Experiment 4: Distance Vector Routing

Aim: To generate routing tables for a network of routers using Distance Vector Routing

Objective: After carrying out this experiment, students will be able to:

- Generate routing tables for a given network using Distance Vector Routing
- Analyze the reasons why Distance Vector Routing is adaptive in nature

Problem statement: You are required to write a program that can generate routing tables for a network of routers. Take the number of nodes and the adjacency matrix as input from user. Your program should use this adjacency matrix and create routing tables for all the nodes in the network. The routing table should consist of one entry per destination. This entry should contain the total cost and the outgoing line to reach that destination.

Analysis: While analyzing your program, you are required to address the following points:

- Why is Distance Vector Routing classified as an adaptive routing algorithm?
- Limitations of Distance Vector Routing

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	20	

Submitted by: Deepak R

Register No: 18ETCS002041

1. Algorithm/Flowchart

Input: Graph and a source vertex src

Output: Shortest distance to all vertices from src. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

1) This step initializes distances from source to all vertices as infinite and distance to source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.

2) This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.
a) Do following for each edge u-v
If dist[v] > dist[u] + weight of edge uv, then update dist[v]

dist[v] = dist[u] + weight of edge uv

3) This step reports if there is a negative weight cycle in graph. Do following for each edge u-v

If dist[v] > dist[u] + weight of edge uv, then “Graph contains negative weight cycle”

The idea of step 3 is, step 2 guarantees shortest distances if graph doesn’t contain negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle

2. Program

main.cpp

```
#include <iostream>

#include "routing_table.hpp"

// Distance Vector Routing
int main(int, char**) {
   using namespace std;
   int order;
   cout << "Enter the Order of the Matrix : ";
   cin >> order;
   cout << "Enter the Adjacency Matrix : " << endl;

   vector<vector<int>> adj_mat;

   for (int i = 0 ; i < order ; i++) {
      std::vector<int> row(order);
      for (int j = 0 ; j < order ; j++) {
         cin >> row[j];
      }
      adj_mat.push_back(row);
   }

   // create the routing table
   RoutingTable routing_table(adj_mat);

   // calculate the shortest paths
   routing_table.apply_bellman_ford();

   // print the table
   routing_table.print_routing_table();

}
```

graph.hpp

```
#pragma once

#include <vector>

// Weighted DiGraph using Adjacency List
class Graph {
   public:
      Graph(int);
      std::vector<std::vector<std::pair<int, double>>> adj_list;
      std::vector<std::vector<int>> adj_matrix;
```

```
    const int order;

    void clear_graph();
    void add_edge(int src, int dest, double weight);

    bool is_empty() { return adj_list.empty(); };

    void print_graph();

};
```

graph.cpp

```
#include "graph.hpp"

#include <iostream>

Graph::Graph(int nodes) : adj_list(nodes), order(nodes), adj_matrix(nodes) {
  for (auto& row : adj_matrix) {
    row.resize(nodes);
    for (auto& node : row) {
      node = 0;
    }
  }
}

void Graph::add_edge(int src, int dest, double weight) {
  if (src >= adj_list.size() || dest >= adj_list.size()) {
    throw "Tried to Add Edge for Vertex that does not exist";
  }

  adj_list.at(src).push_back({dest, weight});

  adj_matrix.at(src).at(dest) = weight;
}

void Graph::clear_graph() {
  this -> adj_list.clear(); this -> adj_list.resize(order);
  for (auto& row : adj_matrix) {
    for (auto& node : row) {
      node = 0;
    }
  }
};

void Graph::print_graph() {
  std::cout << "GRAPH" << std::endl;
  int i = 0;
  for (auto& row : adj_list) {
    std::cout << "NODE " << i << " -> ";
    for (auto& ele : row) {
      std::cout << ele.first << " : " << ele.second << " , ";
    }
    std::cout << std::endl;
    i++;
  }
}
```

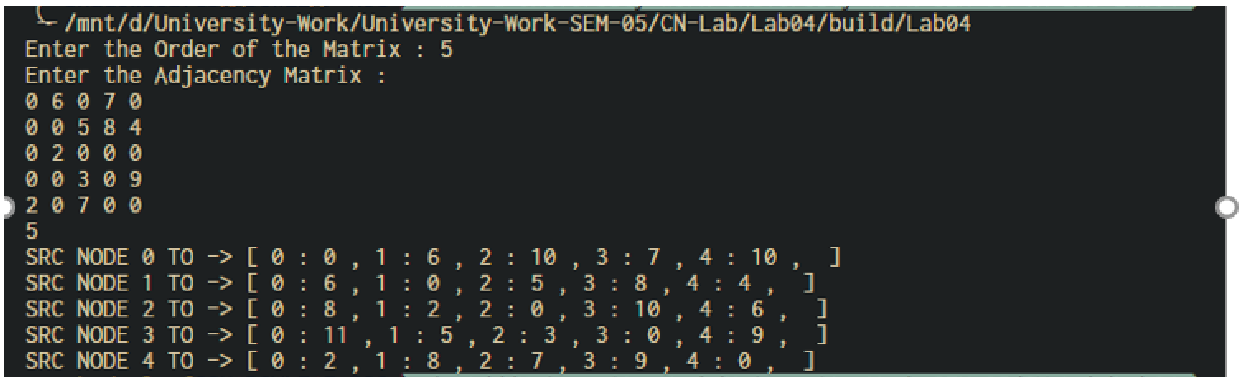
routing_table.hpp

```
#pragma once

#include "graph.hpp"

class RoutingTable {
    public:
        RoutingTable(int nodes);
        RoutingTable(std::vector<std::vector<int>>& adj_matrix);
        Graph nodes;

        std::vector<std::vector<int>> dist_matrix;

        void apply_bellman_ford();
        void print_routing_table();
};
```

routing_table.cpp

```
#include "routing_table.hpp"

#include <limits>
#include <iostream>

RoutingTable::RoutingTable(int nodes) : nodes(nodes), dist_matrix(nodes) {
    // Initialize the table to 0
    for (int i = 0 ; i < nodes ; i++) {
        for (int j = 0 ; j < nodes ; j++) {
            dist_matrix.at(i).push_back(0);
        }
    }
}

RoutingTable::RoutingTable(std::vector<std::vector<int>>& adj_matrix) : RoutingTable(adj_matrix.size()) {
    std::cout << adj_matrix.size() << std::endl;

    for (int i = 0 ; i < this -> nodes.order ; i++) {
        for (int j = 0 ; j < this -> nodes.order ; j++) {
            if (adj_matrix.at(i).at(j) > 0) {
                this -> nodes.add_edge(i, j, adj_matrix.at(i).at(j));
            }
        }
    }
}

void RoutingTable::print_routing_table() {
    int i = 0;
    for (auto& row : this -> dist_matrix) {
        std::cout << "SRC NODE " <<  i << " TO -> [ ";
        int j = 0;
        for (auto& node : row) {
            std::cout << j << " : " << node << " , ";
            j++;
        }
        std::cout << " ]" << std::endl;
        i++;
    }
}

void RoutingTable::apply_bellman_ford() {
    dist_matrix.clear();
    dist_matrix.resize(this -> nodes.order);

    for (int src = 0 ; src < this -> dist_matrix.size() ; src++) {

        auto& curr_dist = dist_matrix.at(src);

        curr_dist.resize(this -> nodes.order);

        // set distance from src to every destination as infinity
        for (auto& node : dist_matrix.at(src)) {
            node = std::numeric_limits<int>::max();
        }
        // distance from source to source is 0
        curr_dist.at(src) = 0;

        // relax all edges V-1 times
        for (int k = 0 ; k <= this -> nodes.order - 1 ; k++) {
            for (int i = 0 ; i < this -> nodes.order ; i++) {
                for (int j = 0 ; j < this -> nodes.order ; j++) {
                    if (this -> nodes.adj_matrix.at(i).at(j) != 0 && curr_dist.at(i) != std::numeric_limits<int>::max()) {
                        int weight = this -> nodes.adj_matrix.at(i).at(j);
                        curr_dist.at(j) =
                            std::min(
                                curr_dist.at(i) + weight,
                                curr_dist.at(j)
                                );
                    }
            }
         }
      }

   }
}
```

3. Results



```
╰─ /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab04/build/Lab04
Enter the Order of the Matrix : 5
Enter the Adjacency Matrix :
0 6 0 7 0
0 0 5 8 4
0 2 0 0 0
0 0 3 0 9
2 0 7 0 0
5
SRC NODE 0 TO -> [ 0 : 0 , 1 : 6 , 2 : 10 , 3 : 7 , 4 : 10 ,  ]
SRC NODE 1 TO -> [ 0 : 6 , 1 : 0 , 2 : 5 , 3 : 8 , 4 : 4 ,  ]
SRC NODE 2 TO -> [ 0 : 8 , 1 : 2 , 2 : 0 , 3 : 10 , 4 : 6 ,  ]
SRC NODE 3 TO -> [ 0 : 11 , 1 : 5 , 2 : 3 , 3 : 0 , 4 : 9 ,  ]
SRC NODE 4 TO -> [ 0 : 2 , 1 : 8 , 2 : 7 , 3 : 9 , 4 : 0 ,  ]
```

Figure 0-1 OUTPUT

The Output here is the distance from each of the node to every other node, in form of a matrix, each of each node will store its own distance table

4. Analysis and Discussions

Distance Vector Routing is a dynamic routing algorithm in which each router computes distance between itself and each possible destination i.e. its immediate neighbors. Routers that use other adaptive protocols, such as grouped adaptive routing, find a group of all the links that could be used to get the packet one hop closer to its final destination. The router sends the packet out any link of that group which is idle. The link aggregation of that group of links effectively becomes a single high-bandwidth connection.

Disadvantages of Distance Vector routing –

- It is slower to converge than link state.
- It is at risk from the count-to-infinity problem.
- It creates more traffic than link state since a hop count change must be propagated to all routers and processed on each router. Hop count updates take place on a periodic basis, even if there are no changes in the network topology, so bandwidth-wasting broadcasts still occur.
- For larger networks, distance vector routing results in larger routing tables than link state since each router must know about all other routers. This can also lead to congestion on WAN links.

Note – Distance Vector routing uses UDP(User datagram protocol) for transportation.

5. Conclusions

A Note about Dynamic Routing

Routers that use some adaptive protocols, such as the Spanning Tree Protocol, in order to "avoid bridge loops and routing loops", calculate a tree that indicates the one "best" link for a packet to get to its destination. Alternate "redundant" links not on the tree are temporarily disabled—until one of the links on the main tree fails, and the routers calculate a new tree using those links to route around the broken link.

6. Comments

a. Limitations of the experiment

Since DVR uses UDP, not all the packets reach the destination, the complexity hence is greater than O(n^3).

b. Limitations of the results obtained

The Routing Table here is pretty small, which does not justify the slowness of the algorithm, the program can be simulated for larger number of nodes and the time can be compared with other algorithms.

c. Learning

We learnt the algorithm of DVR, and its limitations from the implementation of the program, since it has O(n^3) complexity.

d. Recommendations

The node connections can be taken as a adjacency list instead of a matrix, since the matrix will always be larger than the adj-list.