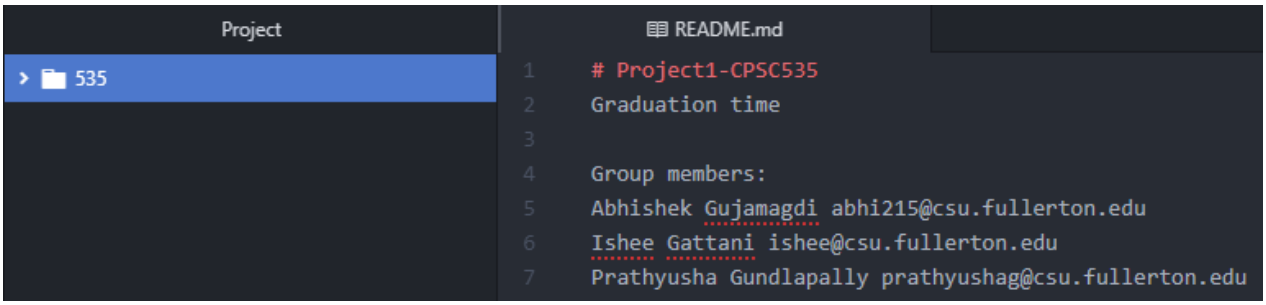
Programming Assignment 1:

Prathyusha Gundlapally: prathyushag@csu.fullerton.edu

Ishee Gattani: ishee@csu.fulleton.edu

Abhishek Gujamagdi: abhi215@csu.fullerton.edu



Pseudo code for overall algorithm

Input: G is a directed acyclic graph (DAG)

T [] = List of Topological sorted nodes

Visited [] = List that keeps track of visited and unvisited nodes.

```
topological_sort (cur_vert, N, adj [] []) {
   visited[cur_vert] = true
   for i = 0 to N
      if adj[cur_vert[i] is true and visited[i] is false
      topological_sort(i)
   T. insert_in_beginning(cur_vert)
}
return T
```

Initialize dist [] = {0, 0, ….}

Initialize longestPath (self, start, end)

Initialize comesfrom dictionary

Create a topological order of all vertices as explained in the above part of the pseudo code.

for each vertex u ∈ V in linearized order do

// compute dist(u) = max $_{(v, u) \in E}$ {dist(v) + 1}

for every v ∈ V do

if (A[u][v] == 1 && dist[v] < dist[u] + 1)

then dist[v] = dist[u] + 1

comesfrom[v]=u

endif

enddo

endfor

return max$_{v \in V}$ {dist(v)}

Initialize maxpath []

maxpath = [end]

while maxpath [-1]! = start:

maxpath. append (comesfrom [maxpath [-1]])

maxpath. reverse ()

return maxpath

Algorithm for Topological Sorting

- call DFS(G) to compute finishing times f[v] for each vertex v
- as each vertex is finished, insert it onto the front of a linked list
- return the linked list of vertices

G: Directed Acyclic Graph

f[v]: Finishing time of the vertex

Pseudo code for Topological sorting

T [] = List of Topological sorted nodes

Visited [] = List that keeps track of visited and unvisited nodes.

topological_sort (cur_vert, N, adj [] []) {

```
    visited[cur_vert] = true
    for i = 0 to N
        if adj[cur_vert][i] is true and visited[i] is false
        topological_sort(i)
    T. insert_in_beginning(cur_vert)
}
```

Algorithm for DAG Longest Path

Longest-path-DAG(G)

Input: Unweighted DAG G = (V, E)

Output: Longest path in G

1: Topologically sort G

2: Initialize array dist [] = {MIN_INT, …. MIN_INT} // size N

3: dist [0] = 0

4: for each vertex u ∈ V in linearized order do

// compute dist(u) = $\max_{(v, u)\in E}$ {dist(v) + 1}

5: for every v ∈ V do

6: if (A[u][v] == 1 && dist[v] < dist[u] + 1)

7: then dist[v] = dist[u] + 1

8: endif

9: enddo

10: endfor

11: return $\max_{v\in V}$ {dist(v)}

Pseudo code for DAG longest path

1) Initialize dist [] = {NINF, NINF, ….} and dist[s] = 0 where s is the source vertex. Here NINF means negative infinite.

2) Create a topological order of all vertices.

3) Do following for every vertex u in topological order.

Do following for every adjacent vertex v of u:

if (dist[v] < dist[u] + weight (u, v)):

dist[v] = dist[u] + weight (u, v)

Description on how to run the code:

*.py files (biggraph.py;graph01.py;graph02.py)are the input files that contain the appropriate code for the topological sorting followed by computing the longest path of DAG along with the length of the path which can be executed as python filename.py into the command line then hit enter. Else, we can also use the IDEs like Jupyter, Pycharm, Spyder, Atom to execute the code by hitting the run command.

Steps:

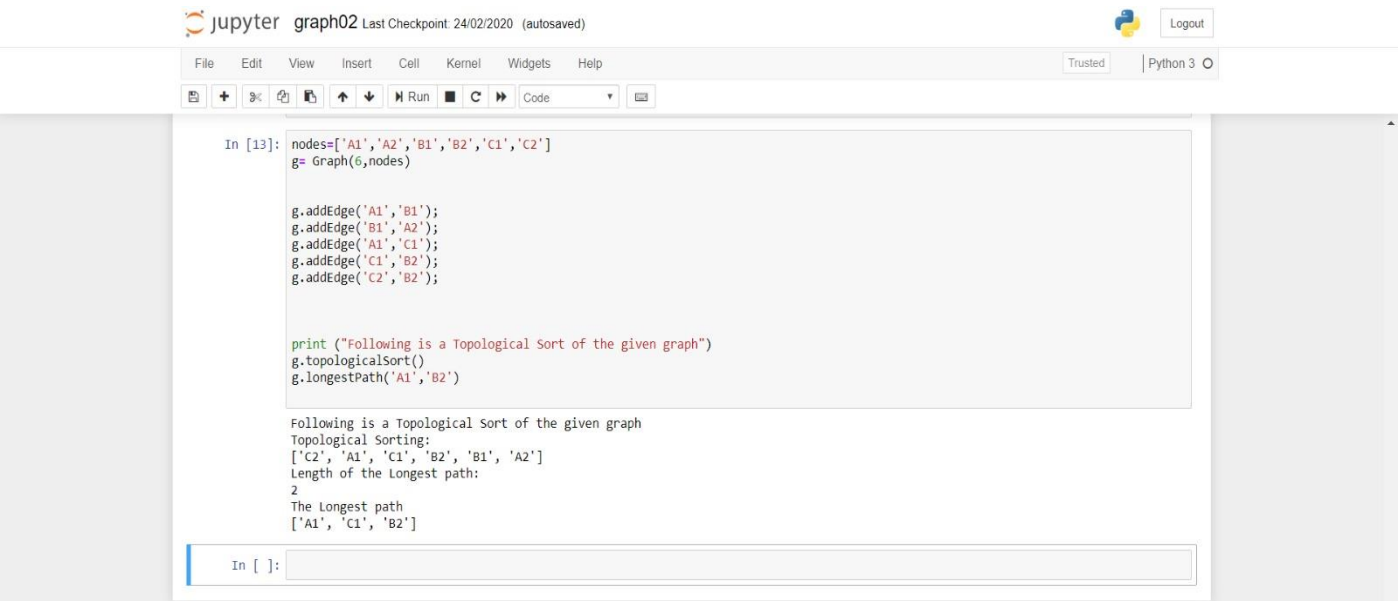
1.User have to provide the graph detailed inputs for each of the above-mentioned python files.

2.Code would be then processed for topological sorting first followed by computing the longest path.

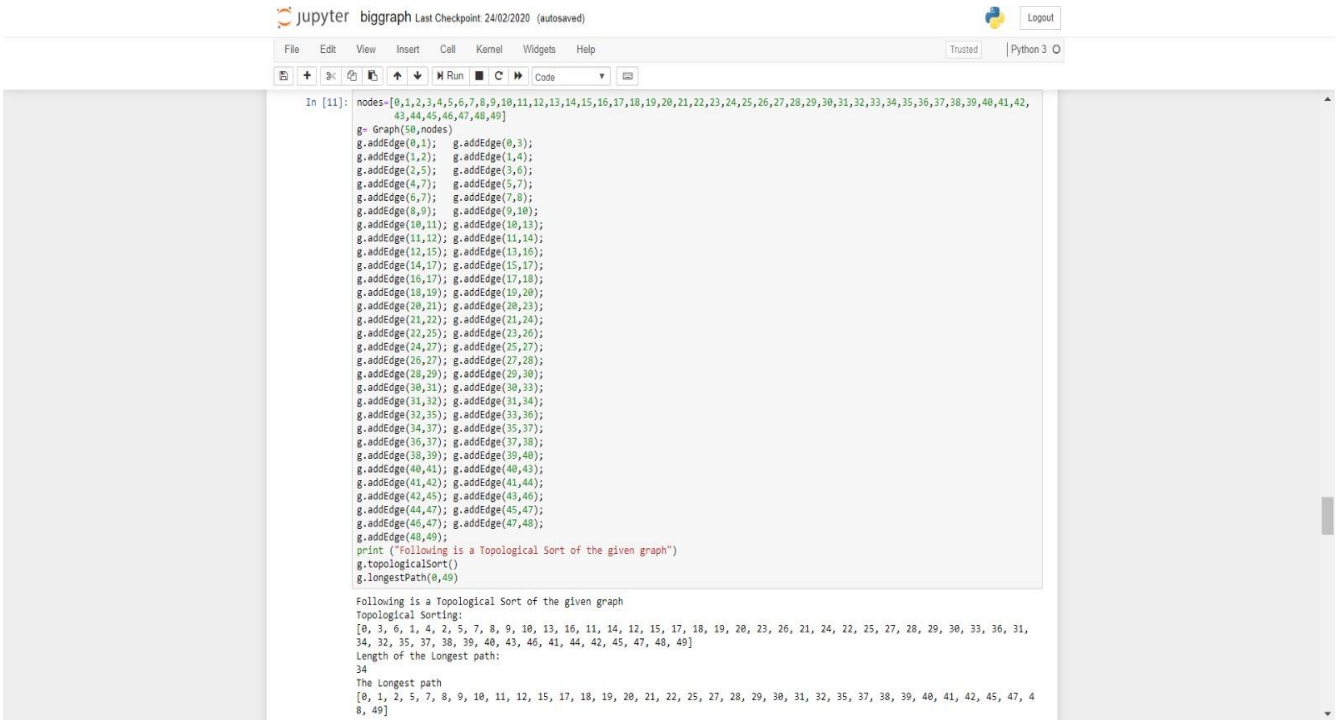
3. And the output would be printed showing the longest path along with the length of the path.

Snapshots of code executing for the three given input files.

graph02.py



biggraph.py



graph01.py