Homework 3

Due 11:59 PM on Monday May 24th, 2021

You will implement Boyer-Moore and a Hash Table.
Key instructions:

- 1. Please write your name at the top of each assignment and cite any references you used (including articles, books, code, websites, and personal communications). If you're not sure whether to cite a source, err on the side of caution and cite it. Remember NOT TO PLAGIARIZE: **all solutions must be written by yourself**.
- 3. Please only use Python 3 for this assignment, the grading of this assignment will be done in a Python 3.7.5 environment.
- 4. Please do NOT use any additional imports, only write your code where you see `TODO: YOUR CODE HERE`, and change your return value accordingly.
- 5. Note on Late Policy: You will **NOT** be able to use a slip day on this assignment. You **MUST** submit your assignment by May 24, so that final grades can be submitted on time.
- 6. Early submissions: you may submit your assignment and have it graded early; if you are not satisfied with the score you will be allowed to edit your submission and submit again for a revised grade. If you have not yet used a slip day, you may revise a second time.
- 7. Please modify and submit the following files:
  - `boyer_moore.py`
  - `hashtable_linear_probing.py`
  - `hashtable_chaining.py`

Boyer-Moore Majority Vote algorithm

In part 1 of HW3, you will implement the Boyer-Moore Majority Vote algorithm as described in lecture.

TODO In the provided class (`boyer_moore.py`), implement

- `add_next_element`
- `get_majority`

You may assume in your code and do not need to check that each call to `add_next_element` will be a single 1-character string drawn from the upper- and lowercase alphabet. We will check that the values of `counter` and `guess` are correct, although we will not check the value of `guess` when `counter` is zero.

Testing Boyer-Moore

When you run `python3 test_boyer_moore.py`, by default it should terminate when 5 cases failed[1]. You can change this behavior by changing `max_test_failures`, or by passing command-line arguments(e.g. `python3 test_boyer_moore.py --max-test-failures 10`). In addition to the final result, we will also grade your code by checking if the counter is correct **at each step**, and if the guess is correct at each step where the counter is not 0.
The sample test cases are in (`testcases_boyer_moore.txt`). Each line has 4 semicolon-delimited fields, which are:

`test_name`	name of the test
`symbols`	string of symbols, comma-delimited
`expected_guess`	symbol, or '!'
`expected_counter`	integer

Hash Table

Implementing a hash table in two different ways

(1) Linear Probing

TODO In the provided class (`hashtable_linear_probing.py`), implement

- `insert`
- `get`

Please read the comments in the file carefully to understand our expectations for inputs and outputs. You ***must*** use linear probing to handle collisions in this implementation. Note that you also need to resize the underlying array when appropriate to ensure that there is always room to insert a new mapping (you can use the provided _resize_array function). An array data structure is provided, as is a hash function `cs5112_hash`.

> ***Do not use Python's built-in hash function! You must use ours (*`cs5112_hash`*).***
>
> ***You must use the provided*** `Fixed_Size_Array` ***type to implement your hash table.***

(2) Chaining (with a linked list)

TODO In the provided class (`hashtable_chaining.py`), implement

- `insert`
- `get`

You ***must*** use chaining with a linked list to handle collisions in this implementation. Again, you will need to resize the underlying array at the appropriate time (you can use the provided _resize_array function). The linked list data structure is provided to you, as is an array data structure and a hash function `cs5112_hash`.

> ***Do not use Python's built-in hash function! You must use ours (*`cs5112_hash`*).***
>
> ***You must use the provided*** `Fixed_Size_Array` ***type to implement your hash table.***

[1] Some cases may pass "by luck" even when you haven't implemented the algorithm yet, because we initialize `guess` as `None` and `counter` as `0`.

You must use the provided linked list node data structure to implement chaining.

Grading Note: In addition to testing the output of your functions, we will also be inspecting how you're storing data in the underlying array/linked-list. Therefore, be sure to implement the algorithms to spec. e.g. For your chaining hash table, you must make sure to insert elements at the *BEGINNING* of any given linked-list chain.

To ensure compatibility with our grading software, please ensure that the provided test files run. You should be able to run the following without errors:

```
python3 hashtable_test.py
```