ANC Assessed Exercise

Robert Allison 1102085a

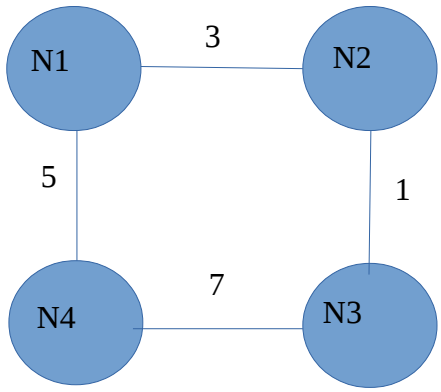
Example Walkthroughs:

Case 1: Normal Convergence

Input network: normalcon.txt

{N1,N2,N3,N4}
(N1,N2,3)
(N2,N3,1)
(N3,N4,7)
(N4,N1,5)

This is a **square** network, represented as such:



To run this program, will will use the command: “java -jar DVRouting.jar normalcon -s -v N1”

As the program runs, we observe Node N1's routing table

Exchange 1

Table for N1
node: 'N1', cost: 0, nextlink: None

Stable: false

At the first exchange, before passing of vectors, we see that N1 has only the entry for itself, with a cost of 0 and no next link

Exchange 2

Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 5, nextlink: N4
node: 'N2', cost: 3, nextlink: N2

Stable: false

After the second exchange, N1 has received information from its direct neighbours, N1 and N4, and has added them to its routing table along with their direct path costs.

```
Exchange 3
------------------------------
Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 5, nextlink: N4
node: 'N3', cost: 4, nextlink: N2
node: 'N2', cost: 3, nextlink: N2
------------------------------
Stable: true
Stability achieved
```

After the third exchange the program ends, as all the nodes report stability. We can see that N1 has received information about N3, which is one hop away, and has deduced the shortest route is through N2, with a total cost of 3+1 =4.
As an bonus, we can add a trace command: java -jar DVRouting.jar normalcon -s -v N1 -t N1,N3,3
And receive this output on exchange 3:

```
Exchange 3
Path from N1 to N3
N1 => N2 => N3
------------------------------
Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 5, nextlink: N4
node: 'N3', cost: 4, nextlink: N2
node: 'N2', cost: 3, nextlink: N2
------------------------------
```

As you can see, we can trace the path from N1 to N3 through N2. A final interesting point to check is what happens when we fail a link. We can use the command: java -jar DVRouting.jar normalcon -e 5 -v N1 -f N1,N2,3, to achieve this, failing the link between N1 -N2 on exchange 3, and running for 5 to see the updates. Taking the relevant output slice:

```
Exchange 3
Failing Link: N1 - N2
------------------------------
Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 5, nextlink: N4
node: 'N3', cost: 4, nextlink: N2
node: 'N2', cost: 2147483647, nextlink: N2
------------------------------
```

We can see from this table, that as the link has been cut, N1 sees N2 as an impassably huge cost.
And when we update:

Exchange 4

Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 5, nextlink: N4
node: 'N3', cost: 12, nextlink: N4
node: 'N2', cost: 13, nextlink: N4

N1 has now recalculated its route, finding a smaller cost route to N2, going N4 => N3 => N2, with a total cost of 5+7+1 = 13.

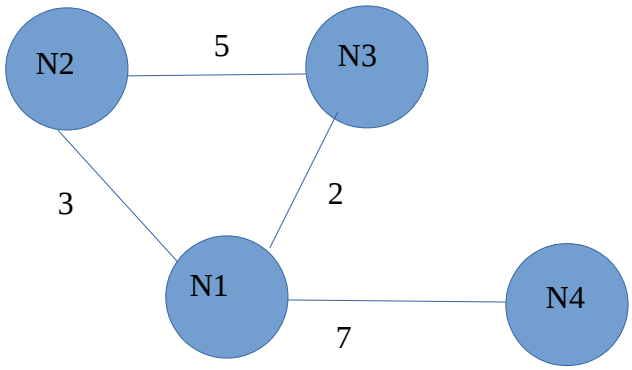
Using this example, we can see that the network functions normally under normal convergence, and can handle link breakages and cost changes effectively.

Case B: Normal Convergence without split horizon

Input network: slowcon.txt

{N1,N2,N3,N4}
(N1,N2,3)
(N2,N3,5)
(N3,N1,2)
(N4,N1,7)

This is a **triangular** network with a **leaf**, as shown here:



With this network, if we cut the link to N4, we should experience a loop. We can simulate this using the command: java -jar DVRouting.jar slowcon -e 5 -v N1 -f N1,N4,3

Producing the output show here:

Exchange 2

Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 7, nextlink: N4
node: 'N3', cost: 5, nextlink: N3
node: 'N2', cost: 3, nextlink: N2

Looks like the links are updating correctly so far.

Exchange 3

Failing Link: N1 - N4

Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 2147483647, nextlink: N4
node: 'N3', cost: 4, nextlink: N2
node: 'N2', cost: 3, nextlink: N2

As we saw in normal convergence, we cut the link, N1 immediately sees it as a huge, impassable cost.

Exchange 4

Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 13, nextlink: N2
node: 'N3', cost: 4, nextlink: N2
node: 'N2', cost: 3, nextlink: N2

But wait, now it is showing that it found a route through N2, with a cost 13? This should not happen, the link is broken!

Exchange 5

Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 16, nextlink: N2
node: 'N3', cost: 4, nextlink: N2
node: 'N2', cost: 3, nextlink: N2

Now the cost is 16! Clearly, if we kept running this, we would see this number increase and increase until it hit infinity, as there is a broken loop here. We can observe that this is correct even later:

Exchange 50

Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 59, nextlink: N2
node: 'N3', cost: 4, nextlink: N2
node: 'N2', cost: 3, nextlink: N2

Clearly, with a slow convergence graph like this, loops and count-to-infinity are a definite problem.

Case B: Normal Convergence without split horizon

Input network: slowcon.txt
The network is the same as last time, but with one small modification in the command: java -jar DVRouting.jar slowcon -e 5 -v N1 -f N1,N4,3 – split
With -split we turn on the network's split horizon functionality.

Exchange 2

Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 7, nextlink: N4
node: 'N3', cost: 5, nextlink: N3
node: 'N2', cost: 3, nextlink: N2

We can see that, as before, the graph is build itself normally

Exchange 3
Failing Link: N1 - N4

Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 2147483647, nextlink: N4
node: 'N3', cost: 4, nextlink: N2
node: 'N2', cost: 3, nextlink: N2

When we break the link, it is immediately registered in node N1, as it is directly connected to to N4

Exchange 4

Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N3', cost: 4, nextlink: N2
node: 'N2', cost: 3, nextlink: N2

Now due to split horizon, N4 simply no longer appears in the list of nodes for N1, as there is no connection to it.

Exchange 5

Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N4', cost: 16, nextlink: N2
node: 'N3', cost: 4, nextlink: N2
node: 'N2', cost: 3, nextlink: N2

But wait, it's appeared again, as a link through N2, just as before.

Exchange 6

Table for N1
node: 'N1', cost: 0, nextlink: None
node: 'N3', cost: 4, nextlink: N2
node: 'N2', cost: 3, nextlink: N2

Now N4 has disappeared again, and it will do so for a few more exchanges, but will occasionally crop up again with an increased value. My split horizon implementation is a bit buggy, but we can see it in action, as it is (mostly) stopping repeat information being sent back in a loop. Most importantly though, using split horizon, this graph can achieve **stability**, which is something it could not do otherwise, as **triangular graphs of this sort generally do not function well with breakages, even with split horizon functionality enabled.**