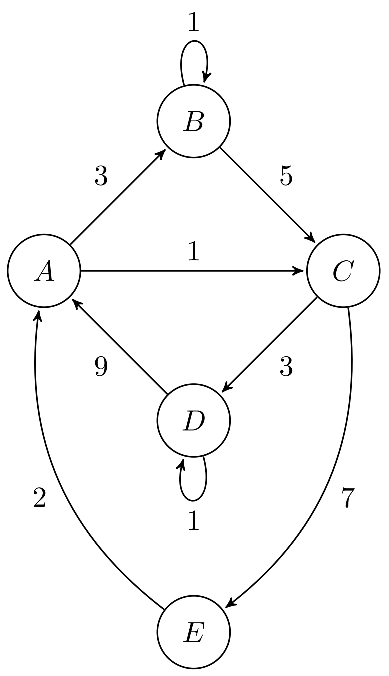
SAN DIEGO STATE UNIVERSITY CS310: Lab 3

The Graph

Due Date: 2019-NOV-20 Late Work: `Not Accepted`

1 Overview

The Graph Abstract Data Type (ADT) introduces a method for modeling the relationship between a set of vertices and the edges connecting them. It builds upon the idea introduced with the liked list where we store dynamic links to the the actual data in memory. No longer bound to simply one connection, however, the dynamic nature of the connections introduces the possibility of wildly diverse shapes. Any vertex in a graph may connect to paths that circle back upon itself.



```
A,B,3
A,C,1
B,B,1
B,C,5
C,D,3
C,E,7
D,D,1
D,A,9
E,A,2
```

In this programming assignment, students shall build an application which constructs a weighted, directed graph from a single CSV file. Students must build both the driver application, responsible for loading the data and displaying information about the graph, as well as the weighted-directed graph itself.

2 Requirements

This assignment incorporates several abstractions, and it necessitates several ADTs covered throughout the semester. Students must implement the *graph* and application identified in this assignment directly from scratch. Students need not implement underlying data structures, like the list, vector, map, or queue, however, and they may instead use the versions provided in the standard library [1].

Using the path to the input file specified as the command line argument, the application shall construct a weighted, directed graph. It shall extract the vertex and edge information from the comma-delineated input file as defined in section 2.2.1. With the graph constructed in the program's memory, the program shall then print several characteristics about the graph as defined below:

- The total number of vertices in the graph.
- List any vertices with exactly zero *inbound* edges.
- List any vertices with *self* edges.
- List any vertices with exactly zero *outbound* edges.
- List the edge with the heaviest weight.

2.1 Graph ADT

Students must design their individual weighted, directed graphs such that they support each of the following functions/methods:

[1] Or the Java Collections Framework.

Signature	Description
`boolean addEdge(String source, String dest, int cost)`	Creates a one-way connection from the source vertex to the destination vertex with the parameter cost. Returns `false` and does not change the graph if either vertex specified does not yet exist in the graph.
`boolean removeEdges(String source, String destination)`	Deletes all edges from the source node to the destination node regardless of cost. Returns `false` if either vertex name is invalid.
`void addVertex(String name)`	Inserts a vertex in the graph. If the vertex already exists in the graph, this method does nothing.
`boolean removeVertex(String name)`	Deletes the vertex, and all references to it, from the graph. It returns `false` if the vertex name does not appear in the graph.
`boolean contains(String name)`	Returns `true` if the specified vertex appears in the graph.
`boolean contains(String source, String dest)`	Returns `true` if the specified edge between the two vertex names exists in the graph.

Furthermore, no details specific to this assignment shall appear in the graph. That is, the graph should not process comma separated files or touch the file system. The graph itself shall *not* write to the terminal screen or `System.out`. The linked-lists and maps you use in the C++ standard library or Java Collections Framework do not send output to the screen, and neither should your data structure. All screen output shall appear in the driver.

Students may build their graphs using node objects, as discussed in lecture, or through a series of Maps/Dictionaries. The implementation specifics remain up the the individual developer, but each solution must support the required functionality.

2.2 Coding Style

Software quality strengthens program security, reliability, and maintainability. In general, all submitted software must:

- Adhere to a consistent naming convention and follow best practices for variable naming.
- Use private, or protected, internal methods to help clean the code.
- Remain free of commented out blocks of code.
- Use consistent formatting.
- The maximum column width is 80 characters.

The graders may elect to subtract points as they deem necessary for coding style and naming convention violations.

2.2.1 Input File Format

Each line in the file serves as either a single vertex entry (introducing a vertex) or a connection entry (identifying a link between two vertex points). Vertex entries contain only one string value indicating the name to associate with the vertex. The label is case-sensitive and may contain spaces and special characters. The comma shall not appear in a label name. Connection entries contain three values: The string name of the source vertex, the string name of the destination vertex, and the integer movement cost to traverse the edge. The program shall ignore lines failing to meet either of these formats.

The newline character denotes the end of an entry, so only one vertex or connection appears on any given line. This behavior differs from standard CSV. Vertex names need not appear before being introduced in a connection. Thus, a single entry may lead to the creation of two new vertices and one edge between them.

3 Delivery

At the start of class on the due date, students shall submit a legible, physical ASCII printout of their source code. Do **not** submit screen-shots of source code or the development environment. Do **not** copy the source into a word-processor. Simply print the source file, which is ASCII, directly. The 80

```
Basic
Basic,Grimer,0
Basic,Tapu Koko,0
Basic,Jirachi,0
Basic,Charmander,0
Stage 1
Basic,Stage 1,1
Grimer,Muk,1
Stage 1,Muk,0
Stage 1,Charmeleon,0
Charmander,Charmeleon,1
Stage 2
Stage 1,Stage 2,1
Stage 2,Charizard,0
Charmeleon,Charizard,1
```

Figure 1: An example of an arbitrary input file meeting the format.

character line limit in defined in the coding standard ensures that the source file prints as it appears on screen.

Additionally, students shall supply sample output after executing the program using the instructor supplied file: `Vietnam.csv`. On a Unix-based environment, one may redirect the program's output to a file using the `>` output redirection operator. For example, one might save the output of the `ls` command in a new file called `curContents.txt` with the following command:

```
healey@edoras:~$ ls > curContents.txt
```

4 Grading

In general, all submissions must meet every requirement in this specification.

Requirement	Points
Coding Style	5
Driver Application	10
Weighted Directed Graph	35

5 Other Notes

The instructor and teaching assistants remain available to assist students with questions that arise during the application's development and design. Due to high demand, however, students may not receive attention.