

CS 562 Progress Report

Haoxiang Wang, Student ID:932359049

For this testing project, a third party python library named “*fuzzywuzzy*” will be tested. The library is implemented to accomplish the fuzzy string matching, and it can be found here on the Github: <https://github.com/seatgeek/fuzzywuzzy>. In this library, several functions are provided in order to handle several different situations. Up to the date writing this report, half of the functions are tested and fuzzywuzzy performed great job on the tests. Only one bug was founded during the test.

To introduce the progress has been made, the functions had been tested will be introduced first. The functions that had been tested up to the report time are “*fuzz.ratio*”, “*fuzz.partial_ratio*”, and “*fuzz.token_sort_ratio*”. Two tsl files were written to test these three functions, and they will be explained in the following paragraphs.

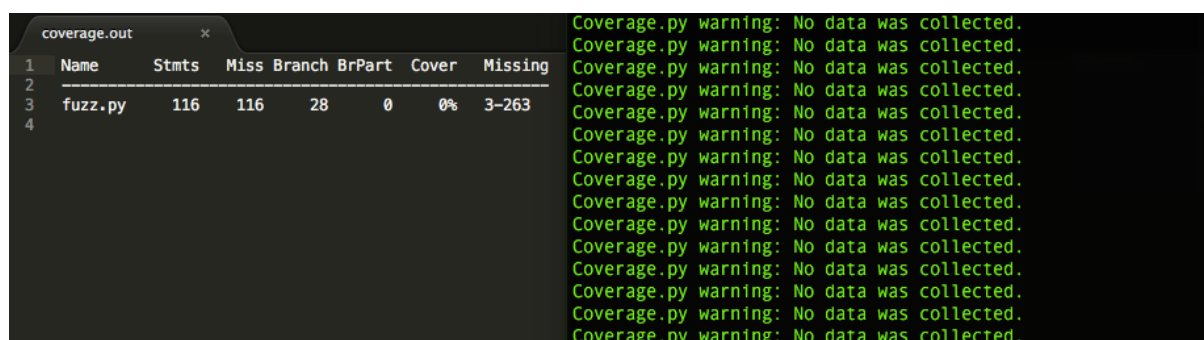
The first tsl file creates the strings contains lower and upper cases of letters along with special symbols with some common parts and passes them to “*fuzz.ratio*” and “*fuzz.partial_ratio*” functions. The “*fuzz.ratio*” function will test the most common part of two strings and return the similarity percentage. While “*fuzz.ratio*” function takes two entire string into account, “*fuzz.partial_ratio*” function only tests part of the strings. If string2 is part of the string1, “*fuzz.partial_ratio*” will return 100 percent match instead of a lower percentage returned by “*fuzz.ratio*”. Therefore, for two strings have a lot common part, the “*fuzz.partial_ratio*” will always return a higher percentage than “*fuzz.ratio*” does. The result turned out that both of these two functions can handle the strings perfectly, and the “*fuzz.partial_ratio*” always returns higher match percentage than “*fuzz.ratio*” does as expected. These two functions worked just fine and no bug is found.

The second tsl file creates the strings with a pretty similar way as the first one does. This file takes “*fuzz.token_sort_ratio*” and “*fuzz.ratio*” into test. The “*fuzz.token_sort_ratio*” ignores the order of “words” in the string. The “words” are separated by “space” in the strings. Therefore, the only difference lies in this file’s creating strings part is taking “space” into account. Considering the fact that “*fuzz.token_sort_ratio*” will sort the words first and then execute the comparing, when string2 contains similar “words” with different order as the string1 does, “*fuzz.token_sort_ratio*” will return a higher percentage than “*fuzz.ratio*” does. Since TSTL executes the commands randomly, there exists the situation that the strings only contain one “word”. Under this situation, these two functions should return the same percentage with respect to their working logics. However, during the test, when the “*fuzz.token_sort_ratio*” is dealing with the strings contain special symbols, it will return a lower percentage than the “*fuzz.ratio*” does. This situation is really obvious when the strings only have one “word”. This result shows that “*fuzz.token_sort_ratio*” could not deal with the special symbols well, and this is a BUG!

Comparing with the pretty raw tstl file written in the project part2, the files that had been used this time cover more situations than the previous one did. They test the situation that strings containing special symbols and space, and by implementing this, a bug is found during the test. This relates to the deeper understanding and more proficient handling of TSTL. However, the project has not been completed yet. There still exists other three functions waiting for the test, and they are *"fuzz.token_set_ratio"*, *"process.extract"*, and *"process.extractOne"*. In these three functions, *"fuzz.token_set_ratio"* provides similar functionality as *"fuzz.token_sort_ratio"* does, so the same bug that found in the *"fuzz.token_sort_ratio"* will be expected also exists in the *"fuzz.token_sort_ratio"*. At the same time, *"process.extract"* and *"process.extractOne"* are from another library named "process", and they implement more complex functionalities than previous functions. Therefore, some new bugs are expected to be found in these two functions. By the end of the term, all of the functions will be tested, and several new bugs are expected.

To sum up the performance of fuzzywuzzy, it does a great job up to the date. Only one bug is found during the test, and other than that, everything is functional and performs a good result. However, the bug that has been found is sort of stupid. The *"fuzz.ratio"* and the *"fuzz.partial_ratio"* functions could handle the special symbols perfectly, while the *"fuzz.token_sort_ratio"* function could not. The only reason I could imagen why this bug exists is *"fuzz.token_sort_ratio"* messes up the "space" with other special symbols. Every time it meets a special symbol, *"fuzz.token_sort_ratio"* treats it as space and sorts it. This will chop one "word" into apart and causes lower matching percentage.

Up to the date, three out of six functions are tested, and there are still other three remaining. The code coverage is impossible to test under the condition I have. Might due to the version of Coverage.py installed in my Mac, the TSTL keep returning the warning "Coverage.py warning: No data was collected." Also, non randomtester nor coverage.out has the useful information for the code coverage evaluation. An image is listed below to show the result I got during the all of my tests.



1	Name	Stmts	Miss	Branch	BrPart	Cover	Missing
2							
3	fuzz.py	116	116	28	0	0%	3-263
4							

Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.
 Coverage.py warning: No data was collected.

Testing the library using TSTL is a fun job to do. Coming up a way to create the required strings to test is a challenging but interesting work. I'm looking forward to the future work and willing to finish all of the tests on fuzzywuzzy in this project.