CSCI 241 Data Structures
Winter 2018
Assignment 3
Due: Monday, March 19

Submitting Your Work

Submit your Java program files in a zip folder via the **P 3** submission item on the course web site. You must submit your assignment by 11:59pm on Monday, March 19.

Your assignment will be evaluated on correct functionality and conformance to the coding standards described at the end of this assignment specification.

The Task

Your task is to write Java class files to read data from a file, construct a graph from that data, and then find the shortest path between any two points on that graph, as requested by the user of the program.

Data File

The provided data file `RoadData.csv` is a comma-separated-value file. Each line describes a road between two towns. Here are the first few lines in the file:

```
Adaminaby,Cooma,52
Adaminaby,Kiandra,38
Albury,Holbrook,65
```

The first line tells us that there is a road from Adaminaby to Cooma with distance 52km. Note that all roads are two-way, so that one line tells us that there is effectively also a road from Cooma to Adaminaby with distance 52km.

The name of the data file is to be specified in a command-line argument for the driver class. The source file for the driver class must be named **Assignment3.java**.

Graph Data Structure

From the data file, your Java software must establish a graph, with the towns as vertices and the roads between the towns as edges. The graph is a collection of vertices and each vertex has a collection of edges. You must choose appropriate data structures for the graph, vertices and edges to ensure efficiency of operations to be performed on those collections.

You are required to produce separate Java classes, each in a separate Java file for: the **graph**, for the **vertex**, and for the **edge** data types. You are free and encouraged to write other classes in addition to these.

Finding the Shortest Path

You are to implement Dijkstra's algorithm to find the shortest path between two specified towns. This algorithm is to be implemented to ensure time efficiency.

Once all the input data has been read and the graph constructed, your program must:

1. Prompt for the names of two towns,
2. Determine the shortest path between those towns,
3. Display the shortest path as a list of towns, showing for each town its distance from the start of the path. This list may be displayed in reverse order, as that is the natural order that can be determined from Dijkstra's algorithm, where we set the predecessor for each vertex.
4. Steps 1 through 3 are to be repeated until the user enters a town of length 0, in which case the program terminates.

Example:

```
> java Assignment3 RoadData.csv
89 towns added to graph
First town: Grafton
Second town: Bathurst
Bathurst (820)
Lithgow (763)
Windsor (672)
Singleton (511)
Muswellbrook (463)
Tamworth (306)
Bendemeer (264)
Uralla (218)
Armidale (195)
Ebor (106)
Grafton
First town:
>
```

Reading Input with Java

This assignment will require you to read input from a file and from the keyboard. Here's how you can do that.

When reading from a file, given a String variable filename, first create a BufferedReader object for the file.

```
BufferedReader in = null;
try {
     in = new BufferedReader(new FileReader(filename));
} catch (Exception e) {
     System.out.println("Cannot open file " + input);
     return 0;
}
```

Similarly, to read from the keyboard:

```
BufferedReader in = null;
try {
     in = new BufferedReader(new InputStreamReader(System.in));
} catch (Exception e) {
     System.out.println("Cannot accept input");
     return;
}
```

In either case, you can now read each line of input:

```
String line = "";
try {
     line = in.readLine();
} catch (Exception e) {
     System.out.println("I/O error: " + e);
}
```

When reading the data from the file, each line needs to be separated into its three parts: the two town names and the distance. Since the values on each line are comma-separated, you can use a `StringTokenizer` to get the parts of the line and use the `Integer.parseInt()` method to convert the distance string into an integer:

```
StringTokenizer tokenizer = new StringTokenizer(line, ",");
String town1 = tokenizer.nextToken();
String town2 = tokenizer.nextToken();
int distance = 0;
try {
     distance = Integer.parseInt(tokenizer.nextToken());
} catch (Exception e) {
     System.out.println("Data error: " + e);
}
```

Coding Standards

1. Use meaningful names for variables that give the reader a clue as to the purpose of the thing being named.

1. Use comments at the start of the program to identify the purpose of the program, the author and the date written.
2. Use comments at the start of each method to describe the purpose of the method, the purpose of each parameter to the method, and the return value from the method (if any).
3. Use comments at the start of each section of the program to explain what that part of the program does.
4. Use consistent indentation.