CS562 Winter 2016 Final Report

Xu Zheng

932-509-227

I. INTRODUCTION

In this project, I have tested a third party Python library called "Algorithms" by using TSTL. "Algorithms", which gained more than 1300 stars on GitHub, is a library of algorithms and data structures implemented in Python. "Algorithms" includes 8 modules: Data Structures, Dynamic Programming, Factorization, Math, Random, Searching, Shuffling and Sorting. Due to the limit of this project, I didn't test all the modules of this library. I focused on testing Data Structures and Sorting modules. The reason I chose the data structures and sorting modules is they are used most frequently. In other word, the quality of these two modules could stand for the quality of the whole library.

According to the test results, I found several bugs in the data structure module and sorting module of this library. The details will be described in the following sections.

II. TECHNICAL DETAILS

Data Structures module includes several common data structures such as Binary Search Tree, Directed Graph, Queue, Stack, Singly Linked List and so on. In this project, I test three of them: Queue, Stack and Singly Linked List. As for the details of testing all the data structures, I want to take the testing of Queue as an example. As we know, a queue is a First-In-First-Out data structure. It means that the first element added to the queue will be the first one to be removed. There are four function calls of Queue: add(value), is_empty(), remove() and size(). So that I can generate several inputs to see whether the outputs are right or not based on the understanding of queue data structure. When I called add() function or remove() function, it is important to check if the change of the size is right. After calling multiple functions calls, I compared the actual outputs with the expected outputs to evaluate the correctness. All the test cases are generated automatically by using TSTL. As for the rest of the data structure modules, I used the same strategy but with some adjustments for the other data structures.

I finished the testing of algorithms by combining all the algorithms I tested: Bogo Sort, Bubble Sort, Merge Sort, Heap Sort and Quick Sort. First of all, I wrote a python function ***is_sorted*** to check whether the input array is sorted correctly. Then I compared the actual sorting outputs with the expected sorting outputs to evaluate the correctness of sorting algorithms. And I used different sorting algorithms to deal with the same inputs since there are several sorting algorithms here. In this way, the correctness of different sorting algorithms could be tested in fewer inputs. In addition, the Bogo Sort algorithm in this library has a function bogo_sort.is_sorted() to check the correctness of the sorting results. I also tested this function by comparing the checking result from this function with my own check function. All the test cases are generated automatically by using TSTL.

III. BUGS

I have tested three data structures: queue, stack and singly linked list. I just found a bug for all the three data structures until now. The detail of the bug is as follows.

All the three data structures have the remove() function. And the bug is found in this function. I found that if the queue, stack or the linked list is empty, the system would fail the test when the remove() function be called. To be more specific, let me take the singly linked list for example. The remove() function for this data structure will return a boolean value when it is called. It means that if the return value is true, then the element has been removed successfully. Otherwise, the element need to be removed has not been found in this list. But in my test, if the linked list is empty and I called the remove() function, the system will fail the test. It means that I can't get the return boolean value which should be false. And the same problem happens in other two data stuctures. The failed test cases are as follow.

```
FINAL VERSION OF TEST, WITH LOGGED REPLAY:
int2 = 14                                                    # STEP 0
queue0 = queue.Queue()                                       # STEP 1
queue0.remove(int2)                                          # STEP 2
```

```
FINAL VERSION OF TEST, WITH LOGGED REPLAY:
int2 = 16                                                    # STEP 0
stack0 = stack.Stack()                                       # STEP 1
stack0.remove(int2)                                          # STEP 2
```

```
FINAL VERSION OF TEST, WITH LOGGED REPLAY:
list0 = singly_linked_list.SinglyLinkedList()                # STEP 0
int4 = 18                                                    # STEP 1
int2 = 6                                                     # STEP 2
list0.add(int2)                                              # STEP 3
list0.remove(int4)                                           # STEP 4
```

In my first version of the TSTL codes, I did not find any bugs. So I made some improvements for the testing codes afterwards. Then I successfully found the bug that is described above. The improvements are I added more actions in the TSTL codes. It turns out I did not found any bugs at first because of the actions are thoroughly considered. In other words, I set too many constraints for the actions. It means that the opportunities of finding the bug have been filtered. At my latest version of the TSTL code, the actions added are not so considered. So that the possibility of finding the bug has been increased.

I also find some bugs in the sorting modules. To be specific, I found that the quick sort and merge sort algorithms do not work properly. When the size of the array is two, then these two algorithms do not sort them correctly. The failed test cases are as follow.

```
FINAL VERSION OF TEST, WITH LOGGED REPLAY:
int8 = 7                                                    # STEP 0
Coverage.py warning: No data was collected.
int9 = 6                                                    # STEP 1
Coverage.py warning: No data was collected.
array0 = []                                                 # STEP 2
Coverage.py warning: No data was collected.
array0.append(int8)                                         # STEP 3
Coverage.py warning: No data was collected.
array0.append(int9)                                         # STEP 4
Coverage.py warning: No data was collected.
quick_sort.sort(array0); print(array0);                     # STEP 5
[7, 6]
FINAL VERSION OF TEST, WITH LOGGED REPLAY:
int0 = 22                                                   # STEP 0
Coverage.py warning: No data was collected.
int8 = 2                                                    # STEP 1
Coverage.py warning: No data was collected.
array0 = []                                                 # STEP 2
Coverage.py warning: No data was collected.
array0.append(int0)                                         # STEP 3
Coverage.py warning: No data was collected.
array0.append(int8)                                         # STEP 4
Coverage.py warning: No data was collected.
merge_sort.sort(array0); print(array0);                     # STEP 5
[22, 2]
```

IV. TESTER

As for me, the tester worked very efficiently. I successfully find some bugs of this system. Overall, I think my tester works very well for the SUT.

V. TSTL

In this section, I will talk about the pros and cons about TSTL. To begin with the pros, my favorite features is the supporting of python codes. It is amazing to allow user to combine the TSTL codes with python codes. This feature increases the flexibility and convenience of testing. In addition to this, the reduction of test cases and code coverage measure is also good features.

As for the cons of the TSTL, I have to say is the documentation. TSTL is not friendly to the people who first use it. The lacking of formal documentation set an obstacle for user to learn how to use it. As far as I am concerned, when I was writing the first version of the code. I spent a lot time in reading the example codes to figure out the way TSTL works. And another defect is that

VI. COVERAGE SUMMARIES

According to the testing results, I found that the code coverage for Data Structure module is relatively low. But for the Sorting module, the code coverage is impressed. The average code coverage is as follow.

Queue:

```
/Users/Xu/Desktop/CS562/codes
**> cat coverage.out
Name                                                                                    Stmts   Miss Branch BrPart  Cover   Missing
-----------------------------------------------------------------------------------------------------------------------------
/Library/Python/2.7/site-packages/algorithms/data_structures/queue.py      13     10      0      0    23%   14-20, 23, 31, 40-54
```

Stack:

```
/Users/Xu/Desktop/CS562/codes
**> cat coverage.out
Name                                                                                    Stmts   Miss Branch BrPart  Cover   Missing
-----------------------------------------------------------------------------------------------------------------------------
/Library/Python/2.7/site-packages/algorithms/data_structures/stack.py      12      9      0      0    25%   13-16, 19, 27, 36-50
```

Singly Linked List:

```
/Users/Xu/Desktop/CS562/codes
**> cat coverage.out
Name                                                                                                  Stmts   Miss Branch BrPart  Cover
  Missing
-------------------------------------------------------------------------------------------------------------------------------------
---------
/Library/Python/2.7/site-packages/algorithms/data_structures/singly_linked_list.py      45     17     10      2    65%
  15-17, 21-24, 27, 30-36, 40, 51, 72, 76, 87, 93-97, 69->72, 86->87
```

Sorting Algorithms:

```
/Users/Xu/Desktop/CS562/codes
**> cat coverage.out
Name                                                                                  Stmts   Miss Branch BrPart  Cover   Missing
-----------------------------------------------------------------------------------------------------------------------------
/Library/Python/2.7/site-packages/algorithms/sorting/bogo_sort.py        15      3      8      0    87%   17-20, 43
/Library/Python/2.7/site-packages/algorithms/sorting/bubble_sort.py       7      1      6      0    92%   17
/Library/Python/2.7/site-packages/algorithms/sorting/heap_sort.py        23      3     10      0    91%   17, 42, 53
/Library/Python/2.7/site-packages/algorithms/sorting/merge_sort.py       19      2      6      0    92%   17, 41
/Library/Python/2.7/site-packages/algorithms/sorting/quick_sort.py       10      2      6      1    81%   17, 32, 31->32
```