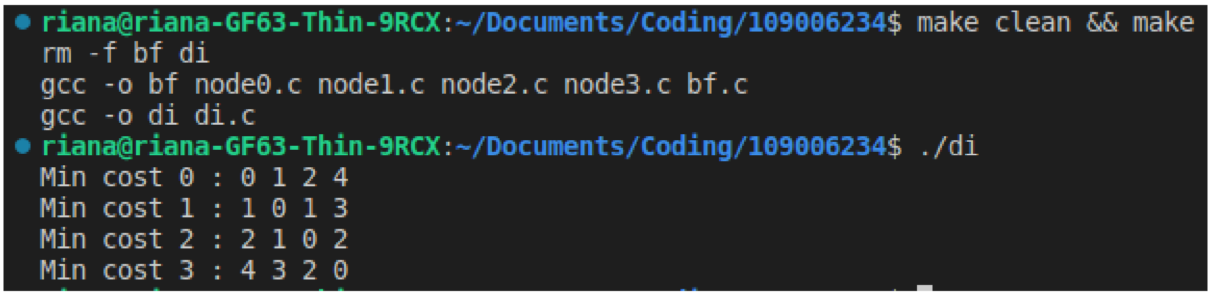
Introduction to Computer Networks - Programming Assignment 3

Anastasia Riana 邱慧莉 - 109006234

Part 1. Link State Routing with Dijkstra’s Algorithm

This algorithm is a greedy algorithm which means selecting the optimal solution that is available at the current moment. In this algorithm, it will start at the source node that we can freely choose, and it will analyze the graph, node per node, to find the best route to the chosen destination. The algorithm will iterate through the graph, while still keeping track of the currently known shortest path. If there’s a route that is better, it will update the values. Once the algorithm successfully finds the best route, that node will be marked as visited and added to the path. The algorithm will continue to iterate the whole graph and append those nodes that will lead to the shortest path into the path. This way, we will have the shortest path that connects the source node to the destination node.

Result:



```
riana@riana-GF63-Thin-9RCX:~/Documents/Coding/109006234$ make clean && make
rm -f bf di
gcc -o bf node0.c node1.c node2.c node3.c bf.c
gcc -o di di.c
riana@riana-GF63-Thin-9RCX:~/Documents/Coding/109006234$ ./di
Min cost 0 : 0 1 2 4
Min cost 1 : 1 0 1 3
Min cost 2 : 2 1 0 2
Min cost 3 : 4 3 2 0
```

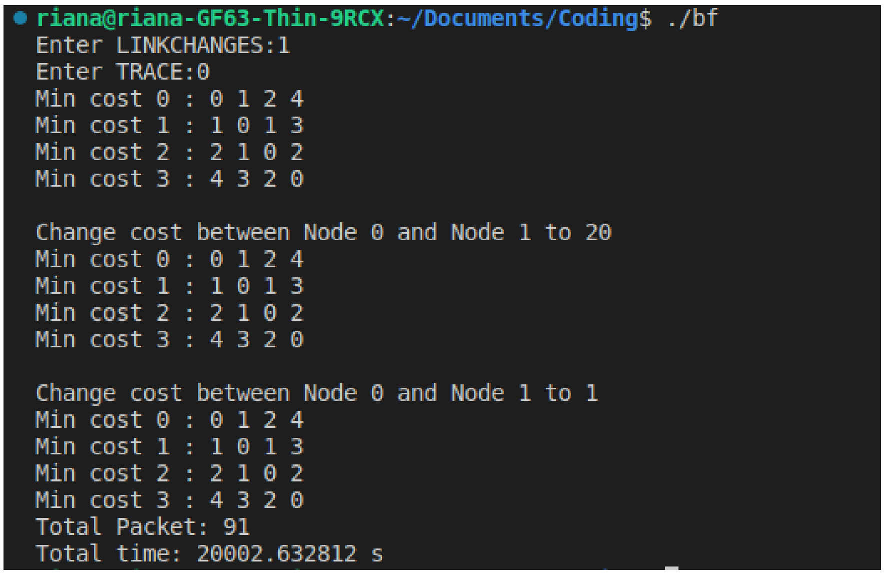
Part 2. Distance Vector Routing with Bellman-Ford Algorithm

This algorithm is an implementation of dynamic programming which is an optimization of recursion. In this algorithm, we will assign a routing table in each router. In **node0.c**, in the **rtinit0 function**, we initialize the network topology by assigning dt0.cost according to the image from the question. Then, we create the three packets to be distributed to the routers. As in node 0, we are connected to node 1, 2 and 3, so we create 3 packets. Then we set all packets at minimal cost according to the dt0 costs. Lastly, we pass all packets to layer 2. In the **rtupdate0 function**, we check if dt0 costs ar e bigger than the dt0 cost of node 0 plus the minimum cost of the received packet. For node 0, we declare the source, destination and minimal cost of all nodes that are connected to node 0. Then using a for loop, we find the minimum of the dt0 costs and assign it to packet 1, 2, 3. Lastly, we also p ass all packets to layer 2.

The same goes with the **node1.c, node2.c and node3.c**, all we need to do is d o the same thing but modify the network topology and packet’s source and d estination. When modifying, we need to keep in mind which nodes that are co nnected to the node that we are designing the routing algorithm.

In the linkhandler function, we first check if the minimum cost changes. In that case, we are going to create another packet to be sent to the other ro uters. The number of packets created depends on which node file we are codi ng on. In node0.c, we will send three packets, each one to node 1, 2 and 3. In node1.c, as the node is only connected to node 0 and 2, we will send aga in two packets, each one to node 0 and 2.

Result:



```
riana@riana-GF63-Thin-9RCX:~/Documents/Coding$ ./bf
  Enter LINKCHANGES:1
  Enter TRACE:0
  Min cost 0 : 0 1 2 4
  Min cost 1 : 1 0 1 3
  Min cost 2 : 2 1 0 2
  Min cost 3 : 4 3 2 0

  Change cost between Node 0 and Node 1 to 20
  Min cost 0 : 0 1 2 4
  Min cost 1 : 1 0 1 3
  Min cost 2 : 2 1 0 2
  Min cost 3 : 4 3 2 0

  Change cost between Node 0 and Node 1 to 1
  Min cost 0 : 0 1 2 4
  Min cost 1 : 1 0 1 3
  Min cost 2 : 2 1 0 2
  Min cost 3 : 4 3 2 0
  Total Packet: 91
  Total time: 20002.632812 s
```

Part 3. Comparison of Dijkstra and Bellman-Ford Algorithms

Q1:

Time complexity of Dijkstra’s algorithm varies depending on its implementation.

- **Case 1:** Given the graph (V,E) is represented as an adjacency matrix and weight (u,v) is stored in w[u,v] and the priority queue Q is represented by an unordered list.

First, we need to add all vertices to Q which will take O(|V|) time. Then, in some instances that we need to remove a node with minimal distance, it will take O(|V|) time. To update Q, we need to recalculate the distance array and it will take O(1). Since the chosen data structure is adjacency matrix, we need to loop for |V| time to update the distance array, which will take O(|V|) per one vertex deleted. Adding those together, the time complexity will be O(|V|)+O(|V|)*O(|V|), hence the time complexity is **O($|V|^2$).**

- **Case 2:** Given the graph (V,E) is represented as an adjacency matrix and weight (u,v) is stored in w[u,v] and the priority queue Q is represented by a binary heap.

It is first needed to create a priority queue of |V| vertices. Since the data are represented in an adjacency matrix, the graph can be traversed using BFS. Iterating all the neighbors using BFS while also updating the distance array will take O(|E|) in an loop for |V| time. For some instances that we need to remove the node with minimal distance value, as it is a heap data structure, using extract-min will take O(log|V|) time. According to the description above, the time complexity will be O(|V|)+O(|E|log|V|)+O(|V|log|V|), hence the time complexity is **O(|E|log|V|)** .

As the first case is still quadratic, the second case will perform better and O(|E|log|V|) is the best that the algorithm can get.

Q2:

In this algorithm, the first step is to initialize the vertices, which takes O(|V|) and a distance array. Next, we set the distance from the starting vertex to all vertices to be infinity and the starting vertex to itself to be zero. Then, the algorithm will start counting the shortest distance which runs for (|V|-1) times. After a certain number of iterations, we will need to relax the edges, where we choose edges for O(E) and each relax function will take O(1). In the relax function, we take a pair of edges to check and assign new weight if the conditions are met. The relax function will take O(E) time. Adding everything together, the time complexity will be **O(VE)**. As the complexity of the Bellman-Ford algorithm can be quadratic, Dijkstra's algorithm is much faster than the Bellman-Ford algorithm.

Q3:

Distance vector algorithms are basing their decision on the best path to a destination based on distance, which is usually measured using metrics such as hops, delay, packet loss, etc. In this case, we are using hops as the distance metric. For each time there's a packet going through a router, that hop is already traversed and the route with the smallest number of hops can be the best route towards the destination. The vector will show the direction for which the packet is going. This protocol will send their routing table to directly connected components within the network.

Q4:

In the link state routing algorithm, in simple terms, we need to tell everyone what we know about our neighbors. In this algorithm, we will create three different tables, to store the neighbor's information, topology table and the actual routing table. Then, the algorithm will go through a few steps.

- Step 1: Discovering neighbors

In this step, for each link state enabled router will periodically send a 'hello' on each link. The neighbors will respond to the messages to identify themselves. With this message exchanging activity, it will build up the neighbor information table.

- Step 2: Calculating the link cost

Some tests, such as throughput, end-to-end delay, etc. will be performed on each router to calculate the cost to each neighbor. Each link state enabled router has to have some type of estimation of the cost for each of its link s.

- Step 3: Building and distributing link state packets

For each router, it will build a packet that contains their neighbors' inf ormation and the link costs to them. In each router, it will also add its i dentity with an age parameter, which will be used to make sure that no pack et will get lost in the network and sequence ID. Lastly, the packet is floo ded to the network.

- Step 4: Finding the shortest path

After going through all these steps, the router can now calculate the short est path using Dijkstra' s algorithm to find the best route.

Q5:

In the distance vector routing algorithm, whenever link cost changes, a nod e will detect a local link cost change. The algorithm will update the dista nce table. In case the cost change is in the least cost path, the algorithm will notify the neighbors about the changes. In simpler terms, in the dista nce vector algorithm, good news will travel fast, while bad news will trave l slow, which can lead to the 'count to infinity' problem.

Problem Faced

Understanding the graphs algorithm is always a bit difficult, though I have studied both algorithms in my data structures class and algorithm cl ass. When I was writing the code, it was a bit challenging too to understan d the data structure that is being used from the source code, for both the implementation of Dijkstra' s and the Bellman-Ford algorithms. I was also c onfused with the Ubuntu environment, especially installing the GNU toolchai n and how to compile and run the program. Nevertheless, the video provided by the TAs on how to run and compile the files really helped a lot.