3.10 Graph Theory

IB Maths - Revision Notes

3.10.1 Introduction to Graph Theory

Parts of a Graph

A **graph** is a mathematical structure that is used to represent objects and the connections between them. They can be used in modelling many real-life applications, e.g. electrical circuits, flight paths, maps etc.

What are the different parts of a graph?

- A **vertex** (point) represents an object or a place
  - **Adjacent vertices** are connected by an edge
  - The **degree** of a vertex can be defined by how many edges are connected to it
- An **edge** (line) forms a connection between two vertices
  - **Adjacent edges** share a common vertex
  - An edge that starts and ends at the same vertex is called a **loop**
  - There may be **multiple edges** connecting two vertices

Types of Graphs

What are the types of graphs?

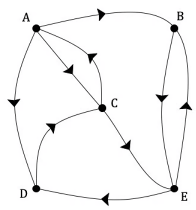
- A **complete graph** is a graph in which each vertex is connected by an edge to each of the other vertices
- The edges in a **weighted graph** are assigned numerical values such as distance or money
- The edges in a **directed graph** can only be travelled along in the direction indicated
  - the **in-degree** of a vertex is the number of edges that lead to that vertex
  - the **out-degree** is the number of edges that leave from that vertex
- A **simple graph** is undirected and unweighted and contains **no loops** or **multiple edges**
- Given a graph G, a **subgraph** will only contain edges and vertices that appear in G
- In a **connected graph** it is possible to move along the edges and vertices to find a route between any two vertices
  - If the graph is **strongly connected**, this route can be in either direction between the two vertices
- A **tree** is a graph in which any two vertices are connected by exactly one path
- A **spanning tree** is a subgraph, which is also a tree, of a graph G that contains all the vertices from G

Exam Tip

- There are a lot of specific terms involved in graph theory and you are often asked to describe them in an exam - make sure you learn the definitions
- Make sure that any graphs you draw are big and clear so they are easy for the examiner to read

Worked example

The graph G shown below is a strongly connected, unweighted, directed graph with 5 vertices.



a) State the in-degree of vertex A.

Only the edge connecting A and C is going into A

In-degree of vertex A = 1

b) Explain why the graph is considered to be strongly connected.

The graph is strongly connected because it is possible to construct a walk in either direction between any two vertices

3.10.2 Walks & Adjacency Matrices

Walks & Adjacency Matrices

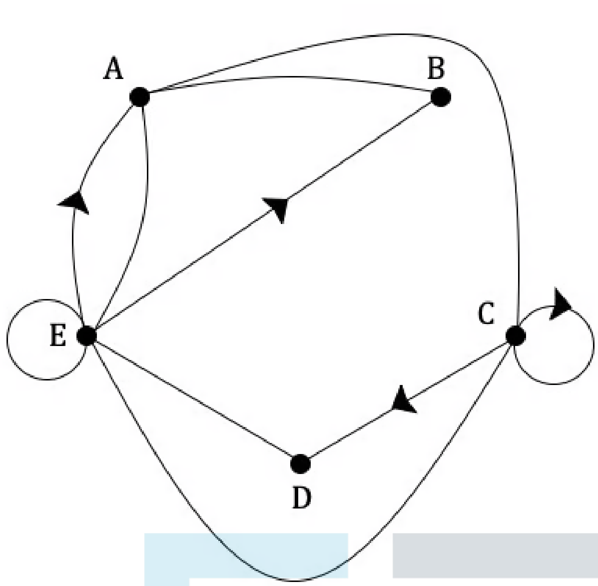
Adjacency matrices are another way to represent graphs and connections between the different vertices.

What is an adjacency matrix?

- An adjacency matrix is a **square** matrix where all of the vertices in the graph are listed as the headings for both the rows (i) and columns (j)
- An adjacency matrix can be used to show the **number of direct connections** between two vertices
- An entry of **0** in the matrix means that there is **no direct connection** between that pair of vertices
- In a **simple** graph the only entries are either 0 or 1
- A **loop** is indicated in an adjacency matrix with a value in the **leading diagonal** (the line from top left to bottom right)
  - In an **undirected matrix** the value in the leading diagonal will be **2** because you can use the loop to travel out of and into the vertex in two different directions
  - In a **directed matrix**, if the loop has been given a direction, the value in the leading diagonal will be **1** as you can only travel along the loop out of and back into the vertex in one direction
  - For a graph with **no loops** every entry in the leading diagonal will be **0**
- An **undirected graph** will be **symmetrical** in the leading diagonal
- The sum of the entries in a **row** is the **in degree** of that vertex
- The sum of the entries in a **column** is the **out degree** of that vertex

Worked example

Let G be the graph below.



Write down the adjacency matrix for G.

	A	B	C	D	E
A	0	1	1	0	1
B	1	0	0	0	0
C	1	0	1	1	1
D	0	0	0	0	1
E	2	1	1	1	2

Number of Walks

What is a walk?

- A **walk** is a **sequence of vertices** that are visited when moving through a graph along its **edges**
- Both **edges** and **vertices** can be revisited in a walk
- The **length of a walk** is the **total number of edges** that are traversed in the walk

How do you find the number of walks in a graph?

- Let $\boldsymbol{M}$ denote the adjacency matrix of a graph. The (i, j) entry in the matrix $\boldsymbol{M}^k$ will give the number of walks of length k from vertex i to vertex j
- If there is an entry of **2** in the leading diagonal of the matrix, this should be changed to a **1 before** the matrix is raised to a power
- The number of walks, between vertex i and vertex j, of length n or less can be given by the matrix $\boldsymbol{S}^n$, where $\boldsymbol{S}^n = \boldsymbol{M}^1 + \boldsymbol{M}^2 + ... + ... \boldsymbol{M}^n$
  - If all of the entries in a single row of $\boldsymbol{S}^n$ are non-zero values then the graph is **connected**

Exam Tip

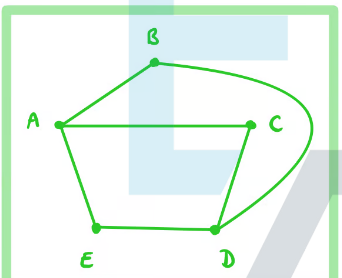
- Read the question carefully to determine if you need to choose a specific power for the adjacency matrix or if you need to play around with different powers!

Worked example

The adjacency matrix $\boldsymbol{M}$ of a graph G is given by

$$\boldsymbol{M} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{matrix} A & B & C & D & E \end{matrix} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

a) Draw the graph described by the adjacency matrix $\boldsymbol{M}$.



b) Find the number of walks of length 4 from vertex B to Vertex E.

Enter the matrix into your GDC and raise it to the power 4

$$M^4 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}^4$$

$$M^4 = \begin{pmatrix} 18 & 0 & 0 & 18 & 0 \\ 0 & 12 & 12 & 0 & 12 \\ 0 & 12 & 12 & 0 & 12 \\ 18 & 0 & 0 & 18 & 0 \\ 0 & 12 & 12 & 0 & 12 \end{pmatrix}$$

Value in row B and column E

The number of walks of length 4 from vertex B to vertex E is 12

c) Find the number of walks of 3 or less from vertex A to vertex C.

Enter the matrix into your GDC and add successive powers of it

$$S^3 = M + M^2 + M^3$$

Value in row A and column C

$$S^3 = \begin{pmatrix} 3 & 7 & 7 & 3 & 7 \\ 7 & 2 & 2 & 7 & 2 \\ 7 & 2 & 2 & 7 & 2 \\ 3 & 7 & 7 & 3 & 7 \\ 7 & 2 & 2 & 7 & 2 \end{pmatrix}$$

The number of walks of length 3 or less from vertex A to vertex C is 7

Weighted Adjacency Tables

A **weighted adjacency table** gives more detailed information about the connection between different vertices in a **weighted graph**.

What is a weighted adjacency table?

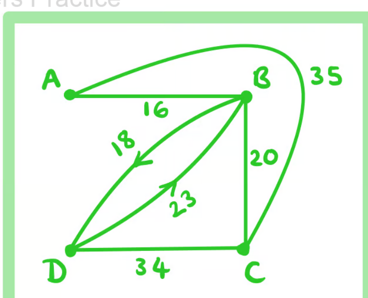
- A weighted adjacency table is different to an adjacency matrix as the **value** in each cell is the **weight** of the edge connecting that pair of vertices
  - Weight could be cost, distance, time etc.
- An **empty cell** can be used to indicate that there is **no connection** between a pair of vertices
- A directed graph is **not** symmetrical along the leading diagonal (the line from top left to bottom right)
- When drawing a graph from its adjacency table be careful when labelling the edges
  - For an **un-directed graph** the two cells between a specific pair of vertices will be the same so connect the vertices with **one edge** labelled with the relevant weight
  - For a **directed graph** if the two cells between a specific pair of vertices have different values draw **two lines** between the vertices and label each with the correct weight and direction
- A weighted adjacency table can be used to work out the weight of different **walks** in the graph

Worked example

The table below shows the time taken in minutes to travel by car between 4 different towns.

	A	B	C	D
A		16	35	
B	16		20	18
C	35	20		34
D		23	34	

a) Draw the graph described by the adjacency table.



b) State the time taken to drive from Town B to Town D.

18 minutes

3.10.3 Minimum Spanning Trees

Kruskal's Algorithm

In a situation that can be modelled by a graph, **Kruskal's algorithm** is a mathematical tool that can be used to **reduce** costs, materials or time.

Why do we use Kruskal's Algorithm?

- Kruskal's algorithm is a series of steps that when followed will produce the **minimum spanning tree** for a **connected graph**
- Finding the minimum spanning tree is useful in a lot of practical applications to connect all of the vertices in the most efficient way possible
- The **number of edges** in a minimum spanning tree will always be **one less** than the **number of vertices** in the graph
- A **cycle** is a **walk** that starts at a given vertex and ends at the **same** vertex.
  - A minimum spanning tree **cannot** contain any cycles.

What is Kruskal's Algorithm?

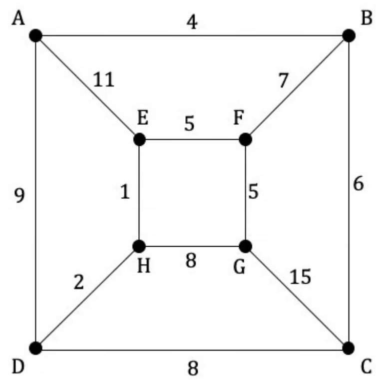
- STEP 1
  Sort the edges in terms of increasing weight
- STEP 2
  Select the edge of least weight (if there is more than one edge of the same weight, either may be used)
- STEP 3
  Select the next edge of least weight that has not already been chosen and add it to your tree provided that it does not make a cycle with any of the previously selected edges
- STEP 4
  Repeat STEP 3 until all of the vertices in the graph are connected

Exam Tip

- When using any of the algorithms for finding the minimum spanning tree, make sure that you state the **order** in which the edges are selected to get full marks for working!

Worked example

Consider the weighted graph G below.

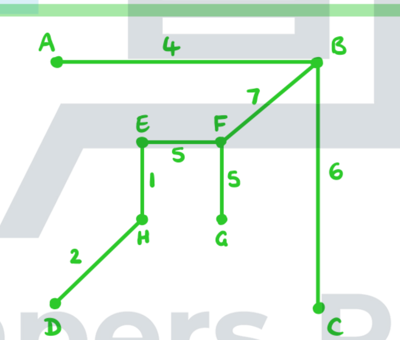


a) Use Kruskal's algorithm to find the minimum spanning tree. Show each step of the algorithm clearly.

List the edges in order of weight

EH (1)
DH (2)
AB (4)
EF (5)
FG (5)
BC (6)
BF (7)
CD (8)
GH (8)
AD (9)
AE (11)
CG (15)

Add the edge of least weight, then the next unused edge of least weight (without making a cycle) until all vertices are connected



Edges added in this order:

EH, DH, AB, EF, FG, BC, BF

Note: EF and FG could be added in either order

b) State the total weight of the minimum spanning tree.

Add up the weights of the edges in the minimum spanning tree

$1 + 2 + 4 + 5 + 5 + 6 + 7 = \boxed{30}$

Prim's Algorithm

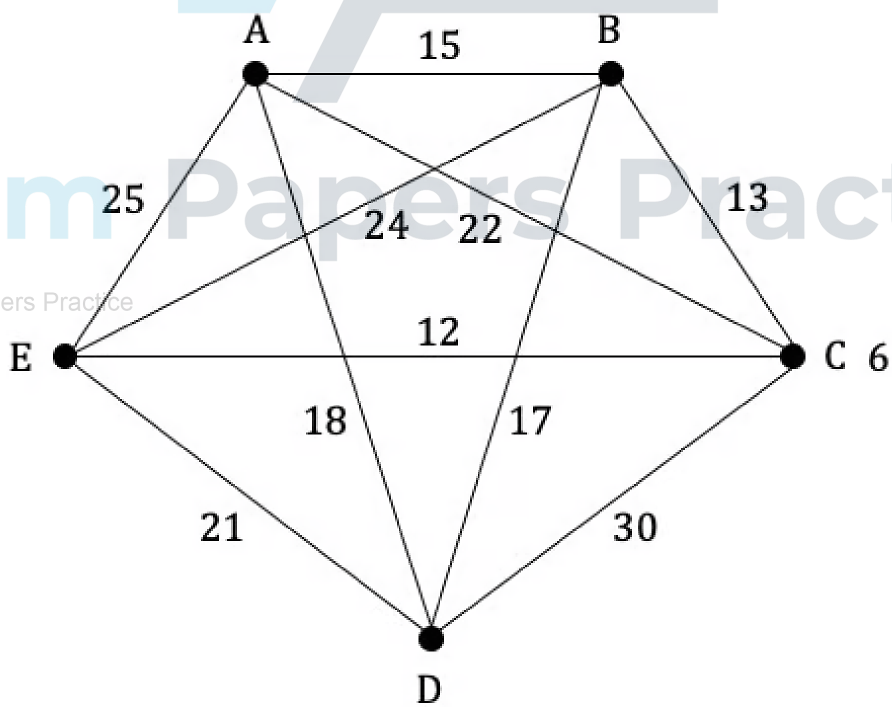
Prim's algorithm is a second method of finding the minimum spanning tree of a graph.

What is Prim's Algorithm?

- **Prim's algorithm** involves adding edges from vertices that are **already connected** to the tree.
- Cycles are **avoided** by only adding edges that are **not** already connected at one end.
- STEP 1
  Start at any vertex and choose the edge of least weight that is connected to it
- STEP 2
  Choose the edge of least weight that is incident (connected) to any of the vertices already connected and does not connect to another vertex that is already in the tree
- STEP 3
  Repeat STEP 2 until all of the vertices are added to the tree

Worked example

Consider the weighted graph below.

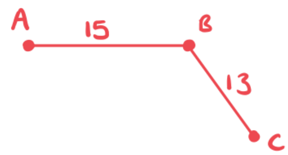


a) Using Prim's algorithm, find the minimum spanning tree.

Select a starting vertex (A) and choose the edge of least weight that is connected to it

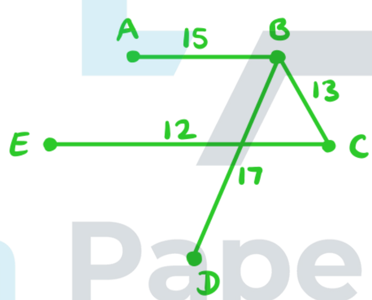
AB (15)

Continued to select the edge of least weight that is connected to any of the other vertices that are already connected in the tree



AB (15)
BC (13)

Repeat this process until all vertices are connected, remembering to record the order in which the edges were added



AB (15)
BC (13)
CE (12)
BD (17)

b) State the total weight of the minimum spanning tree.

Add up the weights of the edges in the minimum spanning tree

$15 + 13 + 12 + 17 = 57$

Prim's Algorithm Using a Matrix

Information may be given to you either in the form of a graph or as a **weighted adjacency table**. Prim's algorithm can be adapted to be used from the adjacency matrix.

How do you apply Prim's algorithm to a matrix?

- A minimum spanning tree is built up from the least weight edges that are incident to vertices already in the tree by looking at the **relevant rows** in the **adjacency table**
- STEP 1
  Select any vertex to start from, cross out the values in the column associated with that vertex and label the row associated with the vertex 1
- STEP 2
  Circle the lowest value in any cell along that row and add the edge to your tree, cross out the remaining values in the column of the cell that you have circled
- STEP 3
  Label the row associated with the same vertex as the column in the previous STEP with the next number
- STEP 4
  Circle the lowest value in any cell along any of the rows that have been labelled and add the edge to your tree, cross out the remaining values in the column of the cell that you have circled
- STEP 5
  Repeat STEPS 3 and 4 until all rows have been labelled and all vertices have been added to the tree

Which should I use Prim's or Kruskal's Algorithm?

- **Kruskal's algorithm** can be used when the information is in graph form whereas **Prim's algorithm** can be used in either graph or matrix form.
- **Prim's algorithm** is sometimes considered to be more **efficient** that **Kruskal's algorithm** as
  - the edges **do not need to be ordered** at the start and
  - it does not rely on **checking for cycles** at each step
- An **exam question** will usually specify which method should be used, otherwise you have the choice
- If you are asked to find the **minimum spanning tree** and the information given in the question is in the form of a **table**, you should use **Prim's algorithm**

Exam Tip

- Look out for questions that ask you to minimise the cost or length etc. from a weighted graph – they are implying that they want you to find the minimum spanning tree!

Worked example

Celeste is building a model city incorporating 6 main buildings that need to be connected to an electrical supply.

Each vertex listed in the table below represents a building and the weighting of each edge is the cost in USD of creating a link to the electrical supply between the given vertices.

	A	B	C	D	E	F
A	-	4	9	8	11	3
B	4	-	13	2	5	12
C	9	13	-	7	1	4
D	8	2	7	-	10	3
E	11	5	1	10	-	15
F	3	12	4	3	15	-

Celeste wants to find the lowest cost solution that links all 6 buildings up to the electrical supply.

a) Starting from vertex A, use Prim's algorithm on the table to find and draw the minimum spanning tree. Show each step of the process clearly.

Start at any vertex and label it 1, cross out the values in colum A and select the edge of least weight in row A

		A	B	C	D	E	F
1	A	-	4	9	8	11	(3)
	B	4	-	13	2	5	12
	C	9	13	-	7	1	4
	D	8	2	7	-	10	3
	E	11	5	1	10	-	15
	F	3	12	4	3	15	-

AF (3)

Label row F 2, delete the remaining values in column F and select the edge of least weight from rows A and F

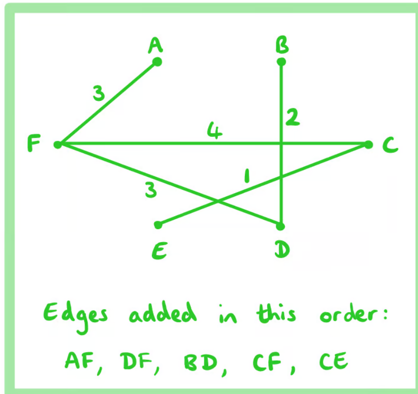
		A	B	C	D	E	F
1	A	-	4	9	8	11	(3)
	B	4	-	13	2	5	12
	C	9	13	-	7	1	4
	D	8	2	7	-	10	3
	E	11	5	1	10	-	15
2	F	3	12	4	(3)	15	-

AF (3)
DF (3)

Continue in this way until all vertices are selected, remembering to record the order in which the edges are added

		A	B	C	D	E	F
1	A	-	4	9	8	11	(3)
4	B	4	-	13	2	5	12
5	C	9	13	-	7	(1)	4
3	D	8	(2)	7	-	10	3
6	E	11	5	1	10	-	15
2	F	3	12	(4)	(3)	15	-

AF (3)
DF (3)
BD (2)
CF (4)
CE (1)



b) State the lowest cost of connecting all of the buildings to the electricity supply.

Add up the weights of the edges in the minimum spanning tree

$3 + 3 + 2 + 4 + 1 = \boxed{13}$

3.10.4 Chinese Postman Problem

Eulerian Trails & Circuits

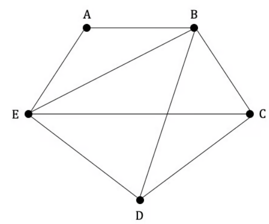
What are Eulerian trails and circuits?

- A **trail** is a walk in which **no edge is repeated**
- An **Eulerian trail** is a trail that visits each **edge** in a graph **exactly once**
- A **circuit** is a trail that begins and ends at the **same vertex**
- An **Eulerian circuit** is a trail that visits each **edge** in a graph **exactly once** and begins and ends at the **same vertex**
- A graph which contains an **Eulerian circuit** is called an **Eulerian graph**
  - In an **Eulerian graph** the degree of each vertex is **even**
- A **semi-Eulerian graph** contains an **Eulerian trail** but **not** an **Eulerian circuit**
  - In a **semi-Eulerian** graph exactly **one pair** of vertices have an **odd** degree
  - These are the **start** and **finish** points of any **Eulerian trail**
- An **adjacency matrix** can be used to determine if a graph is Eulerian or semi-Eulerian as the **degree** of each vertex can be found by inspecting the **sum of the entries** in the rows (out-degree) or columns (in-degree)

> **Exam Tip**
>
> - If you can draw a graph without taking your pen off the paper and without going over any edge more than once then you have an Eulerian or semi-Eulerian graph!

Worked example

Let G be the graph shown below.



a) Show that G is a semi-Eulerian graph.

Look at the degree of each vertex

A : 2
B : 4
C : 3
D : 3
E : 4

G is a semi-Eulerian graph because it has exactly one pair of odd vertices C and D

b) Write down an Eulerian trail for G.

An Eulerian trail must start and finish at C/D

There are several possible Eulerian trails one solution is:
D E A B E C D B C

Chinese Postman Problem

The **Chinese postman problem** requires you to find the **route of least weight** that **starts** and **finishes** at the **same vertex** and traverses **every edge** in the graph. Some edges may need to be traversed twice and the challenge is to **minimise** the total weight of these **repeated edges**.

How do I solve the Chinese postman problem?

- If **all** of the vertices in a graph are **even** then the shortest route will be the sum of the weights of the edges in an **Eulerian circuit**
- If there is **one pair** of **odd vertices** in the graph then the **shortest** route between them will need to be found and repeated before finding an Eulerian circuit
  - There will always be an **even number of odd vertices** as the **total sum of the degrees** of the vertices is **double** the **number of edges**
- If there are **more than two odd vertices**, then **each possible pairing** of the odd vertices must be considered in order to find the **minimum weight** of the edges that need to be repeated
- The **maximum** number of odd vertices that could appear in an exam question is **4**

What are the steps of the Chinese postman algorithm?

- STEP 1
  Inspect the degree of all of the vertices and identify any odd vertices
- STEP 2
  Find the possible pairings between the odd vertices
- STEP 3
  For each possible combination of vertices, find the shortest walk between the vertices and add the edges to be repeated to the graph
- STEP 4
  Write down an Eulerian circuit of the adjusted graph to find a possible route and find the sum of the edges traversed to find the total weight

What variations may there be on the Chinese postman algorithm?

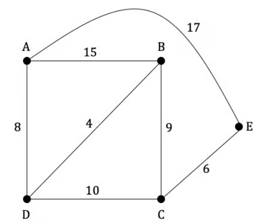
- The **weighting** of the edge between a pair of vertices may be different depending on if it is the **first time** it is being traversed or a **repeat**.
  - For example, if an inspector was checking a pipeline for defects then the first time going along a section of pipeline could take longer during inspection than if it is being repeated in order to get from one vertex to another
- If there are **4 odd vertices** you may be asked to **start** and **finish** at **different vertices**. Find the length of the routes for **all** possible pairings of the odd vertices and choose the **shortest route** between any 2 of them to be **repeated**. The other two odd vertices will be your start and finish points.

Exam Tip

- Look carefully for the shortest route between two vertices exam questions often have graphs where a combination of edges will turn out to be a shorter distance than a more direct route

Worked example

The graph G shown below displays the distances, in kilometres, of the main roads between towns A, B, C, D and E. Each road is to be inspected for potholes.



a) Explain why G does not contain an Eulerian circuit.

Inspect the degree of each vertex

A : 3 (odd)
B : 3 (odd)
C : 3 (odd)
D : 3 (odd)
E : 2 (even)

The graph G does not contain an Eulerian circuit as some of the vertices are odd

b) Find the shortest route that starts and finishes at town A and allows for each road to be inspected.

There are 4 odd vertices so all of the possible pairings must be considered to find the shortest distance that needs to be repeated

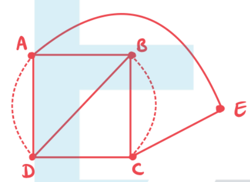
Possible pairings:

AB + CD = 12 (ADB) + 10 (CD) = 22

AC + BD = 18 (ADC) + 4 (BD) = 22

AD + BC = 8 (AD) + 9 (BC) = 17 * shortest

Add the edges to be repeated onto the graph and find an Eulerian circuit starting from A



One possible Eulerian circuit is:

A D A B D C B C E A

c) State the total length of the shortest route.

Add the length of each edge in the graph then add the weight of the repeated edges

15 + 17 + 8 + 4 + 9 + 10 + 6 (Edges in the original graph) + 17 (Repeated edges) = **86 km**

3.10.5 Travelling Salesman Problem

Hamiltonian Paths & Cycles

What are Hamiltonian paths and cycles?

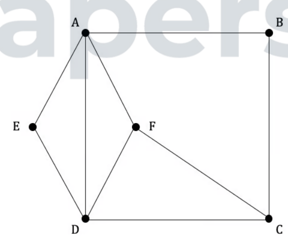
- A **path** is a walk in which **no vertices are repeated**
  - A **Hamiltonian path** is a path in which each **vertex** in a graph is visited **exactly once**
- A **cycle** is a walk that starts and ends at the **same vertex** and **repeats no other vertices**
  - A **Hamiltonian cycle** is a cycle which visits each **vertex** in a graph **exactly once**
- If a graph contains a **Hamiltonian cycle** then it is known as a **Hamiltonian graph**
- A graph is **semi-Hamiltonian** if it contains a **Hamiltonian path** but not a **Hamiltonian cycle**
- The only way to show that a graph is **Hamiltonian** or **semi-Hamiltonian** is to find a **Hamiltonian cycle** or **Hamiltonian trail**

Exam Tip

- If you are given an adjacency matrix and are asked to find a Hamiltonian cycle, make sure that you sketch out the graph first

Worked example

Let G be the graph shown below.



Show that G is a Hamiltonian graph.

To show that the graph is Hamiltonian, identify a Hamiltonian cycle

One possible Hamiltonian cycle is:

A B C F D E A

Travelling Salesman Problem

The **travelling salesman problem** requires you to find the **route of least weight** that **starts** and **finishes** at the **same vertex** and visits every other **vertex** in the graph **exactly once**.

How do I solve the travelling salesman problem?

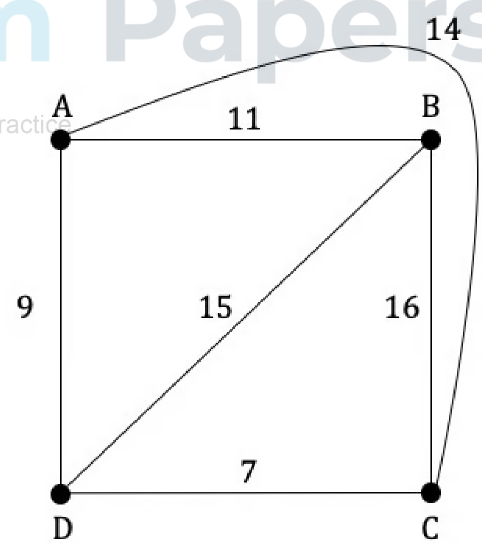
- In the **classical travelling salesman problem** the following conditions are observed:
  - the graph is **complete**
  - the **direct route** between two vertices is the **shortest route** (it satisfies the triangle inequality)
- List **all** of the possible **Hamiltonian cycles** and find the cycle of **least weight**
  - A complete graph with 3 vertices will have 2 possible Hamiltonian cycles, 4 vertices will have 6 possible cycles and a graph with 5 vertices will have 24 possible cycles
- There is **no known algorithm** that guarantees finding the shortest Hamiltonian cycle in a graph so this method is only suitable for **small graphs**

Exam Tip

- To remember the difference between the travelling salesman problem and the Chinese postman problem, remember that the salesman is interested in selling at each destination (vertex) whereas the postman wants to walk along every road (edge) in order to deliver the letters

Worked example

The graph below shows four towns and the distances between them in km.



A salesman lives in city A and wishes to travel to each of the other three cities before returning home.

Find the shortest route that the salesman could take and state the total length of the route.

List all six possible Hamiltonian cycles and their weight

A B C D A = 11 + 16 + 7 + 9 = 43

A D C B A = 43

A B D C A = 11 + 15 + 7 + 14 = 47

A C D B A = 47

A D B C A = 9 + 15 + 16 + 14 = 54

A C B D A = 54

The route is the same but in reverse of the one above and therefore has the same weight

Route of least weight = 43 km

A B C D A

OR

A D C B A

3.10.6 Bounds for Travelling Salesman Problem

This revision note discusses more complex situations for the **travelling salesman problem** and you may wish to refer to the revision note **3.10.5 Travelling Salesman Problem**.

Table of Least Distances

In some real-life contexts a graph may not be **complete** nor satisfy the **triangle inequality**, for example, when looking at a rail network, not every stop will be connected to every other stop and it may be quicker to travel from stop A to stop B via stop C rather than to travel from A to B directly. Thus, the problem is considered to be a **practical travelling salesman problem**.

Finding the **table of least distances** (or weights) can convert a **practical travelling salesman problem** into a **classical travelling salesman problem** that can then be analysed.

What is a table of least distances?

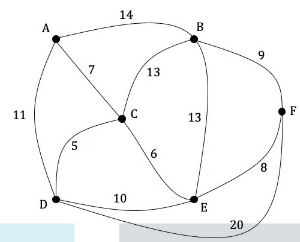
- A **table of least distances** shows the **shortest** distance between any **two vertices** in a graph
  - In some cases, the **direct route** between two vertices may **not** be the **shortest**
- By finding the **table of least distances**, a graph can be converted into a **complete** graph that satisfies the **triangle inequality**
- STEP 1
  Fill in the information for vertices that are adjacent in the graph (at this stage check if the direct connections are actually the shortest route)
- STEP 2
  Complete the rest of the table by finding the shortest route that can be travelled between each pair of vertices that are not adjacent

Exam Tip

- Remember that the table of least values has a line of symmetry along the leading diagonal for an undirected graph, so complete one half carefully first, then map over to the second half

Worked example

The graph *G* below contains six vertices representing villages and the roads that connect them. The weighting of the edges represents the time, in minutes, that it takes to walk along a particular road between two villages.



a) Explain why *G* is not complete graph.

G is not a complete graph as each vertex is not connected to every other vertex by a single edge, e.g. there is no single edge connecting B and D

b) Complete the table of least weights below.

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

Fill in the table with the direct connections from the graph but check that they are the shortest routes as you go along

Shortest route between D and F : D → F = 20
D → E → F = 18

Remember that in an undirected graph there will be a line of symmetry along the leading diagonal so start by filling in one half

	A	B	C	D	E	F
A	–					
B	14	–				
C	7	13	–			
D	11		5	–		
E		13	6	10	–	
F		9		18	8	–

Find the shortest routes between unconnected vertices

A and E : A → C → E = 13

A and F : A → C → E → F = 21

B and D : B → C → D = 18

C and F : C → E → F = 14

	A	B	C	D	E	F
A	–	14	7	11	13	21
B	14	–	13	18	13	9
C	7	13	–	5	6	14
D	11	18	5	–	10	18
E	13	13	6	10	–	8
F	21	9	14	18	8	–

Nearest Neighbour Algorithm

As the number of vertices in a graph increases, so does the number of possible Hamiltonian cycles and it can become impractical to solve. The **nearest neighbour algorithm** can be used to find the **upper bound** for the **minimum** weight **Hamiltonian cycle**.

What is the nearest neighbour algorithm?

- For a complete graph with at least 3 vertices, performing the **nearest neighbour algorithm** will generate a **low** (but not necessarily least) **weight** Hamiltonian cycle
- This low weight cycle can be considered the **upper bound**
- The **best upper bound** is the upper bound with the **smallest value**
- The nearest neighbour algorithm can only be used on a graph that is **complete** and satisfies the **triangle inequality** so the **table of least distances** should be found first

What are the steps of the nearest neighbour algorithm?

- STEP 1
  Choose a starting vertex
- STEP 2
  Follow the edge of least weight from the current vertex to an adjacent unvisited vertex (if there is more than one edge of least weight pick one at random)
- STEP 3
  Repeat STEP 2 until all vertices have been visited
- STEP 4
  Add the final edge to return to the starting vertex

Exam Tip

- If asked to write down the route for the lower bound, don't forget that some of the entries in the table of lowest distances may not be direct routes between vertices!

Worked example

The table below contains six vertices representing villages and the roads that connect them. The weighting of the edges represents the time in minutes that it takes to walk along a particular road between two villages.

	A	B	C	D	E	F
A	-	14	7	11	13	21
B	14	-	13	18	13	9
C	7	13	-	5	6	14
D	11	18	5	-	10	18
E	13	13	6	10	-	8
F	21	9	14	18	8	-

Starting at village A, use the nearest neighbour algorithm to find the upper bound of the time it would take to visit each village and return to village A.

Start at vertex A (column A) and select the edge of least weight (AC), write it down

Move to column C, cross out the value that would join C to A as vertex A has already been visited. Select the edge of least weight (CD) and write down

Move to column D, cross out the values for the vertices already visited (A and C) and select the edge of least weight from the remaining values and write it down

Continue until all vertices have been visited then choose the final edge to get back from the last vertex to the starting position

	A	B	C	D	E	F
A	–	(14)	~~7~~	~~11~~	~~13~~	~~21~~
B	14	–	13	18	13	(9)
C	(7)	13	–	~~5~~	~~6~~	~~14~~
D	11	18	(5)	–	~~10~~	~~18~~
E	13	13	6	(10)	–	~~8~~
F	21	9	14	18	(8)	–

A $\xrightarrow{7}$ C $\xrightarrow{5}$ D $\xrightarrow{10}$ E $\xrightarrow{8}$ F $\xrightarrow{9}$ B $\xrightarrow{14}$ A

53 mins is an upper bound

Deleted Vertex Algorithm

The **deleted vertex algorithm** can be used to find the **lower bound** for the **minimum** weight **Hamiltonian cycle**.

What is the deleted vertex algorithm?

- The deleted vertex algorithm can only be used on a graph that is **complete** and satisfies the **triangle inequality** so the **table of least distances** should be found first
- Deleting **different** vertices may give different results, the **best lower bound** is the lower bound with the **highest value**
- If you have found a **cycle** the **same length** as the **lower bound** then you have found the **shortest route** for the travelling salesman problem
- If the **lower bound** and the **upper bound** are the **same weight** then you have found the **shortest route** for the travelling salesman problem

What are the steps of the deleted vertex algorithm?

- STEP 1
  Choose a vertex and delete it along with all edges that are connected to it
- STEP 2
  Find the minimum spanning tree for the remaining graph (see revision note **3.10.3 Minimum Spanning Trees**)
- STEP 3
  Add the two shortest edges that were deleted from the original graph to the weight of the minimum spanning tree

Exam Tip

- Be careful when using a **weighted adjacency table** not to get confused between using Prim's algorithm and the nearest neighbour algorithm.
  - Remember that **Prim's** is used to find a minimum spanning tree, so vertices can be connected to **several** other vertices and hence can have **more than one value** in a column circled
  - When using the table for the **nearest neighbour** algorithm, vertices **cannot be revisited** so **only one value** will be circled in each column

Worked example

The table below contains six vertices representing villages and the roads that connect them. The weighting of the edges represents the time in minutes that it takes to walk along a particular road between two villages.

	A	B	C	D	E	F
A	-	14	7	11	13	21
B	14	-	13	18	13	9
C	7	13	-	5	6	14
D	11	18	5	-	10	8
E	13	13	6	10	-	8
F	21	9	14	18	8	-

a) By deleting vertex A and using Prim's algorithm, find a lower bound for the time taken to start at village A, visit each of the other villages and return to village A

		A	B	C	D	E	F
	A	–	14	7	11	13	21
(1)	B	14	–	13	18	13	(9)
(4)	C	7	13	–	(5)	6	14
(5)	D	11	18	5	–	10	18
(3)	E	13	13	(6)	10	–	8
(2)	F	21	9	14	18	(8)	–

Add edges in the order: BF (9)
FE (8)
CE (6)
CD (5)

Total weight of minimum spanning tree = 28

Add weights of two edges of least weight connected to A: AC (7)
AD (11)

Lower bound = 28 + 7 + 11 = **46 minutes**

b) Show that by deleting vertex B instead, a higher lower bound can be found.

		A	B	C	D	E	F
①	A	–	14	⑦	11	13	21
	B	14	–	13	18	13	9
②	C	7	13	–	⑤	⑥	14
③	D	11	18	5	–	10	18
④	E	13	13	6	10	–	⑧
⑤	F	21	9	14	18	8	–

Total weight of minimum spanning tree = 7 + 8 + 6 + 5
= 26

Weight of two edges of least weight connected to B : BF (9)
BC/D (13)

Lower bound = 48 mins

This is a higher lower bound than found when deleting vertex A