Final Report

Introduction

This report intends to discuss the performance and result of my final version project. The software under test is a python library called algorithms, which contains several submodules. The submodules that I have tested on this final version of my project including datastructures, searching and sorting. The rest of submodules I have tried to test them, but some of them are hard to understand and impossible to find a way to prove correctness. In addition, the three submodules I have tested containing most part of this library. Thus these three submodules is quite enough to stand for this library.

Accomplishment

I accomplished three submodule of algorithms as I mentioned above. The first part is datastructures. I left three data structures alone in this module without testing them, which are union_find.py, union_find_by_rank.py and union_find_with_path_compression.py, because I am a little bit confused about the implementation in source codes. For others, I tested their general crud (create, read, update, delete) operations as a data structure. BST.tstl file contains the codes for testing binary search tree. I wrote several self-defined methods maxNode(), minNode() and isBST() in order to check if tree was still a binary search tree after each operations. In this data structure, I used python primitive list as a reference to see if an element is really in a tree or not in current testing status. And set a couple of pre-conditions and post-conditions to ensure that tree correctly add or delete some elements. All the conditions also tested the methods that implemented read operation.

I implemented similar strategies to test queue, stack and linked list data structures, which are set a couple of pre and post conditions for each crud operations. In digraph.tstl, I tested directed graph data structure. In the implementation of this data structure, there is a reverse method, which reverses the entire graph. In order to test this method, I had to write a self-defined function to test if the graph really reversed after call this method. In addition, when tested this data structure, I used characters as the contents of graph instead of integers. The undirected graph used similar strategies to test. No bugs have found in this module so far.

Besides datastructures submodule, I tested searching submodule as well. In this module, I just simply execute each search algorithms and print out the result of searching. it turns out that there are several problems and bugs in this submodule. I will talk about the bugs that I found in next section. But basically, I have tested different searching algorithms by using three types of input for better testing, which are integer, character and string. I used some guards for these searching algorithms. Some of them merely check if the searching element is in the list or not while others may check the return value of algorithm.

For sorting submodule, initially I want to take time complexity into account for correctness, but it is really hard to decide. I have set up a couple of time consumption test case for several popular sorting algorithms to compare their performance. But it turns out the algorithm that expected to be a little bit faster is actually sometimes slower. I asked for my classmates and found out that sometimes the big-O time complexity cannot be a criteria for the correctness of sorting algorithms since one need to take the consumption of memory allocation and other low level execution into account when using time() function. So I deleted time complexity measurement and changed my testing strategy to execute all algorithms and check if it is actual sorted.

Bugs

In this section I will talk about some bugs that I found in these submodules. It turns out that datastructures and sorting submodules are working quite well and are bug-free. But searching submodule has several critical bugs in some algorithms.

First, for binary search module, this is no specification said this algorithm need to take a sorted array as input. official document only said input is a list. By inputing unsorted array, binary search get wrong answer based on my post- conditions. Second, there is a bug found in BMH search algorithm. When the input has string with consecutive pattern appeared, python will complain with string out of index. Third, for both breadth first search and depth first search, I always get type error inside source code. If I used {TypeError} to get rid of type error, I cannot get correct result. So there are definitely some bugs inside this module.

Performance of Tester (how well does the tester work)

Overall, My Tester works quite well in some cases. For example, it eventually found some bugs in given context. But certainly, there are still a lot of unsatisfied criteria as a tester. My tester cannot give a good answer on whether a sorting algorithm is correctly implemented with its corresponding fundamental concept. To sum up, it works well for this SUT.

Pros and Cons of TSTL

TSTL is really a excellent tester suite which contains so many amazing features such as easy-to-use grammar structure, the ability to use with python raw codes together, Providing test normalization and reduction and eventually combining with coverage measure which is important to testing a software. Nevertheless, There are still several disadvantages in TSTL. The most critical one I have to say is lacking of document and tutorial. Even thought TSTL has pretty simple grammar, It is going to be way better if there is a complete document to refer. In addition, in term of testing, TSTL sometimes will do several redundant work which is pretty bad for

performance. I occasionally got stuck on some heavily repeated testing. One improvement that TSTL might need is that TSTL should provide better error prompting system, which will definitely make user's life easier. Overall, as a developing test suite, TSTL is really fantastic.

Coverage Summary

Below I will give a table to show the overall coverage of each TSTL file in my project.

File Name	Coverage Percentage	Branched Covered	Statements Covered
BST.tstl	**33.13258520366**	**101**	**70**
LinkedList.tstl	**50.9090909091**	**30**	**22**
queue.tstl	**23.0769230769**	**6**	**3**
stack.tstl	**25.0**	**6**	**3**
digraph.tstl	**75.4385964912**	**44**	**31**
undigraph.tstl	**65.2173913043**	**32**	**23**
searching.tstl	**67.7419354839**	**42**	**29**
sorting.tstl	**93.1558935361**	**215**	**155**