ARIZONA STATE UNIVERSITY

# CSE 310, All Sections — **Data Structures and Algorithms** — Fall 2015

**Project #3**

Milestone due Friday, 11/20/2015; full project due Friday, 12/04/2015

In this project, we represent various real data sets, on social networks, autonomous systems, and the like, as undirected graphs and analyze some of their features.

**Note:** This project is to be completed individually. Your implementation *must* use C/C++ and your code *must* run on the Linux machine `general.asu.edu`. See §2 for full project requirements. No changes to these project requirements are permitted.

As before, use a version control system as you develop your solution to this project. Your code repository should be private to prevent anyone from plagiarizing your work.

## 1 Graphs

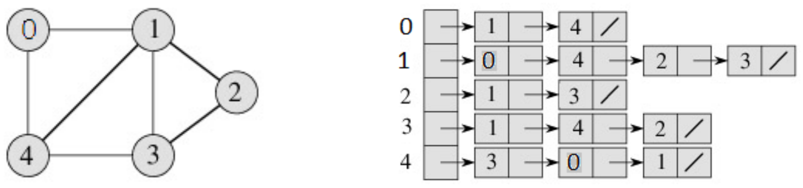
Graphs are a pervasive data structure, and algorithms for working with them are fundamental in computer science. There are a great number of interesting and practical computational problems defined in terms of graphs. The goal in this project is to implement a few of them. To have more chance to interpret the results, we use some real data sets in this project, for example, a social networking data set from Facebook, and a data set that represents how autonomous systems (ASs) in the internet exchange traffic flows with their peers, i.e., their neighbours.

### 1.1 Data Format and Data Sets

#### 1.1.1 Adjacency List Graph Representation

In this project, you **must** use an *adjacency list* to represent the graph. Such a representation of a graph $G = (V, E)$ consists of an array $A$ of $n = |V|$ pointers to linked lists, one for each vertex in $V$. For each $u \in V$, the adjacency list $A[u]$ contains all the vertices $v$ such that there is an edge $(u, v) \in E$. That is, $A[u]$ consists of linked list of all the vertices adjacent to $u$ in $G$. The vertices in each adjacency list are typically stored in an *arbitrary order*.

Figure 1(b) is an adjacency list representation of the undirected graph in Figure 1(a).



(a) An undirected graph. (b) Its adjacency list representation.

Figure 1: An undirected graph $G$ with 5 vertices and 7 edges and its adjacency list representation.

#### 1.1.2 Input Data Format

You must read the input data from standard input (`stdin`). The data is provided in a file as a convenience; you should redirect `stdin` from the file, i.e., no file operations are required, nor should they be used.

The format of the input is as follows:

1. First, the the graph $G = (V, E)$ is provided. The first line gives $n = |V|$, the number of vertices in the graph. The second line gives $m = |E|$, the number of edges in the graph. Each of the next $m$ lines gives an edge $e = (u, v) \in E$ specified by its two endpoints, i.e., $u, v \in \{0, 1, \ldots, n-1\}$. If there are self-loops, i.e., $u = v$, do not insert them into the graph.

   You should allocate an array of pointers of size $n$, and initialize each entry in the array to `NULL`. As each edge $e = (u, v)$ is processed, insertion it into the adjacency list for vertices $u$ and $v$ is performed. Recall that in an undirected graph, each edge appears twice.

2. Once the graph is initialized, commands to perform analyses of the graph then follow. That is, after the last edge in the graph, a line with $k$, the number of commands to execute is given. The subsequent $k$ lines each has one command. See §1.2 for details about the analyses required for each command.

### 1.1.3 Data Sets

Initially, two data sets are provided in the format described in §1.1.2:

1. The first data set, in the file `facebook.txt`, contains data from 4,039 individuals, i.e., each person is represented by a vertex in the graph. The edges in this graph represent interactions between people; this graph has 88,234 edges.

   Online social networking applications such as Facebook, Twitter, and Google+, among others, are very popular. Public and private corporations, as well as researchers, are interested to mine the data and determine what information may be extracted from it.

2. The second data set, in the file `as.txt`, contains 6,474 nodes in autonomous systems (ASs) in the internet. There are edges 13,895 in this graph.

   The graph of routers comprising the Internet can be organized into sub-graphs called Autonomous Systems (AS). Each AS exchanges traffic flows with some peers. We can construct a communication network of who-talks-to- whom from the BGP (Border Gateway Protocol) logs. Analyses of such a graph may yield a better understanding of, say, congestion in the internet.

A smaller data set, as well as more larger data sets will also be provided.

## 1.2 Graph Analyses

Your project must support the following commands on the graph:

- `degree-distribution` prints the degree distribution of the graph. This is presented as a histogram counting the number of nodes of degree $i$, $0 \leq i \leq n$. That is, the node degree is given on the $x$ axis, and the count is the $y$ axis; if $n = |V|$ is large, you may try aggregating the node degree.

- `shortest-path s t`, where `s` is the source vertex and `t` is the destination, prints the length of the shortest path in $G$ between `s` and `t`, and also prints the path taken to achieve it. Dijkstra's single-source shortest path algorithm **must** be implemented to compute the shortest path between `s` and `t`; see §24.3 of our textbook for a discussion of Dijkstra's algorithm.

- `diameter`, prints the diameter of the graph defined as the length of the longest shortest path. The Floyd-Warshall all-pairs shortest path algorithm must be used to compute the diameter; see §25.2 of our textbook for a discussion of the Floyd-Warshall algorithm.

- `components`, prints the number of connected components in the graph, and their size (i.e., the number of vertices in each component). One way to find the connected components is to run a breadth-first search, marking and counting nodes as the search proceeds. If any unmarked vertices exist, start a new breadth-first search starting at any unmarked node. Repeat until all nodes are marked. (Depth-first search also works.)

## 2 Program Requirements for Project #3

1. Write a C/C++ program that reads in a social network data set in the format as described in §1.1, represents it as an adjacency list, and then performs analyses of the data set as described in §1.2. **All memory management must be handled using only `malloc` and `free`, or `new` and `delete`. That is, you may not make use of any external libraries of any type for memory management!**
2. Provide a `Makefile` that compiles your program into an executable named `p3`. This executable must read from standard input directly, or from a file redirected from standard input (this should require no change to your program).

The data sets are provided on Blackboard; use them to test the correctness of your program.

## 3 Submission Instructions

All submissions are electronic. This project has two submission deadlines.

1. The milestone deadline is before midnight on Friday, 11/20/2015.
2. The complete project deadline is before midnight on Friday, 12/04/2015.

### 3.1 Requirements for Milestone Deadline

For the milestone deadline, your project must meet the following requirements

Submit electronically, before midnight on Friday, 11/20/2015 using the submission link on Blackboard for the Project #3 milestone, a zip[1] file named `yourFirstName-yourLastName.zip` containing the following:

**Project State (5%):** In a folder (directory) named `State` provide a brief report (.txt, .doc, .docx, .pdf) that addresses the following:

1. Describe any problems encountered in your implementation for this project milestone.
2. Describe any known bugs and/or incomplete command implementation for the project milestone.
3. Describe any significant collaboration with anyone (peers or otherwise) and/or clearly reference any external code bases used.

**Implementation (50%):** In a folder (directory) named `Code` provide:

1. Your well documented C/C++ source code implementing the commands `degree-distribution` and `components` as described in §1.2.
2. A `Makefile` that compiles your program to an executable named `p3` on the Linux machine `general.asu.edu`. Executing the command `make p3` in the `Code` directory must produce the executable `p3` also located in the `Code` directory.

**Correctness (45%):** The correctness of your program will be determined by running your program on the data sets `facebook.txt` and `as.txt`. In addition, some other data sets will be provided to you on Blackboard prior to the deadline for testing purposes. Of utmost importance in this project is your management of dynamic memory. As described in §2, your program must be able to run commands read from standard input directly, or from a file redirected from standard input. **You must not use file operations to read the input!**

The milestone is worth 40% of the total project grade.

[1]**Do not** use any other archiving program except `zip`.

### 3.2 Requirements for Complete Project Deadline

For the complete project deadline, your project must meet all the requirements in §2.

Submit electronically, before midnight on Friday, 12/04/2015 using the submission link on Blackboard for the complete Project #3, a zip[2] file named `yourFirstName-yourLastName.zip` containing the following:

**Project State (5%):** Follow the same instructions for Project State as in §3.1.

**Implementation (50%):** Your well documented C/C++ source code implementing all four commands described in §1.2. In addition, provide a `Makefile` as described in §3.1.

**Correctness (45%):** Follow the same instructions for Correctness as in §3.1.

## 4 Marking Guide

The project milestone is out of 100 marks.

**Project State (5%):** Summary of project state, use of a zip file, and directory structure required.

**Implementation (50%):** 40% for your code implementing `degree-distribution` and `components` commands as described in §1.2, including proper memory management; 10% for a correct `Makefile`.

**Correctness (45%):** 40% for correct output for the data sets `facebook.txt` and `as.txt` as a minimum, and lack of memory leaks; 5% for redirection from standard input.

The full project is out of 100 marks.

**Project State (5%):** Summary of project state, use of a zip file, and directory structure required.

**Implementation (50%):** 40% for your code implementing all four commands described in §1.2, including proper memory management; 10% for a correct `Makefile`.

**Correctness (45%):** 40% for correct output for the data sets `facebook.txt` and `as.txt` as a minimum, and lack of memory leaks; 5% for redirection from standard input.

[2]**Do not** use any other archiving program except `zip`.