GEOG 490/590
GIS Programming
Lab 3: File I/O and Dijkstra's Shortest Path Algorithm

You will implement an algorithm developed by Edsger Dijkstra (1930-2002) to solve the network shortest path problem in this lab. In addition to the algorithm, there are several other things required to make a program that solves the shortest path problem. First, it needs an interface that a user can interact with the program so that the program is "user-friendly." Second, you need to create a "digital" network in the computer memory that the program can access. In this lab, you will first create the GUI of the program. Then, you will read a text file that defines the network into the computer memory and finally implement the Dijkstra algorithm. You have three lab sessions to complete this lab. The learning goals for you are to learn how to read from/write to text files and to use List and HashTable to implement the Dijkstra algorithm.

The Dijkstra algorithm, developed a half century ago, finds the shortest path between a vertex and every other vertex on a network graph by constructing and searching a shortest path tree. The algorithm is widely used in modern routing applications that you encounter in everyday life (e.g., on Google map).

The algorithm outlined on the wikipedia website has the following steps.

Let the node at which we are starting be called the **initial node**. Let the **distance of node Y** be the distance from the **initial node** to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Assign to every node a distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes as unvisited. Set initial node as current.
3. For current node, consider all its unvisited neighbors and calculate their *tentative* distance (from the initial node). For example, if current node (A) has distance of 6, and an edge connecting it with another node (B) is 2, the distance to B through A will be 6+2=8. If this distance is less than the previously recorded distance (infinity in the beginning, zero for the initial node), overwrite the distance.
4. When we are done considering all neighbors of the current node, mark it as visited. A visited node will not be checked ever again; its distance recorded now is final and minimal.
5. If all nodes have been visited, finish. Otherwise, set the unvisited node with the smallest distance (from the initial node, considering all nodes in graph) as the next "current node" and continue from step 3.

Figure 1 and Table 1 show the network on which you will find the shortest paths.

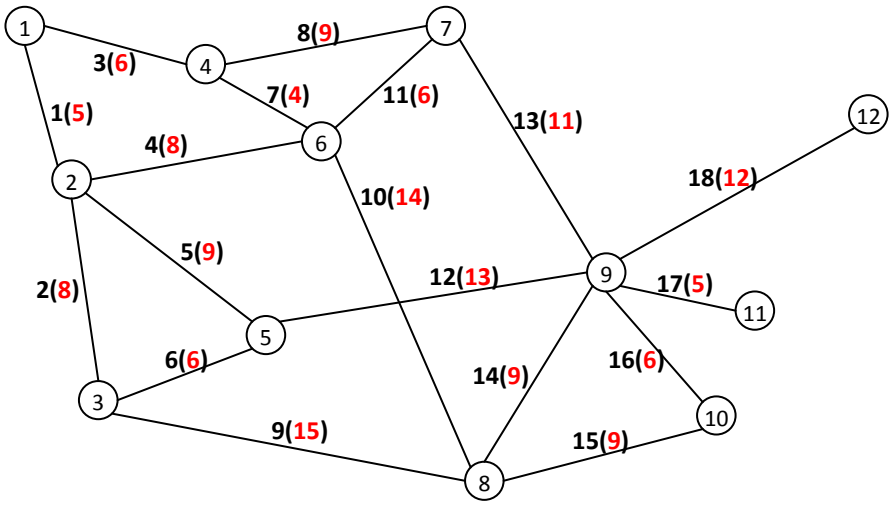

Figure 1. Lab network showing nodes with ID and links with ID and length (in parentheses).

Table 1. Table of edges that define the Lab 3 network.

Edge ID	Node 1	Node 2	Length
1	1	2	5
2	2	3	8
3	1	4	6
4	2	6	8
5	2	5	9
6	3	5	6
7	4	6	4
8	4	7	9
9	3	8	15
10	6	8	14
11	6	7	6
12	5	9	13
13	7	9	11
14	8	9	9
15	8	10	9
16	9	10	6
17	9	11	5
18	9	12	12

PART 1: CREATING THE GUI

1. You can either create this application as a standalone window form application or an ArcMap add-in. Please follow Lab 2 instructions if you want to develop the tool as an ArcMap add-in. Alternatively, select to create a "Windows Forms Application" when you create a new project. For this lab, your program code can be put in the form completely without adding additional modules or classes.

2. Create a form containing the following controls (see Figure 2 as an example):

- Four buttons:
  - Load Network button: to select an input network file
  - Solve button: to solve the shortest path problem
  - Save Result button: to save the route solution
  - Cancel button: to close the form and exit the application
- One textbox to display the name of the selected network file
- Two labels indicating the From and To network nodes
- Two comboboxes to list the nodes on the network
- One OpenFileDialog and one SaveFileDialog (you can find these controls under the Dialogs category)

Figure 2. Example GUI for Lab 3.

3. You need to do some tweaking to make the windows form look and behave normally. First, you can disable the Solve and Save Result buttons (set their Enabled property to False). They can be reactivated later after certain tasks are completed. The cancel button is to terminate the application. You can insert "Me.Close()" into the Click event of the button.

PART 2: FILE I/O

1. First you need to create a text file that defines the network depicted in Figure 1. We will use a simple edge definition format for this lab. Each edge on the network is recorded as having an ID, Node1, Node2, and Length (Table 1). Two-way traffic is allowed on all edges. Use an ASCII text editor (e.g., Notepad.exe) to create the file and save it with a .txt extension name. Please refer to Tuesday's lecture slides on how to use OpenFileDialog and SaveFileDialog controls to read from/write to a text file.

PART 3: CODE DEVELOPMENT

1. Thursday's lecture slides contain extensive instructions on how to implement the Dijkstra shortest path algorithm. Make sure you use VB .NET's debugging tools (e.g., break points, Console.WriteLine) at various stages of the development to make sure each block of code is doing what it's supposed to do. After you finish the coding and complete the debugging process, compile (i.e., build) the program. You can find the compiled executable (.exe) in the bin\Release folder of the project if you develop it as a Windows Forms Application.

Questions

Q1) How does the sample code implement an un-directed network (i.e., with bi-directional edges) with the networkEdges data structure? How does it relate to the implementation of the GetNeighbors function and the format of the input network file? (Hint: what would you do differently if there are one-way edges on the network?)

Q2) Complete the table below.

startNode	endNode	Length	Route (list the nodes)
1	12		
3	7		
10	1		
4	5		