Final Report of Testing Project in CS562

Kun Chen
Email: chenk4@oregonstate.edu
OSU ID: 932-702-228
Date: Mar. 14, 2016

1 Introduction

In this project, a template scripting testing language (TSTL) will be utilized to test a python library of singe linked list. A linked list in is one type of data structures that has a set of nodes to form a sequence. The tested library can be gotten from Github in a personal website, and its link of source is as follows:

https://github.com/ImrulKayes/DataStructuresAlgorithms/blob/master/linkedList-1.py

Because this python library is written by an individual as a training of programming, the library can not be recognized as a standard python library. And this library is expected to exist some flaws which is quite eligible to be a good testing object by using TSTL.

The program in test system have been greatly improved since the start. There are totally 14 functions selected from the tested library for test. 4 in 14 tested functions have been verified to have bugs after test. And in order to test these functions, 8 user defined functions are written by author in TESL program. These user defined functions can count the length of a list, find values in fixed positions, etc. all tested functions as well as user defined functions will be introduced in detail in Chapter 2. In Chapter 3, 4 tested functions will be reported to have bugs in them. These 4 functions are removeElements (), deleteNode (), deleteDuplicates (),swapPairs1 (). Finally, In Chapter 4, discussion about coverage, system under test (SUT) and the quality of TSTL itself will be given.

2 Description of functions

Firstly, all the tested functions in the python library from Github will be introduced. And all these tested functions have been divided into two types after they have been tested by using TSTL. In Table 1, one type of these functions has not been found bugs yet, they are listed in the second column, and the number of this type function is 10; the other type of these functions have been found some bugs, they are in the third column, the number of this type function is 4. These 4 tested functions with bugs will be discussed in detail in Chapter 3 (Bug reports).

Table 1: Functions in single linked list library to be tested

No.	Functions which have been tested but not found bugs	Functions which have been tested and found some bugs
(1)	Insert()	removeElements()
(2)	getHead()	deleteNode()
(3)	getTail()	deleteDuplicates()
(4)	printList()	swapPairs1()
(5)	ReversePrint()	
(6)	reverseList()	
(7)	mergeTwoLists()	
(8)	swapPairs()	
(9)	reorderList()	
(10)	rotateRight()	

Meanwhile in Table 2, how the tested functions in Table 1 work are introduced. In addition, in order to help do the assertion of the tested functions in TSTL, 8 additional user defined functions with a symbol "#" in Table 2 are given. These functions can count the length, find values of a list or other actions. Their features are described in Table 2 as well.

Table 2: Brief description of all functions

No.	Function name	description of functions
(1)	Insert()	insert one or more sequences of data.
(2)	getHead()	go to the head position of the list
(3)	getTail()	go to the tail position of the List
(4)	printList()	print the input list from head to tail
(5)	ReversePrint()	print the input list from tail to head
(6)	reverseList()	reverse the order of the list from tail to head
(7)	mergeTwoLists()	merge two lists to be one list
(8)	swapPairs()	swap every two adjacent nodes in a list by recursion
(9)	reorderList()	reorder a list to the given order
(10)	rotateRight()	rotate the list to the right by given places
(11)	removeElements()	remove a node with the given value from a list
(12)	deleteNode()	delete a node in a list
(13)	deleteDuplicates()	delete all duplicates so that every value appears once
(14)	swapPairs1()	swap every two adjacent nodes in a list
#1	countLength()	count the length of list, input is an object.
#2	lengthHead()	count length of list, input is the head of an object
#3	findFstValue()	get the value in the head position of linked list

#4	findSecValue()	get the value in the second position of linked list
#5	findTrdValue()	get the value in the third position of linked list
#6	findLstValue()	get the value in the tail position of linked list
#7	findFstDuplicates()	find the first coming continued duplicate number
#8	nprintList()	print the whole list, input is an object

3 Bug reports

3.1 removeElements function

The function of removeElements(self, head, val) is to remove nodes from the singe linked list which have a given val. Fig.1 shows the action for testing removeElements () in TSTL program. In this action, we can use findLstValue () to get value in tail of a list with length 4, and then delete the nodes with the same values as the value in tail.

```
# ###### There are bugs in (function removeElements)
# test (removeElements function): we need use (findLstValue function ) to get the value
# in the tail of list and delete this value
# Description of (removeElements function): remove given value from the list
<LINKK>.insert(<INT>)
countLength(~<LINKK>) == 4 -> print "--------------------"; \
~<LINKK,1>.printList(<LINKK,1>.getHead());\
~<LINKK,1>.removeElements(~<LINKK,1>.getHead(),findLstValue(~<LINKK,1>)); \
~<LINKK,1>.printList(<LINKK,1>.getHead()); print "-       -     -     - "; => \
 (countLength(~<LINKK,1>) < 4)
```

Fig.1 Action for testing removeElements function

Fig.2 givens bugs figures after removeElements () has gone through the random test. The lists before and after using removeElements () are both printed. From Fig.2 we can observe after using removeElements (), the value 2 can not be removed from the original list so the list after removing nodes still keeps the same length.

```
REDUCED IN 0.2863240242 SECONDS
STEP 0 INT1 =2
STEP 1 INT0 =2
STEP 2 LINKK0=linkedlist.linkedList()
STEP 3 LINKK0.insert(INT1)
STEP 4 LINKK0.insert(INT1)
STEP 5 LINKK0.insert(INT0)
STEP 6 LINKK0.insert(INT1)
STEP 7 print "--------------------"; LINKK0.printList(LINKK0.getHead());LINKK0.removeElements(LINKK0.get
Head(),findLstValue(LINKK0)); LINKK0.printList(LINKK0.getHead()); print "-       -     -     - ";

FINAL VERSION OF TEST, WITH LOGGED REPLAY:
INT1 =2    # STEP 0
INT0 =2    # STEP 1
LINKK0=linkedlist.linkedList()    # STEP 2
LINKK0.insert(INT1)     # STEP 3
LINKK0.insert(INT1)     # STEP 4
LINKK0.insert(INT0)     # STEP 5
LINKK0.insert(INT1)     # STEP 6
print "--------------------"; LINKK0.printList(LINKK0.getHead());LINKK0.removeElements(LINKK0.getHead(),
findLstValue(LINKK0)); LINKK0.printList(LINKK0.getHead()); print "-       -     -     - ";    # STEP 7
--------------------
{{{{{{{{{{{{{{{{{{{{{{{{{{{
2
2
2
2
}}}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{{{
2
2
2
2
}}}}}}}}}}}}}}}}}}}}}}}}}}}
-       -     -     -
ERROR: (<type 'exceptions.AssertionError'>, AssertionError(), <traceback object at 0x102b6e290>)
TRACEBACK:
  File "/Users/chenkun/tstl/generators/sut.py", line 3165, in safely
    act[2]()
  File "/Users/chenkun/tstl/generators/sut.py", line 2036, in act84
    assert  (countLength(self.p_LINKK[0]) < 4)
STOPPING TESTING DUE TO FAILED TEST
14.0211640212 PERCENT COVERED
0.482296943665 TOTAL RUNTIME
6 EXECUTED
552 TOTAL TEST OPERATIONS
0.137440443039 TIME SPENT EXECUTING TEST OPERATIONS
0.0123064517975 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.000699043273926 TIME SPENT CHECKING PROPERTIES
0.138139486313 TOTAL TIME SPENT RUNNING SUT
0.00443911552429 TIME SPENT RESTARTING
0.286463975906 TIME SPENT REDUCING TEST CASES
51 BRANCHES COVERED
38 STATEMENTS COVERED
```

Fig.2 Bug of removeElements function

3.2 deleteNode function

The function of deleteNode (self, node) is to remove a node from given place from the singe linked list. This function works well when removes a node from the places except the tail position, such as the actions in the head position. But when we want to remove the node in the tail of a list, the function cannot work, the action in TSTL are shown in Fig.3, the its bug is reported in Fig.4.

```
# ###### There are bugs in (deleteNode function )
# # Test (deleteNode function)
# Description of (deleteNodefunction): remove given a node from the list
<LINKK>.insert(<INT>)
~<LINKK> == ~<LINKK>
countLength(~<LINKK>) == 4 -> ~<LINKK>.deleteNode((~<LINKK,1>).getTail());
```

Fig.3 Action for testing deleteNode function

```
Reduced test has 9 steps
REDUCED IN 0.331246137619 SECONDS
STEP 0 INT1 =8
STEP 1 INT0 =3
STEP 2 INT2 =2
STEP 3 LINKK2=linkedlist.linkedList()
STEP 4 LINKK2.insert(INT1)
STEP 5 LINKK2.insert(INT2)
STEP 6 LINKK2.insert(INT0)
STEP 7 LINKK2.insert(INT2)
STEP 8 LINKK2.deleteNode((LINKK2).getTail());

FINAL VERSION OF TEST, WITH LOGGED REPLAY:
INT1 =8     # STEP 0
INT0 =3     # STEP 1
INT2 =2     # STEP 2
LINKK2=linkedlist.linkedList()    # STEP 3
LINKK2.insert(INT1)     # STEP 4
LINKK2.insert(INT2)     # STEP 5
LINKK2.insert(INT0)     # STEP 6
LINKK2.insert(INT2)     # STEP 7
LINKK2.deleteNode((LINKK2).getTail());    # STEP 8
ERROR: (<type 'exceptions.AttributeError'>, AttributeError("'NoneType' object has no attribute 'val'",), <traceback objec
t at 0x10221cb48>)
TRACEBACK:
  File "/Users/chenkun/tstl/generators/sut.py", line 3433, in safely
    act[2]()
  File "/Users/chenkun/tstl/generators/sut.py", line 2311, in act96
    self.p_LINKK[2].deleteNode((self.p_LINKK[2]).getTail());
  File "/Users/chenkun/tstl/generators/linkedlist.py", line 67, in deleteNode
    node.val=node.next.val
STOPPING TESTING DUE TO FAILED TEST
6.34920634921 PERCENT COVERED
0.374979972839 TOTAL RUNTIME
1 EXECUTED
18 TOTAL TEST OPERATIONS
0.0055558681488 TIME SPENT EXECUTING TEST OPERATIONS
0.000560283660889 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
2.62260437012e-05 TIME SPENT CHECKING PROPERTIES
0.0055820941925 TOTAL TIME SPENT RUNNING SUT
0.00133299827576 TIME SPENT RESTARTING
0.331456184387 TIME SPENT REDUCING TEST CASES
25 BRANCHES COVERED
17 STATEMENTS COVERED
```

Fig.4 Bug of deleteNode function

3.3 deleteDuplicates function

The function of deleteDuplicates (self, head) is to remove all duplicate nodes with same values from a list and let every value appear only once. For testing the deleteDuplicates function, we create a user defined function called findFstDuplicates (), this function can find the first coming continued duplicate numbers, if there are no duplicates, this function will return to False. Fig.5 shows the action in the testing program.

```
# There are bugs in (deleteDuplicates function)
# Test deleteDuplicates ()
# Description of (findFstDuplicates function): find the first coming duplicate number
<LINKK>.insert(<INT>)
(countLength(~<LINKK>) == 5) and (findFstDuplicates(~<LINKK>) !=False)  -> \
print "--$--$---$---$----$----"; \
nprintList(~<LINKK,1>); \
~<LINKK,1>.deleteDuplicates(~<LINKK,1>.getHead());\
nprintList(~<LINKK,1>); print "--$    ---$     ---$    ---$" => \
findFstDuplicates(~<LINKK,1>) ==False
```

Fig.5 Action for testing deleteDuplicates function

Fig.6 givens bug figure after deleteDuplicates () have gone through the random test. The list before and after using deleteDuplicates () are both printed. From Fig.6 we can observe after using deleteDuplicates (), the last two values 3 can have been delete duplicates successfully, but for the first three duplicates value 2, they can not been deleted with success.

```
Reduced test has 15 steps
REDUCED IN 0.702847003937 SECONDS
STEP 0 LINKK0=linkedlist.linkedList()
STEP 1 LINKK2=linkedlist.linkedList()
STEP 2 INT2 =2
STEP 3 INT1 =3
STEP 4 LINKK2.insert(INT1)
STEP 5 LINKK0.insert(INT2)
STEP 6 INT1 =2
STEP 7 LINKK0.insert(INT1)
STEP 8 INT0 =3
STEP 9 LINKK0.insert(INT2)
STEP 10 LINKK0.insert(INT0)
STEP 11 INT1 =3
STEP 12 LINKK2.insert(INT1)
STEP 13 LINKK0.insert(INT1)
STEP 14 print "--$--$---$---$----$----"; nprintList(LINKK0); LINKK0.deleteDuplicates(LINKK0.getH
ead());nprintList(LINKK0); print "--$    ---$     ---$    ---$"

FINAL VERSION OF TEST, WITH LOGGED REPLAY:
LINKK0=linkedlist.linkedList()    # STEP 0
LINKK2=linkedlist.linkedList()    # STEP 1
INT2 =2     # STEP 2
INT1 =3     # STEP 3
LINKK2.insert(INT1)     # STEP 4
LINKK0.insert(INT2)     # STEP 5
INT1 =2     # STEP 6
LINKK0.insert(INT1)     # STEP 7
INT0 =3     # STEP 8
LINKK0.insert(INT2)     # STEP 9
LINKK0.insert(INT0)     # STEP 10
INT1 =3     # STEP 11
LINKK2.insert(INT1)     # STEP 12
LINKK0.insert(INT1)     # STEP 13
print "--$--$---$---$----$----"; nprintList(LINKK0); LINKK0.deleteDuplicates(LINKK0.getHead());n
printList(LINKK0); print "--$    ---$     ---$    ---$"    # STEP 14
--$--$---$---$----$----


{{{{{{{{{{{{{{{{{{{{{{{{{{{
2
2
2
3
3
}}}}}}}}}}}}}}}}}}}}}}}}}}}
{{{{{{{{{{{{{{{{{{{{{{{{{{{
2
2
2
3
}}}}}}}}}}}}}}}}}}}}}}}}}}}
--$    ---$     ---$    ---$
ERROR: (<type 'exceptions.AssertionError'>, AssertionError(), <traceback object at 0x102408a70>)
TRACEBACK:
  File "/Users/chenkun/tstl/generators/sut.py", line 2577, in safely
    act[2]()
  File "/Users/chenkun/tstl/generators/sut.py", line 1548, in act63
    assert findFstDuplicates(self.p_LINKK[0]) ==False
STOPPING TESTING DUE TO FAILED TEST
7.40740740741 PERCENT COVERED
0.810941934586 TOTAL RUNTIME
3 EXECUTED
292 TOTAL TEST OPERATIONS
0.0571985244751 TIME SPENT EXECUTING TEST OPERATIONS
0.00626492500305 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.000292778015137 TIME SPENT CHECKING PROPERTIES
0.0574913024902 TOTAL TIME SPENT RUNNING SUT
0.00422215461731 TIME SPENT RESTARTING
0.703090190887 TIME SPENT REDUCING TEST CASES
29 BRANCHES COVERED
20 STATEMENTS COVERED
```

Fig.6 Bug of deleteDuplicates function

3.4 swapPairs1 function

The function of swapPairs1 is to swap every two adjacent nodes in a list. And another tested function called swapPairs () is very similar to swapPairs1(), but in swapPairs () recursion is used. The action for testing swapPairs1 () is shown in Fig.5, and in oder to test this function, we restrict the length of list to be 5 then we assume if this tested function can work successfully, the values in the first and third position before test should be moved to the second and the fourth position after using swapPairs1 function. The same procedure will be carried out to test another swapPairs function too.

```
#There are bugs in swapPairs1
#Test swapPairs1
#Desciption of swapPairs1 (): swap every two adjacent nodes in a list
<LINKK>.insert(<INT>)
(countLength(~<LINKK>) == 5) ->  a= findFstValue(~<LINKK,1>);\
b=findTrdValue(~<LINKK,1>);\
c=~<LINKK,1>.swapPairs1(~<LINKK,1>.getHead()); => \
  (a == c.next.val) and (b==c.next.next.next.val)
```

Fig.7 Action for testing swapPairs1 function

Fig.8 shows the bug of swapPairs1 function in random test.

```
Reduced test has 10 steps
REDUCED IN 0.257967948914 SECONDS
STEP 0 INT0 =7
STEP 1 INT1 =2
STEP 2 INT2 =6
STEP 3 LINKK1=linkedlist.linkedList()
STEP 4 LINKK1.insert(INT2)
STEP 5 LINKK1.insert(INT0)
STEP 6 LINKK1.insert(INT1)
STEP 7 LINKK1.insert(INT1)
STEP 8 LINKK1.insert(INT2)
STEP 9  a= findFstValue(LINKK1);b=findTrdValue(LINKK1);c=LINKK1.swapPairs1(LINKK1.getHead());

FINAL VERSION OF TEST, WITH LOGGED REPLAY:
INT0 =7     # STEP 0
INT1 =2     # STEP 1
INT2 =6     # STEP 2
LINKK1=linkedlist.linkedList()    # STEP 3
LINKK1.insert(INT2)     # STEP 4
LINKK1.insert(INT0)     # STEP 5
LINKK1.insert(INT1)     # STEP 6
LINKK1.insert(INT1)     # STEP 7
LINKK1.insert(INT2)     # STEP 8
 a= findFstValue(LINKK1);b=findTrdValue(LINKK1);c=LINKK1.swapPairs1(LINKK1.getHead());    # STEP 9
ERROR: (<type 'exceptions.AssertionError'>, AssertionError(), <traceback object at 0x101856050>)
TRACEBACK:
  File "/Users/chenkun/tstl/generators/sut.py", line 6681, in safely
    act[2]()
  File "/Users/chenkun/tstl/generators/sut.py", line 4356, in act181
    assert   (a == c.next.val) and (b==c.next.next.next.val)
STOPPING TESTING DUE TO FAILED TEST
16.9312169312 PERCENT COVERED
0.312561988831 TOTAL RUNTIME
1 EXECUTED
71 TOTAL TEST OPERATIONS
0.0184693336487 TIME SPENT EXECUTING TEST OPERATIONS
0.00202870368958 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
7.10487365723e-05 TIME SPENT CHECKING PROPERTIES
0.0185403823853 TOTAL TIME SPENT RUNNING SUT
0.00141310691833 TIME SPENT RESTARTING
0.258177995682 TIME SPENT REDUCING TEST CASES
58 BRANCHES COVERED
47 STATEMENTS COVERED
```

Fig.8 Bug of swapPairs1 function

4. Discussions

4.1 Coverage

My present work had tested 14 functions in the single linked list in the python library. As mentioned in Chapter 3, 4 tested functions have bugs. So in order to get the maximum coverage, in the testing parts of these functions, the assertion parts are temperately closed.

Then under this kind of conditions, the coverage is about 53%, as shown in Fig.9. This coverage have realized the aim of reaching 30% coverage in final report which was proposed in another submitted report (Part 3).

```
52.6455026455 PERCENT COVERED
3.03254294395 TOTAL RUNTIME
100 EXECUTED
10000 TOTAL TEST OPERATIONS
2.54681801796 TIME SPENT EXECUTING TEST OPERATIONS
0.321191310883 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.0117135047913 TIME SPENT CHECKING PROPERTIES
2.55853152275 TOTAL TIME SPENT RUNNING SUT
0.0516123771667 TIME SPENT RESTARTING
0.0 TIME SPENT REDUCING TEST CASES
175 BRANCHES COVERED
142 STATEMENTS COVERED
10-248-116-4:generators chenkun$ cat coverage.out
Name            Stmts   Miss Branch BrPart  Cover   Missing
-----------------------------------------------------------
linkedlist.py     256    114    122     21    53%   7-8, 12-13, 15, 26, 28-34, 37, 43, 46, 48-49, 63, 70, 82, 86, 103, 11
9-148, 155, 157-160, 172-174, 178-179, 183-186, 195, 199, 215-217, 221, 232-268, 272, 274, 300, 305, 316, 327, 331-351, 3
6->37, 45->46, 47->48, 66->->63, 71->->70, 85->86, 154->155, 156->157, 167->172, 177->178, 182->183, 198->199, 202->->195, 2
06->211, 211->217, 212->215, 271->272, 273->274, 304->305, 315->316, 326->327
```

Fig.9 Coverage display in TSTL

4.2 Testing Library and TSTL

(1) Testing Library

The quality of the system under test (SUT) is not quite good, as 4/14 of tested functions in the linked list have bugs. Some bugs in functions have some similarities, e.g. both deleteNode () and deleteDuplicates () cannot well deal with the value in the first node.

(2) TSTL

. good point of TSTL

1. TSTL can provide a random test for functions written in python. It has high efficient to find bugs in tested functions. And the functions in TSTL is very similar to the ones in python, so it is very helpful for users who know python.

2. TSTL can given a clear coverage description for the testing library, so it can help the tester to know which part of the library has been successful tested and which part has not.

. Bad point of TSTL

1. There are some small bugs of TSTL, e.g., if we define the randomized number in one pool very huge, such as <[1..20000]>, the TSTL window will work very slowly. And another if one of tested functions has bugs, TSTL will stop and report the bug of this function, and it can not continue to test other functions.

2. There is lack of tutorial for TSTL. So it is not easy to have a completely understanding of the semantics in TSTL. And another time we want to test a library, we may need to write user define functions in the program, this is not quite convenient and user define functions may also exist bugs. So I hope TSTL can provide standard functions instead of user define functions.