

Final Report

For the term final project, a library dealing with the operations of heap has been fully tested. The library has several functions which implement some operations such as push an element to the heap, pop the heap element out and so on. The python code of this library can be found in <https://hg.python.org/cpython/file/2.7/Lib/heapq.py>. By the end of the term, all the functions in this library are tested by using TSTL and it turns out that all the functions work well. However, there are several things may interested for TSTL developer. For this library, the coverage provides by TSTL is 0 at first. After I delete the “source:”, the coverage changes to -1 instead which means no function has been tested.

Accomplishment

By the end of the term, I covered all functions in my testing library. At the beginning, I only consider the list of “Int” type in the random tester pool. After that, I tried the list of “char” type and even the mix type (the list mixes with Int and char type). I will briefly talk about the functions in the library and how to test it.

- (1) `Heapq.heappush(heap,item)`: This function is used to push a value item onto the heap and maintains the heap invariant. Using random tester, both simple and complex heap are tested to push a random item to heap and does not change the property of the heap.
- (2) `Heapq.heappop(heap)`: This function is tested whether it pops the right item from random heap and the item should be the smallest element from the heap.
- (3) `Heapq.heappushpop(heap,item)`: This function is the combination of push and pop. The tester checks whether the heap push and pop the right element and return the true result.
- (4) `Heapq.heapreplace(heap,item)`: This function first pop the first element in the heap and push the particular element into heap. Same as the former function, we need check whether it pop and push the right element and final return the right heap.
- (5) `heapq.merge(*iterables)`: To check this function, I use the random tester to pick two random heap and merge them together. Also, the heap property is still valid.
- (6) `Heapq.blargest(n,iterable[, key])`: This function return a list with the n largest elements form the dataset defined by iterable.key, if provided. I use the heap variable to record the result so that we can check the correctness.
- (7) `heapq.nsmallest(n, iterable[, key])`: Same with the function above, this function return a list with the n smallest elemetns form the dataset defined by iterable.key. I also record the result to another heap and check the correctness.

Bug

Until now, I didn't find any bugs from my testing library. In order to make sure that there is no bugs, the first strategy is using the logging to record the former status and the latter status. Also, I printed out the results so that I can check it manually and it showed that nothing went wrong. To make it more particular, I set $m = 40$ which means there will be about 400 executes and TSTL gave positive result. Furthermore, I also consider the special case which do the replace and pop when the heap is empty. I tried in the python environment and it shows the “index error” and in the TSTL, I got the same result as it caught the expectation as “index error” then stopped

automatically. To be more confident, I checked the out.txt (the file store the final result) and figured out that the error occurred when the next step is pop or replace and the heap is empty which confirm my assumptions.

Tester's Work

Heap library is a small but useful library, it handles almost all the important operations for heap data structure. However, I strongly believe that the developers or the designers sometimes assume that the users to aware of the input instead of checking whether it is illegal or legal input. For example, the heap merge function, it depends that all the heaps need to be merged should be the sorted one, but even I input the unsorted heap, it does not give me the error alarm. After all, most of the functions are very well tested and running as required. In my opinion, this library is well developed but it will be perfect if the developers take into account end-users misuses.

Wrong with TSTL

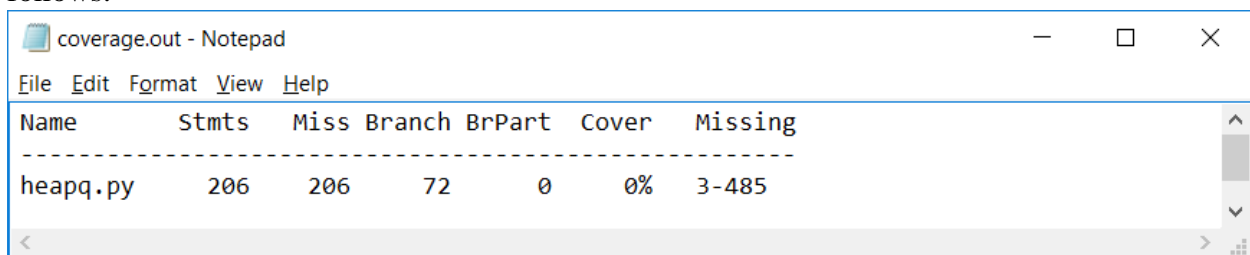
One thing I found disappointing for the TSTL is that it does not provide the reasonable coverage for my testing library. In my case, I finally got 0% for my coverage which means that my TSTL code actually not using any function from my testing library. However, I got the reasonable result for every operation which proofs something is wrong with the coverage detection of TSTL. Another drawback is that TSLT is not completed to testing a lot of libraries. For instance, one of classmates choose the library used to transforming the mass code to the normal code. However, TSTL cannot detect the mass code very well so that my classmate have to change the library to test.

Good with TSTL

The most significant contribution of TSTL is creating the test case by itself and save a lot of time for the testing employees. For original testers, they will need to design all the possible case to test and also the expect result. However, the TSLT can help them to provide infinite case to cover almost all the possible case and using the actions to detect the bugs. Furthermore, the TSTL provides the sut.py and coverage.out which gives us the idea of coverage and how well the test goes to.

Coverage Summaries

For my testing library, the TSTL keeps offering the 0% of coverage and the coverage.out is as follows.



Name	Stmts	Miss	Branch	BrPart	Cover	Missing
heapq.py	206	206	72	0	0%	3-485

I also tried to delete the “source: heapq.py”, however, TSTL gives a even more disappointing number which is -1. But, I am sure that my code already used the function from my library. The example is as follows:

```
[ LOG heapq.heapify(self.p_heap[0]);print "co", self.p_heap[0] : self.p_heap[0]] [9, 5, 5, 7, 2]
[ LOG heapq.heapify(self.p_heap[0]);print "co", self.p_heap[0] : self.p_heap[1]] [8, 7]
co [2, 5, 5, 7, 9]
Coverage.py warning: No data was collected.
[ POST LOG heapq.heapify(self.p_heap[0]);print "co", self.p_heap[0] : self.p_heap[0]] [2, 5, 5, 7, 9]
[ POST LOG heapq.heapify(self.p_heap[0]);print "co", self.p_heap[0] : self.p_heap[1]] [8, 7]
```

In this example, the initial input is [9,5,5,7,2] after using the function `heapq.heapify()`, the heap change to [2,5,5,7,9] which fits the heap property. This can illustrate that I indeed use the function, however, the percentage of coverage is 0%. After that, I tried the “coverage report” command and it shows as follows:

```
PS E:\CS 562\project> coverage report
Name                                                    Stmts   Miss  Cover
-----
sut.py                                                    16662   11812   29%
E:\CS 562\tstl-master\tstl-master\generators\randomtester.py  379     223   41%
-----
TOTAL                                                    17041   12035   29%
```

As we can figure out in the picture, it is even no show the `heapq.py` document. Since I deleted the “source: heapq.py”, the coverage changed to -1. I think it is reasonable to guess that something is wrong after TSTL detect the testing library. It cannot go through the library so that it cannot calculate the coverage.

Summary

TSTL is a quite useful and helpful application/tool for testing python library. Although it still have some bugs to fix especially the coverage detection part, it still offers the sight of the further testing tendency. In addition, from this project, I have a better understanding about the python and the logic to test a library.