CSE 6010
Workshop 2

Due Date: 11:59pm on Thursday, September 21
Submit 'adjmatrix.c', 'contrib.txt', and 'howthingswent.txt" to Gradescope through Canvas
48-hour grace period applies to all deadlines

There are several purposes for the workshop assignments:

- To provide some low-stakes programming practice in C where you can get real-time help from the TAs (if you attend the class in person);
- To cover small programming-related subjects that don't have a logical place among the course topics;
- To give you the opportunity to work together with other students, which can expose you to new ways of thinking about and writing C code; and
- To provide the opportunity for extra credit (if you submit more than 4 workshop assignments).

For this assignment, you may work in groups of 2-3 students. If you prefer, you may also choose to work independently.

If you choose to work as a team, you are expected to **work together for the full assignment**. A divide-and-conquer approach, where each team member works independently on some part of the assignment and the team only interacts when they combine their separate codes to submit, is not in the spirit of this assignment. If you don't want to work with anyone, you should submit independently.

The workshop is designed so that we expect most of you will be able to finish most of the assignments during the **75-minute class period**, possibly with a little extra time offline for cleanup and submission. Nevertheless, we know that different individuals and groups will have different levels of comfort/proficiency with C, and that programming does not always follow a timeline. Thus, it is not strictly required that every submission necessarily will include all topics. As part of your submission, please write a few sentences about how things went in a text file named **'howthingswent.txt'**. This is your opportunity to let us know if you got bogged down in a particular area and how you went about resolving any problems you may have encountered.

Contributions

We ask you to identify how each team member participated by filling in and including the contrib.txt file provided. For each problem, please assess the contribution level for each category according to the following scale.

- 3 = >50% (student did most of the work for that category – so there can only be a maximum of one "3" per category + problem combination)
- 2 = 20-50% (solid contribution)
- 1 = 1-20% (minor contribution)
- 0 = no contribution

The categories are:

- Ideas / planning
- Detailed code design
- Writing code / implementation
- Debugging / testing
- Documentation / comments

You can find the template 'contrib.txt' file on Canvas under Workshop 1. When submitting to Gradescope, make sure your filename is '**contrib.txt**'. **Submit this file even if you are a "group" with one member;** delete the space for contributions by other participants when not needed.

Your assignment: Write a code to read in graph information from a file and output information about the graph to the screen.

- Read in the file **testgraph.txt**, which specifies a directed graph using an adjacency-matrix representation. The first line gives the number of vertices in the graph, which for convenience we will refer to here as N. There are N more lines after the first line, each of which has N integer entries (either 0 or 1) separated by spaces, representing an adjacency matrix. (See Slides from Tuesday of Week 4 for more information on adjacency-matrix representations of graphs.) All values are integers. You may assume that the file is formatted as described; you do not need to handle cases where the formatting assumptions are violated.
- The graph information should be stored as a **1-dimensional** (1D) array of integer values that is **allocated dynamically** using malloc(), after reading in the number of vertices. Briefly, to map a theoretical 2D array a2d to a 1D array a1d, instead of declaring a2d[ROWS][COLS] and accessing an element as a2d[i][j], declare a1d[ROWS*COLS] and access an element as a1d[i*COLS+j]. (See Programming Practice – 9/5 for more on using a 1D array structure to represent a 2D array.) An advantage to a 1D structure is greater ease in passing the array to functions, which can be done as a basic pointer to an array, which effectively is equivalent to just passing the array.
- Write functions that calculate the **in-degree** and **out-degree** of each vertex, taking as arguments an integer pointer to the array representing the graph as well as the number of vertices. Loop over all vertices and output to the screen the in-degree and out-degree of each vertex on the same line. There should be one line for the in-degree and out-degree of vertex 0, another line for the in-degree and out-degree of vertex 1, etc., and the output line should look, for example, like
  3: 1, 2
  for a vertex of index 3 with in-degree 1 and out-degree 2.
- Write a function to find all **2-hop paths with distinct vertices** contained in the graph. If we designate an edge as an ordered pair of ***vertices*** (u, v) such that the edge goes from u to v, a 2-hop path can be thought of as an ordered pair of ***edges*** $(e_1,\ e_2) = \big((u, v), (v, w)\big)$. In other words, the "to" or "destination" vertex of the first edge serves as the "from" or "source"

vertex of the second edge, such that the path proceeds from vertex u to vertex w by way of vertex v. In this case, we add a further restriction that the vertices must be distinct. You may assume that the graph does not include any edges from a vertex back to itself, so that in practice this restriction means that u and w must be different vertices. Your 2-hop function should print to the screen the 3 vertices involved in each 2-hop path from the vertex corresponding to the loop index. For simplicity, each 2-hop path can be written on a new line (e.g., you do not need to include all 2-hop paths originating with the same vertex on the same line; you can treat all paths separately). For example,
2 1 0
would specify a 2-hop path from vertex 2 to vertex 0 via vertex 1.

- Make sure that you are not printing anything other than what was specified.
- Be sure to free all dynamically allocated memory before the program finishes. You may wish to test for memory leaks using **valgrind**, as explained in Workshop 1. (Checking for memory errors/leaks is a good habit to develop!)
- When submitting to Gradescope, make sure your code filename is '**adjmatrix.c**'. For this assignment, please keep all your code including function prototypes and definitions in this one file.

For reference, the in-degrees and out-degrees of each vertex for the graph provided are given below, followed by a table of all 2-hop paths for the graph provided, specified as the 3 vertices visited, in order.

Vertex	In-degree	Out-degree
0	0	3
1	2	1
2	3	2
3	2	2
4	2	1

2-hop paths
0 1 3
0 2 1
0 2 4
0 3 2
0 3 4
1 3 2
1 3 4
2 1 3
3 2 1
3 2 4
3 4 2
4 2 1

Group submissions

Submit your code file named **'adjmatrix.c'** along with **'contrib.txt'** and **'howthingswent.txt'** on **Gradescope** under **Workshop 2**. If you don't know how to submit on Gradescope as a group, you can follow this tutorial on YouTube. If you are working on the workshop as a group, and group members submit individually, we will consider this plagiarism and points would be deducted for this.