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: Agunanne Henry SIGNATURE#1: [illegible] SECTION: S11

The following are my specific contributions to MCO2:

1. main.c implementation
2. Code debugging and fixing
3. Creation of test input files, testing of codes and outputs

Based on my honest estimate, and in consultation with my groupmates, I contributed approximately **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: Innocencio Chazwick SIGNATURE#2: [illegible] SECTION: S11

The following are my specific contributions to MCO2:

1. Code Creation
2. Code debugging and fixing
3. Code tester

Based on my honest estimate, and in consultation with my groupmates, I contributed approximately **33%** to the entire project.

NAME#3: Qinpei LU SIGNATURE#3: [illegible] SECTION: S11

The following are my specific contributions to MCO2:

1. Function codes
2. Code debugging and fixing
3. Code Testing

Based on my honest estimate, and in consultation with my groupmates, I contributed approximately **34%** 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 graph.c -o main
```

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 47-BONUS-MAIN.c graph.c -o 47-BONUS-MAIN
```

- How to run from the command line

```
CCDSALG>47-BONUS-MAIN
```

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).

We implemented our Graph data structure using an adjacency matrix. Specifically, we used a 2D array (adjMatrix) where each cell (adjMatrix[i][j]) indicates the presence of an edge between vertices (i and j). This approach was chosen because it allows for fast edge lookup (constant time) and is simple to code for graphs with a small, fixed number of vertices. While adjacency lists are more memory-efficient for sparse graphs, the adjacency matrix is suitable here due to the program's small maximum vertex limit and the need for straightforward operations like adding edges, checking connectivity, and generating matrix outputs. This design also simplifies file I/O and traversal implementations.

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:

DFS (Depth-First Search) was implemented using an explicit stack (an array of integers representing vertex indices) and a visited array to keep track of visited vertices. The algorithm starts by pushing the start vertex index onto the stack. While the stack is not empty, it pops a vertex, marks it as visited, and outputs it. Then, it pushes all unvisited neighbors of the current vertex onto the stack in reverse order to maintain traversal order. This process continues until the stack is empty.

b. BFS Traversal:

BFS (Breadth-First Search) was implemented using a queue (an array of integers representing vertex indices) and a visited array. The algorithm starts by enqueuing the start vertex index and marking it as visited. While the queue is not empty, it dequeues a vertex, outputs it, and enqueues all unvisited neighbors of the current vertex, marking each as visited. This continues until the queue is empty.

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?

Name #1: Agunanne Henry

Hard: The functions for the traversals and handling file I/O in the required format, preserving input order for adjacency lists

Name #2: Innocencio Chazwick

Hard: Understanding the concepts is difficult itself, and adding the part where we need to translate it to code made it harder, also having to work on different platforms, Windows and Mac, made it difficult to make them cohesive.

Name #3: Qinpei Lu

Medium

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 *****