Problem: To computes Strongly connected components for the given directed graph where the input is given as a text file and output is also given as a text file

Expected Input Format

First line gives number of vertices and number of edges. Every other line represents a directed edge

```
5 7
1 2
2 2
2 3
2 5
3 1
3 4
4 3
```

Expected Output Format:

```
component
4
3
2
1
component
5
```

How to Run the program:

1. cd scc
2. python main.py input_graph.txt

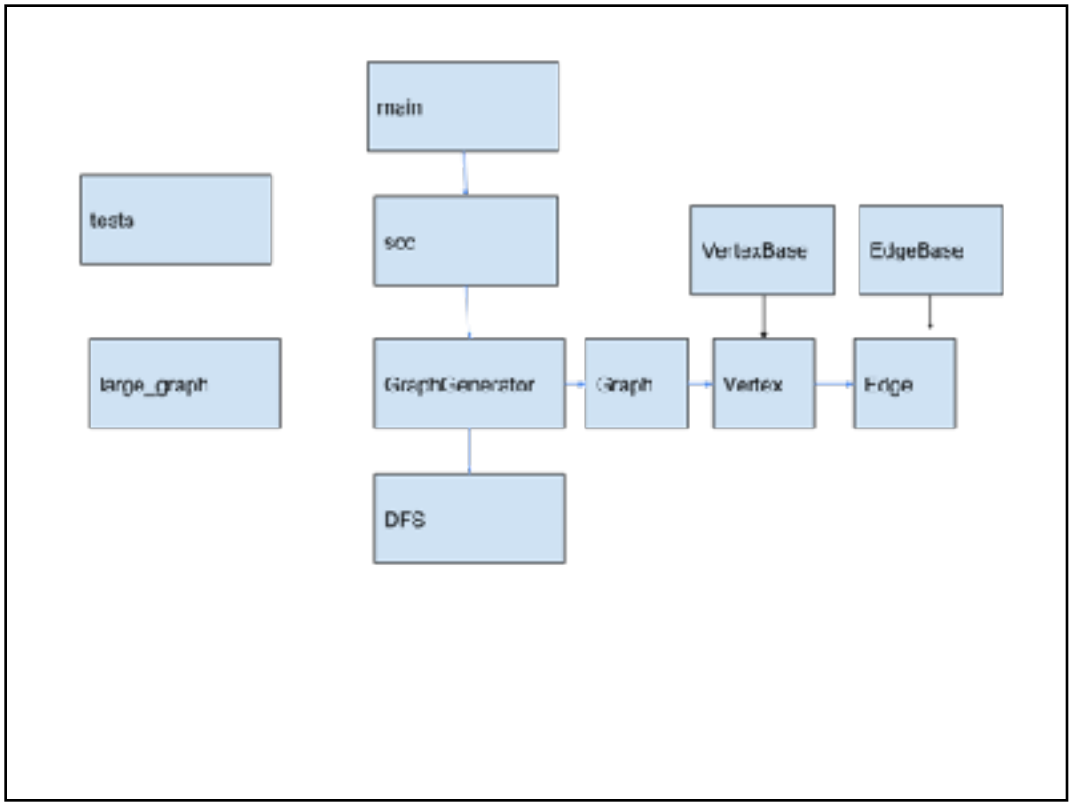
Output:

1. cd scc
2. open output_input_graph.txt

Explanation:

The program starts from the main.py python module which expects a filepath as argument. The input file needs to be given in the expected format to get the desired output.

Graphical view of the call stack



Data structures and classes:

- A Graph is essentially represented as a list of vertices where lists are the basic python datatypes. Adjacency list representation of graph is chosen because its both memory efficient and also DFS essentially is more convenient to implement with such representation.
- A vertex has a value and an adjacency list along with additional parameters specific to the DFS search. Adjacency list in a vertex is a list of edges.
- An edge is represented as a directed edge with a from vertex, to vertex and optional weight
- Graph generator takes the input file as input and generates the graph object in memory.
- DFS computes depth first search on graph object

Steps:

1. Generates graph from input file - O(E)
2. Computes DFS - O(V + E)
3. Generates a transpose of the original graph which is a new copy of the graph but transposed. - O(V + E)
4. Computes DFS again on transposed graph - O(V + E)
5. Writes the output to a text file - O(E)

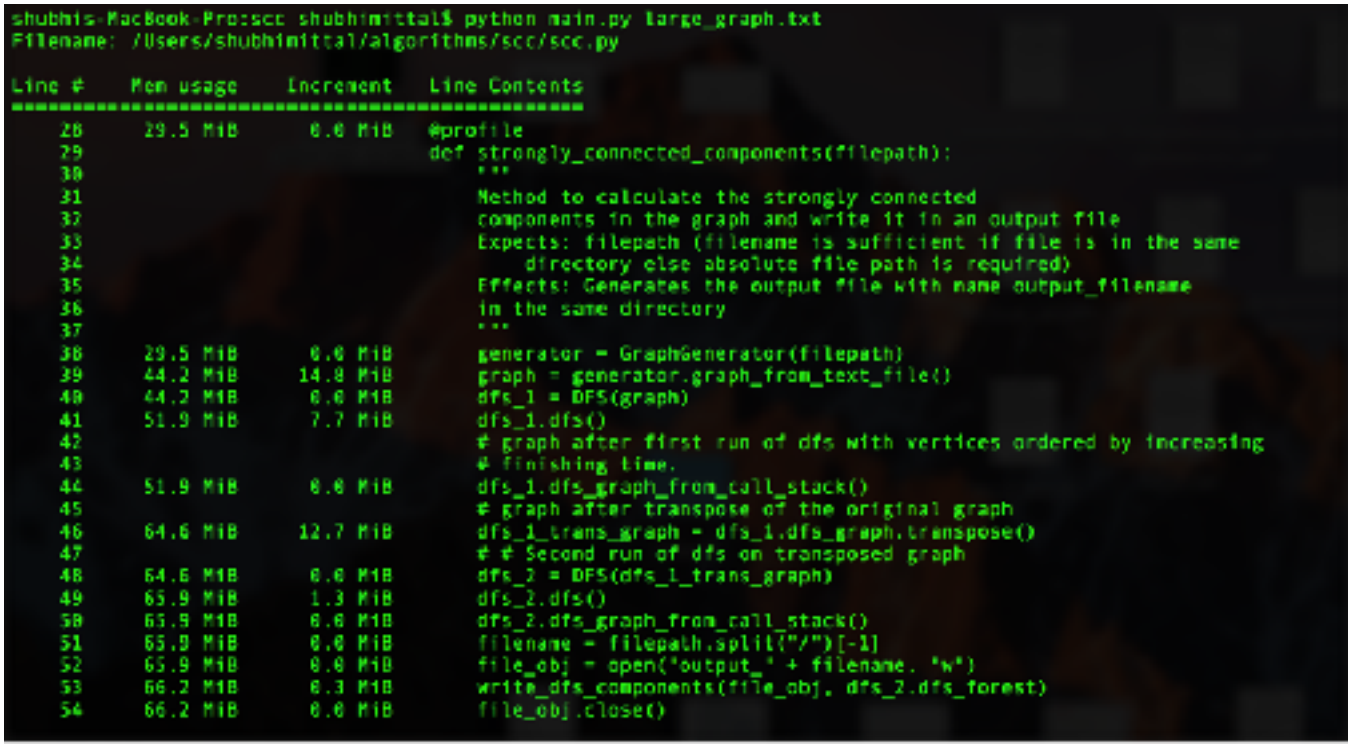
Complexity Analysis:

The overall complexity is O(V + E)

Sanity Tests:

There are some sample graphs that were used to test the sanity of the code. These can be found in tests.py

Efficiency Tests and limitations:
large_graph.py can be used to generate a very large graph with one connected component. The recursion depth for python is increased as python is very strict about the recursion depths. The profiling results are as follows for the input size of 2^14. For latter input the program halts with segmentation fault (out of memory issues). For benchmarking, the program is run on 2.7 GHz Intel Core i5, 8GB DDR3, MacBook Pro.



```
shubhis-MacBook-Pro:scc shubhimittal$ python main.py large_graph.txt
Filename: /Users/shubhimittal/algorithms/scc/scc.py

Line #    Mem usage    Increment   Line Contents
================================================
    28     29.5 MiB      0.0 MiB   @profile
    29                             def strongly_connected_components(filepath):
    30                                 """
    31                                 Method to calculate the strongly connected
    32                                 components in the graph and write it in an output file
    33                                 Expects: filepath (filename is sufficient if file is in the same
    34                                     directory else absolute file path is required)
    35                                 Effects: Generates the output file with name output_filename
    36                                 in the same directory
    37                                 """
    38     29.5 MiB      0.0 MiB       generator = GraphGenerator(filepath)
    39     44.2 MiB     14.8 MiB       graph = generator.graph_from_text_file()
    40     44.2 MiB      0.0 MiB       dfs_1 = DFS(graph)
    41     51.9 MiB      7.7 MiB       dfs_1.dfs()
    42                                 # graph after first run of dfs with vertices ordered by increasing
    43                                 # finishing time.
    44     51.9 MiB      0.0 MiB       dfs_1.dfs_graph_from_call_stack()
    45                                 # graph after transpose of the original graph
    46     64.6 MiB     12.7 MiB       dfs_1_trans_graph = dfs_1.dfs_graph.transpose()
    47                                 # # Second run of dfs on transposed graph
    48     64.6 MiB      0.0 MiB       dfs_2 = DFS(dfs_1_trans_graph)
    49     65.9 MiB      1.3 MiB       dfs_2.dfs()
    50     65.9 MiB      0.0 MiB       dfs_2.dfs_graph_from_call_stack()
    51     65.9 MiB      0.0 MiB       filename = filepath.split("/")[-1]
    52     65.9 MiB      0.0 MiB       file_obj = open('output_' + filename, 'w')
    53     66.2 MiB      0.3 MiB       write_dfs_components(file_obj, dfs_2.dfs_forest)
    54     66.2 MiB      0.0 MiB       file_obj.close()
```