The Progress Report

CS562

Chao Zhang

Prof Groce Alex

This is the final report for test project. My test project is to test a library called LinkedList_Circular.py. In the beginning of the project, I thought this project is only related with the LinkedList_Circular.py file, but when I got start with it, I realized that this LinkedList_Circular.py is also connected with other two files, LinkedList_Single.py and LinkedList_Double.py.

For this term and this course, I focused on the TSTL which is the Template Scripting Testing Language, this tool is used to test the Python code. The first step I did was to learn and understand the syntax and the structure of the TSTL. The syntax and the structure is not too complicate, it uses the pool as the range and uses the act part to finish the test. After we give definition, we can call the act to generating a huge mount of random numbers and pass them to the function which we would like to test with. So to test the Python file, all I need to do is to write down the action, this action should be based on the function of the Python file. For the action, I also need a guard, this guard is a way to check the properties for the function I want to test. In the TSTL, => is an important symbol, before this symbol is the action and after this symbol is the guard. To understanding the basic syntax and structure is the foundation of the TSTL. After understanding of those, I start to working on the test.

The basic Python library structure is circular linked list, it has fifteen functions which are insert, append, returnIndex, updateIndex, deleteIndex, insertBeforeIndex, insertAfterIndex, deleteData, deletePtr, deleteAllData, insertAfterEveryData, insertBeforeEveryData, deleteList, copyList and findMthToLastNode.

I will give some intro to those functions and give the logic of my thought about how to test them.

insert: Insert a data at the head of the list, length == PRE length + 1, then get the first data after inserting

append: Append a data at the end of the list, length == PRE length + 1, then get the last data after inserting

returnIndex: Return the index of a data you first meet, insert a data at the beginning of the list, and return the index 0

updateIndex: Update the data at a specific index, update a data at a index, then get the data at the index and compare the data with what I have updated

deleteIndex: Delete the data at the a specific index 1. length + 1 == PRE length

insertBeforeIndex: Insert a data after a specific index, length == PRE length + 1, then get the data at the index and compare the data with what I have updated

insertAfterIndex: Insert a data before a specific index, length == PRE length + 1, then get the data at the index + 1 and compare the data with what I have updated

deleteData: Delete the data you first meet, length + 1 == PRE length

deletePtr: Delete the data before the current one, length – 1 ==PRE length, then get the data at the index + 1 and compare the data with what I have updated

deleteAllData: Delete all data you indicate, count the number (a) of data in the list, then length + a == PRE length

InsertAfterEveryData: Insert a data after every data you indicate, count the number (a) of data you indicate, then length == PRE length + a and the number of data you insert >= a

InsertBeforeEveryData: Insert a data Before every data you indicate, count the number (a) of data you indicate, then length == PRE length + a and the number of data you insert >= a

deleteList: Delete the list, the length of the list == 0

copyList: Copy a list, the length of the list == the length of the list be copyed, two lists should be equal

findMthToLastNode: Find a nth data from the last, the index of the data == the length of the list – n

With all those functions I tested, I just found one bug, it is in the insertBeforeIndex function. The definition the author gives is to add the new data before the assigned position. So for this function, let me explain this bug, assume that the list have 3 data in it, 1,2,3, their

position is 0,1and 2. When I call this function like insertBeforeIndex(2,4), this list will be changed to 1,2,3,4. This is correct, but if I call this function like insertBeforeIndex(3,4), the list is still like 1,2,3,4, but as the definition this list has no position 3, the position is end with 2. This is the only bug I found, it is also appears in the insertAfterIndex and findMthToLastNode functions. This function probably is not a real bug but a logical problem. So it can be fixed easily.

The TSTL is a good tool to test the Python file, it can generate thousands of data randomly as I designed, then it will pass them to those functions I want to test and helped to find out the problems. It is simple to understand and easy to use. But for use the TSTL, we need to know something about the Python and we also need to spend some times on the syntax and the structure for the TSTL. The only problem I found with the TSTL is the TSTL can not generate an list with one data inside, it can generate the list with more than one data but when I try to generate the list with one data, it with run and gives an empty list.

After the test, my TSTL covered 80% of the code like the picture shown.

```
80.02139037433 PERCENT COVERED
175.524266005 TOTAL RUNTIME
1 EXECUTED
9 TOTAL TEST OPERATIONS
164.464091539 TIME SPENT EXECUTING TEST OPERATIONS
0.00068187713623 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
6.91413879395e-06 TIME SPENT CHECKING PROPERTIES
164.464098454 TOTAL TIME SPENT RUNNING SUT
0.00107502937317 TIME SPENT RESTARTING
10.9461681843 TIME SPENT REDUCING TEST CASES
173 BRANCHES COVERED
132 STATEMENTS COVERED
```

```
Name                    Stmts   Miss Branch BrPart  Cover   Missing
-------------------------------------------------------------------
LinkedList_Circular.py     248    23    126      3     80%   8-14,46->52, 144->147, 165->167
```

This shown how many branch tested and how many branch I missed. The result of the coverage will never be reach to 100% because in the LinkedList_Circular.py, there are many definitions, those lines will be never test. So the coverage couldn't be 100% forever.