How to Know That a Recursive Case is Implemented Correctly?

Tracing: Diagram the whole computational process (only feasible for very small examples)

Induction: Check that f(n) is correct as long as f(n-1) **...** f(0) are.
(*This the recursive leap of faith.*) **(Functional Abstraction!)**

How to Get Started on a Recursive Function?

Start with the recursive case: Come up with a process that involves recursion by writing down calls you might make, what they would return, and how to use those return values.

Base case(s): Once you have the recursive case, you can trace to see where it leads.

What else helps?

Convert from iteration: If the function is easy for you to write using **while**, then you could convert it later to recursion.

Describe the recursion: Start with a description of the recursive case before writing code.

Discussion Review: Hailstone

- If **n** is even, divide it by 2
- If **n** is odd, multiply it by 3 and add 1
- Repeat until **n** is 1.
- Print out the values and return the number of steps

```
>>> a = hailstone(10)
10
5
16
8
4
2
1
>>> a
7
```

```
def hailstone(n):
    """Print out the hailstone sequence
    starting at n, and return the number of
    elements in the sequence."""
    print(n)
    if n % 2 == 0:
        return even(n)
    else:
        return odd(n)

def even(n):
    return hailstone(n // 2) + 1

def odd(n):
    if n == 1:
        return 1
    return hailstone(n * 3 + 1) + 1
```

Spring 2024 Midterm 1 Question 4(e)

Definition. A *dice integer* is a positive integer whose digits are all from 1 to 6.

```
def streak(n):
    """Return whether positive n is a dice integer in which all the digits are the same.

    >>> streak(22222)
    True
    >>> streak(4)
    True
    >>> streak(22322)  # 2 and 3 are different digits.
    False
    >>> streak(99999)  # 9 is not allowed in a dice integer.
    False
    """
    return (n >= 1 and n <= 6) or (n > 9 and n % 10 == n // 10 % 10 and streak(n // 10))
```

Come up with a process: how can we use the result of streak on a smaller input?

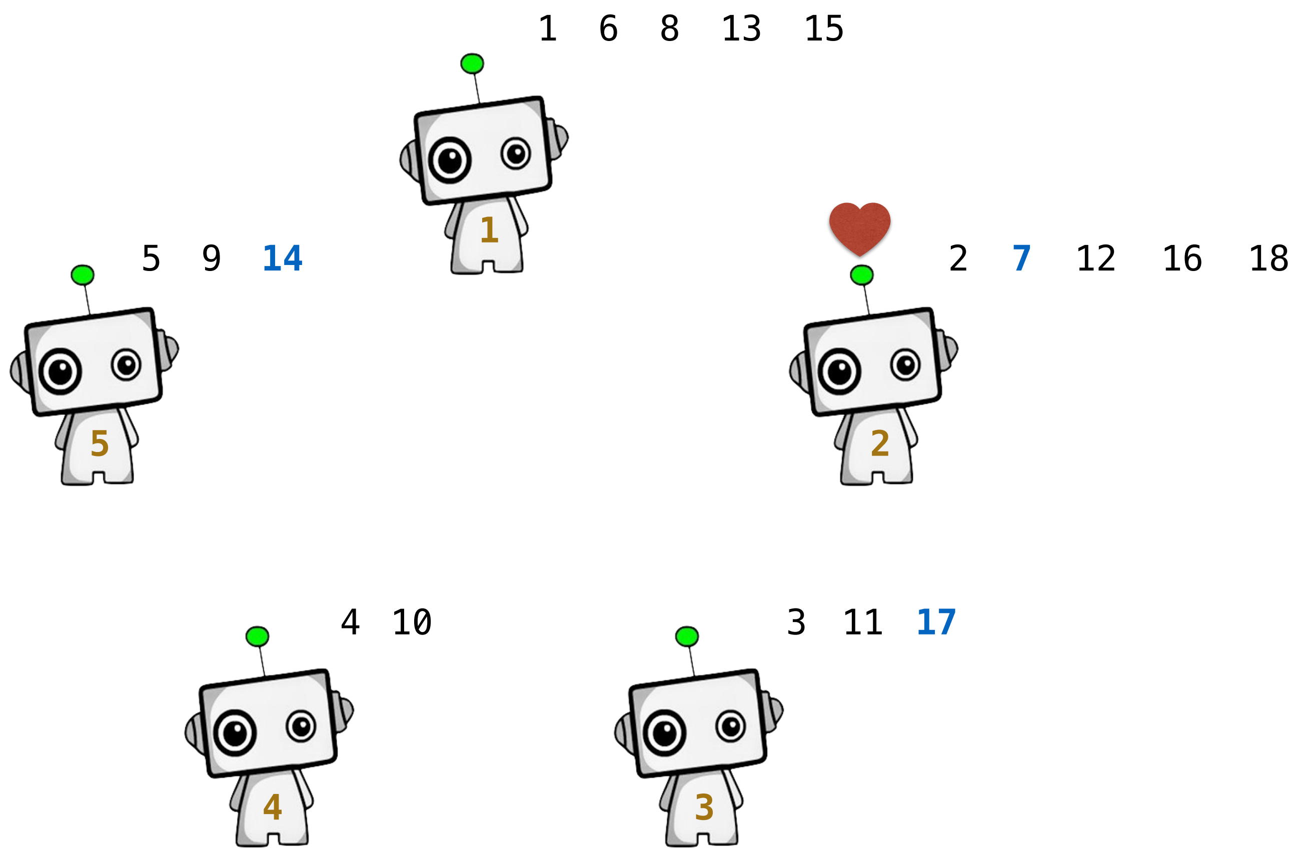
Consider example recursive calls you might make: streak(2222), streak(2), ...

A Process: In a streak, every digit except the last is a streak, and the last matches

https://pythontutor.com/cp/composingprograms.html#code=def%20streak%28n%29%3A%0A%20%20%20%20return%20%28n%20%3E%3D%201%20and%20n%20%3C%3D%206%29%20or%20%28n%20%3E%209%20and%20n%20%25%2010%20%3D%3D%20n%20//%2010%20%25%2010%20and%20streak%28n%20//%2010%29%29%0A%0Astreak%2822222%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Discussion Review: Sevens

Players in a circle count up from 1 in the clockwise direction. If a number is divisible by 7 or contains a 7 (or both), switch directions. With 5 players, who says 18?



https://commons.wikimedia.org/wiki/File:Mini-Robot.png; originally from http://kryptonite-bot.ml

The Game of Sevens

Players in a circle count up from 1 in the clockwise direction. If a number is divisible by 7 or contains a 7 (or both), switch directions. If someone says a number when it's not their turn or someone misses the beat on their turn, the game ends.

Implement sevens(n, k) which returns the position of who says n among k players.

1. Pick an example input and corresponding output.
2. Describe a process (in English) that computes the output from the input using simple steps.
3. Figure out what additional names you'll need to carry out this process.
4. Implement the process in code using those additional names.

```
n:         the final number
k:         how many players
i:         the current number
who:       the current player
direction: who's next
```

(Demo)

Mutual Recursion

Mutually Recursive Functions

Two functions f and g are mutually recursive if f calls g and g calls f.

```
def unique_prime_factors(n):
    """Return the number of unique prime factors of n.

    >>> unique_prime_factors(51)  # 3 * 17
    2
    >>> unique_prime_factors(9)   # 3 * 3
    1
    >>> unique_prime_factors(72000) = 2 * 2 * 2 * 2 * 2 * 2 * 3 * 3 * 5 * 5 * 5
    3
    """
    k = smallest_factor(n)
    def no_k(n):
        "Return the number of unique prime factors of n other than k."
        if n == 1:
            return 0
        elif n % k != 0:
            return unique_prime_factors(n)
        else:
            return no_k(n // k)
    return 1 + no_k(n)
```

```
def smallest_factor(n):
    "The smallest divisor of n above 1."
```

Find the smallest (prime) factor

Keep removing that factor until only other factors are left

And then count the prime factors of that

count	what remains
0	72000
1 (k = 2)	1125
2 (k = 3)	125
3 (k = 5)	1

Tree Recursion

Counting Partitions

```
The number of partitions of a positive integer n, using parts up to size m, is the number
of ways in which n can be expressed as the sum of positive integer parts up to m in
increasing order.
```

`count_partitions(6, 4)`

~~1 + 5 = 6~~ 5 > m (4)

~~4 + 2 = 6~~ Decreasing order

2 + 4 = 6

1 + 1 + 4 = 6

3 + 3 = 6

1 + 2 + 3 = 6

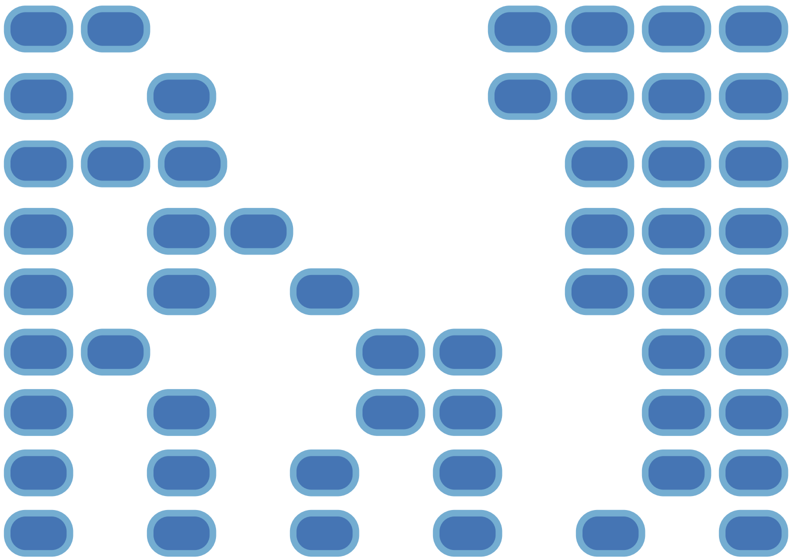
1 + 1 + 1 + 3 = 6

2 + 2 + 2 = 6

1 + 1 + 2 + 2 = 6

1 + 1 + 1 + 1 + 2 = 6

1 + 1 + 1 + 1 + 1 + 1 = 6

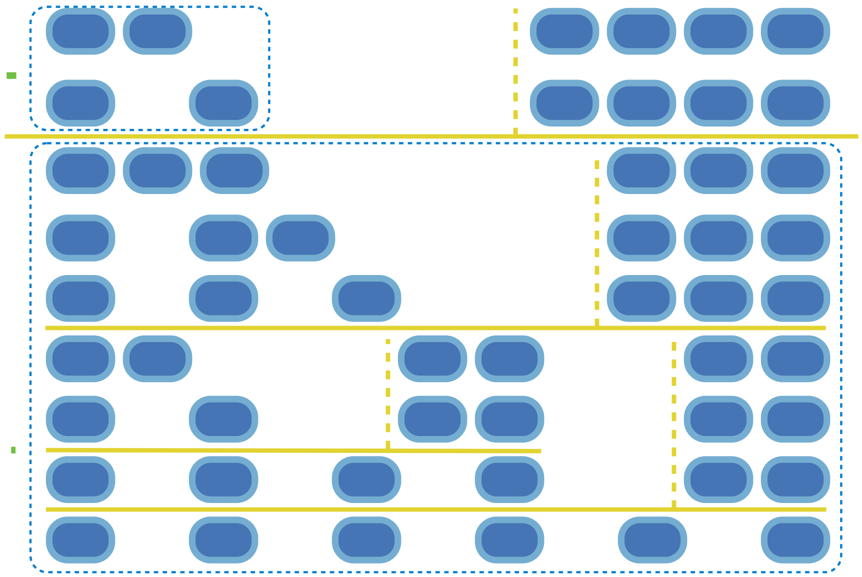


Counting Partitions

The number of partitions of a positive integer n, using parts up to size m, is the number of ways in which n can be expressed as the sum of positive integer parts up to m in non-decreasing order.

count_partitions(6, 4)

- Recursive decomposition: finding simpler instances of the problem.
- Explore two possibilities:
  - Use at least one 4
  - Don't use any 4
- Solve two simpler problems:
  - count_partitions(2, 4)
  - count_partitions(6, 3)
- Tree recursion often involves exploring different choices.



Counting Partitions

The number of partitions of a positive integer n, using parts up to size m, is the number of ways in which n can be expressed as the sum of positive integer parts up to m in increasing order.

- Recursive decomposition: finding simpler instances of the problem.
- Explore two possibilities:
  - Use at least one 4
  - Don't use any 4
- Solve two simpler problems:
  - count_partitions(2, 4)
  - count_partitions(6, 3)
- Tree recursion often involves exploring different choices.

```python
def count_partitions(n, m):
    if n == 0:
        return 1 # We found a way
    elif n < 0:
        return 0 # We subtracted too much
    elif m == 0:
        return 0 # We ran out of numbers

    else:
        with_m = count_partitions(n-m, m)
        without_m = count_partitions(n, m-1)
        return with_m + without_m
```

(Demo)

pythontutor.com/composingprograms.html#code=def%20count_partitions%28n,%20m%29%3A%0A%20%20%20%20if%20n%20%3D%3D%200%3A%0A%20%20%20%20%20%20%20%20return%201%0A%20%20%20%20elif%20n%20<%200%3A%0A%20%20%20%20%20%20%20%20return%200%0A%20%20%20%20elif%20m%20%3D%3D%200%3A%0A%20%20%20%20%20%20%20%20return%200%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20with_m%20%3D%20count_partitions%28n-m,
%20m%29%20%0A%20%20%20%20%20%20%20%20without_m%20%3D%20count_partitions%28n,
%20m-1%29%0A%20%20%20%20%20%20%20%20return%20with_m%20%2B%20without_m%0A%20%20%20%20%20%20%20%20%0Aresult%20%3D%20count_partitions%285,%203%29%0A%0A#%201%20%2B%201%20%2B%201%20%2B%201%20%2B%201%20%3D%205%0A#%201%20%2B%201%20%2B%201%20%2B%202%20%2B%20%20%20%3D%205%0A#%201%20%2B%202%20%2B%202%20%2B%20%20%20%20%20%20%20%3D%205%0A#%201%20%2B%201%20%2B%203%20%2B%20%20%20%20%20%20%20%3D%205%0A
20%2B%203%20%2B%20%20%20%20%20%20%20%20%20%20%20%3D%205&mode=display&origin=composingprograms.js&cumulative=false&py=3&rawInputLstJSON=[]&curInstr=0