CSE 6010
Assignment 3
Connected Components

Initial Submission Due Date: 11:59pm on Thursday, October 5
Final Submission Due Date: 11:59pm on Thursday, October 12
Peer review and code reflections assignments due October 19 will be posted separately
Submit codes as described herein to Gradescope through Canvas
48-hour grace period applies to all deadlines

In this assignment, you will consider a network of social contacts being exposed to a transmissible disease. The network can be represented using a graph, where the vertices represent people and edges represent relationships. Furthermore, relationships are categorized as frequent and occasional contacts, represented by edge labels of 1 and 2, respectively. The disease can be transmitted to contacts of either type. We will consider all relationships to be reciprocal, resulting in an undirected graph.

Your task is to write a program in C to quantify an aspect of disease transmissibility by **calculating the number of connected components** of a graph. A connected component of a graph is a set of nodes such that a path exists from any node in the component to any other node in the component. A graph will consist of one or more connected components. Note that there are no paths between nodes that are in different connected components. Also, it is possible for a single node to be isolated, leading to a connected component consisting of just that node. Here you will count the number of connected components of a graph representing a social network consisting either of ***all contacts*** (both frequent and occasional) or of only ***frequent contacts*** (e.g., for the graph obtained by **excluding** all edges representing occasional contacts, which are those edges with label 2, and using only those edges with label 1). The idea is that having the individuals sorted into a larger number of groups by eliminating "occasional" contacts (e.g., by remaining at home) would limit the spread of the disease. Computing the number of connected components is a way to study the effect of such isolation.

To count the number of connected components of each graph, you will use a variant of **Depth-First Search** (DFS). Essentially, you may start from any node as the source and perform DFS, recording each visited node, until you backtrack to the source. The nodes that were visited, including the selected source, form a connected component. If any unvisited nodes remain, choose one of those as the new source and again perform DFS to visit all nodes of another connected component. Repeat until all nodes have been visited, keeping count of the number of connected components.

Specifications:

- Your code should ***read in the graph information from a file***. The first line of the file will specify the number of vertices (N); you should assume the vertices are represented as integers from 0 to N-1. Each following line of the file will specify an edge, listing the "from" vertex, the "to" vertex, and the edge label (1 or 2). A small test file will be provided; a larger file will be used to test your final submission.
- The graph must be represented in your code as an ***adjacency list*** and the memory to store it should be ***allocated dynamically***.
- Your code should ***define two structures:*** one to represent the ***overall graph*** (with its corresponding adjacency lists) and a second to represent each ***"to" vertex*** in the adjacency

list of a given "from" vertex. The "from" vertex is specified implicitly as the index in the vertex array; you do not need to represent it in any other way.

- The name of the file containing the graph information should be read as the ***first command-line argument***. You do not need to perform any validation and may assume that the user will specify a valid filename.
- The ***second command-line argument*** should be an integer specifying the maximum edge label to include when constructing the graph whose connected components will be calculated, either 1 or 2. If the argument is 1, only connections with a label of 1 should be included; if the argument is 2, connections with labels of 1 or 2 should be included. If a different number is entered, your program should not crash and instead should do something reasonable (e.g., you could default to one of the values, or choose the nearest value to the integer entered), but the specific behavior is not important. You do not need to perform any additional validation and may assume that the user will specify a valid integer.
- All ***dynamically allocated memory should be freed*** (consider using valgrind to test for memory leaks).

To receive full credit, your code must be well structured and documented so that it is easy to understand. Be sure to include comments that explain your code statements and structure.

A small test graph file is provided for you; we will check results for at least one additional graph.

Final submission: You should submit to Gradescope through Canvas the following files.

(1) your code (all .c and .h files); no specific filenames are required for your code but ***note the specification for your executable file below***.

(2) the **Makefile** you use to compile your program such that your executable file that will run your program is named '**connected**'. You may use the Makefile provided for previous assignments as a starting place.

(3) a series of 2 slides **saved as a PDF** and named **slides.pdf**, structured as follows:

- Slide 1: your name and a brief explanation of how you developed/structured your program. This should not be a recitation of material included in this assignment document but should focus on the main structural and functional elements of your program (e.g., the purpose of any loops you used, the purpose of any if statements you used to change the flow of the program, the purpose of any functions you created, etc.). You are limited to one slide.
- Slide 2: a discussion of why you believe your program functions properly, with reference to at least one test case.

Initial submission: Your initial submission does not need to include the slides or Makefile. It will not be graded for correctness or even tested but rather will be graded based on the appearance of a good-faith effort to complete the majority of the assignment.

Some hints:

- Start early!
- Identify a logical sequence for implementing the desired functionality. Then implement one piece at a time and verify each piece works properly before proceeding to implement the next.