CS14 Winter 2016 Lab 4
Finding Short Cycles in a Directed Graph

November 28, 2016

DUE: Submit by iLearn by 1:00am Saturday, December 3, 2016
LATE SUBMISSIONS NOT ALLOWED FOR THIS ASSIGNMENT!

1 Introduction

The purpose of this assignment is to give you practice solving *shortest path problems* on a **directed** graph, G, with $|V|$ verticies in $V = \{0, 1, \ldots\}$ and $|E|$ directed edges in $E = \{a, b, c, \ldots\}$, where each edge $e \in E$ has a non-negative "weight", $W(e)$, and connects in *one direction only* from vertex $Start(e)$ to vertex $End(e)$.

Your task is to

(a) build the graph by reading a description from an input file,

(b) prompt the user to choose *three distinct vertices* from the graph, say v_i, v_j, v_k

(c) find the shortest directed cycle passing through all three of the selected vertices in either direction (i.e., the cycle could be either $v_i, \ldots, v_j, \ldots, v_k, \ldots, v_i, \ldots$ or $v_i, \ldots, v_k, \ldots, v_j, \ldots, v_i, \ldots$) and output a description of the vertices visited along the cycle

(d) check the shorted directed cycle reported in part (c) and output a message stating whether or not it contains any *repeated vertices*.

You may use the standard input and output (i.e., the terminal window) to ask for user input, and to display all output.

2 Building the graph

You will build your graph by reading the original data file from an early version of a famous computer game called the "Colossal Cave Adventure" or simply "Adventure", ***the story of which is described here in Wikipedia***.

This link gets you a copy of the `TEXT.TXT` data file you need for this assignment, the format of which I will describe in detail in this section. If you find yourself with too much free time on your hands over the holidays, ***this other link*** gets you a complete copy of the FORTRAN source code for the entire game. (This is what the code written by expert programmers looked like 40 years ago.)

2.1 Long description for each room in the cave

The data file `TEXT.TXT` is divided into a series of sections, each containing a different part of the description of the game. Section 1 of the data file lets you to create the *list of vertices* for the graph, represented in your program by a `vector` of `nodeType` objects. It begins with one line that consists of the digit 1 and nothing else, and ends with a *dummy room description* which says `-1,END`. In between these two boundary lines for section 1 is a series of *long descriptions* for each room (i.e., vertex) in the cave. Each of these lines starts with an integer that indicates the room number (i.e., vertex ID, where `1` is the first value) followed

by a comma, and the rest of the line is a text string that forms (part of) the description for indicated room number. These long descriptions usually span multiple lines, like this:

```
1,YOU ARE STANDING AT THE END OF A ROAD BEFORE A SMALL BRICK BUILDING.
1,AROUND YOU IS A FOREST.  A SMALL STREAM FLOWS OUT OF THE BUILDING AND
1,DOWN A GULLY.
2,YOU HAVE WALKED UP A HILL, STILL IN THE FOREST.  THE ROAD SLOPES BACK
2,DOWN THE OTHER SIDE OF THE HILL.  THERE IS A BUILDING IN THE DISTANCE.
3,YOU ARE INSIDE A BUILDING, A WELL HOUSE FOR A LARGE SPRING.
4,YOU ARE IN A VALLEY IN THE FOREST BESIDE A STREAM TUMBLING ALONG A
4,ROCKY BED.
5,YOU ARE IN OPEN FOREST, WITH A DEEP VALLEY TO ONE SIDE.
6,YOU ARE IN OPEN FOREST NEAR BOTH A VALLEY AND A ROAD.
```

Thus, the objects your program uses to represent each vertex should should include a field consisting of a `vector` of `string` objects to store these long descriptions.

2.2 Short description for selected rooms

Section 2 begins with one line that consists of the digit 2 and nothing else, and ends with a line that consists of `-1` and nothing else. In between these two boundary lines for section 2 are shorter, one-line descriptions for many rooms, like this:

```
1,YOU'RE AT END OF ROAD AGAIN.
2,YOU'RE AT HILL IN ROAD.
3,YOU'RE INSIDE BUILDING.
4,YOU'RE IN A VALLEY.
5,YOU'RE IN FOREST.
6,YOU'RE IN FOREST.
7,YOU'RE AT SLIT IN STREAMBED.
8,YOU'RE OUTSIDE GRATE.
9,YOU'RE BELOW THE GRATE.
10,YOU'RE IN COBBLE CRAWL.
11,YOU'RE IN DEBRIS ROOM.
13,YOU'RE IN BIRD CHAMBER.
```

Notice that every room with a multi-line description in section 1 has a shorter description in section 2. However, some of the rooms with single-line descriptions are missing from section 2 (like 12), while others receive an even-shorter desciption in section 2 (such as 3 and 5).

In the game, the computer always prints a room description when you entered it. It used the long description at your the first visit to the room, but switches to the short description (if any) when you return to a previously-visited room.

2.3 List of edges

Section 3 begins with one line that consists of the digit 3 and nothing else, and ends with a line that consists of `-1` and nothing else. In between these two boundary lines for section 3 is a sequence of lines consisting of exactly 25 comma-separated integer values, which represent a *starting room number*, an *ending room number*, one or more *action verbs*, and enough zeros to fill up the line, like this:

```
1,2,2,44,29,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1,3,3,12,19,43,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1,4,5,13,14,46,30,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1,5,6,45,43,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1,8,63,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2,1,2,12,7,43,45,30,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2,5,6,45,46,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

The data on one of these lines has the following interpretation: if the player's current location is *starting room number*, and he/she enters a command containing one of the specified *action verbs*, then they will move immediately to the *ending room number* (i.e., they cross the directed edge from starting room to ending room). Thus, for example, suppose the player is currently in room 1. If he/she enters one of the action verbs 2, 44 or 29, then the player moves to room 2. Similarly, if he/she enters one of the action verbs 3, 12, 19, or 43, then the player moves to room 3.

Use each of these data lines to create a *dircted edge* from the starting room number to the ending room number, and set the weight of that edge to the *maximum numerical value* of any of the specified action verbs for this edge. For example, the sample input shown above defines five outgoing edges exiting from node 1:

- edge $(1, 2)$ has weight 44
- edge $(1, 3)$ has weight 43
- edge $(1, 4)$ has weight 46
- edge $(1, 5)$ has weight 45
- edge $(1, 8)$ has weight 63

Note that in a small number of cases, the *ending room number* is an *impossibly-large value* compared to the total number of rooms defined in section 1 of the data file. These illegal room numbers represent situations in the game that require a more complex response than simply moving to an adjacent room. Here is **one case that you must include** in your program:

- ending room number `303` is only used in combination with starting room numbers `117` and `122`, where it triggers a special function call that controls travel across a bi-directional *troll bridge* (pun intended) between these two rooms. The player is not allowed to cross the bridge without paying the troll who guards it. **Include both of these edges in your graph, and assign weight 100 to each of them.**

Ending room number `301` is another special case, which is used to control travel through a small passage between rooms `99` and `100` which is too narrow for the player to pass through while carrying a specific treasure. However, for this assignment we can ignore ending room number `301` because there are other lines in section 3 that define a pair of ordinary edges connecting the same pair of rooms. Each of the remaining special cases in section 3 have a destination room number ≥ 500.

> **For this assignment, you may ignore every line in section 3 with destination room number ≥ 300 except 303, which must be included.**

2.4 Remaining sections of the input file

Ignore all remaining sections of the input file. They provide information to the game which is not relevant to this assignment.

3 Required data structures for representing the graph G

Construct a vector V of `room` objects to represent the *list of vertices* for the graph. For you own sanity, I strongly recommend that you include a *dummy vertex* as the 0-element of the vector, since the data file follows the rules of FORTRAN by assuming that 1 is the smallest subscript in the array of vertices. Each element of vector V should be a `struct` that includes the following fields

- `longStory` which stores the long description from section 1, as a vector of `string` values
- `shortStory` which stores the short description from section 2 as a single `string` value
- `Next` which stores the list of outgoing edge numbers as a vector of `int` values

- `Previous` which can be used for temporary storage of an `int` edge number by the shortest path algorithm (described below). Once the shortest path algorithm has finished calculating the minimum distance from a particular starting vertex to all other vertices, these `Previous` entries allow us to *travel backwards along the shortest path* from any location in the graph back to the starting point. This information allows you to construct a vector that is an ordered list of all vertices that belong to that shortest path.

Construct a vector E of `edge` objects to represent the *list of edges* for the graph. Each element of vector E should be a `struct` with fields

- `Start` which stores the starting room number as an `int` value
- `End` which stores the ending room number as an `int` value
- `W` which stores the edge weight as a `double` value

Note that many references (including Prof. Sheltons lecture slides) use a graph representation where each vertex object includes a vector of adjacent *vertices* (not edges), but trust me, a vector of *edge numbers* will make your life much easier!

4 Finding the shortest directed cycle through 3 specified vertices

Once your program has read the data file and created the data structures described above, you can undertake the main task of this assignment, which is to find the shortest *directed cycle* passing through 3 specified vertices.

4.1 Choosing the 3 vertices belonging to the cycle

If you've ever used the navigation application on your smart phone, then you understand the basic concept. If you want to go from one location to another (say from Riverside to the Los Angeles International Airport) the navigation app will determine what it thinks to be the best route and give you turn-by-turn instructions for following that path. However, if you want to make a stopover along the way, or you have specific preferences (i.e., "I hate driving on the 91 through Corona") then you can specify a few *way points* that forces the program to choose a route that passes through certain locations (i.e., your intended stopover) or bypasses locations you wish to avoid.

The goal of this assignment is to find a directed path through the graph that begins and ends and the same vertex and passes through two distinct *way points* along the way. You will prompt the user to enter the three vertex numbers using a dialog of writes to and reads from the standard input. For each of the three vertices, your program should ask the user to pick a vertex number between 1 and the maximum. Once the user has selected a vertex number, your program should respond by displaying the *long description* for that room, then asking the user whether they are satisfied with their choice or wish to make another selection.

Once the user has confirmed all 3 vertex selections, your program will attempt to find the shortest directed cycle that passes through all 3 vertices in any order. If your program cannot find such a cycle, it should report the failure and ask whether the user wants to try again. Your failure report should be a series of sentences, each of which states that a specific one of the six possible ordered pairs of vertices is not connected by a directed path, where each vertex is identified by its room number and short description, like this:

```
Sorry, no directed cycle passes through vertices 4, 12, and 99 because:
No path from 4 (YOU'RE IN A VALLEY) to 12 (YOU ARE IN AN AWKWARD SLOPING EAST/WEST CANYON).
No path from 4 (YOU'RE IN A VALLEY) to 99 (YOU'RE IN ALCOVE)
```

PLEASE NOTE that the above is just an example to show you the format of the failure report. I have no idea whether any of the statements shown in this example are actually true. Also, if you really want to impress your friends and family by your programming handiwork, I suggest that you spend some programming effort trying to perfect an algorithm for trimming the first few words from the beginning of each room description to improve readability. All the short descriptions in section 2 follow a very similar pattern, but unfortunately some of the one-line descriptions in section 1 are very different.

4.2 Dijkstra's weighted shortest-path algorithm

Dijkstra's algorithm is discussed at length in section 9.3.2 of the textbook, and in the lecture notes, so I will not spend time here explaining how it works. However, you must modify the algorithm to: (a) record the actual shortest paths from the starting location to the other vertices, and not just the total weight of all edges in the path, and (b) terminate as soon as it has found all three of the chosen vertices, rather than continuing until it has found every vertex in the entire graph.

To solve the *shortest directed cycle problem*, you must to run Dijkstra's *shortest weighted path algorithm* three times, using each of the three chosen vertices v_i, v_j and v_k as the starting point for the distance calculation. Since:

- Running the algorithm starting from v_i gives us $D(v_i, v_j)$ and $D(v_i, v_k)$ [assuming both paths exist].
- Running the algorithm starting from v_j gives us $D(v_j, v_i)$ and $D(v_j, v_k)$ [assuming both paths exist].
- Running the algorithm starting from v_k gives us $D(v_k, v_i)$ and $D(v_k, v_j)$ [assuming both paths exist].

Thus:

- If $D(v_i, v_j)$, $D(v_j, v_k)$, $D(v_k, v_i)$ all exist, then the "clockwise" cycle $v_i, \ldots, v_j, \ldots, v_k, \ldots, v_i, \ldots$ is possible.
- If (v_i, v_k), $D(v_k, v_j)$, $D(v_j, v_i)$ all exist, then the "counter-clockwise" cycle $v_i, \ldots, v_k, \ldots, v_j, \ldots, v_i, \ldots$ is possible.
- Therefore:
  1. If neither cycle is possible, then report failure and ask the user to choose a different set of vertices.
  2. If only one of the cycles is possible, then declare it to be the shortest directed cycle.
  3. Otherwise, calculate $(D(v_i, v_j) + D(v_j, v_k) + D(v_k, v_i)) - (D(v_i, v_k) + D(v_k, v_j) + D(v_j, v_i))$.
     - If the result is non-positive, then the shortest path is $v_i, \ldots, v_j, \ldots, v_k, \ldots, v_i, \ldots$
     - Otherwise, the shortest path is $v_i, \ldots, v_k, \ldots, v_j, \ldots, v_i, \ldots$