Sebastian Kurpiel & Cat Litten

Team L Project One

21st September 2017 / COMP330

Data Structures

Dictionaries

- Dictionaries were a key component of this project, with there being seven in total.
    1. Notes_dictionary -- this was the main structure which many of the other methods relied upon. It is also the only dictionary which was created as and stayed an *order dictionary*. key/value relationship for this dictionary was filename : document text
    2. Agg_counts was a simple dictionary which kept a running tally of how many ‘hits’ there were for any given search. The key within the dictionary related to an integer which incremented with each found match
    3. Found_words was based on the same concept as agg_counts, however it was utilized within the method find_specific_word and this the key/value relationship was filename : number of words matching the non-standard regex query
    4. Frequent_words was another counting dictionary. The key/value relationship here was word : number of instances within the entire body of text.
    5. Connections was instituted as a dictionary to assist the method compare_hash_to_bang. The key/value relationship was filename : ‘^’ matches
    6. Note_identifier was also a helper dictionary to compare_hash_to_bang. The key/value relationship was filename : ‘!’ match
    7. Runnable_reports was a way to store neatly the association between search inquiries as menu options and their regex search parameters. The key/value relationship was menu option : regex expression

Arrays

- There were only three arrays utilized within this project
    1. Report_choices stored as a string value the report options presented to the user.
    2. The values stored within agg_count was an array of matches
    3. The values stored within connections were also arrays

Graph

- A graph was implemented briefly in Python, but the meaning of topological sort for this project was never fully clear and progress lagged. Ultimately an ordered dictionary for the notes was chosen as it at least allowed for sequential reporting. The code remains however to implement a graph, but not to traverse it.

Project Flow

The user is presented with options which feed into various methods depending on the desired report run. We will cover these in order:

[0] hashtags, [1] dollar signs, [2] unique identifiers, [3] mentions, [4] URLs, [5] carots

- This will go to runnable_reports to retrieve the regex parameter which it will then give as input along with the name of the search, the file text, and file name helped along by main_search supplying many of these variables through reading notes_dictionary key and value pairs.

[6] which notes reference other notes

- This will run three methods sequentially: parse_notes_for_identifiers, parse_notes_for_carots, and compare_carot_to_bang. These methods will collect the ‘^’ and ‘!’ matches and run through the values of ‘!’ to see if any of the ‘^’ arrays contain a match.

[7] most frequently used words

- This will directly run the search most_frequent_words which parse the documents for all words and keeps a dictionary tally of them. The dictionary is then reverse ordered to have the highest value keys at the start and the beginning ten key/value pairs are printed

[8] search a specific mention, keyword, or general word

- This will directly run the method find_specific_word which immediately prompts the user with another menu asking if they are looking for mentions (@), keywords (#), or just any words within the documents. The user then chooses their search and appropriate regex is formed to run a report.

[9] report all keyword types

- This is a for loop operating on notes_dictionary which performs all the reports of options 0 through 5.

[10] done running reports

- This will print an exit message and stop the program.

Testing

Overall testing using unittest in python is pretty straight forward.

We would grab the regex equations from the main program and input them into basic sudo code below.

The basic sudo code for each regex test went as follows:

```
#def test_something(self)

# make test_note="to what you're using regex for"

#make bad_note=”tow what you don’t want it to read”

#self.assertRegex("to what you set test_note")

#self.assertNotTRegex(“to what you set bad_note to”)
```

After they ran and passed, we would manually test the open text function in the main code by seeing if opened in the terminal. We also went through all the options to see if any bug could’ve been spotted. This in our opinion was easier than writing an elaborate code and the timed saved was put into improving the main code.