Mod 3 - Running Time Analysis

Identify common bottlenecks in algorithms caused by poor data structure choice, then write better algorithms.

You are provided with 2 "naive" algorithms. We call these naive because they are common first-pass approaches to solving a problem, but can be significantly improved with a bit of thought to our data structures.

Optimization 1 - enrypt a string

`Cypher.cypher_naive()` is a variant of what is commonly know as a "Ceasar cypher". To enrypt a message with this cypher:

- Every letter is shifted according to it's ascii value. You can use `ord()` to get the ascii value of a character, and `chr()` to convert a value to the appropriate character; e.g. with a shift of 5 `'a'` becomes `'f'` and `'C'` becomes `'H'`:

```
>>> shift = 5
>>> ord('a') # ascii value of 'a'
97
>>> chr(97)
'a'
>>> chr(97+shift)
'f'
>>> ord('C')
67
>>> chr(67+shift)
'H'
```

- The string is reversed.
- Every letter in the string is offset left by a specified amount.

For instance, if our initial string is `'abcD'`, our string after shifting 2, reversing, and offsetting 1 is `'cFed'`:

initial	shift(2)	reversed	offset(1)
abcD	cdeF	Fedc	edcF

`cypher_naive()` works, but it is slow - strings are immutable, so shifting our strings and reversing one character at a time is quadratic (we have to rewrite the entire string every time we replace a single letter). To speed things up, use a list as an intermediate step.

1) Add a test case for `cypher_opt()` in `TestCypher`. It should have the same input-output behavior as `cyhper_naive()`.
2) Implement your optimzied ($\mathcal{O}(n)$) algorithm `Cypher.cypher_opt()`.
3) Add another test case to `TestCypher` that uses `cypher_naive()` to test `cypher_opt()` - because `cypher_naive()` is correct, you can use it to test random messages.

Good news! We've already done steps 1 and 3 for you - you just have to do step 2. However, you should take some time to make sure you understand how our test satisfy steps 1 and 3 - it will help you with the rest of this assignment.

Optimization 2 - find_pairs()

`FindPairs.find_pairs_naive(L, target)` takes two inputs - a list of integers and a target interger. The goal is to return a set of all pairs of distinct integers in the list that sum to the target.

```
>>> L = [1,2,3,4,5]
>>> find_pairs_naive(L, 7)
{(2,5), (3, 4)}
```

In the returned pairs:

- the first item should be the one earlier in the list (`(2, 5)` is the correct pair above, not `(5, 2)`, because `2` is before `5` in the passed in list).
- We are only interested in distinct pairs - `{(3,3)}` would not be a valid pair in the list above if the target was 6.

`find_pairs_naive()` works, but it is $\mathcal{O}(n^2)$ - it requires testing membership in a list, which is slow. Rewrite it to use a set as the intermediary container for items you have already visited, which will reduce the running time to $\mathcal{O}(n)$.

1) Implement one or more tests for `find_pairs_naive()` in `TestFindPairs.py`.
2) Once you are satisifed with your unittest(s), implement a test for `find_pairs_optimized()`.
3) Implement `find_pairs_optimized()` as described above.
4) Add another test that uses `find_pairs_naive()` to test `find_pairs_opt()` using a large number of random lists and target values; see part 1 of this assignment for an example.

Imports

No imports allowed on this assignment, with the following exceptions:

- Any modules you have written yourself (e.g. you can import `hw.py` into `TestHw.py`)
- For testing only (do not use these for functionality in any other classes/algorithms):
  - unittest
  - random
  - string
  - time

Submission

Students must submit by the deadline (typically Tuesday at 11:59 PM EST) to recieve credit.

At a minimum, submit the following. you should also submit any other code required for your solution to run (e.g. any modules you write yourself and import in your solution).

- `Cypher.py`
  - `cypher_naive()`
  - `cypher_opt()`
- `TestCypher.py`
  - `class TestCypher()`

        * test_cypher_naive()
        * test_cypher_opt()
        * test_cypher_opt_random()
- FindPairs.py
    - find_pairs_naive()
    - find_pairs_opt()
- TestFindPairs.py
    - class TestFindPairs()
        * test_find_pairs_naive()
        * test_find_pairs_opt()
        * test_find_pairs_opt_random()