1221437
Michaels

Homework 02

COSC 2316
14th of February 2021

1 Change Maker

1.1 Psuedocode

For this, I will be make a ChangeMaker class to help manage all of this. In essence, the main reason for this is to separate the monetary system and the actually making of change. The ChangerMaker class will consist of an instance variable denom_list and a method make_change.

```
class ChangeMaker:

    # because of how this particular change making algorithm
    # works, denom_list will need some processing first
    def __init__(self, denom_list: List[int]):
        check that there are no negative denominations

        make sure denom_list contains only unique values

        sort in descending order

        initialise

    def make_change(value_to_make_change: int):
        check that incoming value is not negative

        Instantiate an empty dictionary mapping the denomination
        to the amount required to make the change

        for each denomination in denom_list
            Divide the amount needed to be changed by the denomination

            Map that amount to the corresponding denomination

            Update the amount needing to be changed by subtracting
            the (denomination * amount calculated above)

        return dictionary
```

1.2 Runtime Analysis

$f(n) = O(len(denom\_list))$ as $n \to \infty$. We only need to iterate through the amount of denominations there are. No other non-constant operations are made.

$f(n) = \Omega(len(denom\_list))$ as $n \to \infty$. There is no early exit condition that would cause the algorithm to end earl. This is because we still need to assign a zero count to the denominations after we've hit 0 for the amount left that needs to be changed.

$f(n) = \theta(len(denom\_list))$ as $n \to \infty$. This is because $O(h(n)) = \Omega(h(n))$

1.3 Greedy Approach Non-optimal Cases

1.3.1 denom_list = [21, 11, 7, 1]

```python
denom_list: List[int] = [21, 11, 7, 1]
change_maker: ChangeMaker = ChangeMaker(denom_list=denom_list)

optimal: Dict[int, int] = {21: 0, 11: 0, 7: 2, 1: 1}
actual: Dict[int, int] = change_maker.make_change(15)

optimal_count: int = sum(x for x in optimal.values())
actual_count: int = sum(x for x in actual.values())

print(optimal_count)
print(actual_count)
print(optimal_count < actual_count)
```

```
>>> 3
>>> 5
>>> True
```

1.3.2 denom_list = [56, 23, 11, 1]

```python
denom_list: List[int] = [56, 23, 11, 1]
change_maker: ChangeMaker = ChangeMaker(denom_list=denom_list)

optimal: Dict[int, int] = {56: 0, 23: 0, 11: 5, 1: 0}
actual: Dict[int, int] = change_maker.make_change(55)

optimal_count: int = sum(x for x in optimal.values())
actual_count: int = sum(x for x in actual.values())

print(optimal_count)
print(actual_count)
print(optimal_count < actual_count)
```

```
>>> 5
>>> 11
>>> True
```

2 Balanced Parentheses

2.1 Psuedocode

The main driver for this function will be with our stack. With the nature of a stack being LIFO, this means we can easily infer a pairing system by adding and removing elements from the string. That is to say, our stack should only ever been populated, when it is, with openers. In the end after we've traversed the entire string and done some popping and adding, our stack will then determine whether the string is indeed balanced.

```
def balance_check(string: str):
    initialise the stack
    for each char in string:
        is the char a closer:
            take a peek at the stack
            if we cannot peak because the stack is empty:
                can already determine not balanced
            else lets use that peeked value
                getting mapping of the peek to see if it's the right opener
                if it isn't the right opener:
                    can already determine not balanced
                else:
                    pop from the stack
        else:
            push the char to the stack

    if the stack is empty:
        balanced
    else:
        not balanced
```

2.2 Runtime Analysis

$f(n) = O(len(string)$ as $n \to \infty$. We might only need to iterate through the length of the string. No other non-constant operations are made.

$f(n) = \Omega(1)$ as $n \to \infty$. There are many different strings that can cause this, but the characteristic of these strings are such that if the first character is closer type, then we immediately know it's not balanced.

There exist no $\theta(g(n))$ for $f(n)$ as $O(h(n)) \neq \Omega(h(n))$.

2.3 Walking Through Examples

Example 1

```
Initialisation:
string = ((()))
stack = []

end of string? no
char = (
is closing? no
    push to stack
stack = [(]

end of string? no
char = (
is closing? no
    push to stack
stack = [(, (]

end of string? no
char3 = (
is closing? no
    push to stack
stack = [(, (, (]

end of string? no
char = )
is closing? yes
    is stack empty? no
```

```
        what's on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(, (]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what's on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what's on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = []

end of string? yes
is stack empty? yes
    is balanced
```

Example 2

```
Initialisation:
string = (()()(()))
stack = []

end of string? no
char = (
is closing? no
    push to stack
stack = [(]

end of string? no
```

```
char = (
is closing? no
    push to stack
stack = [(, (]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what’s on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(]

end of string? no
char = (
is closing? no
    push to stack
stack = [(, (]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what’s on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(]

end of string? no
char = (
is closing? no
    push to stack
stack = [(, (]

end of string? no
char = (
is closing? no
    push to stack
stack = [(, (, (]

end of string? no
```

```
char = )
is closing? yes
    is stack empty? no
        what’s on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(, (]
end of string? no
char = )
is closing? yes
    is stack empty? no
        what’s on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what’s on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = []

end of string? yes
is stack empty? yes
    is balanced
```

Example 3

```
Initialisation:
string = ((()()()())
stack = []

end of string? no
char = (
is closing? no
    push to stack
stack = [(]
end of string? no
char = (
is closing? no
    push to stack
stack = [(, (]

end of string? no
char = (
is closing? no
    push to stack
stack = [(, (, (]

end of string? no
char = (
is closing? no
    push to stack
stack = [(, (, (]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what’s on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(, (]

end of string? no
char = (
is closing? no
    push to stack
stack = [(, (, (]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what’s on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(, (]
end of string? no
char = (
is closing? no
    push to stack
stack = [(, (, (]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what's on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(, (]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what's on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(]

end of string? yes
is stack empty? no
    not balanced
```

Example 4

```
Initialisation:
string = ()(())((()()))
stack = []

end of string? no
char = (
is closing? no
    push to stack
stack = [(]
end of string? no
char = )
is closing? yes
    is stack empty? no
        what’s on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = []

end of string? no
char = (
is closing? no
    push to stack
stack = [(]

end of string? no
char = (
is closing? no
    push to stack
stack = [(, (]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what’s on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what’s on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = []

end of string? no
char = (
is closing? no
    push to stack
stack = [(]

end of string? no
char = (
is closing? no
    push to stack
stack = [(, (]

end of string? no
char = (
is closing? no
    push to stack
stack = [(, (, (]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what's on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(, (]

end of string? no
char = (
is closing? no
    push to stack
stack = [(, (, (]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what's on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(, (]

end of string? no
char = )
```

```
is closing? yes
    is stack empty? no
        what’s on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = [(]

end of string? no
char = )
is closing? yes
    is stack empty? no
        what’s on top of the stack? (
            is this the correct opener? yes
                pop the stack
stack = []

end of string? no
char = )
is closing? yes
    is stack empty?
        not balanced
```