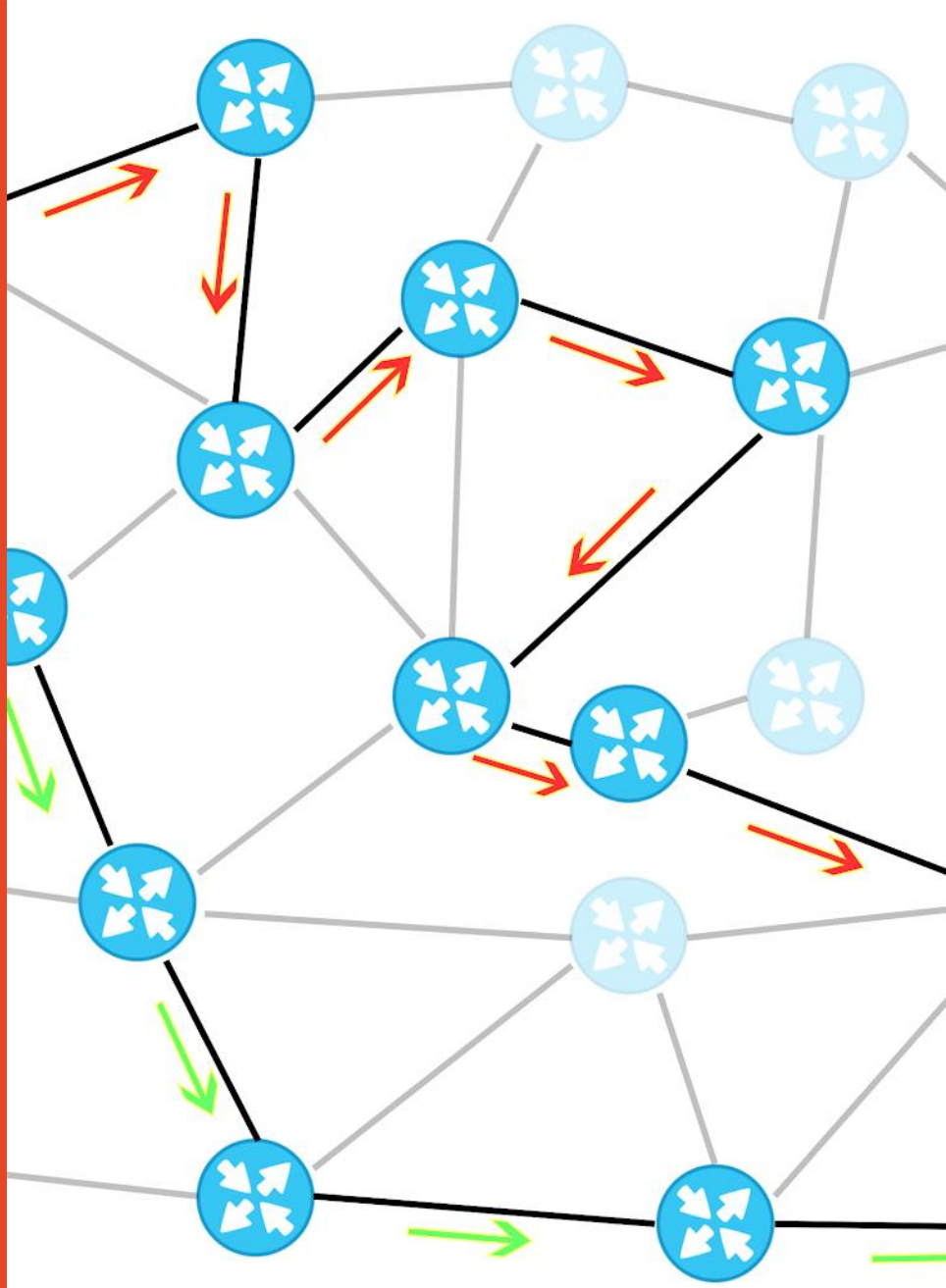
Distributed Systems

Lab 2: Routing and Communication



THE UNIVERSITY OF SYDNEY

Outline

- *Exercise 1*: Router Information Protocol (RIP)
- *Exercise 2*: Dijkstra's Shortest Path Algorithm
- *Exercise 3*: Path calculation using NetworkX
- *Exercise 4*: Generate Network Topology with NetworkX

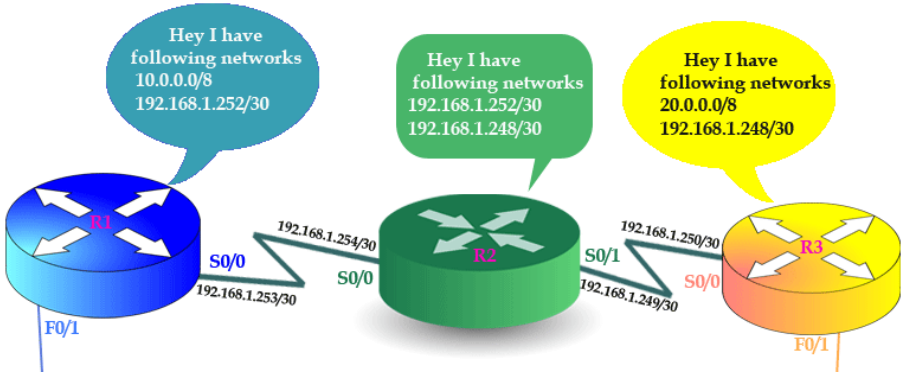
Exercise 1

Lab 2 – COMP3221

THE UNIVERSITY OF SYDNEY

Exercise 1

Router Information Protocol (RIP)



Recap:

- A distance-vector algorithm that uses **hop count** as its primary metric
  - Decide which path to put a packet on to get to its destination
- Each RIP router maintains a routing table
  - A list of all the destinations the router knows how to reach
- Each router broadcasts its entire routing table to its closest neighbors
  - Until all RIP hosts within the network have the same knowledge of routing paths
  - **Network Convergence**

Image Source: https://www.computernetworkingnotes.com/ccna-study-guide/rip-routing-information-protocol-explained.html

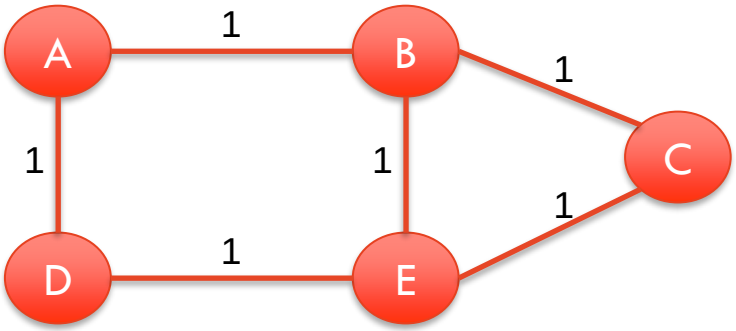
Exercise 1

Router Information Protocol (RIP)

❑ **Our problem:**

Execute (by hand) the RIP protocol on the communication graph represented in Figure 1 where nodes represent routers and edges represent communication links between routers.

Consider initially that the routing tables are empty (containing only node-local information). Given that no failures occur (neither message losses, nor node failures), write down the final routing table each router obtains after convergence of the protocol.



Routing Table of Router		
Dest.	**Dir**	**Cost**

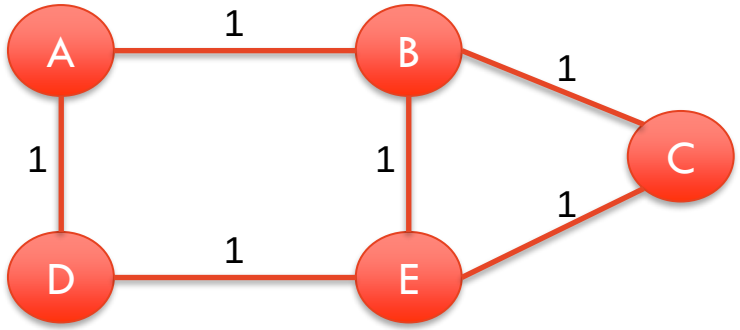
Exercise 1

Router Information Protocol (RIP)

Let's start the algorithm from scratch: all routing tables are empty

Routing table of router A		
Dest.	**Dir**	**Cost**
A	local	0

Routing table of router B		
Dest.	**Dir**	**Cost**
B	local	0



Routing table of router C		
Dest.	**Dir**	**Cost**
C	local	0

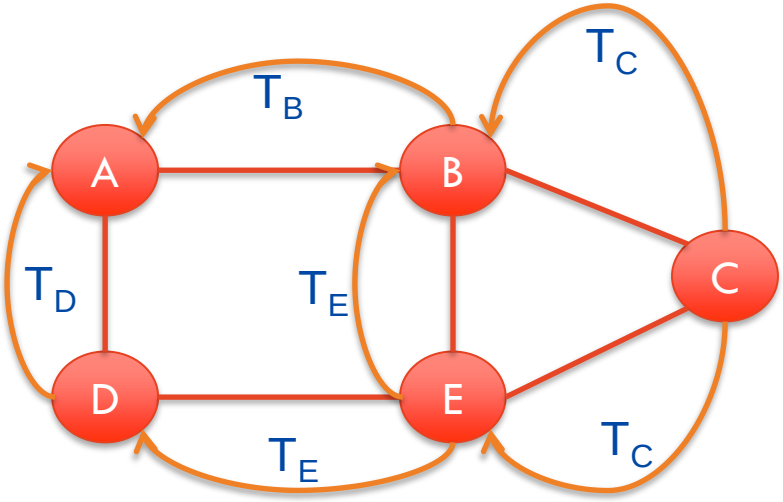
Routing table of router D		
Dest.	**Dir**	**Cost**
D	local	0

Routing table of router E		
Dest.	**Dir**	**Cost**
E	local	0

Exercise 1

Router Information Protocol (RIP)

– Example: How is routing table A built from scratch?



Routing table of router A		
Dest.	**Dir**	**Cost**
A	local	0
B	B	1
D	D	1

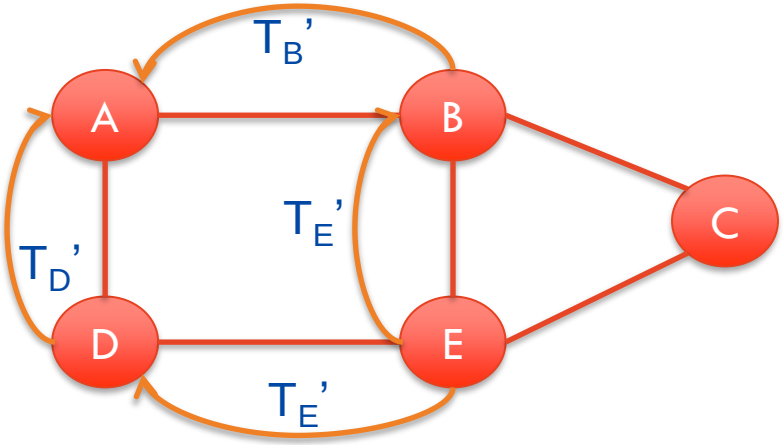
Routing table of router B		
Dest.	**Dir**	**Cost**
B	local	0
A	A	1
C	C	1
E	E	1

Iteration 1
A: knows B and D (both at a distance of 1 hop)
B: knows A, C, and E (all at a distance of 1 hop)

Exercise 1

Router Information Protocol (RIP)

– Example: How is routing table A built from scratch?



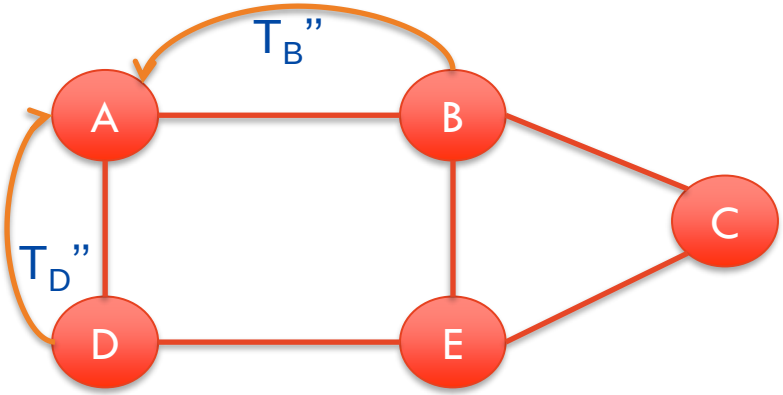
Routing table of router A		
Dest.	**Dir**	**Cost**
A	local	0
B	B	1
C	B	2
D	D	1
E	B	2

Iteration 2
A: knows B and D (1 hop), C (via B, 2 hops), and E (via B or D, 2 hops)

Exercise 1

Router Information Protocol (RIP)

– Example: How is routing table A built from scratch?



Routing table of router A		
Dest.	**Dir**	**Cost**
A	local	0
B	B	1
C	B	2
D	D	1
E	B	2

Iteration 3
A: knows B and D (1 hop), C and E (via B, 2 hops) (No changes)

Exercise 1

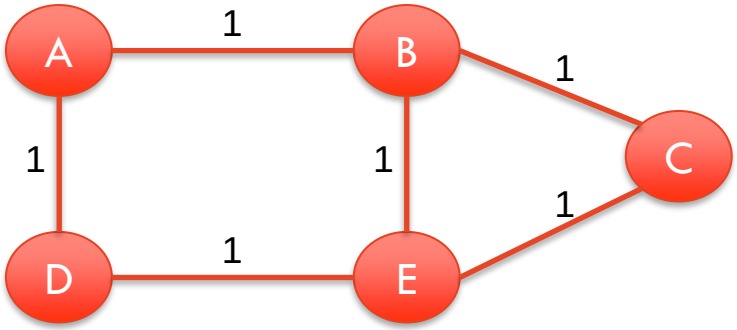
Router Information Protocol (RIP)

Dest	Dir	cost
A	local	0
B	B	1
C	B	2
D	D	1
E	B	2

Table of router A

Dest	Dir	cost
A	A	1
B	local	0
C	C	1
D	A	2
E	E	1

Table of router B



Dest	Dir	cost
A	B	2
B	B	1
C	local	0
D	E	2
E	E	1

Table of router C

Dest	Dir	cost
A	A	1
B	A	2
C	E	2
D	local	0
E	E	1

Table of router D

Dest	Dir	cost
A	B	2
B	B	1
C	C	1
D	D	1
E	local	0

Table of router E

Load balancing occurs when a router has several equal-cost paths to the same destination. E.g., A to E (via B or D)

What happens if the link A-B fails?

- Set the cost of the failure link to ***inf***
- Network Reconvergence

Summary: RIP algorithm

- Advantages:
  - Simple
  - Efficient in small networks

- Limitations:
  - Costs based on the number of hops is unrealistic (bandwidth of links counts)
  - No big deal: this is just a matter of defining link weights (support negative values)
  - Inefficient in large networks as loops may occur before the convergence state is reached

Exercise 2

Lab 2 – COMP3221

THE UNIVERSITY OF SYDNEY

Exercise 2

Dijkstra's Shortest Path Algorithm

Recap:

- A link-state routing algorithm
- Find the shortest path from a node (called the "source node") to all other nodes in the graph
  - Produce a shortest path tree
  - A subpath is also the shortest path between its source and destination

Image Source: https://www.programiz.com/dsa/dijkstra-algorithm

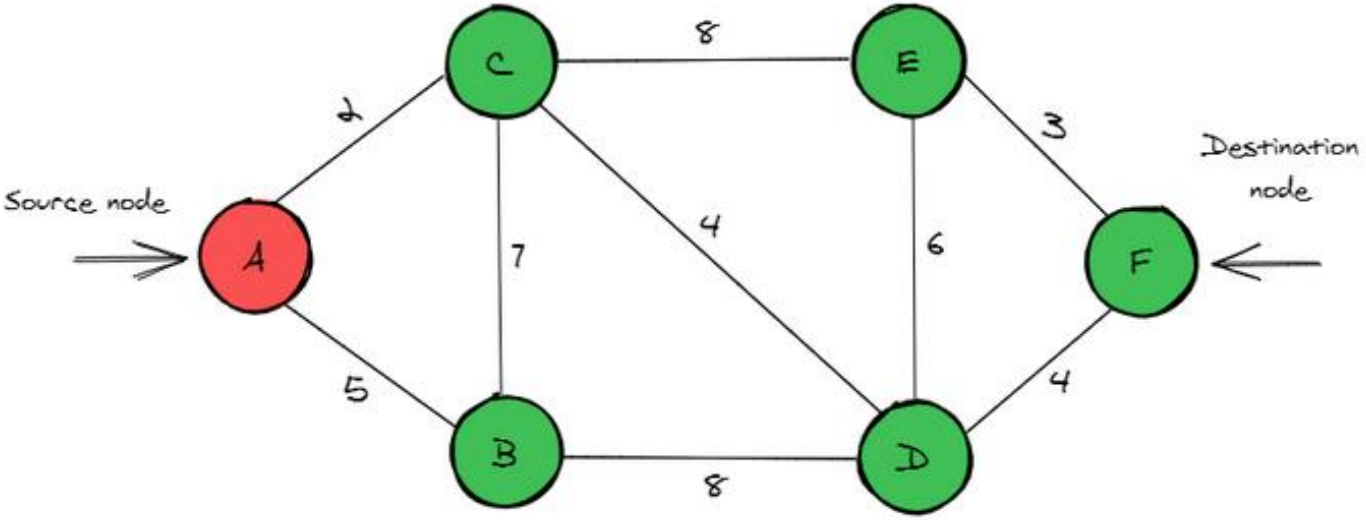
Dijsktra's algorithm

Notation:

- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

```
1  Initialization:
2    N' = {u}
3    for all nodes v
4      if v adjacent to u
5          then D(v) = c(u,v)
6      else D(v) = ∞
7
8  Loop
9     find w not in N' such that D(w) is a minimum
10    add w to N'
11    update D(v) for all v adjacent to w and not in N' :
12       D(v) = min( D(v), D(w) + c(w,v) )
13    /* new cost to v is either old cost to v or known
14     shortest path cost to w plus cost from w to v */
15 until all nodes in N'
```

Dijsktra's algorithm



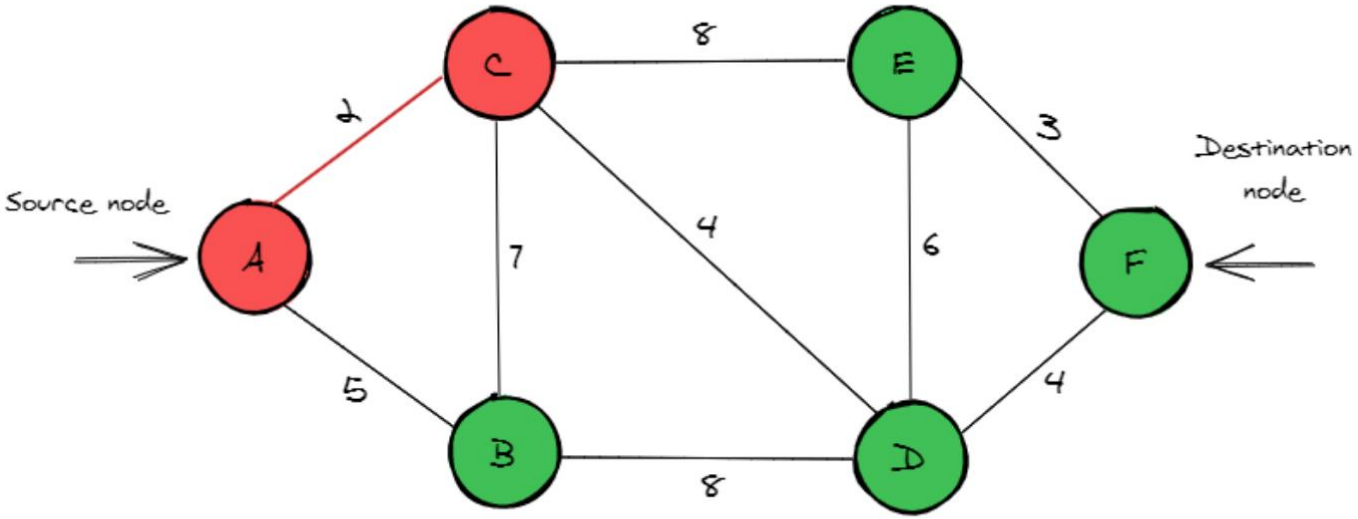
Shortest Path: A

Visited	Unvisited
	~~A~~
	B
	C
	D
	E
A	F

Shortest Distances

A	B	C	D	E	F
0	5	2	inf	inf	inf

Dijsktra's algorithm



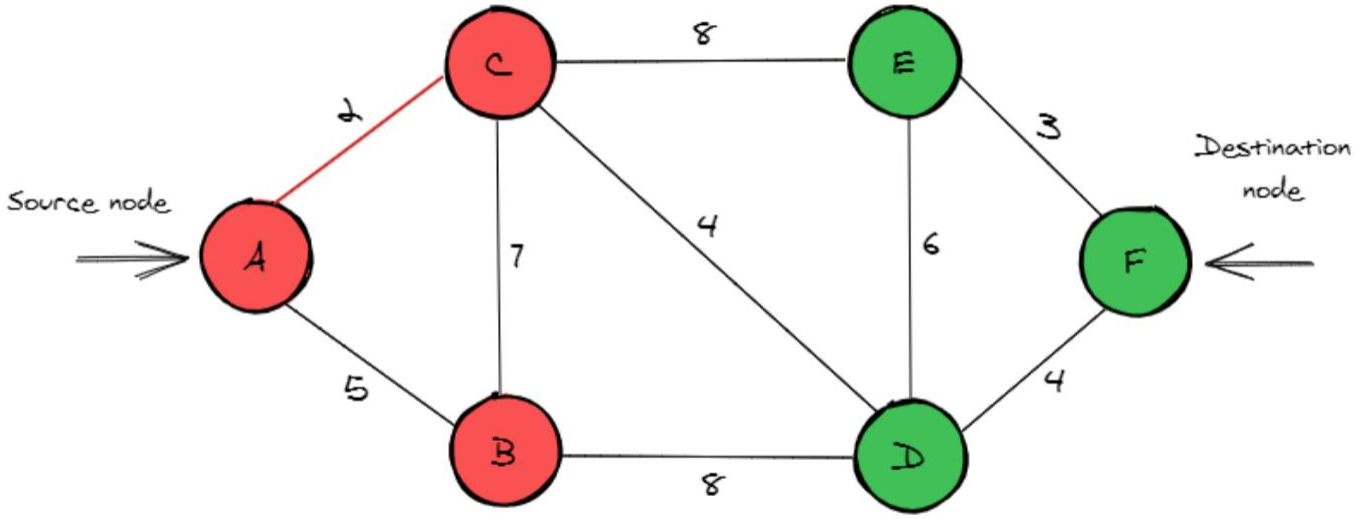
Visited	Unvisited
	~~A~~
	B
	~~C~~
	D
C	E
A	F

Shortest Distances

A	B	C	D	E	F
0	5	2	6	10	inf

Shortest Path: A → C

Dijsktra's algorithm



Shortest Path: A → C

Visited

B
C
A

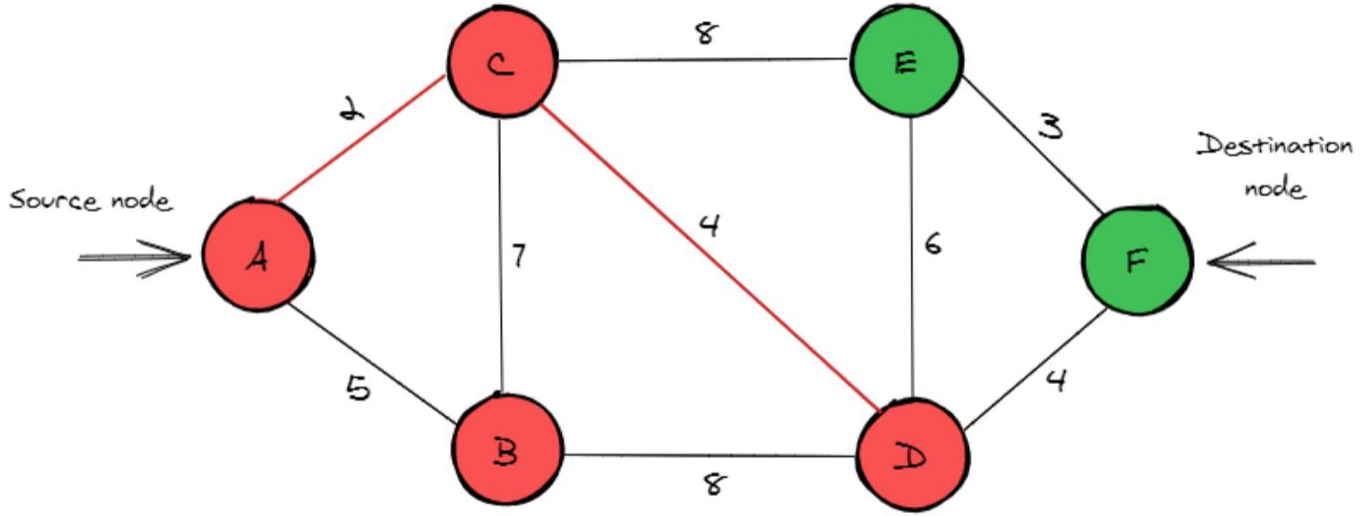
Unvisited

~~A~~
~~B~~
~~C~~
D
E
F

Shortest Distances

A	B	C	D	E	F
0	5	2	6	10	inf

Dijsktra's algorithm



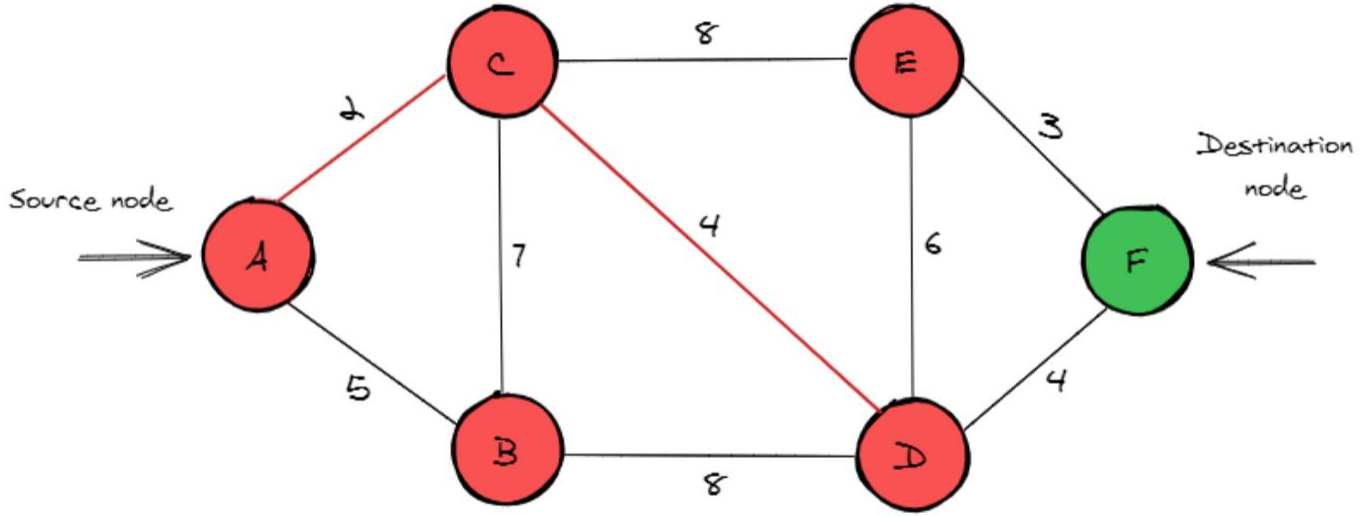
Visited	Unvisited
	~~A~~
	~~B~~
D	~~C~~
B	~~D~~
C	E
A	F

Shortest Distances

A	B	C	D	E	F
0	5	2	6	10	10

Shortest Path: A → C → D

Dijsktra's algorithm



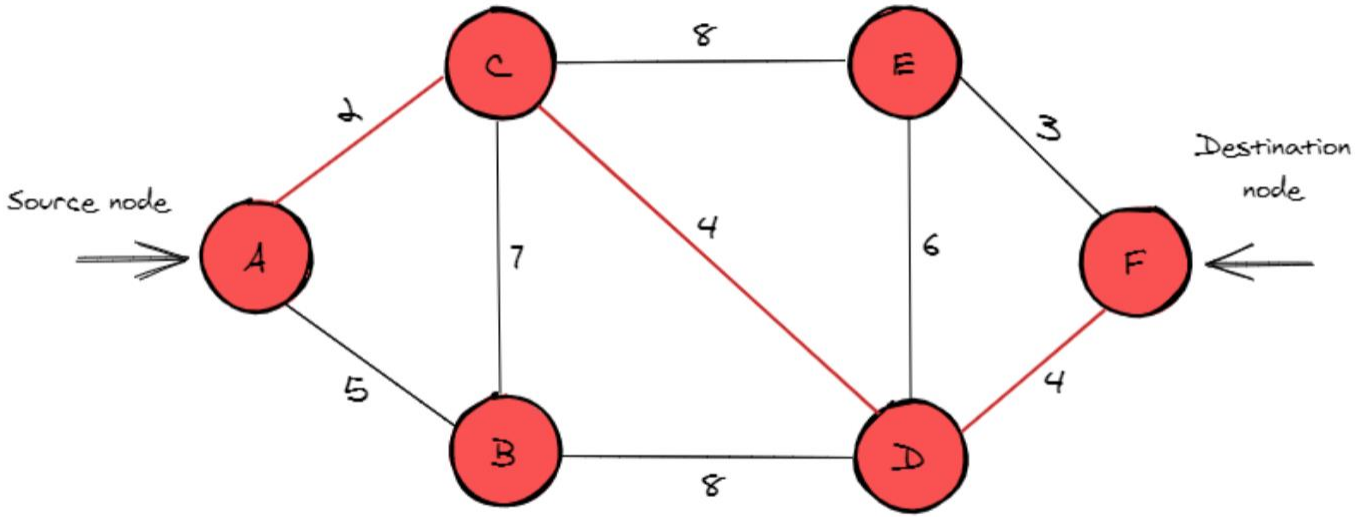
Visited	Unvisited
	~~A~~
~~E~~	~~B~~
D	~~C~~
B	~~D~~
C	~~E~~
A	F

Shortest Distances

A	B	C	D	E	F
0	5	2	6	10	10

Shortest Path: A → C → D

Dijsktra's algorithm



Visited	Unvisited
F	~~A~~
E	~~B~~
D	~~C~~
B	~~D~~
C	~~E~~
A	~~F~~

Shortest Distances

A	B	C	D	E	F
0	5	2	6	10	10

Shortest Path: A → C → D → F

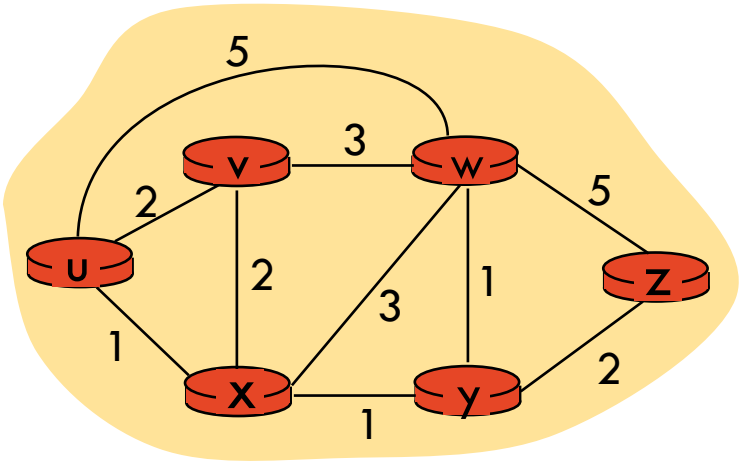
Exercise 2

Dijkstra's Shortest Path Algorithm

❑ **Our problem:**

Complete (by hand) the Table 1 and find the shortest path from node *u* to each other node in the network following Dijkstra's algorithm. Link costs are given next to each edge. In Table 1, *D*(*v*) represents the current cost of path from source to destination *v* and *p*(*v*) represents predecessor node along path from source to *v*.

Step	Node	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0						
1						
2						
3						
4						
5						
Result						

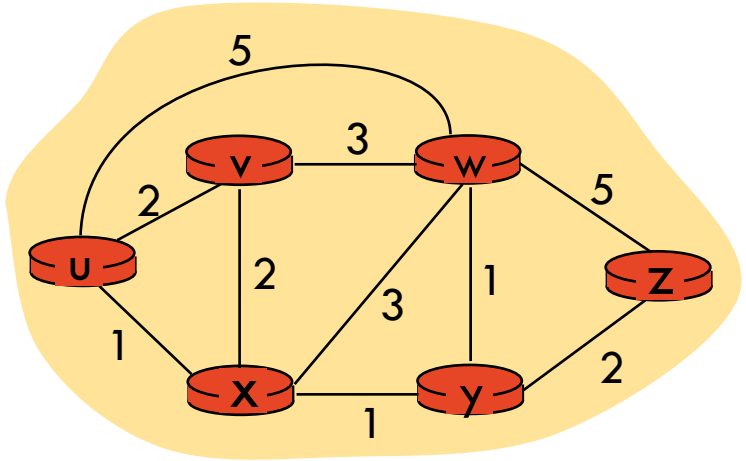


What is the shortest-path from node u to node z ?

Exercise 2: Dijkstra's algorithm

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					
Result		**2,u**	**3,y**	**1,u**	**2,x**	**4,y**

```
8  Loop
9    find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for all v adjacent to w and not in N'
12     D(v) = min( D(v), D(w) + c(w,v) )
13   /* new cost to v is either old cost to v or known
14    shortest path cost to w plus cost from w to v */
15 until all nodes in N'
```



D(v): current value of cost of path from source to dest. v

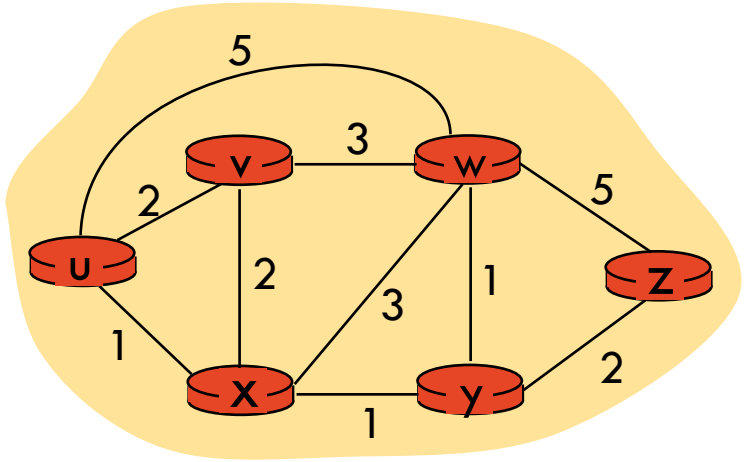
Exercise 2: Dijkstra's algorithm

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					
Result		**2,u**	**3,y**	**1,u**	**2,x**	**4,y**

Resulting shortest-path to z:

- **z ← y ← x ←u**

```
8  Loop
9    find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for all v adjacent to w and not in N'
12     D(v) = min( D(v), D(w) + c(w,v) )
13   /* new cost to v is either old cost to v or known
14    shortest path cost to w plus cost from w to v */
15 until all nodes in N'
```



Summary: Dijkstra's algorithm

Algorithm complexity: n nodes

- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)

Exercise 3

Lab 2 – COMP3221

THE UNIVERSITY OF SYDNEY

Exercise 3

Path calculation in large scale networks

- Many graph libraries are available
  - Python: NetworkX - https://networkx.github.io
- Efficient implementations for many commonly known graph algorithms are available
- Installation instructions for NetworkX can be found at https://networkx.github.io/documentation/stable/install.html
- Example: Dijkstra's algorithm

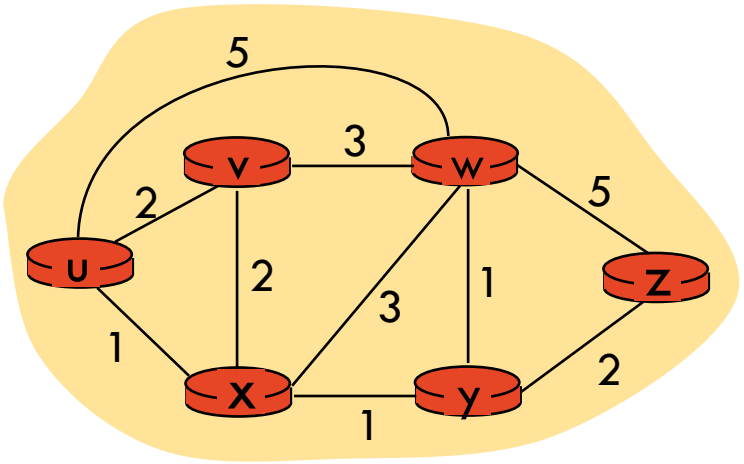
```
>>> G = nx.path_graph(5)
>>> print(nx.shortest_path(G, source=0, target=4))
[0, 1, 2, 3, 4]
>>> p = nx.shortest_path(G, source=0) # target not specified
>>> p[4]
[0, 1, 2, 3, 4]
>>> p = nx.shortest_path(G, target=4) # source not specified
>>> p[0]
[0, 1, 2, 3, 4]
>>> p = nx.shortest_path(G) # source, target not specified
>>> p[0][4]
[0, 1, 2, 3, 4]
```

Exercise 3

Path calculation in large scale networks

- **Our problem:**

Create the same network in Figure 2 using the sample code below




```
1  import networkx as nx;
2
3  G = nx.Graph()
4  G.add_edge('u','v',weight=2)
5  G.add_edge('u','w',weight=5)
6  // complete all edges
```

Find (and verify with the result of Exercise 2) the shortest path from node *u* to node *z* using the NetworkX shortest path algorithm **shortest_path.**

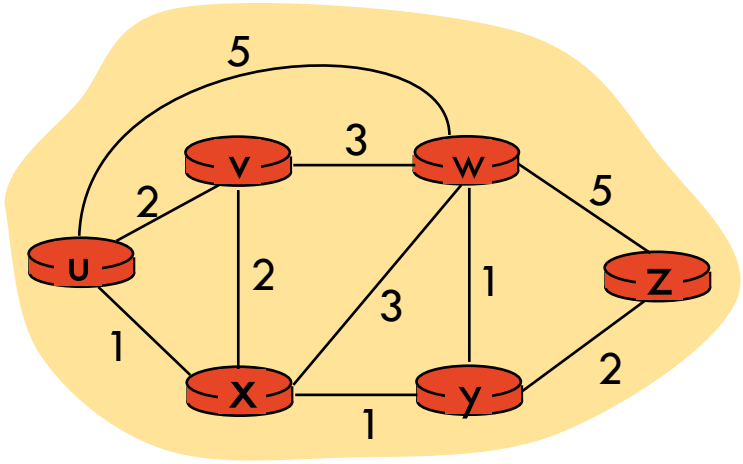
- What is the total cost of the shortest path ?
- What is the shortest path if all edge costs are equal to 1 ?

Exercise 3

Path calculation in large scale networks

- **Our problem:**

Create the same network in Figure 2 using the sample code below




```
1  import networkx as nx;
2
3  G = nx.Graph()
4  G.add_edge('u','v',weight=2)
5  G.add_edge('u','w',weight=5)
6  G.add_edge('u','x',weight=1)
7  G.add_edge('v','x',weight=2)
8  G.add_edge('v','w',weight=3)
9  G.add_edge('x','w',weight=3)
10 G.add_edge('x','y',weight=1)
11 G.add_edge('y','w',weight=1)
12 G.add_edge('y','z',weight=2)
13 G.add_edge('w','z',weight=5)
```

Exercise 3

Path calculation using NetworkX

- **Our problem:**

Create the same network in Figure 2 using the sample code below

Find (and verify with the result of Exercise 2) the shortest path from node *u* to node *z* using the NetworkX shortest path algorithm **shortest_path.**

```
>>> nx.shortest_path(G, source='u', target='z', weight='weight')
>>> ['u', 'x', 'y', 'z']
```

- What is the total cost of the shortest path ?

```
>>> nx.shortest_path_length(G, source='u', target='z', weight='weight')
>>> 4
```

- What is the shortest path if all edge costs are equal to 1 ?

```
>>> nx.shortest_path(G, source='u', target='z')
>>> ['u', 'w', 'z']
```