Data Structure and Algorithm Assignment 3

Challenging Task

Name: Om Ashish Mishra

Registration Number: 16BCE0789

Slot: G2

Question:

In this assignment you are given a list of countries and its neighbors. The data file (countries.dat) is in the following format:

```
Country Name >> Neighbor Country Name#1[: XX km]; Neighbor Country Name#2[: XX km];
```

Here [:] means it is optional. In most cases, the country has a border, but there are a few cases that do not. Note that every line ends with a semicolon.

Example

--

```
arctic Territory (Australia); American Samoa (United States) >> ;
Ecuador >> Colombia: 590 km; Peru: 1,420 km;
```

--

Australian Territory (Australia). However, no border length is given this case. For such examples, you can assume that the border length is 0, however, do not forget to add the neighboring country to the graph.

In the second example, American Samoa (United States) has no neighbors, however, this co Land and American Samoa are territories of other countries (in this case France and US respectively), however, in this assignment we will treat them as separate countries.

The third example Ecuador >> Colombia: 590 km; Peru: 1,420 km; Ecuador, has two (2) neighbors. The border between Ecuador and Colombia is 590km, the border between Ecuador and Peru is 1420 km.

Your assignment will be to write a menu-driven console interface that allows the user to list the countries and check if paths between two countries exist (and if so, what is the shortest path in terms of # of countries visited [note that border length does not matter in the shortest path computation]). Additional functionalities must also be provided; see details in the following for additional functionalities.

Example of menu

========== Assignment 5 ==========

1. List all countries and bordering countries
2. Find shortest path
3. List all countries with X neighbors
4. Find countries with borders larger than X km
5. Exit

Details

1. List all countries and bordering countries

List all countries by integer ID [0 ~ N-1] and its neighboring countries.

2. Find shortest path

Find shortest path between two countries in terms of number of countries visited (you do not need to factor in the countries border in the shortest path computation). If a path does not exist, output that there is no path.

3. List all countries with X neighbors

List all countries with N neighbors, where the user provides the size of N.

4. Find countries with borders larger than X km

Find all countries with borders larger than X (where the user inputs X). Print the country and its neighbors.

Answer:

Algorithm of the program:

1. Declare all the required header files
2. Declare the respective prototype of the functions
3. Then we go for the menu driven programming in main function
   (1) List all countries and bordering countries
   (2) Find shortest path
   (3) List all countries with X neighbors
   (4) Find countries with borders larger than X km

   The user enters the choice1:

   Switch (choice1):

   1. Case 1:
      Insertion of the **root country** followed by the neighbor countries using linked list.
      Enter the root country: X
      Enter the number of neighbors: n
      If n is 0 then choice 1, if n>0 then choice 2 and if n<0 default condition
      Switch(n)
      a. Case 1: //First example type
         If(Only one root is present)
         Neighbor countries: 0
         Minimum distance covered is: 0
         break

      b. Case 2:
         Then are a *for loop i<-n do* //Third example type
         Enter the Neighbor's ID: V (i) // V represents the vertices at ith position
         Distance covered from root is: E (i) // E represents the edges at ith position

If the Neighbor of root country is 0 // Second example type

Then,

Neighbor countries: 0
Minimum distance covered is: 0
break

c. Default:
Wrong choice and the program returns to the menu driven options.
break

break

2. Case 2:
Now we apply the Dijkstra algorithm. Here we call the dijkstra's function.
1) Create a set ***sptSet* (shortest path tree set)** that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.
2) Assign a distance value to all vertices in the input graph. Initialize all distance values as **INFINITE**. Assign distance value as 0 for the source vertex so that it is picked first.
3) While *sptSet* doesn't include all vertices
a) Pick a vertex u which is not there in *sptSet* and has minimum distance value. Here u represents the initial point from which the next target will proceed to having the minimum distance.
b) Include **u** to *sptSet*.
c) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex **v**, if sum of distance value of u (from source) and weight of edge

u-v, is less than the distance value of v, then update the distance value of v ,i.e., **sptSet(v)=min{(old)sptSet(v), sptSet(u)+weight (u,v)}**

Note: Here weight means the distance between the countries.

```
dijkstra (int graph[V][V], int source)
{
   int dist[V];     // The output array.  dist[i] will hold the shortest
                    // distance from src to i

   bool sptSet[V(i)]; // sptSet[i] will true if vertex i is included in shortest
                      // path tree or shortest distance from source to i is finalized

   // Initialize all distances as INFINITE and stpSet[] as false
   for i = 0 to V do
     dist[i] = INT_MAX, sptSet[i] = false;

   // Distance of source vertex from itself is always 0
   dist[source] = 0;      //type example 1 and 2

   // Find shortest path for all vertices
   for count = 0 to V-1 do
   {
     // Pick the minimum distance vertex from the set of vertices not
     // yet processed. u is always equal to src in first iteration.
     u = minDistance(dist, sptSet);

     // Mark the picked vertex as processed
     sptSet[u] = true;

     // Update dist value of the adjacent vertices of the picked vertex.
     for  v = 0 to V do

       // Update dist[v] only if is not in sptSet, there is an edge from
       // u to v, and total weight of path from source to  v through u is
       // smaller than current value of dist[v]
       if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                              && dist[u]+graph[u][v] < dist[v])
```

```
            dist[v] = dist[u] + graph[u][v]
            sptSet(v)=min{(old)sptSet(v), sptSet(u)+weight (u,v)}
        }
      }
        // print the constructed distance graph
        Display(dist, V);
    }

    break
```

3. Case 3:

   Then we print the countries with neighbors.

   If n is equal to 0: // type 1 example

   Print: Neighbor countries: 0

   Minimum distance covered is: 0

   Else

   If n is greater than 0 and the value stored in the neighbor is also 0 then, // type 2 example

   Print: Neighbor countries: 0

   Minimum distance covered is: 0

   If n is greater than 0 and the value stored in the neighbor is not 0 then, // type 3 example

   Run a for loop to print the neighbor countries

   for i to n do

   Print: Neighbor countries' ID: V(i)

   From the Dijkstra's algorithm:

   Minimum distance covered is: Final sptSet(V) result.

   break

4. Case 4:

   The distance is given by the user: distance

In this the Dijkstra's algorithm will remain the same only one thing extra will be added that
If the distance given is by user is less than the graph representation path, then

We print the countries' and the neighbors' IDs.
break

5. Default:
   Wrong choice, please enter the right choice go back to menu driven.
   break

4. The functions used above are defined
5. Exit() : termination of the program.

THANK YOU