

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
```

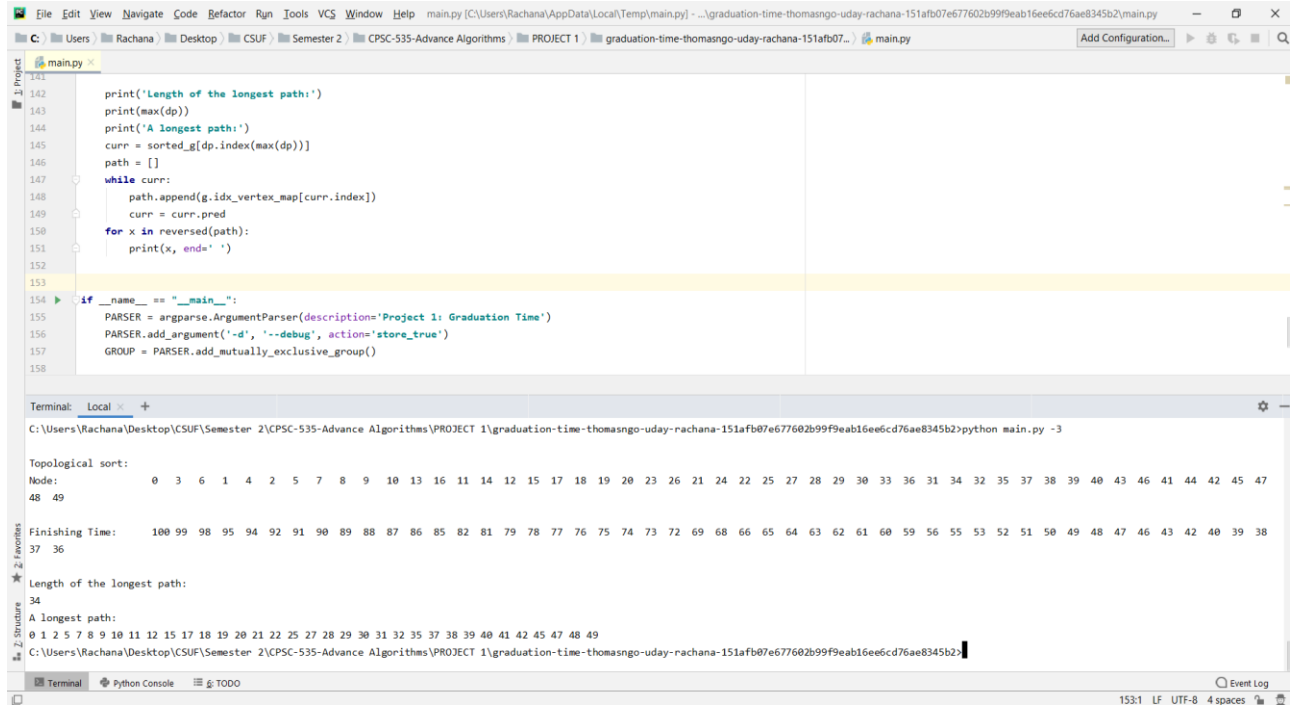
```
    else
```

```
        pathLength[i] = pathLength[sortedArray.index(element.predecessor)] + 1
```

```
return max(pathLength)
```

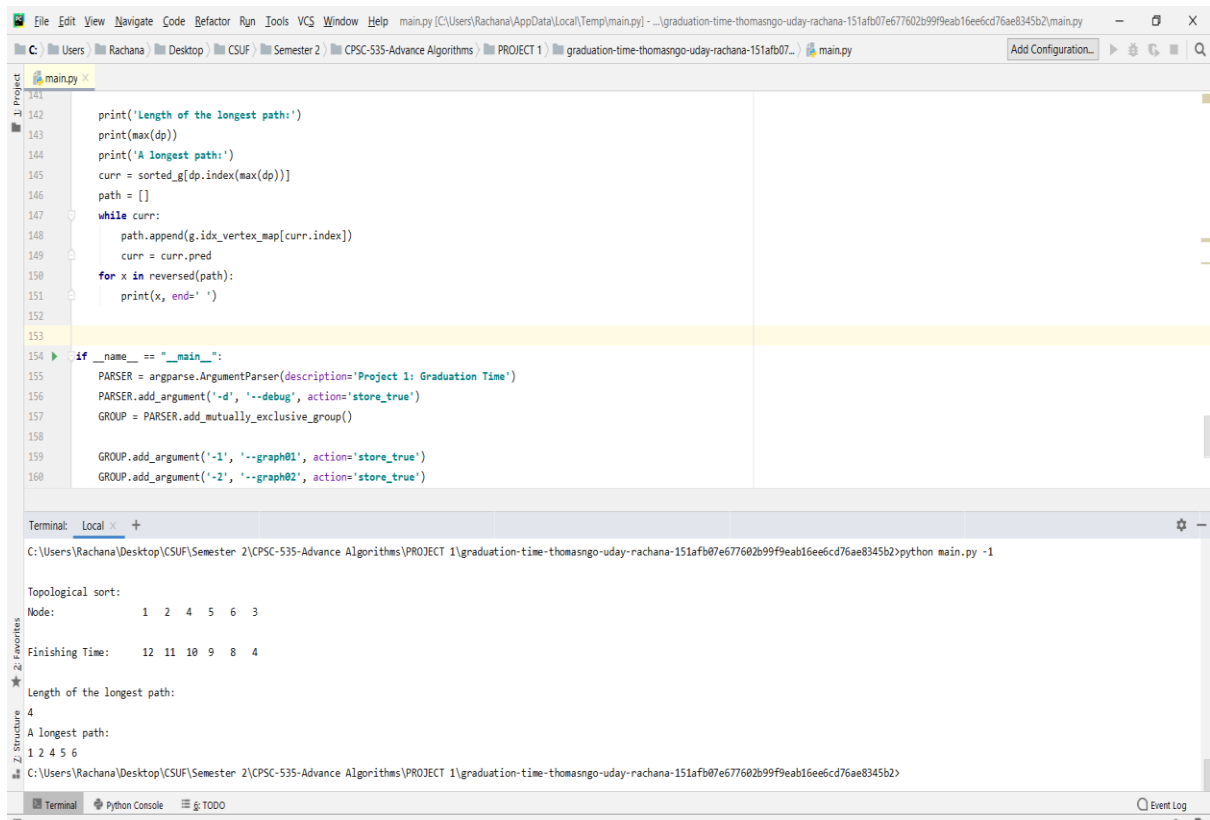
Screenshots

Screenshot 1: biggraph.txt



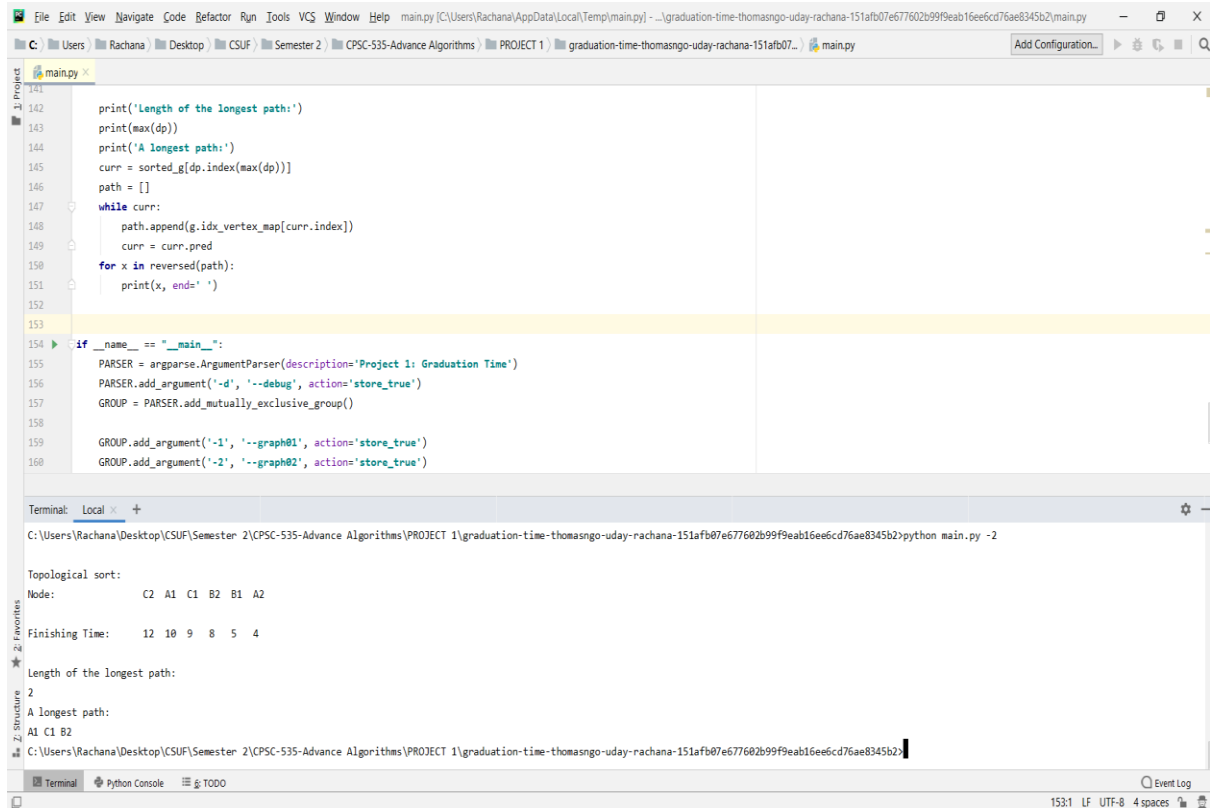
```
File Edit View Navigate Code Refactor Run Tools VCS Window Help main.py [C:\Users\Rachana\AppData\Local\Temp\main.py] - ... \graduation-time-thomasngo-uday-rachana-151afb07e677602b99f9eab16ee6cd76ae8345b2\main.py
C:\Users\Rachana\Desktop\CSUF\Semester 2\CPSC-535-Advance Algorithms\PROJECT 1\graduation-time-thomasngo-uday-rachana-151afb07e677602b99f9eab16ee6cd76ae8345b2\main.py
main.py
print('Length of the longest path:')
print(max(dp))
print('A longest path:')
curr = sorted_g[dp.index(max(dp))]
path = []
while curr:
    path.append(g.idx_vertex_map[curr.index])
    curr = curr.pred
for x in reversed(path):
    print(x, end=' ')
if __name__ == "__main__":
    PARSER = argparse.ArgumentParser(description='Project 1: Graduation Time')
    PARSER.add_argument('-d', '--debug', action='store_true')
    GROUP = PARSER.add_mutually_exclusive_group()
Terminal: Local x +
C:\Users\Rachana\Desktop\CSUF\Semester 2\CPSC-535-Advance Algorithms\PROJECT 1\graduation-time-thomasngo-uday-rachana-151afb07e677602b99f9eab16ee6cd76ae8345b2>python main.py -3
Topological sort:
Node:      0  3  6  1  4  2  5  7  8  9  10 13 16 11 14 12 15 17 18 19 20 23 26 21 24 22 25 27 28 29 30 33 36 31 34 32 35 37 38 39 40 43 46 41 44 42 45 47 48 49
Finishing Time: 100 99 98 95 94 92 91 90 89 88 87 86 85 82 81 79 78 77 76 75 74 73 72 69 68 66 65 64 63 62 61 60 59 56 55 53 52 51 50 49 48 47 46 43 42 40 39 38 37 36
Length of the longest path:
34
A longest path:
0 1 2 5 7 8 9 10 11 12 15 17 18 19 20 21 22 25 27 28 29 30 31 32 35 37 38 39 40 41 42 45 47 48 49
C:\Users\Rachana\Desktop\CSUF\Semester 2\CPSC-535-Advance Algorithms\PROJECT 1\graduation-time-thomasngo-uday-rachana-151afb07e677602b99f9eab16ee6cd76ae8345b2
```

Screenshot 2: graph01.txt



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help main.py [C:\Users\Rachana\AppData\Local\Temp\main.py] - ... \graduation-time-thomasngo-uday-rachana-151afb07e677602b99f9eab16ee6cd76ae8345b2\main.py
C:\Users\Rachana\Desktop\CSUF\Semester 2\CPSC-535-Advance Algorithms\PROJECT 1\graduation-time-thomasngo-uday-rachana-151afb07e677602b99f9eab16ee6cd76ae8345b2\main.py
main.py
print('Length of the longest path:')
print(max(dp))
print('A longest path:')
curr = sorted_g[dp.index(max(dp))]
path = []
while curr:
    path.append(g.idx_vertex_map[curr.index])
    curr = curr.pred
for x in reversed(path):
    print(x, end=' ')
if __name__ == "__main__":
    PARSER = argparse.ArgumentParser(description='Project 1: Graduation Time')
    PARSER.add_argument('-d', '--debug', action='store_true')
    GROUP = PARSER.add_mutually_exclusive_group()
    GROUP.add_argument('-1', '--graph01', action='store_true')
    GROUP.add_argument('-2', '--graph02', action='store_true')
Terminal: Local x +
C:\Users\Rachana\Desktop\CSUF\Semester 2\CPSC-535-Advance Algorithms\PROJECT 1\graduation-time-thomasngo-uday-rachana-151afb07e677602b99f9eab16ee6cd76ae8345b2>python main.py -1
Topological sort:
Node:      1  2  4  5  6  3
Finishing Time: 12 11 10 9 8 4
Length of the longest path:
4
A longest path:
1 2 4 5 6
C:\Users\Rachana\Desktop\CSUF\Semester 2\CPSC-535-Advance Algorithms\PROJECT 1\graduation-time-thomasngo-uday-rachana-151afb07e677602b99f9eab16ee6cd76ae8345b2
```

Screenshot 3: graph02.txt



The screenshot shows an IDE window with a Python file named `main.py`. The code implements a graph algorithm to find the longest path. It includes a `while` loop to build the path and a `for` loop to print the path in reverse order. The code also features a command-line argument parser for debugging and selecting between two graph types.

```
141
142     print('Length of the longest path:')
143     print(max(dp))
144     print('A longest path:')
145     curr = sorted_g[dp.index(max(dp))]
146     path = []
147     while curr:
148         path.append(g.idx_vertex_map[curr.index])
149         curr = curr.pred
150     for x in reversed(path):
151         print(x, end=' ')
152
153
154 > if __name__ == "__main__":
155     parser = argparse.ArgumentParser(description='Project 1: Graduation Time')
156     parser.add_argument('-d', '--debug', action='store_true')
157     group = parser.add_mutually_exclusive_group()
158
159     group.add_argument('-1', '--graph01', action='store_true')
160     group.add_argument('-2', '--graph02', action='store_true')
```

The terminal output shows the results of running the program with the `--graph02` flag:

```
Terminal: Local x +
C:\Users\Rachana\Desktop\CSUF\Semester 2\CPSC-535-Advance Algorithms\PROJECT 1\graduation-time-thomasngo-uday-rachana-151afb07e677602b99f9eab16ee6cd76ae8345b2>python main.py -2

Topological sort:
Node:          C2 A1 C1 B2 B1 A2

Finishing Time: 12 10 9 8 5 4

Length of the longest path:
2
A longest path:
A1 C1 B2
```

The status bar at the bottom indicates the file is at line 153, column 1, using UTF-8 encoding with 4 spaces.