Scientific Computing

Feb 7, 2025

Announcements

→ HW 2 due Mon. Feb. 17

→ Mon, Feb 17, no in-person lecture and no office hours

→ Wed, March 5, midterm exam, in person portion (in class)

→ Fri, March 7, no class, extra office hours for take-home portion (time TBD)

Today

→ Unix command line

→ The coding process

Office Hours:

Mon + Fri

9:30am - 10:30am

Cudahy 307

(8) cat [file name] - print a whole file to the screen

(9) head [file name] - print first 10 lines of a file

(10) tail [file name] - print last 10 lines

"-n" to change the number for head and tail, like

head -n 20 [file]

(11) less [file name] - open a file in a way you can scroll around but not edit

"q" to quit

Note: you can use the up and down arrows to scroll through your history of commands.

You can do anything on a terminal.
A few non-beginner things.

(12) nano [filename] opens a terminal-based file editor. Keyboard shortcuts for everything, [ctrl] + [key] on Mac. You may need to install nano on some platforms.

(13) touch [filename] just puts a blank file in the current folder

You can write whole programs ("bash scripts") to do anything.

When you run Python, its "active directory" is whatever your p.w.d. is in the terminal window where you're running it.

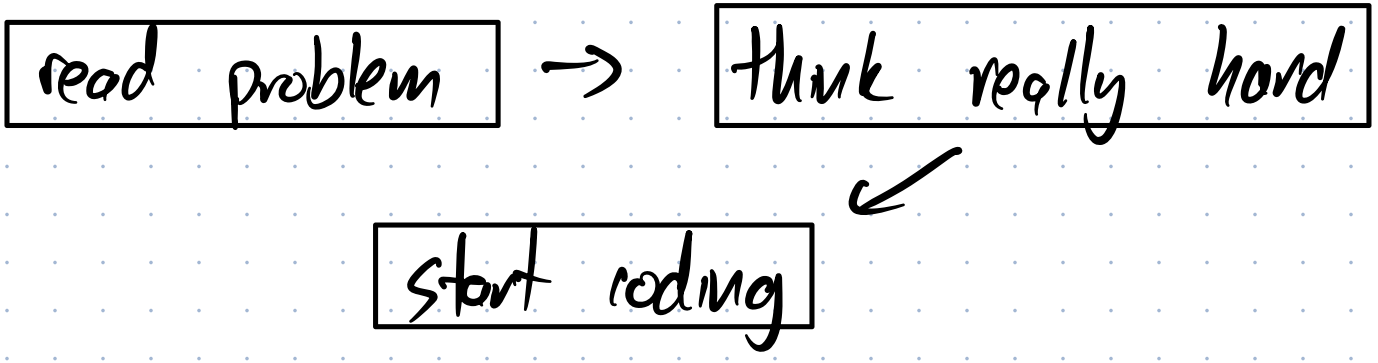
Example:

```
with open("five_letter_words.txt", "r") as f:
    words = f.readlines()

#do any words have no vowels?
vowels = ["a", "e", "i", "o", "u"]
print([w for w in words if not any(v in w for
                                   v in vowels)])
```

Topic 4 - The Coding Process

Hardest Approach:



Too many steps at once, all in your head

Better Process:

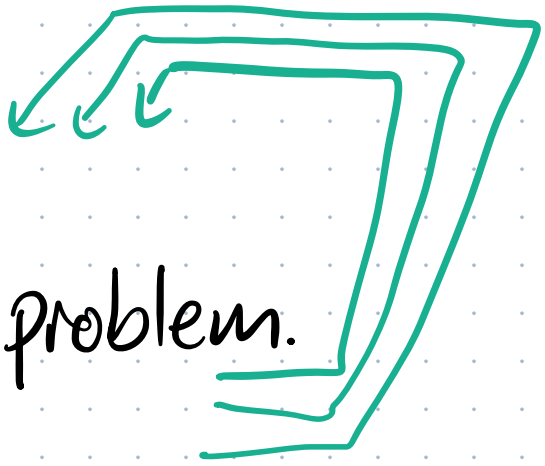
1) Read the problem.

Better Process:

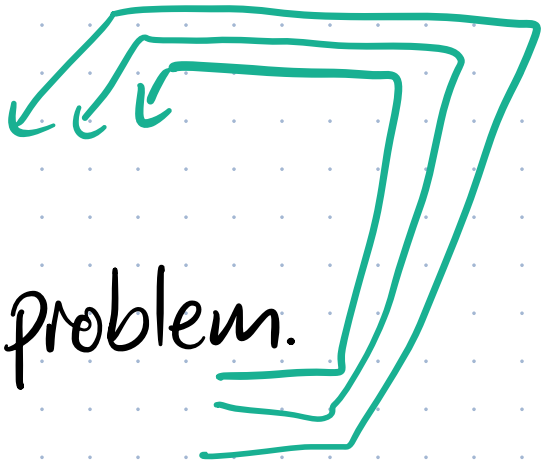
1) Read the problem.
2) Think about the problem.

Better Process:

1) Read the problem.
2) Think about the problem.



Better Process:



1) Read the problem.
2) Think about the problem.

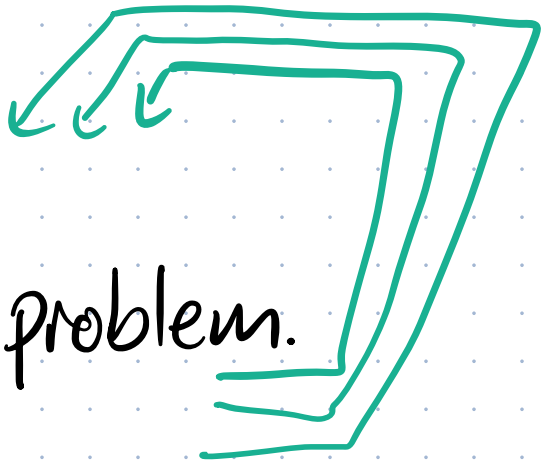
3) Do some examples by hand to see if you understand the problem.

   Leg. Longest Collatz sequence:

   $20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$

   length 8)

Better Process:



1) Read the problem.
2) Think about the problem.

3) Do some examples by hand to see if you understand the problem.

   Leg. Longest Collatz sequence:

   $20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$

   length 8)

4) Think about how you could solve it.

   What steps did you do by hand?

Better Process:

1) Read the problem.
2) Think about the problem.
3) Do some examples by hand to see if you understand the problem.
   e.g. Longest Collatz sequence:
   $20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$
   length 8)
4) Think about how you could solve it.
   What steps did you do by hand?
5) Write out, by hand, in words, the steps of your algorithm (pseudocode)

Ex: (Collatz)

the length of the longest chain

```
set longest_chain = 0
set longest_num = 0        # the # that made that longest chain
loop over "num" from 1 to 1 million:
    compute length of chain for num
    if length > longest_chain:
        longest_chain = length
        longest_num = num
answer is longest_num
```

This part is many steps! You can think about it separately, or even as a function.

6) Now start coding!

As you code:

7) "Rubber Ducking"

Talk to a rubber duck, out loud, explaining what you're doing in each line, and why

8) Pause often to test a few lines of code at a time before writing more.

→ Do they do what you thought?

→ Is your loop looping over the right thing? (print something to check!)

→ Does the list you built contain the things you thought?

If it's not working:

9) Debug it! Think of small test cases.
(1 to 10 instead of 1 to 1 million)

Add lots of print statements to see what values the variables hold in each step and see if they are what you expect.

When it's working:

10) Test it! Test the small examples from step 3.
Do you get the expected answer?

Test with big examples:

- does the speed make sense?

Topic 5 - Search Spaces and Brute Force

Most of our problems can be summarized as:

"Out of all ways to do [blank]:

(1) Do any of them satisfy this list of constraints?

and/or

(2) Which one is optimal? "

"Out of all ways to do [blank]:
(1) Do any of them satisfy this list of constraints?
and/or
(2) Which one is optimal? "

Greedy Algos gave us a quick way to get a [blank] that might be decent, but in most cases is not at all guaranteed to be optimal.

They don't check every [blank] - in fact they only check a single one!

The search space of a problem is the set of all possible "things" that may or may not satify your constraints and that all have some score that you want to minimize or maximize.

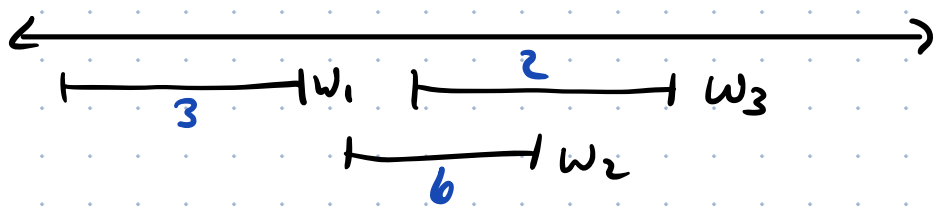
The next few lectures are focused on ways to actually check the entire search space to find the optimal solution.

guaranteed

The most obvious way to do this is brute force: generate every single element of the search space, and check whether it satisfies the constraints and if so, what its score is.

Ex 1: Weighted Interval Scheduling

3 requests



Search space: all subsets of $\{w_1, w_2, w_3\}$

candidate	satisfies constraints	score
$\{\}$	✓	0
$\{w_1\}$	✓	3
$\{w_2\}$	✓	6
$\{w_3\}$	✓	2
$\{w_1, w_2\}$	✓	9 optimal
$\{w_1, w_3\}$	✓	5
$\{w_2, w_3\}$	X	8 - irrelevant
$\{w_1, w_2, w_3\}$	X	11 - irrelevant