7/27 Lecture Agenda

- Announcements (including midterm discussion)
- Part 5-3: Edit Distance
- 10 minute break!
- Part 5-4: Knapsack

Announcements

- Pre-HW5 out tonight (due Weds. of next week)
- HW5 out on Friday (due Fri. of next week)
- Midterms are all graded but will be scanned after class. Grades (and solutions) will be posted ASAP after that.
- The summer grading change basis / withdrawal deadline is Friday at 5 PM. I will post some context (very rough estimated grades) by Thurs.

Soooo, about the midterm

- It was hard. Really hard.

- I intended it to test deep understanding, but didn't mean to make it that hard or time-crunch-y.

- Really hard exams can be discouraging.
    - I got a 15.5/32 (I still remember the score!) on a data structures midterm and thought it meant I wasn't cut out for CS. But there's no reason to believe that exam scores even measure that kind of potential – taking exams and being a good algorithm designer are very different things...

- This was the first *in-person* exam I have written for a large CS class. (Ironically, a takehome I wrote for 161 last year was widely viewed as easier and fun)

Things I wish I had done differently

- Fewer questions overall (the issue was that I tried to cover *all* major content at least somewhat)

- More questions testing foundational knowledge (like the red/black tree and SelectSort ones), fewer questions testing really deep / tricky understanding (like the Karatsuba / Strassen one, or the Dijkstra's modification)

- Short answer questions were supposed to spare you from writing out a lot of work, but giving partial credit based *only on the final answer* was also not ideal

My philosophy

- I don't like to change the rules after they've been stated if it ends up hurting some while helping others.
  - e.g., I could have given *really really* generous partial credit, but that would not have been consistent with people's expectations going into (and during) the exam.
  - Something like "your final score can clobber your midterm score" would've really needed to be in the syllabus from the start, and I can't add that now.

- But I've always stated that I have discretion over the overall grade cutoffs.
  - The exam being really hard doesn't mean I'll give worse grades. If anything I'll be even *more* sympathetic.
  - If I see someone improve a lot from the MT to the final, for example, I can take that into account when setting cutoffs (for everyone).

Does this mean that in practice, final grades will depend almost entirely on the midterm?

- No. The final, while it won't be as hard or time-intensive as the midterm, will also allow demonstration of deep as well as basic understanding.
- Also, doing well on the homework is still meaningful (and there are bonus opportunities.)
- The midterm, although it does have higher variance, is still 90 of the 600 total points.

Does this mean that we will all get lower grades?

- No. If anything it probably means *the opposite*, since I'll take into account that it may have been hard to demonstrate your full knowledge on the midterm under time pressure.

Final thoughts

- HW4 has a midterm feedback question that is essentially free points – please do share your opinions on the midterm and your advice for the final!

- HW5 will also have a question that offers a chance to take a second look at a part of the midterm that you found difficult.

7/27 Lecture Agenda

- Announcements (including midterm discussion)
- Part 5-3: Edit Distance
- 10 minute break!
- Part 5-4: Knapsack

WORLD 5-3

Edit D_stencce

Divide and Conquer

Sorting & Randomization

Data Structures

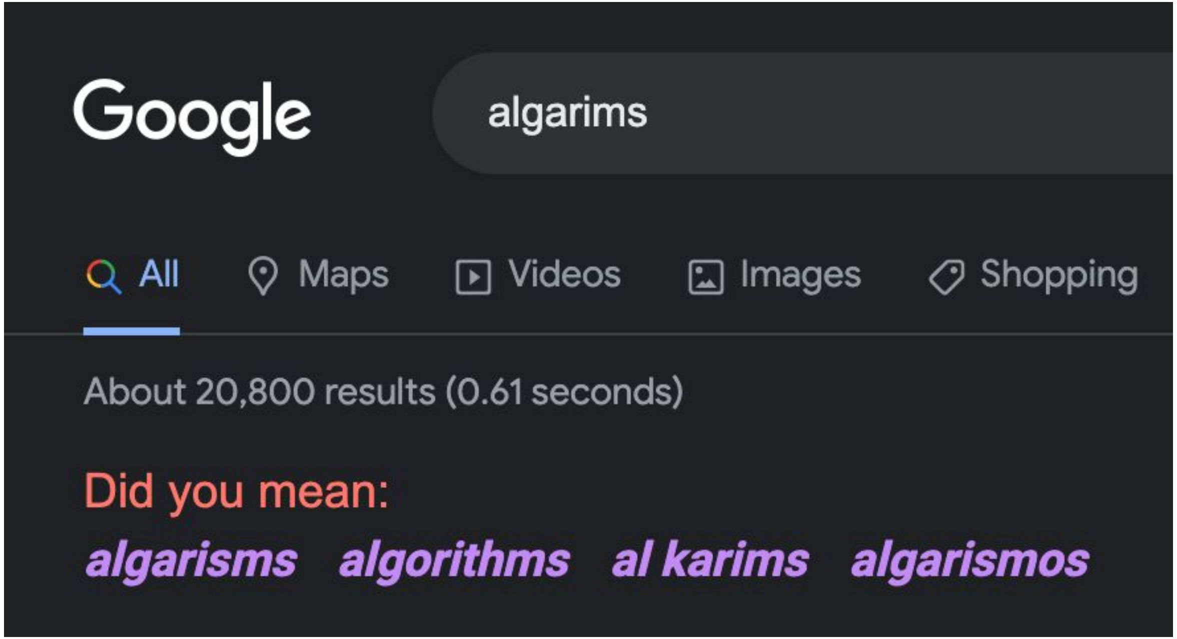
Graph Search

Dynamic Programming

Greed & Flow

Special Topics

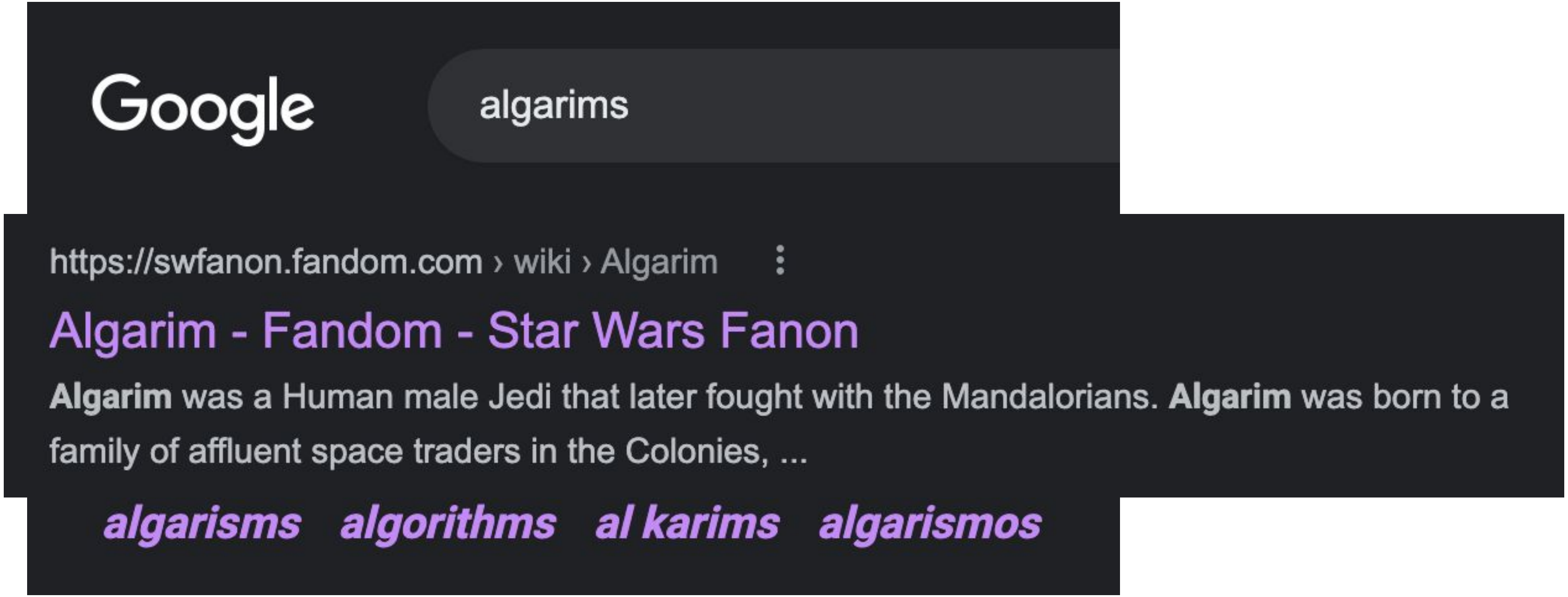
Did you mean...



About 20,800 results (0.61 seconds)

Did you mean: *algarisms* *algorithms* *al karims* *algarismos*

Did you mean...



algarims

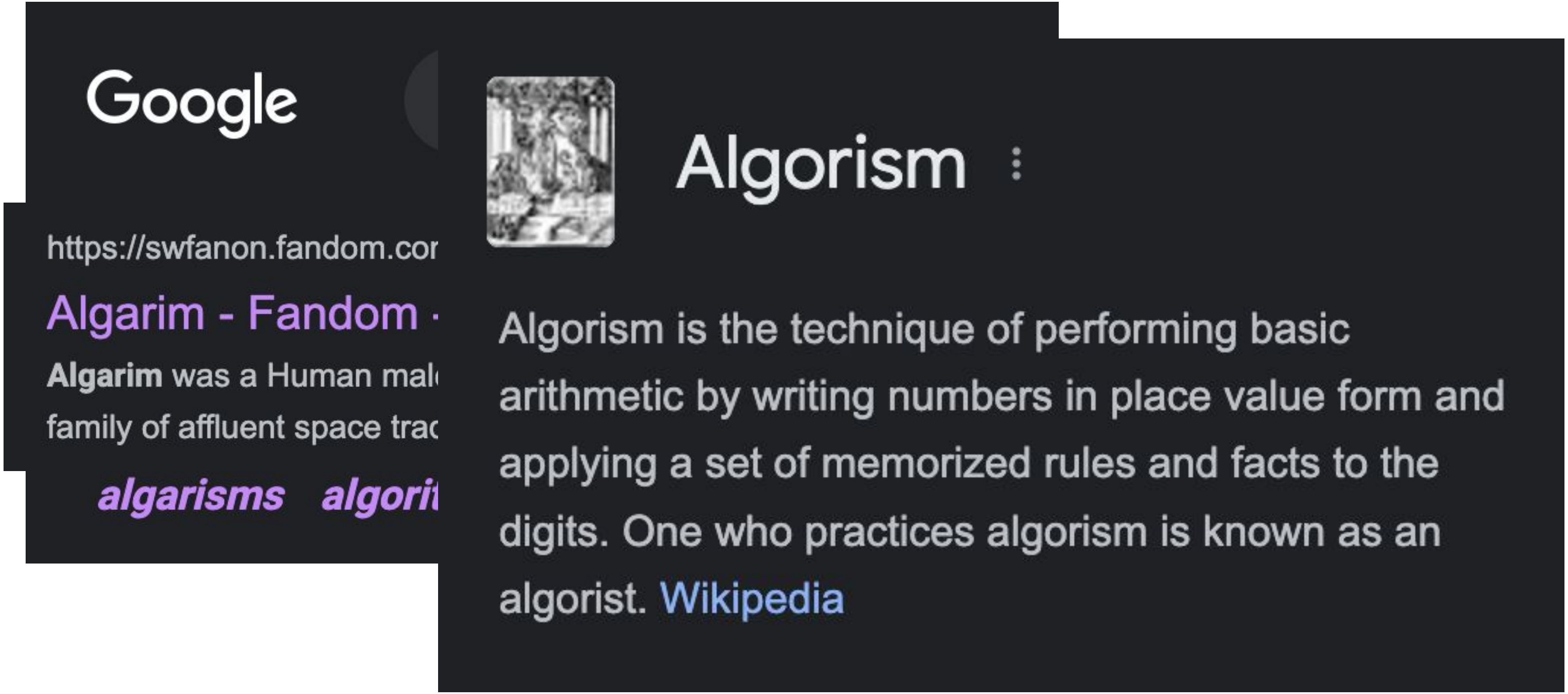
https://swfanon.fandom.com › wiki › Algarim

Algarim - Fandom - Star Wars Fanon

Algarim was a Human male Jedi that later fought with the Mandalorians. **Algarim** was born to a family of affluent space traders in the Colonies, ...

algarisms ***algorithms*** ***al karims*** ***algarismos***

Did you mean…



Google

https://swfanon.fandom.cor

Algarim - Fandom ·

Algarim was a Human mal

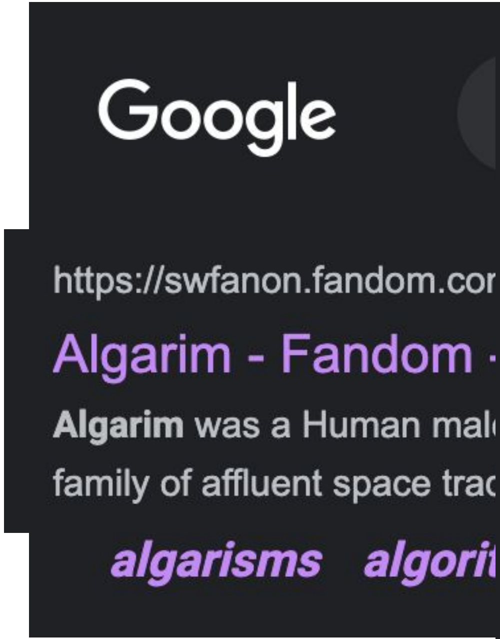
family of affluent space trac

algarisms *algorit*

Algorism

Algorism is the technique of performing basic arithmetic by writing numbers in place value form and applying a set of memorized rules and facts to the digits. One who practices algorism is known as an algorist. Wikipedia

Did you mean...



Google

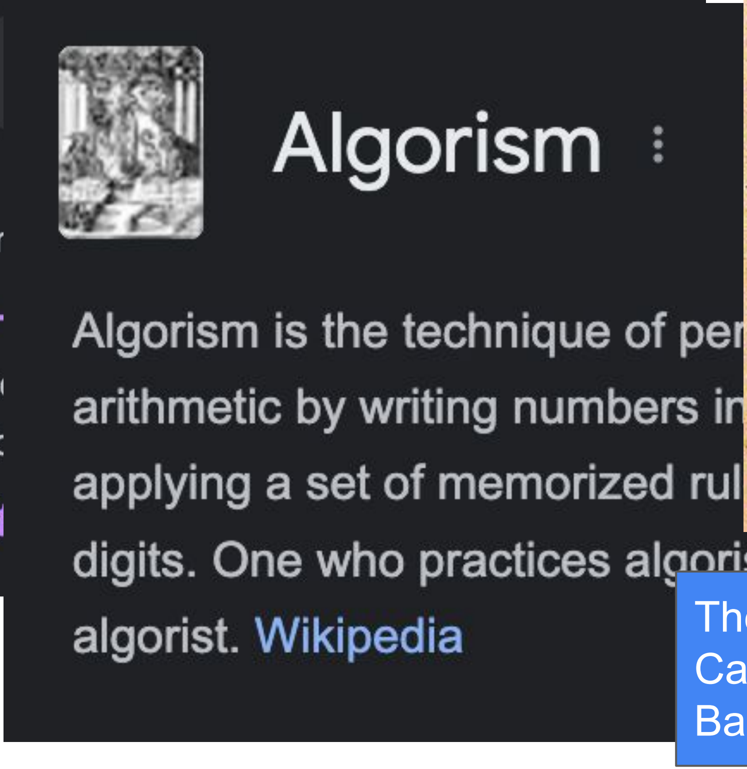
https://swfanon.fandom.cor

Algarim - Fandom ·

Algarim was a Human mal

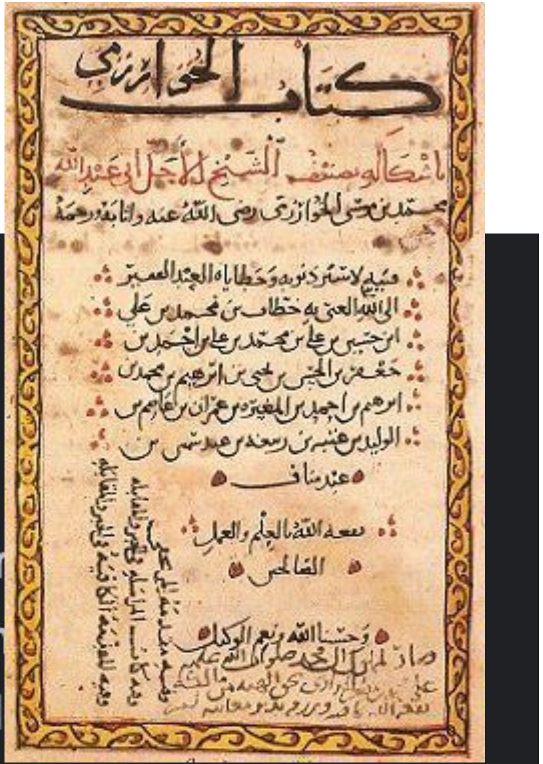
family of affluent space trac

algarisms ***algorit***



Algorism ⋮

Algorism is the technique of per arithmetic by writing numbers in applying a set of memorized rul digits. One who practices algorism is known as an algorist. Wikipedia



كتاب الخوارزمي
محمد بن موسى الخوارزمي رضي الله عنه

The Compendious Book on Calculation by Completion and Balancing (al-Khwārizmī)

Edit distance

- How many steps apart are **alligator** and **algorithm**?

- Suppose that one "step"
  is any of the following operations:
  - **insert** one letter
  - **delete** one letter
  - **substitute** one letter for another

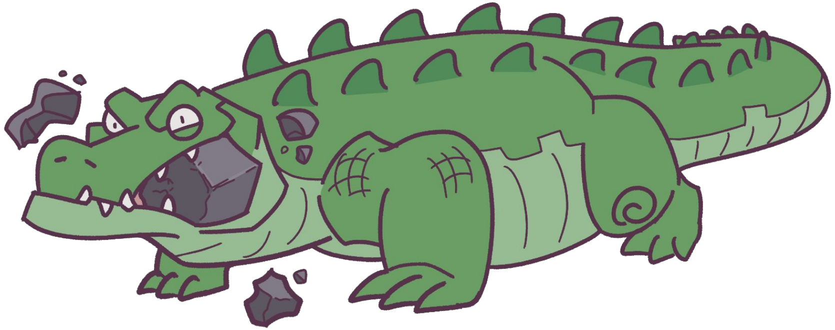
One 7-step path

`alligator`	
`aligator`	*delete first* `l`
`algator`	*delete* `i`
`algotor`	*substitute* `o` *for second* `a`
`algortor`	*insert* `r`
`algoritor`	*insert* `i`
`algorithr`	*substitute* `h` *for second* `o`
`algorithm`	*substitute* `m` *for second* `r`

But is this optimal?

We could have done it with 6 substitutions…

```
alligator
algigator
algogator
algorator
algoritor
algorithr
algorithm
```



OK, this is hard to just eyeball.
We need an algorithm!

Wait a minute...

- If the two words we're comparing are not the same length, then we have to use insertion(s) and/or deletion(s).

- But if the two words we're comparing are the same length, do we ever need insertions/deletions?
  - and would we ever need to use both? or just one or the other?

Substitutions may not be enough!

Consider comparing **`sisi`** and **`iris`**.

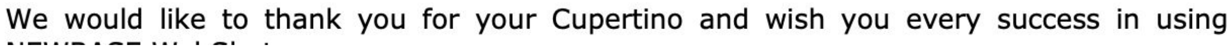
If we used just substitutions, it would take 4 steps.

But we can do it in 3 steps with one deletion, one substitution, and one insertion:

`sisi`	
`isi`	*delete first* **`s`**
`iri`	*substitute* **`r`** *for* **`s`**
`iris`	*insert* **`s`**

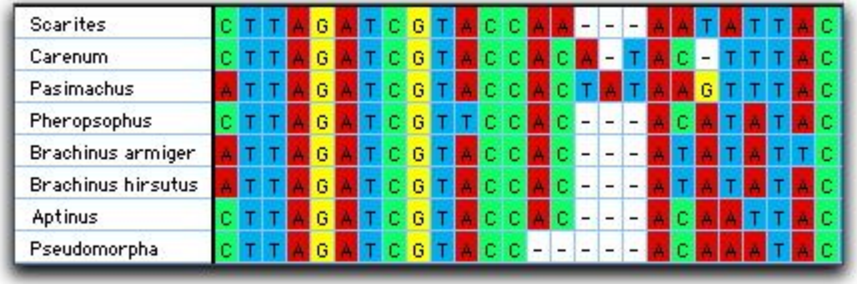
Why do we care?

- Besides spell-checking, that is…



We would like to thank you for your Cupertino and wish you every success in using

- **DNA / protein sequence alignment** is a lot like this too! E.g., given a bunch of sequences of the same gene / protein in different species, which are most similar (and perhaps therefore closely related)?
  - (Though with some kinds of operations being much more common/plausible than others…)



Scarites	C	T	T	A	G	A	T	C	G	T	A	C	C	A	A	-	-	-	A	A	T	A	T	T	A	C
Carenum	C	T	T	A	G	A	T	C	G	T	A	C	C	A	C	A	-	T	A	C	-	T	T	T	A	C
Pasimachus	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	T	A	T	A	A	G	T	T	T	A	C
Pheropsophus	C	T	T	A	G	A	T	C	G	T	T	C	C	A	C	-	-	-	A	C	A	T	A	T	A	C
Brachinus armiger	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	T	A	T	A	T	T	C
Brachinus hirsutus	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	T	A	T	A	T	A	C
Aptinus	C	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	C	A	A	T	T	A	C
Pseudomorpha	C	T	T	A	G	A	T	C	G	T	A	C	C	-	-	-	-	-	A	C	A	A	A	T	A	C

So how do we actualy find edit distance?

How about BFS?

- Start at the first word, try to reach the second
- Make all alterations that lead to strings 1 step away…
- Then make all alterations (to *those*) that lead to strings 2 steps away…
- Repeat until the target is found

How about BFS?

Like an overambitious US road trip, **this visits way too many states**!

`alligator` -> **`blligator`**, ..., **`lligator`**, ..., **`aalligator`**...

How about BFS?

Like an overambitious US road trip, **this visits way too many states!**

`alligator` -> **`blligator`**, …, **`lligator`**, …, **`aalligator`**…

There are actually almost 500 first moves from here!

- 260ish insertions (10 places to insert * 20 letters)
    - *actually fewer since there can be two ways to get the same result*
- 9 deletions
- 225 substitutions (9 letters to overwrite * 25 new options)

This explodes too fast. We get overwhelmed before we find the target.

How about BFS?

Like an overambitious US road trip, **this visits way too many states**!

`alligator` -> **`blligator`**, ..., **`lligator`**, ..., **`aalligator`**...

There are actually almost 500 first moves from here!

- 260ish insertions (10 places to insert * 20 letters)
  - *actually fewer since there can be two ways to get the same result*
- 9 deletions
- 225 substitutions (9 letters to overwrite * 25 new options)

This explodes too fast. We get overwhelmed before we find the target.

This is more like CS109. The exact details aren't as important for us here.

Meet in the middle

Does the BFS work better if we simultaneously explore from both ends?

Answer: Yes, and it makes a practical difference, but it's not enough to truly solve this problem.

Meet in the middle

1

Does the BFS work better if we simultaneously explore from both ends?

Answer: Yes, and it makes a practical difference, but it's not enough to truly solve this problem.

000

Meet in the middle

Does the BFS work better if we simultaneously explore from both ends?

Answer: Yes, and it makes a practical difference, but it's not enough to truly solve this problem.

```
1
```

```
∅, 0, 01, 10, 11
```

```
000
```

Meet in the middle

Does the BFS work better if we simultaneously explore from both ends?

Answer: Yes, and it makes a practical difference, but it's not enough to truly solve this problem.

```
1

∅, 0, 01, 10, 11

00, 001, 010, 100,
0000, 0001, 0010,
0100, 1000

000
```

Meet in the middle

Does the BFS work better if we simultaneously explore from both ends?

Answer: Yes, and it makes a practical difference, but it's not enough to truly solve this problem.

```
1

∅, 0, 01, 10, 11

00, 001, 010, 011,
100, 101, 110, 111

00, 001, 010, 100,
0000, 0001, 0010,
0100, 1000

000
```

Meet in the middle

Does the BFS work better if we simultaneously explore from both ends?

Answer: Yes, and it makes a practical difference, but it's not enough to truly solve this problem.

Both explorations can still get pretty big!

1

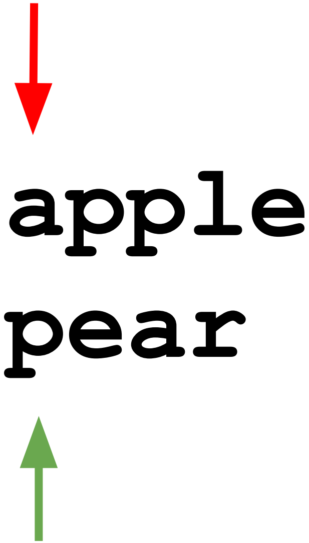
∅, 0, 01, 10, 11

00, 001, 010, 011,
100, 101, 110, 111

00, 001, 010, 100,
0000, 0001, 0010,
0100, 1000

000

How about a more directed approach?

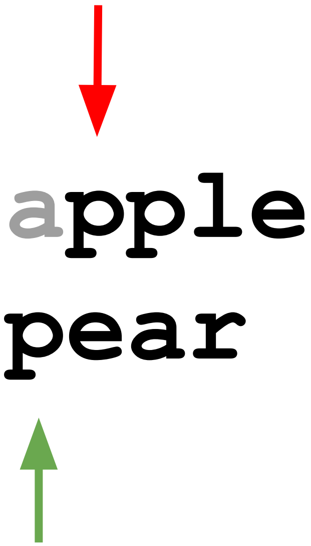


```
apple
pear
```

Step through the strings together, modifying the first one.

What are our options here?

How about a more directed approach?



`apple`

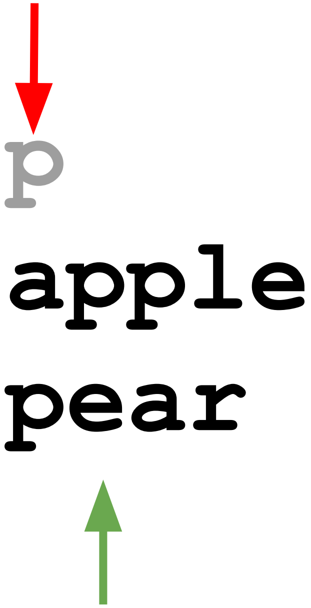
`pear`

Step through the strings together, modifying the first one.

What are our options here?

- Delete the `a` and advance the first pointer.

How about a more directed approach?



p

apple

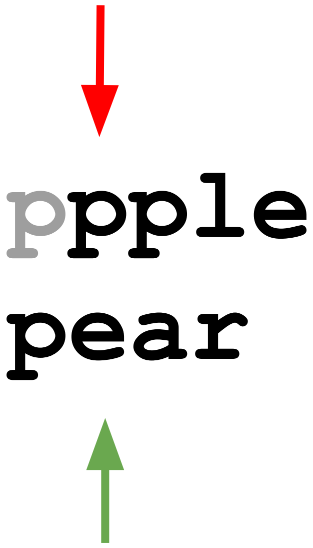
pear

Step through the strings together, modifying the first one.

What are our options here?

- Delete the `a` and advance the first pointer.
- Insert a `p` to match the `p` in `pear`, and advance the second pointer. (The first pointer is still pointing at `a`)

How about a more directed approach?



`ppple`

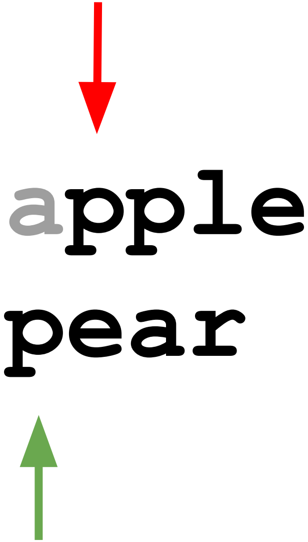
`pear`

Step through the strings together, modifying the first one.

What are our options here?

- Delete the `a` and advance the first pointer.
- Insert a `p` to match the `p` in `pear`, and advance the second pointer. (The first pointer is still pointing at `a`)
- Change the `a` to `p`, and advance both pointers.

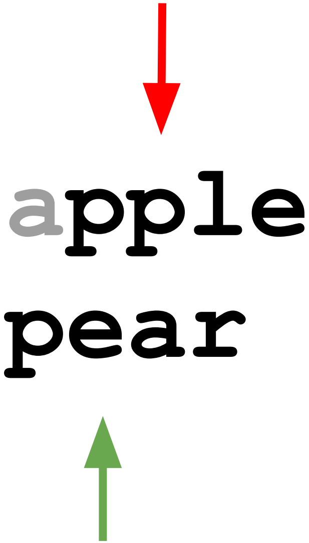
When the pointers agree, we advance both for free!



apple

pear

When the pointers agree, we advance both for free!

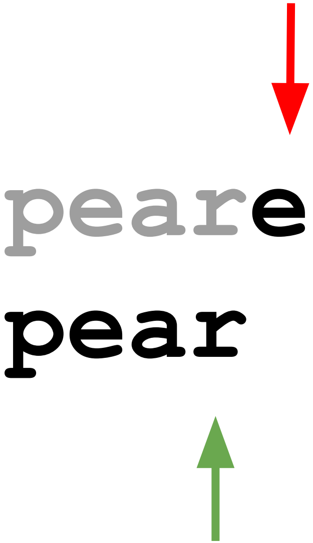


apple

pear

This is "free" because it doesn't correspond to an insertion, deletion, or substitution.

Dealing with leftovers



peare

pear

Even though we reached the end of `pear`*, we need to pay to delete that extra* `e`*... we're not done until **both** pointers reach the end!*

The choices

- Deletion: advance the first pointer and pay 1.
- Insertion: advance the second pointer and pay 1.
- Substitution: advance both pointers and pay 1.
- Both pointers point at the same thing: advance both pointers and pay 0.

How do we minimize the total cost, without just trying everything like in BFS?

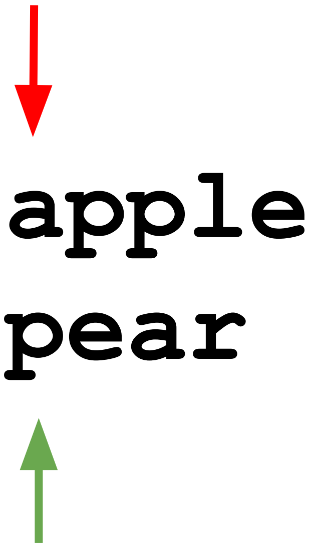
Dynamic programming to the rescue!

A state in this problem is given by

(position of first pointer, position of second pointer)

We ask: what's the *least* we can have spent so far to get to this state?

Then, what's the least we can have spent to get to the final state?



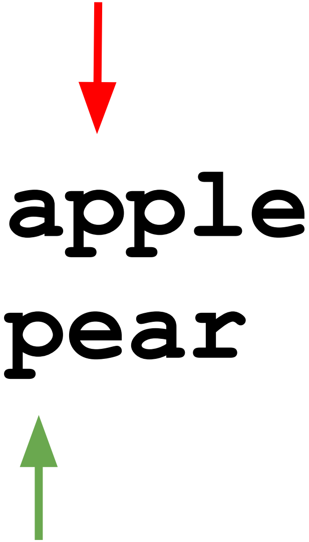
apple
pear

solve(0, 0) = min(

solve(1, 0) + 1, deletion

solve(0, 1) + 1, insertion

solve(1, 1) + 1) substitution



apple

pear

solve(0, 0) = min(

solve(1, 0) + 1,

solve(0, 1) + 1,

solve(1, 1) + 1)

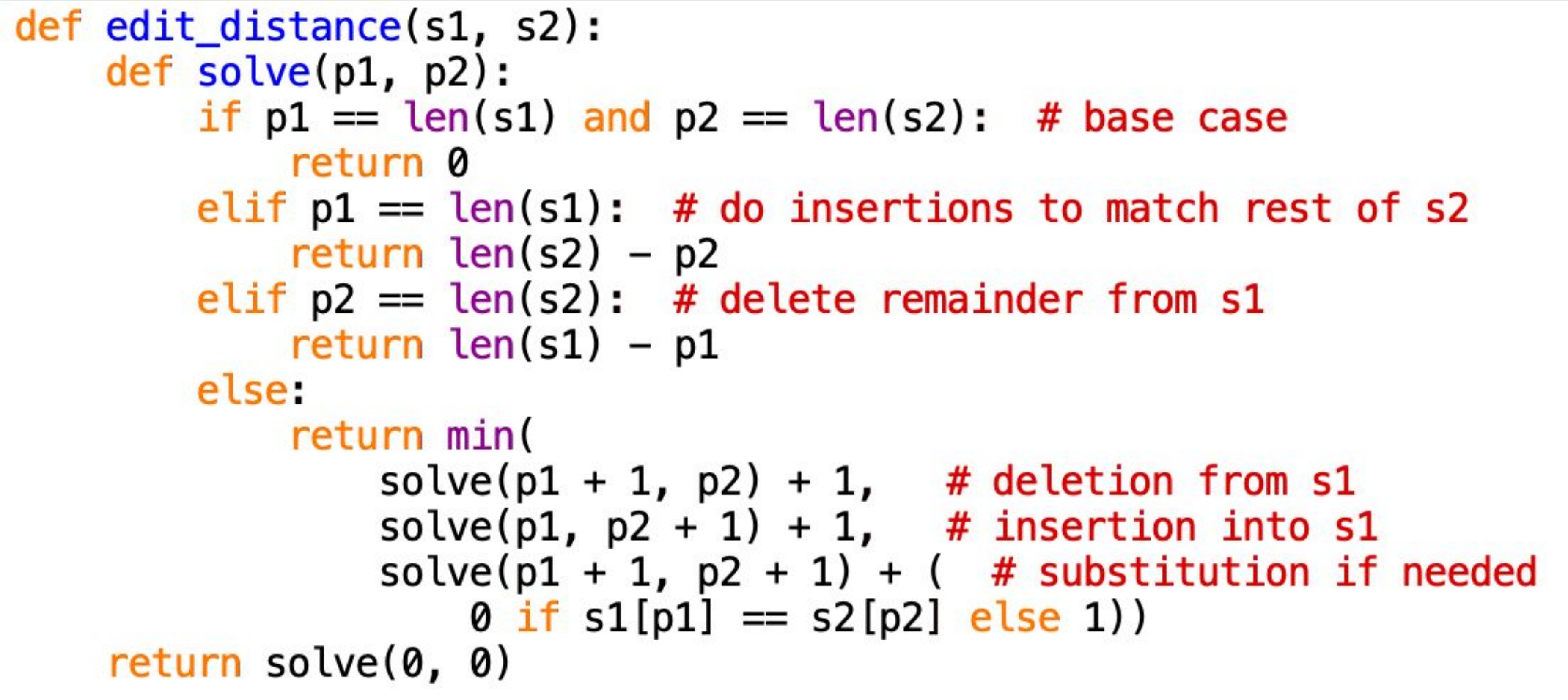
solve(1, 0) = min(

solve(2, 0) + 1, deletion

solve(1, 1) + 1, insertion

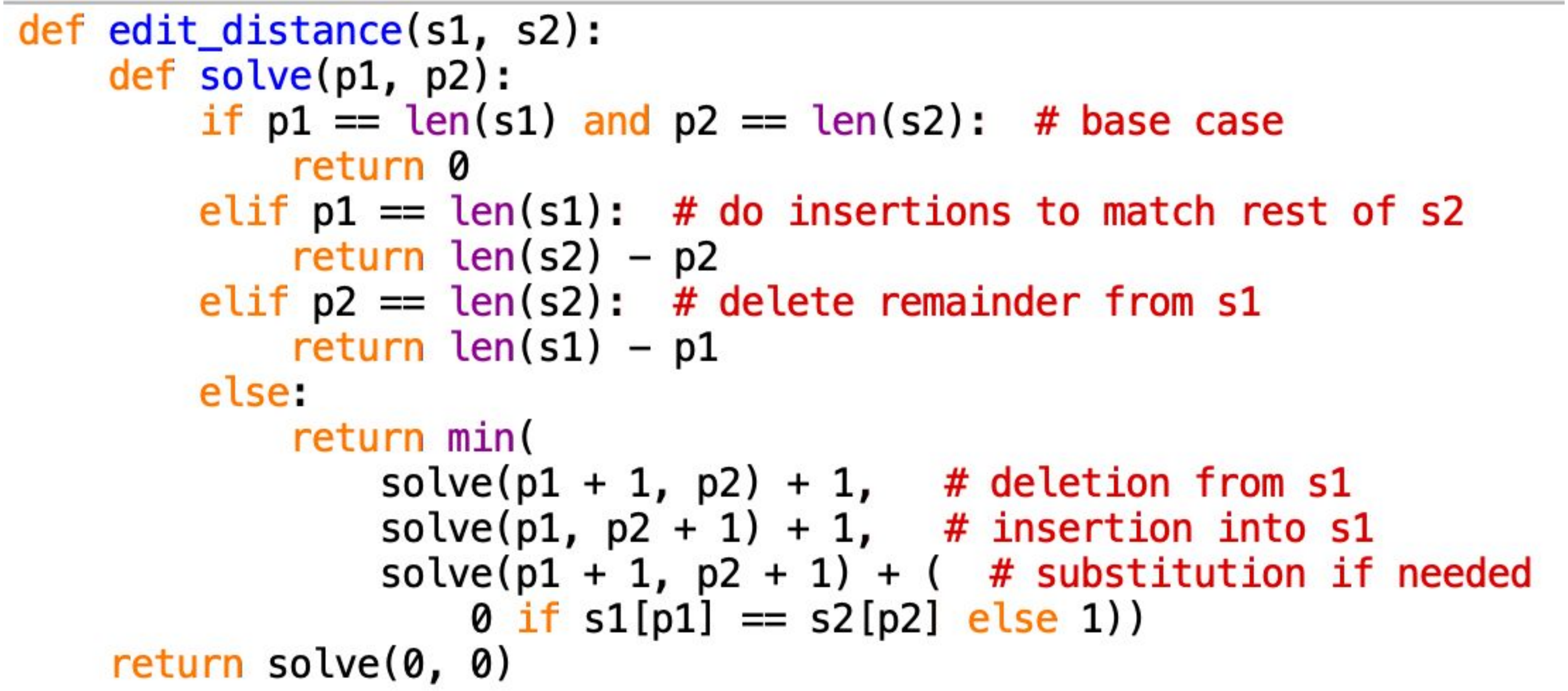
solve(2, 1) + 0) free

Turning it into code

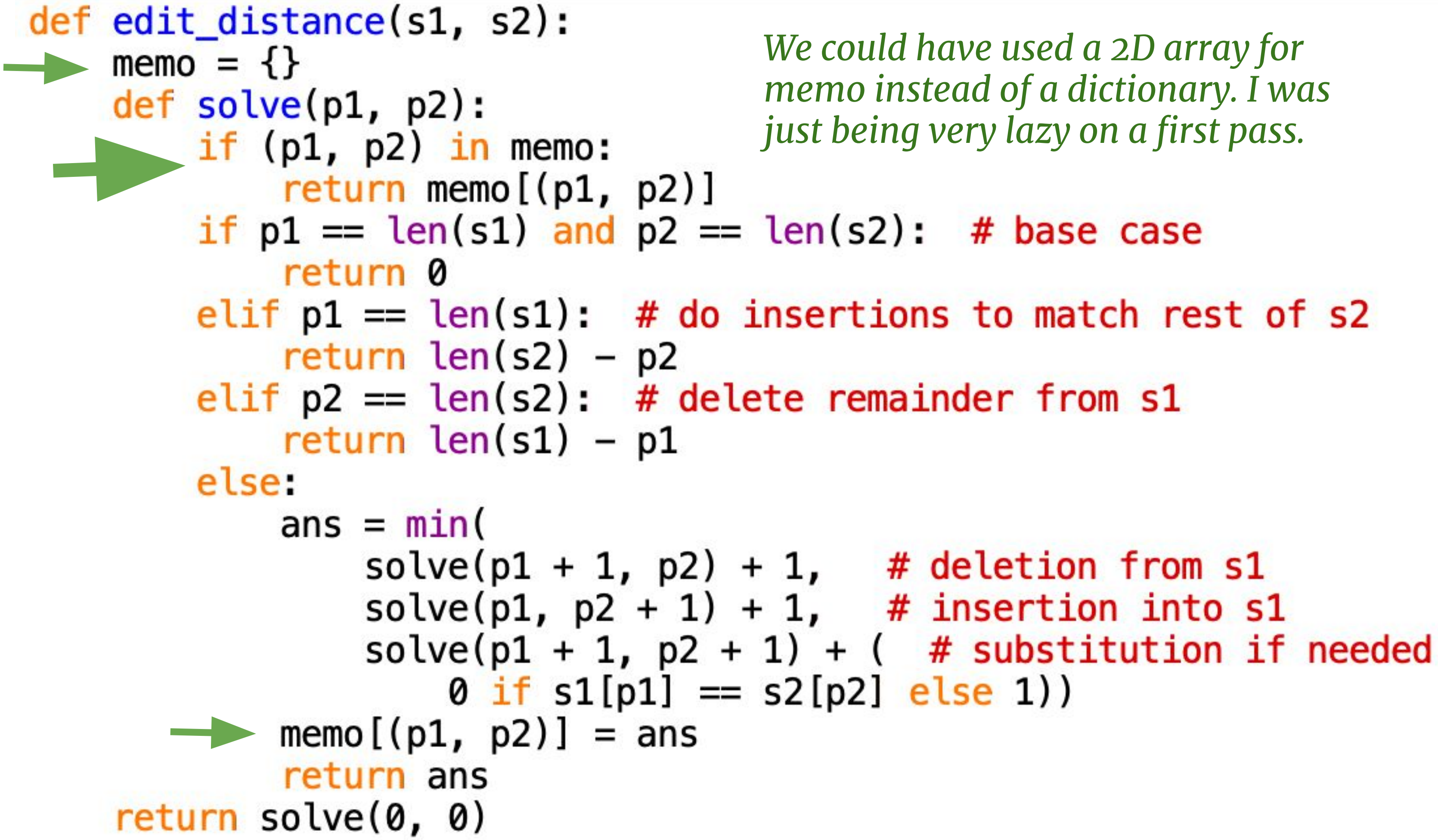


```python
def edit_distance(s1, s2):
    def solve(p1, p2):
        if p1 == len(s1) and p2 == len(s2):  # base case
            return 0
        elif p1 == len(s1):  # do insertions to match rest of s2
            return len(s2) - p2
        elif p2 == len(s2):  # delete remainder from s1
            return len(s1) - p1
        else:
            return min(
                solve(p1 + 1, p2) + 1,   # deletion from s1
                solve(p1, p2 + 1) + 1,   # insertion into s1
                solve(p1 + 1, p2 + 1) + (  # substitution if needed
                    0 if s1[p1] == s2[p2] else 1))
    return solve(0, 0)
```

But we repeatedly compute the same subproblems!

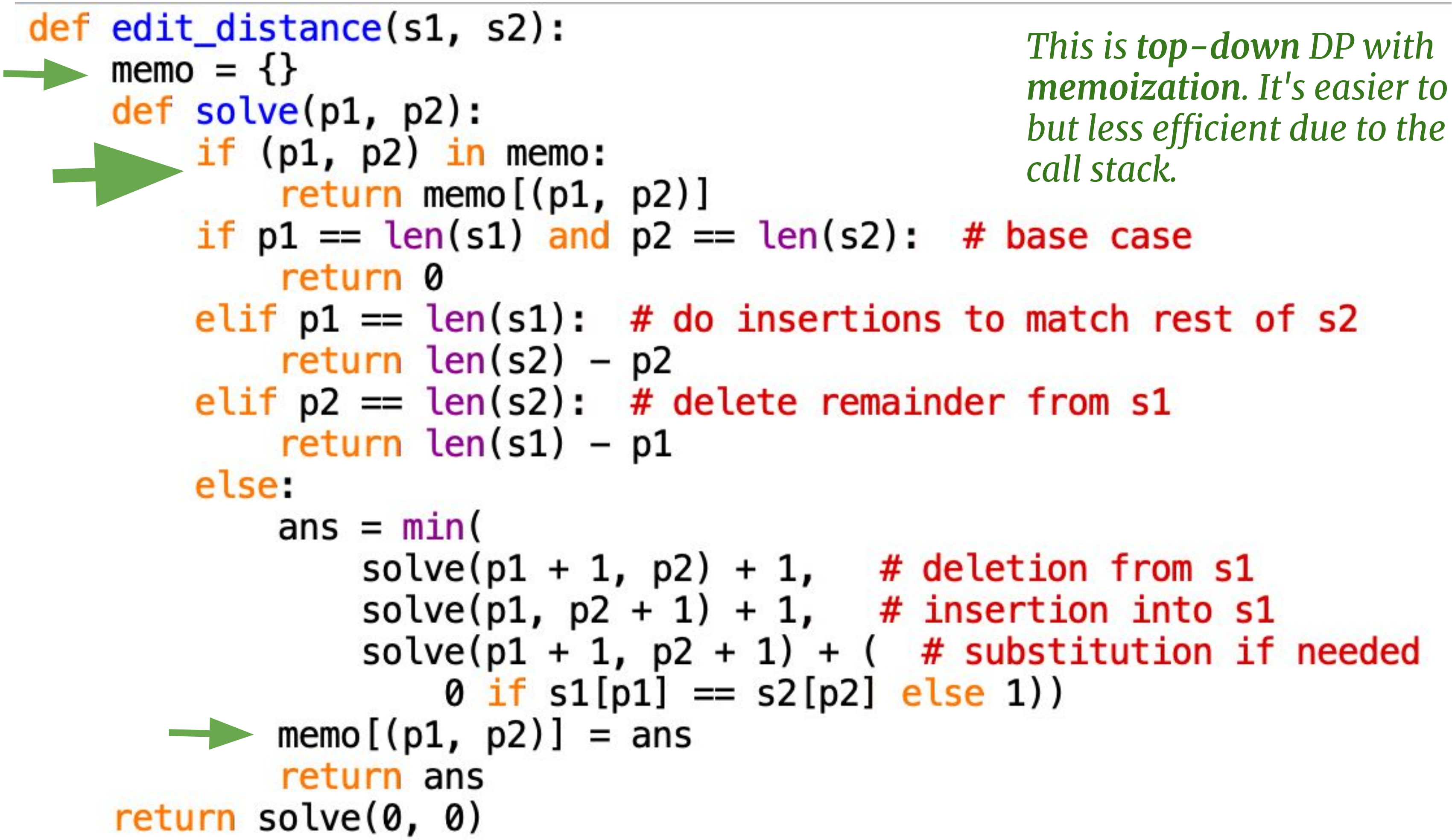


```python
def edit_distance(s1, s2):
    def solve(p1, p2):
        if p1 == len(s1) and p2 == len(s2):  # base case
            return 0
        elif p1 == len(s1):  # do insertions to match rest of s2
            return len(s2) - p2
        elif p2 == len(s2):  # delete remainder from s1
            return len(s1) - p1
        else:
            return min(
                solve(p1 + 1, p2) + 1,   # deletion from s1
                solve(p1, p2 + 1) + 1,   # insertion into s1
                solve(p1 + 1, p2 + 1) + (  # substitution if needed
                    0 if s1[p1] == s2[p2] else 1))
    return solve(0, 0)
```



```python
def edit_distance(s1, s2):
    memo = {}
    def solve(p1, p2):
        if (p1, p2) in memo:
            return memo[(p1, p2)]
        if p1 == len(s1) and p2 == len(s2):  # base case
            return 0
        elif p1 == len(s1):  # do insertions to match rest of s2
            return len(s2) - p2
        elif p2 == len(s2):  # delete remainder from s1
            return len(s1) - p1
        else:
            ans = min(
                solve(p1 + 1, p2) + 1,    # deletion from s1
                solve(p1, p2 + 1) + 1,    # insertion into s1
                solve(p1 + 1, p2 + 1) + (  # substitution if needed
                    0 if s1[p1] == s2[p2] else 1))
            memo[(p1, p2)] = ans
            return ans
    return solve(0, 0)
```

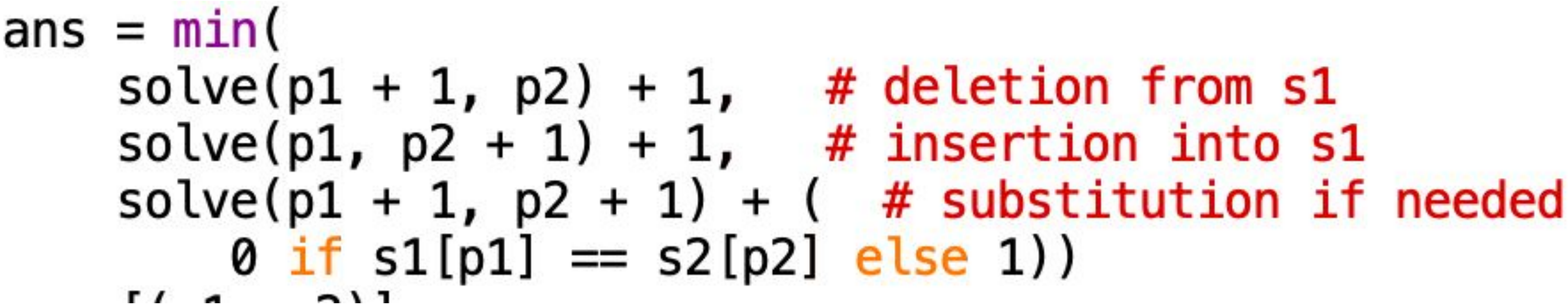
We could have used a 2D array for memo instead of a dictionary. I was just being very lazy on a first pass.



```
def edit_distance(s1, s2):
    memo = {}
    def solve(p1, p2):
        if (p1, p2) in memo:
            return memo[(p1, p2)]
        if p1 == len(s1) and p2 == len(s2):  # base case
            return 0
        elif p1 == len(s1):  # do insertions to match rest of s2
            return len(s2) - p2
        elif p2 == len(s2):  # delete remainder from s1
            return len(s1) - p1
        else:
            ans = min(
                solve(p1 + 1, p2) + 1,   # deletion from s1
                solve(p1, p2 + 1) + 1,   # insertion into s1
                solve(p1 + 1, p2 + 1) + (  # substitution if needed
                    0 if s1[p1] == s2[p2] else 1))
            memo[(p1, p2)] = ans
            return ans
    return solve(0, 0)
```

This is ***top-down*** *DP with* ***memoization****. It's easier to write, but less efficient due to the larger call stack.*

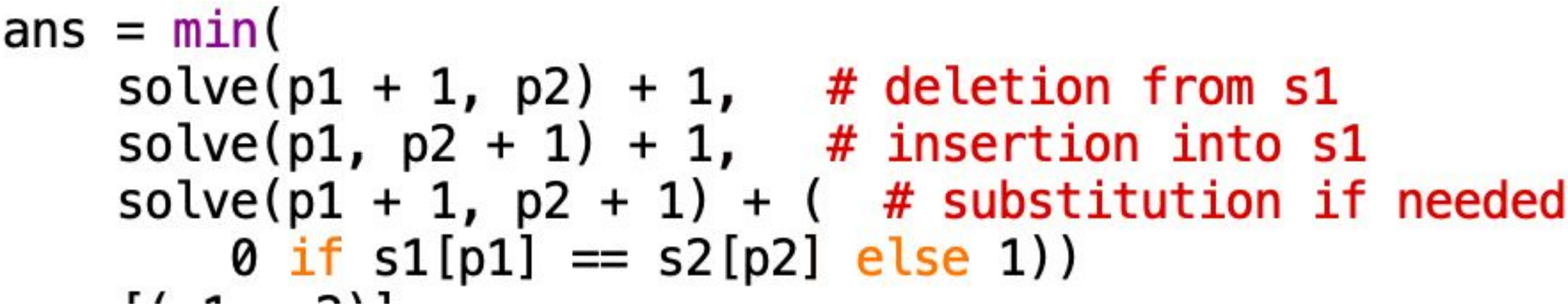
Running time



```
ans = min(
    solve(p1 + 1, p2) + 1,    # deletion from s1
    solve(p1, p2 + 1) + 1,    # insertion into s1
    solve(p1 + 1, p2 + 1) + (  # substitution if needed
        0 if s1[p1] == s2[p2] else 1))
```

Notice that every choice advances at least one pointer.

Running time



```
ans = min(
    solve(p1 + 1, p2) + 1,    # deletion from s1
    solve(p1, p2 + 1) + 1,    # insertion into s1
    solve(p1 + 1, p2 + 1) + (  # substitution if needed
        0 if s1[p1] == s2[p2] else 1))
```

Notice that every choice advances at least one pointer.

The pointers can only go so far, and there is no backtracking, so the running time is **$O(L_1L_2)$**, where L_1 and L_2 are the lengths of the two words. (There are $L_1 + 1$ places the first pointer could be, and $L_2 + 1$ places the second pointer could be, so the product is $O(L_1L_2)$.)

Space

What is the space complexity of this algorithm?

Space

What is the space complexity of this algorithm?

We memoize a result for each state, and there are O(L_1L_2) states, so this is also **O(L_1L_2)**.

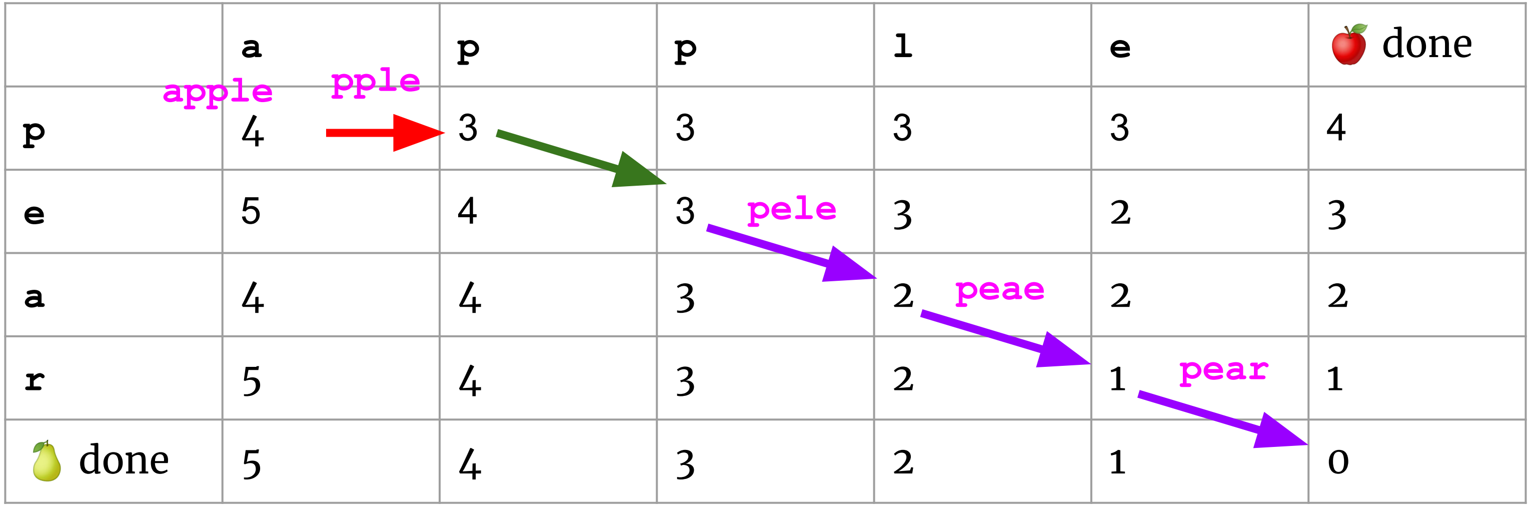
The intermediate results

A value in the table is the cost (in number of operations) of solving from that state.

	a	p	p	l	e	🍎 done
p	4	3	3	3	3	4
e	5	4	3	3	2	3
a	4	4	3	2	2	2
r	5	4	3	2	1	1
🍐 done	5	4	3	2	1	0

The intermediate results

A value in the table is the cost (in number of operations) of solving from that state.



apple
pple
pele
peae
pear

	a	p	p	l	e	🍎 done
p	4	3	3	3	3	4
e	5	4	3	3	2	3
a	4	4	3	2	2	2
r	5	4	3	2	1	1
🍐 done	5	4	3	2	1	0

7/27 Lecture Agenda

- Announcements (including midterm discussion)
- Part 5-3: Edit Distance
- 10 minute break!
- Part 5-4: Knapsack

7/27 Lecture Agenda

- Announcements (including midterm discussion)
- Part 5-3: Edit Distance
- 10 minute break!
- Part 5-4: Knapsack

Pack That Knapsack

Divide and Conquer

Sorting & Randomization

Data Structures

Graph Search

Dynamic Programming

Greed & Flow

Special Topics