CS 562 -- Winter Quarter 2016

Applied Software Engineering

Final Report

Professor: Alex Groce

Name: Kai Shi

ID: 932-504-512

March 2016

In this term, I learned how to use TSTL to test the python file. And I choose a python file related to Linklist as my SUT. At the end of project, I finished test all the functions of my python file. Two bugs of SUT are found. Also, I found a small bug of TSTL.

1. Test Function Methods

create_by_head: **1.** length(LINKED) should equal to the length of list inserted.

2. If it actually insert a list by head, the head of the <LINKED> should not equal to the first element of <LIST>.

First bug would appear in this situation, which will be discussed in the Bug Section.

create_by_tail: **1.** length(LINKED) should equal to the length of list inserted.

2. If it actually insert a list by head, the head of the <LINKED> should equal to the first element of <LIST>.

clear: It's simply to judge length of LINKED whether equals to 0.

is_empty: **1.** The result after executing is_empty function is a.

2. I defined a function called checklength (self, b) in <@ @>. When executing this function, b is the result of is_empty, a. If b is true, which means is_empty succeed. When it succeed, the length of LINKED should be zero. Also, if b is false, which means is_empty returns false, and in this situation length of LINKED should greater than 0. If it obeys these two situations, checklength will return True, otherwise is False.

3. checklength(<LINKED,1>, a) == True, it means it can work correctly.

getData_by_index: **1.** Set a random index by length of <LIST> -2, which must can be successful. Also I write a checkgetbyindex(b,d) function to test the value getData_by_index(c) and LIST[c] whether equal.

2. Set a random index by length of <LIST> +2, which must cannot be successful. Check if it returns False.

getData_by_value: **1.** Check whether value 1 first appears at index 0.

2. Check whether number larger than random <INT> would return False.

insertData: **1.** Set f be the result of insertData. g be the result of getData_by_index(<INT>).

2. Write the ckeckinsert function, if g is False, and f is False, it returns True. If g = LIST[<INT>], and f is True, it returns True too. Otherwise False.

deleteData_by_index: **1.** Set e be the result of getData_by_index(<INT>), f be result of deleteData_by_index(<INT>). Then g is result of executing getData_by_index(<INT>) again.

2. Write the checkdelidx(f, g, e) function, if g, f, and e are False, or f is True and e equals to g, returning True. Otherwise False.

deleteData_by_value: **1.** Set e be deleteData_by_value(<INT>), f be getData_by_value(<INT>).

2. Using written checkdelval (e, f) function to test if it succeeds.

The second bug would appear in this function, which will be discussed in the Bug Section.

delete_repeat: Using written function checkrepeat(<LINKED>, <INT>) to check if there is still same value elements in the LINKED.

delete_one: After deleting_one a random element, getData_by_this value would be False.

merge_linklist:
1. First sort two list <LISTA> and <LISTB>. Then create two LINKED, LINKEDA and LINKEDB with create_by_tail. Set e be result of linklist.merge_linklist(<LINKEDA>,<LINKEDB>).
2. Using checkmerge(e) function to test it. checkmerge(e) function checks whether the current element is larger than previous element.

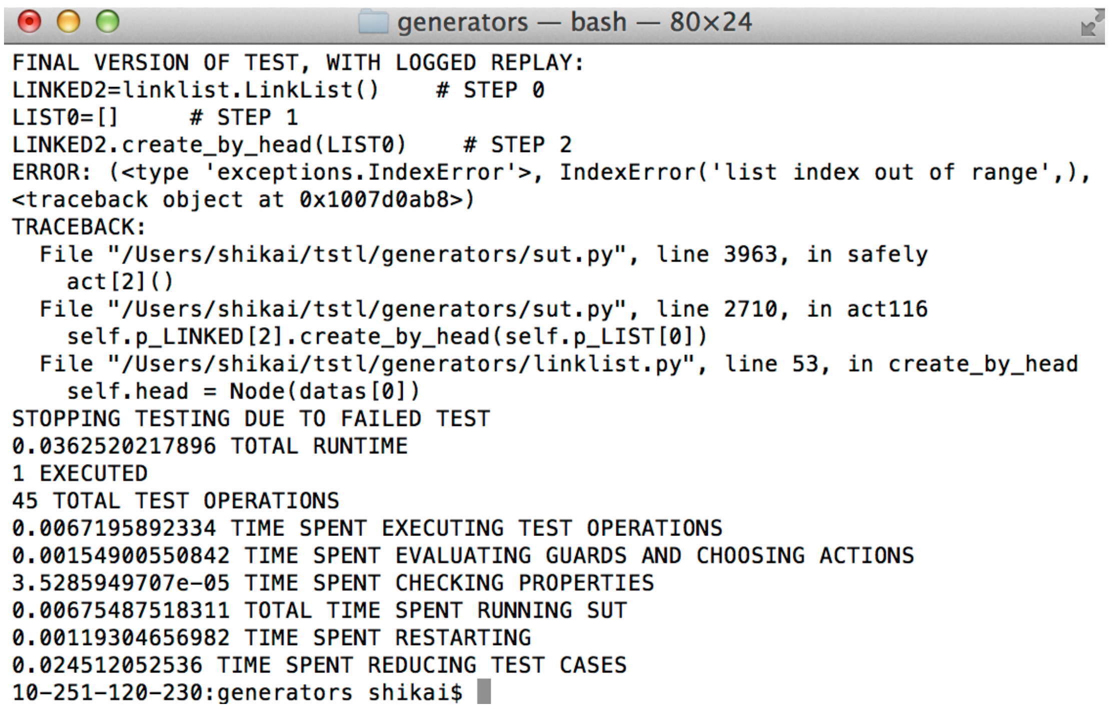
2. Bug Report Section

Through all the testing process, I found two bugs of my SUT python file. First bug is in the create_by_head and create_by_tail function and I called it "Node(data[0])" Bug. Second bug is in the deleteData_by_value function and I called it "No clear" Bug.

2.1. "Node(data[0])" Bug

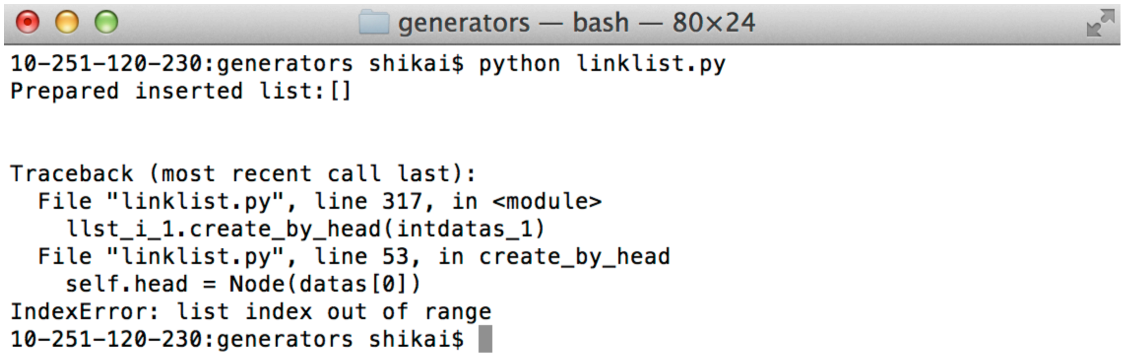
In the test function, LINKED is the target Linklist, LIST is the list we want to insert. First we initial an empty list and randomly append <INT> in the list. Then length of LINKED should be equal to the length of LIST we get because the LINKED is empty at the beginning. So, if two length equal, it means we insert successfully.

When I test this function, I initialize the empty <LIST>:[ ], then, appending <INT> in the LIST. When executing create_by_head function, it would report bug. I called it "Node(data[0])" bug.



```
FINAL VERSION OF TEST, WITH LOGGED REPLAY:
LINKED2=linklist.LinkList()    # STEP 0
LIST0=[]     # STEP 1
LINKED2.create_by_head(LIST0)    # STEP 2
ERROR: (<type 'exceptions.IndexError'>, IndexError('list index out of range',),
<traceback object at 0x1007d0ab8>)
TRACEBACK:
  File "/Users/shikai/tstl/generators/sut.py", line 3963, in safely
    act[2]()
  File "/Users/shikai/tstl/generators/sut.py", line 2710, in act116
    self.p_LINKED[2].create_by_head(self.p_LIST[0])
  File "/Users/shikai/tstl/generators/linklist.py", line 53, in create_by_head
    self.head = Node(datas[0])
STOPPING TESTING DUE TO FAILED TEST
0.0362520217896 TOTAL RUNTIME
1 EXECUTED
45 TOTAL TEST OPERATIONS
0.0067195892334 TIME SPENT EXECUTING TEST OPERATIONS
0.00154900550842 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
3.5285949707e-05 TIME SPENT CHECKING PROPERTIES
0.00675487518311 TOTAL TIME SPENT RUNNING SUT
0.00119304656982 TIME SPENT RESTARTING
0.024512052536 TIME SPENT REDUCING TEST CASES
10-251-120-230:generators shikai$
```

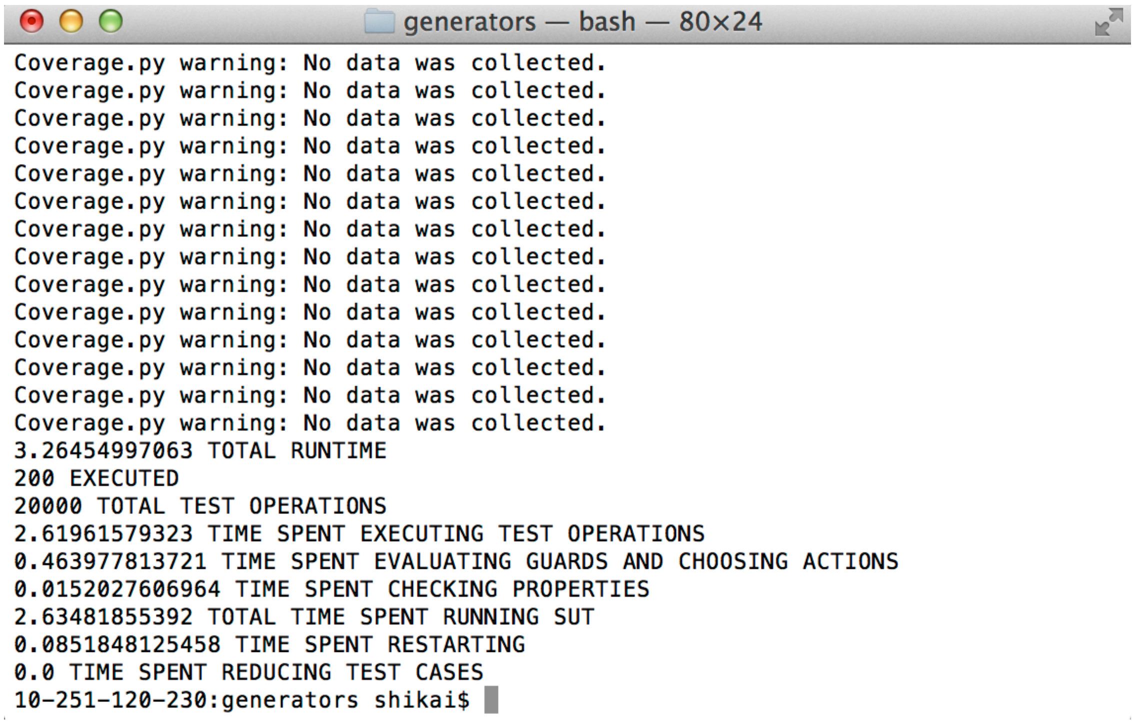
It says when we insert an empty list in the Linklist, it would crashed due to the Node(datas[0]). So, I write a print function in python file to print the result of inserting empty list. It also crashed and reported bugs like this:



```
10-251-120-230:generators shikai$ python linklist.py
Prepared inserted list:[]


Traceback (most recent call last):
  File "linklist.py", line 317, in <module>
    llst_i_1.create_by_head(intdatas_1)
  File "linklist.py", line 53, in create_by_head
    self.head = Node(datas[0])
IndexError: list index out of range
10-251-120-230:generators shikai$
```

Therefore, it is the bug of this python file. Then, I analysis the reason it report this bug. I see in the create_by_head(self, datas) function, the author first define self.head = Node(data[0]), but he does not consider when datas is empty. When datas is empty, data[0] would be empty too, and head would also be empty. This function would failed. That's the reason why it cannot insert empty list to the Linklist.

So, I changed <LIST>: = [] to <LIST>: = [1,2]. It means I initialize the list as [1,2] rather than empty. It can insert list successfully.



```
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.
3.26454997063 TOTAL RUNTIME
200 EXECUTED
20000 TOTAL TEST OPERATIONS
2.61961579323 TIME SPENT EXECUTING TEST OPERATIONS
0.463977813721 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.0152027606964 TIME SPENT CHECKING PROPERTIES
2.63481855392 TOTAL TIME SPENT RUNNING SUT
0.0851848125458 TIME SPENT RESTARTING
0.0 TIME SPENT REDUCING TEST CASES
10-251-120-230:generators shikai$ 
```

That's the first bug I found in this project. Also, the same bug would appear in the create_by_tail(self, datas) function.

The other bug of the python file is the clear() function. It also caused by "Node(data[0])" bug. I thought after executing clear() function, the length of LINKED should be 0. So, I write this: ~<LINKED>.clear() => (length(<LINKED,1>) == 0). But it would report bug, which says length(<LINKED,1>) not equals 0. So, I thought in this file, when it is init, it defined self.head = Node(0). So, its length is at least 1 at the beginning. So, I should write (length(<LINKED,1>) == 1). It can be executed. But in fact, when we clear a Linklist, the length should be 0. If this is the black box test, the function would be considered as failed. But in this project, I have read his codes, so I consider it successfully clear the whole Linklist.

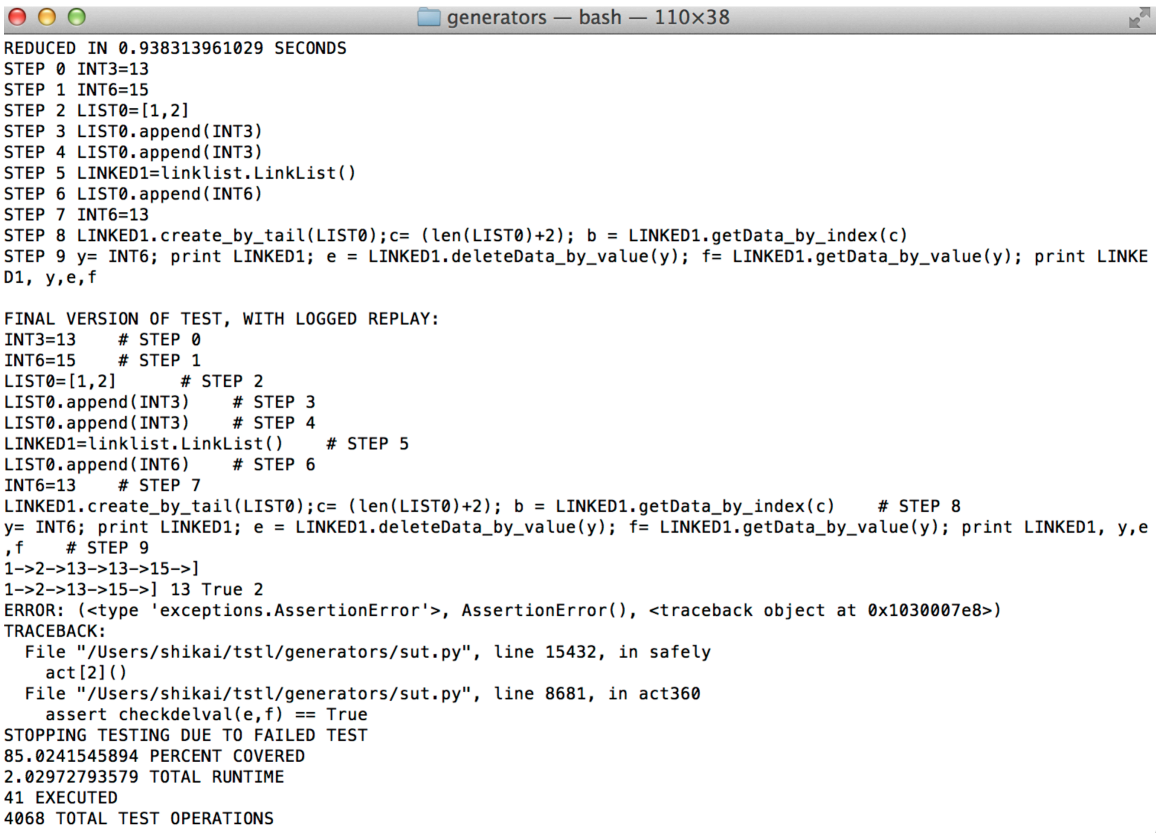
2.2. "No clear" Bug

When testing deleteData_by_value function, I set e = deleteData_by_value(<INT>), f = getData_by_value(<INT>). Then I use checkdelval (e, f) function to test if it succeeds.

```
<@

def checkdelval(e,f):

    if e == False and f==False:

        return True

    elif e == True and f==False:

        return True

    else:

        return False

@>
```

This function means if deleteData_by_value succeeds, we will never find the same value in the LINKED again. So f should be False.

Then, TSTL report the following bug:



```
REDUCED IN 0.938313961029 SECONDS
STEP 0 INT3=13
STEP 1 INT6=15
STEP 2 LIST0=[1,2]
STEP 3 LIST0.append(INT3)
STEP 4 LIST0.append(INT3)
STEP 5 LINKED1=linklist.LinkList()
STEP 6 LIST0.append(INT6)
STEP 7 INT6=13
STEP 8 LINKED1.create_by_tail(LIST0);c= (len(LIST0)+2); b = LINKED1.getData_by_index(c)
STEP 9 y= INT6; print LINKED1; e = LINKED1.deleteData_by_value(y); f= LINKED1.getData_by_value(y); print LINKE
D1, y,e,f

FINAL VERSION OF TEST, WITH LOGGED REPLAY:
INT3=13    # STEP 0
INT6=15    # STEP 1
LIST0=[1,2]      # STEP 2
LIST0.append(INT3)    # STEP 3
LIST0.append(INT3)    # STEP 4
LINKED1=linklist.LinkList()    # STEP 5
LIST0.append(INT6)    # STEP 6
INT6=13    # STEP 7
LINKED1.create_by_tail(LIST0);c= (len(LIST0)+2); b = LINKED1.getData_by_index(c)    # STEP 8
y= INT6; print LINKED1; e = LINKED1.deleteData_by_value(y); f= LINKED1.getData_by_value(y); print LINKED1, y,e
,f    # STEP 9
1->2->13->13->15->]
1->2->13->15->] 13 True 2
ERROR: (<type 'exceptions.AssertionError'>, AssertionError(), <traceback object at 0x1030007e8>)
TRACEBACK:
  File "/Users/shikai/tstl/generators/sut.py", line 15432, in safely
    act[2]()
  File "/Users/shikai/tstl/generators/sut.py", line 8681, in act360
    assert checkdelval(e,f) == True
STOPPING TESTING DUE TO FAILED TEST
85.0241545894 PERCENT COVERED
2.02972793579 TOTAL RUNTIME
41 EXECUTED
4068 TOTAL TEST OPERATIONS
```

Analyzing why it would report this bug, we can find that the bug appears when LINKED is 1-> 2-> 13-> 13->15, and we want to delete 13. But the result after deleting 13, LINKED becomes 1-> 2-> 13-> 15. There is still a 13 in the LINKED. Therefore, e is True, but f returns 13's index 2. So, checkdelval (e, f) returns False because True != 2.

That means it could only delete the first element, which value equals to the value need to be deleted. But in his description, this function should delete all elements, which value equals to the value need to be deleted.

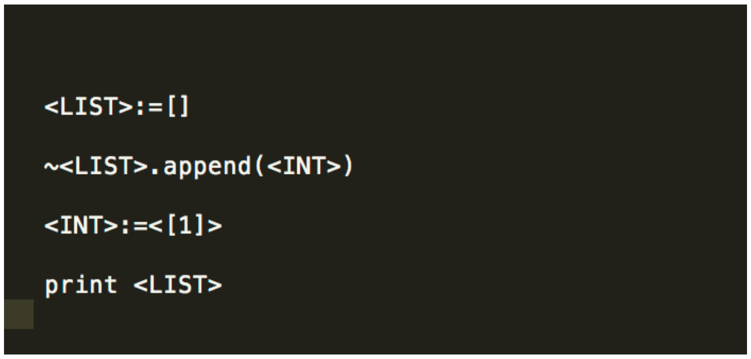
3. Quality of the SUT

The quality of the SUT is between poor quality and well done. Because his initialize function is not good. He didn't consider when list need to be inserted is empty, what should codes do. However, it is not a big problem because this function can still insert non-empty list into the Linklist. The second bug appears in deleteData_by_value. It cannot delete all the value in the Linklist. And other functions like getData_by_index, he writes well. "Node(data[0])" and "No clear" bugs are the only two bugs among 12 functions. It's a not very high percentage bugs but it's not very low neither. Therefore, the quality of the SUT is between poor quality and well done. Tester work is not so good, but it's not so bad too.

4. Wrong with TSTL

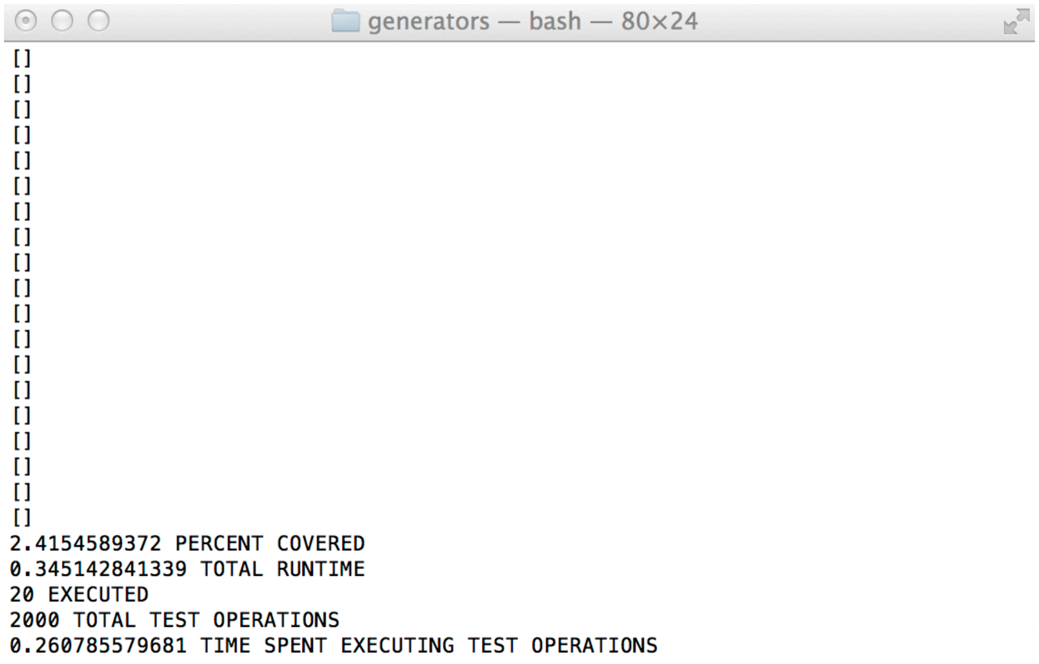
During this project, I found a small bug with tstl. At the beginning, I want to generate a random list <[1,2]> to test creat_by_head, it can generate correctly. But when I only need to generate only one value list, such as <[1]>, it would be empty and generate nothing.

Here is the tstl code I execute:



```
<LIST>:=[]

~<LIST>.append(<INT>)

<INT>:=<[1]>

print <LIST>
```

Here is the result, we expect [1,1….] …, but we only get empty lists.



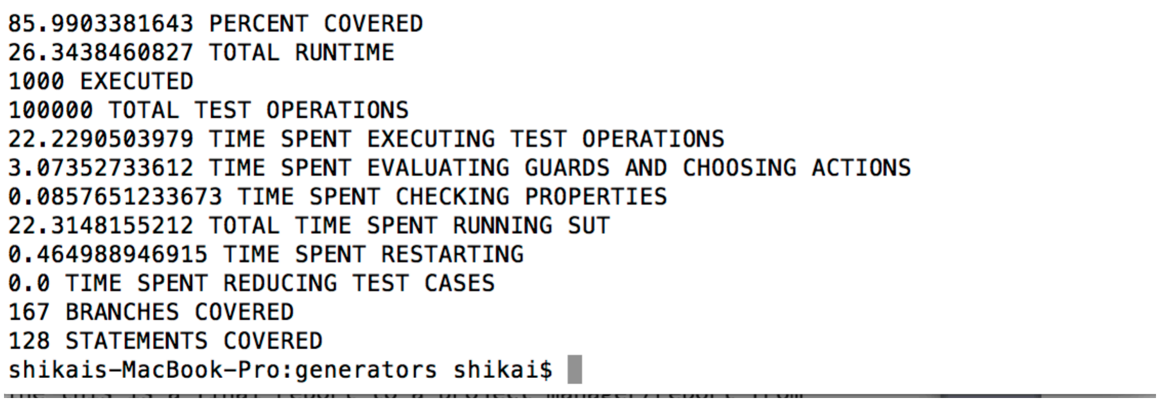
```
[]
[]
[]
[]
[]
[]
[]
[]
[]
[]
[]
[]
[]
[]
[]
[]
[]
[]
[]
2.4154589372 PERCENT COVERED
0.345142841339 TOTAL RUNTIME
20 EXECUTED
2000 TOTAL TEST OPERATIONS
0.260785579681 TIME SPENT EXECUTING TEST OPERATIONS
```

5. Good about TSTL

From software testing aspect, testing must obey three main key words: random, large amount, repeat. TSTL can provide these three conditions when testing SUT. In addition, it's easy to change the input in TSTL file. For example, if we want to change integer list to char list, we only need to change <INT> to <Char>. Also, TSTL can give tester chance to test many boundaries of input to test SUT.

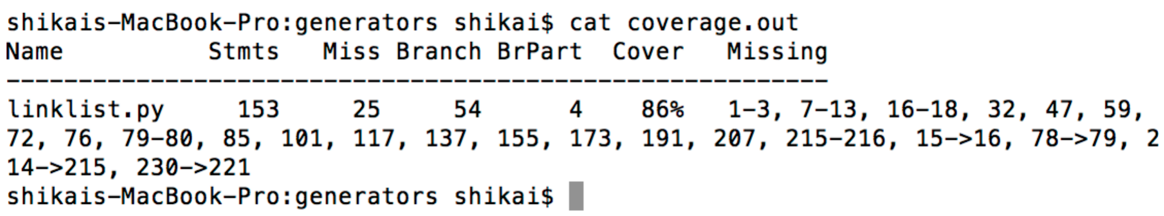
6. Coverage

After testing all functions of my SUT, my coverage reaches almost 86 percent. The result is:



```
85.9903381643 PERCENT COVERED
26.3438460827 TOTAL RUNTIME
1000 EXECUTED
100000 TOTAL TEST OPERATIONS
22.2290503979 TIME SPENT EXECUTING TEST OPERATIONS
3.07352733612 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.0857651233673 TIME SPENT CHECKING PROPERTIES
22.3148155212 TOTAL TIME SPENT RUNNING SUT
0.464988946915 TIME SPENT RESTARTING
0.0 TIME SPENT REDUCING TEST CASES
167 BRANCHES COVERED
128 STATEMENTS COVERED
shikais-MacBook-Pro:generators shikai$ 
```

And cat the coverage.out file we can look up the information of coverage:



```
shikais-MacBook-Pro:generators shikai$ cat coverage.out
Name          Stmts   Miss Branch BrPart  Cover   Missing
---------------------------------------------------------
linklist.py     153     25     54      4    86%   1-3, 7-13, 16-18, 32, 47, 59,
72, 76, 79-80, 85, 101, 117, 137, 155, 173, 191, 207, 215-216, 15->16, 78->79, 2
14->215, 230->221
shikais-MacBook-Pro:generators shikai$ 
```

It shows my tstl cover 86 per cent branches and shows the missing branches. The reason why the coverage couldn't be 100 per cent is there are many definition functions and codes in the beginning of the SUT. For example, the writer defines the Node Class at the beginning. These kinds of branches would never be tested. Therefore, the coverage could not be 100 per cent.