CCDSALG Term 3, AY 2024 – 2025
Project 2 – Social Network (Undirected Graph Application)

Groupings : At most 3 members in a group (sign-up in Canvas)
Deadline : Refer to the Canvas submission page
Percentage : 20% of the Final Grade
Programming Language: C

For this project, you will:

a. design and implement an undirected graph representation of relationships among members of a Social Networking Site (SNS), and
b. implement graph traversal algorithms (BFS and DFS)

Watch and learn from the 1st three videos in this playlist Introduction to Graph Theory: A Computer Science Perspective - YouTube.

Refer to the following algorithm visualizations:
https://www.cs.usfca.edu/~galles/visualization/BFS.html
https://www.cs.usfca.edu/~galles/visualization/DFS.html

I. INPUT DATA

Figure 1 shows an example undirected graph for an SNS where a vertex contains the name of a member[1], and an edge indicates a link between two members. For example, Diana is a friend of Bruce, and vice-versa.

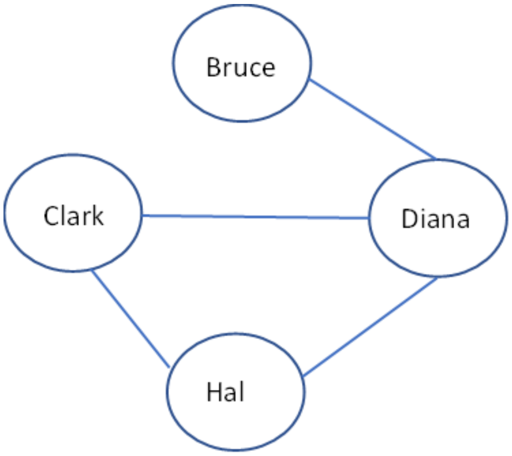

Figure 1: Example undirected graph

A named **text file** contains information about the input data, i.e., the undirected graph. Using Figure 1 as an example, the adjacency information about the graph are encoded and stored in a text file named as **G.TXT** as shown below:

[1] There are other attributes aside from name (such as gender, age, etc.) but we will not consider them to simplify the discussion and implementation details of the project.

G. TXT

```
4
Diana  Hal Bruce  Clark -1
Bruce  Diana  -1
Hal    Clark  Diana  -1
Clark  Hal Diana -1
```

The first line encodes the number of vertices in the graph. In the example above, the first line contains 4.

It is then followed by several lines where the number of lines is the same as the number of vertices. Each line indicates adjacency information similar to the information in an adjacency list. The first value in each line indicates the vertex ID (or label), followed by the vertex IDs of vertices that are adjacent to it. There is AT LEAST one space separating vertex IDs.

For example, in the third line we see: `Bruce  Diana  -1`

The first value in this line indicates the ID (or label) which is Bruce. Thereafter it is followed by vertex ID of Diana. This means that vertex Bruce and vertex Diana are adjacent to each other, i.e., there is an edge (Bruce, Diana). Please refer again to Figure 1 for the visualization. The last token in each line is a -1 which is a sentinel value that we use to indicate that there are no more adjacent vertices. (Note: As an analogy, in a linked list, a pointer value of NULL means that there is no more node, i.e., it is the end of the linked list. In the same manner, a value of -1 indicates the end of the list of vertices).

The last line shows: `Clark  Hal Diana -1`

Based on the explanation above, we have the following interpretation. Vertex Clark is adjacent to vertex Diana and vertex Hal. That is, there are two edges in the graph (Clark, Hal) and (Clark, Diana).

ASSUMPTIONS ON THE INPUT TEXT FILE and DATA

- The filename of the text file is also the name of the graph. For example, G.TXT stores the data for a graph named as G. As a matter of convention, we will use a single capital letter for the graph name. For example, graph A, graph B, …, graph Z with data stored into corresponding text files named as A.TXT, B.TXT, …, Z.TXT.
- The vertex ID (or label) is a string with a maximum length of 8 characters (excluding the null byte).
- Vertex IDs are case sensitive! Thus, “Diana” is different from “DIANA” and “DiaNa”.
- The maximum number of vertices in a graph is 20.
- The vertex IDs can be listed in any order in the input text file. Do NOT assume that they are sorted in any manner.
- For simplicity, we assume that the information stored in the input text file are correct. There is no need to perform any validation on the file contents.

II. OUTPUT

Produce SIX output text files with contents and file formats described below. Follow the specifications for the output text file format described for each output file. Other than the required contents, do NOT encode any other unnecessary characters. Your output text file with be compared automatically with an expected output text file. If you do NOT comply with the encoding/format specifications, your output will be marked as incorrect which will result into a lower score for your project grade.

Output Text File #1: set of vertices *V(G)* and set of edges *E(G)*

- The first line of output should contain the elements of *V(G)* enumerated in ascending order.
- The second line of output should contain the elements of *E(G)* enumerated also in ascending order by the start vertex. The start and end vertex for each edge should also be ascending order. For example, the edge should be encoded in the text file as (Bruce, Diana) not (Diana, Bruce).
- There should be NO space/blanks in the two lines of output.
- The filename of the output text file should be a concatenation of the filename of the input text file and the string "-SET". The file extension should be "TXT".

An example is shown below using "G.TXT" as input text file.

G-SET.TXT

```
V(G)={Bruce,Clark,Diana,Hal}
E(G)={(Bruce,Diana),(Clark,Diana),(Clark,Hal),(Diana,Hal)}
```

Output Text File #2: list of vertex IDs with the corresponding degree for each vertex.

- The vertices should be listed in alphabetical order.
- Encode at least one space between the vertex ID and its degree. The two columns need not be aligned.
- The filename of the output text file should be a concatenation of the filename of the input text file and the string "-DEGREE". The file extension should be "TXT". An example is shown below using "G.TXT" as input text file.

G-DEGREE.TXT

```
Bruce    1
Clark    2
Diana    3
Hal      2
```

Output Text File #3: adjacency List representation of the graph

- The list should be sequenced in the same manner as in the input text file.
- The links (pointers) should be encoded using "->" (*dash* and *greater than* characters with no space in between), and the NULL pointer value should be encoded using "\" (*backslash* character).
- The filename of the output text file should be a concatenation of the filename of the input text file and the string "-LIST". The file extension should be "TXT". An example is shown below using "G.TXT" as input text file.

G-LIST.TXT

```
Diana->Hal->Bruce->Clark->\
Bruce->Diana->\
Hal->Clark->Diana->\
Clark->Hal->Diana->\
```

Output Text File #4: Adjacency Matrix representation of the graph.

- The row and column labels should be the same as the vertex IDs and in the same sequence as they appeared in the input text file.
- You are free to decide on the spacing between tokens.
- The filename of the output text file should be a concatenation of the filename of the input text file and the string "-MATRIX". The file extension should be "TXT". An example is shown below using "G.TXT" as input text file.

G-MATRIX.TXT

```
        Diana Bruce Hal Clark
Diana   0     1     1   1
Bruce   1     0     0   0
Hal     1     0     0   1
Clark   1     0     1   0
```

Output Text File #5: a list of vertex IDs that correspond to a Breadth First Search (BFS) traversal sequence from a specified start vertex.

- As a rule, if there are several candidate vertices to visit (from the current vertex), we first visit the vertex with the lowest vertex ID.
- The filename of the output text file should be a concatenation of the filename of the input text file and the string "-BFS". The file extension should be "TXT". An example is shown below using "G.TXT" as input text file with "Clark" as starting vertex.

G-BFS.TXT

```
Clark Diana Hal Bruce
```

Output Text File #6: a list of vertex IDs that correspond to a Depth First Search (DFS) traversal sequence from a specified start vertex (note: same start vertex as in BFS traversal)

- As a rule, if there are several candidate vertices to visit (from the current vertex), we first visit the vertex with the lowest vertex ID.
- The filename of the output text file should be a concatenation of the filename of the input text file and the string "-DFS". The file extension should be "TXT". An example is shown below using "G.TXT" as input text file with "Clark" as starting vertex.

G-DFS.TXT

```
Clark Diana Bruce Hal
```

III. REQUIRED USER INTERACTION

There should be minimal program interaction, as shown in the sample runs below. Make sure to produce the same interaction and output text files. Do NOT display/print any unnecessary characters. The following steps should be performed when the program executes:

It will first ask the user to input the filename and extension of the input text file containing data about the undirected graph.

If the text file does not exist, the program should output "File <FILENAME.TXT> not found." error message and then terminate. There should NOT be any output text file created of size 0 byte.

If the text file exists, its contents will be processed, and thereafter, the program will produce Output Text Files #1 to #4. It will then ask the user to input the vertex ID to be used for both the BFS and DFS traversals. If the vertex ID does not exist, the program outputs "Vertex <ID> not found." and then terminates. If the vertex ID exists, Output Text Files #5 and #6 will be produced, and the program terminates.

Please refer to the examples below for the different scenarios that should be handled.

Example Run #1: G.TXT file exists, start vertex exists, and the six output files will be produced.

```
Input filename: G.TXT
Input start vertex for the traversal: Clark
```

Example Run #2 - G.TXT file exists, start vertex does NOT exist, only Output Text Files #1 to #4 will be produced. Recall that vertex IDs are case-sensitive; thus, "Clark" is a different string from "CLARK".

```
Input filename: G.TXT
Input start vertex for the traversal: CLARK
Vertex CLARK not found.
```

Example Run #3: G.TXT file exists, start vertex does NOT exist, only Output Text Files #1 to #4 will be produced.

```
Input filename: G.TXT
Input start vertex for the traversal: Logan
Vertex Logan not found.
```

Example Run #4: Z.TXT file does not exist; there will be no output text file.

```
Input filename: Z.TXT
File Z.TXT not found.
```

IV. TASKS

Task 1. Design your Undirected Graph data structure.

- Decide how you will represent and implement the graph data structure. Will you use an adjacency matrix? or will you use an adjacency list representation?
- Determine the modules and functions that you will need.

Task 2. Implement the modules including the algorithms for BFS and DFS traversals.

- Implement and compartmentalize the functions for reading the input file contents and storing the graph data into your graph data structure representation.
- Practice modular programming. That is, compartmentalize the data structure and operations by storing the implementation codes in separate source code files. For example, the codes for the graph data structure are stored in `graph.h` and `graph.c`.
- The implementation codes for graph traversals may be encoded inside in `graph.c` or into a separate file, for example, `traversal.c`.
- STRICTLY REQUIRED FUNCTIONS:

- There should be one function in charge of reading/processing the input file, and storing the vertex and edge information into your chosen representation (either an adjacency matrix or adjacency list). The function should have at least two parameters: the first should be filename and extension of the input file, and the second is the graph data structure. An example function call may be written as

  ```
  retval = ReadInputFile(strInputFileName, GDS, …)
  ```

  where **`retval`** is the variable that contains the return value of the **`ReadInputFile()`** function, **`strInputFileName`** is a string that contains for example "G.TXT", **`GDS`** is the parameter that corresponds to the Graph Data Structure (either adjacency matrix or adjacency list), and … are other parameters that you may need; for example, if you want to store the number of vertices and the number of edges. You may use different function name and parameter names. You may set the return type of the function to void.
- The input text file should be read ONLY ONCE during the program execution.
- There should be one function dedicated to producing a specific output text file. This means there should be six such functions. Each function should have at least two parameters: the first should be the filename and extension of the output file, and second is the graph data structure. You have the option to include the number of vertices, and edges if you deem them necessary. For example, the following function call will produce the contents of the Output Text File #1.

  ```
  retval = ProduceOutputFile1(strOutputFileName, GDS, numVertices, …)
  ```

  where **`strInputFileName`** is a string that contains for example "G-SET.TXT".
  The name of the input text file or a pointer to such a file should NOT be included anymore in the parameter list because the information about the graph, i.e., vertices and edges are already encoded in the GDS parameter. You may use different function names and parameter names. You may set the return type of the function to void.
- Use only standard C library functions.
- Do not use goto statement.

Task 3. Integrate the modules.

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

Task 4. Test & debug your solution exhaustively!

- Design your test cases. Create your own input text files with different number of vertices.
- Perform exhaustive testing. Make sure that you test different scenarios including:
  - a graph with only one vertex
  - a graph with multiple vertices but with zero number of edges
  - a graph with multiple vertices with some of the vertices having a degree of 0

Task 5. Document your solution.

- Give a brief description of how you implemented the algorithms.
- Disclose in clear details what is/are not working or what functionalities you were not able to implement.

VI. BONUS OPPORTUNITY 😊

This part is OPTIONAL -- you need not do it. If you do it, and your solution worked correctly during the demo, you will get 10 bonus points. If your solution did not work correctly during the demo, you will get a 0 (no point at all). In other words, it's ALL-OR-NOTHING. Let us not please waste each other's time. Please do NOT submit your solution if it it is not working correctly based on the exhaustive testing that you have done.

Create a separate C program with its own **main()** function named **GROUPNUMBER-BONUS.c** (for example 01-BONUS.c for group number 1) that will *check if graph **H** is a subgraph of graph **G**.* Data for these graphs are stored in two separate text files, for example **G.TXT** and **H.TXT**, as described in Section I. The program should:

1. Input/read the contents of the two files and store the graph data using your chosen graph representation scheme (adjacency matrix or adjacency list). For example, input G.TXT (as the first graph), and then input H.TXT (as the second graph). These means there are two adjacency matrices or adjacency lists, one for graph G and the other for graph H.
2. Check/determine if the second graph is a subgraph of the first graph.
3. Produce **Output Text File #7** formatted as follows:
    a. List of vertices in the second graph (ex. **H**) in alphabetical order marked with a '**+**' if it is also a vertex in the first graph (ex. G), or with a '**-**' if it is not in G.
    b. List of edges in the second graph in alphabetical order marked with '**+**' if it is also an edge in the first graph or with a '**-**' if it is not in G.
    c. Last line of output is either "____ is a subgraph of ____." or "____ is a NOT subgraph of ____." where the blanks are replaced with the names of the first and second graphs respectively.
4. The filename of the output text file should be a concatenation of the filenames of the first and second text files with a dash in between, and the string "-SUBGRAPH". The file extension should be "TXT".

Consider the following examples.

Example #1: let the first graph be **G.TXT** which is same as in Section I on page 2, and let the second graph **H.TXT** be as follows:

H. TXT

```
3
Hal    Diana  -1
Diana  Hal Bruce  -1
Bruce  Diana  -1
```

The corresponding output text file will be:

G-H-SUBGRAPH. TXT

```
Bruce +
Diana +
Hal +
(Bruce,Diana) +
(Diana,Hal) +
H is a subgraph of G.
```

NOTES: There should be no space in the edges. Don't forget to encode a period in the last line of output.

Exercise #1: Work this out on your own. What will be the output if the first graph is H.TXT and the second graph is G.TXT?

Example #2: let the first graph be **G.TXT**, and let the second graph **K.TXT** be as follows:

K. TXT

```
4
Hal    Jean Diana  -1
Diana  Hal Jean Bruce  -1
Bruce  Jean Diana  -1
Jean   Bruce Hal Diana -1
```

The corresponding output text file will be:

G-K-SUBGRAPH. TXT

```
Bruce +
Diana +
Hal +
Jean -
(Bruce,Diana) +
(Bruce,Jean) -
(Diana,Hal) +
(Diana,Jean)-
(Hal,Jean) -
K is a not subgraph of G.
```

NOTES: There should be no space in the edges. Don't forget to encode a period in the last line of output.

Exercise #2: What will be the output if the first graph is K.TXT and the second graph is G.TXT?
Exercise #3: What will be the output if the first graph is H.TXT and the second graph is K.TXT?
Exercise #4: What will be the output if the first graph is K.TXT and the second graph is H.TXT?

VI. DELIVERABLES

Submit via Canvas a ZIP file named GROUPNUMBER.ZIP (for example 01.zip for group number 1) which contains:

- Source files for the graph data structure, traversal algorithms, and other modules. Make sure to include sufficient and proper code documentation. Indicate the programmer and code tester names in the first few lines of your source codes. Describe what a function does and briefly describe lines of code that may not be easy to understand.
- Three example input text files with 5, 10 and 20 vertices respectively (which you used in testing).
- Corresponding output text files; 6 output text files for each input text file (total of 18 output text files).
- If you did the Bonus Opportunity, submit also the corresponding source files and sample input and output text files.
- Documentation named GROUPNUMBER.PDF (for example 01.PDF for group number 1). Use the accompanying template `MCO2-Documentation-Template.DOCX` file.

Do NOT include any EXEcutable file in your submission!!!

VII. WORKING WITH GROUPMATES

You are to accomplish this project in collaboration with your fellow students. Form a group of at least 2 to at most 3 members. The group may be composed of students from different sections. Make sure that each member of the group has approximately the same amount of contribution to the project. Problems with groupmates must be discussed internally within the group and, if needed, with the lecturer.

VIII. NON-SUBMISSION POLICY

Non-submission of the project will result in a score of 0 for this graded assessment.

IX. HONESTY POLICY AND INTELLECTUAL PROPERTY RIGHTS

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

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

X. RUBRIC FOR GRADING

REQUIREMENT	**MAX. POINT CONTRIBUTION**
1. Correctly produced Output Text File #1 (Set of Vertices & Edges)	10
2. Correctly produced Output Text File #2 (Vertices With Degrees)	10
3. Correctly produced Output Text File #3 (Adj. Matrix Visualization)	10
4. Correctly produced Output Text File #4 (Adj. List Visualization)	10
5. Correctly produced Output Text File #5 (BFS Traversal)	22.5
6. Correctly produced Output Text File #6 (DFS Traversal)	22.5
7. Documentation	10
8. Compliance: deduct 1 point for every instruction not complied with (examples: incorrect output file names, extraneous contents in output files, etc.)	5
9. Bonus :-) // note: ALL-OR-NOTHING	10
MAX. TOTAL	110/100

Question? Please post it in the Canvas Discussion thread. Thank you for your cooperation.

サルバドール・フロランテ