CS 61A SU26

Lecture 06: Tree Recursion

June 30, 2026

Rebecca Dang

Join pollev.com/rebeccadang831 (or scan QR code) on your phone or laptop

We'll begin at Berkeley time (10 minutes after), as per tradition!

Potpourri

5 min

- Announcements
- Quiz 1 Poll
- Computing in the News

Announcements

- **Instructor Tea Hours Wed 11:30 am - 12:30 pm in Soda 795 (check Calendar before going)**
  - Like OH, but more casual
  - Only non-course questions (e.g. career, professional development, course recommendations/planning, being a student at Berkeley, grad school, research, etc.)
- **No post-lecture questions today or tomorrow** because I have go to somewhere directly after (sorry!)
- **Askademia** is down until further notice
- **Follow-up to student who asked me a question about their Luhn sum code yesterday (Ed post)**

Quiz 1 Survey

PollEv

Computing in the News

- Lumo, Proton's privacy-focused AI chatbot, gets an upgrade (TechCrunch)
- Apple says it is releasing updates early in response to AI cybersecurity concerns (Reuters)
- Diamond Thermal Conductivity: A New Era in Chip Cooling (IEEE Spectrum)
    - The $1 Trillion Problem Diamonds Can Solve (Cleo Abram, HUGE* If True)
- Why tech firms are raising PC and console prices - and blaming AI for chip costs (BBC)

Review

10 min

- Cascade
- Sum odd squares

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Select all implementations of the cascade function such that cascade(123) results in the pictured output.

Cascade: Which correct implementation is better?

- Implementation A is more readable
- Implementation B is more concise
- If you have to choose between readable or concise, choose readable (95% of the time)

More on `cascade` and also `inverse_cascade`: YouTube videos

Review: Sum of Odd Squares

- Implement `sum_odd_squares` recursively and iteratively
- Download starter code `06.py` from the course website under Lecture 6
- Run doctests with `python3 -m doctest 06.py`
- ~~5 min: Try implementing it yourself~~
- 5 min: Pair program with the person next to you
  - What worked?
  - What didn't?
  - Why?

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

True or false: sum_odd_squares_recursive is a higher-order function.

Tree Recursion

20 min

- Tree recursion definition
- Tree recursion-type problems
- Fibonacci
- Count Partitions
- Demo: Wikipedia Philosophy Phenomenon (if time)

Tree recursion definition

A function is **tree recursive** if it makes more than 1 recursive call (in a single recursive case).

We call it tree recursive because if you draw a **diagram** of the recursive calls, it **forms an (upside-down) tree shape**.

***Not to be confused with: Doing recursion with the tree data structure, which *may or may not* be tree recursion** (more on that later!)

Tree recursion-type problems

- Counting/searching ways to do something
- Exploring different choices
- "Try all ways to do it in Case A, and combine that with all ways to do it in mutually exclusive Case B"

Fibonacci (1 of 3)

- Recall: The Fibonacci sequence is defined as:
  - 0th Fibonacci number is 0
  - 1st Fibonacci number is 1
  - Fibonacci number n is the sum of the previous 2 numbers
  - Ex: 0, 1, 1, 2, 3, 5, 8, 13, ...
- Iterative implementation covered in Lecture 03:

```
def fib(n):
    curr, nxt = 0, 1
    k = 0
    while k < n:
        curr, nxt = nxt, curr + nxt
        k += 1
    return curr
```

Fibonacci (2 of 3)

The mathematical definition of Fibonacci is naturally recursive!

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

Fibonacci (3 of 3)

recursionvisualizer.com

Try `virfib(5)`

What do you notice?

Visualizer created by Pamela Fox, former UC Berkeley CS 61A lecturer, now Principal Cloud Advocate at Microsoft.

"vir" comes from Virahanka, an Indian mathematician who first proposed the Fibonacci sequence.

Count Partitions (1 of 7)

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. For example, there are 9 partitions of 6 using parts up to 4.

1. 6 = 2 + 4
2. 6 = 1 + 1 + 4
3. 6 = 3 + 3
4. 6 = 1 + 2 + 3
5. 6 = 1 + 1 + 1 + 3
6. 6 = 2 + 2 + 2
7. 6 = 1 + 1 + 2 + 2
8. 6 = 1 + 1 + 1 + 1 + 2
9. 6 = 1 + 1 + 1 + 1 + 1 + 1

Count Partitions (2 of 7)

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. For example, there are 9 partitions of 6 using parts up to 4.

1. **6 = 2 + 4**
2. **6 = 1 + 1 + 4**
3. 6 = 3 + 3
4. 6 = 1 + 2 + 3
5. 6 = 1 + 1 + 1 + 3
6. 6 = 2 + 2 + 2
7. 6 = 1 + 1 + 2 + 2
8. 6 = 1 + 1 + 1 + 1 + 2
9. 6 = 1 + 1 + 1 + 1 + 1 + 1

Count Partitions (3 of 7)

`count_partitions(6, 4)` is really just adding up:

1. `count_partitions(6 - 4, 4    )`
    a. If we're using **at least one** 4, the only part remaining to partition is `6 - 4 = 2`
2. `count_partitions(6    , 4 - 1)`
    a. If we're using **not using any** 4, we can only use parts up to `4 - 1 = 3`

Count Partitions (4 of 7)

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

Count Partitions (5 of 7)

```
def count_partitions(n, m):
    if n == 0:
        return 1
    else:
        with_m = count_partitions(n - m, m)
        without_m = count_partitions(n, m - 1)
        return with_m + without_m
```

Count Partitions (6 of 7)

```
def count_partitions(n, m):
    if n == 0:
        return 1
    elif n < 0:
        return 0


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

Count Partitions (7 of 7)

```
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: Wikipedia Philosophy Phenomenon (if time) (1 of 2)

- The Wikipedia Philosophy Phenomenon is that for 97% of Wikipedia articles, if you keep clicking on the first hyperlink you'll eventually get to the Philosophy article
  - See also: YouTube talk by Matthew Prebeg, road-to-philosophy.com
- I had Gemini write a demo for this (edited a bit), let's try it out!
- **Note:** You're not expected to understand all of the code or how the setup works, but this is a fun example of how recursion can be used in real-world applications!
  - How to make this tree recursion? What if we searched the first 2 hyperlinks, or n hyperlinks?

Demo: Wikipedia Philosophy Phenomenon (if time) (2 of 2)

To run the demo:

1. Create a `lec06` directory
2. Download `06-wikipedia.py` from Lecture 6 on the course website into that directory
3. Open your terminal and `cd` to that directory
4. Install `uv` in the terminal (instructions; I recommend the "standalone installer" for your operating system)
5. Run `uv init --bare` (this will create a few files, ignore them)
6. Run `uv add requests beautifulsoup4` (this will edit a few files, ignore that)
7. Run `uv run python3 06-wikipedia.py`
8. You can edit the URL on line 101 with an article of your choice OR leave it as it is (it will pick a random article), then redo Step 7

5 min break

Practice

(reminder to self to use high contrast theme)

40 min

- Count Stairs
- Mario Number

Practice: Count Stairs

- Implement `count_stairs`: You are walking up n stairs and you may either take 1 step or 2 steps as you go. How many different ways can you reach the top?
- Download starter code `06.py` from the course website under Lecture 6
- Run doctests with `python3 -m doctest 06.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?

Practice: Mario Number

Mario needs to traverse a video game level that contains Piranha plants. The level is represented as a binary string (`int` of only 0's and 1's), and Piranha plants are represented by a 0. Mario only moves forward and can either **step** (move forward one space) or **jump** (move forward two spaces) from each position. **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?*

- Download starter code `06.py` from the course website under Lecture 6
- Run doctests with `python3 -m doctest 06.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked? What didn't? Why?