CCDSALG MCO2 Specifications - Graph Implementation Term 3 AY 2024 - 2025

Ryan Austin Fernandez
2401 Taft Avenue
Manila, Philippines
ryan.fernandez@dlsu.edu.ph

ABSTRACT

This document outlines the specifications for a software project, specifically a graph operation implementation. The graph data structure should be implemented using manually implemented linked lists, and the operations should be implemented using stack, queue, and heap implementations coded from scratch.

Author Keywords

data structures, algorithm analysis, graphs

CCS Concepts

•**Theory of computation** → **Data structures design and analysis;**

INTRODUCTION

Let us apply what you learned about graphs. For this project, you will develop your graph implementation. Read, understand, and comply with the project specs described below.

This document is divided into seven sections: your objectives for this project, a description of the software you will develop, the list of deliverables, the groupwork policies, non-submission policies, honesty, and intellectual property policy, the grading rubrics, and the reference list.

OBJECTIVES

Your general objective in this study is to implement elementary graph operations using your implementation of linked lists, stacks, queues, and heaps.

The specific objectives are:

1. To build a working implementation of the linked list, stack, queue, and heap data structures in the C language;
2. To implement basic graph operations from scratch;
3. To practice modular programming for large or complex systems;
4. To implement a proper software testing procedure for the developed application; and
5. To write documentation of the developed application.

Design and implement your stack, queue, and heap data structures. Stacks will be used for depth-first search. Queues will be used for the breadth-first search. Heaps are optional for this project but may be used to speed up an implementation of Prim's algorithm and Djikstra's algorithm.

Practice modular programming. That is, compartmentalize the data structures and operations by storing the implementation codes in separate source code files. For example, the codes for the stack data structure are stored in stack.h and stack.c, while the codes for the queue data structure are stored in queue.h and queue.c. Do not use the goto statement.

Implement the algorithms for (a) adding vertices, (b) adding edges, (c) computing the degree of a vertex, (d) edge checking, (e) breadth-first search, (f) depth-first search, (g) connectivity check, (h) minimum spanning tree, and (i) shortest path algorithm. Again, practice good programming. Compartmentalize each algorithm implementation in separate source code files, for example, graph.c, traversal.c, etc.

Integrate the modules. Create the main module, which will include the other modules, and call the appropriate functions to achieve the task.

Test your solution. Create test cases and perform exhaustive testing. You should test each graph operation and special cases under each. You should test a full end-to-end interaction with at least 100 total commands or queries.

Hint: Use input and output redirection to minimize keyboard input. Encode your infix expression in a text file. Run your .exe file in the command line. For example:

```
CCDSALG> solution < input.txt > output.txt
```

where solution is the exe file, input.txt is a text file containing the test cases, and result.txt is the text file that will contain the corresponding output.

Document your solution. Give a brief description of how you implemented the data structures. For example, did you use arrays? Did you use a linked list (dynamic memory allocation)? Did you use a circular queue? Give a brief description of how you implemented the algorithms. Disclose what is not

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

working (for example, your solution does not handle minimum spanning trees).

Documentation Format

Abstract
Provide a summary of the paper, including a summary of the implementation details of the stack, queue, and heap data structures, the algorithms, the testing process, and results, and any limitations of the implementation (what features do not work?)

Stack, Queue, and Heap Implementation
Describe how your group implemented the stack, queue, and heap data structures from scratch.

Graph Implementation
Describe how you implemented the graph. Did you use adjacency matrices or adjacency lists? How did you implement these? What data structures did you use? How did you integrate these into the algorithm?

Graph Operation Implementation
For the operations, how did you implement them? What data structures did you use for each one?

Testing Process
How did you come up with your test cases? What types of test cases did you include? What were the results of your testing? What fixes did you have to implement after specific iterations of testing?

Limitations
What features of your application do not fully work? What challenges did you encounter in attempting to get these features to work? What attempts did you make in trying to make these work?

References
Provide a complete list of references in APA format. Failure to do so will result in a grade of 0.0 for this major course output.

Appendix A: Record of Contribution
Provide a record of contribution for each major activity for the case study.

1. Each row represents an activity, and each cell represents the percentage each member contributed to that activity. Each row's contributions must total 100.
2. The **TOTAL** row must be normalized to add up to 100.

Have each member sign the member list above the table, above their printed full name.

Example:

Group Members:

- P1: Sonic The Hedgehog
- P2: Miles "Tails" Prower
- P3: Knuckles the Echidna

Activity	P1	P2	P3
Stack Implementation	50.0	25.0	25.0
Queue Implementation	33.3	33.3	33.3
Heap Implementation	33.3	33.3	33.3
Basic Graph Implementation	25.0	50.0	25.0
Add Vertex	10.0	10.0	80.0
Add Edge	10.0	10.0	80.0
Degree Check	10.0	10.0	80.0
Edge Check	10.0	10.0	80.0
BFS	10.0	10.0	80.0
DFS	10.0	10.0	80.0
Path Check	10.0	10.0	80.0
MST	10.0	10.0	80.0
Shortest Path	10.0	10.0	80.0
Testing	10.0	10.0	80.0
Documentation	10.0	10.0	80.0
Raw Total	251.6	251.6	996.6
TOTAL	**16.78%**	**16.78%**	**66.44%**

SOFTWARE DESCRIPTION

Input and Output Description

The input is a series of commands, each corresponding to a particular graph operation or a query about the graph. You are processing an **undirected, weighted graph**. Set up a while loop and process commands until an 11 is inputted, as described below:

1. Add Vertex
   Format: `1 <name>`
   <name> is comprised of at most 256 characters only, including lowercase letters, uppercase letters, numbers, and underscore. If you are familiar with regular expressions, it follows `/^[A-Za-z0-9_]{1,256}$/`.

2. Add Edge
   Format: `2 <name1> <name2> <weight>`
   The names follow the same format as in command 1. Weight is a positive integer from 1 to 100 inclusive only.

3. Get Degree
   Format: `3 <name>`
   The name follows the same format as in command 1. Print the degree of <name> on its own line

4. Edge-Check
   Format: `4 <name1> <name2>`
   The names follow the same format as in command 1. Check if the two are directly connected. Print "1" if yes. Print "0" if no.

5. BFS
   Format: `5 <name>`
   The name follows the same format as in command 1. Print the BFS sequence for <name> as taught in class. Print one vertex name per line. If there is a choice between two adjacent vertices, go to the one that is lexicographically less.

6. DFS
   Format: `6 <name>`

The name follows the same format as in command 1. Print the DFS sequence for <name> as taught in class. Print one vertex name per line. If there is a choice between two adjacent vertices, go to the one that is lexicographically less.

7. Path-Check
   Format: `7 <name1> <name2>`
   The names follow the same format as in command 1. Check if the two are connected through some path in the graph. Print "1" if yes. Print "0" if no.

8. MST
   Format: `8`
   Print the definition of the MST as in command 10 (see below).

9. BONUS: Shortest Path
   Format: `9 <name1> <name2>` The names follow the same format as in command 1. Print the total weight of the shortest path and the nodes in the sequence, starting from <name1> until <name2>.

10. Print Graph
    Format: `10` print the graph definition as follows:

```
<name> = (V,E)
V = {<vertex_1>, <vertex_2>, ..., <vertex_n>}
E = {
    (<edge_u_1>, <edge_v_1>, <w_1>),
    (<edge_u_2>, <edge_v_2>, <w_2>),
    ...,
    (<edge_u_n>, <edge_v_n>, <w_n>),
}
```

    Since this is an undirected graph, print `(<edge_u_i>, <edge_v_i>, <w_i>)` if and only if `<edge_u_i>` is lexicographically before `<edge_v_i>`.

11. End program
    Format: `11`
    Terminate the loop.

Required Program Interaction

Figures 1 to 4 show an example of the required program interaction from start to finish in sequence.

Those in black font represent the user input. Those in blue font represent the output of your program Those in orange are not part of the interaction—they serve as explanations to help you understand what each line means.

The program should run in a loop when executed. The user inputs the command. The program will then output the specified response. To terminate the loop, the user inputs the command '11'.

The program should have a MINIMALIST interaction. This means that you should NOT have any prompts and that you should NOT print anything else except the required output. Non-compliance will result in point deductions.

DELIVERABLES

Submit via Canvas a ZIP file that contains the following:

- Source files for the stacks, queues, heaps, graphs, and main module
- Test case file named TESTCASE.TXT, which contains at least 100 sample command inputs.
- Corresponding RESULT.TXT that contains the result.
- Documentation named <GROUPNAME>.PDF. For example "Team Sonic.PDF"

Do NOT include any EXEcutable file in your submission.

The deadline for submission of all deliverables is Monday, July 28, 2025, at 5:00 PM.

It is highly recommended that you use Overleaf when typesetting your document. Your document should look like this document you are reading right now. The read-only template for this specifications document can be found here https://www.overleaf.com/read/crbcyyvhfcwy#0eee4d. You must make a copy of the Overleaf project to make edits.

Follow the file naming convention: CCDSALG_<section>_<lastnames of members>.zip, e.g., CCDSALG_S11_theHedgehogProwertheEchidna.zip

WORKING WITH GROUPMATES

You are to accomplish this project in collaboration with your fellow students. Form a group of at least two to at most three members. You may only select groupmates from the same section. Make sure that each group member makes approximately the same contribution to the project. Problems with groupmates must be discussed internally within the group and, if needed, with the teacher.

NON-SUBMISSION AND LATE SUBMISSION POLICY

If the project is not submitted three days after the due date, it will result in a score of 0 for this graded assessment.

Late submissions will receive a 10-point deduction per day (counted every second beyond 5:00 PM on July 28, 2025). No late submissions will be accepted after July 31, 2025, 5:00 PM.

HONESTY POLICY AND INTELLECTUAL PROPERTY RIGHTS

Honesty policy applies. You are encouraged to read and gather the information you need to implement the project. However, please note that you are NOT allowed to borrow and/or copy-and-paste – in whole or in part any existing related program code from the internet or other sources (such as printed materials like books or source codes by other people not online). You should develop your own code from scratch by yourselves, i.e., in cooperation with your groupmates.

Cheating or intellectual dishonesty is punishable with a final grade of 0.0 in the course.

```
1 Sonic                                                 // these commands add the vertices to the graph

1 Knuckles

1 Maria

1 Shadow

1 Tails

1 Eggman

2 Sonic Tails 10                                        // these commands add the edges to the graph

2 Knuckles Sonic 8

2 Knuckles Tails 7

2 Shadow Sonic 8

2 Maria Shadow 10

2 Eggman Shadow 1

10                                                      // this command prints the graph
G = (V,E)                                               // we see the printout here.
V = {Eggman, Knuckles, Maria, Shadow, Sonic, Tails}     // Vertices and edges are in lexicographical order
E = {
    (Eggman, Shadow, 1),
    (Knuckles, Sonic, 8),
    (Knuckles, Tails, 7),
    (Maria, Shadow, 10),
    (Shadow, Sonic, 8),
    (Sonic, Tails, 10),
}
```

Figure 1. Sample Interaction for MCO2 Commands 1, 2, and 10

```
3 Sonic                                                 // these are queries for vertex degrees
3                                                       // just print the degree alone in its own line

3 Knuckles
2                                                       // just print the degree alone in its own line

3 Tails
2                                                       // just print the degree alone in its own line

3 Eggman
1                                                       // just print the degree alone in its own line

3 Shadow
3                                                       // just print the degree alone in its own line

4 Sonic Tails                                           // these are queries for whether edges exist
1                                                       // edge exists so print '1' alone in its own line

4 Maria Tails
0                                                       // edge doesn't exist so print '0' alone in its own line
```

Figure 2. Sample Interaction for MCO2 Commands 3 and 4

RUBRIC FOR GRADING

The grading rubrics are shown in Table 1.

Please post any questions or concerns in the Canvas Discussion thread. Thank you for your cooperation.

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

[2] C.J.A. Delfinado. *Data structures and algorithms*. **https://books.google.com.ph/books?id=EcBUswEACAAJ**

[3] Mark Allen Weiss. 1995. *Data structures and algorithm analysis (2nd ed.)*. Benjamin-Cummings Publishing Co., Inc., USA.

```
5 Sonic                                                 // this is a query for BFS starting from Sonic
Sonic                                                   // print the BFS order, using lexicographical order to break ties
Knuckles
Shadow
Tails
Eggman
Maria

5 Eggman                                                // this is a query for BFS starting from Eggman
Eggman                                                  // print the BFS order, using lexicographical order to break ties
Shadow
Maria
Sonic
Knuckles
Tails

6 Sonic                                                 // this is a query for DFS starting from Sonic
Sonic                                                   // print the DFS order, using lexicographical order to break ties
Knuckles
Shadow
Eggman
Maria
Tails

6 Eggman                                                // this is a query for DFS starting from Eggman
Eggman                                                  // print the DFS order, using lexicographical order to break ties
Shadow
Maria
Sonic
Knuckles
Tails

7 Shadow Eggman                                        // this is a query to check if we can reach Eggman from Shadow
1                                                      // a path exists so print '1' alone in its own line

7 Eggman Sonic                                         // this is a query to check if we can reach Sonic from Eggman
1                                                      // a path exists so print '1' alone in its own line
```

Figure 3. Sample Interaction for MCO2 Commands 5, 6, and 7

```
8                                                       // this is a query for an MST computation
MST(G) = (V,E)                                          // we print the MST here
V = {Eggman, Knuckles, Maria, Shadow, Sonic, Tails}
E = {
    (Eggman, Shadow, 1),
    (Knuckles, Sonic, 8),
    (Knuckles, Tails, 7),
    (Maria, Shadow, 10),
    (Shadow, Sonic, 8),
}
Total Edge Weight: 34                                   // also print the total edge weight

9 Sonic Shadow                                          // this is a shortest path query from Sonic to Shadow
Sonic -> Shadow; Total edge cost = 8                    // we print the path as shown here, along w/ the total cost

9 Maria Knuckles                                        // this is a shortest path query from Maria to Knuckles
Maria -> Shadow -> Sonic -> Knuckles; Total edge cost = 26 // we print the path as shown here, along w/ the total cost

11                                                      // we stop the program with command '11'
```

Figure 4. Sample Interaction for MCO2 Commands 8, 9, and 11

Table 1. Grading Rubric for CCDSALG MCO2 AY2425 T3

CRITERIA	COMPLETE	INCOMPLETE	NO MARKS
Basic Graph Elements (Vertices and Edges) (3 points)	Graph representation (matrix or list) is correctly implemented and fully functional. (3 points)	Graph representation (matrix or list) is somewhat implemented and partially functional. (2 points)	Graph representation (matrix or list) is not implemented or functional. (0 points)
Stack Implementation (3 points)	Stacks are fully implemented and working. (3 points)	Stacks are somewhat implemented and working. (2 points)	Stacks are fully not implemented. (0 points)
Queue Implementation (3 points)	Queues are fully implemented and working. (3 points)	Queues are somewhat implemented and working. (2 points)	Queues are not fully implemented. (0 points)
Add Vertex (6 points)	Adding vertices is working properly. (6 points)	Adding vertices is working properly in some cases. (3 points)	Adding vertices is NOT working properly. (0 points)
Add Edge (10 points)	Adding edges is working properly. (10 points)	Adding edges is working properly in some cases. (5 points)	Adding edges is NOT working properly. (0 points)
Get Degree (6 points)	Degree computation is implemented and works correctly. (6 points)	Degree computation is implemented and works correctly sometimes. (3 points)	Degree computation is NOT implemented. (0 points)
Edge check (3 points)	Edge check is implemented and works correctly. (3 points)	Edge check is implemented and works correctly sometimes. (2 points)	Edge check is NOT implemented. (0 points)
BFS (12 points)	BFS is fully implemented and correctly working. (12 points)	BFS is implemented with slight errors in prioritizing adjacent vertices based on lexicographical order. (6 points)	BFS is NOT implemented. (0 points)
DFS (12 points)	DFS is fully implemented and correctly working. (12 points)	DFS is implemented with slight errors in prioritizing adjacent vertices based on lexicographical order. (6 points)	DFS is NOT implemented. (0 points)
Path-Check (12 points)	Path check is implemented and working. (12 points)	Path check is implemented and working sometimes. (6 points)	Path check is NOT implemented. (0 points)
MST (15 points)	MST is fully implemented (either Kruskal's or Prim's Algorithm can be used) and correctly working. (15 points)	MST is fully implemented (either Kruskal's or Prim's Algorithm can be used) with minor errors. (8 points)	MST is NOT implemented. (0 points)
Documentation (15 points)	document clearly explains how each feature is implemented. (15 points)	document explains how each feature is implemented but glosses over important details. (8 points)	NO document submission. (0 points)
BONUS: Heap Implementation (10 points)	Heaps are fully implemented and working. (10 points)	Heaps are somewhat implemented and working. (5 points)	Heaps are fully not implemented. (0 points)
BONUS: Shortest Path (10 points)	Shortest path algorithm is implemented using Dijkstra's Algorithm and working properly. (10 points)	Shortest path algorithm is implemented using Dijkstra's Algorithm and works sometimes. (5 points)	Shortest path is NOT implemented (0 points)
Maximum Total Points: 120/100			