Implementation of Link-State Routing Protocol

Veereshwaran Rangasamy Chettiar Ramamoorthi

Synopsis

Routing Protocols

Routing protocols are used in the implementation of routing algorithms to facilitate the exchange of routing information between networks, allowing routers to build routing tables dynamically.

There are three classes of routing protocol:

1. Distance Vector Routing protocol
2. Link State Routing protocol
3. Hybrid Routing protocol

Link-state Routing Protocol

Link-state routing protocols, sometimes called *shortest path first*, are built around a well-known algorithm from graph theory, Dijkstra'a shortest path algorithm. Open Shortest Path First (OSPF) for IP is one such example of Link State Routing Protocol.

Dijkstra’s Algorithm

Dijkstra’s algorithm is used for finding the shortest path from one vertex to all other vertices in a weighted graph. It starts from the minimum weight edge from the source vertex and discovers new edges originating from the source vertex of the edge having minimum weight at each step.

1. Introduction

Dijkstra's algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1956 and published in 1959, is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms. For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path first is widely used in network routing protocols, most notably IS-IS and OSPF (Open Shortest Path First).

1.1. Dijkstra's Algorithm

The algorithm can be modified to solve many different problems. Dijkstra's algorithm solves the single-source shortest-path problem when all edges have non-negative weights. It is a greedy algorithm and starts at the source vertex, s, it grows a tree, T, that ultimately spans all vertices reachable from S. Vertices are added to T in order of distance i.e., first S, then the vertex closest to S, then the next closest, and so on.

Here is the algorithm for a graph G with vertices V = $\{v_1, \ldots v_n\}$ and edge weights w_{ij} for an edge connecting vertex v_i with vertex v_j. Let the source be v_1.

Initialize a set S = ∅. This set will keep track of all vertices that we have already computed the shortest distance to from the source.

Initialize an array D of estimates of shortest distances. D[1] = 0, while D[i] = ∞, for all other i. (This says that our estimate from v_1 to v_1 is 0, and all of our other estimates from v_1 are infinity.)

For the dijkstra's algorithm to work it should be directed- weighted graph and the edges should be non-negative. If the edges are negative then the actual shortest path cannot be obtained. At the k th round, there will be a set called Frontier of k vertices that will consist of the vertices closest to the source and the vertices that lie outside frontier are computed and put into New Frontier. The shortest distance obtained is maintained in sDist[w]. It holds the estimate of the distance from s to w. Dijkstra's algorithm finds the next closest vertex by maintaining the New Frontier vertices in a priority-min queue. The algorithm works by keeping the shortest distance of vertex v from the source in an array, sDist. The shortest distance of the source to itself is zero. sDist for all other vertices is set to infinity to indicate that those vertices are not yet processed. After the algorithm finishes the processing of the vertices sDist will have the shortest distance of vertex w to s.

While S != V do the following:

1) Find the vertex (not is S) that corresponds to the

   minimal estimate of shortest distances in array D.

2) Add this vertex, v_i into S.

3) Recompute all estimates based on edges emanating

   from v. In particular, for each edge from v, compute

   $D[i]+w_{ij}$. If this quantity is less than D[j], then set

   $D[j] = D[i]+w_{ij}$

1.2. Dijkstra's Pseudocode

ShortestPath(*G*, *v*) // a little miss leading since the output is only the distance
input: A simple undirected weighted graph *G*
with non negative edge weights and a start vertex, *v*.
output: *D*(*u*) the distance *u* is from *v*.

Initialize *D*(*v*) = 0 and *D*(*u*) = ∞ for *u* != *v*
Initialize priority queue *Q* of vertices using *D* as key.
while *Q* is not empty **do**
u = *Q*.removeMin()
for each vertex *z* adjacent to *u* **and** in *Q* **do**
if *D*(*u*) + *w*((*u*, *z*)) < *D*(*z*) **then**
D(*z*) = *D*(*u*) + *w*((*u*, *z*))
update *z* in *Q*
return *D*

2. Design of the program

The program is implemented in python to read the given routing file, compute the routing table for each router and find the shortest path between any source and any destination.

The program flow is as follows:

1. Display the menu to the user as follows:
   1. Create a network topology
   2. Build a Connection Table
   3. Shortest Path to destination router
   4. Modify a topology
   5. Exit
2. If the user selects the options as "1. Create a network topology", prompt the user to enter the input file name.
   - I) Read all the lines from the file and put it into a list.
   - II) Perform various check on the file to validate the contents of the file such as
     a. verification of presence of a square matrix
     b. validation of valid weights in integers
   - III) Once verification is complete, values are read and put into ***dictionary*** having row number as key, and value as a ***sub-dictionary*** of column number as key and value as list of distance and row number (represents as source router ***(S_c)***).
     {Row1:{Column1:[distance1:Row1], Column2:[distance2:Row1],..},
     Row2:{Column1:[distance1:Row2], Column2:[distance2:Row2],..}}For instance:
     {' 4': {' 4': [0, ' 4'], ' 5': [1, ' 4'], ' 6': [-1, ' 4'],….}, ' 5': {' 4': [1, ' 5'], ' 5': [0, ' 5'], ' 6': [-1, ' 5'], ' 7': [-1, ' 5'],….}}

3. If the user selects the option as “2. Build a Connection Table”, prompt the user to enter the source router ***(S_r)***.
    a. Check whether the user gives the valid router number, if the user enter an invalid router, display error message and prompt the user with the menu again
    b. Once the user enters a valid router number, fetch the ***sub-dictionary*** of the router number entered and store it as ***initial route table (R_t)*** of that source router.
    c. Create an empty list called ***visited*** to keep track of the routers visited.
    d. Create a list called unvisited and copy all the routers provided in the route table to mark those are ***unvisited***.
    e. Take the router that is unvisited and having a minimum distance from the source router (using a sorting method), mark it as source router ***(S_c)*** and start traversing the adjacent routers, and compute the distance from the source router ***(S_r)*** to the corresponding adjacent routers.
    f. If the adjacent routers can be reached with a shorter distance when compared to distance mentioned for the respective adjacent router in the initial ***route table,*** replace the minimal distance in the initial route table as well the source router as the current source router***(S_c)*** for the respective adjacent router.
    g. Else maintain the same distance and source router***(S_c)*** values and check for next minimal distance router.
    h. Mark the visited routers as “visited” by appending the router into the visited list.
    i. For each router that is not yet visited, the step 3.e, 3.f, 3.g, 3.h continues until it updates the route table with all shortest distances.
    j. Compute the path of source router to destination router by doing a recursive call and on the source router***(S_c)*** of the destination router until it reached the source router***(S_r)***
    k. Store all these paths in a dictionary ***(D_p)*** having the source as key and list of minimum distance and path as value.
    l. Now display the connection table having all the destination routers and the path of those destination routers from the source routers.
    m. Now again, prompt the user with same menu.
4. If the user selects as “3.Shortest Path to destination router”, ask the user to enter a specific destination router.
    a. Check whether the user enters a valid destination router
    b. Fetch the corresponding value of the destination router from the dictionary ***(D_p)*** and display the path as well as the minimum distance
5. If the user select as “4.Modify a topology”, prompt the user to enter the router to be removed.
    a. Check whether the user enters a valid router to be removed.
    b. Once the router to be removed is found to be valid router, create a temporary dictionary similar to that of ***R_t*** and copy all the values from ***R_t*** excluding the router to be removed.
    c. Once again redo all the steps done in the step 3 to recompute a new connection table with all the available routers.

d. Display the same menu again to the user.

6. If the user selects as “5. Exit”, exit the application

3. Test Report for sample network file

Input: new.txt

0	1	-1	6	-1	-1	3	-1	-1	-1
1	0	3	-1	2	-1	1	-1	-1	-1
-1	3	0	-1	2	5	-1	-1	1	-1
6	-1	-1	0	-1	-1	7	-1	-1	-1
-1	2	2	-1	0	1	-1	1	-1	-1
-1	-1	5	-1	1	0	-1	-1	9	-1
3	1	-1	7	-1	-1	0	9	-1	-1
-1	-1	-1	-1	1	-1	9	0	2	1
-1	-1	1	-1	-1	9	-1	2	0	2

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command1

Input original network topology matix data file:new.txt

Review original topology matrix:

0	1	-1	6	-1	-1	3	-1	-1	-1

1	0	3	-1	2	-1	1	-1	-1	-1
-1	3	0	-1	2	5	-1	-1	1	-1
6	-1	-1	0	-1	-1	7	-1	-1	-1
-1	2	2	-1	0	1	-1	1	-1	-1
-1	-1	5	-1	1	0	-1	-1	9	-1
3	1	-1	7	-1	-1	0	9	-1	-1
-1	-1	-1	-1	1	-1	9	0	2	1
-1	-1	1	-1	-1	9	-1	2	0	2
-1	-1	-1	-1	-1	-1	-1	1	2	0

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command2

Enter the source router:2

Router2 Connection Table

Destination	Interface
0	2->1->0
1	2->1

3 2->1->0->3

4 2->4

5 2->4->5

6 2->1->6

7 2->8->7

8 2->8

9 2->8->9

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command4

Enter the router to remove4

Router2 Connection Table

Destination Interface

==========

0 2->1->0

1 2->1

3 2->1->0->3

5 2->5

6 2->1->6

7 2->8->7

8 2->8

9 2->8->9

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command4

Enter the router to remove8

Router2 Connection Table

Destination Interface

==========

0 2->1->0

1 2->1

3 2->1->0->3

5 2->5

6 2->1->6

7 2->1->6->7

9 2->1->6->7->9

================:Menu:================

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

=====================================

Enter a command4

Enter the router to remove6

Router2 Connection Table

Destination	Interface
0	2->1->0
1	2->1
3	2->1->0->3
5	2->5

================:Menu:================

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command4

Enter the router to remove1

Router2 Connection Table

Destination Interface

==========

5 2->5

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command3

Enter a destination router: 5

Distance is 5 and Path is 2->5

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command1

Input original network topology matix data file:new.txt

Review original topology matrix:

0	1	-1	6	-1	-1	3	-1	-1	-1
1	0	3	-1	2	-1	1	-1	-1	-1
-1	3	0	-1	2	5	-1	-1	1	-1
6	-1	-1	0	-1	-1	7	-1	-1	-1
-1	2	2	-1	0	1	-1	1	-1	-1
-1	-1	5	-1	1	0	-1	-1	9	-1
3	1	-1	7	-1	-1	0	9	-1	-1
-1	-1	-1	-1	1	-1	9	0	2	1
-1	-1	1	-1	-1	9	-1	2	0	2
-1	-1	-1	-1	-1	-1	-1	1	2	0

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command2

Enter the source router:8

Router8 Connection Table

Destination	Interface
0	8->2->1->0
1	8->2->1
2	8->2
3	8->2->1->0->3
4	8->2->4
5	8->2->4->5
6	8->2->1->6
7	8->7
9	8->9

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command4

Enter the router to remove2

Router8 Connection Table

Destination	Interface
0	8->7->4->1->0
1	8->7->4->1
3	8->7->4->1->0->3
4	8->7->4
5	8->7->4->5
6	8->7->4->1->6
7	8->7
9	8->9

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command4

Enter the router to remove7

Router8 Connection Table

Destination	Interface
0	8->5->4->1->0
1	8->5->4->1
3	8->5->4->1->0->3
4	8->5->4
5	8->5
6	8->5->4->1->6
9	8->9

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command4

Enter the router to remove6

Router8 Connection Table

Destination	Interface
0	8->5->4->1->0

1 8->5->4->1

3 8->5->4->1->0->3

4 8->5->4

5 8->5

9 8->9

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command2

Enter the source router:4

Router4 Connection Table

Destination Interface

==========

0 4->1->0

1 4->1

3 4->1->0->3

5 4->5

8 4->5->8

9 4->5->8->9

================:Menu:================

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

=====================================

Enter a command4

Enter the router to remove1

Router4 Connection Table

Destination	Interface
5	4->5
8	4->5->8
9	4->5->8->9

================:Menu:================

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command3

Enter a destination router: 8

Distance is 10 and Path is 4->5->8

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command3

Enter a destination router: 9

Distance is 12 and Path is 4->5->8->9

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command1

Input original network topology matix data file:new.txt

Review original topology matrix:

0	1	-1	6	-1	-1	3	-1	-1	-1
1	0	3	-1	2	-1	1	-1	-1	-1
-1	3	0	-1	2	5	-1	-1	1	-1
6	-1	-1	0	-1	-1	7	-1	-1	-1
-1	2	2	-1	0	1	-1	1	-1	-1
-1	-1	5	-1	1	0	-1	-1	9	-1
3	1	-1	7	-1	-1	0	9	-1	-1
-1	-1	-1	-1	1	-1	9	0	2	1
-1	-1	1	-1	-1	9	-1	2	0	2
-1	-1	-1	-1	-1	-1	-1	1	2	0

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command

Invalid entry

================:Menu:================

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

=====================================

Enter a command2

Enter the source router:9

Router9 Connection Table

Destination	Interface
0	9->7->4->1->0
1	9->7->4->1
2	9->8->2
3	9->7->4->1->0->3
4	9->7->4
5	9->7->4->5
6	9->7->4->1->6
7	9->7
8	9->8

================:Menu:================

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command3

Enter a destination router: 3

Distance is 11 and Path is 9->7->4->1->0->3

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command4

Enter the router to remove7

Router9 Connection Table

Destination Interface

==========

0 9->8->2->1->0

1	9->8->2->1
2	9->8->2
3	9->8->2->1->0->3
4	9->8->2->4
5	9->8->2->4->5
6	9->8->2->1->6
8	9->8

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command3

Enter a destination router: 3

Distance is 13 and Path is 9->8->2->1->0->3

==========:Menu:==========

1.Create a network topology

2.Build a Connection Table

3.Shortest Path to destination router

4.Modify a topology

5.Exit

==========

Enter a command5

Exiting the menu

4. Instructions on compiling and running the file

A. The python code can be compiled and run from python2.7 interpreter.
    a. To run:
       python <filename.py>
B. The complete path of the file should be provided if the input file is not in the same directory.
C. The distance values are entered in the input file with single spaces as delimiters.
D. Router numbers are coded starting from 0 to the size of routers.
E. Open a routing table input file and then build a connection table first and then try with all other options.