CCDSALG Term 3, AY 2024 – 2025

Project 2 Documentation – Social Network (Undirected Graph Application)

DECLARATION OF INTELLECTUAL HONESTY / ORIGINAL WORK

I declare that the project that I'm submitting (as part of a group) is the product of my own intellectual efforts. No part of the project was copied from any source, and no part was shared with another person.

NAME#1:_____*Jack Owen L. Morgan*______ SIGNATURE#1: ______ [illegible] _______________ SECTION: S11_

The following are my specific contributions to MCO2:

1. Conceptualizations
2. Testing
3. Documentation

Based on my honest estimate, and in consultation with my groupmates, I contributed approximately **33.33%** to the entire project.

Example: if you encoded 40% it would mean you did 40% of the entire project on your own from say from conceptualization, to testing and documentation, while the remaining 60% were done by your groupmates. Your contribution PLUS the contribution of your groupmates should total to 100%.

NAME#2:_____Benette Enzo V. Campo______ SIGNATURE#2: ______ [illegible] _________ SECTION: _S12_

The following are my specific contributions to MCO2:

1. Development
2. Testing
3. Documentation

Based on my honest estimate, and in consultation with my groupmates, I contributed approximately **33.33%** to the entire project.

NAME#3: _____Marcus Timothy V. Ramos_____ SIGNATURE#3: ______ [illegible] _______________ SECTION: _S12_

The following are my specific contributions to MCO2:

1. Testing
2. Debugging
3. Documentation

Based on my honest estimate, and in consultation with my groupmates, I contributed approximately **33.33%** to the entire project

1. Indicate how to compile (if it is a compiled language) your codes, and how RUN (execute) your program from the COMMAND LINE. Examples are shown below highlighted in yellow. Replace them accordingly. Make sure that all your group members test what you typed below because I will follow/copy-paste them verbatim. I will initially test your solution using some of the sample input text file that you submitted. Thereafter, I will run it again using my own test data:

- How to compile from the command line

  CCDSALG> **gcc -Wall main.c -o main.exe**

Next, answer the following questions:

a. Is there a compilation (syntax error) in your codes? (YES or NO). _NO_

   WARNING: the project will automatically be graded with a score of **0** if there is syntax error in any of the submitted source code files. Please make sure that your submission does not have a syntax error.

b. Is there any compilation warning in your codes? (YES or NO) _NO_

   WARNING: there will be a 1 point deduction for every unique compiler warning. Please make sure that your submission does not have a compiler warning.

- How to run from the command line

  CCDSALG>**main**

Did you do the BONUS part (YES or NO) _YES_.

NOTE: Please do NOT submit a solution if it is not working correctly based on the exhaustive testing that you have done.

If you did the BONUS part, indicate below:

- How to compile from the command line

  CCDSALG> **gcc -Wall 30-BONUS.c -o 30-BONUS.exe**

- How to run from the command line

  CCDSALG>**30-BONUS**

2. How did you implement your Graph data structure? Did you implement it using adjacency matrix? adjacency list? Why? Explain briefly (at most 5 sentences).

For our graph data structure, we implemented it using an adjacency list because the program will work with undirected graphs. Since the maximum number of vertices is 20 and it is an undirected graph, the graph would be considered a sparse graph. Using an adjacency list would be faster and more efficient in terms of handling a graph data structure, especially if the graph is sparse.

3. How did you implement the DFS and BFS traversals? What other data structures (aside from the graph representation) did you use? Explain your algorithm succinctly.

a. DFS Traversal:

The DFS Traversal works by using a recursive helper dfs() that first starts with the starting node and then collects its unvisited neighbors in an array. It then utilizes a sort function by using the qsort() function in C that sorts the nodes' neighbors lexicographically. If the neighbor is not visited, the function calls itself recursively. The order of recursive calls is what determines the DFS of the graph.

b. BFS Traversal:

The BFS Traversal works by starting at a specified vertex and prints the vertex IDs in a lexicographical order. The algorithm also utilizes a queue data structure which stores each node and then collects its neighbors to check if it is visited or not. A sort algorithm is also used by using the qsort() function in c, which is a quick sort that sorts the neighbors lexicographically. Then the output is written to a text file.

4. Disclose **IN DETAIL** what is/are NOT working correctly in your solution. Please be honest about this. NON-DISCLOSURE will result in severe point deduction. Explain briefly the reason why your group could not make it work.

For example:

The following are NOT working (buggy):

a.

b.

We were not able to make them work because:

a.

b

5. What do you think is the level of difficulty of the project (was it easy, medium or hard)? Which part is hard (if you answer was "hard") and why?

Jack Morgan #1: Overall. the level of difficulty of this project is around medium-to-hard. The toughest part was to correctly implement and manage the dynamic adjacency lists and to write separate qsort.

Name #2: For me the level of difficulty of this project was around medium-to-hard as well. Properly implementing previous topics on how to handle and manage the vertices via arrays and edges via linked lists, and also how to correctly implement the dynamic adjacency list with 'malloc'.

Marcus Timothy #3: The level of difficulty of this project is somewhat hard because we had to figure out how to make our graph implementation efficient. We also had to make sure that the BFS and DFS traversals and the other outputs would work properly by implementing previous topics such as a queue and sort.

6. Fill out the table below.

REQUIREMENT	**AVE. OF SELF-ASSESSMENT**
1. Correctly produced Output Text File #1 (Set of Vertices & Edges)	10 (max. 10 points)
2. Correctly produced Output Text File #2 (Vertices With Degrees)	10 (max. 10 points)
3. Correctly produced Output Text File #3 (Adj. Matrix Visualization)	10 (max. 10 points)
4. Correctly produced Output Text File #4 (Adj. List Visualization)	10 (max. 10 points)
5. Correctly produced Output Text File #5 (BFS Traversal)	22.5 (max. 22.5 points)
6. Correctly produced Output Text File #6 (DFS Traversal)	22.5 (max. 22.5 points)
7. Documentation	10 (max. 10 points)
8. Compliance: deduct 1 point for every instruction not complied with (examples: incorrect output file names, extraneous contents in output files, etc.)	5 (max. 5 points)
9. Bonus :-)	10 (max. 10 points
TOTAL SCORE	110 over 100.

NOTE: The evaluation that the instructor will give is not necessarily going to be the same as what you indicated above. The self-assessment serves primarily as a guide on how your project will be evaluated.

***** THE END *****