FuzzyWuzzy Testing Framework: Ratios

Wyatt Morris, Chance Leaird, Juan Carlos Crispi

CSCI 362

Dr. Bowring

1 December 2015

Table of Contents

Introduction:

Our group is The Darkness, and the members of The Darkness are Wyatt Morris, Chance Leaird and Juan Carlos Crispi. Our final report will include each deliverable we have submitted throughout the semester. The deliverables are intended to express our progress we made on our testing framework. The software that we developed a testing framework for is called FuzzyWuzzy. FuzzyWuzzy is software that is intended to compare two strings and return their ratio comparing the similarities between the two strings. There are several different attributes of the software that will return a different type of ratio depending on the attribute you choose. In order to test FuzzyWuzzy, we built a testing framework that tests five different attributes by comparing the expected output of the program with the actual output of the program.

Chapter 1:

Selection of HFOSS Project

Evergreen is a software that helps libraries handle, manage, and gather materials. This was our initial choice of HFOSS project. It is a program made for libraries of varying size. One of the main prerequisites for Evergreen to run is the routing network called openSRF. It was when The Darkness tried to run openSRF, that we realized we were in over our heads. OpenSRF requires an xmpp; the choice for their project was Jabber. Jabber is necessary for two networks to communicate and it lets the networks communicate at almost real time. We could not get openSRF to run once we configured it, and our team has decided to switch HFOSS software for our tests.

Since we failed to get Evergreen running, we chose a different open source project called FuzzyWuzzy. FuzzyWuzzy is a software that will take two string inputs, match them up, and return the ratio of the two strings. The ratio is determined by trying to match up the characters in each string with one another. We were able to get FuzzyWuzzy cloned and running much easier; the software is written in python.

Chapter 2:

Test Plan

Our experiences thus far into the project have mostly consisted of us trying to figure out how to test certain methods in the fuzz.py component. We eventually figured out how to pass our own two strings in as parameters and get the ratio returned to us on our terminal. After this we decided what our test cases would be and created our testing process. Our testing process will involve us testing five different methods of the fuzz component. We will test these methods by comparing the value that the program returns with an expected value that the program is supposed to return. We have created a schedule with completion dates for the testing script, and each of the tests that we will eventually run.

Chapter 3:

Testing Framework Architecture

Architectural Description of Framework

Our framework is intended run all the tests located in the /testCases folder; there are 25 testCases in the /testCases folder. It does this in the command line by execution of ./ runAllTests.sh. Once this has been run, runAllTests.sh accesses the test.py file which pulls the test identification number, the component, the requirements, the inputs, and the expected outputs from each .txt file in the /testCases folder. Once it gathers this information, these parameters are passed through to the driver, and the tests are run on the ratio method (or other specified method) in fuzz.py. The expected outcome and the actual outcome are both printed.

Framework Structure

```
/TestAutomation
    /Project_files
    /scripts
        runAllTests.sh
        test.py
    /testCases
        testCase1.txt
        testCase2.txt
        testCase3.txt
        testCase4.txt
        testCase5.txt
    /testCasesExecutables
    /temp
    /oracles
```

/docs

README.txt

/reports

testReport.html

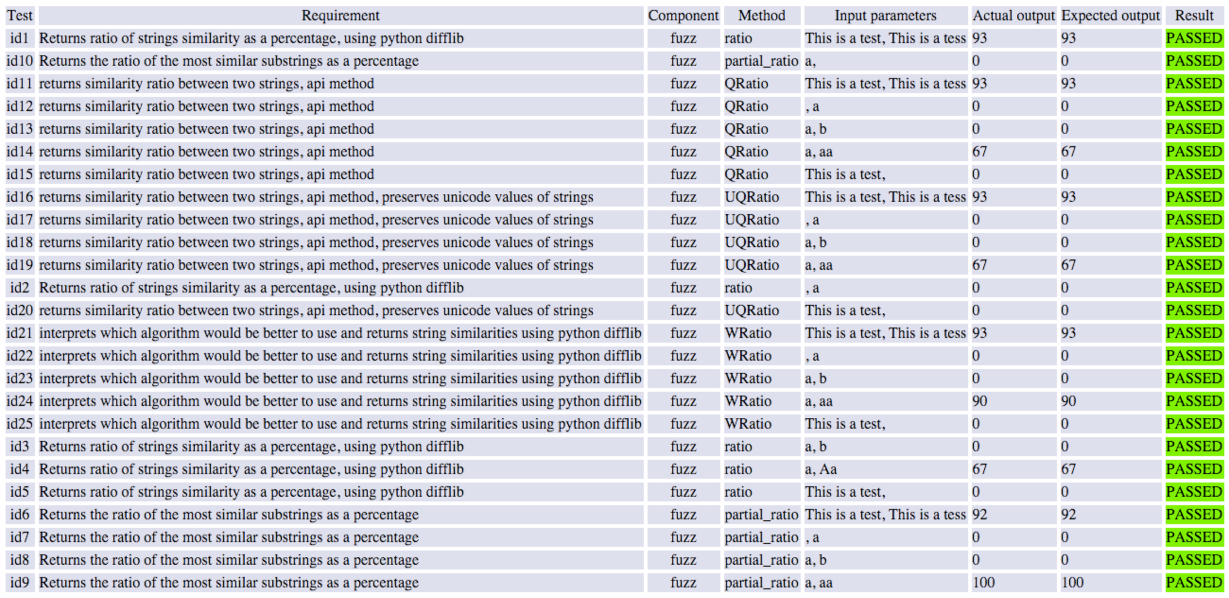
How To

Python 2.7 or greater must be installed on the system that will be executing the test cases. After python is installed, fuzzywuzzy must be installed. This can be done by navigating to the project_files folder and typing the command “sudo python setup.py install”. After fuzzywuzzy is installed, “./runAllTests.sh”. This will execute all test cases in the /testCases directory.

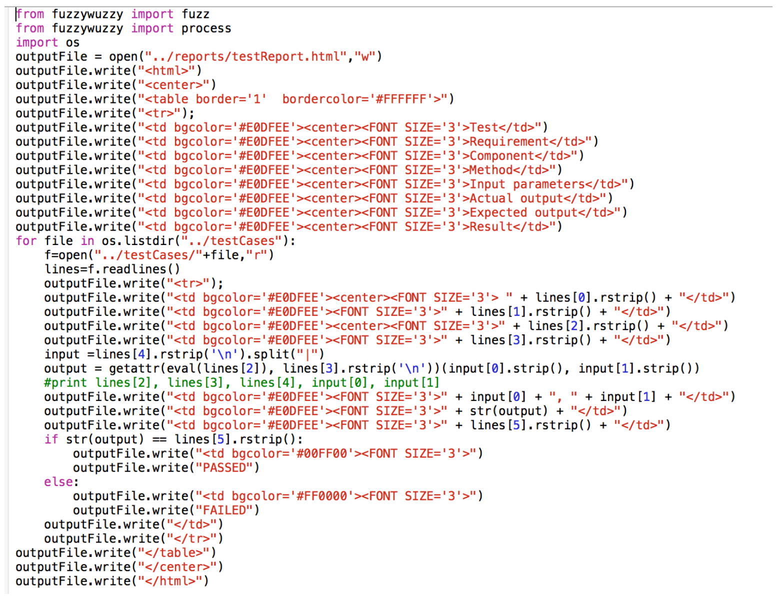
Chapter 4:

Creating All Test Cases

Once we had our script created to read test files, and test the software it was not of great difficulty for us to finish creating the rest of the 25 test cases. In our 25 test cases we are testing the methods ratio, partial_ratio, QRatio, UQRatio, and WRatio. The ratio method takes the exact ratio of the two strings, comparing each character and its position in the string. The partial_ratio method takes the ratio of the two strings, but as long as one of the two full strings is partially contained in the other string, the method will return 100 (equivalence) if partial_ratio is called. QRatio and UQRatio are API methods that run the same algorithm as the ratio method, however UQRatio preserves unicode values. WRatio has a selection algorithm that determines whether to use ratio or partial_ratio algorithms for string comparison. For example, if we were to run the ratio method on the two strings "test" and "tests" the return value would be 80; if we were to run the method partial_ratio on the strings "test" and "tests" the return value would be 100. Our tests have gone smoothly with many passed tests. When we created our 20 new test cases with the different method our script worked fine. Each of the methods takes two parameters, being two strings. There is one case that fails currently (testid 04) which we are still trying to figure out. Our experience thus far has been difficult be we have completed everything required up to this deadline.



Test	Requirement	Component	Method	Input parameters	Actual output	Expected output	Result
id1	Returns ratio of strings similarity as a percentage, using python difflib	fuzz	ratio	This is a test, This is a tess	93	93	PASSED
id10	Returns the ratio of the most similar substrings as a percentage	fuzz	partial_ratio	a,	0	0	PASSED
id11	returns similarity ratio between two strings, api method	fuzz	QRatio	This is a test, This is a tess	93	93	PASSED
id12	returns similarity ratio between two strings, api method	fuzz	QRatio	, a	0	0	PASSED
id13	returns similarity ratio between two strings, api method	fuzz	QRatio	a, b	0	0	PASSED
id14	returns similarity ratio between two strings, api method	fuzz	QRatio	a, aa	67	67	PASSED
id15	returns similarity ratio between two strings, api method	fuzz	QRatio	This is a test,	0	0	PASSED
id16	returns similarity ratio between two strings, api method, preserves unicode values of strings	fuzz	UQRatio	This is a test, This is a tess	93	93	PASSED
id17	returns similarity ratio between two strings, api method, preserves unicode values of strings	fuzz	UQRatio	, a	0	0	PASSED
id18	returns similarity ratio between two strings, api method, preserves unicode values of strings	fuzz	UQRatio	a, b	0	0	PASSED
id19	returns similarity ratio between two strings, api method, preserves unicode values of strings	fuzz	UQRatio	a, aa	67	67	PASSED
id2	Returns ratio of strings similarity as a percentage, using python difflib	fuzz	ratio	, a	0	0	PASSED
id20	returns similarity ratio between two strings, api method, preserves unicode values of strings	fuzz	UQRatio	This is a test,	0	0	PASSED
id21	interprets which algorithm would be better to use and returns string similarities using python difflib	fuzz	WRatio	This is a test, This is a tess	93	93	PASSED
id22	interprets which algorithm would be better to use and returns string similarities using python difflib	fuzz	WRatio	, a	0	0	PASSED
id23	interprets which algorithm would be better to use and returns string similarities using python difflib	fuzz	WRatio	a, b	0	0	PASSED
id24	interprets which algorithm would be better to use and returns string similarities using python difflib	fuzz	WRatio	a, aa	90	90	PASSED
id25	interprets which algorithm would be better to use and returns string similarities using python difflib	fuzz	WRatio	This is a test,	0	0	PASSED
id3	Returns ratio of strings similarity as a percentage, using python difflib	fuzz	ratio	a, b	0	0	PASSED
id4	Returns ratio of strings similarity as a percentage, using python difflib	fuzz	ratio	a, Aa	67	67	PASSED
id5	Returns ratio of strings similarity as a percentage, using python difflib	fuzz	ratio	This is a test,	0	0	PASSED
id6	Returns the ratio of the most similar substrings as a percentage	fuzz	partial_ratio	This is a test, This is a tess	92	92	PASSED
id7	Returns the ratio of the most similar substrings as a percentage	fuzz	partial_ratio	, a	0	0	PASSED
id8	Returns the ratio of the most similar substrings as a percentage	fuzz	partial_ratio	a, b	0	0	PASSED
id9	Returns the ratio of the most similar substrings as a percentage	fuzz	partial_ratio	a, aa	100	100	PASSED

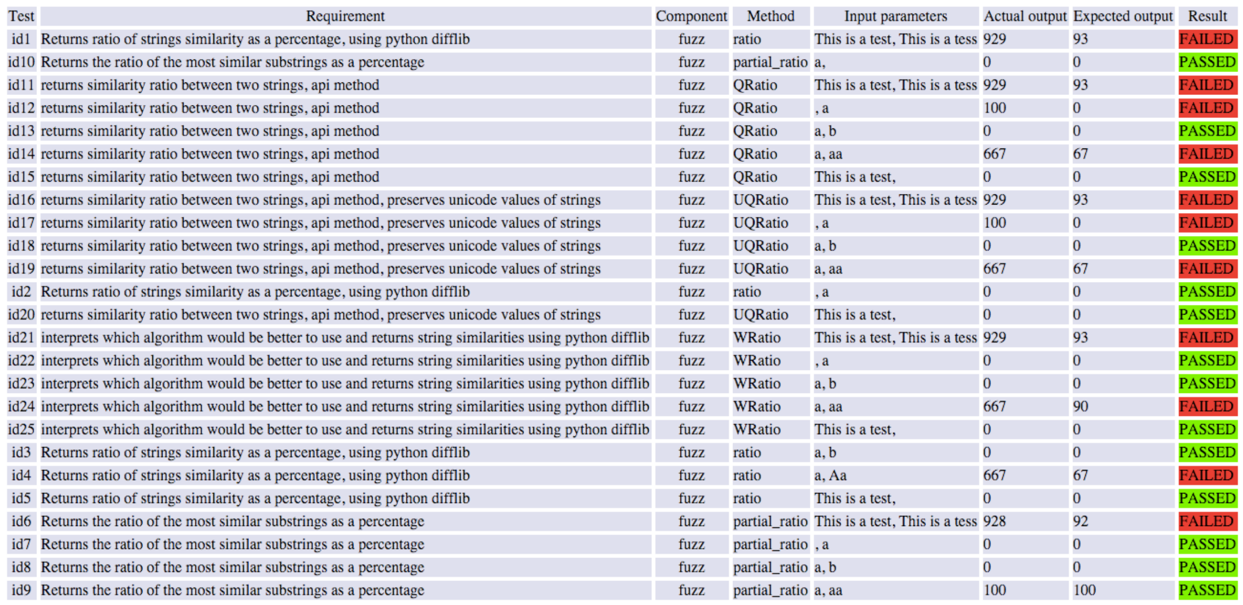


```
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
import os
outputFile = open("../reports/testReport.html","w")
outputFile.write("<html>")
outputFile.write("<center>")
outputFile.write("<table border='1'  bordercolor='#FFFFFF'>")
outputFile.write("<tr>");
outputFile.write("<td bgcolor='#E0DFEE'><center><FONT SIZE='3'>Test</td>")
outputFile.write("<td bgcolor='#E0DFEE'><center><FONT SIZE='3'>Requirement</td>")
outputFile.write("<td bgcolor='#E0DFEE'><center><FONT SIZE='3'>Component</td>")
outputFile.write("<td bgcolor='#E0DFEE'><center><FONT SIZE='3'>Method</td>")
outputFile.write("<td bgcolor='#E0DFEE'><center><FONT SIZE='3'>Input parameters</td>")
outputFile.write("<td bgcolor='#E0DFEE'><center><FONT SIZE='3'>Actual output</td>")
outputFile.write("<td bgcolor='#E0DFEE'><center><FONT SIZE='3'>Expected output</td>")
outputFile.write("<td bgcolor='#E0DFEE'><center><FONT SIZE='3'>Result</td>")
for file in os.listdir("../testCases"):
    f=open("../testCases/"+file,"r")
    lines=f.readlines()
    outputFile.write("<tr>");
    outputFile.write("<td bgcolor='#E0DFEE'><center><FONT SIZE='3'> " + lines[0].rstrip() + "</td>")
    outputFile.write("<td bgcolor='#E0DFEE'><FONT SIZE='3'>" + lines[1].rstrip() + "</td>")
    outputFile.write("<td bgcolor='#E0DFEE'><center><FONT SIZE='3'>" + lines[2].rstrip() + "</td>")
    outputFile.write("<td bgcolor='#E0DFEE'><FONT SIZE='3'>" + lines[3].rstrip() + "</td>")
    input =lines[4].rstrip('\n').split("|")
    output = getattr(eval(lines[2]), lines[3].rstrip('\n'))(input[0].strip(), input[1].strip())
    #print lines[2], lines[3], lines[4], input[0], input[1]
    outputFile.write("<td bgcolor='#E0DFEE'><FONT SIZE='3'>" + input[0] + ", " + input[1] + "</td>")
    outputFile.write("<td bgcolor='#E0DFEE'><FONT SIZE='3'>" + str(output) + "</td>")
    outputFile.write("<td bgcolor='#E0DFEE'><FONT SIZE='3'>" + lines[5].rstrip() + "</td>")
    if str(output) == lines[5].rstrip():
        outputFile.write("<td bgcolor='#00FF00'><FONT SIZE='3'>")
        outputFile.write("PASSED")
    else:
        outputFile.write("<td bgcolor='#FF0000'><FONT SIZE='3'>")
        outputFile.write("FAILED")
    outputFile.write("</td>")
    outputFile.write("</tr>")
outputFile.write("</table>")
outputFile.write("</center>")
outputFile.write("</html>")
```

Chapter 5:

Fault Injection

We have injected 5 faults into the fuzz component of the project. Each fault is injected into methods that we are testing in our testCase files. We injected two different faults into the ratio method; each fault causes the majority of the tests to fail. We inject these faults by altering calculations being done in the methods such as changing integer values and altering arithmetic expressions in calculations. There is another fault in partial_ratio that also changes an integer value that is used for calculations in computing string ratios. The last two faults are in qratio; in order to inject fault into this method we altered return statements by substituting the string ratio values with a constant value that is returned. These faults prove that our script can differentiate between a test that has failed and a test that has passed. For The Darkness, this was one of the easiest assignments of the project. It was a process of trial and error in order to find out what would make our code break, and eventually we figured out what would cause our tests to fail.



Test	Requirement	Component	Method	Input parameters	Actual output	Expected output	Result
id1	Returns ratio of strings similarity as a percentage, using python difflib	fuzz	ratio	This is a test, This is a tess	929	93	FAILED
id10	Returns the ratio of the most similar substrings as a percentage	fuzz	partial_ratio	a,	0	0	PASSED
id11	returns similarity ratio between two strings, api method	fuzz	QRatio	This is a test, This is a tess	929	93	FAILED
id12	returns similarity ratio between two strings, api method	fuzz	QRatio	, a	100	0	FAILED
id13	returns similarity ratio between two strings, api method	fuzz	QRatio	a, b	0	0	PASSED
id14	returns similarity ratio between two strings, api method	fuzz	QRatio	a, aa	667	67	FAILED
id15	returns similarity ratio between two strings, api method	fuzz	QRatio	This is a test,	0	0	PASSED
id16	returns similarity ratio between two strings, api method, preserves unicode values of strings	fuzz	UQRatio	This is a test, This is a tess	929	93	FAILED
id17	returns similarity ratio between two strings, api method, preserves unicode values of strings	fuzz	UQRatio	, a	100	0	FAILED
id18	returns similarity ratio between two strings, api method, preserves unicode values of strings	fuzz	UQRatio	a, b	0	0	PASSED
id19	returns similarity ratio between two strings, api method, preserves unicode values of strings	fuzz	UQRatio	a, aa	667	67	FAILED
id2	Returns ratio of strings similarity as a percentage, using python difflib	fuzz	ratio	, a	0	0	PASSED
id20	returns similarity ratio between two strings, api method, preserves unicode values of strings	fuzz	UQRatio	This is a test,	0	0	PASSED
id21	interprets which algorithm would be better to use and returns string similarities using python difflib	fuzz	WRatio	This is a test, This is a tess	929	93	FAILED
id22	interprets which algorithm would be better to use and returns string similarities using python difflib	fuzz	WRatio	, a	0	0	PASSED
id23	interprets which algorithm would be better to use and returns string similarities using python difflib	fuzz	WRatio	a, b	0	0	PASSED
id24	interprets which algorithm would be better to use and returns string similarities using python difflib	fuzz	WRatio	a, aa	667	90	FAILED
id25	interprets which algorithm would be better to use and returns string similarities using python difflib	fuzz	WRatio	This is a test,	0	0	PASSED
id3	Returns ratio of strings similarity as a percentage, using python difflib	fuzz	ratio	a, b	0	0	PASSED
id4	Returns ratio of strings similarity as a percentage, using python difflib	fuzz	ratio	a, Aa	667	67	FAILED
id5	Returns ratio of strings similarity as a percentage, using python difflib	fuzz	ratio	This is a test,	0	0	PASSED
id6	Returns the ratio of the most similar substrings as a percentage	fuzz	partial_ratio	This is a test, This is a tess	928	92	FAILED
id7	Returns the ratio of the most similar substrings as a percentage	fuzz	partial_ratio	, a	0	0	PASSED
id8	Returns the ratio of the most similar substrings as a percentage	fuzz	partial_ratio	a, b	0	0	PASSED
id9	Returns the ratio of the most similar substrings as a percentage	fuzz	partial_ratio	a, aa	100	100	PASSED