**CCDSALG Term 3, AY 2023 – 2024**

**Project 2 – Social Network (Undirected Graph Application)**

**Groupings** : At most 3 members in a group
**Deadline** : July 25, 2024 (H), before 8:00 A.M.
**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)

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.

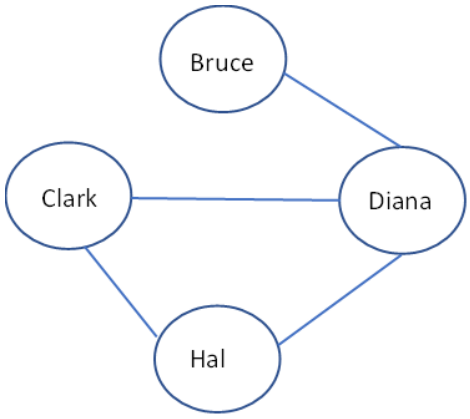


Figure 1: Example undirected graph

[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.

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 GRAPH.TXT as shown below:

**GRAPH. TXT**

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

For example, in the second line, we see **Bruce Diana -1**

The first value in this line indicates the ID (or label) of the first vertex, which is Bruce. Thereafter, it is followed by the 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 number 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. Similarly, a value of -1 indicates the end of the list of vertices).

The third line shows: **Clark Diana Hal -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, Diana) and (Clark, Hal). …*and so on…*

For simplicity, we assume that the information stored in the input text file are correct.

**II. OUTPUT**

Based on the contents of the input text file, the Social Network program will produce as output a text file named TRAVERSALS.TXT. It contains three sets of information, specifically:

1. A list of vertex IDs with the corresponding degree for each vertex. Using the graph in Figure 1 as an example, the output will be (shown in blue text color):
2. A list of vertex IDs that correspond to a BFS traversal from a specified start vertex.
3. A list of vertex IDs that correspond to a DFS traversal from a specified start vertex (note: same start vertex as in BFS traversal).

An example showing the contents of the output file TRAVERSAL.TXT (using GRAPHS.TXT as input) is given below:

**TRAVERSALS.TXT**

```
Bruce   1
Clark   2
Diana   3
Hal     2

Clark Diana Hal Bruce

Clark Diana Bruce Hal
```

The first four lines in blue color show the vertex IDs with their corresponding degrees. The list of vertex IDs should be in the same sequence as in the input text file. Based on the example output, we see that vertex Bruce has a degree of 1, and Diana has a degree of 3.

The line in green color shows the BFS traversal sequence. The starting vertex is Clark.

The line in purple color shows the DFS traversal sequence. The starting vertex is also Clark.

NOTES:

- Notice that the IDs follow the same upper case lower case mix as they originally appeared in the input text file (GRAPH.TXT).
- For the traversals: 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.

**III. REQUIRED USER INTERACTION**

There should be minimal program interaction, as shown in the sample runs below. The program will just ask the user to input the name of the input text file containing data about the undirected graph. If the text file exists, its contents will be processed. It will then ask the user to input the vertex ID to be used for the BFS and DFS traversals. If the vertex ID exists, the required output file named TRAVERSALS.TXT will be produced, and the program terminates. If the vertex ID does not exist, the program outputs "Vertex <ID> not found." and then terminates. If the text file does not exist, the program outputs "<FILENAME.TXT> not found." error message and then terminates.

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

Example Run #1 - GRAPH.TXT file exists, start vertex exists, and the output file TRAVERSALS. TXT will be produced.

```
Input filename: GRAPH.TXT
Input start vertex for the traversal: CLARK
```

Notice that CLARK (even if it is all caps) was found in the vertices of GRAPH.TXT. This means that the string search should be case-INsensitive.

Example Run #2 - GRAPH.TXT file exists, start vertex does NOT exist, there will be no output file.

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

Example Run #3 – XYZ.TXT file does not exist; there will be no output text file.

```
Input filename: XYZ.TXT
XYZ.TXT not found.
```

**IV. TASKS**

**Task 1. Design and implement your Undirected Graph data structure.**

- Decide how you will implement the graph data structure. Will you use an adjacency matrix? or will you use an adjacency list representation?
- Determine the functions that you will need to implement for your graph data structure.
- 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`**.
- Do not use goto statement.

**Task 2. Implement the algorithms for BFS and DFS traversals.**

- The implementation codes may be stored in **`graph.c`** or into a separate file, for example, **`traversal.c`**.

**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 your solution.**

- Design your test cases, and perform exhaustive testing.

**Task 5. Document your solution.**

- Give a brief description of how you implemented the algorithms. Disclose what is not working.

**V. DELIVERABLE**

Submit via Canvas a ZIP file named GROUPNAME.ZIP which contains:

- Source files for the graph data structure, traversal algorithms, and other modules.
- Input text file you used in testing your solution.
- Corresponding output file named TRAVERSALS. TXT.
- Documentation named GROUPNAME.PDF (see attached Word template).

Do NOT include any EXEcutable file in your submission.

**VI. 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.

**VII. NON-SUBMISSION POLICY**

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

**VIII. 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 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.**

**IX. RUBRIC FOR GRADING**

| REQUIREMENT | MAX. POINT CONTRIBUTION |
|---|---|
| 1. Correctly produced the list of vertices and their corresponding degrees | 15 points |
| 2. Correctly produced the BFS traversal | 35 points |
| 3. Correctly produced the DFS traversal | 35 points |
| 4. Documentation | 10 points |
| 5. Compliance (deduct 1 point for every instruction not complied with) | 5 points |
| Bonus opportunity #1: Drawing of the given graph | 10 points |
| Bonus opportunity #2: Drawing of the BFS tree | 10 points |

As indicated above, there are two opportunities to earn bonus points for this project if you were able to produce drawings of the graphs (using circles/boxes for the vertices, and line segments for the edges). Note that the bonus points will be awarded IF and ONLY IF the drawing produced is CORRECT. Partial bonus points will NOT be awarded if the drawing is incorrect.

Request: If you know in advance that your solution to the bonus portion is not working correctly, then please DO NOT submit it so as not to waste precious time for both of us.

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

*サルバドル・フロランテ*