Welcome!

Bit by Bit Week 4

1

This Weeks Goals

What we will be working on

4 Pillars to Work on

1 **LeetCode**

- Recursion 1 and 2
- Dynamic Programming Qs

2 **Cracking the Coding Interview**

- Recursion
- Dynamic Programming

3 **Behavioral Question**

- What are your strengths and weaknesses?

4 **Application Material**

- Networking calls

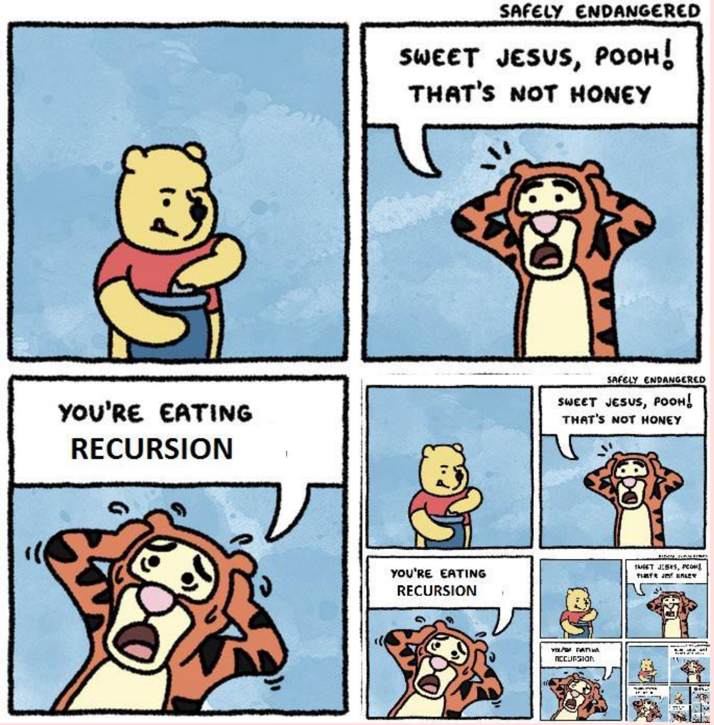
2

Technical Review

Thinking about Recursion and Dynamic Programming

Recursion

- Functions that call themselves
- **Run time:** O(number of recursive calls * work per recursive call).
- **Space:** O(depth_of_recursion * space_per_recursive_call)



Steps for Recursion

1. Base Case
   - When will the problem terminate?
2. Parameters
   - Think of a small sub problem - what information do we constantly need?
3. Recursive Step
   - What do we need to return?
   - How do we use the recursion step to calculate the return value?

Example - Fibonacci Sequence

1. Base Case
   - Program stops when we get to 1 or 0
2. Parameters
   - We need to pass in the current number we are calculating for
   - Ex/ f(3)
3. Recursive Step
   - We want to return the sum of the past two numbers
   - We don’t currently know what the past two numbers were
   - We can call recursion on those numbers

```
def Fibonacci(n):

    if n==0: return 0

    elif n==1: return 1

    else:  return Fibonacci(n-1)+Fibonacci(n-2)
```

6 Recursive Patterns

Iteration

Any problem that can be solved with iteration can be solved with recursion

Sub Problems

Any problem like Fibonacci or Towers of Hanoi that can be broken down into smaller problems.

Selection

Finding all combinations of an input that match a criteria. Example is the knapsack problem.

Sorting

Many sorting approaches can use recursion to solve them.

Divide and Conquer

Backbone of many of our searching algorithms. Split the data structure into two.

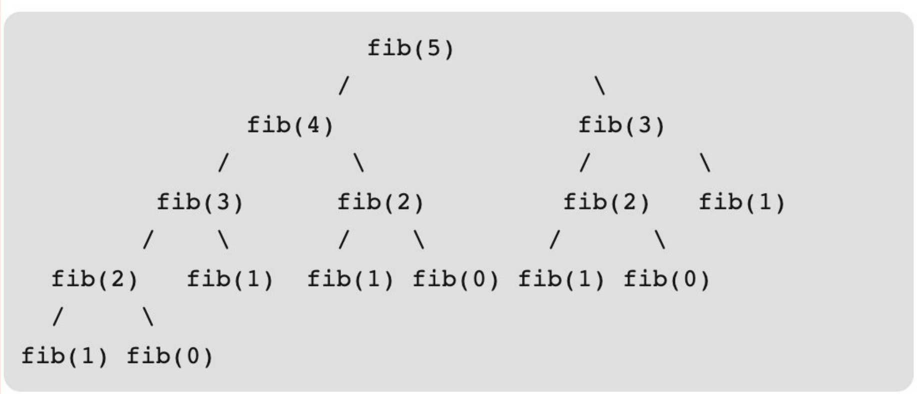
Depth First Search

Backbone of many of our searching algorithms. Split the data structure into two.

Note: We also really like recursion for tree problems!!!

Checking Recursion

1. Try to walk through the problem yourself with simpler cases
2. For medium cases, you can try to type it out to keep track of variables.
3. If you are testing an example with a lot of recursive calls, it's great to use a Recursive Tree



```
                       fib(5)
                      /                \
               fib(4)               fib(3)
              /        \              /       \
         fib(3)     fib(2)        fib(2)   fib(1)
        /    \       /    \        /      \
  fib(2)   fib(1)  fib(1) fib(0) fib(1) fib(0)
  /     \
fib(1) fib(0)
```

Dynamic Programming

- Dynamic programming is the idea of breaking problems down into subproblems
- **Run time:** number of unique states/subproblems * time taken per state
- **Space** can change depending on the approach but typically O(N)
- Typically want to maximize or minimize something and has sub problems

51 coins
40 coins
Total = 90 coins

Gold Analogy

Steps for Dynamic Programming

1. Deciding the State
   - State = parameters that can identify a certain position in a given problem
2. Finding the relationships between States
   - What is the recursive or iterative call that relates states to each other?
3. Adding memorization or tabulation
   - Store the state answer so we don’t have to recalculate it

Top Down Vs. Bottoms Up

Top Down

- Solve problems recursively
- If n is 5, start at 5 and break into smaller subproblems (5-1), (5-2) etc
- Memoization - storing most recent state values in memory. Does not have to be sequential

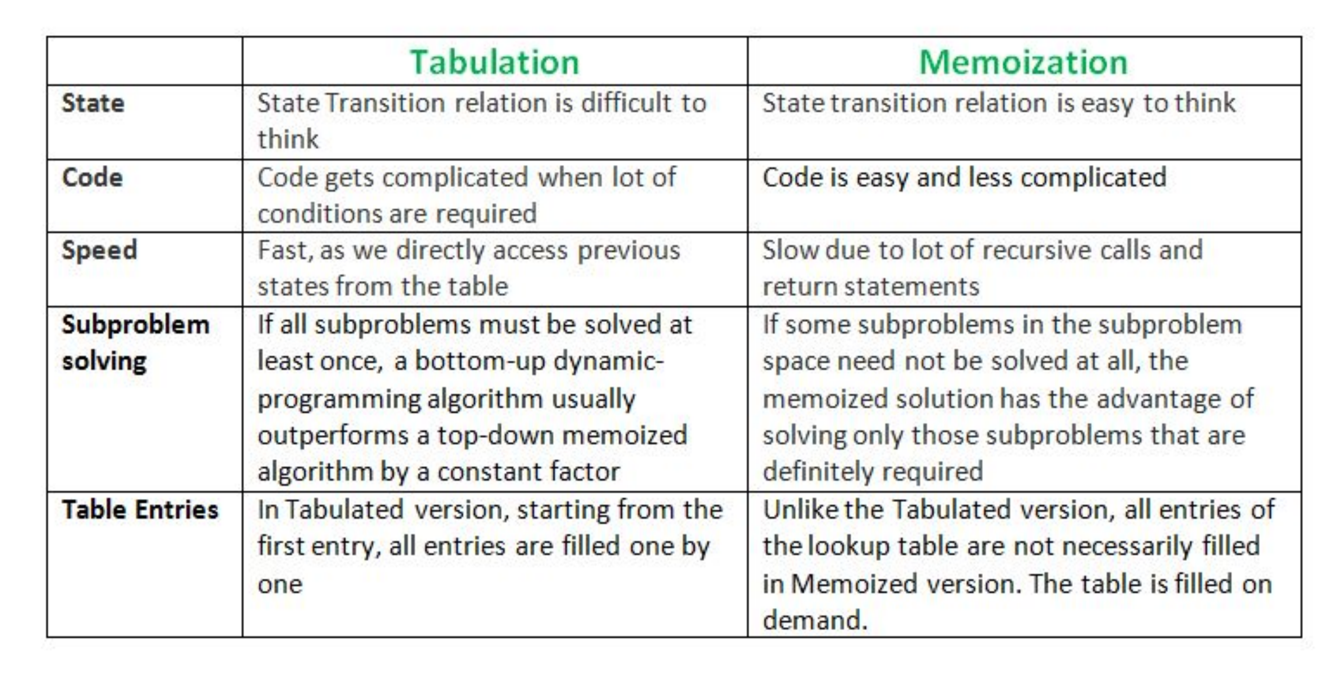
Bottom Up

- Solve problems iteratively
- If n is 5, you will start at 1 and iterate until you get to 5
- Could be better for super long recursion problems
- Tabulation used here - since we start from beginning, table / array used for storage is filled sequentially and we are accessing states from the table to calculate them

Memorization vs Tabulation

Bottom Up

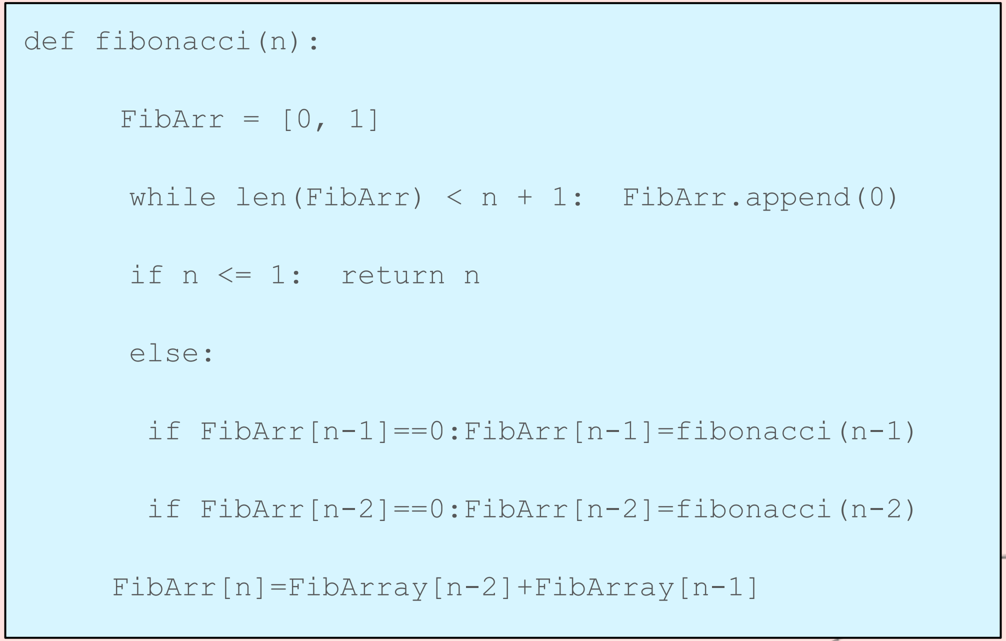
Top Down



	Tabulation	Memoization
State	State Transition relation is difficult to think	State transition relation is easy to think
Code	Code gets complicated when lot of conditions are required	Code is easy and less complicated
Speed	Fast, as we directly access previous states from the table	Slow due to lot of recursive calls and return statements
Subproblem solving	If all subproblems must be solved at least once, a bottom-up dynamic-programming algorithm usually outperforms a top-down memoized algorithm by a constant factor	If some subproblems in the subproblem space need not be solved at all, the memoized solution has the advantage of solving only those subproblems that are definitely required
Table Entries	In Tabulated version, starting from the first entry, all entries are filled one by one	Unlike the Tabulated version, all entries of the lookup table are not necessarily filled in Memoized version. The table is filled on demand.

Example - Fibonacci Sequence

1. State
   - FibArr[n]
2. Relationship
   - FibArr[n] = FibArr[n-1] + FibArr[n+1]
3. Memoization
   - Keeping the past calculated values already in the FibArr array so we can look them up if they exist instead of recalculating



```
def fibonacci(n):

      FibArr = [0, 1]

       while len(FibArr) < n + 1:  FibArr.append(0)

       if n <= 1:  return n

       else:

        if FibArr[n-1]==0:FibArr[n-1]=fibonacci(n-1)

        if FibArr[n-2]==0:FibArr[n-2]=fibonacci(n-2)

      FibArr[n]=FibArray[n-2]+FibArray[n-1]
```

This is a top down approach

Example - Fibonacci Sequence (better)

Only keeping track of past two variables.

Bottom up Approach

```
def fibonacci(n):
    a = 0, b = 1
    if n == 0: return a
    elif n == 1:  return b
    else:
        for i in range(2,n+1):
            c = a + b
            a = b
            b = c
        return b
```

3

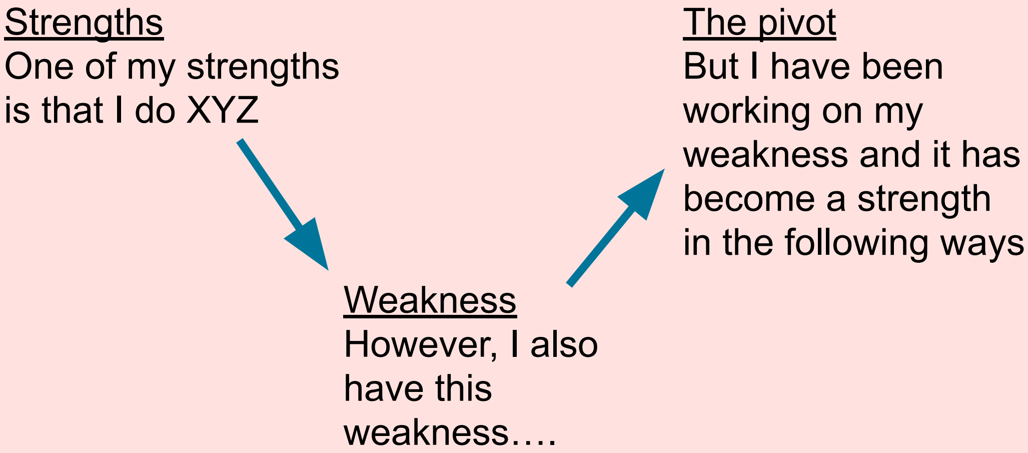
Behavioral Tips

Strengths, Weaknesses and Referrals

“What are you strengths and weaknesses”

- Know your strengths
- Don’t give real weaknesses
- Don’t give stupid weaknesses
- Know your weaknesses too
- End on a positive note

My personal “strengths and weaknesses”



Strengths
One of my strengths is that I do XYZ

Weakness
However, I also have this weakness….

The pivot
But I have been working on my weakness and it has become a strength in the following ways

Networking - key steps

GOAL: To get a referral!

✓ Finding <u>companies</u> you are interested in

✓ Finding <u>individuals</u> you want to talk to at said companies

  a. Recruiters
  b. Engineers

✓ <u>Emailing</u> individuals asking to learn more

1. <u>Speaking</u> on the phone
2. <u>Asking</u> for referrals

Networking - speaking on the phone

- Don’t be shy- be confident!
- Be genuinely interested, people can tell when you waste their time
- Prepare questions so you don’t run out of things to talk about / make sure you learn about everything important
  - Look at their LinkedIn and see what interests you
- Send thank you notes post conversation
  - Try to remember a key aspect of the convo to include

Networking - asking for referrals

- Often at the end of the conversation if you hint that you are applying and are really excited about the company, <u>they will offer to refer you.</u>
- They get $$ when they refer people who get the job
- You **CAN** straight up ask for a referral if they don't say anything!
    - Some people will be awkward about it and maybe won't give one (rare)
    - Most people will say of course or ask for a bit more info before they do!
- Send email after thanking them and reminding them about referral politely

4

Mock Interviews

Feedback and Drills

Great Job!

Everyone has been doing a fabulous job!

- Some struggles with recursion (but hopefully this cleared some things up!)

Keep talking :)

Practice with Dynamic Programming

Question:

Given an integer n, find the minimum number of steps to reach integer 1.

At each step, you can:

Subtract 1,

Divide by 2, if it is divisible by 2

Divide by 3, if it is divisible by 2

Doc: Go to this link here. Breakout room one will solve Top Down on Page 1 and room two will solve Bottom up on page two.

5

Questions

What didn’t make sense?