Project 1: Graduation Time

Group members;

Thomas Ngo tngo0508@csu.fullerton.edu
Rachana Chittari chittari.rachana@csu.fullerton.edu
Uday Margadi uday5746@csu.fullerton.edu

Report

For the given problem, read from the input file and generate a graph for the given vertices and edges which can store index, color, predecessor, discovery time(d), finishing time(f) of each node.

Create an adjacency matrix[i][j] of 0s and 1s, where 1 indicates existence of an edge from i to j, 0 when there is no edge from i to j.

Topologically sort the graph using topological sort algorithm and print the vertices based on their finishing time in descending order.

To find the longest path use a list data structure to store the number of predecessors each node has, the max() function on the list gives the length of the longest path and display the nodes in the longest path.

Pseudocode for Overall Algorithm:

```
//Read the input file from the argument
 i=0
 while True:
        line = f.readline().strip()
        i = i+1
        vertices[line] = node[i]

//Append edges
 While True:
        line = f.readline().strip()
        edges.append(line.split())

//Create a graph and Adjacency Matrix with the given vertices and edges
 for e in edges:
```

```
            adj_matrix[e[0]][e[1]] = 1

//Topologically sort the graph
  initialise t = 0
  create an adjacency matrix with 1s and 0s
            adj_matrix[i][j] = 1 <-- if there exists an edge from node i to node j
            adj_matrix[i][j] = 0 <-- if there exists no edge from node i to node j

    if adj_matrix[i][j] == 1 and node.color == 'white'
        dfs_visit(node, vertices)
            t = t+1
            node.d = t
            node.color = 'gray'
            for index, successor in enumerate(adj_matrix[node].index)
                adj_node = map[index]
                if successor == 1 and adj_node.color == 'white'
                    adj_node.predecessor = node
                    dfs_visit(adj_node, vertices)
            node.color = 'black'
            t == t+1
            node.f = t
            sorted_array.append(node)
    print(list(reversed(sorted_array)))

//Finding the Longest Path
 //the following condition helps us to store predecessor of a node
 //if adj_node for node exists and adj_node.color == 'white'
//          [adj_node].predecessor = node
  pathLength = [0] * len(sorted_array)
  for i, element in enumerate(sorted_array)
            if element.predecessor == None
                    return pathLength[i] = 0
```

```
        else
                pathLength[i] = pathLength[sorted_array.index(element.predecessor)] + 1
print(max(pathLength))

//To display the Longest path

current_node = sorted_array[pathlength.index(max(pathLength))]
while current_node;
        path.append(current_node.index)
        current_node = current_node.predecessor
for x in reversed(path):
        print(x)
```

Pseudocode for Topological Sort:

```
 initialise t = 0
 create an adjacency matrix with 1s and 0s
        matrix[i][j] = 1 <-- if there exists an edge from node i to node j
        matrix[i][j] = 0 <-- if there exists no edge from node i to node j

 if matrix[i][j] == 1 and node.color == 'white'
     dfs_visit(node, vertices)
         t = t+1
         node.d = t
         node.color = 'gray'
         for index, successor in enumerate(matrix[node].index)
              next_node = map[index]
              if successor == 1 and next_node.color == 'white'
                 next_node.predecessor = node
                 dfs_visit(next_node, vertices)
         node.color = 'black'
         t == t+1
```

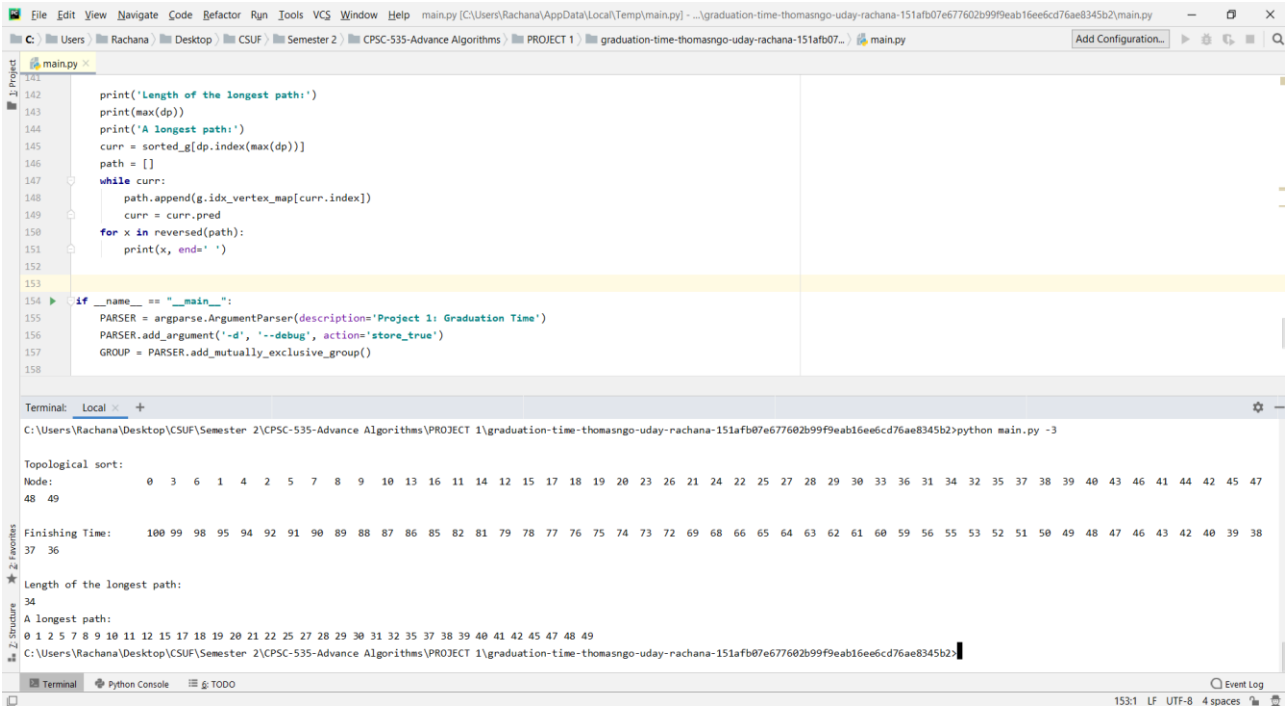
```
            node.f = t
            sorted_array.append(node)
    return list(reversed(sorted_array))
```

Pseudocode for Longest path:

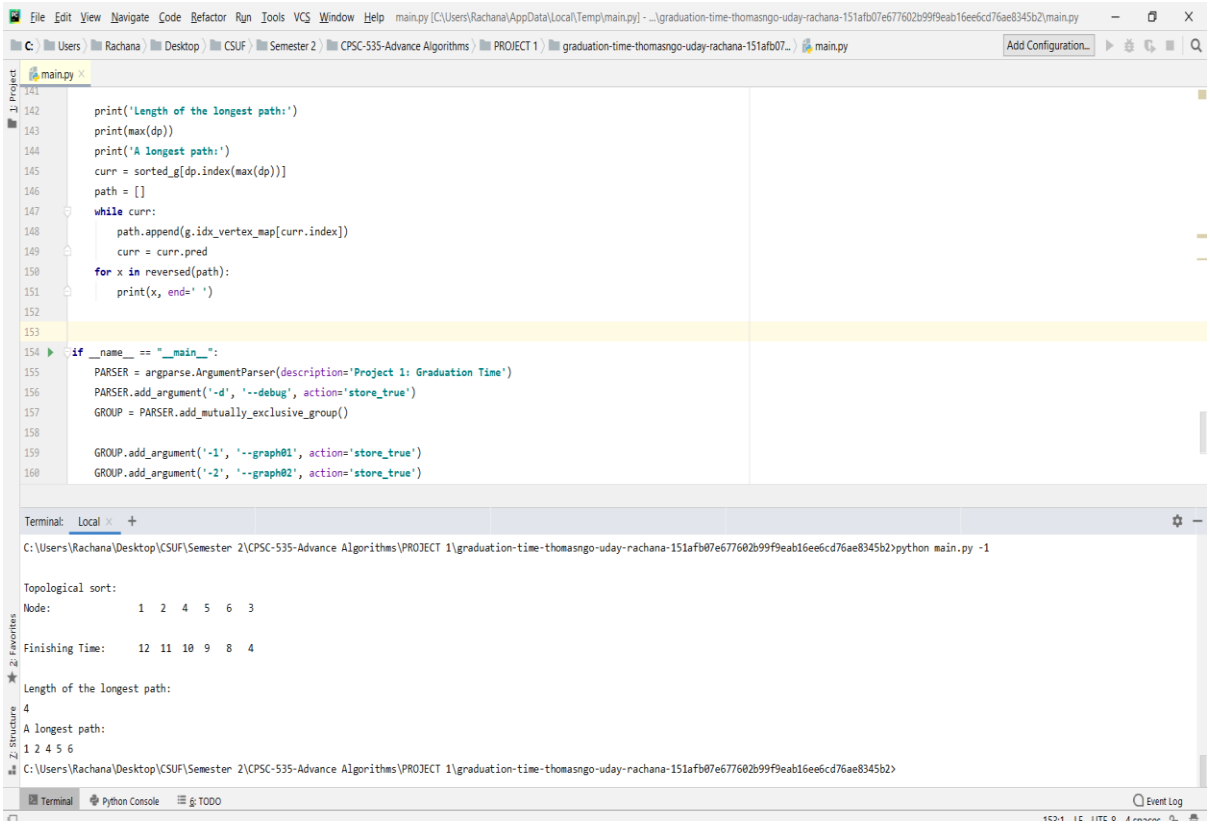
```
Create a graph G with given vertices and edges
topologically sort the graph and store it in sortedArray[]
the following condition helps us to store predecessor of a node
//if adj_node for the current_node exists and adj_node.color == 'white'
      //  [adj_node].predecessor = node

pathLength = [0] * len(sortedArray)
for i, element in enumerate(sortedArray)
        if element.predecessor == None
                return pathLength[i] = 0
        else
                pathLength[i] = pathLength[sortedArray.index(element.predecessor)] + 1
return max(pathLength)
```

Screenshots

Screenshot 1: biggraph.txt



Screenshot 2: graph01.txt



Screenshot 3: graph02.txt