./routing -top topologyfile -conn connectionsfile -rt routingtablefile -ft forwardingfile -path pathsfile -flag hop|dist|relb|degree -p 0|1

Metrics

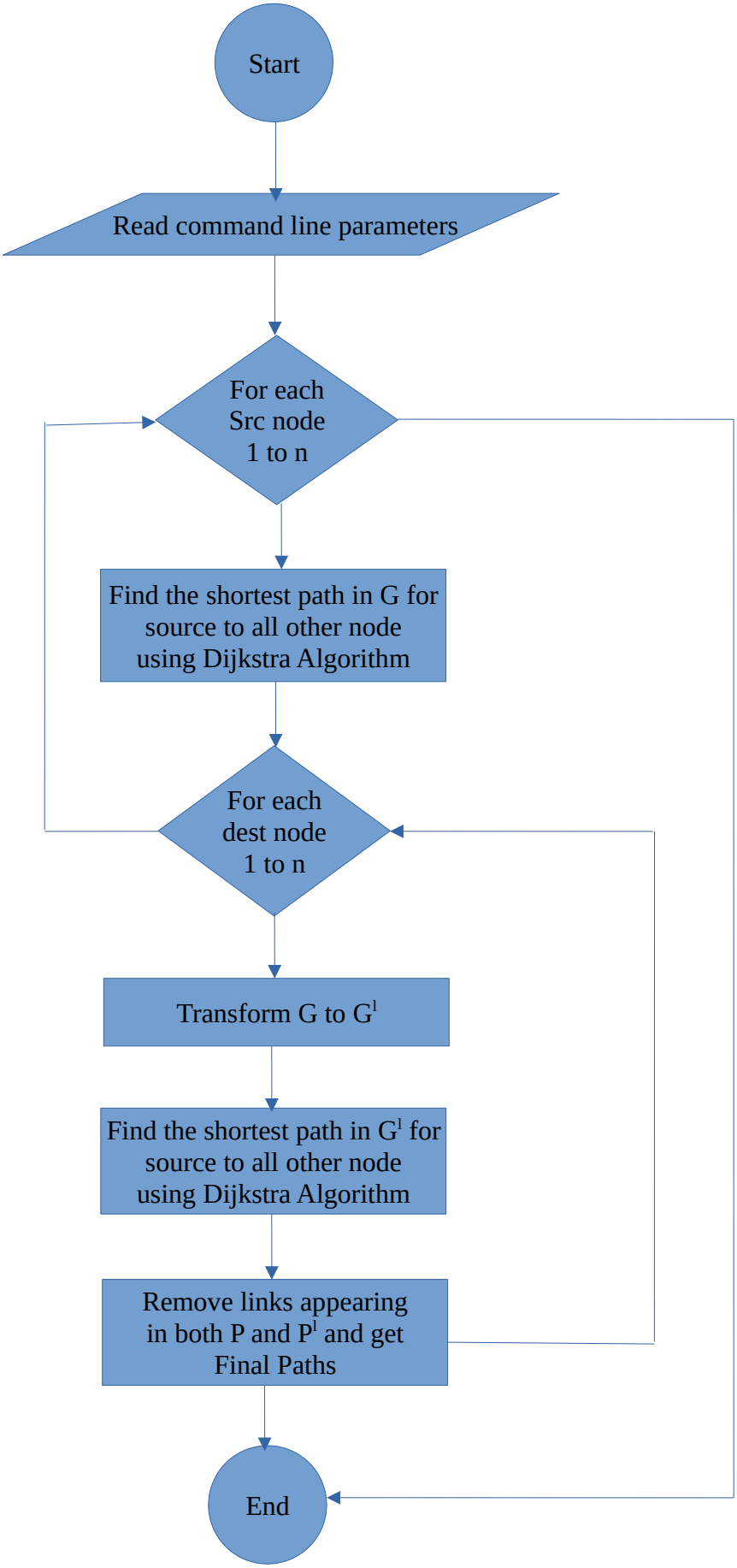
hop -> Hop is the number of intermediate nodes between the source and destination. Dijkstra's Algorithm picks the path which has less number of hop counts.

dist -> Distance nothing but delay. Dijkstra's Algorithm picks the path which has less delay between source and destination.

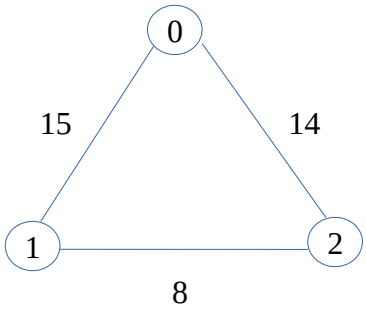
Relb -> The average reliability values given in the topology file should be inverted and stored in an array (average unreliable array), so that Dijkstra's algorithm picks the path which is less unreliable. Formula to calculate the Unreliability.

```
averageUnReliability[j][i]=averageUnReliability[i][j]= 1.0 - averageReliability[i][j];
```

degree -> Degree is the number of connections to a node. So calculate the degree for each node and store it an array. Dijkstra's algorithm picks the node which has the lowset degree to establish the path between source and destination.



Suurballe Algorithm - Flow Chart



Topology file:

3 3
0 1 2 15 0.6
0 2 5 14 0.6
1 2 2 8 0.7

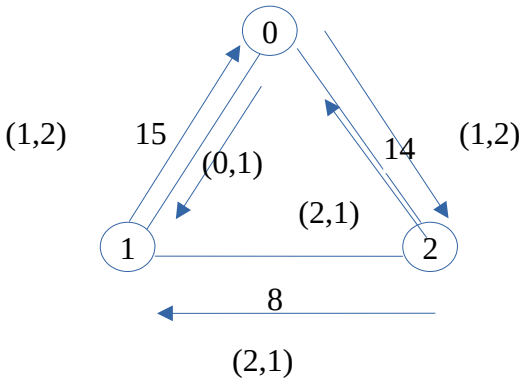
Source	Destination	Delay	Capacity	Avg Reliability
0	1	2	15	0.6
0	2	5	14	0.6
1	2	2	8	0.7

Connection file:

6
0 1 5 0 **6**
0 2 1 1 **17**
1 2 6 7 **9**
2 1 5 8 **7**
2 0 5 8 **7**
1 0 9 9 **9**

source destination min avg max bandwidth

Pessimistic Approach.



0,1 - Required bandwidth = 6
Available

```
      0            1            2
[[0, 15, 14], [15, 0, 8], [14, 8, 0]]
```

for connection request 0,1 connection is established on first path

(0,2) – Required bandwidth = 17
Available

```
      0            1            2
 [[0, 9, 14], [15, 0, 8], [14, 8, 0]]
```

There is no path availaible for this connection request (0,2)

(1,2) – Required bandwidth = 9
Available

```
      0           1           2
 [[0, 9, 14], [15, 0, 8], [14, 8, 0]]
```
for connection request 1,2 connection is established on second path
After connection
```
      0           1           2
 [[0, 9, 5], [6, 0, 8], [14, 8, 0]]
```

(2,1) – Required bandwidth = 7
Available
```
      0           1           2
 [[0, 9, 5], [6, 0, 8], [14, 8, 0]]
```
for connection request 2,1 connection is established on first path

(2,0) – Required bandwidth = 7
Available
```
      0           1           2
 [[0, 9, 5], [6, 0, 8], [14, 1, 0]]
```
for connection request 2,0 connection is established on first path

(1,0) – Required bandwidth = 9
Available
```
      0           1           2
 [[0, 9, 5], [6, 0, 8], [7, 1, 0]]
```
There is no path availaible for this connection request (1,0)

Forwarding table

Paths
```
0->1        VC ID 0
1->0->2     VC ID 1
2->1        Ignore
2->0        VC ID 2
```

Node 0

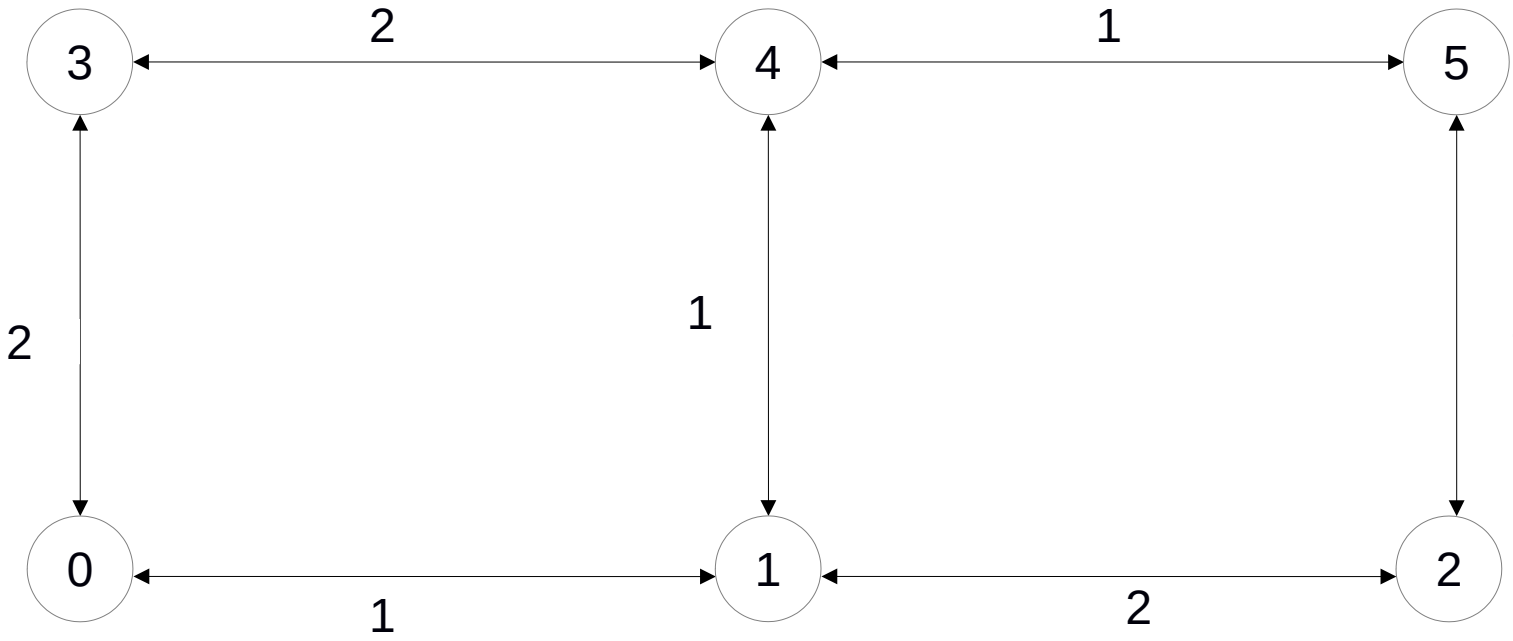
Node ID of Incoming Port	VC ID	Node ID of Outgoing Port	VC ID
-1	-1	1	0
1	1	2	1
2	2	-1	-1

RoutingTable: ROOT NODE 0

Destination Node	path delay	path cost	path(from source to destination)
0	0	0	0
0	0	0	0
1	2	1	0->1
1	7	2	0->2->1
2	5	1	0->2
2	4	2	0->1->2

PathFile:

Conn. Id	Source	Destination	Path	VC ID List	Path Cost
0	0	1	0->1	0,-1	1
1	1	2	1->0->2	1,1,-1	1
2	2	1	2->1	1,-1	1
3	2	0	2->0	2,-1	1



How to find the disjoint paths using P1 and P2

P1 = 0->1->4->5
P2 = 0->3->4->1->2->5

Finding Final Path FP1
TempP1 = 0->1->4->5
P2 = 0->3->4->1->2->5

Initialize
FP1 = 0 //take the first element of TempP1.

Step1 :Initialize
TempP1 <- P1
N <- Starting node of TempP1
FP1 <- N

Step2: N <- Point to next node of TempP1
Step3: Append N to FP1
Step4: Check P2 Contains N and N is not destination If Yes Step5 ELSE Step 6
Step5: Remove the rest of the path from TempP1 after N and copy the Rest of the path from P2 after N
Step6: If N is destination Goto Step ELSE go to Step2

P = 0->1->4->5
P2 = 0->3->4->**1**->2->5
Iteration-1
TempP1 = 0->1->4->5
N =0
FP1=0
dest=5

Iteration1:N=1
FP= 0->1

If {0->3->4->**1**->2->5} contains N && N != dest
TemP1= 0->1
TempP2 = 2->5
TempP1= 0->1->2->5

If N == N
break the loop

Iteration2:N=2

FP= 0->1->2

//TempP1= 0->1->2->5

If {0->3->4->1->2->5} contains N && N != dest

TemP1= 0->1->2

TempP2 = 5

TempP1= 0->1->2->5

If N == N

break the loop

Iteration3:N=5 //TempP1= 0->1->2->5

FP= 0->1->2->5

If {0->3->4->1->2->5} contains N && N != dest //Condition Fails

{}

If N == N //Yes

break the loop

//Repeate same for Path2 to find the second disjoint path