Report of Progress

This report intends to discuss the progress of my TSTL project for now and the next steps for completing this project. This is part three of entire project. The thing needs to be mentioned again is that the software under test of my project is a python library called Algorithms. This library has a lot of submodules. The way that I test them is to test them individually. Especially for the datastructures submodule, I broke it into multiple TSTL files for each data structure to test. For sorting and searching submodules, I test them in a single file with all different sorting and searching algorithms.

So far, 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. 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.

For sorting submodule, I am still working on it. I am trying to find a best way to test its correctness. This is going to be one of the next thing I will do. For now, I merely output the sorting result and apparently all sorting algorithms can correctly sort arbitrary list. From start to now, my testing project has huge progress. I am able to find a reasonable way to describe the correctness of data structures instead of just writing simple program to look for python report errors. And I have tested the most essential parts of this library, which contain nearly 60 percent of codes. Hopefully, I can test almost 80 percent of this library. The quality of SUT is not good enough since there still are several bugs I can find.

In term of coverage, I have to talk them one by one since I break down the module into multiple submodules to test. Following I will list the coverage for each test file as well as discuss about it (All of the coverage information is tested by setting timeout equal to 60 seconds):

BST.tstl:
33.13258520366 PERCENT COVERED
101 BRANCHES COVERED
70 STATEMENTS COVERED

Since in binary_search_tree.py, there is a lot of functions that are difficult to measure its correctness, there still are some functions remains to add to test files later. But the main part of this structure has been tested very well and get enough coverage.

LinkedList.tstl:
50.9090909091 PERCENT COVERED
30 BRANCHES COVERED
22 STATEMENTS COVERED
queue.tstl
23.0769230769 PERCENT COVERED
6 BRANCHES COVERED
3 STATEMENTS COVERED
stack.tstl
25.0 PERCENT COVERED
6 BRANCHES COVERED
3 STATEMENTS COVERED

Above three data structures have fairly simple implementation so the source code is not abundant. That why they get less branches and statements coverage than BST.tstl.

digraph.tstl
75.4385964912 PERCENT COVERED
44 BRANCHES COVERED
31 STATEMENTS COVERED
undigraph.tstl
65.2173913043 PERCENT COVERED
32 BRANCHES COVERED
23 STATEMENTS COVERED

Directed graph and undirected graph data structures are somehow a little more complex. But my test case can still cover large enough percentage, which is enough to prove the correctness of this implementation

searching.tstl
67.7419354839 PERCENT COVERED
42 BRANCHES COVERED
29 STATEMENTS COVERED

Since I found some bugs in several searching algorithms, I am not able to show the coverage of the algorithms that contain bugs. I will try to figure a way to show them in next stage. But overall the coverage is decent.

I stored all of the coverage output files into a single coverage directory under my project directory. All details can be viewed in these files that I stored.