Ryan Ngoon

Sam Jain

Lab 9

Possible Data Structures:

- Hash Table
  - The Hash table's keys would be the line numbers and each line would be hashed with the data of whether it has been covered or not. As the lines gets covered, the data in the hashtable of whether its been covered or not is flipped to true. To get the coverage percentage, you would divide the number of keys that were not covered with the total number of keys.
- Array
  - The array would be initialized with the size of the file, with each element of the array representing whether the line has been covered or not.
- Binary Search Tree
  - The tree would be initialized which each line with all the nodes containing whether it's been visited or not.

Starts at igor.py → runs tracers to watch the lines of code as they run and keep track of things → tracers: located in ctracer folder. Tracer.h TRACER USES STACK DATA STRUCTURE

There are 2 ints in tracer, one for ALL of the lines, and the other for the lines that are run

Detailed Code Examination

1. Data.py

a. This file creates a hash table summarizing the line coverage. The comments were more useful in understanding this. The data being stored is the coverage of each file with the filenames being used as keys. The data being added is the coverage data and it is being used to summarize the total coverage of all the files being tested. The code is very readable, utilizing more comments than I would use, and it feels clear and easy to update.

2. Collector.py
    a. This creates a stack of collectors which collects trace data. The comments were also more useful in understanding the code. The data being stored is the trace data. The data being added is the collector class and it is used to keep track of the tracer of the threads. Again, the code is readable and the comments are very helpful in understanding what each class and function does. This makes the code feel like it would not be difficult to update.
3. Results.py
    a. Finds the chunks of code that are and are not run, formats the ones that are not run to put in the report. An arc is a chunk of code that runs together, for lack of a better definition. The program creates a list of the arcs in the code, and then it adds the arcs that were run to a list and the arcs that were not run to a list. When measuring branches, the code counts up the number of exit points for each line of code and then looks to make sure that each exit is reached throughout the tests, the program then adds the lines to each list (run and not run) as necessary.
4. Summary.py

a. Using the lists of lines from Results.py, Summary.py formats the lines in a way that can be outputted and it calculates the percentage of lines covered. It formats the data from those lists and throws it into a file.

This code is very well documented, but it is difficult to read syntactically for someone at our level due to its use of more advanced Python techniques. The documentation allowed people at our level to be able to see what each function did and the purpose of each object. With regards to contributing to this code, I would like to, at some point, contribute to this code, once I gain more experience with Python. My own code is not nearly as complex as the code here and uses only one language for the most part. This code uses c (.h and .c files) and python (.py files) together which is not something we have used before.

https://coverage.readthedocs.io/en/latest/howitworks.html