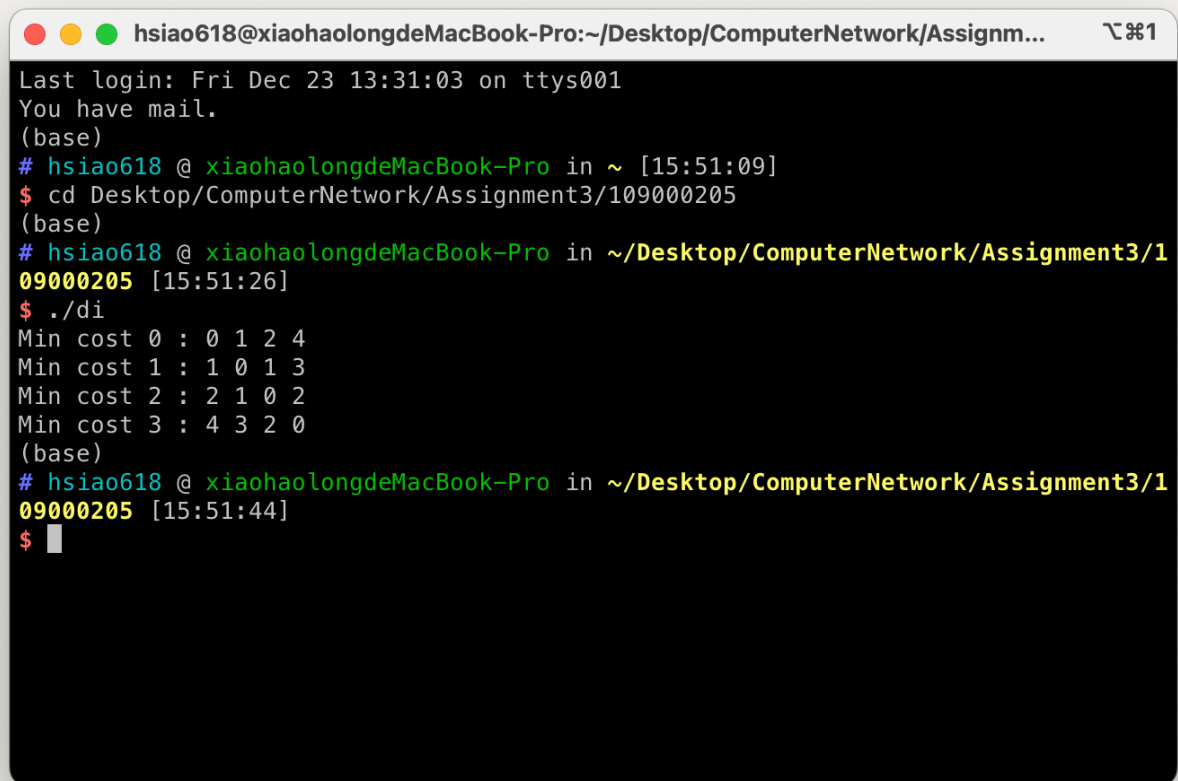


# Assignment 3 Report (109000205)

---

## Part 1

### Execution Result



```
hsiao618@xiaohaolongdeMacBook-Pro:~/Desktop/ComputerNetwork/Assignm...  
Last login: Fri Dec 23 13:31:03 on ttys001  
You have mail.  
(base)  
# hsiao618 @ xiaohaolongdeMacBook-Pro in ~ [15:51:09]  
$ cd Desktop/ComputerNetwork/Assignment3/109000205  
(base)  
# hsiao618 @ xiaohaolongdeMacBook-Pro in ~/Desktop/ComputerNetwork/Assignment3/109000205 [15:51:26]  
$ ./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  
(base)  
# hsiao618 @ xiaohaolongdeMacBook-Pro in ~/Desktop/ComputerNetwork/Assignment3/109000205 [15:51:44]  
$
```

### Implementation

- First, I copied the `table[4][4]` to an new array called `cost[4][4]`, preventing from modify the `table[4][4]`.

```

1  int cost[4][4]={infinity};
2
3  for(int i = 0; i < 4; i++) {
4      for(int j = 0; j < 4; j++) {
5          if(table[i][j] == 0) {
6              cost[i][j] = infinity;
7          } else {
8              cost[i][j] = table[i][j];
9          }
10     }
11 }

```

- Second, update the status of the source node.

```

1  for(int i = 0; i < 4; i++) {
2      nodes[id].dist[i] = cost[id][i];
3  }
4
5  for(int i = 0; i < 4; i++) {
6      nodes[id].visit[i] = 0;
7  }
8
9  nodes[id].dist[id] = 0;
10 nodes[id].visit[id] = 1;

```

- Third, find out the closest node to the source node.
- Then visit the closest node and update the distance to each node via the closest node.

```

1  int count = 1;
2  int shortestDistance;
3  int closestNode;
4
5  for(count = 1; count < 3; count++) {
6
7      shortestDistance = infinity;
8

```

```

9      int i = 0;
10     while (i < 4) {
11         if(nodes[id].dist[i] < shortestDistance &&
nodes[id].visit[i] == 0) {
12             shortestDistance = nodes[id].dist[i];
13             closestNode = i;
14         }
15         i++;
16     }
17
18     nodes[id].visit[closestNode] = 1;
19     i = 0;
20
21     while (i < 4) {
22         if(nodes[id].visit[i] == 0) {
23             if(shortestDistance + cost[closestNode][i] <
nodes[id].dist[i]) {
24                 nodes[id].dist[i] = shortestDistance +
cost[closestNode][i];
25             }
26         }
27         i++;
28     }
29 }

```

## Problems

- I put lots of effort to understanding the procedure of Dijkstra's algorithm and try to convert the concept of the algorithm to the real code.
- I referenced the procedure of Dijkstra's algorithm from the Internet and spent some time on troubleshooting and finally fully realize the code and the steps.
- Reference:
  - <https://www.programiz.com/dsa/dijkstra-algorithm>

## Part 2

### Execution Result

```
hsiao618@xiaohaolongdeMacBook-Pro:~/Desktop/ComputerNetwork/Assignm... ㄿ%1
(base)
# hsiao618 @ xiaohaolongdeMacBook-Pro in ~/Desktop/ComputerNetwork/Assignment3/1
09000205 [15:56:07]
$ ./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 4 3 5
Min cost 1 : 4 0 1 3
Min cost 2 : 3 1 0 2
Min cost 3 : 5 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: 54
Total time: 20004.210938 s
(base)
```

### Implementation

Since the function between each node is similar, I will only explain the node0. **The only difference is that node1 and node3 will not communicate with each other since there aren't any directed edge between them, which indicated that they don't need to send the update packet to each other.**

#### rtinit0()

- Create an new packet and store the cost from node0 to others in the `packet.mincost[]`.
- Use a for loop to set the packet's destination and use `tolayer2()` to send it out.

```

1  extern void rtinit0()
2  {
3      int id = 0;
4      struct rtpkt packet;
5
6      for(int i = 0; i < 4; i++) {
7          packet.mincost[i] = 999;
8      }
9
10     for(int i = 0; i < 4; i++) {
11         for(int j = 0; j < 4; j++) {
12             if (dt0.costs[i][j] < packet.mincost[i]) {
13                 packet.mincost[i] = dt0.costs[i][j];
14             }
15         }
16     }
17
18     for(int i = 1; i < 4; i++) {
19         packet.sourceid = id;
20         packet.destid = i;
21         tolayer2(packet);
22     }
23 }

```

## rtupdate0(struct rtpkt \*rcvdpkt)

- Declare the variables which can record the status of changed and the source of the received packet.
- Copy the sender's shortest distance to each node to the `tempCost[ ]`.
- Construct an array(`old_shortest[ ]`) which records the shortest distance from the current node to each node.
- Re-calculate the shortest path to each node by applying the concept of **Bellman-Ford** algorithm.
- Construct an new array(`shortest[ ]`) which records the shortest distance after the process that re-calculating the shortest distance.
- Compare the old shortest distance and the new shortest distance.

- Exists any differences: set the `changed` to 1 in order to notify that we need to send the update packet to our neighbors for later process.
- No differences: No need to send the update packet since the shortest distance to each node remain the same.

```

1  extern void rtupdate0(struct rtpkt *rcvdpkt)
2  {
3      int changed = 0;  // check update or not
4      int tempCost[4];  // store the min cost of rcvdpkt so far
5      int des = rcvdpkt->destid;
6      int src = rcvdpkt->sourceid;
7
8      if(des != 0)
9          printf("error\n");
10
11     for(int i = 0; i < 4; i++) {
12         tempCost[i] = rcvdpkt->mincost[i];
13     }
14
15     int min = 999;
16     int old_shortest[4] = {999, 999, 999, 999};
17     old_shortest[0] = 0;
18     for(int i = 0; i < 4; i++) {
19         for(int j = 0; j < 4; j++) {
20
21             if(dt0.costs[i][j] < min && dt0.costs[i][j] >= 0)
22                 min = dt0.costs[i][j];
23
24         }
25         old_shortest[i] = min;
26         min = 999;
27     }
28
29     for(int i = 0; i < 4; i++) {
30         int curCost = tempCost[i] + dt0.costs[src][src];
31         dt0.costs[i][src] = curCost;
32     }
33
34     min = 999;

```

```

35  int shortest[4] = {999, 999, 999, 999};
36  shortest[0] = 0;
37  for(int i = 0; i < 4; i++) {
38      for(int j = 0; j < 4; j++) {
39
40          if(dt0.costs[i][j] < min && dt0.costs[i][j] >= 0)
41              min = dt0.costs[i][j];
42
43      }
44      shortest[i] = min;
45      min = 999;
46  }
47
48  for(int i = 0; i < 4; i++) {
49      if(old_shortest[i] != shortest[i])
50          changed = 1;
51  }
52
53  // prepare a new packet if we need to inform our neighbors
54  if(changed == 1) {
55      // initialize our update packet
56      struct rtpkt updatePkt;
57      for(int i = 0; i < 4; i++) {
58          updatePkt.mincost[i] = 999;
59      }
60      // store the mincost to the update packet
61      for(int i = 0; i < 4; i++) {
62          for(int j = 0; j < 4; j++) {
63              if(dt0.costs[i][j] < updatePkt.mincost[i])
64                  updatePkt.mincost[i] = dt0.costs[i][j];
65          }
66      }
67
68      //send the update packet to our neighbors
69      for(int i = 1; i < 4; i++) {
70          updatePkt.sourceid = 0;
71          updatePkt.destid = i;
72          tolayer2(updatePkt);
73      }

```

```
74     }
75
76 }
```

## linkhandler0(int linkid, int newcost)

- Record the oldcost since we need to update the cost of current node in later process.
- Since only an edge's weight changed, we only need to update the cost related to that edge by using the equation `dt0.costs[i][linkid] - oldcost + newcost`.
- Store the new cost to weight of edge between the current node and the node{linkid}.
- Construct a array(`shortest`) which records the shortest distance from the current node to each node.
- Prepare for the update packet which is going to be sent to other nodes, including `packet.mincost[] = shortest[]` and setting the `packet.sourceid` and `packet.destid`.
- Send the update packet out by invoking `tolayer2()`.

```
1  extern void linkhandler0(int linkid, int newcost)
2  {
3      int oldcost = dt0.costs[1][1];
4
5      for (int i = 0; i < 4; i++) {
6          int newValue = dt0.costs[i][linkid] - oldcost + newcost;
7          dt0.costs[i][linkid] = newValue;
8      }
9
10     dt0.costs[1][1] = newcost;
11
12
13     // construct a shortest array
14     int min = 999;
15     int shortest[4] = {999, 999, 999, 999};
16     shortest[0] = 0;
17     for(int i = 0; i < 4; i++) {
18         for(int j = 0; j < 4; j++) {
```



```

19
20     if(dt0.costs[i][j] < min && dt0.costs[i][j] >= 0)
21         min = dt0.costs[i][j];
22
23     }
24     shortest[i] = min;
25     min = 999;
26 }
27
28 // initialize our update packet
29 struct rtpkt updatePkt;
30 for(int i = 0; i < 4; i++) {
31     updatePkt.mincost[i] = shortest[i];
32 }
33
34 //send the update packet to our neighbors
35 for(int i = 1; i < 4; i++) {
36     updatePkt.sourceid = 0;
37     updatePkt.destid = i;
38     tolayer2(updatePkt);
39 }
40
41 }

```

## Problems

- Part2 is the part that I spend most of the time in this Assignment.
- I encountered some bugs when the edge's weight changed, I thought the mistake is from the `linkhandler()`, but after discussing with the classmates, I modify my `rtupdate()` to meet the requirements.
- Before modifying, I didn't construct `old_shortest` and `shortest` which are the old shortest distance array and the new shortest distance array. By fixing that, I can compare the differences between the two array and determine whether I should send the update packet to other nodes or not.

## Part 3

(1) What is the time complexity of Dijkstra's algorithm? Please briefly describe the steps of the algorithm to justify the time complexity.

- My implementation of Dijkstra's algorithm is a greedy algorithm.
- I used a nested loop to find out the closest node to the source node and use that node to update the shortest distance to others.
- The procedure above indicates that the time-complexity of Dijkstra's algorithm in my implementation is  $O(|V|^2)$ , where  $|V|$  is the number of vertices.
- However, if we use a Fibonacci heap to deal with the problem, the time complexity can be reduced to  $O((|V| + |E|)\log|V|)$ , where  $|V|$  is the number of the vertices and  $|E|$  is the number of edges.

(2) What is the time complexity of Bellman-Ford algorithm? Please briefly describe the steps of the algorithm to justify the time complexity.

- In Bellman-Ford algorithm, we need to initialize the shortest distance array to each node.
- Then apply a nested for loop to traverse each node with each edge and update the cost.
- Thus the time-complexity is  $O(|V| * |E|)$ , where  $|V|$  is the number of the vertices and  $|E|$  is the number of edges.

(3) How does the distance vector routing algorithm send routing packets? (To all nodes or only to neighbors)

- Distance Vector Routing Algorithm sends routing packets only to its neighbors.
- The communicating with the neighbors takes place at regular intervals.
- The router shares its routing table with its neighbors and updates the table according to the neighbors' table.
- Moreover, it often uses the Bellman-Ford algorithm for making the routing table and UDP for transportation.

(4) How does the link state routing algorithm send routing packets?  
(To all nodes or only to neighbors)

- Link State Routing Algorithm send routing packet to all nodes.
- The communicating with the neighbors takes place whenever there is a change.
- The router sends its table only to all the routers through flooding.
- Moreover, it often uses Dijkstra's algorithm for making routing table and TCP/HTTP for transportation.

(5) When a link cost changes, which steps does the distance vector algorithm take?

- Once the distance changed, the router will update its routing table and correct the cost of the shortest distance to each router.
- Then it will construct a packet which contains all the shortest distance to other nodes and use UDP send this packet to the neighbor routers in order to share the information(routing table).