Edexcel A Level Further Maths: Decision Maths 1

Your notes

Shortest Path Algorithms

Contents

Dijkstra's Algorithm

Your notes

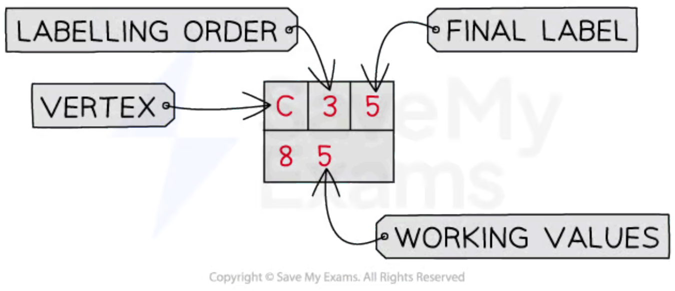
Dijkstra's Algorithm

What is Dijkstra's algorithm?

- **Dijkstra's algorithm** is used to find the **shortest distance** between **two vertices** in a network
  - More precisely, if the algorithm is applied to a whole network it will find the shortest distance between a (fixed) **start vertex** and **every other vertex**
  - However, if the **destination (end)** vertex is reached before the entire network has been considered the algorithm can stop
    - This would be useful, say, in a larger network as it would reduce the time for the algorithm to complete
- It can be used in practical applications to **reduce time, cost or distance** between two points
- Fun fact ...
  - ... a version of Dijkstra's algorithm was used in the Pacman video game to control the movement of the ghosts that chased Pacman!

What notation is used in Dijkstra's algorithm?

- Each vertex will be given a **labelling box**



  - Values in the **'Vertex'** box are simply the **label of the vertex**
  - The value in the **'Labelling order'** box is the vertex's place in the order of being assigned a value in its **'Final label'** box
  - The value in the **'Final label**' box is the **shortest distance** of the vertex from the start vertex
    - 'Final label' is sometimes referred to as 'permanent label'
  - The most recently written value in the **'Working values'** box is the **current distance** of the vertex from the start vertex, the final value in this box will be the same as the value in the 'Final label' box

What are the steps of Dijkstra's algorithm?

- **STEP 1**
  Draw a **labelling box** for each vertex in the network (if not already given) and complete the **'Vertex'** box for each vertex

Your notes

- **STEP 2**
  Complete the labelling box for the **start vertex**:
  - Put **'1'** in the **'Order of labelling'** box
  - Put **'0'** in the **'Final label'** box

- **STEP 3**
  Inspect **all vertices** that are **connected by an edge** to the vertex that has **most recently** had its **'Final label'** box completed
  - This only applies to vertices that do **not** already have a completed 'Final label' box

Put a **working value** in the relevant box for each of these vertices

  - The working value should be the **sum** of the value in the **'Final label' box of the vertex whose final label was most recently completed** and the **length of the edge connecting the vertices**
  - If there is already a working value, only replace it if the **new working value is smaller**

- **STEP 4**
  Look at **all** vertices with a value in the **'Working value'** box (ignoring vertices which have their 'Final label' box completed)
  Identify the vertex with the **smallest working value** and complete its labelling box:
  - Put the **next sequential number** in the **'Order of labelling'** box
  - Put the **current working value** into its **'Final label'** box

- **STEP 5**
  Repeat steps 3 and 4 until the **end vertex** has its **'Final label'** box completed
  - (or until every vertex has a final label)

- **STEP 6**
  Move **backwards** from the **destination** (end) **vertex** towards the start vertex to identify the **shortest path**
  Two vertices lie on the shortest path if the **difference** between the values in their **'Final label'** boxes is **equal** to the weight of the **arc** that connects them

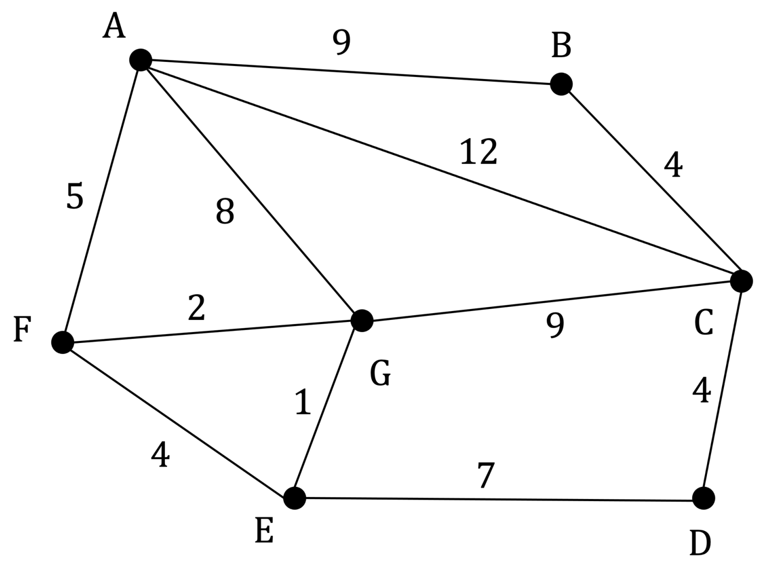
Examiner Tip

- Avoid crossing out working values that are replaced with smaller working values
  - examiners like to be able to see all of the values in your diagram clearly!

Your notes

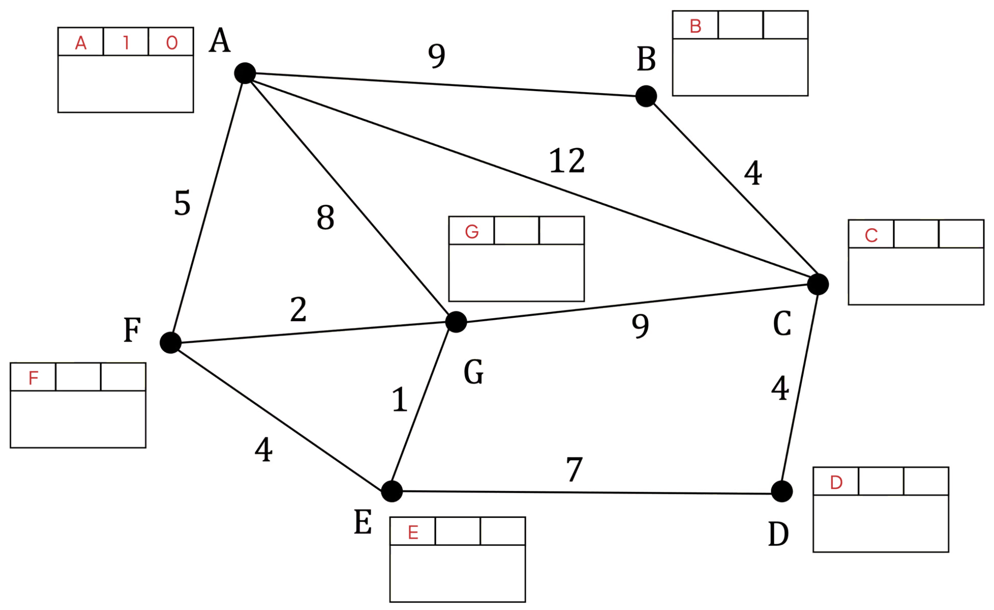
Worked example

A network is shown in the diagram below.



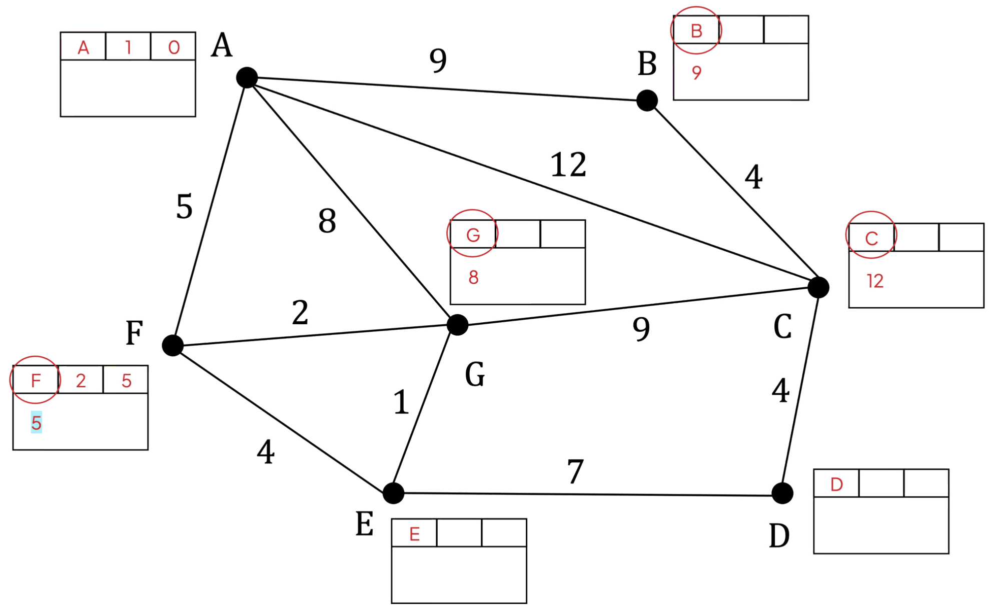
Using Dijkstra's algorithm, find the shortest path between vertex A and vertex D and state its length.

- **STEP 1**
  Draw in the labelling boxes for all vertices
- **STEP 2**
  Complete the labelling box for the start vertex, vertex A



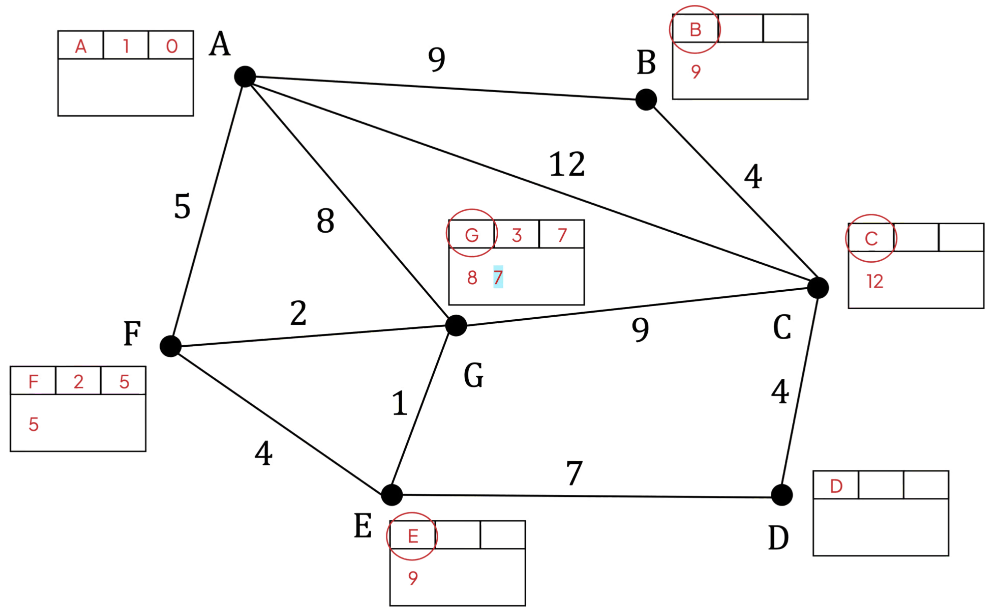
- **STEP 3**
  Vertices B, C, F and G are all connected to vertex A by an edge
  Put the weights of the edges into the working values box at each of these vertices

- **STEP 4**
  Inspect **all** vertices with a working value for the one with the lowest working value
  Vertex F has the lowest working value, 5, so label vertex F with '2' in the 'Order of labelling' box and '5' in the 'Final label' box

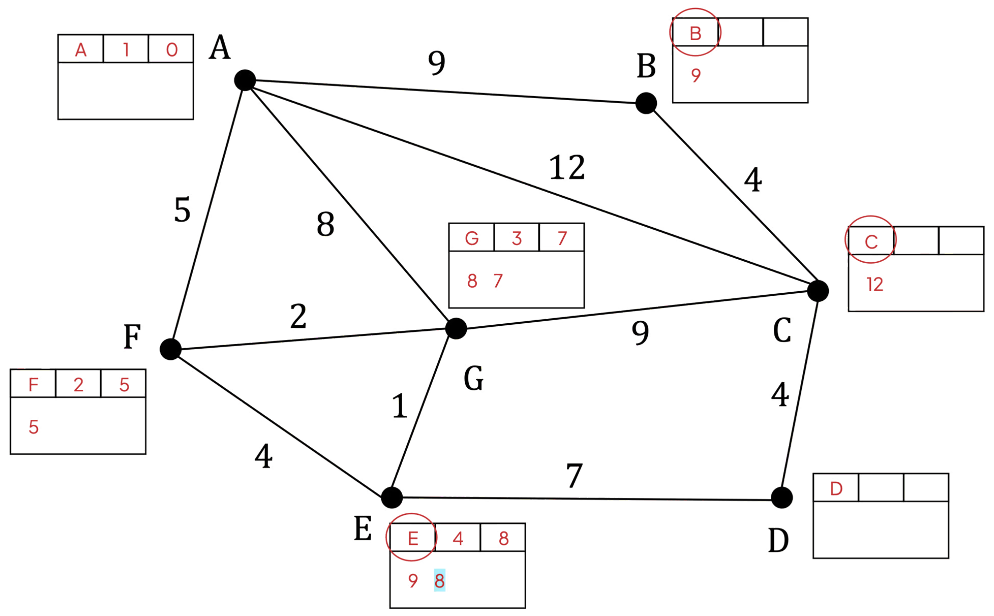


Your notes

- **STEP 5**
  Repeat Steps 3 and 4 until the end vertex has its 'Final label' box completed
- **STEP 3**
  For all vertices connected directly to F, find the 'Working value' from F
  The working value of vertex G from vertex F is lower than when it comes directly from vertex A
  Write this new working value into the box for G
- **STEP 4**
  Identify the smallest working value from all vertices with working values but no 'Final label'
  The smallest of these values is at vertex G
  Label vertex G with '3' and transfer its working value into the 'Final value' box

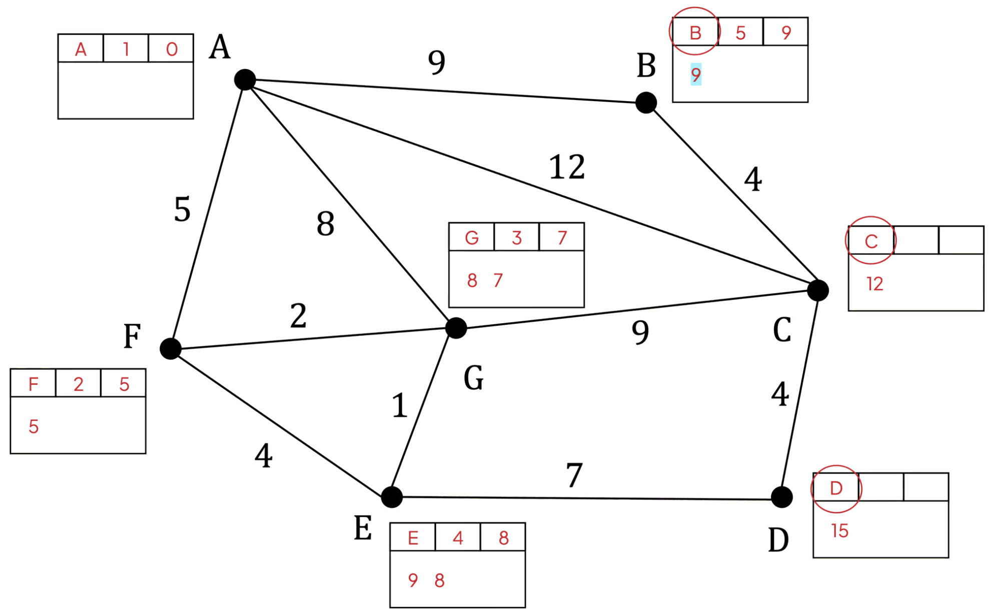


- **STEP 3**
  Find the 'Working value' for all vertices directly connected to vertex G
  There is no need to replace the working value at vertex C, as the distance travelled directly from G will be bigger
- **STEP 4**
  Identify the smallest working value from vertices B, C and E, (vertex E)
  Label that vertex '4' and transfer its working value into the 'Final value' box



- **STEP 3**
  For all of the vertices that are connected to vertex E find the 'Working value'

- **STEP 4**
  Identify the smallest working value from vertices B, C and D, (vertex B)
  Label that vertex '5' and transfer its working value into the 'Final value' box



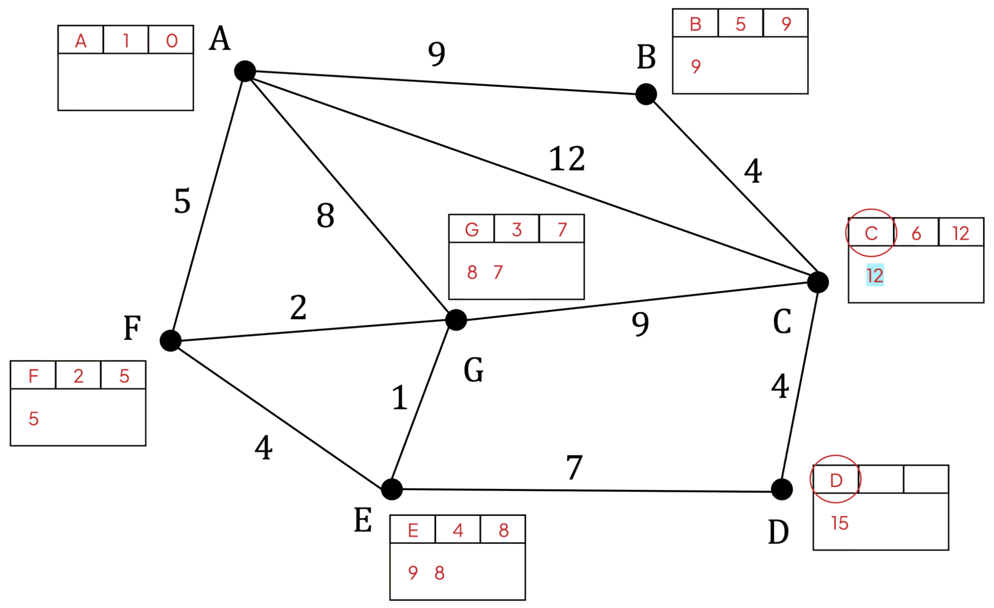
- **STEP 3**
  For all of the vertices that are connected to vertex B find the 'Working value'

Your notes

Your notes

For vertex C you would not put in a new working value as it would be bigger than the current value

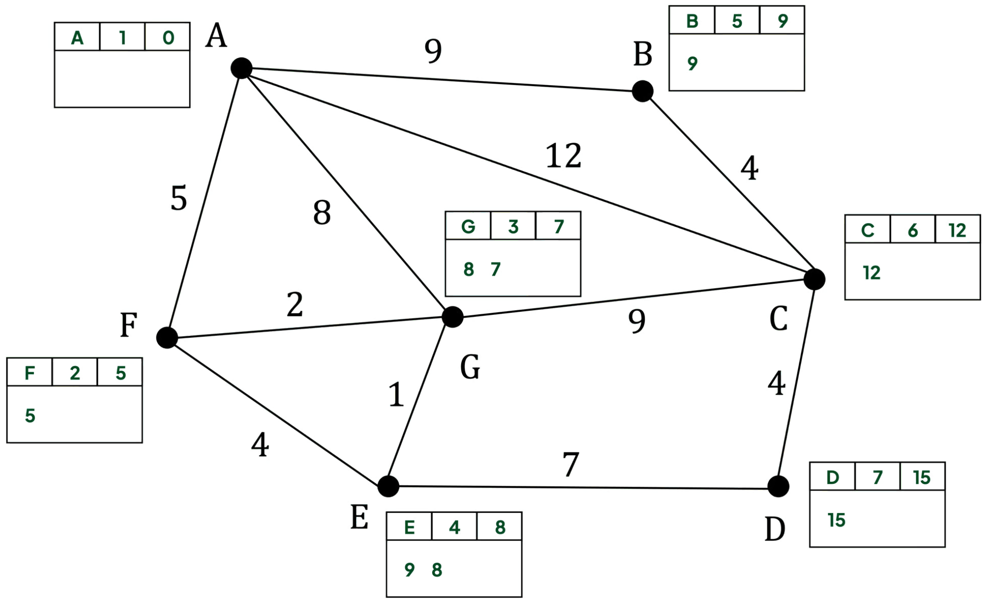
- **STEP 4**
  Identify the smallest working value from vertices C and D, (vertex C)
  Label that vertex '6' and transfer its working value into the 'Final value' box



- **STEP 3**
  Vertex D is the only unlabelled vertex, so find its 'Working value'
  Although vertex D is directly connected to vertex C, a new working value is not needed at D (it would be bigger than the value already there)

- **STEP 4**
  Label vertex D as vertex '7' and transfer its working value into the 'Final value' box

Your notes



You do not need to show all of the stages of working as separate diagrams
It is enough to show the final diagram with all vertices fully labelled

- **STEP 6**
  You can find the shortest length by working backwards from vertex D

15 - 7 = 8 (D → E)

8 - 1 = 7 (E → G)

7 - 2 = 5 (G → F)

5 - 5 = 0 (F → A)

Shortest route: AFGED

Floyd's Algorithm

Your notes

Floyd's Algorithm

What is Floyd's Algorithm?

- **Floyd's algorithm** finds the **shortest distance** *and* the **shortest route** between any **pair of nodes** in a **network**
- This is achieved by setting up and updating a pair of **matrices**
  - The **first** is a matrix of (least) distances between nodes
    - it is generally called the **distance matrix**
    - the final distance matrix shows the **shortest distance** between any pair of nodes
  - The **second** matrix allows the shortest route between nodes to be deduced
    - it is generally called the **route matrix**
- At each stage (iteration) of Floyd's algorithm, both the **distance matrix** and **route matrix** are **updated**
- Floyd's algorithm will seem long winded and slow at first
  - But the symmetry (in undirected networks) in the distance matrix can reduce the amount of work involved whilst also acting as a check

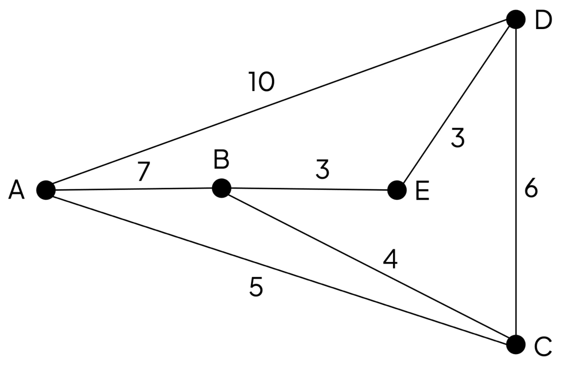
How do I set up the initial distance matrix and initial route matrix for Floyd's algorithm?

- The **initial distance matrix** has the *direct* distances between nodes
  - If there is no *direct* distance the symbol ∞ is used (to represent/assume a very large distance!)
  - In an **undirected network,** all distance matrices will exhibit **symmetry**
    - this is because the distance from node A to C, say, will be the same as the distance from C to A
    - this will apply to all pairs of nodes
    - this helps reduce the work required in Floyd's algorithm but also acts as a means of checking each distance matrix
- The **initial route matrix** assumes the shortest route between each pair of nodes is a *direct* link
  - So the shortest route between A and D, say, is to go directly from A to D
  - It doesn't matter if there is no direct link between a pair of nodes
    - this will be accounted for later in Floyd's algorithm

Your notes

Worked example

For the network shown below, set up the initial distance matrix and the initial route matrix for Floyd's algorithm.



Use the direct distances given on the diagram

Where there is no direct link, use ∞ for the (initial) distance

There is a direct link between nodes A and C, with a distance of 5, so row A, column C will contain a 5
(This is also true for row C, column A, as the network is undirected)

There is no direct link between nodes A and E, so use ∞ (also true for row E, column A)

Initial distance matrix

	A	B	C	D	E
A	–	7	5	10	∞
B	7	–	4	17	3
C	5	4	–	6	∞
D	10	17	6	–	3
E	∞	3	∞	3	–

The initial route matrix has the destination node directly in place for each starting node

Initial route matrix

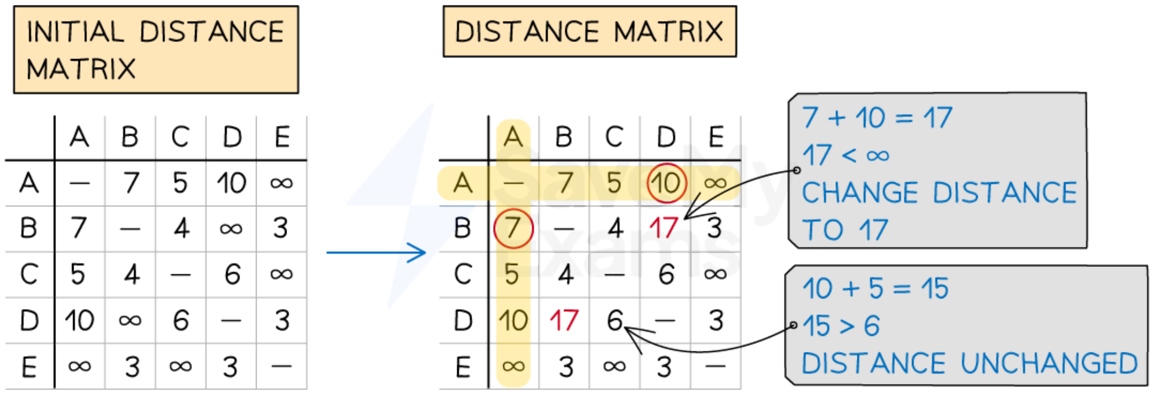
	A	B	C	D	E
A	A	B	C	D	E
B	A	B	C	D	E
C	A	B	C	D	E
D	A	B	C	D	E

E	A	B	C	D	E

Your notes

How do I update the distance matrix in Floyd's algorithm?

- For the FIRST iteration highlight the **first** row and the **first** column of the **distance** matrix
  - For the other cells in the distance matrix (except those labelled with a '-')
    - add the corresponding highlighted distances together
    - if this is **less** than the current distance, then the current distance gets updated



Initial distance matrix:

	A	B	C	D	E
A	–	7	5	10	∞
B	7	–	4	∞	3
C	5	4	–	6	∞
D	10	∞	6	–	3
E	∞	3	∞	3	–

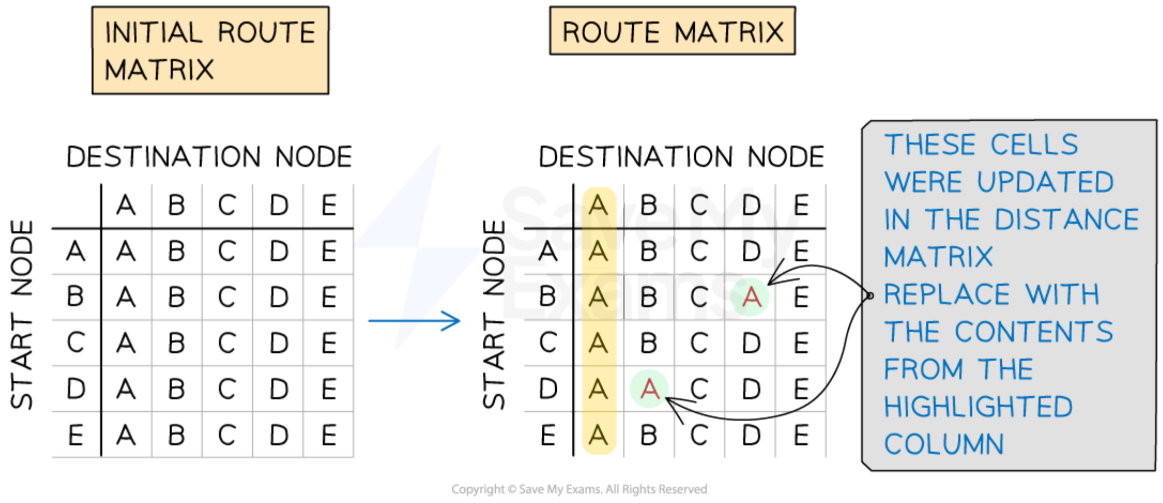
Distance matrix:

	A	B	C	D	E
A	–	7	5	10	∞
B	7	–	4	17	3
C	5	4	–	6	∞
D	10	17	6	–	3
E	∞	3	∞	3	–

- This process of updating distances is repeated for each iteration of Floyd's algorithm
  - The number of iterations will be equal to the number of nodes
    - On the second iteration, the SECOND row and SECOND column would be highlighted
    - On the k^{th} iteration, the k^{th} row and column would be highlighted

How do I update the route matrix in Floyd's algorithm?

- For the FIRST iteration, highlight the **first column** of the **route** matrix
- Find the corresponding cells that have been updated in the **distance** matrix for this iteration
  - These cells are updated with the node in the highlighted column


INITIAL ROUTE MATRIX
ROUTE MATRIX
DESTINATION NODE
START NODE
THESE CELLS WERE UPDATED IN THE DISTANCE MATRIX REPLACE WITH THE CONTENTS FROM THE HIGHLIGHTED COLUMN
Copyright © Save My Exams. All Rights Reserved

	A	B	C	D	E
A	A	B	C	D	E
B	A	B	C	D	E
C	A	B	C	D	E
D	A	B	C	D	E
E	A	B	C	D	E

	A	B	C	D	E
A	A	B	C	D	E
B	A	B	C	A	E
C	A	B	C	D	E
D	A	A	C	D	E
E	A	B	C	D	E

- This process of updating routes is repeated for each iteration of Floyd's algorithm
  - The number of iterations will be equal to the number of nodes
    - On the second iteration, the SECOND column would be highlighted
    - On the k^{th} iteration, the k^{th} column would be highlighted

Examiner Tip

- An exam question is unlikely to ask you to do all iterations of Floyd's algorithm
  - Make sure you can recognise where to start if given matrices after, say, the third iteration

Your notes

Worked example

The distance and route matrices below follow one complete iteration of Floyd's algorithm.

Distance matrix

	A	B	C	D	E
A	-	7	5	10	∞
B	7	-	4	17	3
C	5	4	-	6	∞
D	10	17	6	-	3
E	∞	3	∞	3	-

Route matrix

	A	B	C	D	E
A	A	B	C	D	B
B	A	B	C	A	E
C	A	B	C	D	E
D	A	A	C	D	E
E	A	B	C	D	E

Complete the remaining iterations of Floyd's algorithm, showing both the distance and route matrices after each iteration.

2nd Iteration

In the distance matrix, row A, column E, $3 + 7 = 10$ and $10 < \infty$ so the distance is changed to 10

Row C, column E, $3 + 4 = 7$ and $7 < \infty$ so the distance is changed to 7

Likewise for row E and columns A and C (note the symmetry as the network is undirected)

In the route matrix, corresponding cells are changed

Distance matrix

	A	**B**	**C**	**D**	**E**
A	-	7	5	10	**10**
B	7	-	4	17	3
C	5	4	-	6	**7**
D	10	17	6	-	3
E	**10**	3	**7**	3	-

Route matrix

	A	**B**	**C**	**D**	**E**
A	A	B	C	D	**B**
B	A	B	C	A	E
C	A	B	C	D	**B**
D	A	A	C	D	E
E	**B**	B	**B**	D	E

3rd iteration

Row B, column D, $6 + 4 = 10$ and $10 < 17$ so distance is changed to 10

This is the same for row D, column B and the corresponding route matrix cells are updated too

Distance matrix

	A	**B**	**C**	**D**	**E**

Route matrix

	A	**B**	**C**	**D**	**E**

A	-	7	5	10	10
B	7	-	4	10	3
C	5	4	-	6	7
D	10	10	6	-	3
E	10	3	7	3	-

A	A	B	C	D	B
B	A	B	C	C	E
C	A	B	C	D	B
D	A	C	C	D	E
E	B	B	B	D	E

4th iteration

There are no changes in this iteration, but note that there still could be changes in the fifth iteration

Distance matrix

	A	**B**	**C**	**D**	**E**
A	-	7	5	10	10
B	7	-	4	10	3
C	5	4	-	6	7
D	10	10	6	-	3
E	10	3	7	3	-

Route matrix

	A	**B**	**C**	**D**	**E**
A	A	B	C	D	B
B	A	B	C	C	E
C	A	B	C	D	B
D	A	C	C	D	E
E	B	B	B	D	E

5th iteration (final iteration)

There are two cells to change on this iteration

Distance matrix

	A	**B**	**C**	**D**	**E**
A	-	7	5	10	10
B	7	-	4	6	3
C	5	4	-	6	7
D	10	6	6	-	3
E	10	3	7	3	-

Route matrix

	A	**B**	**C**	**D**	**E**
A	A	B	C	D	B
B	A	B	C	E	E
C	A	B	C	D	B
D	A	E	C	D	E
E	B	B	B	D	E

Final matrices

Final distance matrix

	A	B	C	D	E
A	**-**	**7**	**5**	**10**	**10**
B	**7**	**-**	**4**	**6**	**3**
C	**5**	**4**	**-**	**6**	**7**

Final route matrix

	A	B	C	D	E
A	**A**	**B**	**C**	**D**	**B**
B	**A**	**B**	**C**	**E**	**E**
C	**A**	**B**	**C**	**D**	**B**

D	**10**	**6**	**6**	**–**	**3**
E	**10**	**3**	**7**	**3**	**–**

D	**A**	**E**	**C**	**D**	**E**
E	**B**	**B**	**B**	**D**	**E**

Your notes

How do I use Floyd's algorithm to find shortest distances and routes?

- The **shortest distance** is found by reading off the value from the final **distance matrix**
- To find the **shortest route** between nodes, use the **final route matrix**
  - start on the **row** for the **starting** node
  - find the cell that is under the **destination** node, then move to that row
  - repeat until the cell contains the **destination** node

Examiner Tip

- Exam questions will often involve the final matrices so questions can be asked about shortest distances and routes
- The far majority of networks are undirected
  - there will be symmetry in each distance matrix
  - this can cut down the amount of work needed at each iteration of Floyd's algorithm
  - but the repetition can also act as a check

Worked example

The final distance and route matrices for Floyd's algorithm are given below.

Final distance matrix

	A	B	C	D	E
A	-	7	5	10	10
B	7	-	4	6	3
C	5	4	-	6	7
D	10	6	6	-	3
E	10	3	7	3	-

Final route matrix

	A	B	C	D	E
A	A	B	C	D	B
B	A	B	C	E	E
C	A	B	C	D	B
D	A	E	C	D	E
E	B	B	B	D	E

a) i) Find the shortest distance between nodes A and E.

ii) Find the shortest distance between nodes B and C.

Final distance matrix

	A	B	C	D	E
A	-	7	5	10	10
B	7	-	4	6	3
C	5	4	-	6	7
D	10	6	6	-	3
E	10	3	7	3	-

i) Use the final distance matrix to find where row A meets column E

10

Also from row E, column A

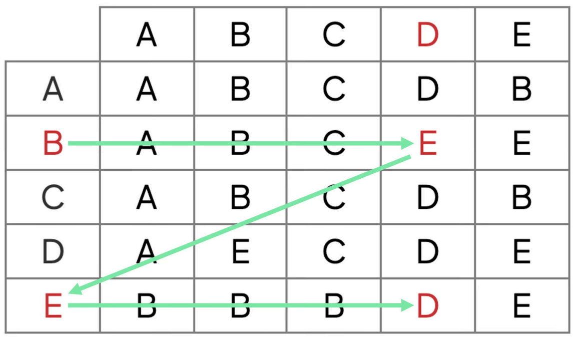
ii) Use the final distance matrix to find where row B meets column C

4

Also from row C, column B

b) Find the shortest route between nodes B and D.

Final route matrix



	A	B	C	D	E
A	A	B	C	D	B
B	A	B	C	E	E
C	A	B	C	D	B
D	A	E	C	D	E
E	B	B	B	D	E

Start on row B (start node)
Node E is indicated under the destination node column (D) for this row, so move to row E
Node D is indicated under the destination node (D) so the route is complete

The shortest route is BED

Comparing Dijkstra's & Floyd's Algorithms

Your notes

Comparing Dijkstra's & Floyd's Algorithms

What is the difference between Dijkstra's and Floyd's algorithms?

- Both algorithms find the **shortest path,** and its **weight** (length) between **nodes** on a graph
- **Dijkstra's** algorithm finds the shortest path between a **fixed starting node** and every **other** node in the network
  - This would be useful where a starting point **cannot be moved**
    - e.g. a power station, a distribution warehouse
  - The **advantage** of Dijkstra's algorithm are speed
    - the algorithm can be stopped once the desired end node is reached
  - The **disadvantage** of Dijkstra's algorithm is the limited information it provides (compared to Floyd's)
- **Floyd's** algorithm finds the shortest path between **any** pair of **start** and **end** nodes
  - This would be useful where a starting point can be **anywhere** on the network
    - e.g. a robot inspecting a network of pipelines
  - The **advantage** of Floyd's algorithm is the amount of information it provides (compared to Dijkstra)
  - The **disadvantage** of Floyd's algorithm is the time it takes (especially manually without a computer!)
- **Repeating** Dijkstra's algorithm, taking each node in turn as the starting node, would produce the same information as Floyd's algorithm