ARIZONA STATE UNIVERSITY

CSE 310, SLN 30912 — **Data Structures and Algorithms** — Spring 2022
Instructor: Dr. Violet R. Syrotiuk

Project #3
There will be no milestone submission; the complete project is due Sunday, 04/24/2022

This project studies how graphs arising from real data sets evolve over time. The goals include:

1. Gaining experience representing real world data sets as directed graphs.
2. Implementing a number of classic graph algorithms.
3. Plotting and interpreting the trends in the graphs over time.

Note: This project **must** be completed **individually**. You **must** implement your solution in C/C++. Your code will be evaluated in Gradescope (Ubuntu 18.04 operating system, with version 7.5.0 of `gcc` and `g++`).

You **may not** use any external libraries to implement any part of this project, aside from the standard libraries for I/O, dynamic memory allocation, and string functions (*i.e.*, `stdlib.h`, `stdio.h`, `string.h`, and their equivalents in C++). In particular, you **may not** use the `vector` class. If you are in doubt about what libraries you may use, ask on `Ed Discussion`.

By convention, any program you write must exit with a **return value of zero** to signal normal completion; you may introduce non-zero return values to signal abnormal completion.

You **must** use a version control system as you develop your solution to this project, *e.g.*, GitHub or similar. Your code repository **must** be *private* to prevent anyone from plagiarizing your work.

The rest of this project description is organized as follows. §1 includes a description of the format of the graph data sets, the commands that your graph properties application must support, and the data that needs to be collected for the report. §2 summarizes the requirements for this project. Finally, §3 briefly describes the submission requirements for the project deadline; see Canvas for details.

1 Graph Evolution

In recent years, there has been considerable interest in graphs in technological, sociological, and scientific settings: computer networks (routers or autonomous systems connected together), networks of users exchanging e-mail or instant messages, citation networks and hyperlink networks, social networks (who trusts whom, who talks to whom, *etc.*), and countless more. Many properties of the graphs representing these networks are of interest, such as the in- and out-degree distribution, and the distances between pairs of nodes, to name only a few. Most empirical studies focus on static graphs, but because almost all real-world networks *evolve* over time by the addition and deletion of nodes and edges, in this project we are interested in the properties of the graphs over time.

1.1 Data Set Format

In this project, you will ultimately work with two real data sets. The first data set arises from citations of papers in the field of high-energy physics, while the second data set arises from citations in U.S. patents.

Each data set is made up of two text files, call them `edges.txt` and `dates.txt`. The file `edges.txt` contains the information about edges in the graph specified by their two endpoints. Specifically, if a paper p_i cites paper p_j (or a patent p_i cites a patent p_j), then the graph contains a directed edge from p_i to p_j and the file `edges.txt` contains a line with two tab-separated integers, p_i and p_j.

The file `dates.txt` includes information about the year that each paper was published, or the year that each patent was granted. Hence for each paper (or patent) p_i, the file `dates.txt` contains a line giving the vertex p_i followed by a year in the form `YYYY` (tab separated).

We will use the two files `edges.txt` and `dates.txt` to construct directed graphs, and examine their evolution over time.

1.1.1 Citation Data Sets

Two data sets are provided in the format described in §1.1:

1. The first data set, in the files `cit-HepPh.txt` and `cit-HepPh-dates.txt`, contains paper citation data from 22,824 distinct papers published in the field of high-energy physics for the years 1992–1999; it contains 201,300 citations. Specifically, `cit-HepPh.txt` contains the directed edges for the paper citation data, while `cit-HepPh-dates.txt` contains the year each paper was published.

2. The second data set, in the files `cit-Patents.txt` and `cit-Patents-dates.txt`, contains patent citation data from 3,774,768 distinct U.S. patents for the years 1963–1999; it contains 16,518,948 citations. Specifically, `cit-Patents.txt` contains the directed edges for the patent citation data, while `cit-Patents-dates.txt` contains the year that each patent was granted.

Note that the size of the graph (*i.e.*, the number of vertices and edges in the graph) constructed depends on the range of years specified in the command; this is described next.

A much smaller data set (on 11 vertices) used in the example in §1.2.1 will be provided for use in developing and testing your code for this project. It is described in a `README` file posted on Canvas.

1.2 Commands

In this project all input comes from files. Specifically, you will be given three command line parameters each of which is a file name: `edges.txt`, `dates.txt`, and `commands.txt`. That is, you will invoke your executable implementing this project, named `gp` for graph properties, as:

```
./gp edges.txt dates.txt commands.txt
```

where the file `edges.txt` contains the directed edges for the data set, `dates.txt` contains the year associated with each vertex, and `commands.txt` contains the commands to process on the data set.

Your `gp` application is to support the following commands. Angle brackets `<>` contain parameters to the command, while the other strings are literals. In all cases, you must echo the command before answering it. In the sample output, tabs are displayed as three spaces.

- `start-graph <YYYY`$_s$`> <YYYY`$_f$`>`, constructs a directed graph $G = (V, E)$ on all vertices in years starting from `YYYY`$_s$ and ending in `YYYY`$_f$, where `YYYY`$_s \leq$ `YYYY`$_f$.

  First, determine $n = |V|$, the number of vertices from the `dates.txt` file in the given range. Store the vertex labels in an array V of size n.

  You are to represent the graph using *adjacency lists*. Because the vertex labels are large integers, you need to be able to map a vertex label in V to an index of its adjacency list. Sort the array V in increasing order and use the index i of vertex label $V[i]$ as the index of its adjacency list. That is, $A[i]$ is the adjacency list of the vertex with label $V[i]$, for $0 \leq i < n$.

  Use the file `edges.txt` to decide which edges to insert into E; *both endpoints must be in V for the given year(s) for the edge to be inserted into E.* In order to guarantee that the results of other commands are unique, *order the edges in each adjacency list in increasing order by vertex label.*

  The output of the `start-graph` command is the number of vertices $n = |V|$, and the number of edges $m = |E|$ in the resulting graph G.

  A `start-graph` command is always paired with an `end-graph` command. All commands following the `start-graph` are to be applied to the graph G constructed for the given starting and ending years.

- `out-degree`, prints the out-degree distribution of the graph G. The out-degree of a vertex $V[i]$ is the number of out-edges from it to any other vertex in G; *i.e.*, count the number of vertices in the list $A[i]$. The out-degree can range from zero to n.

  The out-degree distribution counts, for each out-degree d, $0 \leq d \leq n$, the number of vertices with out-degree d. Output the average out-degree, and the out-degree distribution printing only non-zero degrees.

- `diameter`, use the Floyd-Warshall algorithm to compute the diameter of the graph G, *i.e.*, the length of the longest shortest path. The graph is unweighted, so use an edge weight of one (1) for all edges. The path length is then interpreted as a hop count. The Floyd-Warshall all-pairs shortest path algorithm is in §25.2 of our textbook.

  Output the diameter of G.

- `scc`, computes the number of *strongly connected components* (SCCs) in the graph G, and their size (*i.e.*, the number of vertices in each component). Use depth-first search (DFS) is commonly used to find the strongly connected components in a graph starting with the smallest vertex label and visiting vertices in numerical order; see §22.5 of our textbook.

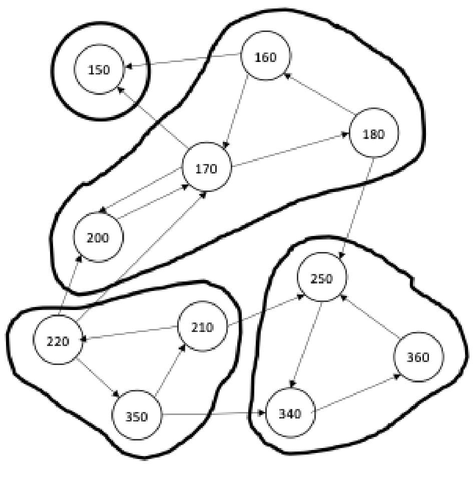
  Output the number of strongly connected components in G, in addition to their size. Only output non-zero sized components.

- `end-graph`, indicates the end of the sequence of commands on the graph G constructed in the `start-graph` command, and exit the program.

1.2.1 Sample Input and Output

Suppose that the following sequence of commands is to be executed on the 11-node graph in the figure below. The comments are not part of the input; they are only explanatory.

```
start-graph 2000 2002 // Construct graph G=(V,E) for the 3 years 2000-2002, output n=|V| and m=|E|
out-degree // Compute the average out-degree, and the out-degree distribution of G
diameter // Compute the diameter of G
scc // Compute the number of strongly connected components in G and their size
end-graph // exit the program
```



Example: Directed graph on 11 vertices.

Vertex identifiers after sorting.

	0	1	2	3	4	5	6	7	8	9	10
V	150	160	170	180	200	220	230	250	340	350	360

Adjacency list of vertex label $V[i]$ stored in $A[i]$ in increasing order.

$$\begin{tikzcd}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
A & \boxed{\text{NIL}} & \boxed{} \arrow[d] & \boxed{} \arrow[d] & \boxed{} \arrow[d] & \boxed{} \arrow[d] & \boxed{} \arrow[d] & \boxed{} \arrow[d] & \boxed{} \arrow[d] & \boxed{} \arrow[d] & \boxed{} \arrow[d] & \boxed{} \arrow[d] \\
 & & 0 \arrow[d] & 0 \arrow[d] & 1 \arrow[d] & 2 & 2 \arrow[d] & 5 \arrow[d] & 8 & 10 & 6 \arrow[d] & 7 \\
 & & 2 & 3 \arrow[d] & 7 & & 4 \arrow[d] & 7 & & & 8 & \\
 & & & 4 & & & 9 & & & & &
\end{tikzcd}$$

```
Command: start-graph 2000 2002

The graph G for the years 2000-2002 has:
   |V| = 11 vertices
   |E| = 18 edges

Command: out-degree

The graph G has average out-degree 18/11 = 1.63.
The out-degree distribution is:
   Out-degree : Number of vertices
   0 : 1
   1 : 4
   2 : 4
   3 : 2
```

The four strongly connected components of the example graph are circled in thicker lines. By definition, there are no directed paths from one SCC to another. Hence the longest shortest path occurs in the largest component and has length 3.

```
Command: diameter

The graph G has diameter 3.

Command: scc

The graph G has 4 strongly connected components:
   Size : Number
   1 : 1
   3 : 2
   4 : 1
```

1.3 Report Requirements for Project #3

Recall that we are trying to study trends in the graphs over time. Therefore, for the paper citation data set, and for the patent citation data set, provide a report containing the following:

1. Plot the number of edges m versus the number of nodes n for graphs for at least a sequence of four years. That is for the years 1992–1995 for the paper citation data set, *i.e.*, for 1992, 1992–1993, 1992–1994, and 1992–1995; if you can go past 1995, please do! You may want to use a logarithmic scale on your x- and/or y-axis.
2. Plot a histogram of the node out-degree distribution for at least a sequence of four years. Use, *e.g.*, Excel, gnuplot, *etc.*, to plot the distribution. If $n = |V|$ is large, you may need to try aggregating the node degree in your plot. You should have at least of 4 degree distributions.
3. Plot the average node out-degree for graphs for at least a sequence of four years.
4. Plot the diameter of the graphs for at least a sequence of four years.
5. Plot the number of strongly-connected components of the graphs for at least a sequence of four years.

Your report should try to summarize any trends that you see in your plots for the graph properties.

2 Program Requirements for Project #3

Write a C/C++ program, that compiles into an executable named `gp` for graph properties, and takes three command line parameters `edges.txt`, `dates.txt`, and `commands.txt` corresponding to input file names for the directed edge set, vertex dates, and commands to process, respectively; see §1.1.1 and §1.2.

Your program must use a modular design, in C/C++, *i.e.*, you must provide at least the following source code modules (each in a separate file):

- A main program, which coordinates all other modules,

- a module that provides any utility services,
- a module that implements the representation of G using adjacency lists and associated functions,
- a module that implements the Floyd-Warshall algorithm on G and associated functions, and
- a module that implements the depth-first search (DFS) on G and from it, the strongly connected components, and any associated functions.

3 Submission Deadline

This project is due before 11:59pm on Sunday, 04/22/2022. See §2 for the project requirements.

It is your responsibility to submit your project well before the time deadline!!! Do not expect the clock on your machine to be synchronized with the one on Gradescope! Late projects are not accepted.

An unlimited number of submissions are allowed. The last submission will be graded.

A detailed marking guide will be posted on Canvas soon. Sample input files that adhere to the format described in §1.1 and §1.2 are provided on Canvas; use Gradescope to test the correctness of your program.