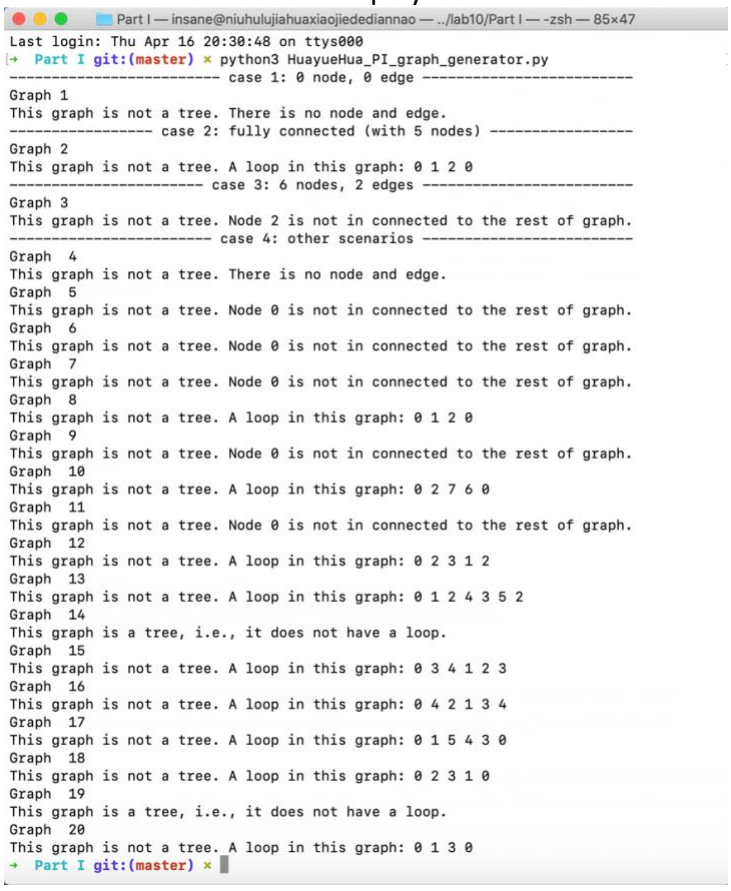
Name: Huayue Hua
USC ID: 9817961224
Email: huayuehu@usc.edu

Lab10

Mandatory Part

HEADS UP: All files in the mandatory part of this lab should be in the SAME folder
Commands to be run for each part:
Part I: **python3 HuayueHua_PI_graph_generator.py**
Part II: **python3 HuayueHua_PII_top_execute.py**
Part III: **python3 HuayueHua_PIII_clique_checker.py**

Part I Graph Valid Tree Checker
There are two files for this part:
HuayueHua_PI_tree_checker.py
HuayueHua_PI_graph_generator.py
(1) To run this part:
Simply run the following command:
python3 HuayueHua_PI_graph_generator.py
Result like this should be displayed in the terminal interface:



(2) In **HuayueHua_PI_graph_generator.py**, I import HuayueHua_PI_tree_checker as tc. Every time I generate a new graph, input.txt would be rewrite, then tc.main() would be called to check whether this graph is a tree or not.

(3) To write cycle checker function in **HuayueHua_PI_tree_checker.py**, I take https://www.geeksforgeeks.org/check-given-graph-tree/ as a reference.

The idea for **tree checker** is to read node_num, edge_num and edges connection information from the input.txt, then I check if the graph has cycle. If there is a cycle, print out corresponding information. If there is no cycle, check the connectivity to see if the graph is a tree.

Pseudocode for **cycle_checker()** function:

```
1.  def cycle_checker(node, parent, visited):
2.      visited[node] = True
3.
4.      for i in graph[node]:
5.          if i not visited yet:
6.              recursively check other node connected using DFS fashion:
7.                  if detect cycle => return True
8.          elif node i is visited and node i is not the parent node => cycle detected
9.              return True
10.
11.     return False
```

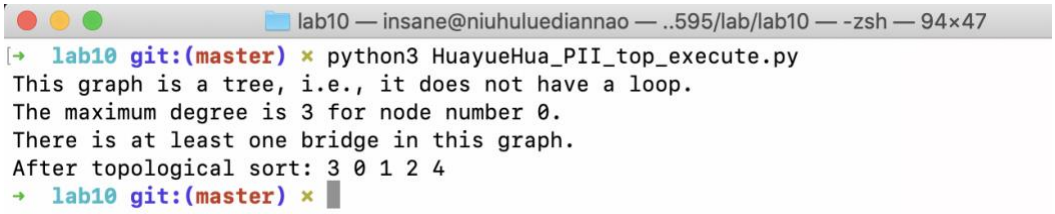
Pseudocode for **connectivity_checker()** function:

```
1.  def connectivity_checker(graph, node_num):
2.      initially set visited array to False
3.
4.      for each node:
5.          for i in graph[node]:
6.              set visited[i] to True
7.
8.      for each node:
9.          check if visited[node] is False,:
10.             return False if there is a False
11.
12.     return True
```

Part II Miscellaneous

There are two files for this part:
HuayueHua_PII_top_execute.py
HuayueHua_PII.cpp

(1) To run this part:
Simply run the following command:
python3 HuayueHua_PII_top_execute.py
Result like this should be displayed in the terminal interface:



(2) In **HuayueHua_PII.cpp**, I do the following three things:
Calculate the maximum degree
Check whether there is a bridge in the graph
Perform topological sorting

To calculate the maximum degree, I simply find the degree of every node in the graph and return the maximum one.

To write bridge checker part, I take https://www.geeksforgeeks.org/bridge-in-a-graph/ as a reference. In bridge_checker() function, visited array is to track whether a node has been visited or not. Visited_time and less_time stores the discovery time of the visited node and less time that the visited tree can be visited by other path if exists. Parent stores the parent node of current node. We initial visited array to false and all parent to -1. Iterate all the nodes, if not visited yet, perform DFS recursively to update visited_time and less_time information to decide whether there is a bridge in the graph or not.

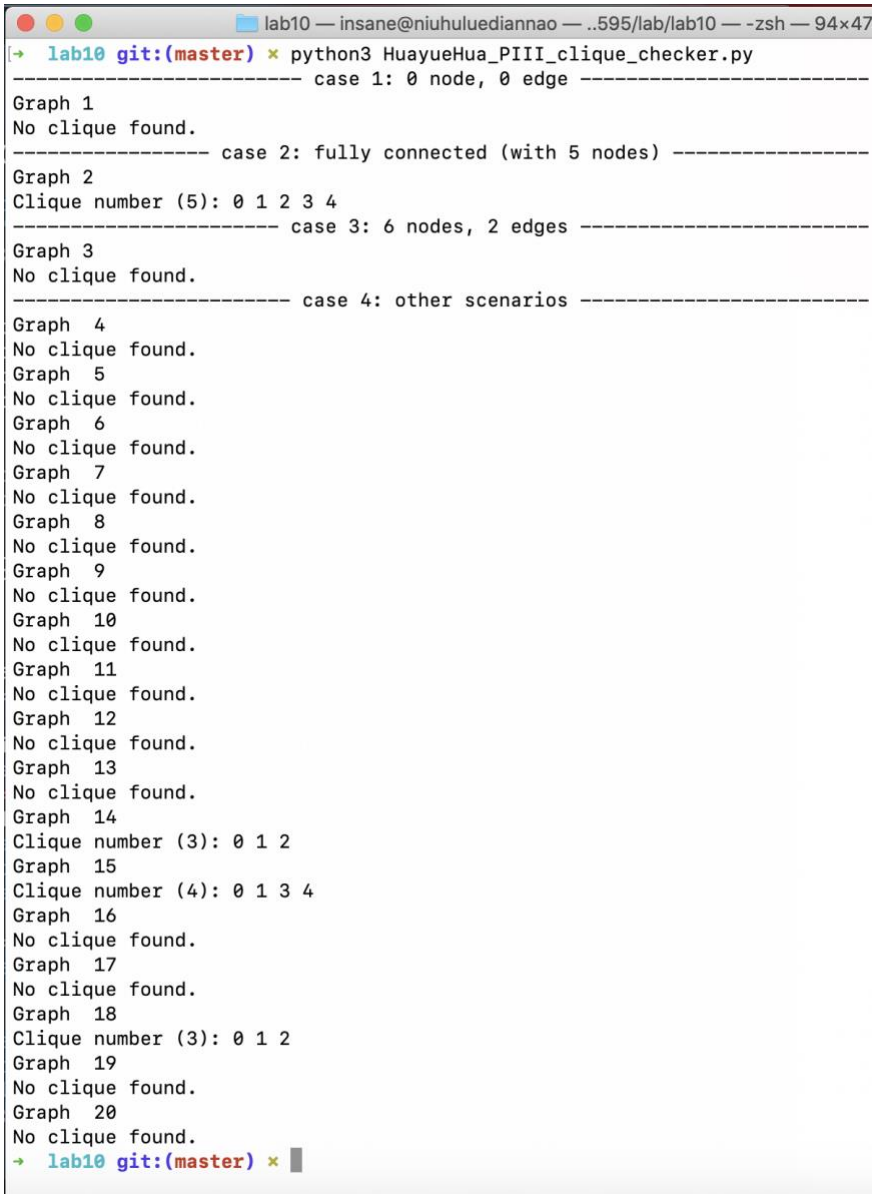
To write topo sort part, I take https://songlee24.github.io/2015/05/07/topological-sorting/ as a reference. When adding edges in the previous procedure, I use in_degree to check the direction of each edge, so if a node with in_degree equals to 0, this node is a root node. Push the root node to queue, for each node in the queue, print out the current node in order and also push all the adjacent nodes to the current node to queue. In this way, node will be sorted.

(3) In **HuayueHua_PII_top_execute.py**, I firstly call tree checker in part I to check whether the graph represented by input.txt is a tree or not. If it is a tree, then using subprocess.call() to help run **HuayueHua_PII.cpp.**

Part III Find the maximum clique of a graph

There are two files for this part:
HuayueHua_PIII_clique_checker.py
HuayueHua_PIII_max_clique_calc.py

(1) To run this part:
Simply run the following command:
python3 HuayueHua_PIII_clique_checker.py
Result like this should be displayed in the terminal interface:



(2) In **HuayueHua_PIII_max_clique_calc.py**, for each edge, when an edge is added to the present nodes list, check if the present nodes list forms a clique, keeping adding nodes until the list does not form a clique. In **HuayueHua_PIII_clique_checker.py**, it's the same as the part I graph generator, except that it imports HuayueHua_PIII_max_clique_calc.

To write this part, I take https://www.geeksforgeeks.org/maximal-clique-problem-recursive-solution/ as a reference