2.1

Data Structures:

- A hash table could be used to keep track of the line numbers that have been covered. We would go through the code, recording which lines have been covered and add them and the file name to the hash table. The elements will be sorted based on the file name and they will be compared to the other lines within that file.
- A graph could be used to connect all the lines through functions in the code to create a spanning tree. Lines will be added to the graph as vertices based on when they are called. This will be used to filter out the lines that will not run such as empty lines or comments.
- A queue could be used to go through the file that coverage was called upon line by line. Each line will be enqueued and when it needs to be run it will be dequeued. This way we can keep track of the lines that have already been run.

2.2

There is a small readme with most of the information on the docs elsewhere. There are also many extraneous files, files like Makefile, .travis.yml are confusing without the background knowledge to know what they do. Most of the src is Python which is nice. The code itself is pretty well commented, if quite complex. Some of the methods in data have wrappers and hidden methods (_name) which make it harder to read

2.3

Data.py

The main function of this file is to define a class CoverageDataJson. This class seems to be an interface into the rest of the code, providing a consistent api for reading the data collected by the

program. This means that the devs can change how the program works but as long as you interface with this, you will remain safe.

The comments are more useful, they are verbose and consistent. The code is pretty clean too.

They use sets, iterables, and JSON as the main types in this file. JSON is the main type here, used to store the data about each file, lines and arcs are some of the fields. Most of this is imported with lists/sets and updated into the file.

It's readable, but hard to parse through, comments are very useful. This code is cleaner, yet harder to use than mine.

Collector.py

This manages the collector which has tracerers that manage each file. These tracers look to be multithreaded. The comments are very useful here, they give a good explanation of what is happening and why.

There is a stack of collectors where only the top can be active at once. The collectors collect the data from each trace function which traces a thread of a file. The code quality is good, especially for being such a complex function dealing with threading. Each functions use is fairly clear, some are confusing due to it being a class. This code is commented better than mine and written cleaner, I would be happy to maintain this.

Results.py

There are two classes here, one called numbers, which seems to hold the actual numeric data of the run (files, statements, excluded...) and Analysis which applies functions to and returns the numbers. There are fewer comments, so the code is more useful.

There are graphs here, following the arcs, as well as hash maps (dicts). Arcs are graphs of how lines map in between data. Missing_branch_arcs returns a dict of each line mapped to a list of other lines used in analysis.

The code here is fairly clean, but not as well commented as some other functions. Some of the functions are unclear, especially those with wrappers. The code itself is simpler so I wouldn't feel that bad maintaining it.

Summary.py

The main function of this file is to combine the results of the coverage file and output them to the user. The comments are very useful as they clearly explain what is happening in each section of the code.

A list is used to store coverage results from each file, populated by the analyze function.

The second list is used to output the coverage data. It is a list of tuples containing text and percentage of coverage for the output line. This coverage percentage is obtained from the previous list.

The code is readable due to the comments in every function. Since the readability is good I feel that it can be maintained and updated easily.