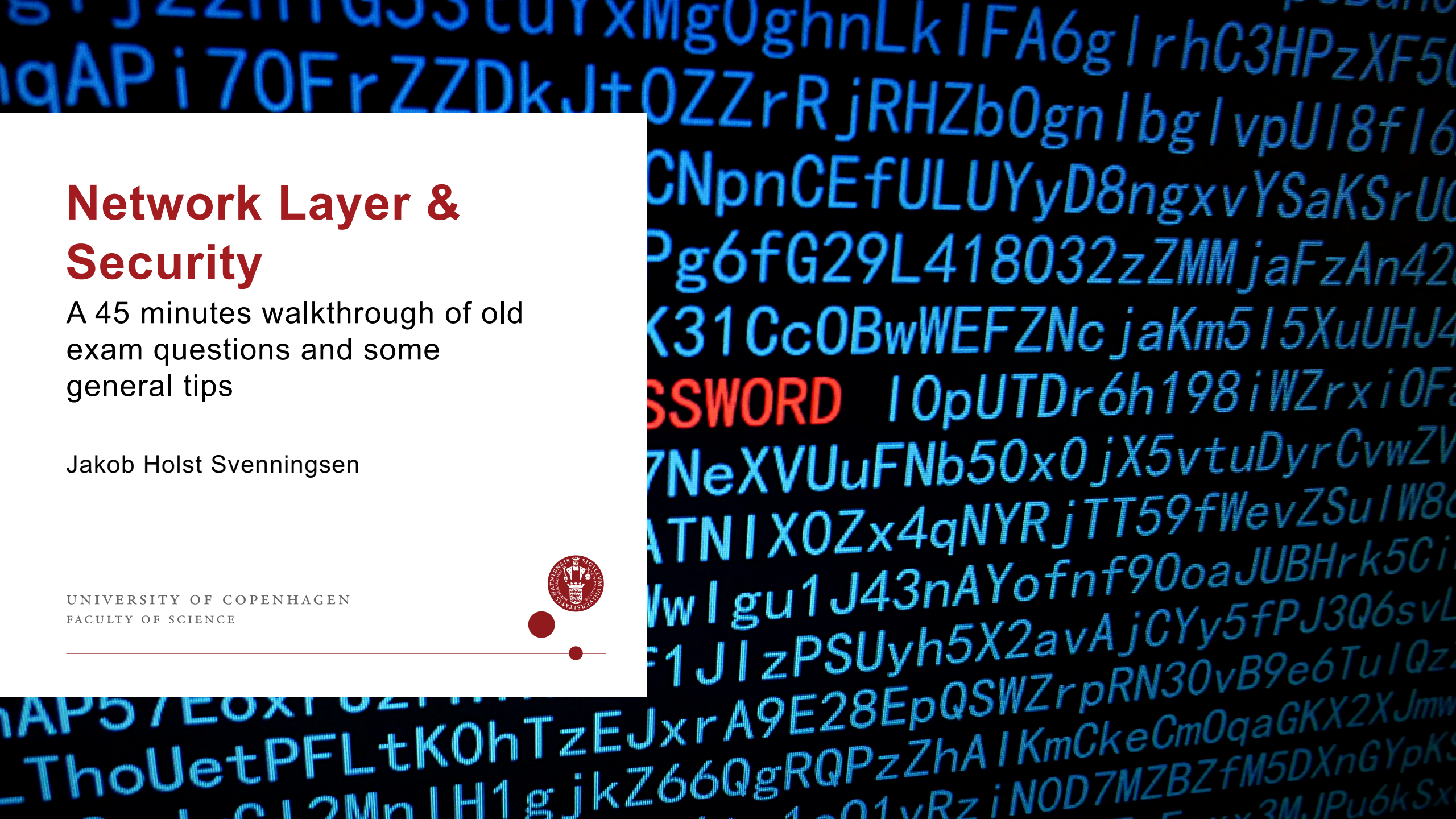
Network Layer & Security

A 45 minutes walkthrough of old exam questions and some general tips

Jakob Holst Svenningsen

UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE



Network Layer

Good things to remember about the Network Layer

- Routers forward datagrams
- Network Protocol used is **IP** (IPv4 and IPv6)
- Network control plane
  - Controls forwarding process (Routing)
- Network data plane
  - Performs actual forwarding (Forwarding)
- Routing Algorithms
  - calculates path from sender to receiver
- Network Service Model
  - Best-effort service
  - Encryption is possible at network layer (e.g. IPSec)

- In FW table longest prefix matching is performed
- Packet loss can occur in the router, when there is no memory left for incoming packets
- **Subnet** e.g. 223.1.1.0/24 means hosts here have address 223.1.1.xxx (first 24 bits are fixed)
- **DHCP** is an app-layer protocol for automatic assignment of IP-addresses
- **Network Address Translation (NAT)** solves problem with number of available IPs

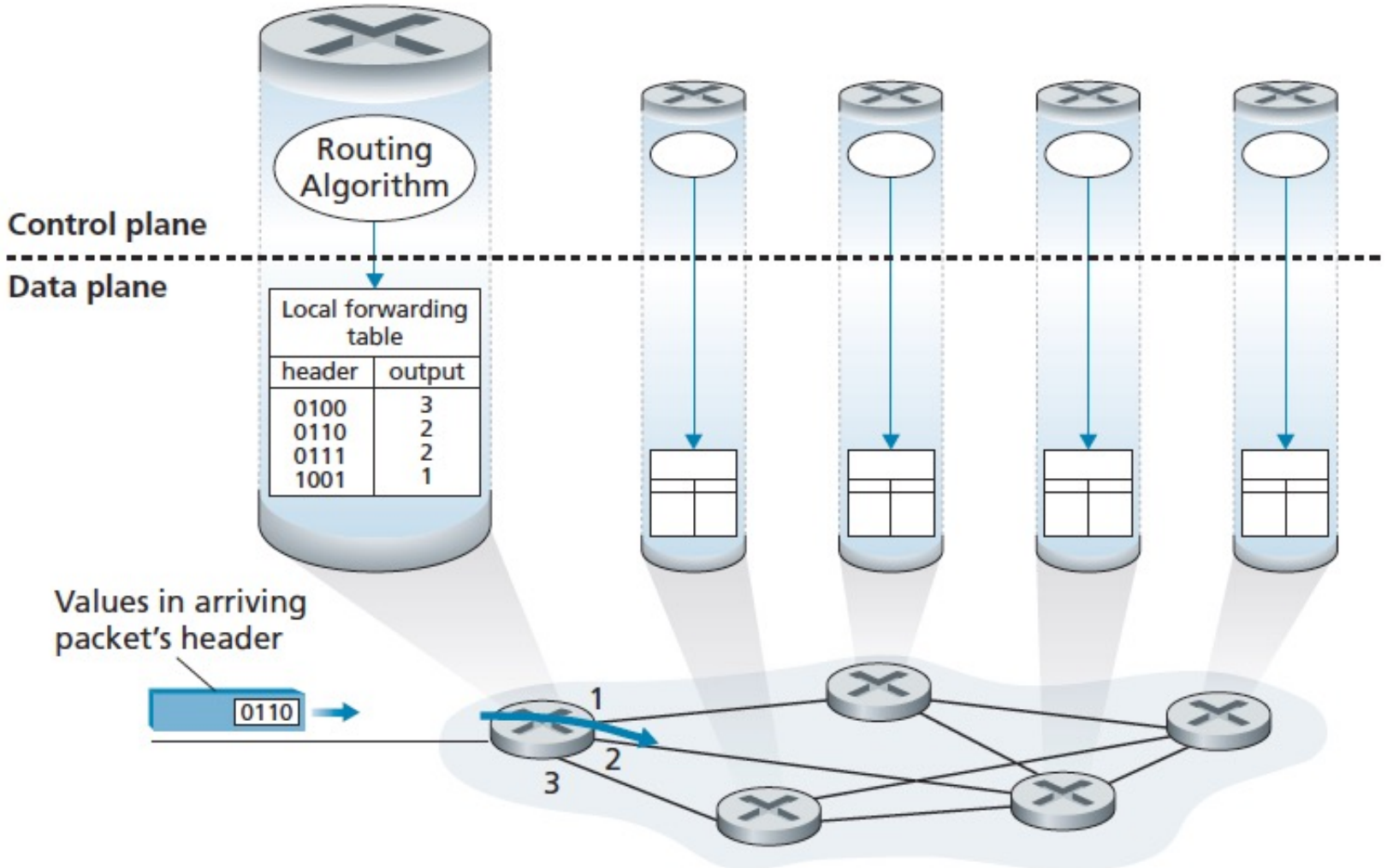


Figure 4.2 ♦ Routing algorithms determine values in forward tables

Control plane is responsible for routing, while the actual forwarding happens at the data plane

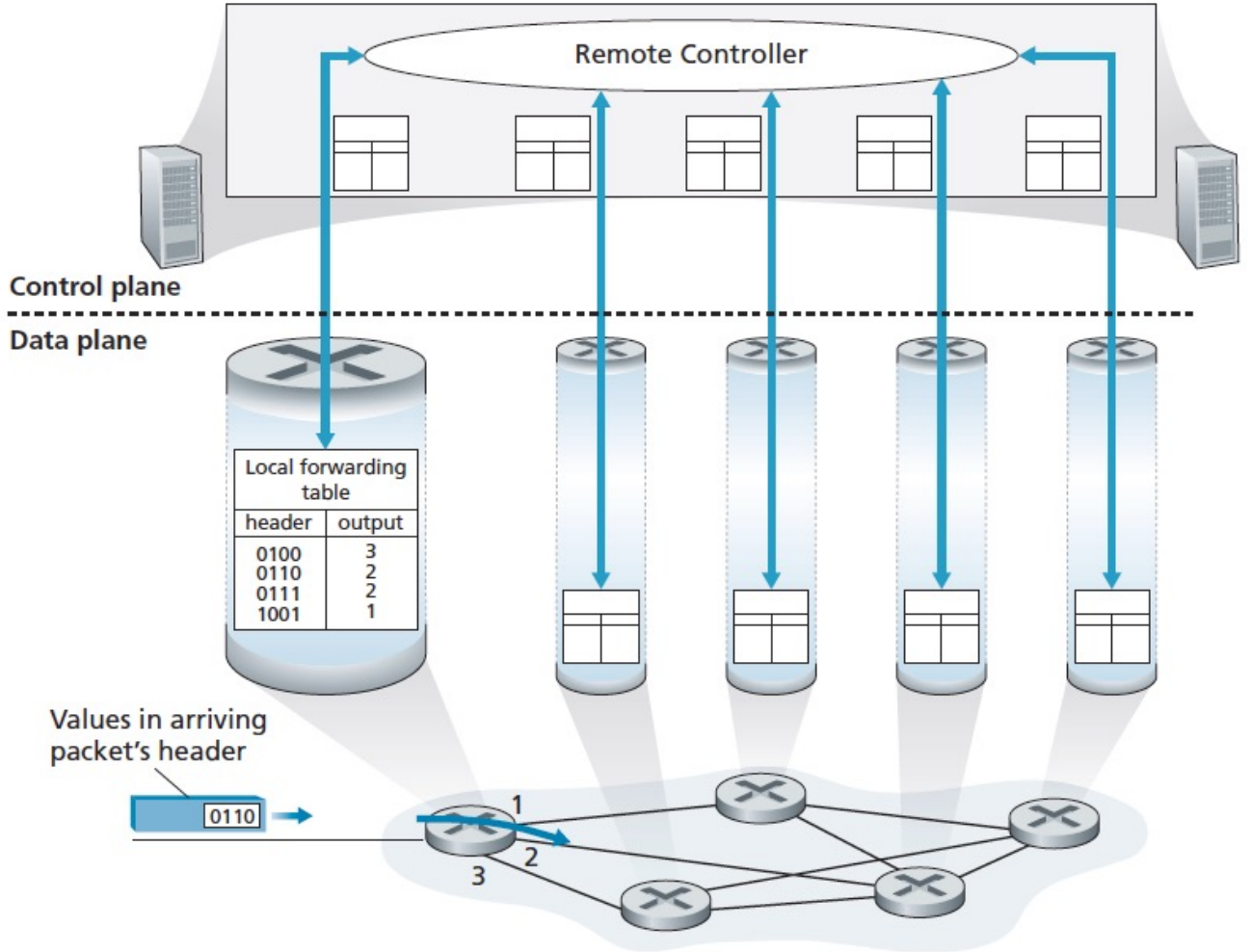


Figure 4.3 ♦ A remote controller determines and distributes values in forwarding tables

Today, the routing is often implemented in software and a centralized server (Remote Controller) tells the router how datagrams should be forwarded. It is however, possible to manually alter a forwarding table in each router.

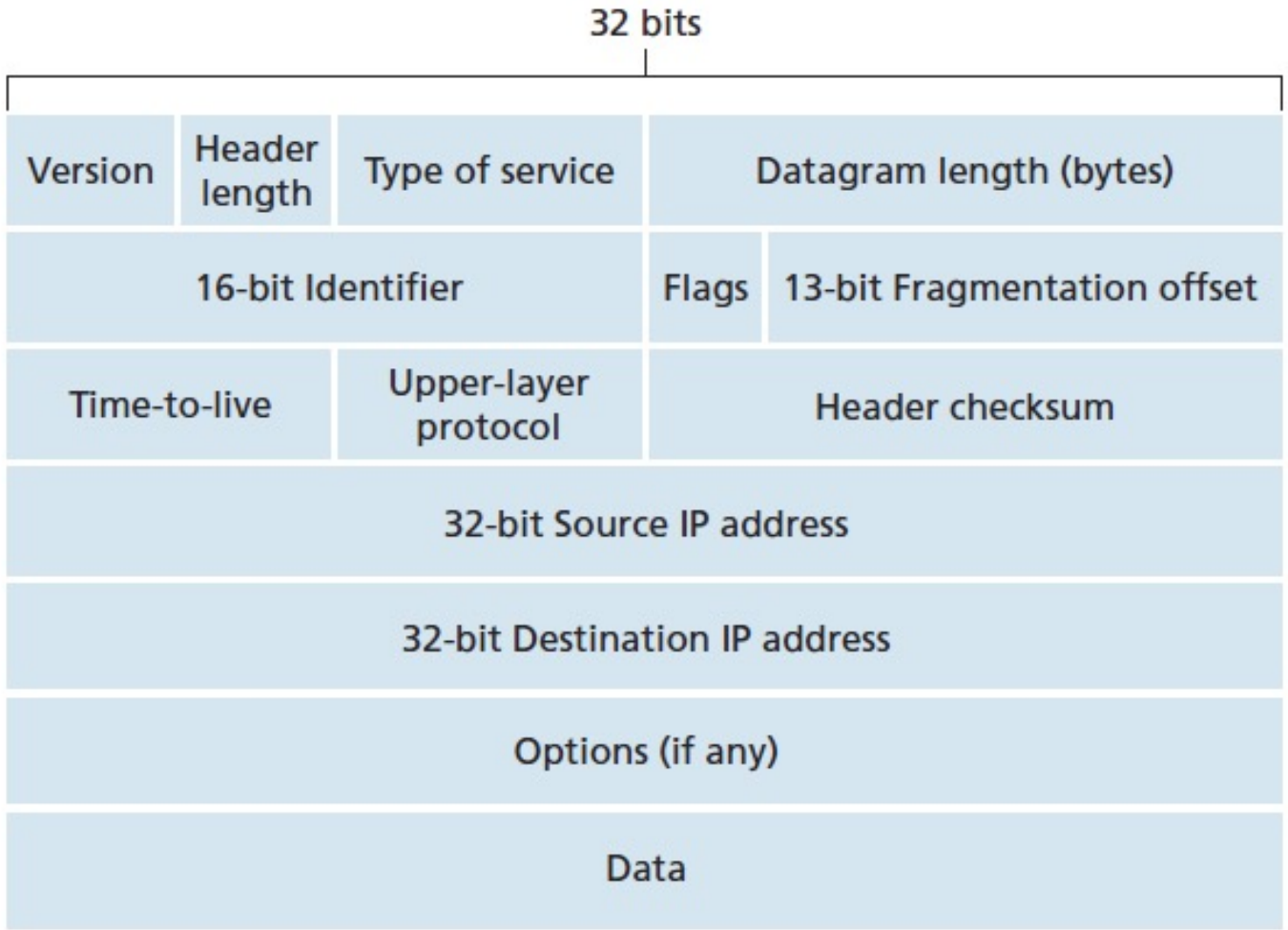


Figure 4.17 ♦ IPv4 datagram format

The IP-datagram is just yet another protocol that looks like this

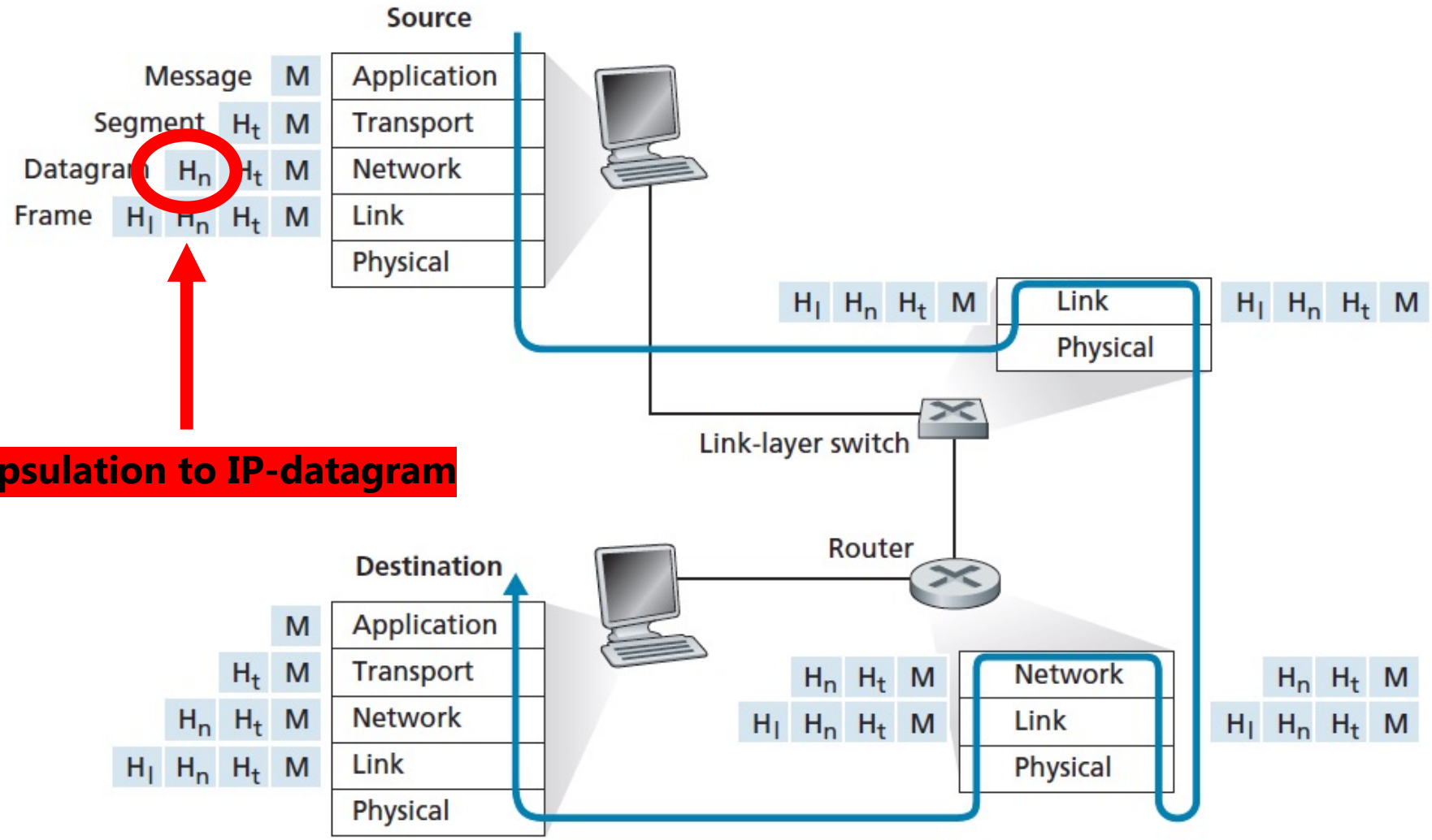


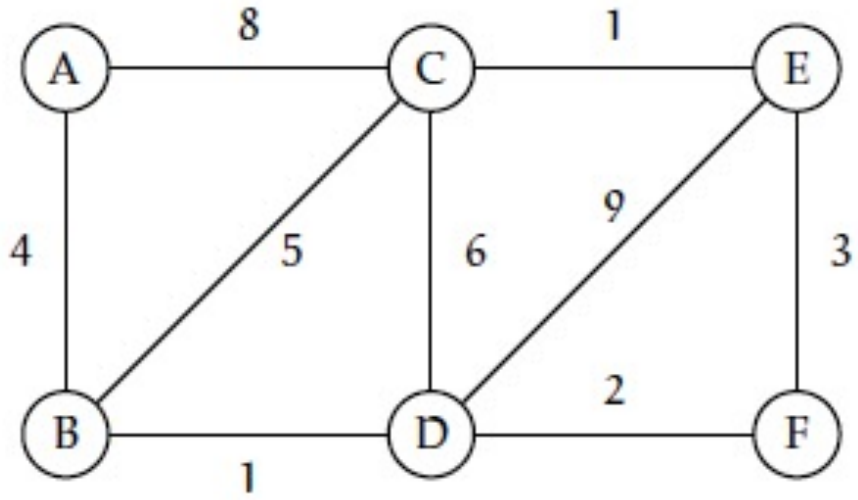
Figure 1.24 ♦ Hosts, routers, and link-layer switches; each contains a different set of layers, reflecting their differences in functionality

In a broader context, higher level packets are encapsulated to IP-datagrams at the network layer. At the destination, we "unwrap" layer by layer. This abstraction allows arbitrary Application layer protocols to rely on IP

Link-State (LS) Routing Algorithm – Reexam 2020-21

- Centralized Routing Algorithm
- Essentially Dijkstra's shortest paths algorithm

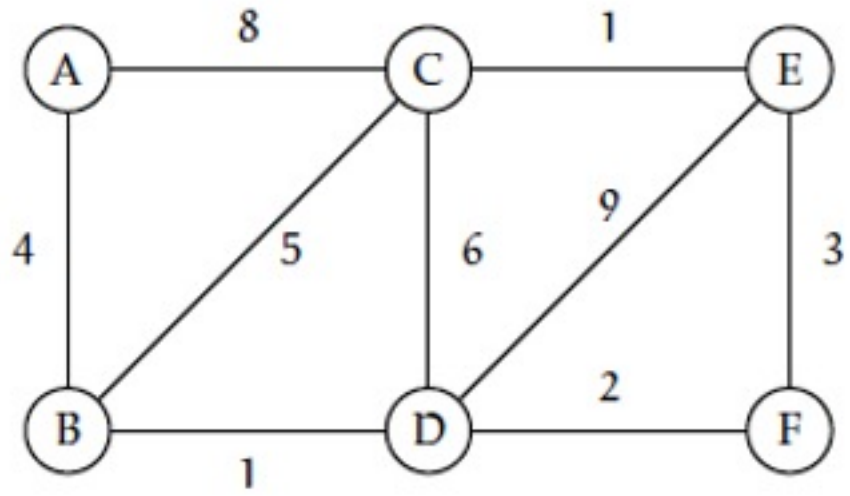
Question 3.3.1: Consider the network topology outlined in the graph below.



Apply the link state routing algorithm and compute the forwarding table on node **A** by filling out the following tables.

Solution Method

Here, **A**, is the src



- D(v): Cost of least path from **u** (source) to **v** in this iteration
- p(v): Previous node of **v** along least-cost path
- N': subset of nodes, that are part of shortest path

Initialization:

```
N' = {u}
for all nodes v
  if v is a neighbor of u
    then D(v) = c(u,v)
  else D(v) = ∞
```

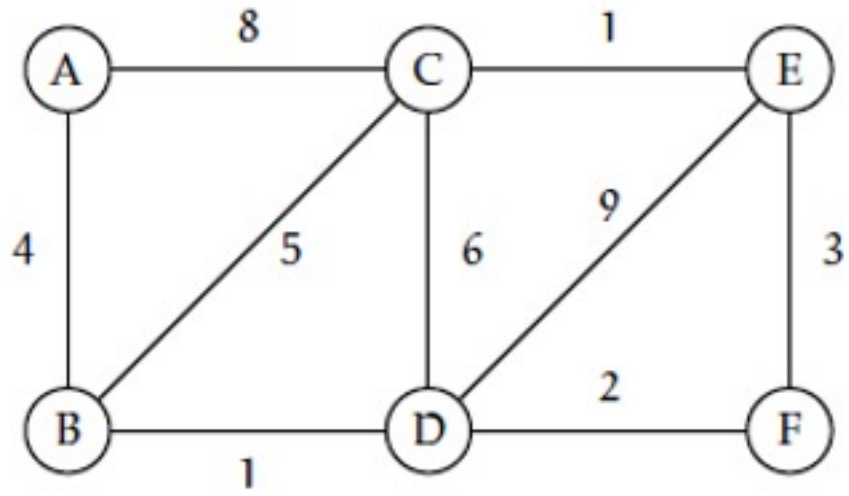
Steps of the algorithm:

Step	N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	**A**	**4,A**	**8,A**	**inf**	**inf**	**inf**
1						
2						
3						
4						
5						

Forwarding table on node A:

Solution Method

Here, **A**, is the src




```
Loop
 find w not in N' such that D(w) is a minimum
 add w to N'
 update D(v) for each neighbor v of w and not in N':
       D(v) = min(D(v), D(w)+ c(w,v) )
  /* new cost to v is either old cost to v or known
   least path cost to w plus cost from w to v */
until N'= N
```

1: If we choose B as w, then D(w) is minimum

2: Iterate over all neighbours of w (which is B here) that are not in the subset N'

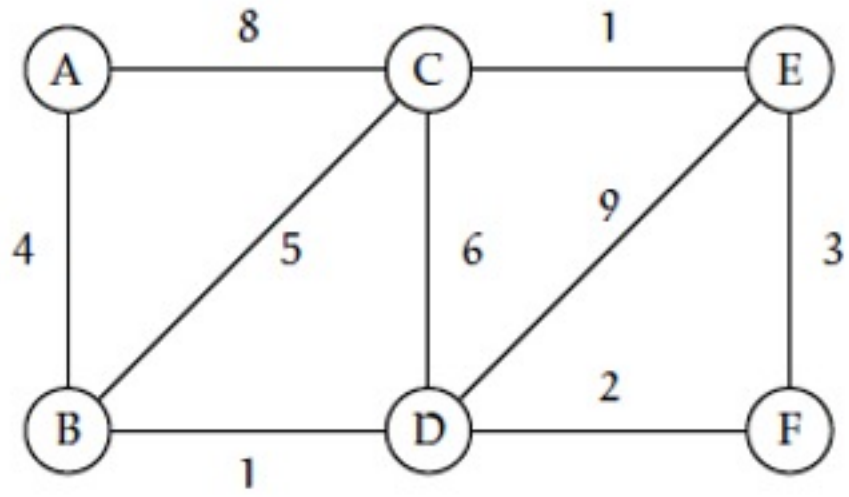
Steps of the algorithm:

Step	N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	4,A	8,A	inf	inf	inf
1	A,B	4,A	8,A	5,B	inf	inf
2						
3						
4						
5						

Forwarding table on node A:

Solution Method

Here, **A**, is the src




```
Loop
 find w not in N' such that D(w) is a minimum
 add w to N'
 update D(v) for each neighbor v of w and not in N':
       D(v) = min(D(v), D(w)+ c(w,v) )
  /* new cost to v is either old cost to v or known
   least path cost to w plus cost from w to v */
until N'= N
```

1: If we choose D as w, then D(w) is minimum

2: Iterate over all neighbours of w (which is D here) that are not in the subset N'

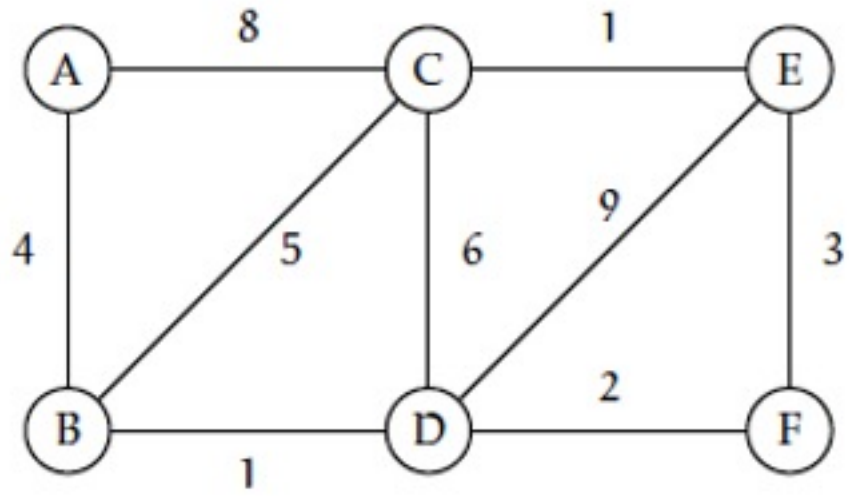
Steps of the algorithm:

Step	N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	4,A	8,A	inf	inf	inf
1	A,B	4,A	8,A	5,B	inf	inf
2	A,B,D	4,A	8,A	5,B	14,D	7,D
3						
4						
5						

Forwarding table on node A:

Solution Method

Here, **A**, is the src




```
Loop
 find w not in N' such that D(w) is a minimum
 add w to N'
 update D(v) for each neighbor v of w and not in N':
       D(v) = min(D(v), D(w)+ c(w,v) )
  /* new cost to v is either old cost to v or known
   least path cost to w plus cost from w to v */
until N'= N
```

1: If we choose F as w, then D(w) is minimum

2: Iterate over all neighbours of w (which is F here) that are not in the subset N'.

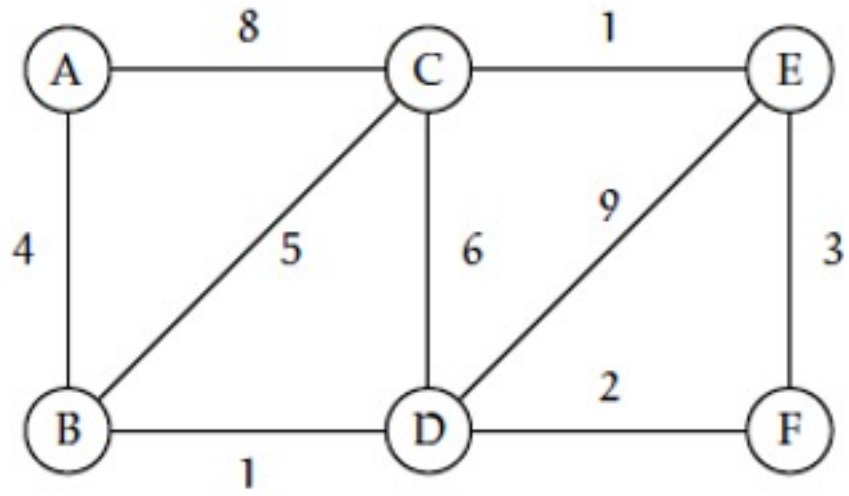
Steps of the algorithm:

Step	N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	4,A	8,A	inf	inf	inf
1	A,B	4,A	8,A	5,B	inf	inf
2	A,B,D	4,A	8,A	5,B	14,D	7,D
3	A,B,D,F	4,A	8,A	5,B	10,F	7,D
4						
5						

Forwarding table on node A:

Solution Method

Here, **A**, is the src




```
Loop
 find w not in N' such that D(w) is a minimum
 add w to N'
 update D(v) for each neighbor v of w and not in N':
       D(v) = min(D(v), D(w)+ c(w,v) )
  /* new cost to v is either old cost to v or known
   least path cost to w plus cost from w to v */
until N'= N
```

1: If we choose C as w, then D(w) is minimum

2: Iterate over all neighbours of w (which is C here) that are not in the subset N'.

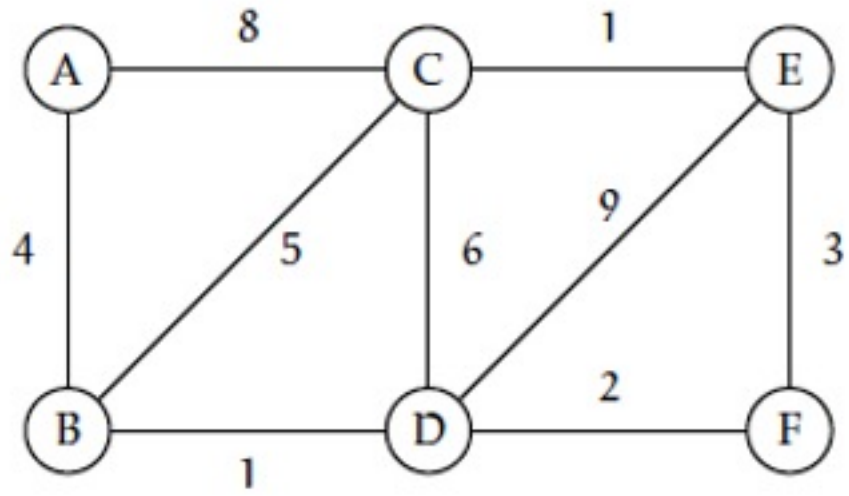
Steps of the algorithm:

Step	N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	4,A	8,A	inf	inf	inf
1	A,B	4,A	8,A	5,B	inf	inf
2	A,B,D	4,A	8,A	5,B	14,D	7,D
3	A,B,D,F	4,A	8,A	5,B	10,F	7,D
4	A,B,D,F,C	4,A	8,A	5,B	9,C	7,D
5						

Forwarding table on node A:

Solution Method

Here, **A**, is the src




```
Loop
 find w not in N' such that D(w) is a minimum
 add w to N'
 update D(v) for each neighbor v of w and not in N':
       D(v) = min(D(v), D(w)+ c(w,v) )
  /* new cost to v is either old cost to v or known
   least path cost to w plus cost from w to v */
until N'= N
```

1: If we choose E as w, then D(w) is minimum

2: Iterate over all neighbours of w (which is E here) that are not in the subset N'.

Steps of the algorithm:

Step	N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	4,A	8,A	inf	inf	inf
1	A,B	4,A	8,A	5,B	inf	inf
2	A,B,D	4,A	8,A	5,B	14,D	7,D
3	A,B,D,F	4,A	8,A	5,B	10,F	7,D
4	A,B,D,F,C	4,A	8,A	5,B	9,C	7,D
5	A,B,D,F,C,E	4,A	8,A	5,B	9,C	7,D

Forwarding table on node A:

Solution Method

- To arrive at a final solution, we see that if we stand at node **A** then:
  - Shortest path from **A** to **B** is by taking edge **(A,B)** and here the total cost is 4
  - Shortest path from **A** to **C** is by taking edge **(A,C)** and here the total cost is cost 8
  - Shortest path from **A** to **D** is by taking edge **(A,B)** and here the total cost is cost 5
  - Shortest path from **A** to **E** is by taking edge **(A,C)** and here the total cost is cost 9
  - Shortest path from **A** to **F** is of cost 7. We see that the previous node on this path is **D** and we know that the shortest path to **D** is by taking edge **(A,B).** Hence shortest path from **A** to **F** is by taking edge **(A,B)**.

Forwarding table on node A:

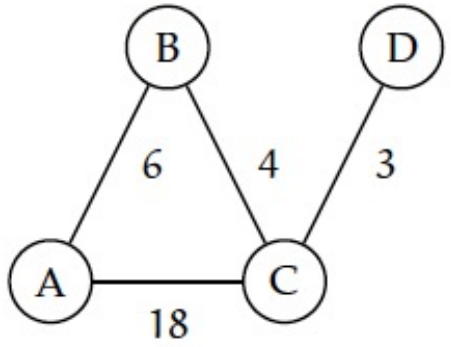
Destination node	Edge
B	(A,B)
C	(A,C)
D	(A,B)
E	(A,C)
F	(A,B)

Remember that 'edge' here means standing at **A** in which direction should you go such that you end up taking the shortest path to the destination node

Distance-Vector (DV) Routing Algorithm – Exam 2021-22

- De-centralized Routing Algorithm
- An iterative, asynchronous, and distributed algorithm
- Information propagates in the network
  - Hence, we need to iterate, until information has propagated

Consider the network topology outlined in the graph below



Question 3.3.1: Describe how to apply the Distance Vector Algorithm for Node A in the above diagram, so that it gets a complete routing table to all other Nodes. You do not need to describe each individual step on each Node in detail, but should be able to provide a generalised description that could apply to any Node. For this answer you can assume no connection costs change.

Node A table

	A	B	D	D
A				
B				
C				
D				

Node B table

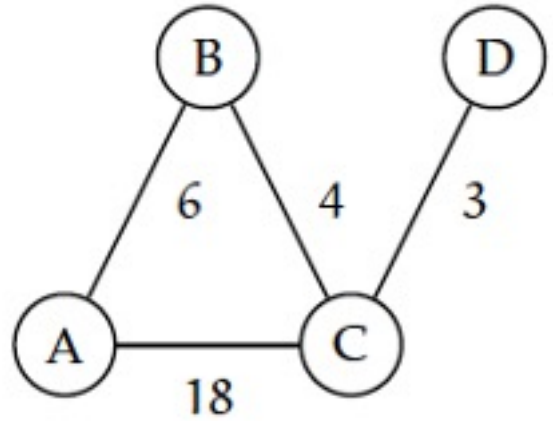
	A	B	D	D
A				
B				
C				
D				

Node C table

	A	B	D	D
A				
B				
C				
D				

Node D table

	A	B	D	D
A				
B				
C				
D				



Initialization:

```
for all destinations y in N:
   Dx(y)= c(x,y)/* if y is not a neighbor then c(x,y)= ∞ */
for each neighbor w
   Dw(y) = ? for all destinations y in N
for each neighbor w
   send distance vector  Dx = [Dx(y): y in N] to w
```

- **Initialize distance vector for each node with distance to reachable neighbours**
- **Distance to self is simply 0**
- **If a neighbour is unreachable, initialize to infinity**

Node A table

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table

	A	B	C	D
A				
B				
C				
D				

Node C table

	A	B	C	D
A				
B				
C				
D				

Node D table

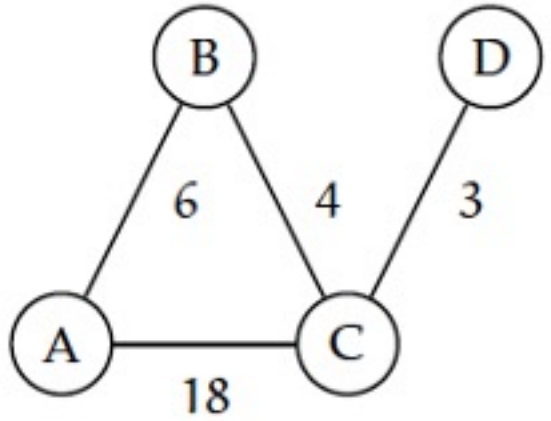
	A	B	C	D
A				
B				
C				
D				

Initializing distance vector for A

- From A we can immediately reach B (cost 6) and C (cost 18)
- D is not immediately reachable, and hence initialized to inf (Abbrevated IF)
- At this point we do not know what the Distance Vectors of A's neighbours are, so these are simply inifitialized to IF
- Remember that the algorithm is run for each node asynchronously
  - Therefore this procedure has to be repeated for both A,B,C and D

Initialization:

```
for all destinations y in N:
   Dx(y)= c(x,y)/* if y is not a neighbor then c(x,y)= ∞ */
for each neighbor w
   Dw(y) = ? for all destinations y in N
for each neighbor w
   send distance vector  Dx = [Dx(y): y in N] to w
```



- Initialize distance vector for each node with distance to reachable neighbours
- Distance to self is simply 0
- If a neighbour is unreachable, initialize to infinity

Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

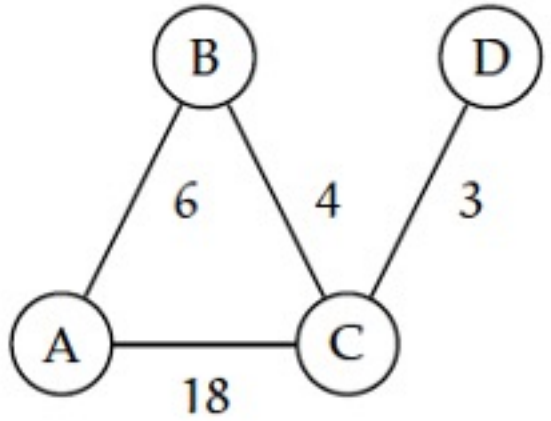
	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

We initialize B, D and C in the same manner

- From B we can reach A (cost 6) and C (cost 4)
- From C we can reach A (cost 18), B (cost 4) and D (cost 3)
- From D we can reach C (cost 3)

Initialization:

```
for all destinations y in N:
   Dx(y)= c(x,y)/* if y is not a neighbor then c(x,y)= ∞ */
for each neighbor w
   Dw(y) = ? for all destinations y in N
for each neighbor w
   send distance vector  Dx = [Dx(y): y in N] to w
```



- **Initialize distance vector for each node with distance to reachable neighbours**
- **Distance to self is simply 0**
- **If a neighbour is unreachable, initialize to infinity**

Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

Node A table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 1)

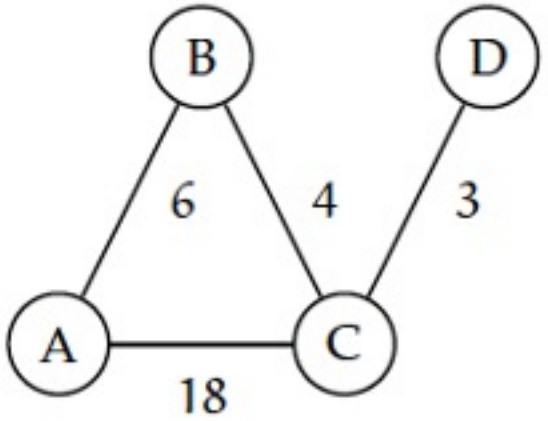
	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Node D table (Iter 1)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Send distance vector to neighbours

- A informs B & C
- B informs A & C
- C informs A, B & D
- D informs C



Initialization:

```
for all destinations y in N:
   Dx(y)= c(x,y)/* if y is not a neighbor then c(x,y)= ∞ */
for each neighbor w
   Dw(y) = ? for all destinations y in N
for each neighbor w
   send distance vector  Dx = [Dx(y): y in N] to w
```

Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

Node A table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 1)

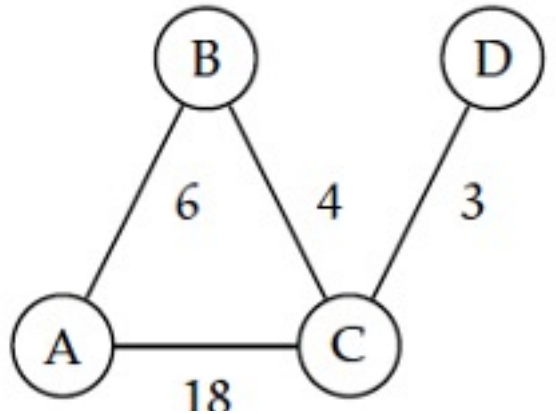
	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Node D table (Iter 1)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Now DVs have been sent to neighbours and we enter the loop

- Both A,B,C and D have indeed received a DV from some neighbour, so therefore we have to execute the **for loop** on both A,B,C and D




```
loop
    wait   (until I see a link cost change to some neighbor w or
              until I receive a distance vector from some neighbor w)

    for each y in N:
        Dx(y) = minv{c(x,v) + Dv(y)}

    if Dx(y) changed for any destination y
        send distance vector Dx = [Dx(y): y in N] to all neighbors

forever
```

Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

Node A table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

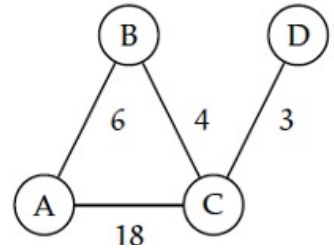
Node D table (Iter 1)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

We execute the for loop for Node A table

- Here we check if there is a shorter path from A to B
- $D_A(B) = min_v\{c(A,v) + D_v(B)\}$

  $= \min\{c(A,B) + D_B(B), c(A,C) + D_C(B)\} = \min\{6+0, 18+4\} = 6$
- In the above, we are essentially checking if we by going through one of our neighbours can obtain a shorter path to B
- $D_A(B)$ did **not** change as the shortest path from A to B was already 6
- We also check if there is a shorter path from A to C and D:




```
loop
    wait   (until I see a link cost change to some neighbor w or
            until I receive a distance vector from some neighbor w)

    for each y in N:
        Dx(y) = minv{c(x,v) + Dv(y)}

    if Dx(y) changed for any destination y
        send distance vector Dx = [Dx(y): y in N] to all neighbors

forever
```

Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

Node A table (Iter 1)

	A	B	C	D
A	**0**	**6**	**10**	**21**
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Node D table (Iter 1)

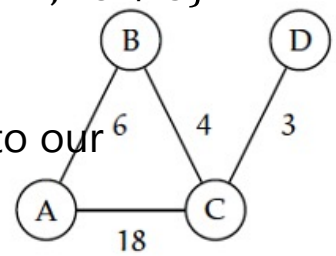
	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

We execute the for loop for Node A table

- **Checking if there is a shorter path from A to C**
- $D_A(C) = min_v\{c(A,v) + D_v(C)\}$

  $= \min\{c(A,B) + D_B(C), c(A,C) + D_C(C)\} = \min\{6+4, 18+0\} = 10$
- Great, we saw that we can go from A to C via B at cost 10!
- **Checking if there is a shorter path from A to D**
- $D_A(D) = min_v\{c(A,v) + D_v(D)\}$

  $= \min\{c(A,B) + D_B(D), c(A,C) + D_C(D)\} = \min\{6+IF, 18+3\} = 21$
- Great, we can go from A to D at cost 21!
- As there were changes, we **send our updated DV** to our neighbors




```
loop
    wait   (until I see a link cost change to some neighbor w or
               until I receive a distance vector from some neighbor w)

    for each y in N:
        Dx(y) = minv{c(x,v) + Dv(y)}

    if Dx(y) changed for any destination y
        send distance vector Dx = [Dx(y): y in N] to all neighbors

forever
```

Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

Node A table (Iter 1)

	A	B	C	D
A	**0**	**6**	**10**	**21**
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Node D table (Iter 1)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Node A table (Iter 2)

	A	B	C	D
A	**0**	**6**	**10**	**21**
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 2)

	A	B	C	D
A	0	6	10	21
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

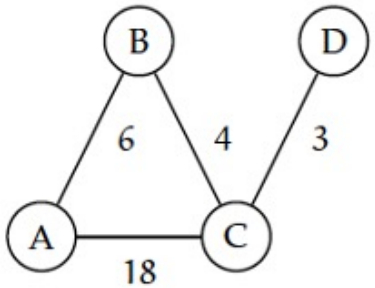
Node C table (Iter 2)

	A	B	C	D
A	A	0	6	10
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Node D table (Iter 2)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

But remember! The algorithm also needs to be run for the B, C and D tables!



Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

Node A table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	**6**	**0**	**4**	**7**
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 1)

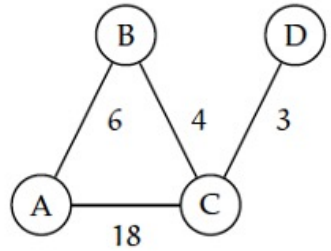
	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Node D table (Iter 1)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Remember that we also have to run the algorithm for the B,C and D tables:

- **Checking if there is a shorter path from B to A**
  - There is not
- **Checking if there is a shorter path from B to C**
  - There is not
- **Checking if there is a shorter path from B to D**
  - There is: By going via C, the cost is 7
- Hence, we need to send our DV to neighbors



Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

Node A table (Iter 1)

	A	B	C	D
A	**0**	**6**	**10**	**21**
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	**6**	**0**	**4**	**7**
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	3	0

Node D table (Iter 1)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Node A table (Iter 2)

	A	B	C	D
A	**0**	**6**	**10**	**21**
B	6	0	4	7
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 2)

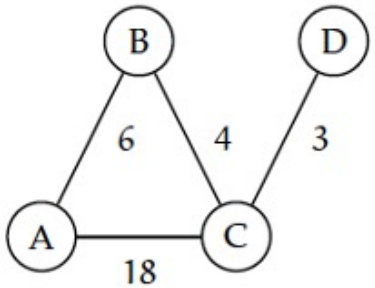
	A	B	C	D
A	0	6	10	21
B	6	0	4	7
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 2)

	A	B	C	D
A	A	0	6	10
B	6	0	4	7
C	18	4	0	3
D	IF	IF	3	0

Node D table (Iter 2)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0



Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

Node A table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	7
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 1)

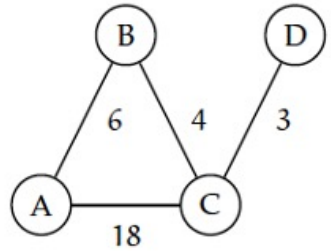
	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	**10**	**4**	**0**	**3**
D	IF	IF	3	0

Node D table (Iter 1)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Remember that we also have to run the algorithm for the B,C and D tables:

- **Checking if there is a shorter path from C to A**
  - There is is, by going via B, we get cost 4 + 6 = 10
  - Hence, we need to send our DV to neighbors
- **Checking if there is a shorter path from C to B**
  - There is not
- **Checking if there is a shorter path from C to D**
  - There is not



Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

Node A table (Iter 1)

	A	B	C	D
A	**0**	**6**	**10**	**21**
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	**6**	**0**	**4**	**7**
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	**10**	**4**	**0**	**3**
D	IF	IF	3	0

Node D table (Iter 1)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	3	0

Node A table (Iter 2)

	A	B	C	D
A	0	6	10	21
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 2)

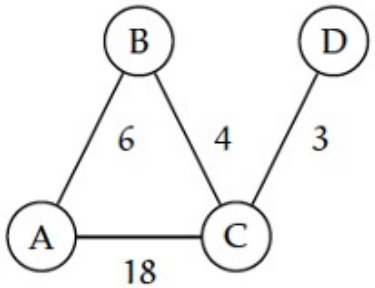
	A	B	C	D
A	0	6	10	21
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 2)

	A	B	C	D
A	A	0	6	10
B	6	0	4	7
C	10	4	0	3
D	IF	IF	3	0

Node D table (Iter 2)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	10	4	0	3
D	IF	IF	3	0



Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

Node A table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	7
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	10	4	0	3
D	IF	IF	3	0

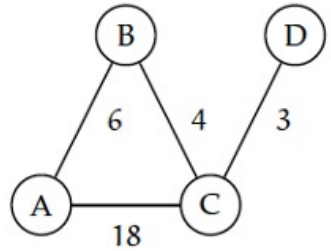
Node D table (Iter 1)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	**21**	**7**	**3**	**0**

Remember that we also have to run the algorithm for the B,C and D tables:

- **Checking if there is a shorter path from D to A**
  - There is, by going from D to A via C (3+18) = 21
- **Checking if there is a shorter path from D to B**
  - There is by going via C (3+4) = 7
- **Checking if there is a shorter path from D to C**
  - There is not

- Once again, update DV and send to neighbors



Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

Node A table (Iter 1)

	A	B	C	D
A	**0**	**6**	**10**	**21**
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	**6**	**0**	**4**	**7**
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	**10**	**4**	**0**	**3**
D	IF	IF	3	0

Node D table (Iter 1)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	**21**	**7**	**3**	**0**

Node A table (Iter 2)

	A	B	C	D
A	0	6	10	21
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 2)

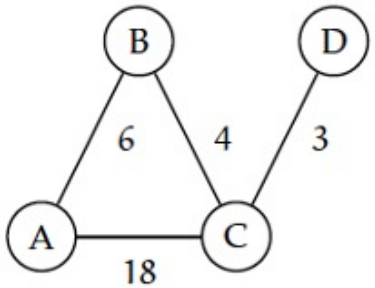
	A	B	C	D
A	0	6	10	21
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 2)

	A	B	C	D
A	A	0	6	10
B	6	0	4	7
C	10	4	0	3
D	21	7	3	0

Node D table (Iter 2)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	10	4	0	3
D	21	7	3	0



Get ready for yet another iteration...

Node A table (Init)

	A	B	C	D
A	0	6	18	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node B table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	6	0	4	IF
C	IF	IF	IF	IF
D	IF	IF	IF	IF

Node C table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node D table (Init)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	IF	IF	IF	IF
D	IF	IF	3	0

Node A table (Iter 1)

	A	B	C	D
A	0	6	10	21
B	6	0	4	IF
C	18	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	7
C	18	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 1)

	A	B	C	D
A	0	6	18	IF
B	6	0	4	IF
C	10	4	0	3
D	IF	IF	3	0

Node D table (Iter 1)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	18	4	0	3
D	21	7	3	0

Node A table (Iter 2)

	A	B	C	D
A	0	6	10	21
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 2)

	A	B	C	D
A	0	6	10	21
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 2)

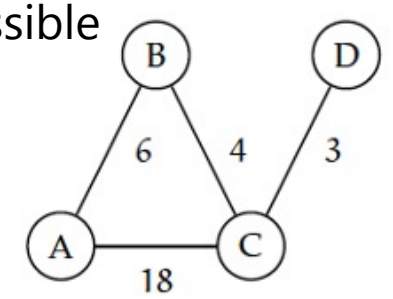
	A	B	C	D
A	A	0	6	10
B	6	0	4	7
C	10	4	0	3
D	21	7	3	0

Node D table (Iter 2)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	10	4	0	3
D	21	7	3	0

For Node A table:
We can get to D via B, hence A to D cost 6+7 = 13
Otherwise no optimizations

For Node B table:
No optimizations possible



For Node C table:
No optimizations possible

For Node D table:
We can get to A via C, hence D to A cost 3+10 = 13

Node A table (Iter 2)

	A	B	C	D
A	**0**	**6**	**10**	**13**
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 2)

	A	B	C	D
A	0	6	10	21
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 2)

	A	B	C	D
A	A	0	6	10
B	6	0	4	7
C	10	4	0	3
D	21	7	3	0

Node D table (Iter 2)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	10	4	0	3
D	**13**	**7**	**3**	**0**

For Node A table:
We can get to D via B, hence A to D cost 6+7 = 13
Otherwise no optimizations

For Node B table:
No optimizations possible

For Node C table:
No optimizations possible

For Node D table:
We can get to A via C, hence D to A cost 3+10 = 13

Node A table (Iter 3)

	A	B	C	D
A	0	6	10	13
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 3)

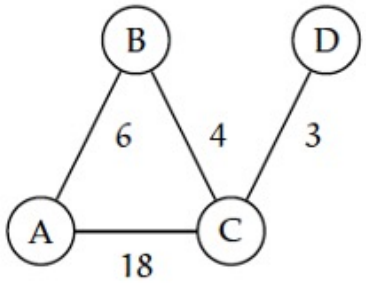
	A	B	C	D
A	0	6	10	13
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 3)

	A	B	C	D
A	0	6	10	13
B	6	0	4	7
C	10	4	0	3
D	13	7	3	0

Node D table (Iter 3)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	10	4	0	3
D	13	7	3	0



Node A table (Iter 2)

	A	B	C	D
A	**0**	**6**	**10**	**13**
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 2)

	A	B	C	D
A	0	6	10	21
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 2)

	A	B	C	D
A	A	0	6	10
B	6	0	4	7
C	10	4	0	3
D	21	7	3	0

Node D table (Iter 2)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	10	4	0	3
D	**13**	**7**	**3**	**0**

For Node A table:
We can get to D via B, hence A to D cost 6+7 = 13
Otherwise no optimizations

For Node B table:
No optimizations possible

For Node C table:
No optimizations possible

For Node D table:
We can get to A via C, hence D to A cost 3+10 = 13

Node A table (Iter 3)

	A	B	C	D
A	0	6	10	13
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node B table (Iter 3)

	A	B	C	D
A	0	6	10	13
B	6	0	4	7
C	10	4	0	3
D	IF	IF	IF	IF

Node C table (Iter 3)

	A	B	C	D
A	0	6	10	13
B	6	0	4	7
C	10	4	0	3
D	13	7	3	0

Node D table (Iter 3)

	A	B	C	D
A	IF	IF	IF	IF
B	IF	IF	IF	IF
C	10	4	0	3
D	13	7	3	0

For Node A table:
No optimizations possible

For Node B table:
No optimizations possible

For Node C table:
No optimizations possible

For Node D table:
No optimizations possible

As there are no more optimizations, changes have propagated and we stop

Final table (Composed of DVs from A,B,C and D)				
	A	**B**	**C**	**D**
A	0	6	10	13
B	6	0	4	7
C	10	4	0	3
D	13	7	3	0

Security

Good things to remember about Security

- Questions within security are often open-ended and you need to know the **theory** well
- Here, I will briefly give you a brush up on Confidentiality and Integrity
- **Learning goal:**
  - **Kompetencer i at** "Ræsonnere omkring enkle sikkerhedsegenskaber for et givent system"
  - **Viden om** "Basal anvendt kryptografi, og operationelle metoder til at sikre styresystemsmekanismer, netværksprotokoller og applikationer."
  - As an example, that does **NOT** mean being able to generate arbitrary RSA keys by hand, but rather understand when and why RSA can be beneficial and when it is not.

Confidentiality

- A typical aim in IT-security is to establish confidential communication between Alice and Bob
  - Use encryption to ensure **confidentiality**
  - RSA is assymetric (private,public), i.e. keypair
  - However encrypting each message through RSA is computationally expensive

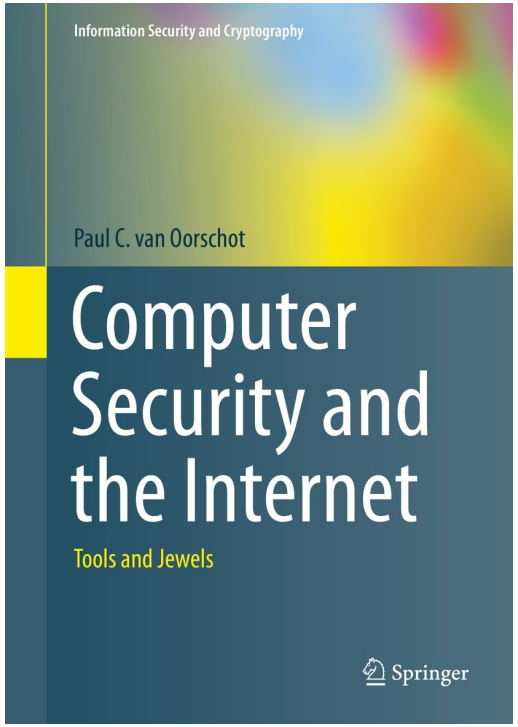
    so typically RSA is used to encrypt a symmetric session key
  - Further communication is then encrypted using a session key
  - This is essentially what happens in the **TLS,** remember **TLS** is what makes **HTTP -> HTTPS**
  - **Note:** Just because something is confidential, it does not guarantee protection against e.g. replay attacks.
  - Also in Databases, passwords are not stored in plaintext
    - Instead, a Hash of (Password+Salt) is stored – the salt helps guard against Rainbow Tables and dictionary attacks

Integrity

- Another aim is for Alice and Bob to ensure that messages are no altered while in transit
  - Therefore, a **checksum** can be used, just as we did in A3 and A4
  - A checksum is a hash of the message, using a **cryptographic hashing algorithm** such as SHA-256, SHA-1 or MD5. If this is sent along the message, the receiver can generate his own hash and compare just as in A3 and A4.
  - **Integrity** does not by itself guarantee the identity of the sender. For that some kind of **authentication** is required
    - Hash(message + secret) is called a **Message Authentication Code (MAC)**
    - If only Alice and Bob know the **secret,** authentication proves identity of sender
    - Another tool is the use of **Digital Signatures** where a **private key** is used to sign (encrypt) the hash of a message. Receiver can use senders **public key** to decrypt the hash of the message and compare with his own hash of the message

If you are interested in IT-Security

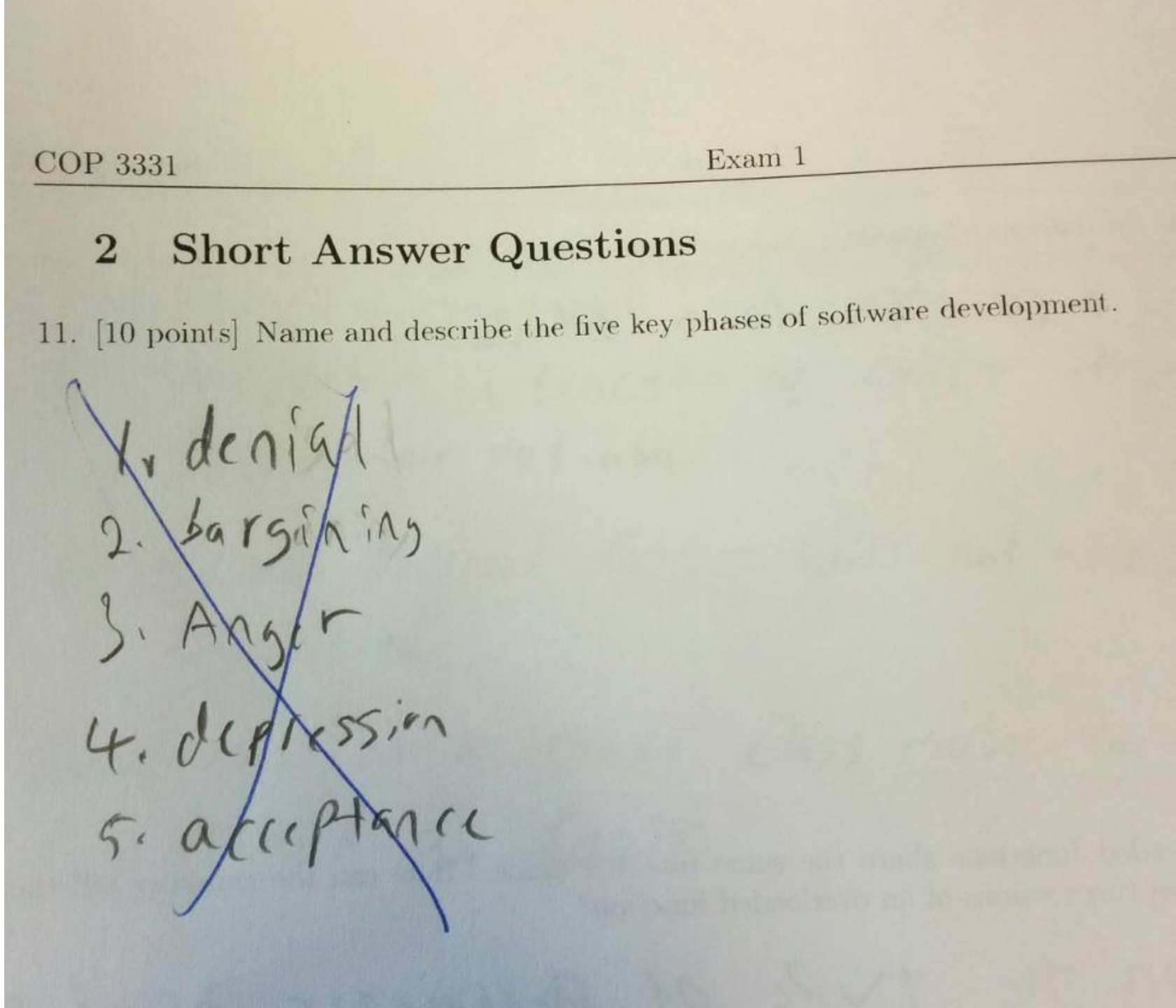
- CompSys gives only a very brief introduction to IT-Security
- If you are interested in the field, I recommend taking the elective Bsc course IT-Security in blok 1 → See https://kurser.ku.dk/course/ndaa09025u/
- The course gives you a theoretical foundation and you get to try out some practical things through the assignments



Postludium

- Thank you for now, it has been a pleasure teaching you
- **Remember** you have come a long way! In week 1 of compSys, 99% of you didn't know what a pointer or malloc was. Also, **C** was just a letter in the alphabet.
- Good luck at the compSys exam (and also good luck at MAD/SS, etc.)
- I will see you around DIKU
- And for those of you taking the course **Participatory design af informationssystemer**, I will see you there ☺

And remember: Even though it may be tempting, don't answer this at the exam...



COP 3331 Exam 1

2 Short Answer Questions

11. [10 points] Name and describe the five key phases of software development.

1. denial
2. bargaining
3. Anger
4. depression
5. acceptance