I like parentheticals (a lot).

"Sometimes (when I nest them (my parentheticals) too much (like this (and this))) they get confusing."

Write a function that, given a sentence like the one above, along with the position of an opening parenthesis, finds the corresponding closing parenthesis.

Example: if the example string above is input with the number 10 (position of the first parenthesis), the output should be 79 (position of the last parenthesis).

Gotchas

We can do this in $O(n)$ time.

We can do this in $O(1)$ additional space.

Breakdown

How would you solve this problem by hand with an example input?

Try looping through the string, keeping a count of how many open parentheses we have.

Solution

We simply walk through the string, starting at our input opening parenthesis position. As we iterate, we keep a count of how many additional "(" we find as `open_nested_parens`. When we find a ")" we decrement `open_nested_parens`. If we find a ")" and `open_nested_parens` is 0, we know that ")" closes our initial "(", so we return its position.

Python 3.6

```python
def get_closing_paren(sentence, opening_paren_index):
    open_nested_parens = 0

    for position in range(opening_paren_index + 1, len(sentence)):
        char = sentence[position]

        if char == '(':
            open_nested_parens += 1
        elif char == ')':
            if open_nested_parens == 0:
                return position
            else:
                open_nested_parens -= 1

    raise Exception("No closing parenthesis :(")
```

Complexity

$O(n)$ time, where n is the number of chars in the string. $O(1)$ space.

The for loop with `range` keeps our space cost at $O(1)$. It might be more Pythonic to use:

Python 2.7

```python
for char in sentence[position:]:
```

but then our space cost would be $O(n)$, because in the worst case `position` would be 0 and we'd take a slice of the entire input.

What We Learned

The trick to many "parsing" questions like this is *using a stack* to track which brackets/phrases/etc are "open" as you go.

So next time you get a parsing question, one of your first thoughts should be "use a stack!"

In *this* problem, we can realize our stack would only hold '(' characters. So instead of storing each of those characters in a stack, we can store the *number* of items our stack *would be holding*.

That gets us from $O(n)$ space to $O(1)$ space.

It's pretty cool when you can replace a whole data structure with a single integer :)

Share Tweet Share

Editor

Wanna review this one again later? Or do you feel like you got it all?

Mark as done Pin for review later

← course home

Next up: Bracket Validator →

Editor