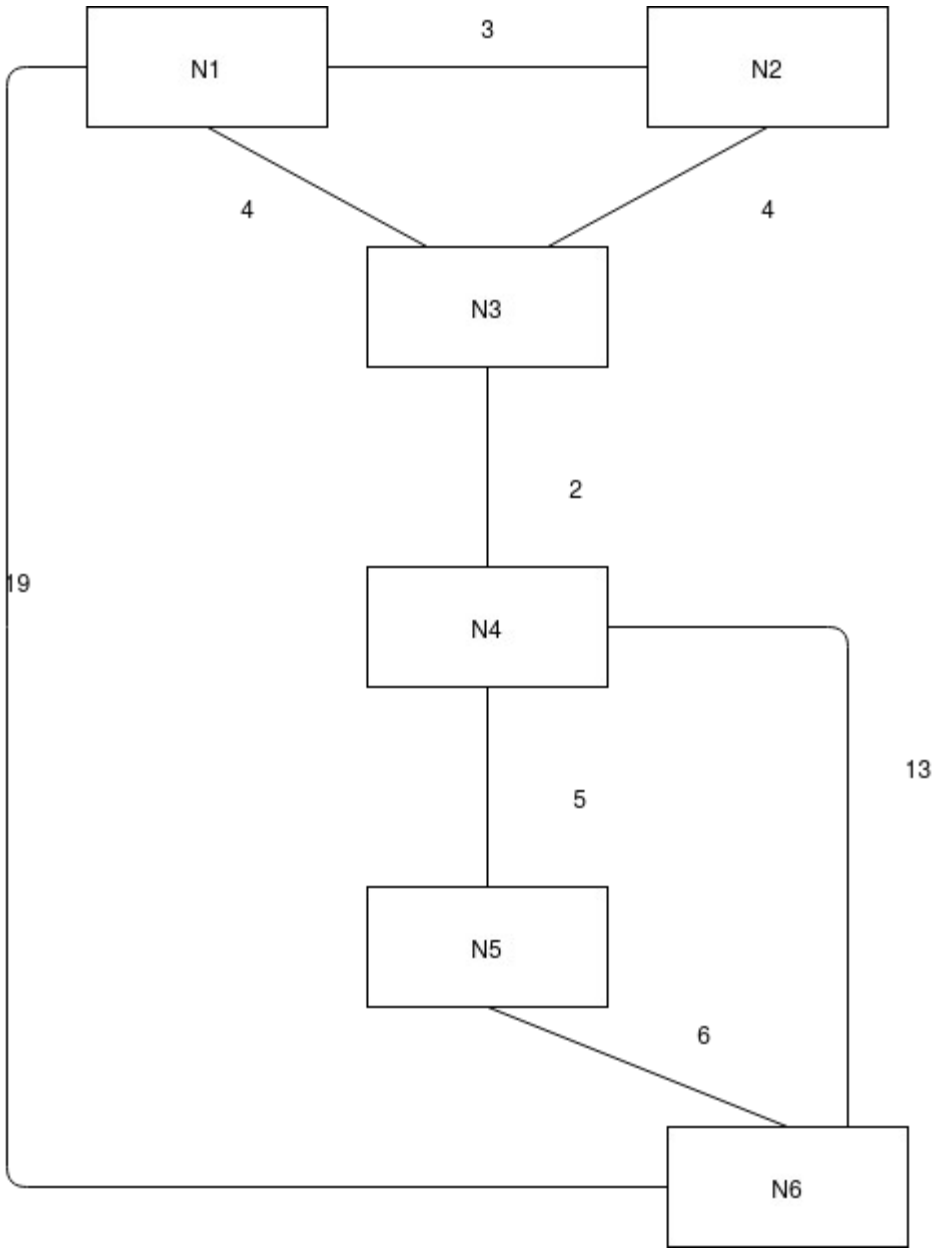
Example networks walk-through

Case 1:

The network I have chosen to represent looks like this



In terms of input into my program, the input file looks like this:

{ "nodes" : ["N1", "N2", "N3", "N4", "N5", "N6"],
"connections" : [["N1", "N2" , 3], ["N2", "N3" , 4], ["N3", "N1", 4],["N3", "N4", 2], ["N4", "N5" , 5], ["N4", "N6", 13,], ["N5", "N6", 6], ["N1", "N6", 19]]}

Example networks walk-through

This walk-through can be run using the command `python3 case1.py`. It can also be run manually using the application and the file case1.txt

If using the application version, the commands are:
load_file case1.txt
converge_run_updates N1 N4
print_tables

The final routing tables produced by look like this:

```
Final Routing tables
N1
--------------------
Node | Link | Distance |
N2     N2     3
N3     N3     4
N6     N3     17
N4     N3     6
N5     N3     11


N2
--------------------
Node | Link | Distance |
N1     N1     3
N3     N3     4
N6     N3     17
N4     N3     6
N5     N3     11


N3
--------------------
Node | Link | Distance |
N2     N2     4
N1     N1     4
N4     N4     2
N6     N4     13
N5     N4     7
```

Example networks walk-through

```
N4
--------------------
Node | Link | Distance |
N3     N3     2
N5     N5     5
N6     N5     11
N2     N3     6
N1     N3     6


N5
--------------------
Node | Link | Distance |
N4     N4     5
N6     N6     6
N3     N4     7
N1     N4     11
N2     N4     11


N6
--------------------
Node | Link | Distance |
N4     N5     11
N5     N5     6
N1     N4     17
N2     N4     17
N3     N4     13
```

This is a very straight forward example. On each exchange, the nodes send their distance vector tables to their neighbours. The nodes then check all the data they have and update their distance vector tables to have the lowest distance.

To look at an example route, consider node N4. In the final tables, N4 has decided to route via N5 and not directly via N6, as this is a shorter route. However, when initialised, N4 decided to route via N6. It did not know N5 has a connection to N6 and so decided to use its direction connection:

```
N4
--------------------
Node | Link | Distance |
N3     N3     2
N5     N5     5
N6     N6     13
```

Example networks walk-through

After the first exchanges are sent, N4 now routes via N5 to get to N6 as this is a shorter distance. N5 will have provide its distance vector table to N4 and N4 will have compared tables to realise this. It can also be seen N4 has learned of routes to N2 and N1 from N3 and added these to the distance vector table

```
Exchange 1:
N4
--------------------
Node | Link | Distance |
N3     N3     2
N5     N5     5
N6     N5     11
N2     N3     6
N1     N3     6
```

Other nodes, however, take longer to realise this change. N1 knows it is initially connected to N6 for a distance of 19. This is greater than the distance via N3, N4, N5 to N6. However, it takes until the third exchanges for N1 to learn this different link option.

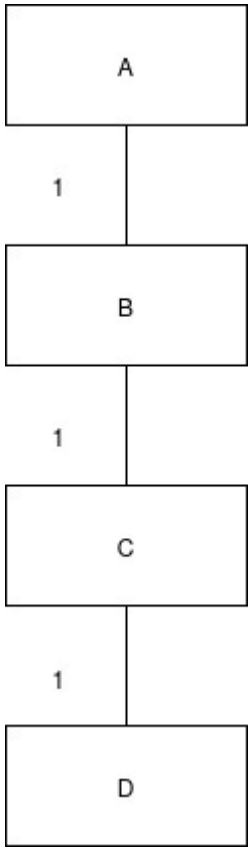
```
Exchange 1
N1
--------------------
Node | Link | Distance |
N2     N2     3
N3     N3     4
N6     N6     19
N4     N3     6
N5     N6     25

Exchange 2
N1
--------------------
Node | Link | Distance |
N2     N2     3
N3     N3     4
N6     N6     19
N4     N3     6
N5     N3     11
```

Example networks walk-through

```
Exchange 3
N1
--------------------
Node | Link | Distance |
N2     N2     3
N3     N3     4
N6     N3     17
N4     N3     6
N5     N3     11
```

This delay is because for every exchange from N4, there is a delay where N3 must update their tables before advertising on the next exchange any changes made. So every exchange to N1 from N4 is delayed twice, once for N4 to advertise to N3 and the again for N3 to advertise to N1. As a result the final update occurs at exchange 3 and the network then converges. Every other learned path is similar to this. In this simple example, this delay has very little affect. Node 1 sends traffic the "wrong" way for the time it takes two exchanges to occur. But it can be seen that at very large networks, this may take a long time to propagate through.

Case 2:
The network I have chosen to represent looks like this:



Example networks walk-through

In terms of input into my program, the file looks like this:
{ "nodes" : ["A", "B", "C", "D"],
"connections": [["A", "B",1],["B", "C", 1], ["C", "D" , 1]]}

This walk-through can be seen using the command python3 case2.py or run manually in the application using case2.txt

The commands to use with the application are:
load_file case2.txt
set_count_to_infinity 16
converge_run_updates
print_tables
down_link A B
converge_run_updates B C D
print_tables
undo
activate_split_horizon
converge_run_updates B C D
print_tables

The network converges normally, taking 3 exchanges. The final routing tables look as follows:

```
A
---------------------
Node | Link | Distance |
B     B      1
C     B      2
D     B      3


B
---------------------
Node | Link | Distance |
A     A      1
C     C      1
D     C      2
```

Example networks walk-through

```
C
--------------------
Node | Link | Distance |
B     B      1
D     D      1
A     B      2


D
--------------------
Node | Link | Distance |
C     C      1
B     C      2
A     C      3
```

This is to be expected and is a relatively fast convergence taking only 3 exchanges. However, when a link fails, the network does not recover or converge very easily. In the case for this example, when link A – B fails, there is a loop to infinity. No node has a complete map of the network, just the next node to on the route to every other node. As a result, only Node B can set the distance of A to be infinite. On the exchange update, node B receives from Node C that it can access A for a distance of 2(*via Node B*). Similarly, Node C will receive the update from Node B that the link to A is now infinite, while simultaneously Node D will advertise to Node C it can reach Node A in 3 jumps(*via Node C*). As a result, the infinite value for Node A is essentially ignored and each Node adds the distance of their connection to the update received and they set the new distance vector table. B will now advertise 3 jumps to A, C will advertise 4 jumps and D will still advertise 3 jumps to Node A on the next exchange. This will then increment slowly to infinity.

```
Link A, B fails
Exchange 1
B
--------------------
Node | Link | Distance |
A     C      3
C     C      1
D     C      2

C
--------------------
Node | Link | Distance |
B     B      1
D     D      1
A     D      4
```

Example networks walk-through

```
D
--------------------
Node | Link | Distance |
C     C      1
B     C      2
A     C      3


Exchange 2
B
--------------------
Node | Link | Distance |
A     C      5
C     C      1
D     C      2

C
--------------------
Node | Link | Distance |
B     B      1
D     D      1
A     D      4

D
--------------------
Node | Link | Distance |
C     C      1
B     C      2
A     C      5

Exchange 3
B
--------------------
Node | Link | Distance |
A     C      5
C     C      1
D     C      2

C
--------------------
Node | Link | Distance |
B     B      1
D     D      1
A     B      6
```

Example networks walk-through

```
D
--------------------
Node | Link | Distance |
C     C      1
B     C      2
A     C      5
```

Normally, there is a limit on this value of infinity to prevent loops like this lasting forever(*or until the memory storing the distance number overflows*). In the case of this program, infinity was set to the value of 16. Any number higher than sixteen was considered infinity and so the network converges after 16 exchanges.

Using the split horizon method prevents this “infinite” looping. Instead of C advertising back to B, an impossible route, it only advertises to D and D doesn’t advertise to anyone. B then advertises to C that A is now of distance infinite, which updates and then advertises to D on the next exchange and which updates. This reduces convergence from the arbitrary number of exchanges defined by the count to infinity value to 2 exchanges. The merit of split horizon can therefore be seen in this case, to greatly reduce time for convergence

```
Exchange 1
B
--------------------
Node | Link | Distance |
A     A     inf
C     C     1
D     C     2

C
--------------------
Node | Link | Distance |
B     B     1
D     D     1
A     B     inf

D
--------------------
Node | Link | Distance |
C     C     1
B     C     2
A     C     3
```

Example networks walk-through

```
Exchange 2
B
--------------------
Node | Link | Distance |
A     A     inf
C     C     1
D     C     2

C
--------------------
Node | Link | Distance |
B     B     1
D     D     1
A     B     inf

D
--------------------
Node | Link | Distance |
C     C     1
B     C     2
A     C     inf
```