Tree Recursion

Announcements

Recursion Review

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.*)

Abstraction: Assume f behaves correctly (on simpler examples), then use it to implement f.

Spring 2024 Midterm 1 Question 4(e)

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

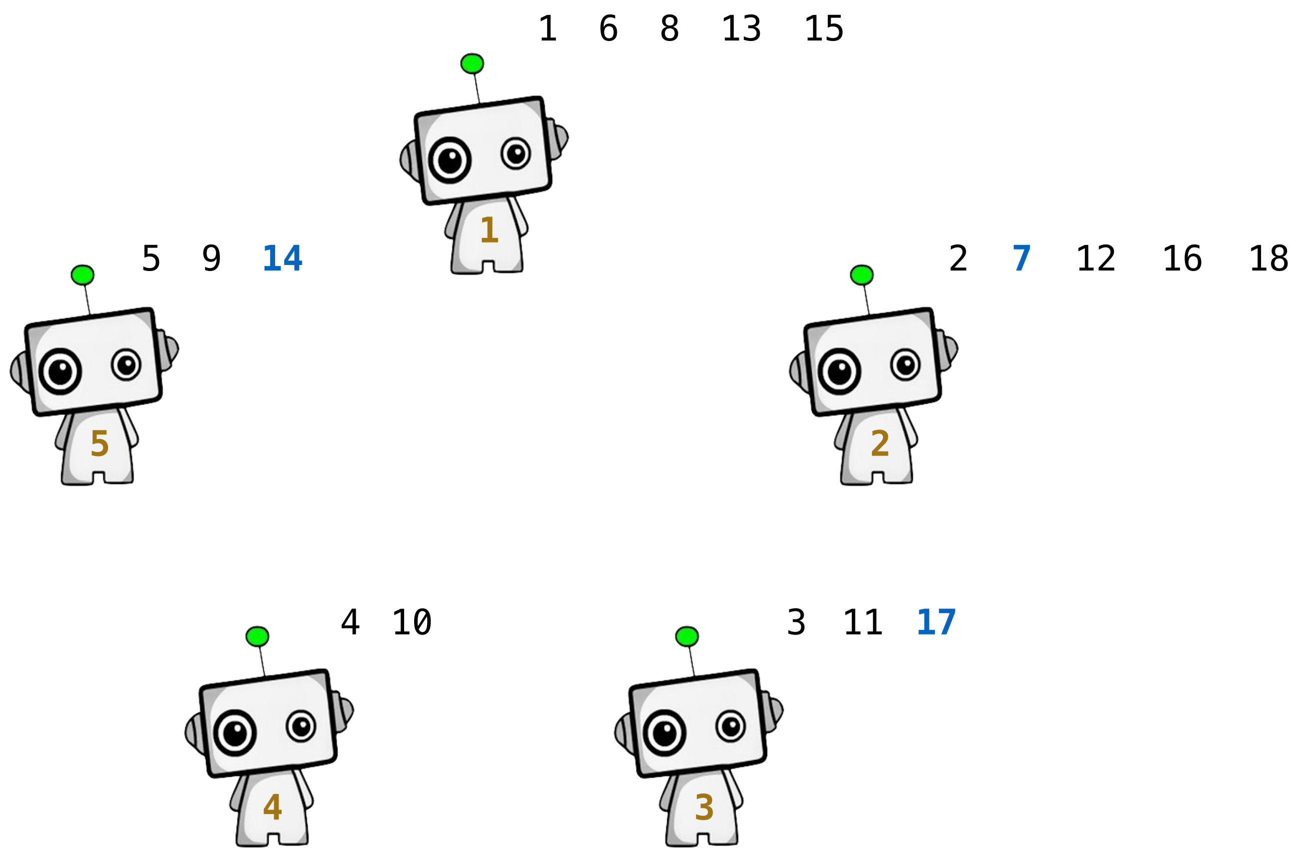
```
def streak(n):
    """Return whether positive n is a dice integer in which all the digits are the same.

    >>> 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))
```

Idea: In a streak, all pairs of adjacent digits are equal.

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(576) # 2 * 2 * 2 * 2 * 2 * 2 * 3 * 3
    2
    """
    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."
```

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)
```

2 + 4 = 6

1 + 1 + 4 = 6

3 + 3 = 6

1 + 2 + 3 = 6

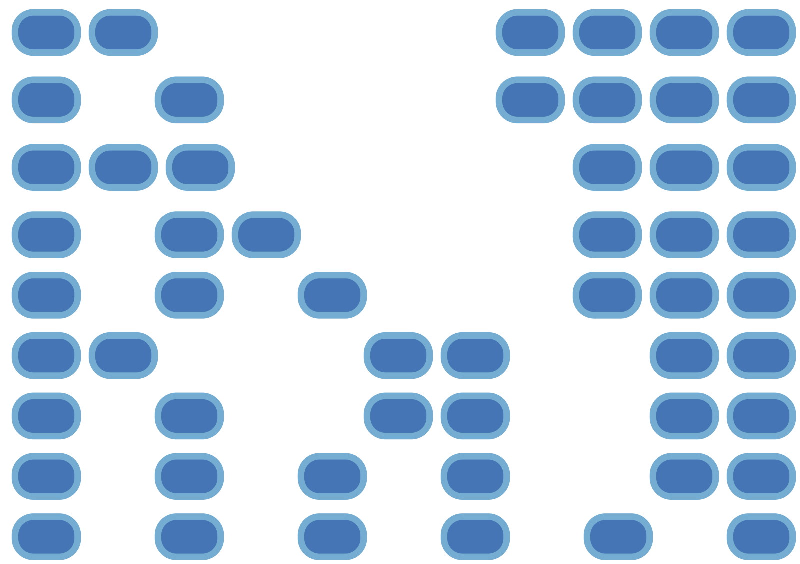
1 + 1 + 1 + 3 = 6

2 + 2 + 2 = 6

1 + 1 + 2 + 2 = 6

1 + 1 + 1 + 1 + 2 = 6

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

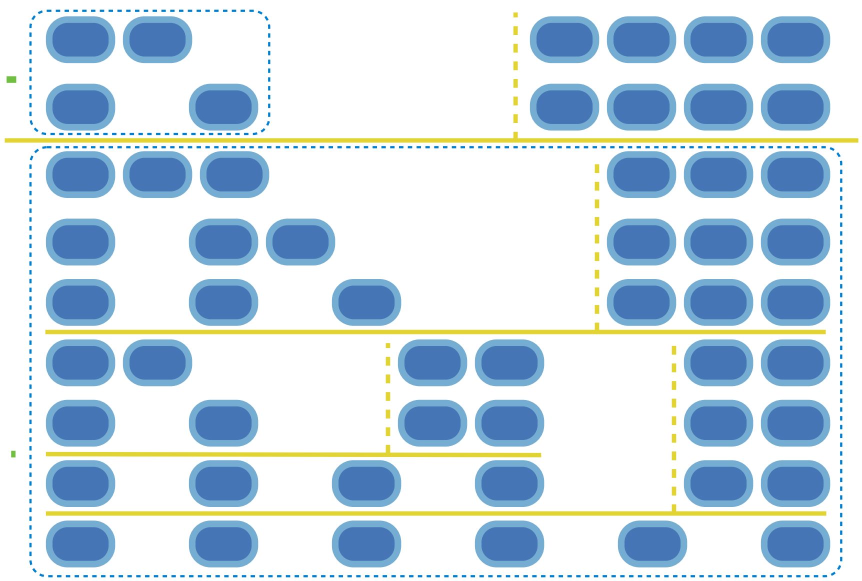


Counting Partitions

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.

```
def count_partitions(n, m):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0:
        return 0
    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%20%3D%205&mode=display&origin=composingprograms.js&cumulative=false&py=3&rawInputLstJSON=[]&curInstr=0

Spring 2023 Midterm 2 Question 5

Definition. When parking vehicles in a row, a motorcycle takes up 1 parking spot and a car takes up 2 adjacent parking spots. A string of length n can represent n adjacent parking spots using % for a motorcycle, <> for a car, and . for an empty spot.

For example: '.%%.<><>' (Thanks to the Berkeley Math Circle for introducing this question.)

Implement **count_park**, which returns the number of ways that vehicles can be parked in n adjacent parking spots for positive integer n. Some or all spots can be empty.

```python
def count_park(n):
    """Count the ways to park cars and motorcycles in n adjacent spots.
    >>> count_park(1)  # '.' or '%'
    2
    >>> count_park(2)  # '..', '.%', '%.', '%%', or '<>'
    5
    >>> count_park(4)  # some examples: '<><>', '.%%.', '%<>%', '%.<>'
    29
    """
    if n < 0:
        return 0
    elif n == 0:
        return 1
    else:
        return count_park(n-2) + count_park(n-1) + count_park(n-1)
```