Applied Mathematics

5th Year

Dr. Brendan Williamson

PUBLISHED BY INSTITUTE OF EDUCATION

BRENDANWILLIAMSON.IE

First printing, August 2023

Contents

Introduction v

I Graph Theory

1 New Syllabus Exam Questions 3
1.1 A Not So New Syllabus 3
1.2 Moodle Resources 3
1.3 New Syllabus Applied Maths Papers 3

2 Introduction to Graphs 5
2.1 What is a Graph? 5
2.2 Graph Types, Properties and Definitions 5
2.3 Adjacency Matrices 13
2.4 Adjacency Matrices for Digraphs 17
2.5 Trees 19
2.6 Exam Questions 21
2.7 Summary 23
2.8 Notes on the Exam, and Work Still to Cover 24
2.9 Homework 25
2.10 Homework Solutions 31
2.11 Revision 35

3 Algorithms on Graphs 37
3.1 What is an Algorithm? 37
3.2 Kruskal's Algorithm 37
3.3 Prim's Algorithm 41
3.4 Prim's Algorithm: Matrix Form 44

3.5 Notes on Kruskal and Prim's Algorithm . . . 47
3.5.1 Similarities . . . 47
3.5.2 Differences . . . 47
3.6 Dijkstra's Algorithm . . . 48
3.7 Dijkstra's Algorithm for Directed Graphs . . . 52
3.8 Notes on Dijkstra's Algorithm . . . 55
3.9 Exam Questions . . . 56
3.9.1 Prim & Kruskal's Algorithm . . . 56
3.9.2 Dijkstra's Algorithm . . . 63
3.10 Summary . . . 68
3.11 Notes on the Exam, and Work Still to Cover . . . 69
3.12 Homework . . . 70
3.13 Homework Solutions . . . 73
3.14 Revision . . . 78

4 Project Scheduling . . . 79
4.1 Planning a Project Efficiently . . . 79
4.2 Precedence Tables and Activity Networks . . . 79
4.3 Time to Complete a Project . . . 95
4.4 Cascade Charts and Scheduling . . . 102
4.5 Scheduling with a Limited Workforce . . . 110
4.6 Exam Questions . . . 115
4.7 Summary . . . 129
4.8 Notes on the Exam, and Work Still to Cover . . . 130
4.9 Homework . . . 131
4.10 Homework Solutions . . . 136
4.11 Revision . . . 144
4.12 Cascade Chart Templates . . . 145

5 Dynamic Programming . . . 149
5.1 Starting at the End . . . 149
5.2 Routing Problems . . . 149
5.3 A Note on Dynamic Programming . . . 157
5.4 Stock Control . . . 158
5.5 Allocation of Resources . . . 161
5.6 Equipment Replacement and Maintenance . . . 163
5.7 Exam Questions . . . 166
5.8 Summary . . . 170
5.9 Notes on the Exam, and Work Still to Cover . . . 171
5.10 Homework . . . 172
5.11 Homework Solutions . . . 176
5.12 Revision . . . 182

II Mechanics

6 Linear Motion . . . 185
6.1 Some Initial Terminology . . . 185
6.2 Five Quantities, Five Equations . . . 187
6.3 Freefall . . . 192
6.4 Multi-Part Journeys . . . 195
6.5 More Complex Multi-Part Journeys . . . 198
6.6 Velocity-Time Graphs . . . 206
6.7 Multi-Object Journeys . . . 208
6.8 More Algebraic Problems . . . 215
6.9 Displacement-Time Graphs . . . 219
6.10 Exam Questions . . . 221
6.11 Summary . . . 234
6.12 Notes on the Exam, and Work Still to Cover . . . 235
6.13 Homework . . . 236
6.14 Homework Solutions . . . 239
6.15 Revision . . . 253

7 Connected Particles . . . 255
7.1 Newton's Three Laws of Motion . . . 255
7.2 Sources of Forces and Basic Problems . . . 256
7.3 More Advanced Problems 1 . . . 260
7.4 Resolving Forces . . . 268
7.5 More Advanced Problems 2 . . . 272
7.6 More Advanced Problems 3 . . . 278
7.7 Exam Questions . . . 285
7.8 Summary . . . 287
7.9 Notes on the Exam, and Work Still to Cover . . . 288
7.10 Homework . . . 289
7.11 Homework Solutions . . . 295
7.12 Revision . . . 312

8 Vectors . . . 313
8.1 The $\vec{i} - \vec{j}$ plane and Vector Arithmetic . . . 313
8.2 Polar Form Vs Rectangular Form . . . 318
8.3 The Dot Product and Angles Between Vectors . . . 321
8.4 Miscellaneous Facts About Vectors . . . 323
8.5 Summary . . . 324
8.6 Notes on the Exam, and Work Still to Cover . . . 324
8.7 Revision . . . 325
8.8 Revision Solutions . . . 326

9 Circular Motion . . . 329

9.1 Motion in a Circle, Periodic Motion, and Radians . . . 329
9.2 Centripetal Acceleration . . . 331
9.3 Basic Problems . . . 334
9.4 Motion in a Horizontal Circle . . . 336
9.5 Horizontal Circular Motion Due to Friction . . . 340
9.6 Principle of Conservation of Energy . . . 347
9.7 Motion in a Vertical Circle . . . 350
9.8 Hooke's Law . . . 354
9.9 Exam Questions . . . 359
9.10 Summary . . . 361
9.11 Notes on the Exam, and Work Still to Cover . . . 362
9.12 Homework . . . 362
9.13 Homework Solutions . . . 367
9.14 Revision . . . 383
9.15 Revision Solutions . . . 387

10 Collisions . . . 407

10.1 Momentum, Impulse and Elasticity . . . 407
10.2 Direct Collisions . . . 411
10.3 Direct Collisions Exam Questions . . . 417
10.4 Vertical Circular Motion and Collisions . . . 422
10.5 Oblique Collisions . . . 424
10.6 More Algebraic Oblique Collisions . . . 429
10.7 Oblique Collisions Exam Questions . . . 434
10.8 Summary . . . 436
10.9 Notes on the Exam, and Work Still to Cover . . . 437
10.10 Homework . . . 438
10.11 Homework Solutions . . . 441
10.12 Revision . . . 447

11 Projectiles . . . 449

11.1 From One Dimension to Two . . . 449
11.2 Basic Projectile Problems . . . 449
11.3 Unknown Initial Speed and/or Projection Angle . . . 455
11.4 Projectiles That Bounce . . . 479
11.5 Exam Questions . . . 489
11.6 Summary . . . 493
11.7 Notes on the Exam, and Work Still to Cover . . . 494
11.8 Homework . . . 495
11.9 Homework Solutions . . . 498
11.10 Revision . . . 507

Introduction

This book is designed to be read as a single document, in the order in which the chapters appear. In each chapter questions, examples, definitions etc. are on a common numbering system, meaning that Definition 4.7 is the 7th text box in Chapter 4. Similarly figures are on a separate common numbering system, so that Figure 7.12 is the 12th picture in Chapter 7. This book also appears in pdf format on Moodle, in which all references are hyperlinked for easier navigation.

At the end of each chapter is a summary, which explains how the content of the chapter fits into the Leaving Cert syllabus and maps onto past (and possibly future) exam questions, as well as commenting on the differences between the old (pre-2023) and new syllabus. There is also a set of homework problems (separated by section) with solutions provided at the end of the chapter. This is to be completed as we work through the chapter. There is also a revision section at the end of each chapter for students re-reading the chapter or preparing for a test.

This book is comprehensive, assuming no knowledge of Applied Maths beyond this book and so you don't need any other book or notes to study for the course. It is also designed so that it can be read as revision weeks or months after we first cover them. As such you shouldn't need to spend much time taking notes and can instead concentrate on the class. However you may want to take notes occasionally if there is something mentioned in class that isn't covered clearly in the book.

If you have any questions about anything you can reach me at
bwilliamson@instituteofeducation.ie.

Graph Theory

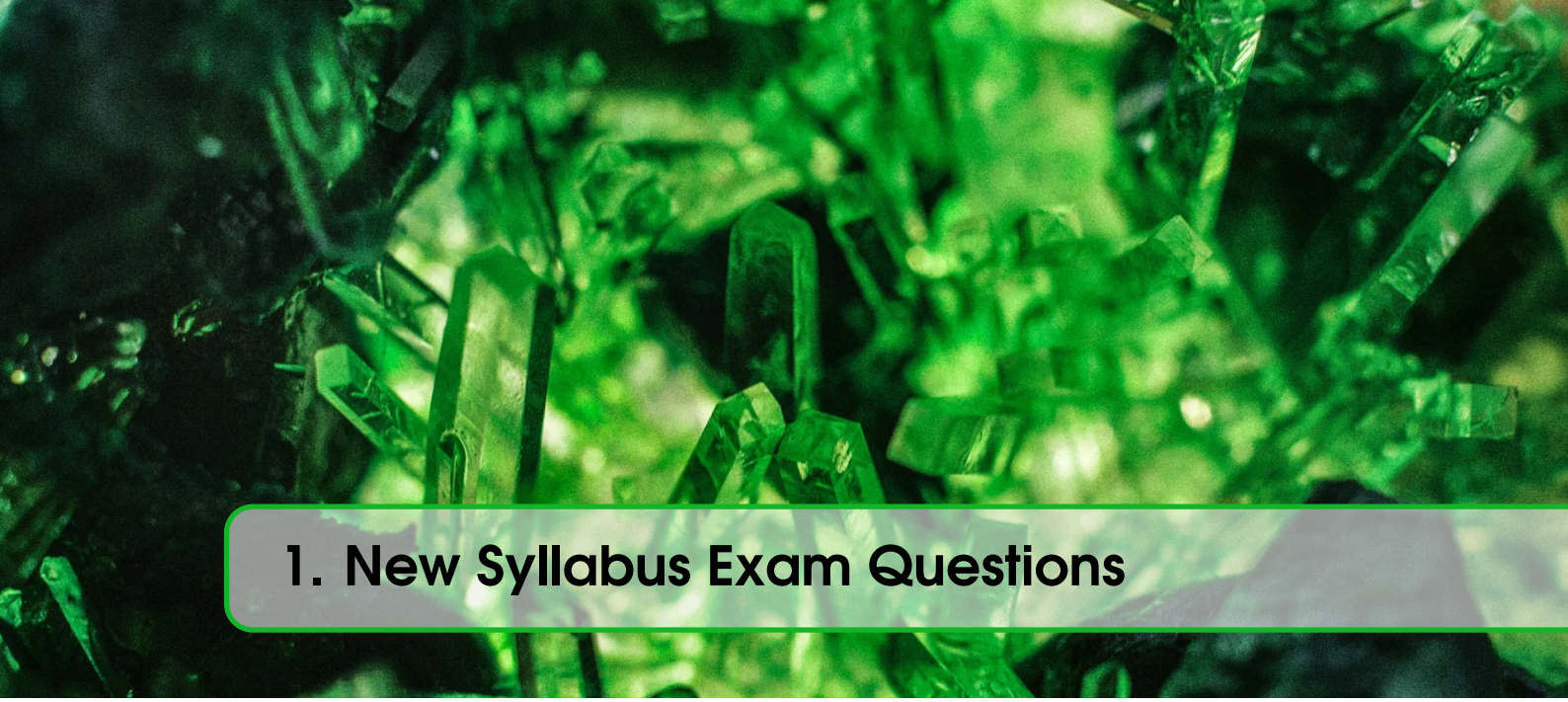
1 New Syllabus Exam Questions 3

2 Introduction to Graphs 5

3 Algorithms on Graphs 37

4 Project Scheduling 79

5 Dynamic Programming 149



1. New Syllabus Exam Questions

1.1 A Not So New Syllabus

Although we have few past papers for the new Applied Maths syllabus, much of the newer parts of the syllabus have existed in an almost identical way in other subjects in other parts of the world. In particular, the topics in Part I have long been part of the syllabi of some A level exams called "Further Decision 1" and "Further Decision 2" (abbreviated to D1, D2) operated by Edexcel.

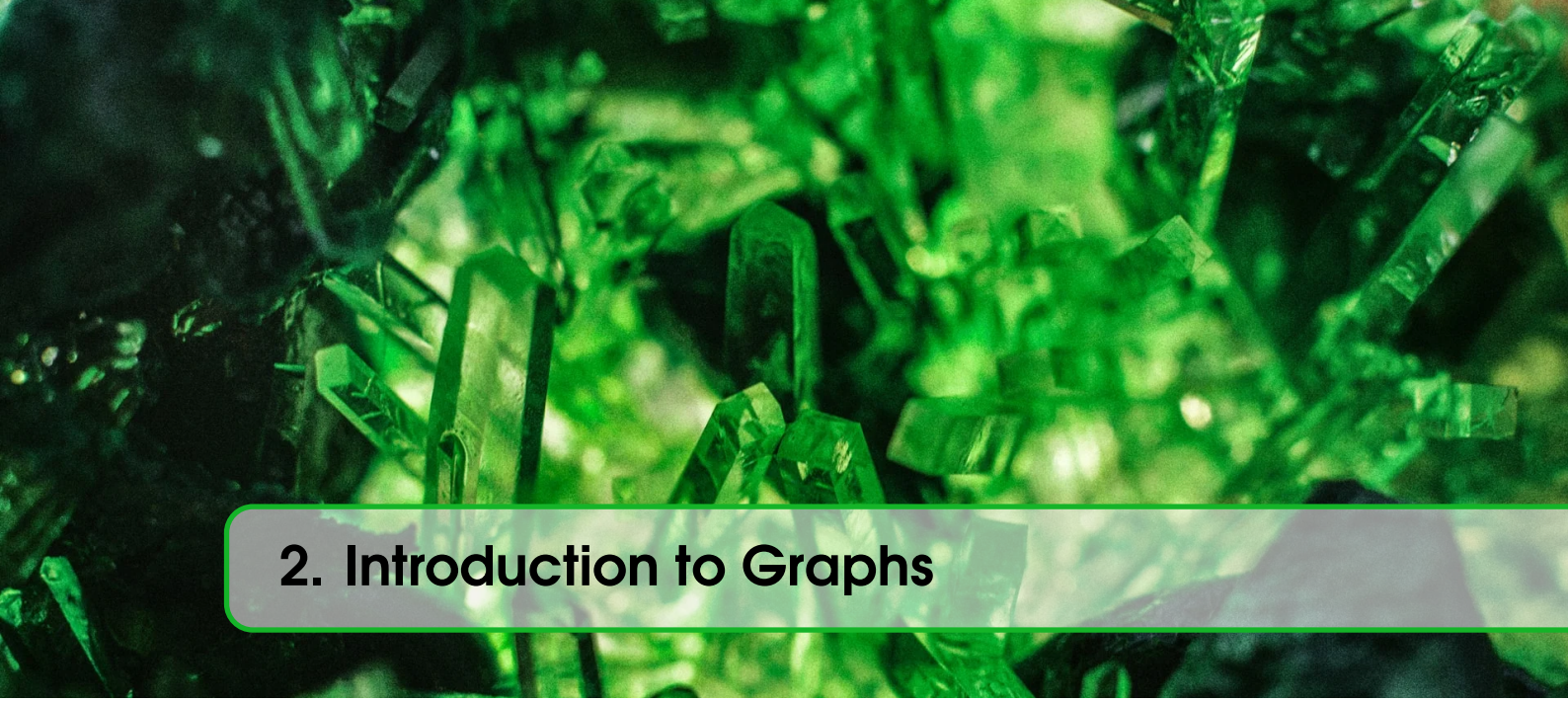
1.2 Moodle Resources

There is a zip file on Moodle titled "Edexcel Past Exams". In this folder there are 6 subfolders, one each for new and old D1, D2 and International D1 past exam papers (there is no International A Level D2 exam). As this is not our syllabus to begin with we're not concerned with syllabus changes between the new and old exams.

In each folder there are three subfolders, one containing the question papers, one containing the marking scheme, and one containing the answer booklet. The answer booklet can be ignored for most problems unless it is specifically referenced in the exam paper. Collectively these folders contain exam papers going back to 2001, often with multiple exams per year. Any question with an (R) are exams for regional areas whose time zone is not GMT and so were given on the same date but at a different time to the main exam (and so are different but equally relevant exams). There is also an Excel spreadsheet in each folder showing what questions are relevant to each topic we have studied. Everything except dynamic programming can be found in the D1 exams, with dynamic programming appearing in the D2 exams. Not everything in these exams is covered in the Applied Maths syllabus, so a blind reading without this spreadsheet is not recommended.

1.3 New Syllabus Applied Maths Papers

There is one sample paper for the Applied Maths exam released by the SEC, as well as the 2023 paper which is the first year to cover the new syllabus. These are also available on Moodle and the relevant questions will be studied in their respective chapters. The differences between these papers and the Edexcel exam papers will also be highlighted where relevant.

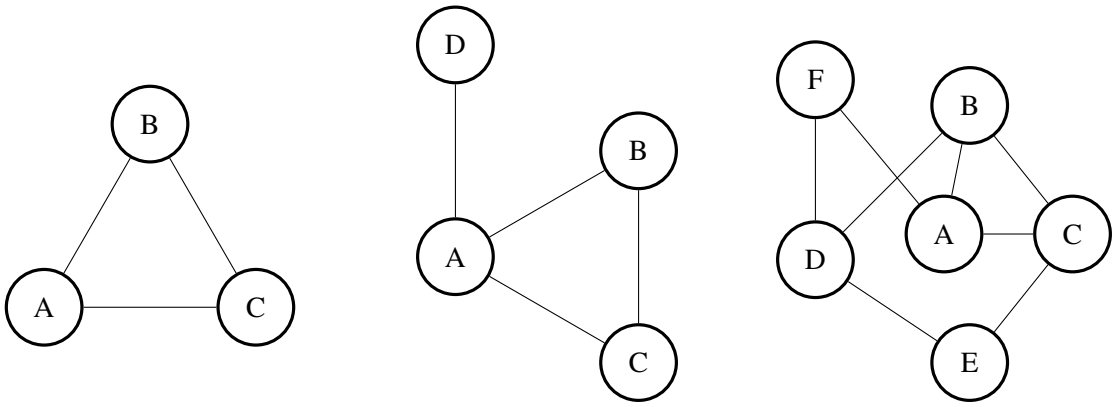


2. Introduction to Graphs

2.1 What is a Graph?

Definition 2.1 A **graph** is a collection of vertices (or nodes), and edges (arcs) connecting some or all of them.

Here are some examples of graphs.



(a) (b) (c)

Figure 2.1

We will often talk about "the vertex A", or "the edge AB". So why do we care about graphs? Graphs can provide a simple representation of many phenomena.

2.2 Graph Types, Properties and Definitions

The following is a (fictional) graph representing cities in Europe with an airport. Cities are connected by an edge if there is a direct flight between them.

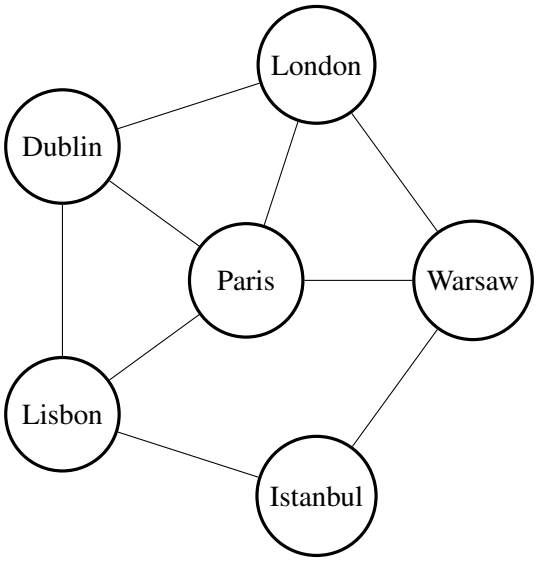
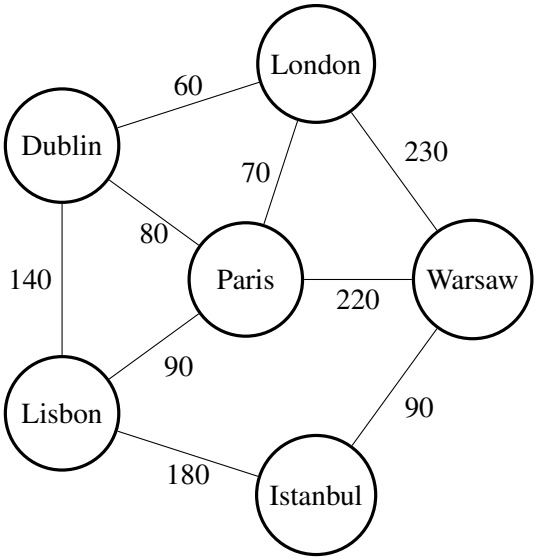


Figure 2.2

From this graph we can figure out the possible ways we can get from, say, Dublin to Warsaw. This graph can also tell us the route with the fewest number of stops. However what if we care about the cost of our trip?

Definition 2.2 A **weighted graph** is a graph where a non-negative number (called a weight) is assigned to each edge.

The following is an updated version of Figure 2.2 where the weights represent the cost, in Euros, of each flight between the two cities.

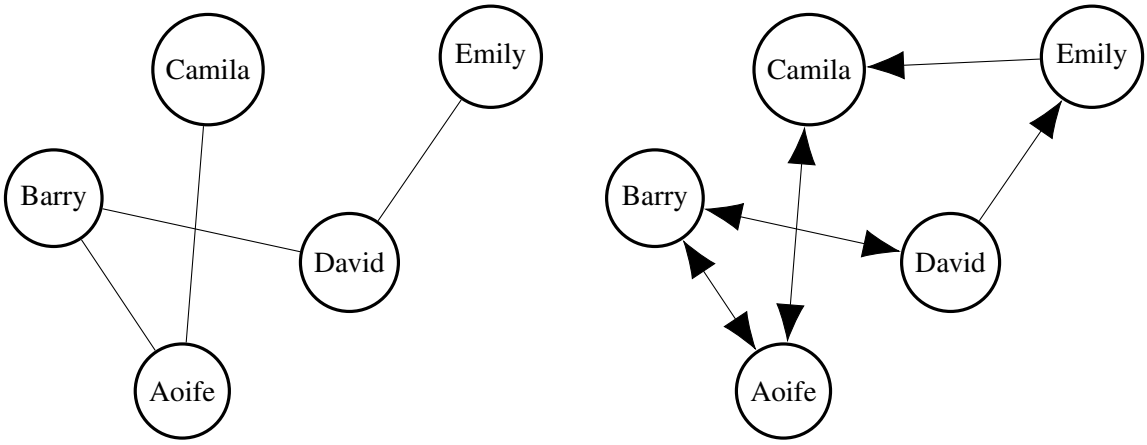


Using this graph we can not only see which routes have the shortest number of flights, but that the cheapest route is Dublin $\rightarrow$ London $\rightarrow$ Warsaw for €290.

Sometimes it makes sense to consider a connection between nodes that only goes one way.

Definition 2.3 A **directed graph** (or **digraph** for short) is a graph in which each edge has an arrow representing direction. Otherwise it is an **undirected graph**.

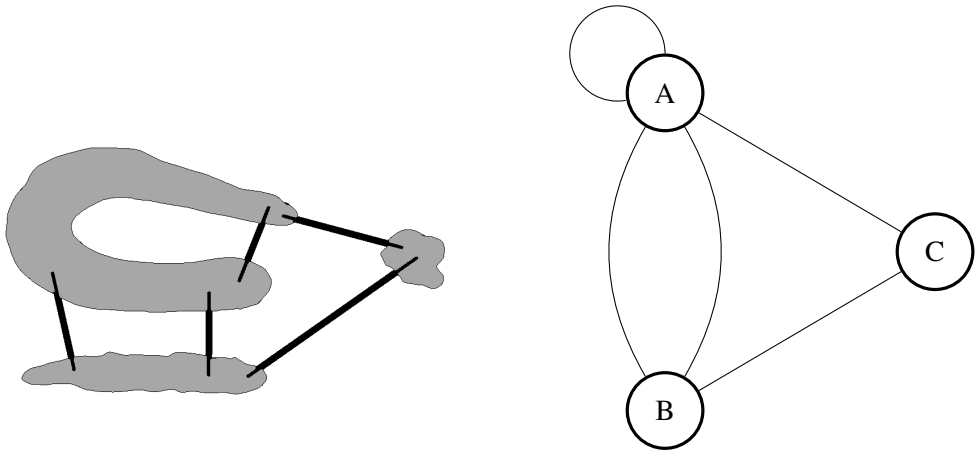
For example, the following graphs show a group of people and their connections on social media. On the first graph two people are connected by an edge if they are friends on Snapchat (which is a two-way relationship). On the second, a directed edge from one node A to another edge B is drawn if person A follows person B on Instagram (which can be a one-way relationship).



(a) Friendships on Snapchat (b) Follows on Instagram

Figure 2.3

Consider the map below that shows three islands, and the bridges that go between them. How would we represent this as a graph? We could represent each island as a vertex, and each bridge as an edge. This leads to the given graph.



(a) A map of the islands (b) A graph representation

Figure 2.4

Note that there is more than one edge between islands A and B, and that there is an edge from A back to itself. Although we haven't seen this so far, there is nothing wrong with it. However in many situation we don't expect to see these types of graphs.

Definition 2.4 A **loop** is an edge from a vertex back to itself. A graph is **simple** if it has no loops and no multiple edges between any two nodes, otherwise it is a **multigraph**. In the case of

a directed graph, edges are only considered multiple edges if they are between the same nodes and in the same direction.

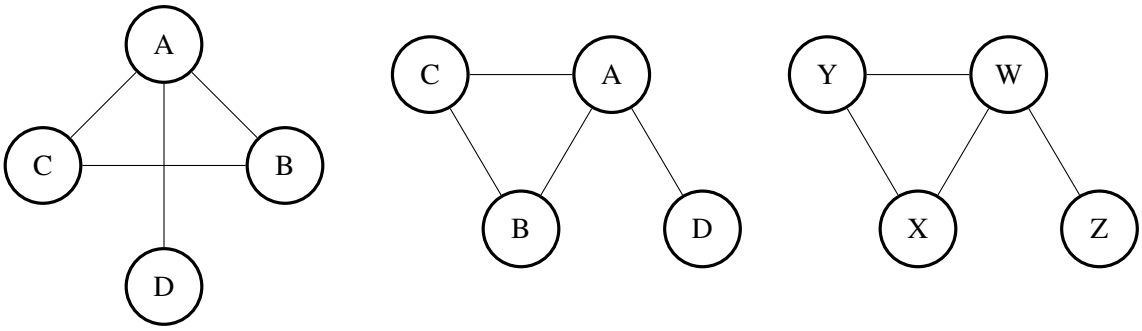
Question 2.5 Which of the following graphs would you expect to be simple, and which ones might not be?

(a) A graph where each node is a country, and two countries are connected by an edge if they played a football match against each other in 2021.

(b) A directed graph where each node is a person, and an edge with an arrow from person A to person B is drawn if person A is a parent of person B.

(c) A directed graph where each node is a person, and an edge with an arrow from person A to person B is drawn if person A gave a haircut to person B during the Covid-19 lockdown.

You may have noticed by now that there are multiple ways to draw the same graph, or describe the same situation using a graph. The three graphs below are the same.



(a) (b) (c)

Figure 2.5

Even changing the names of the vertices doesn't change that it's the same graph, much like the equations $x^2+7x+12=0$ and $t^2+7t+12=0$ are the same equations. If your friend is studying one and you're studying another, you're really looking at the same thing.

Definition 2.6 Two graphs which contain the same number of vertices connected by edges in the same way are **identical** or **isomorphic**. In the case of weighted and/or directed graphs, the edges must also have the same weight and/or direction, and in the cases of multigraphs multiple edges and loops must appear in the same way.

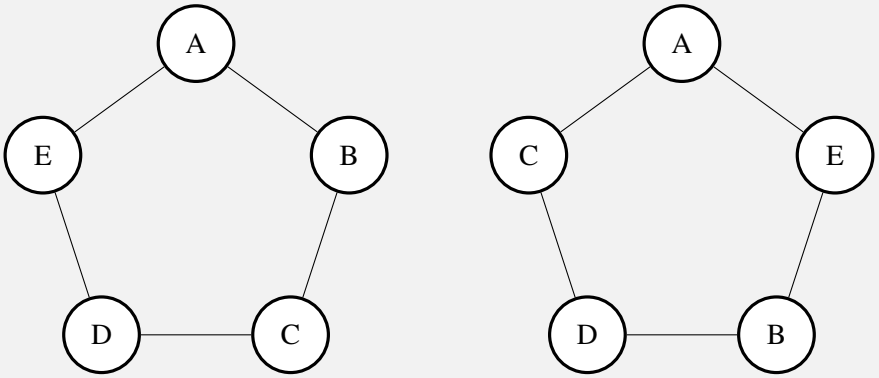
Note 2.7 To see if two graphs are isomorphic, there are a couple of approaches. You can look at one graph, imagine moving the vertices around and see if you can make it look like the other graph, ignoring vertex names. This can actually be done online, at csacademy.com/app/graph_editor, where you can physically drag the vertices around and watch the visualisation of the graph change. You can also take a more mathematical approach by associating each vertex in one graph with a vertex in one another without doubling up, in an "edge-preserving way". In the case of the

second and third graph above, we have the following association.

$$A \to W,$$
$$B \to X,$$
$$C \to Y,$$
$$D \to Z.$$

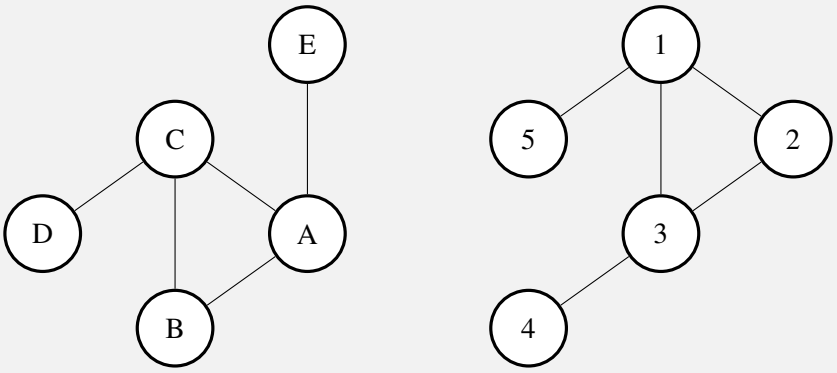
This association is edge preserving, because in the first graph AB is an edge, and in the second graph WX is an edge. On the other hand, BD is not an edge, and neither is XZ.

Question 2.8 Which of the following pairs of graphs are isomorphic?



(a) (b)

Figure 2.6



(a) (b)

Figure 2.7

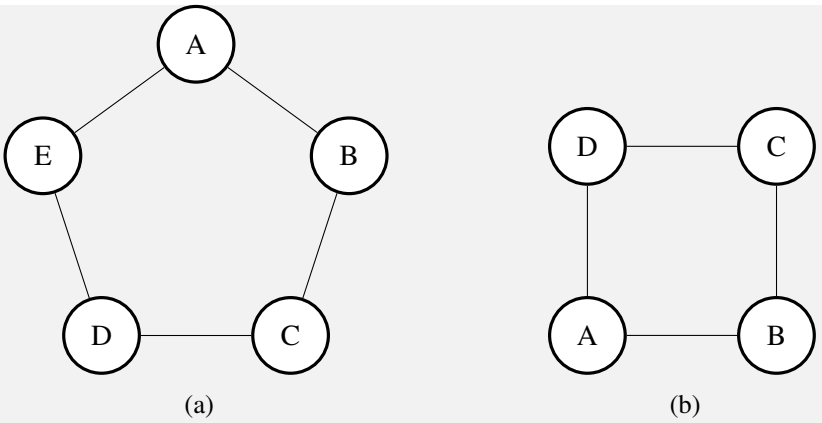


Figure 2.8

Note 2.9 If two graphs have a different number of vertices or a different number of edges they cannot be isomorphic. Moreover, if one graph has a vertex with, say, 5 edges coming out of it, and there is no vertex in the other graph with 5 edges coming out of it then they can't be isomorphic.

When discussing the graph of airports and their connecting flights we talked about using the graph to figure out how to get from Dublin to Warsaw. This is a common topic of study when looking at graphs; excursions around the graphs via the edges.

Definition 2.10

1. If two vertices have an edge joining them, we say they are **adjacent**. The joining edge is said to be **incident** with each of the vertices. When talking about incidence/adjacence in a directed graph we ignore direction.
2. The number of edges incident to a vertex is the **degree**, or **order** of the vertex. Loops count as 2 edges.
3. A **walk** of length n is a sequence of $n+1$ vertices, where consecutive vertices are adjacent. It is of length n because instead of considering the $n+1$ vertices we can consider the n relevant incident edges. In the case of a directed graph the edge direction must be from the previous vertex to the next one. A walk is **closed** if the first and last vertex are the same, and **open** if they are distinct.
4. A **path** is a walk in which no vertex is repeated. All paths are open.
5. A **cycle** is a closed walk in which no vertices (except for the first and last) are repeated.

The degree of a vertex is not a concept we will use very often, save for this result we will use later.

Theorem 2.11 — Hand-shaking Lemma. The sum of the degrees of all vertices in a graph is equal to twice the number of edges.

We won't use the concept of degree or this result in any complicated mathematics. However later on when we list edges we will use it to make sure we haven't missed any edges.

Described less formally, a walk is any route in a graph between vertices along edges. A path is a

walk that doesn't visit any vertex more than once, and a cycle is a walk that begins and ends at the same vertex, but otherwise doesn't visit any vertex more than once. Consider the following graph.

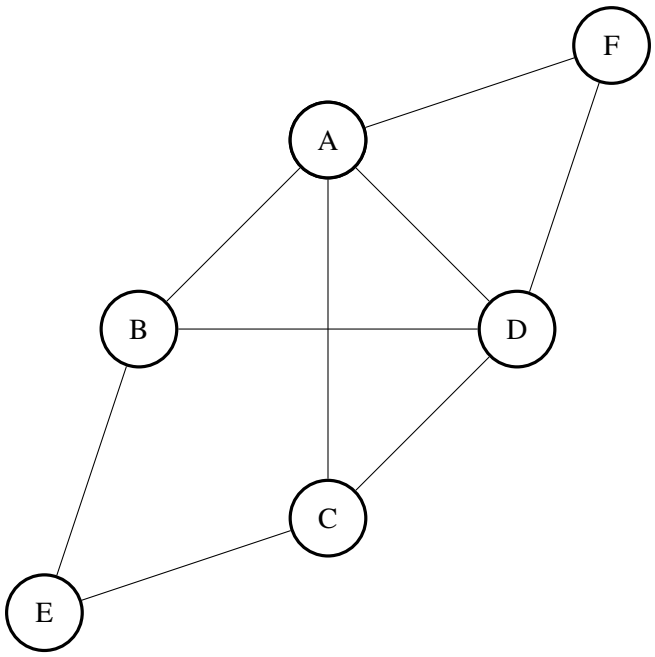


Figure 2.9

Here *ABDACD* is an open walk, but it is not a path because *D* and *A* are visited more than once. *ABDF* is a path, and *ABDFA* is a cycle. *ABCD* isn't even a walk, because there is no edge connecting *B* and *C*.

Note 2.12 Between walks, paths and cycles you can remember which is which in the following way.

Type	In common usage	Mnemonic
WALK	Walks around town can go in circles all day.	**WA**nder as you **LiK**e
PATH	Paths are direct even if they are not straight.	**PA**r**T(H)**icular about the nodes.
CYCLE	Cycles are sequences that repeat in a loop.	**CYCLE** $\approx$ **CIRCLE**

Table 2.1

Sometimes we are interested not in particular paths between vertices, but whether such a path exists at all.

Definition 2.13
A graph is **connected** if there is a path between any two vertices. Otherwise it is **disconnected**.

Question 2.14 Which of the following graphs would you expect to be connected?

(a) Each vertex is a town/city in the island of Ireland with a train station. Two vertices are adjacent if they are neighbouring stops on a train line.

(b) Each vertex is a town/city in the UK (so including Northern Ireland) with a train station. Two vertices are adjacent if they are neighbouring stops on a train line.

(c) Each vertex is a student in this class. Two students are adjacent if they ever went to the same secondary school (even at different times).

(d) Each vertex is a county in Ireland. Two counties are adjacent if they share a border.

If we're dealing with disconnected graphs, we may be interested in looking at one part of the graph. This leads to the following definitions.

Definition 2.15

1. The graph H is a **subgraph** of the graph G if all of the vertices and edges in H are vertices and edges in G.
2. A **connected component** H of a graph G is a connected subgraph such that if A is a vertex in H, all edges incident to A in G are in H.

Note 2.16 Less formally, a subgraph H of a graph G is the graph G with some (or maybe none, technically G is a subgraph of itself) of its vertices and edges removed. Note that if a vertex is removed all incident edges must be removed; we can't have edges going to nowhere.

Note 2.17 Connected components are a more complex idea. According to our definition, a connected component H of G that contains A must contain all edges incident to A. This means that all vertices adjacent to A must also be in H, along with all edges incident to them, and so on. Therefore H contains all vertices connected by a path to A along with all of their incident edges and can hence be described as the **largest** connected subgraph that contains A. Even less formally, when looking at a disconnected graph such as Figure 2.10, each of the two "pieces" is a connected component.

Question 2.18 Given the following graph, which of the following are subgraphs? Which subgraphs are connected components?

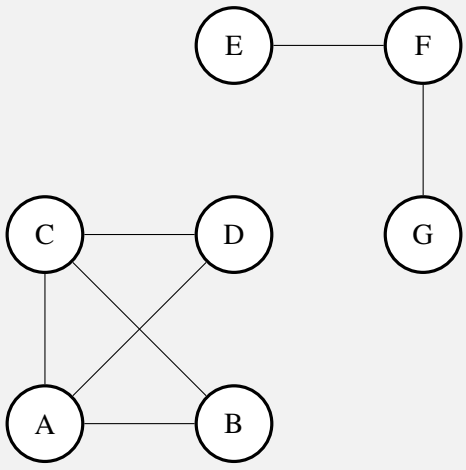


Figure 2.10

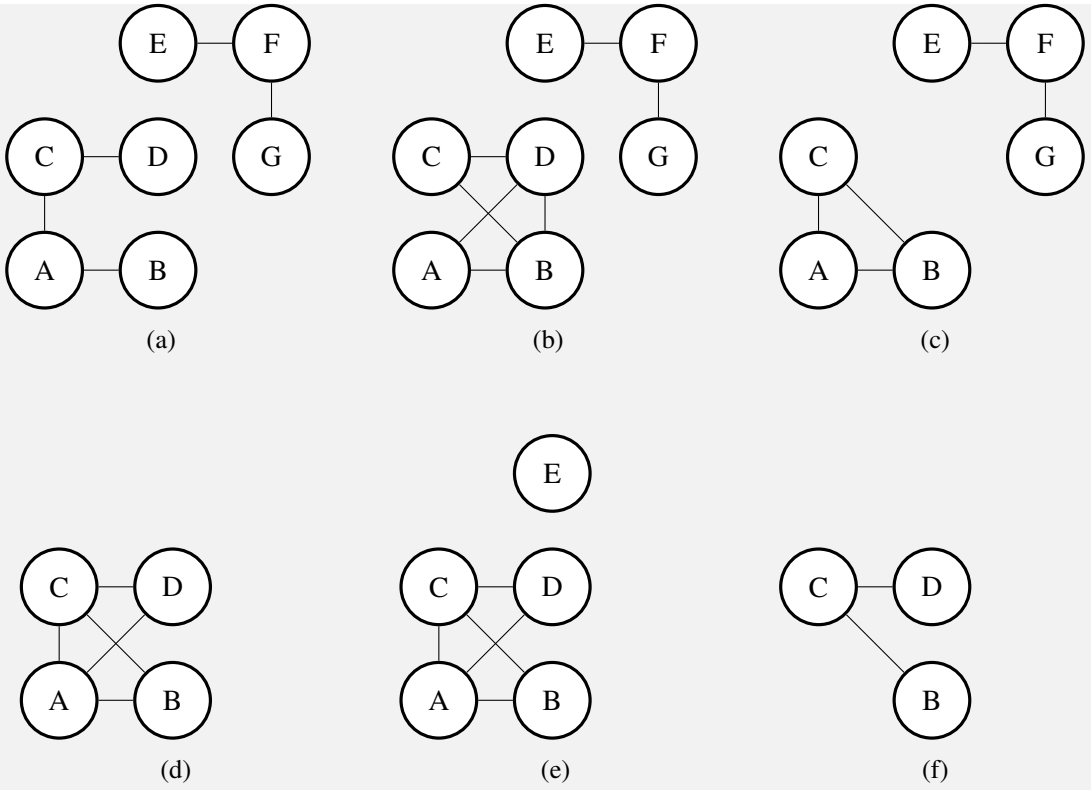


Figure 2.11

2.3 Adjacency Matrices

Given a graph, how many walks, or paths are there between two vertices? Counting the number of paths between two vertices is a hard problem that people still study today. Here we will stick to counting the number of walks between two vertices. To do this we will first define the adjacency matrix.

Definition 2.19 The **adjacency matrix** of a graph is a rectangular array of numbers where the number in row A, column B is the number of edges from vertex A to vertex B.

Example 2.20 Write down the adjacency matrix of the following graph.

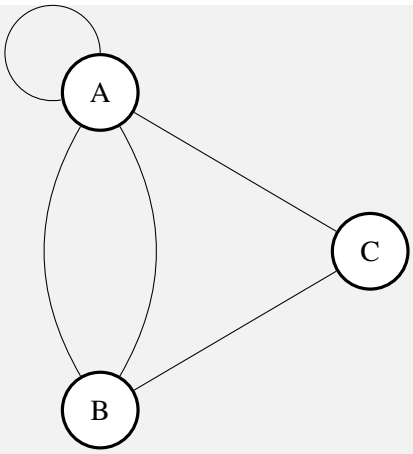


Figure 2.12

This graph has the following adjacency matrix.

$$\begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

Note 2.21 When writing adjacency matrices the indexing goes from top to bottom, and from left to right, like how we write.

Matrices are used in a wide variety of contexts. The matrix in Example 2.20 is a 3×3 matrix, because it has 3 rows and 3 columns (to be clear, rows are horizontal lines and columns are vertical lines). We will sometimes refer to "row A" as the row associated with vertex A, or "the 1st row", which is the first row from the top completely independent of the indexing. Not all matrices are square, or have non-negative or even integer entries. The matrix

$$\begin{pmatrix} 1 & 3 & 7 & 8.5 \\ 0.1 & -4 & -1.2 & 0 \end{pmatrix}$$

is a 2×4 matrix because it has 2 rows and 4 columns (the rows come first). However In Leaving Cert Applied Maths we will only deal with square matrices. We can do basic arithmetic with matrices, namely addition and multiplication. Consider the two matrices

$$M = \begin{pmatrix} 1 & 2 \\ -3 & 5 \end{pmatrix}, \quad N = \begin{pmatrix} 5 & -1 \\ -2 & 0 \end{pmatrix}.$$

Because they have the same dimensions, we can add them. To get the matrix $M+N$, we simply add the terms in the same position;

$$M+N = \begin{pmatrix} 1 & 2 \\ -3 & 5 \end{pmatrix} + \begin{pmatrix} 5 & -1 \\ -2 & 0 \end{pmatrix} = \begin{pmatrix} 1+5 & 2-1 \\ -3-2 & 5+0 \end{pmatrix} = \begin{pmatrix} 6 & 1 \\ -5 & 5 \end{pmatrix}.$$

We can also multiply these matrices, because they are square matrices of the same size (some rectangular matrices of different sizes can also be multiplied but we won't discuss this here).

$$MN = \begin{pmatrix} 1 & 2 \\ -3 & 5 \end{pmatrix} \begin{pmatrix} 5 & -1 \\ -2 & 0 \end{pmatrix} = \begin{pmatrix} 1(5)+2(-2) & 1(-1)+2(0) \\ -3(5)+5(-2) & -3(-1)+5(0) \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ -25 & 3 \end{pmatrix}.$$

What is going on here? To get the term in the 1st row, 1st column of MN, we go along the 1st row of M and the 1st column of N, multiplying terms as we go and adding the products together. To get the term in the 1st row, 2nd column of MN, we go along the 1st row of M and the 2nd column of N, and so on. This method can be extended to 3×3 matrices. If

$$P=\begin{pmatrix} 3 & 2 & -5 \\ 1 & -3 & 4 \\ -2 & 0 & 1 \end{pmatrix}, \quad Q=\begin{pmatrix} -4 & -2 & 3 \\ -1 & 0 & 2 \\ 3 & -2 & 1 \end{pmatrix},$$

then

$$\begin{aligned} PQ &= \begin{pmatrix} 3 & 2 & -5 \\ 1 & -3 & 4 \\ -2 & 0 & 1 \end{pmatrix}\begin{pmatrix} -4 & -2 & 3 \\ -1 & 0 & 2 \\ 3 & -2 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 3(-4)+2(-1)-5(3) & 3(-2)+2(0)-5(-2) & 3(3)+2(2)-5(1) \\ 1(-4)-3(-1)+4(3) & 1(-2)-3(0)+4(-2) & 1(3)-3(2)+4(1) \\ -2(-4)+0(-1)+1(3) & -2(-2)+0(0)+1(-2) & -2(3)+0(2)+1(1) \end{pmatrix} \\ &= \begin{pmatrix} -29 & 4 & 8 \\ 11 & -10 & 1 \\ 11 & 2 & -5 \end{pmatrix}. \end{aligned}$$

Question 2.22 Calculate NM and QP.

Note 2.23 As we can see, $MN \neq NM$ and $PQ \neq QP$, so matrix multiplication is not commutative.

We can also take powers of square matrices, as exponentiation is just repeated multiplication.

$$M^2 = MM = \begin{pmatrix} 1 & 2 \\ -3 & 5 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ -3 & 5 \end{pmatrix} = \begin{pmatrix} 1(1)+2(-3) & 1(2)+2(5) \\ -3(1)+5(-3) & -3(2)+5(5) \end{pmatrix} = \begin{pmatrix} -5 & 12 \\ -18 & 19 \end{pmatrix}.$$

We can take higher powers of matrices using lower powers. Matrix indices obey the same rules as indices for numbers. In particular

$$M^{m+n} = M^m M^n$$

for any $m,n \in \mathbb{N}$. Although matrix multiplication is not commutative ($MN \neq NM$ usually), it is *associative*, i.e.

$$\begin{aligned} M^3 &= (MM)M = M^2M, \\ &= M(MM) = MM^2 \end{aligned}$$

so that we can calculate M^3 by calculating M^2M or MM^2. Similarly,

$$\begin{aligned} M^4 &= M^3M \\ &= M^2M^2 \\ &= MM^3. \end{aligned}$$

Question 2.24 If

$$R=\begin{pmatrix} 3 & 1 \\ 2 & 2 \end{pmatrix}$$

calculate R^2 and R^3.

So why do we care about powers of matrices? Back to graphs, we have the following result.

Theorem 2.25 Given a graph with adjacency matrix M, the term in row A, column B of M^n is the number of walks of length n from vertex A to vertex B.

How do we use this theorem? Consider the adjacency matrix we had before:

$$\begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}.$$

Call this matrix M, and drop the indexing on the side when do our arithmetic.

$$M^2 = MM = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 6 & 3 & 3 \\ 3 & 5 & 2 \\ 3 & 2 & 2 \end{pmatrix}.$$

$$M^3 = M^2M = \begin{pmatrix} 6 & 3 & 3 \\ 3 & 5 & 2 \\ 3 & 2 & 2 \end{pmatrix} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 15 & 15 & 9 \\ 15 & 8 & 8 \\ 9 & 8 & 5 \end{pmatrix}.$$

$$M^4 = M^2M^2 = \begin{pmatrix} 6 & 3 & 3 \\ 3 & 5 & 2 \\ 3 & 2 & 2 \end{pmatrix} \begin{pmatrix} 6 & 3 & 3 \\ 3 & 5 & 2 \\ 3 & 2 & 2 \end{pmatrix} = \begin{pmatrix} 54 & 39 & 30 \\ 39 & 38 & 23 \\ 30 & 23 & 17 \end{pmatrix}.$$

So considering walks from A to C, there are 3 walks of length 2, 9 walks of length 3 and 30 walks of length 4. Considering walks from B back to itself, there are 5 walks of length 2, 8 walks of length 3 and 38 of length 4.

Question 2.26 A certain graph has the following adjacency matrix (note that this is the same matrix as that in Question 2.24).

$$\begin{array}{c} \\ A \\ B \end{array} \begin{array}{c} \begin{array}{cc} A & B \end{array} \\ \begin{pmatrix} 3 & 1 \\ 2 & 2 \end{pmatrix} \end{array}$$

(a) How many walks of length 2 are there between A and B?

(b) How many walks of length 3 are there between A and B?

(c) How many walks of length 3 are there from A to itself?

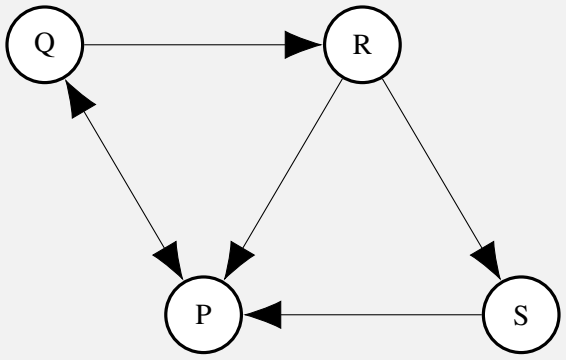
Note 2.27 It is common for students to forget whether Theorem 2.25 applies to paths or walks. It may help to remember that **M**atrices tell us about **W**alks because W is an upside-down M.

Note 2.28 Calculating matrices is cumbersome, and it gets more time-intensive the larger the matrix is and the higher the power is. You can use matrix calculators online (e.g. matrixcalc.org/en) to calculate products and powers of larger matrices and matrices to a high power. However it is

important to be able to work with smaller matrices by hand.

2.4 Adjacency Matrices for Digraphs

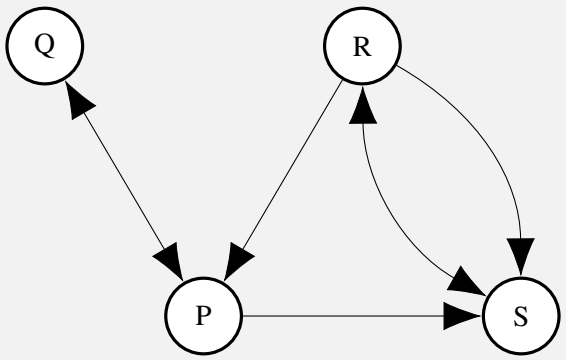
Example 2.29 Write down the adjacency matrix of the following graph.



This graph has adjacency matrix

$$\begin{array}{c} \\ P \\ Q \\ R \\ S \end{array} \begin{array}{c} \begin{array}{cccc} P & Q & R & S \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Question 2.30 Write down the adjacency matrix of the following graph.



Note 2.31 Unlike with the previous adjacency matrix, in the matrix in Example 2.29 the term in row P, column R is 0 but the term in row R, column P is 1. This is because there is an edge from R to P, but not from P to R as our graph is directed. However Theorem 2.25 still holds. If we

call this matrix M, we can calculate that

$$M^2 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix},$$

$$M^3 = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 2 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}.$$

So when it comes to walks of length 2, there is a walk from Q to P but not from P to Q. There are 2 walks of length 3 from Q to P.

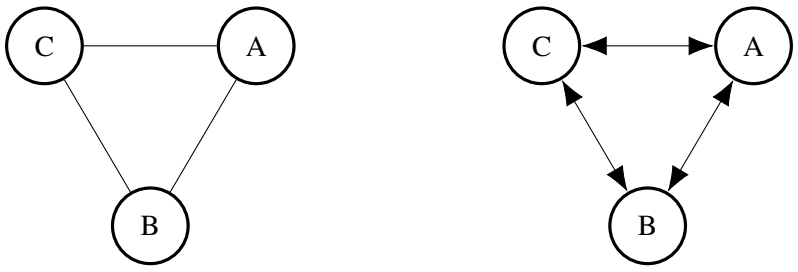
To formalise the contrast between adjacency matrices for directed and undirected graphs we have the following definition and result.

Definition 2.32 A matrix is **symmetric** if it is square and the i-th row is the same as the i-th column for all i.

The reason we use the word symmetric is that you can see the symmetry of the matrix if you view the line from the top-left to bottom-right as the axis of symmetry.

Theorem 2.33 A graph is undirected if and only if its adjacency matrix is symmetric.

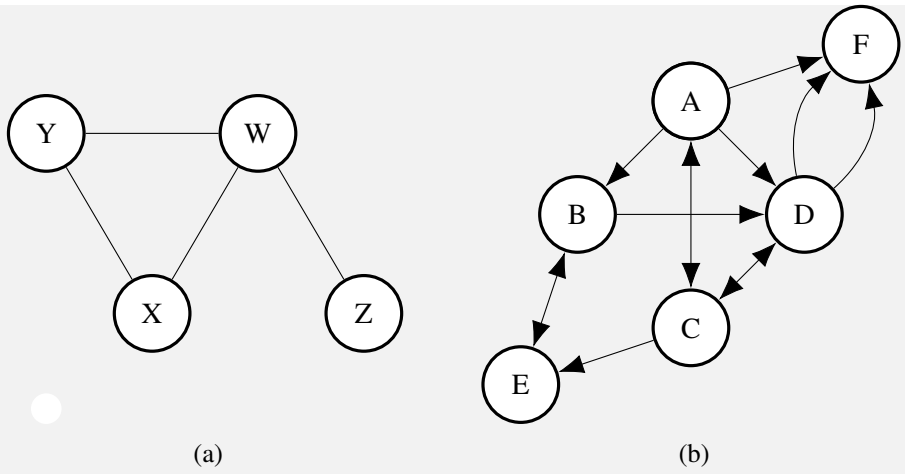
This may be confusing, as it may have seemed that a graph is directed whenever its edges have arrows representing direction. However if a digraph has a symmetric adjacency matrix then every edge has arrows in both directions. In this case it is fundamentally indistinguishable from an undirected graph, and so we can think of it as undirected. For example, both of the graphs below can be seen as identical and undirected.



(a) (b)

Figure 2.13

Question 2.34 Write down the adjacency matrix of the following graphs, and draw the graphs of the given adjacency matrices.



(a) (b)

Figure 2.14

$$\begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ P \\ Q \\ R \\ S \\ T \end{array}\begin{array}{c} \begin{array}{ccccc} P & Q & R & S & T \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 2 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 & 0 \end{pmatrix} \end{array}$$

2.5 Trees

One example of a graph we've avoided so far is a family tree.

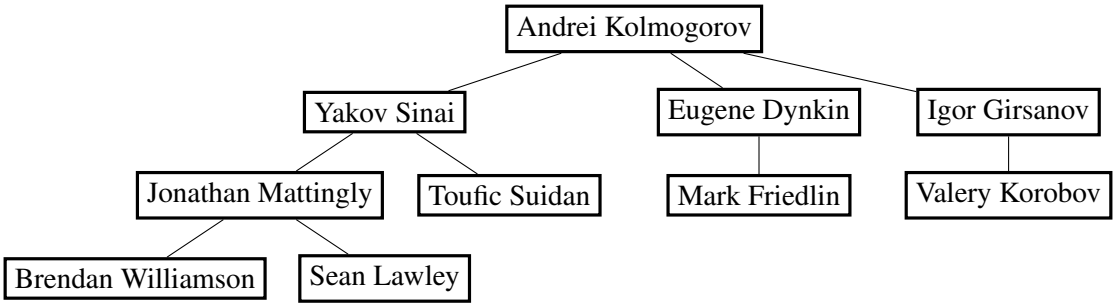


Figure 2.15: A partial family tree of mathematicians and their PhD advisors.

This is how we usually see family trees, but by drawing an edge from "parent" to "child" we can get something more like the graphs we're used to.

There are a large class of graphs that can be arranged to look like family trees, i.e. they are isomorphic to a tree graph. Although this is where they got their name, it's actually not why we care about them.

Definition 2.35 A **tree** is a connected graph that contains no cycles.

This definition, although it seems disconnected from our original view of a family tree, makes sense when you think of an actual tree. In a tree the trunk breaks into branches, then more branches, and

so on, but branches never join up again, which would be necessary to make a cycle.

Theorem 2.36

1. A tree with n vertices has $n-1$ edges.
2. For any two vertices in a tree, there is exactly one path between them.

So why do we care about trees? Many structures other than families can be represented as tree graphs, such as corporate hierarchies, file systems and decision trees, but that's not the real reason. Consider the following problem. A weighted graph, shown below, represents six locations as vertices. The edges are proposed routes for power lines, and the weight is the cost (in millions of Euros).

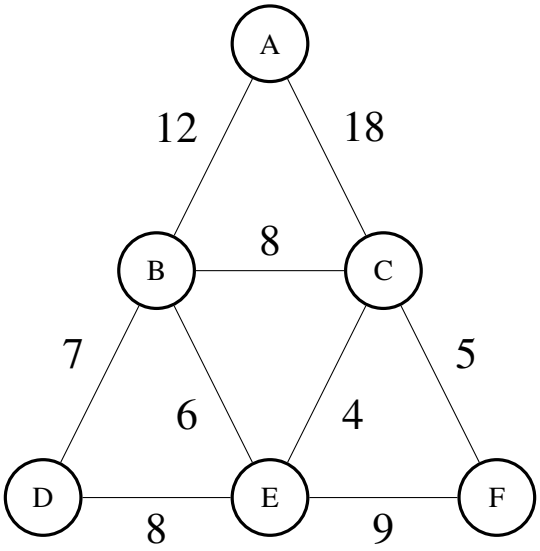


Figure 2.16

Because power moves quickly and we're not worried about overloading a single line, we want to connect every vertex in the cheapest way. The resulting graph that we make will be a tree, because if it has a cycle we can remove an edge and every vertex will still be connected. There are also issues if cycles are included in fast moving networks such as power, phone or internet lines or networks. This motivates the following definitions.

Definition 2.37

1. A **spanning tree** of a graph G is a subgraph of G which is a tree connecting all vertices of G.
2. A **minimum spanning tree** of a weighted graph G is a spanning tree with minimum total weight across all edges.

Note that for a graph G there may be more than one minimum spanning tree, i.e. two or more spanning trees may have equal total weight which is also the minimum possible weight for a spanning tree.

So how do we find a minimum spanning tree of the graph in Figure 2.16? We'll answer this question, and more, in Chapter 3.

2.6 Exam Questions

There are few A level exam questions on material in this chapter, which existed mostly to prepare students for later chapters. Those exam questions that exist often ask students about definitions and applications of the Handshaking Lemma.

Example 2.38 — Old D1 January 2002 Q3(ii).

(ii) A connected network N has seven vertices.

(a) State the number of edges in a minimum spanning tree for N.

(1)

A minimum spanning tree for a connected network has n edges.

(b) State the number of vertices in the network.

(1)

Figure 2.17

The answer to (a) is 6 and relies on the student knowing that the minimum spanning tree also has 7 vertices, and hence as a tree has $7-1=6$ edges. Similarly, the answer to (b) is $n+1$.

Example 2.39 — Old D1 January 2007 Q5(a).

(a) Explain why a network cannot have an odd number of vertices of odd degree.

(2)

Figure 2.18

The Handshaking Lemma states that the sum of the degrees of a network is twice the number of edges. In particular this means that the sum of the degrees is even, and so an even number of degrees must be odd numbers.

Example 2.40 — Old D1 January 2010 Q2(a).

Prim's algorithm finds a minimum spanning tree for a connected graph.

(a) Explain the terms

(i) connected graph,

(ii) tree,

(iii) spanning tree.

Figure 2.19

A connected graph is a graph where there is a path between any two vertices. A tree is a connected graph containing no cycles. A spanning tree is a subgraph of a graph that is a tree and contains all vertices in the original graph.

Example 2.41 — Old D1 January 2015 Q5(d).

A connected graph V has n nodes. The sum of the degrees of all the nodes in V is m. The graph T is a minimum spanning tree of V.

(d) (i) Write down, in terms of m, the number of arcs in V.

(ii) Write down, in terms of n, the number of arcs in T.

(iii) Hence, write down an inequality, in terms of m and n, comparing the number of arcs in T and V.

Figure 2.20

The answer to (i) is $\frac{m}{2}$ from the Handshaking Lemma. The answer to (ii) is $n-1$ as T is a tree containing all n nodes from V. The answer to (iii) is

$$n-1 \leq \frac{m}{2}$$

as T is a subgraph of V.

One major difference between the Edexcel syllabus and the Leaving Cert syllabus is that the Edexcel syllabus does not include adjacency matrices, or matrices in general. When revising this chapter, students should separately make sure they understand adjacency matrices, can calculate powers of adjacency matrices and understand their significance.

Example 2.42 — Sample Paper Q1(a).

Question 1

(a) A directed graph is represented by the adjacency matrix $M = \begin{pmatrix} 0 & 0 & 2 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$.

(i) Draw the graph represented by M.

(ii) Calculate M^2.

(iii) What information is provided by the elements of M^2?

Figure 2.21

(i)

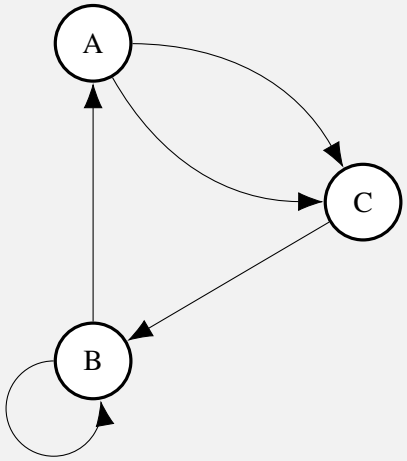


Figure 2.22

(ii)

$$M^2 = \begin{pmatrix} 0 & 0 & 2 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 2 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$
$$= \begin{pmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 1 & 1 & 0 \end{pmatrix}$$

(iii) The elements in M^2 are the number of walks of length 2 from one vertex to another. For example, from row 1 column 2 there are 2 walks of length 2 from A to B.

2.7 Summary

- Know the definition of a directed/undirected, weighted/unweighted graph, loop, and simple graph/multigraph.
- Be able to spot if two graphs are isomorphic.
- Know the definition of incidence, adjacence, degree and understand the Handshaking Lemma.

- Know the definition of a walk/path/cycle, subgraphs, connected components and trees/spanning trees/minimum spanning trees.
- Know how to write an adjacency matrix given a graph, and be able to draw a graph given an adjacency matrix, including whether or not the adjacency matrix is representing a directed or undirected graph.
- Know how to multiply small square matrices by hand, and larger square matrices using a computer.

2.8 Notes on the Exam, and Work Still to Cover

This chapter serves mostly as a foundation for the rest of the chapters in Part I, and so few exam questions will be based on this topic. The exam questions most likely to appear are those which ask students to find adjacency matrices given graphs and vice versa, calculate products or powers of matrices and ask students to use powers of matrices to give the number of walks between vertices of a certain length. There may also be some exam questions which ask for definitions, or ask students if certain walks are paths, certain graphs are trees, certain subgraphs are connected components, etc.However this is speculative after only having two past papers (including the sample paper).

2.9 Homework

Graph Types, Properties and Definitions

1. Which of the following graphs are simple, and which are multigraphs? Explain your reasoning.

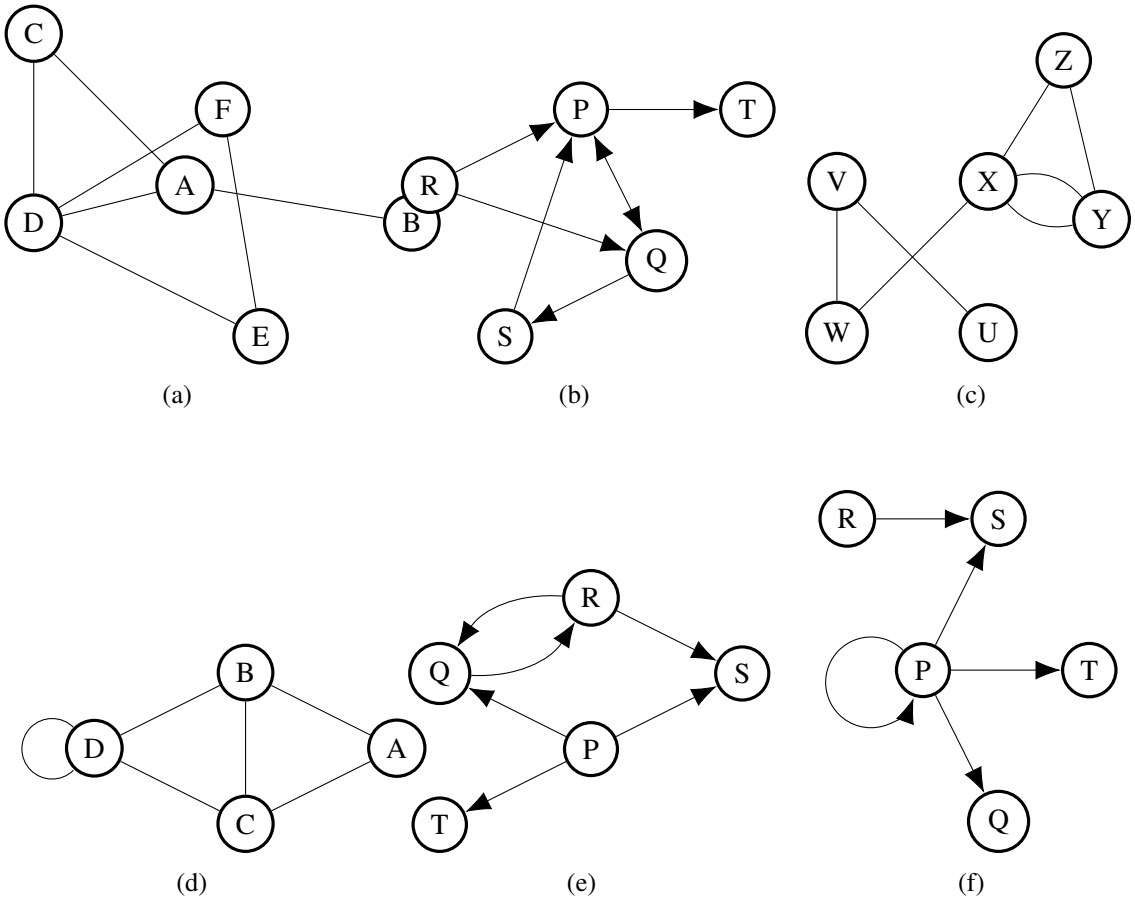


Figure 2.23

2. In the following graphs, write down the vertices adjacent to each vertex, the edges incident to each vertex, and the degree of each vertex. Then count the number of edges, and hence verify the Hand-shaking Lemma.

   (a)

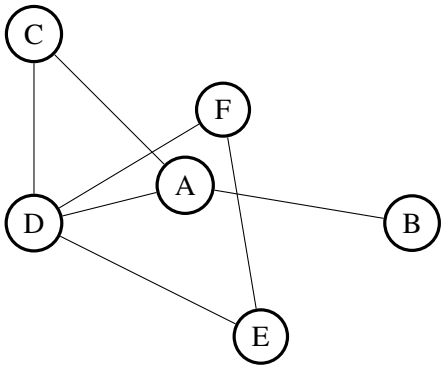


Figure 2.25

Vertex	Adjacent Vertices	Incident Edges
A	B, C, D	AB, AC, AD
B		
C		
D		
E		
F		

Table 2.2

(b)

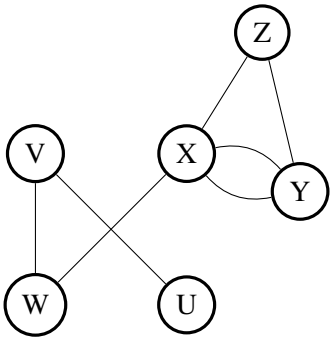


Figure 2.27

Vertex	Adjacent Vertices	Incident Edges
U		
V		
W		
X		
Y		
Z		

Table 2.3

(c)

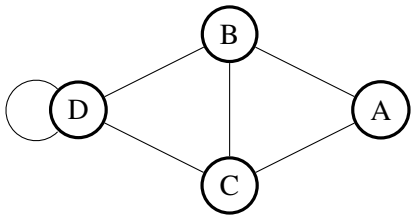


Figure 2.29

Vertex	Adjacent Vertices	Incident Edges
A		
B		
C		
D		

Table 2.4

3. Which of the following pairs of graphs are isomorphic? Explain your answer.

(a)

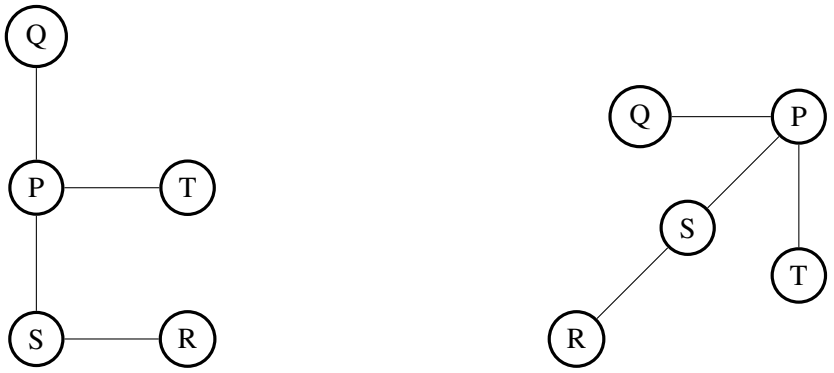


Figure 2.30

(b)

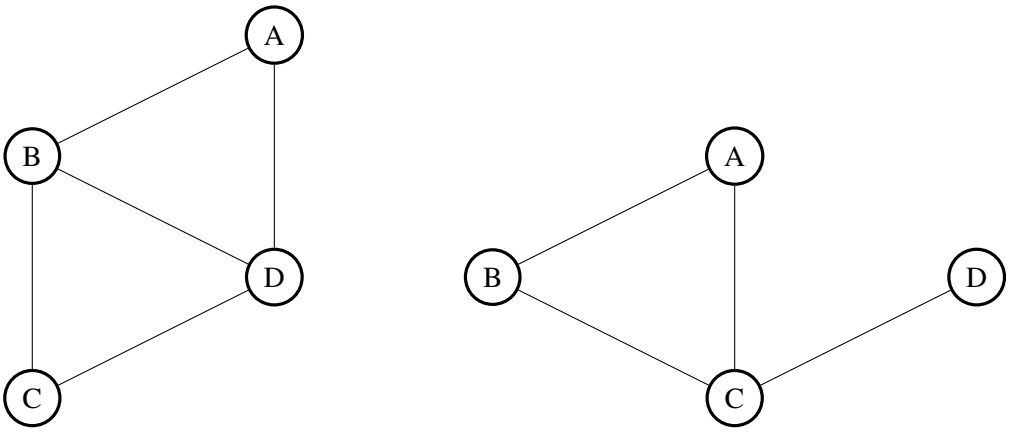


Figure 2.31

(c)

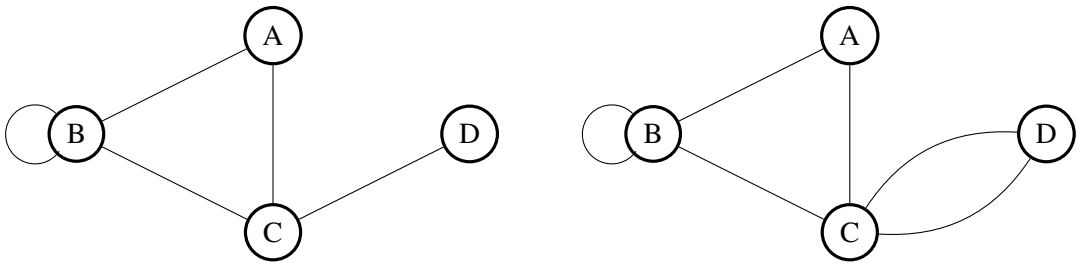
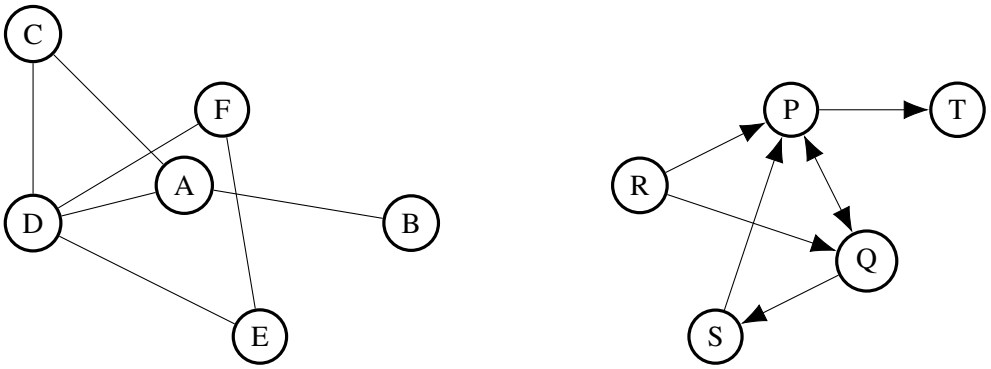


Figure 2.32

4. In each of the following graphs, write down a walk that is not a path, a path, and a cycle.



(a) (b)

Figure 2.33

5. Given the following graph, draw

   (a) a connected component,

   (b) a connected subgraph that is not a connected component,

   (c) a disconnected subgraph.

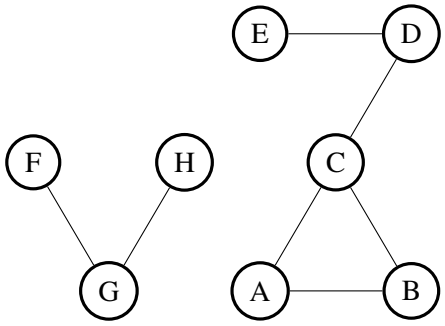


Figure 2.34

Adjacency Matrices

6. Given the following graphs, write down their adjacency matrices.

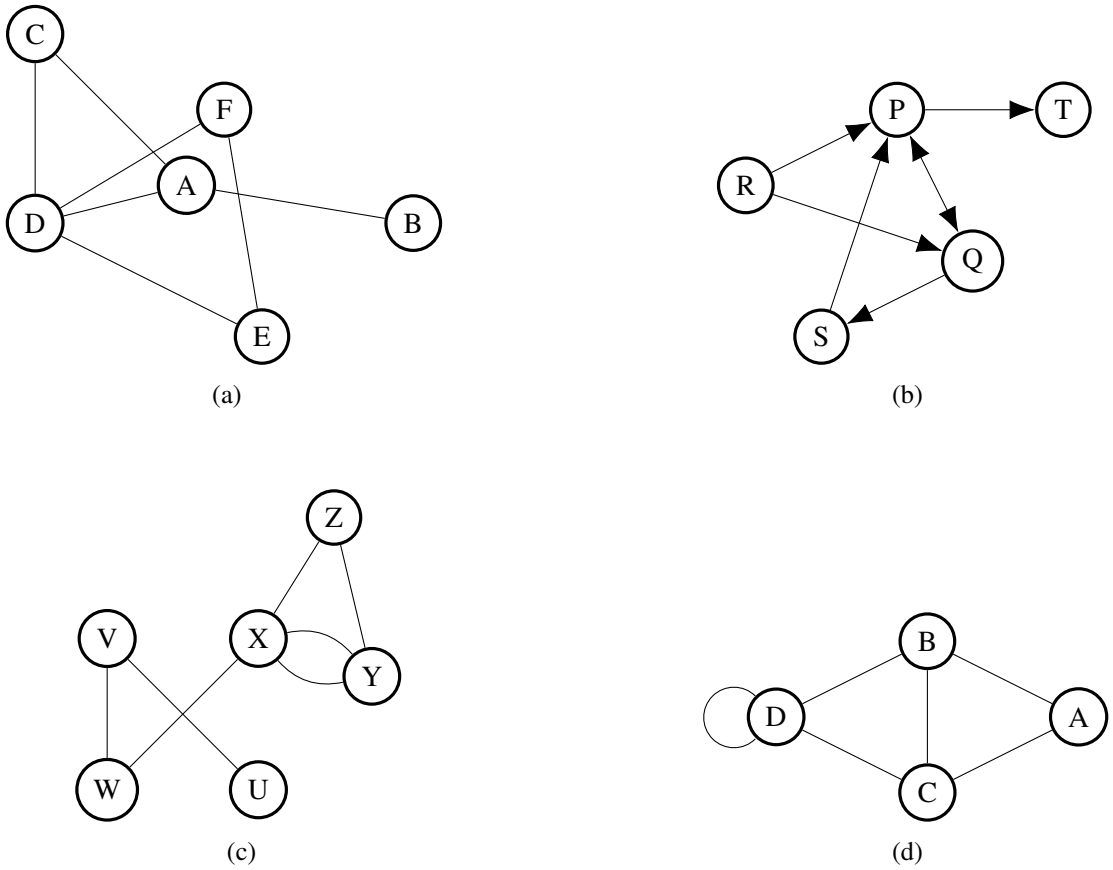


Figure 2.35

7. Given the following adjacency matrices, draw the corresponding graph.

$$\begin{array}{c} \\ A \\ B \\ C \end{array}\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ P \\ Q \\ R \\ S \\ T \end{array}\begin{array}{c} \begin{array}{ccccc} P & Q & R & S & T \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 2 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 2 & 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 & 1 \end{pmatrix} \end{array}$$

8. Given the following matrices,

   (a) calculate M^2 by hand (show your work),

   (b) calculate M^3 by hand (show your work),

   (c) calculate N^5 using a computer (no need to show work).

   (d) How many walks of length 3 are there from A to C?

   (e) How many walks of length 5 are there from P to T?

$$M = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array} \qquad N = \begin{array}{c} \\ P \\ Q \\ R \\ S \\ T \end{array} \begin{array}{c} \begin{array}{ccccc} P & Q & R & S & T \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 2 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 2 & 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 & 1 \end{pmatrix} \end{array}$$

Trees

9. Given the following graph, explain why it is not a tree, and find any spanning tree of the graph.

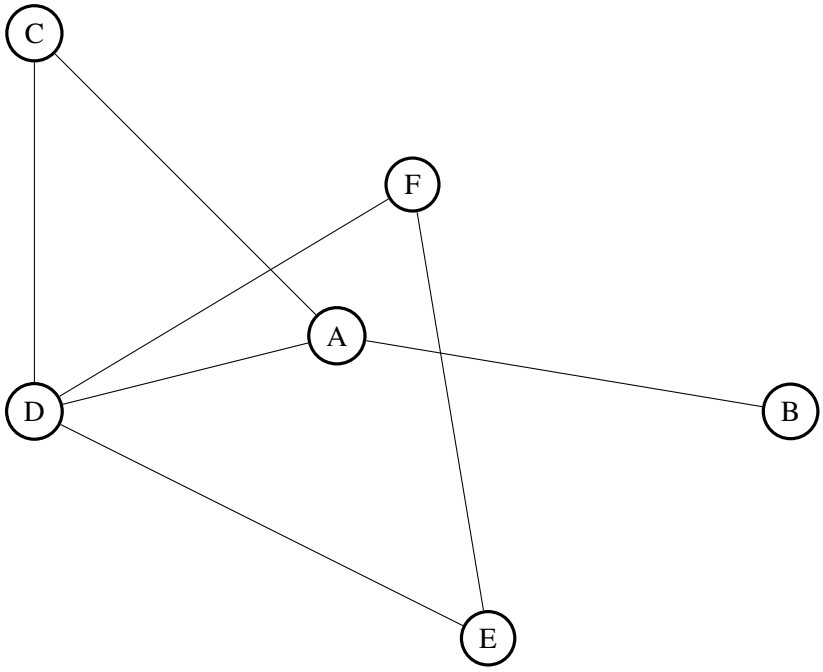


Figure 2.36

2.10 Homework Solutions

Graph Types, Properties and Definitions

1. (a) Simple, no loops or multiple edges.

   (b) Simple, no loops or multiple edges.

   (c) Multigraph, multiple edge XY.

   (d) Multigraph, loop at D.

   (e) Simple, no loops or multiple edges.

   (f) Multigraph, loop at P.

2. (a)

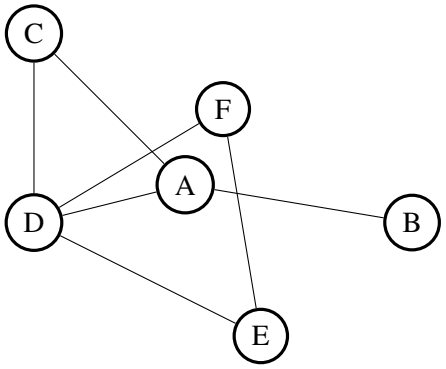


Figure 2.37

Vertex	Adjacent Vertices	Incident Edges
A	B, C, D	AB, AC, AD
B	A	AB
C	A, D	AC, CD
D	A, C, E, F	AD, CD, DE, DF
E	D, F	DE, EF
F	D, E	DF, EF

Table 2.5

Note: ED instead of DE, etc., are also acceptable.

$$\begin{aligned}\text{Sum of Degrees} &= 3+1+2+4+2+2\\ &= 14.\\ \text{Number of Edges} &= 7 \quad \checkmark\end{aligned}$$

(b)

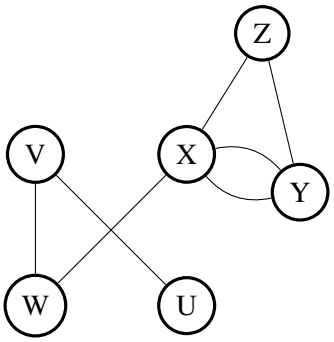


Figure 2.38

Vertex	Adjacent Vertices	Incident Edges
U	V	UV
V	U, W	UV, VW
W	V, X	VW, WX
X	W, Y, Y, Z	WX, XY, XY, XZ
Y	X, X, Z	XY, XY, YZ
Z	X, Y	XZ, YZ

Table 2.6

Note: It is recommended to list adjacent vertices/incident edges multiple times as it

makes counting degree easier, and indicates that there are repeated edges.

$$\begin{aligned}\text{Sum of Degrees} &= 1+2+2+4+3+2\\ &= 14.\\ \text{Number of Edges} &= 7 \quad \checkmark\end{aligned}$$

(c)

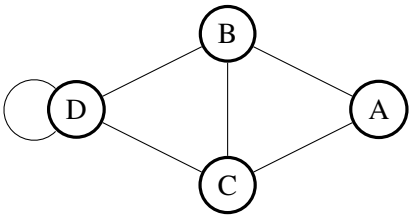


Figure 2.39

Vertex	Adjacent Vertices	Incident Edges
A	B, C	AB, AC
B	A, C, D	AB, BC, BD
C	A, B, D	AC, BC, CD
D	B, C, D	BD, CD, DD, DD

Table 2.7

Note: It is necessary to list loops as edges, and self-adjacent vertices as so. It is actually recommended to list loops twice so that they can be easily counted twice when counting degree.

$$\begin{aligned}\text{Sum of Degrees} &= 2+3+3+4\\ &= 12.\\ \text{Number of Edges} &= 6 \quad \checkmark\end{aligned}$$

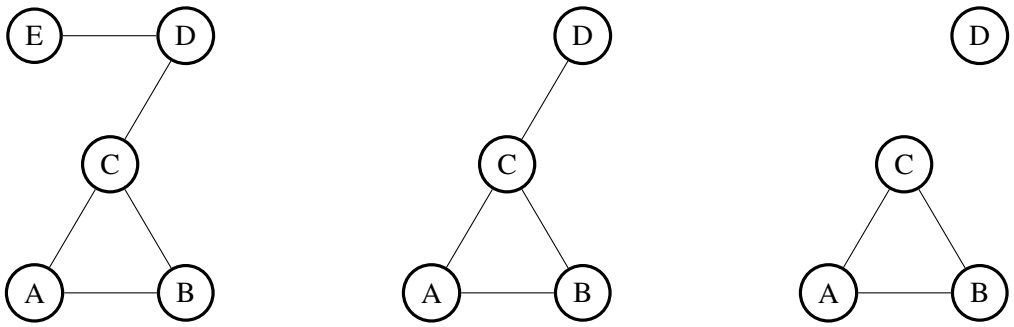
3. (a) Isomorphic, both graphs have the same edges.

   (b) Not isomorphic, different number of edges.

   (c) Not isomorphic, different number of edges.

4. *ACDAB* is a walk, *CDAB* is a path and *ACDA* is a cycle. There are many other answers to this question.

5.



(a) (b) (c)

There are multiple answers to this question.

Adjacency Matrices

6. (a)

	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	0	0	0	0
C	1	0	0	1	0	0
D	1	0	1	0	1	1
E	0	0	0	1	0	1
F	0	0	0	1	1	0

(b)

	P	Q	R	S	T
P	0	1	0	0	1
Q	1	0	0	1	0
R	1	1	0	0	0
S	1	0	0	0	0
T	0	0	0	0	0

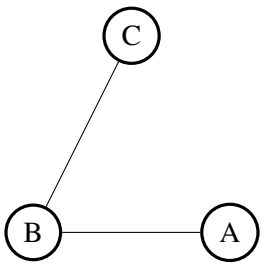
(c)

	U	V	W	X	Y	Z
U	0	1	0	0	0	0
V	1	0	1	0	0	0
W	0	1	0	1	0	0
X	0	0	1	0	2	1
Y	0	0	0	2	0	1
Z	0	0	0	1	1	0

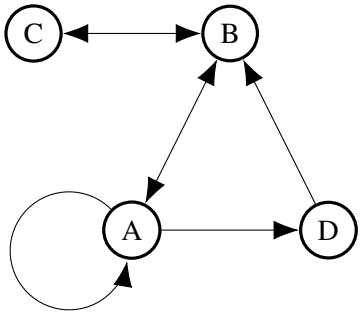
(d)

	A	B	C	D
A	0	1	1	0
B	1	0	1	1
C	1	1	0	1
D	0	1	1	1

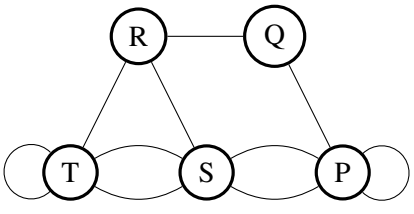
7.



(a)



(b)



(c)

8. (a)

$$M^2 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

(b)

$$M^3 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

(c)

$$N^5 = \begin{pmatrix} 186 & 109 & 133 & 303 & 209 \\ 109 & 30 & 74 & 90 & 109 \\ 133 & 74 & 96 & 207 & 154 \\ 303 & 90 & 207 & 265 & 315 \\ 209 & 109 & 154 & 315 & 240 \end{pmatrix}$$

(d) 1

(e) 209

Trees

9. There is a cycle $ACDA$. One can get a spanning tree by removing any one of AC, AD, CD, and any one of DE, EF, DF.

2.11 Revision

When revising this chapter, students should make sure they know all of their definitions, as it is possible that exam questions may ask for them in some form. Beyond that, the only problems that students can practice are those where they multiply matrices. By using matrix calculators such as matrixcalc.org/en, students can make an unlimited amount of their own matrices to multiply or take powers of and check their answers afterwards.

As this is a new topic pas exam questions on this chapter are only from the sample paper (Q1 (a)) and the 2023 paper (Q1 (a)).

3. Algorithms on Graphs

3.1 What is an Algorithm?

An algorithm is a sequence of well defined instructions designed to solve a specific problem. We will use algorithms to solve many problems on graphs. In particular, both Kruskal's Algorithm and Prim's Algorithm give us a minimum spanning tree of a simple, connected, weighted, undirected graph.

3.2 Kruskal's Algorithm

Algorithm 3.1 — Kruskal's Algorithm. To find a minimum spanning tree of a simple, connected, weighted, undirected graph G with n vertices, Kruskal's algorithm consists of the following steps.

1. List the edges in ascending order of weight. If two edges have the same weight, order doesn't matter.
2. Beginning with the subgraph H which contains all vertices and no edges, add the edge of least weight to H.
3. If the next edge on the list does not create a cycle, add it to H. If it does, cross it out.
4. Repeat step 3 until you have $n-1$ edges. H is now a minimum spanning tree of G.

Example 3.2 Using Kruskal's Algorithm find a minimum spanning tree for the following graph.

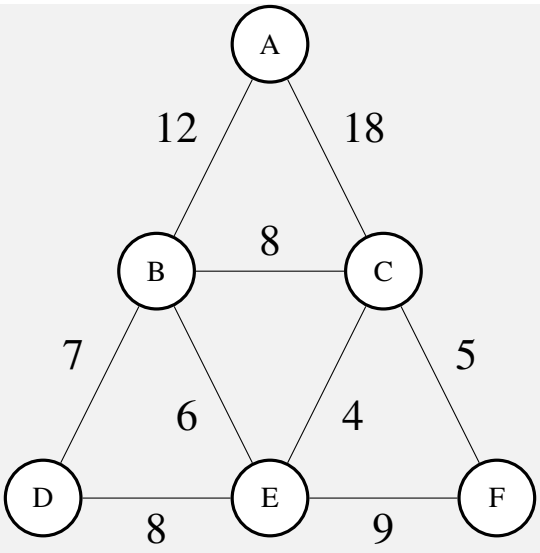


Figure 3.1

Step 1

First we list all of the edges in ascending order of weight.

Edge	Weight
CE	4
CF	5
BE	6
BD	7
DE	8
BC	8
EF	9
AB	12
AC	18

It's important to check that we haven't missed any edges. Using the Handshaking Lemma (Theorem 2.11), our vertices have sum of degrees $2+4+4+2+4+2=18$ so we should have $\frac{18}{2}=9$ edges, which we do. This is our main use for the Handshaking Lemma in this course. Our edges are also in the right order.

Steps 2, 3 & 4

We include CE as the first edge. We can add CF, then BE, and BD. We can't then add DE as it would create a cycle. We also can't add BC or EF, but we can add AB. Now that we have 5 edges we know we are done.

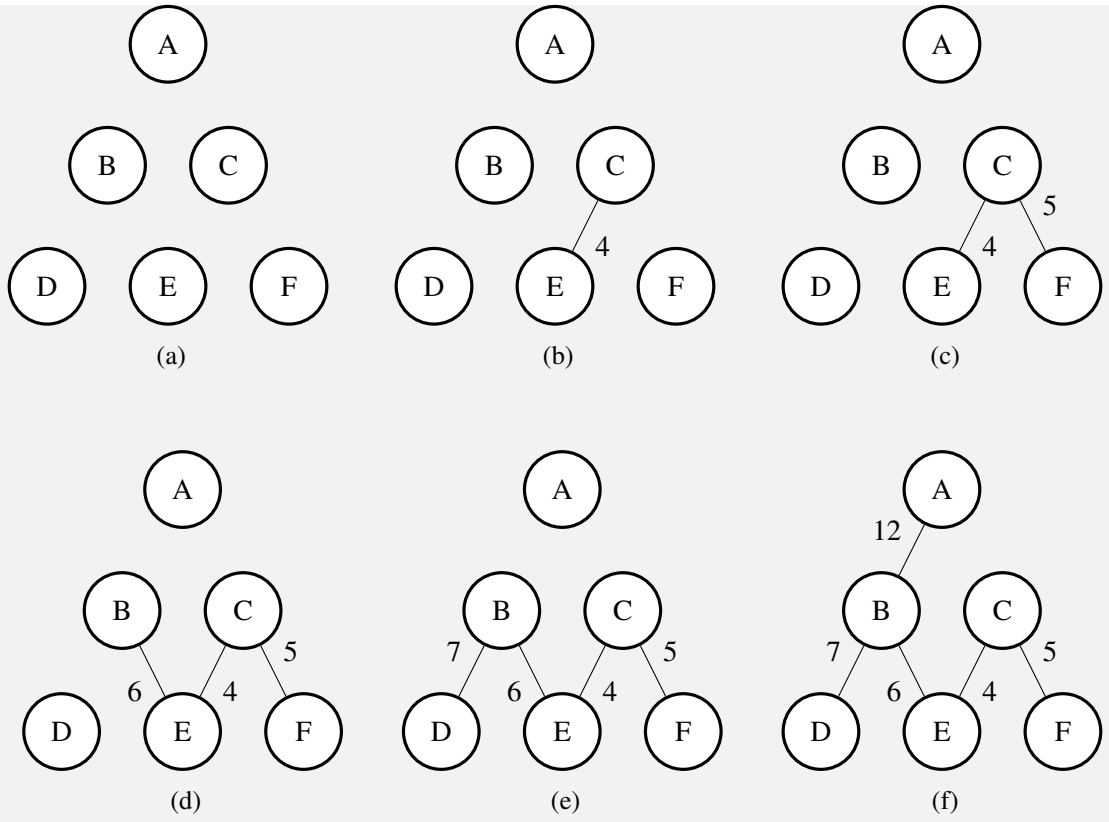


Figure 3.2

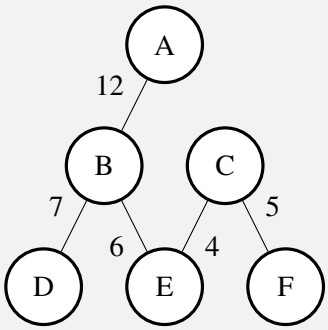


Figure 3.3: Minimum spanning tree

Edge	Weight	
CE	4	✓
CF	5	✓
BE	6	✓
BD	7	✓
DE	8	✗
BC	8	✗
EF	9	✗
AB	12	✓
AC	18	

Table 3.1: Included and excluded edges

Note 3.3 Notice that we could have swapped the order of the edges *DE* and *BC* in the list. However it wouldn't have changed our decision making; neither edge would be included anyway.

Rule 3.4 In order for more than one minimum spanning tree to exist there have to be at least two edges of equal weight, and the choice to include one must affect whether or not another is included. In other words, when two or more edges have equal weight, deciding what order they appear in the list must affect the final tree constructed. If this is not the case the minimum spanning tree found is unique.

Question 3.5 Use Kruskal's Algorithm to find a minimum spanning tree of the graph below.

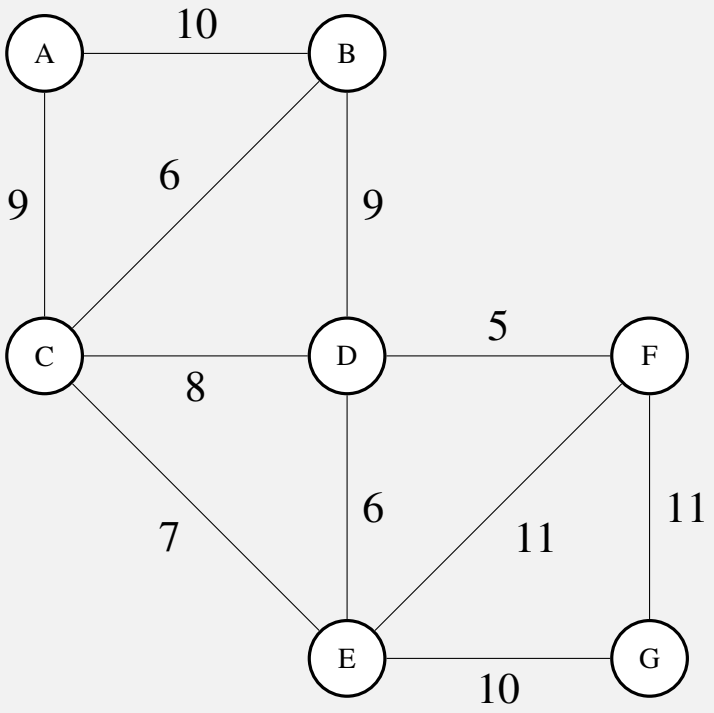


Figure 3.4

Question 3.6 Use Kruskal's Algorithm to find a minimum spanning tree of the graph below.

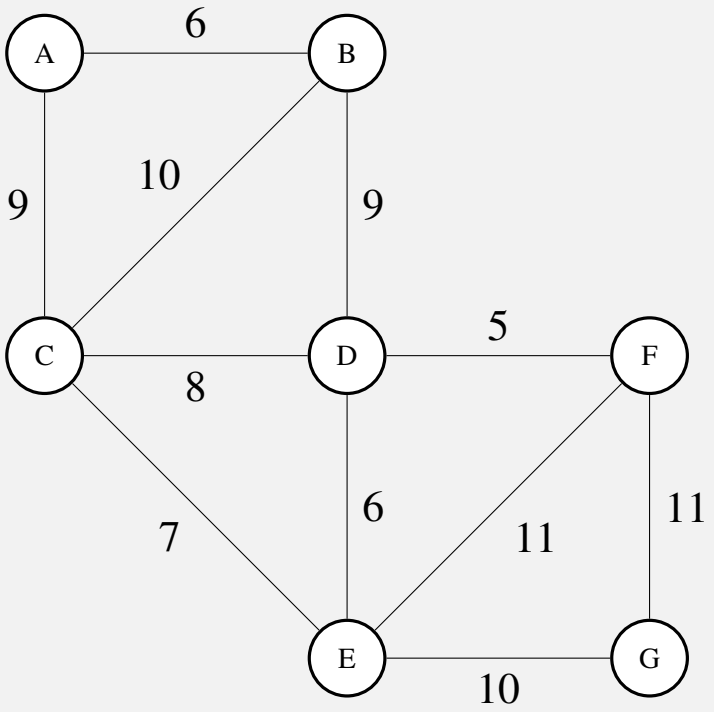


Figure 3.5

Note 3.7 Notice that in Question 3.5 edge AC is included but BD is not, regardless of the order in which the edges are placed. However in Question 3.6 whichever edge is higher up the list is included and then the other is not, making Figure 3.5 the first graph we have seen with more than one minimum spanning tree.

3.3 Prim's Algorithm

There is a second algorithm for finding the minimum spanning tree of a graph. It is known as **Prim's Algorithm**, although it was originally published by Jarník in 1930 before being republished by Prim (1957) and Dijkstra (1959). It will still find a minimum spanning tree, it just differs in its execution. We will discuss the differences between the two methods later.

Algorithm 3.8 — Prim's Algorithm. To find a minimum spanning tree of a simple, connected, weighted, undirected graph G with n vertices, Prim's algorithm consists of the following steps.

1. Choose a vertex at random. This creates the tree T.
2. Choose the edge of least weight that is incident to this vert Add this edge and the corresponding adjacent vertex to T.
3. Choose the edge of least weight that is incident to only one vertex in T. Add this edge and the corresponding adjacent vertex to T.
4. Repeat step 3 until T has $n-1$ edges, or equivalently contains all vertices in G. T is now a minimum spanning tree of G.

Note: If at any point there is a choice between two or more edges of equal weight, choose one at random.

Note 3.9 T is a tree at all stages in Algorithm 3.8, eventually becoming a minimum spanning tree. This is why in step 3 we choose an edge which is incident to only one vertex in T. If it was incident to both it would create a cycle.

Example 3.10 Using Prim's Algorithm, find a minimum spanning tree for the following graph.

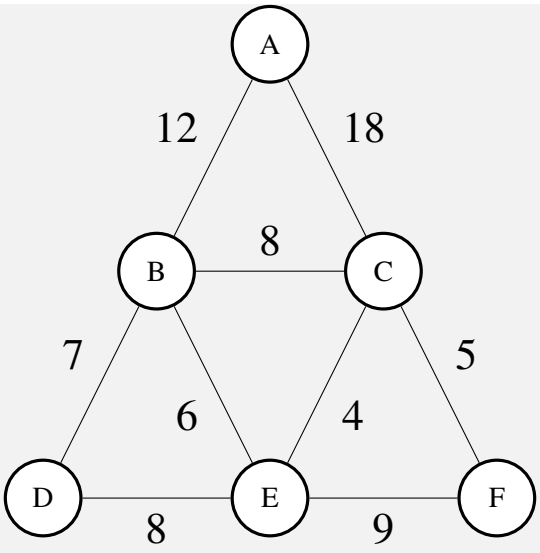


Figure 3.6

Let's start with vertex E. The following sequence of graphs shows the evolution of our tree T until we have a minimum spanning tree.

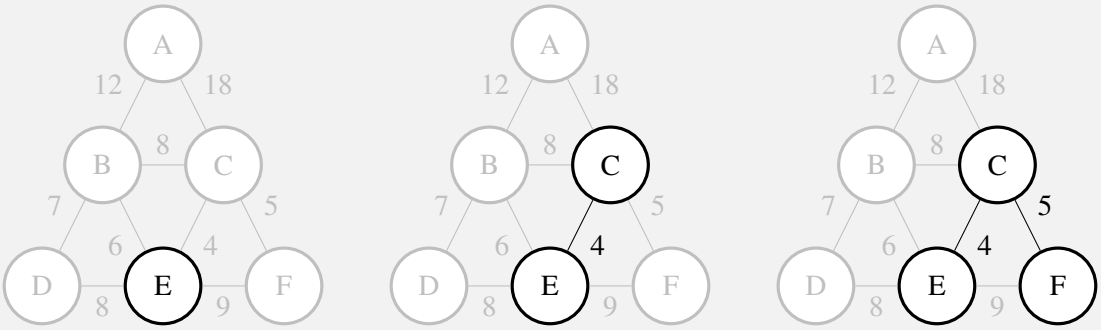


Figure 3.7

To better understand the steps, we forst chose vertex E at random. We then had four edges to choose from, DE, BE, CE and FE. Edge CE has the lowest weight, so we added CE and vertex

C to the tree T. We then had six edges to choose from, AC, BC, BE, DE, FE and CF. Edge CF had the lowest weight so we added CF and F. We then added edge BE (and B), then BD (and D), and finally AB (and A). As we then had 5 edges, or equivalently T contains all vertices, we were done.

Rule 3.11 If at any point one chooses at random between edges of equal weight, there may be more than one minimum spanning tree. To see if the spanning tree is unique, write down the equal weight edges and choose one. If the other edge(s) are not in the final minimum spanning tree the minimum spanning tree is not unique. If this never happens in the construction of the minimum spanning tree then it is unique.

Question 3.12 Use Prim's Algorithm to find a minimum spanning tree of the graph below.

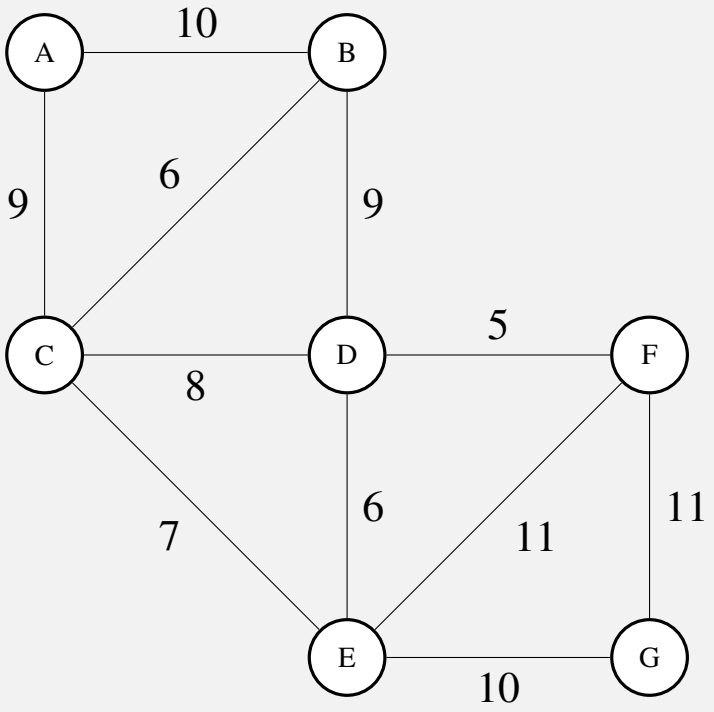


Figure 3.8

Question 3.13 Use Prim's Algorithm to find a minimum spanning tree of the graph below.

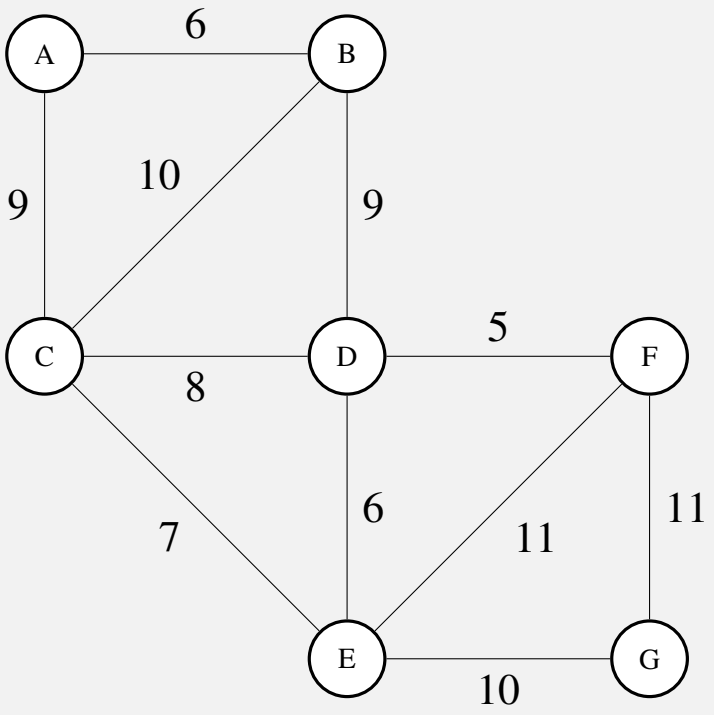


Figure 3.9

Note 3.14 Notice that in Question 3.12 you never had to choose between two edges of equal weight, making the minimum spanning tree you found unique. However in Question 3.13 you had to choose between edges *AC* and *BD*, and whichever one you chose the other wasn't included, making the minimum spanning tree non-unique.

3.4 Prim's Algorithm: Matrix Form

Prim's Algorithm can also be applied in a less visual way, in the form of a distance matrix.

Definition 3.15 A **distance matrix** for a graph is a square array where the entry in row *A*, column *B* is the weight of the edge between vertices *A* and *B* (if it exists) or blank (if it does not).

Back to our original example in Figure 3.6, we have the following distance matrix.

	A	B	C	D	E	F
A	-	12	18	-	-	-
B	12	-	8	7	6	-
C	18	8	-	-	4	5
D	-	7	-	-	8	-
E	-	6	4	8	-	9
F	-	-	5	-	9	-

Table 3.2

The number in row *A*, column *B*, for example, is the weight of the edge between *A* and *B*. A dash is given if there is no edge between the two vertices. Note that this distance matrix differs in style to the matrices we saw before. In fact it may be more suitable to refer to this at a distance table.

However the convention in other literature is to refer to this as a distance matrix, so we will do the same here.

Algorithm 3.16 — Prim's Algorithm, Matrix Form. To find a minimum spanning tree of a simple, connected, weighted, undirected graph G with n vertices given in distance matrix form, Prim's algorithm consists of the following steps.

1. Choose a vertex, i.e. a column, at random. Cross out the corresponding row.
2. Choose the edge in the column with the smallest edge weight. This has the effect of choosing a second column. Cross out the corresponding row, which contains this edge.
3. Look at all selected columns, and choose the edge in those columns which has the smallest edge weight and is not crossed out. This has the effect of choosing another column. Cross out the corresponding row, which contains this edge.
4. Repeat step 3 until all columns are selected.

Note: If at any point there is a choice between two or more edges of equal weight, choose one at random.

Example 3.17 Apply the matrix form of Prim's Algorithm to the distance matrix given below.

	A	B	C	D	E	F
A	-	12	18	-	-	-
B	12	-	8	7	6	-
C	18	8	-	-	4	5
D	-	7	-	-	8	-
E	-	6	4	8	-	9
F	-	-	5	-	9	-

Table 3.3

The following sequence of tables show Prim's Algorithm in action. We'll also start by choosing vertex E, to see the same sequence of decisions in a different context.

					*	
	A	B	C	D	E	F
A	-	12	18	-	-	-
B	12	-	8	7	6	-
C	18	8	-	-	4	5
D	-	7	-	-	8	-
E	-	~~6~~	4	~~8~~	-	~~9~~
F	-	-	5	-	9	-

Table 3.4

			*		*	
	A	B	C	D	E	F
A	-	12	18	-	-	-
B	12	-	8	7	6	-
C	~~18~~	~~8~~	-	-	**(4)**	~~5~~
D	-	7	-	-	8	-
E	-	~~6~~	4	~~8~~	-	~~9~~
F	-	-	5	-	9	-

Table 3.5

			*		*	*
	A	B	C	D	E	F
A	-	12	18	-	-	-
B	12	-	8	7	6	-
C	~~18~~	~~8~~	-	-	**(4)**	~~5~~
D	-	7	-	-	8	-
E	-	~~6~~	4	~~8~~	-	~~9~~
F	-	-	**(5)**	-	~~9~~	-

Table 3.6

		*	*		*	*
	A	B	C	D	E	F
A	-	12	18	-	-	-
B	~~12~~	-	~~8~~	~~7~~	**(6)**	-
C	~~18~~	~~8~~	-	-	**(4)**	~~5~~
D	-	7	-	-	8	-
E	-	~~6~~	4	~~8~~	-	~~9~~
F	-	-	**(5)**	-	~~9~~	-

Table 3.7

		*	*	*	*	*
	A	B	C	D	E	F
A	-	12	18	-	-	-
B	~~12~~	-	~~8~~	~~7~~	**(6)**	-
C	~~18~~	~~8~~	-	-	**(4)**	~~5~~
D	-	**(7)**	-	-	~~8~~	-
E	-	~~6~~	4	~~8~~	-	~~9~~
F	-	-	**(5)**	-	~~9~~	-

Table 3.8

	*	*	*	*	*	*
	A	B	C	D	E	F
A	-	**(12)**	~~18~~	-	-	-
B	~~12~~	-	~~8~~	~~7~~	**(6)**	-
C	~~18~~	~~8~~	-	-	**(4)**	~~5~~
D	-	**(7)**	-	-	~~8~~	-
E	-	~~6~~	4	~~8~~	-	~~9~~
F	-	-	**(5)**	-	~~9~~	-

Table 3.9

Notice here that we made the exact same sequence of decisions. We chose edges CE, then CF, BE, BD and AB.

Note 3.18 The reason this is the same Prim's algorithm as before is as follow. The algorithm begins the same in that we choose a vertex at random and then the "cheapest" edge. This edge was CE, so we then looked at all of the edges incident to C or E. Continuing in this way, the selected columns are the vertices already included in the tree T. The crossed out edges are the edges that cannot be chosen, as if we choose a crossed out edge in a selected column it is incident to two edges in T, and thus will create a cycle if included.

Rule 3.19 The test for whether or not a minimum spanning tree found using the matrix form of Prim's Algorithm is unique is identical to the first form of Prim's Algorithm. If at any point one chooses at random between edges of equal weight, there may be more than one minimum spanning tree. To see if the spanning tree is unique, write down the equal weight edges and choose one. If the other edge(s) are not in the final minimum spanning tree the minimum spanning tree is not unique. If this never happens in the construction of the minimum spanning tree then it is unique. However to check if, say, AB is in the minimum spanning tree, you must check position AB **and** BA in the distance matrix.

Question 3.20 Create a distance matrix for the graph in Figure 3.4, and apply the matrix version of Prim's Algorithm to find a minimum spanning tree.

Question 3.21 Create a distance matrix for the graph in Figure 3.5, and apply the matrix version of Prim's Algorithm to find a minimum spanning tree.

3.5 Notes on Kruskal and Prim's Algorithm

3.5.1 Similarities

- **Uniqueness:** Sometimes there is more than one minimum spanning tree. If two or more edges have the same edge weight, and in applying Kruskal's or Prim's algorithm you can choose to add one of them, and furthermore the inclusion of one affects whether another will be included, the minimum spanning tree is not unique.

- **Algorithm Type:** Both Kruskal's and Prim's Algorithm are known as **greedy** algorithms. A greedy algorithm is one that makes the most desirable decision now, without regard to the consequences down the line (hence the name).

- **Undirected:** Both algorithms are only for undirected graphs.

- **Weighted:** Both algorithms are only for weighted graphs.

- **Simple:** It is stated that these algorithms can only be used for simple graphs. One could use these algorithms on multigraphs by removing loops and multiple edges that are not the edge of minimum weight between two nodes, and apply the algorithm to the resulting subgraph. Because this is not an interesting generalisation it is not something we will study.
- **Connected:** It is also stated that these algorithms can only be used for simple graphs. For a disconnected graph, one could apply either algorithm to each connected component. Again, as this is not an interesting generalisation it is not something we will study.

3.5.2 Differences

- **Visualisation:** Prim's algorithm can be applied to a distance matrix. However to apply Kruskal's algorithm one needs a picture of the graph. Sometimes data is more naturally given in the form of a distance matrix, and when the number of nodes are large it can be hard to even draw a graph.
- **Speed:** Prim's algorithm is faster when the graph is dense (i.e. when there are many edges), and Kruskal's algorithm is faster when the graph is sparse (i.e. when there are few edges). By faster we mean which one is eventually faster when the number of vertices is very large. This is an important consideration in computer science.
- **Connectedness:** Prim's algorithm constructs a larger tree each step. The subgraph in Kruskal's algorithm is often not a tree and may even be disconnected at some steps along the way.
- **Approach:** Kruskal's algorithm works by adding the cheapest valid arc at each step. Prim's algorithm works by adding a new point in the cheapest way. (**K**ruskal $\leftrightarrow$ ar**k**, **P**rim $\leftrightarrow$ **p**oint).

Note 3.22 It is common for students to get confused between Prim and Kruskal's algorithm. To remember which is which, Kruska**L** is when we make a **L**ist, and Pri**M** is when we start with a vertex at rando**M**, or use a **M**atrix.

3.6 Dijkstra's Algorithm

Given a graph, how to we find the shortest path between two vertices? This is a reasonable question, and one that GPS systems such as Google Maps answer all the time. Disjkstra's Algorithm gives us the shortest distance from one vertex, called the source vertex, to any other vertex in the graph. We'll first consider Dijkstra's Algorithm for an undirected graph, as in this case the shortest path from the source vertex to any other vertex is the same as the shortest path from this vertex to the source vertex.

Algorithm 3.23 — Dijkstra's Algorithm for Undirected Graphs. Given a simple, connected, weighted, undirected graph G where the weights represent a distance, to find the shortest path between any vertex and the source vertex A, Dijkstra's algorithm consists of the following steps.

1. Draw a table where the first column lists the vertices, and the second is titled "Tentative Paths".
2. Start with the vertex A. The distance from A to A is 0. As this is the actual distance to A, put a tick beside vertex A.
3. (a) Consider every vertex adjacent to A. The weighted edge incident to A and these vertices gives a "tentative" path from A to each of these vertices, along with a distance. Write down these tentative paths (and the distance) in the table.

   (b) Choose the vertex with the smallest tentative distance; this tentative distance is the actual distance from A to this vertex (via the associated path). As the shortest path from A to this vertex is now known put a tick beside this vertex.
4. (a) Consider all vertices adjacent to the vertex ticked in the previous step that are not already ticked. Use the distance and path from A to the ticked vertex and the edge connecting the ticked vertex to these adjacent vertices to give a tentative distance and path from A to these adjacent vertices. If any of these adjacent vertices already have a tentative path, the new tentative path is whichever path has a smaller distance (if they're equal choose one at random).

   (b) Choose the unticked vertex with the smallest tentative distance (if two of more are equal choose one at random); this tentative distance is the actual distance from A to this vertex (via the associated path). Tick the vertex, as the shortest path from A to this vertex is now known.
5. Repeat step 4 until all vertices are ticked/chosen.

Note 3.24 If in steps 3 or 4 a new and old tentative distance are equal and one is chosen at random, and moreover if this tentative distance ends up being the actual distance for this vertex then the shortest path to this vertex is not unique. Other paths confirmed to be the shortest path after this may not be unique either. On the other hand, if two unticked vertices have a joint lowest tentative distance and one is chosen at random the other one will be chosen in the next step so it doesn't matter which one is chosen.

Example 3.25 Use Dijkstra's Algorithm to find the shortest paths between all vertices and vertex A in the graph below.

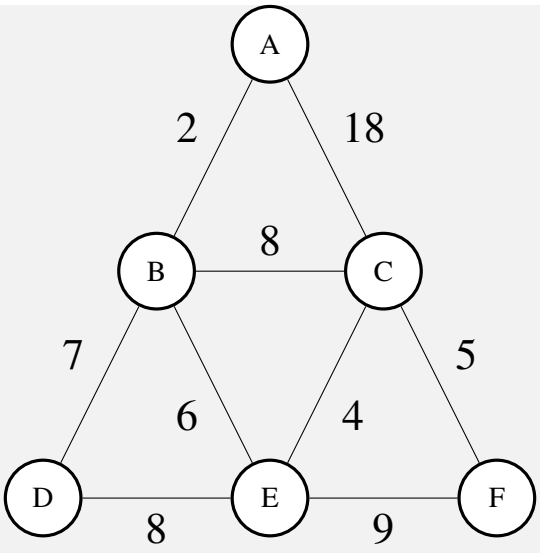


Figure 3.10

The following sequence of tables show Dijkstra's Algorithm in action. By the end of step 2 we have Table 3.10. By the end of step 3 we have Table 3.11. All subsequent tables are results of repeated applications of step 4. In the case where we get a new tentative distance the worst of the two tentative paths is removed with a ~~strikethrough~~. In reality, you may notice that a new path is non-optimal before you write it down and choose not to add it at all.

	Vertex	Tentative Paths
✓	A	(0)
	B	
	C	
	D	
	E	
	F	

Table 3.10

	Vertex	Tentative Paths
✓	A	(0)
✓	B	AB(2)
	C	AC(18)
	D	
	E	
	F	

Table 3.11

	Vertex	Tentative Paths
✓	A	(0)
✓	B	AB(2)
	C	~~AC(18)~~, ABC(10)
	D	ABD(9)
✓	E	ABE(8)
	F	

Table 3.12

	Vertex	Tentative Paths
✓	A	(0)
✓	B	AB(2)
	C	~~AC(18)~~, ABC(10)
✓	D	ABD(9)
✓	E	ABE(8)
	F	ABEF(17)

Table 3.13

	Vertex	Tentative Paths
✓	A	(0)
✓	B	AB(2)
✓	C	~~AC(18)~~, ABC(10)
✓	D	ABD(9)
✓	E	ABE(8)
	F	ABEF(17)

Table 3.14

	Vertex	Tentative Paths
✓	A	(0)
✓	B	AB(2)
✓	C	~~AC(18)~~, ABC(10)
✓	D	ABD(9)
✓	E	ABE(8)
✓	F	~~ABEF(17)~~, ABCF(15)

Table 3.15

For example, consider the application of step 4 after Table 3.12. We have just ticked/chosen vertex E. There are four vertices adjacent to E; B, C, D and F. B is already ticked so we ignore it. For C, we could add the path $ABEC$ to the list of tentative paths to C, which consists of ABE, the optimal path from A to E, combined with the direct path EC from E to CD. However the distance is then $8+4$, the length of ABE plus the length of EC. This path is worse than the tentative path ABC we already have for C, so we could not bother writing it at all. The same is true for D and the path $ABED(16)$, but for F we actually do add the path. Out of the unticked vertices C has the lowest tentative distance, so we tick this vertex; the actual shortest path from A to C is ABC and is of length 10.

Note 3.26 Once students are more familiar with the steps outlined in Dijkstra's Algorithm, they may notice that step 3 and 4 are not fundamenally different. Moreover they can both be seen as a two-step process where we add tentative paths, then choose a new vertex, repeating until all vertices are chosen/ticked. However this memory tool will be more useful after students have become familiar with the method as it is more formally outlined in Algorithm 3.23.

Question 3.27 Use Dijkstra's Algorithm to find the shortest distance from vertex A to each of the vertices in the graph below.

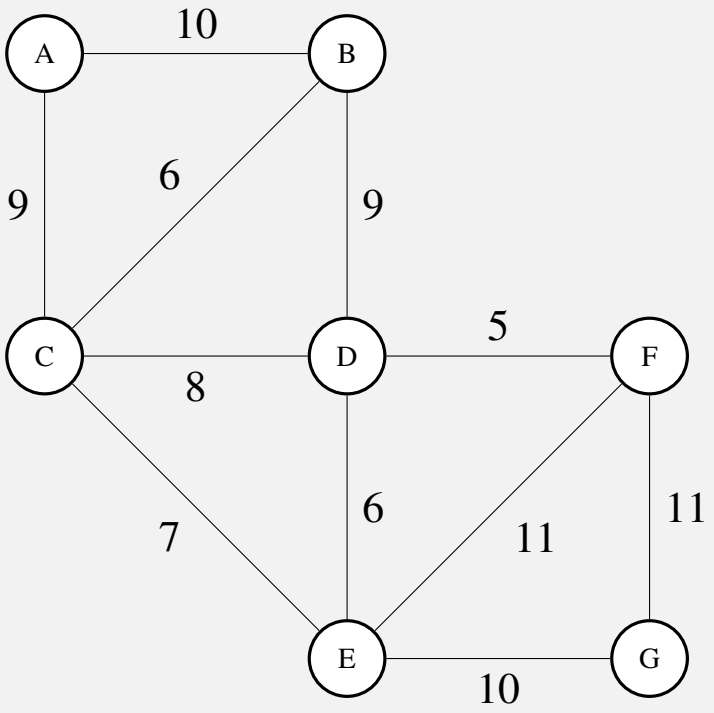


Figure 3.11

Note 3.28 There is an interesting principle at play here. Looking at our solution, the optimal path from A to F is $ABCF$. The optimal path from A to C is ABC, a "subpath" of $ABCF$. Similarly the optimal path between A and B is AB. You could also probably convince yourself (without applying Dijkstra's Algorithm again) that the optimal path between B and C is BC as well, another subpath of $ABCF$. This is no accident.

Theorem 3.29 — Bellman's Principle of Optimality. Any part of an optimal path is itself optimal.

What does this mean? Well, if the optimal path from A to F is $ABCF$, then the optimal path to C has to be ABC, and the optimal path to B has to be AB. In real world terms, if the shortest path from you house to school passes the library, then this path also has to contain the shortest path to the library. If instead there was a faster way to the library you could use that path on your way to school, making your trip to school shorter!

Example 3.30 In a given graph, the shortest path from A to L is $ABGHKL$. What is the shortest path from B to K?

It's $BGHK$.

Question 3.31 In a given graph, the shortest path from A to L is $ABGHKL$. What is the shortest path from G to L?

3.7 Dijkstra's Algorithm for Directed Graphs

Dijkstra's Algorithm also works on directed graphs. It is not fundamentally different, except in one small respect.

Algorithm 3.32 — Dijkstra's Algorithm for Directed Graphs. Given a simple, connected, weighted, directed graph G where the weights represent a distance, Algorithm 3.23 can be followed with some alterations to steps 3 and 4.

1. When trying to find the shortest paths **from** A to each vertex, only consider adjacent vertices if the edge goes from the ticked vertex to the adjacent vertex.
2. When trying to find the shortest paths **to** A from each vertex, only consider adjacent vertices if the edge goes from the adjacent vertex to the ticked vertex.

Example 3.33 Use Dijkstra's Algorithm to find the shortest paths from A to all other vertices in the graph below.

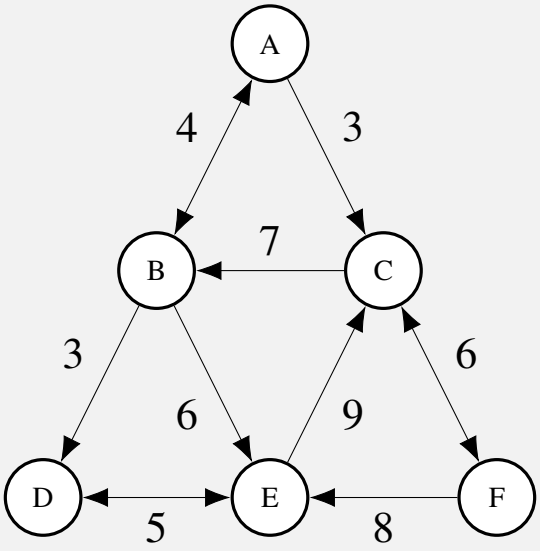


Figure 3.12

The following sequence of tables show Dijkstra's Algorithm in action.

	Vertex	Tentative Paths
✓	A	(0)
	B	
	C	
	D	
	E	
	F	

Table 3.16

	Vertex	Tentative Paths
✓	A	(0)
	B	AB(4)
✓	C	AC(3)
	D	
	E	
	F	

Table 3.17

	Vertex	Tentative Paths
✓	A	(0)
✓	B	AB(4)
✓	C	AC(3)
	D	
	E	
	F	ACF(9)

Table 3.18

	Vertex	Tentative Paths
✓	A	(0)
✓	B	AB(4)
✓	C	AC(3)
✓	D	ABD(7)
	E	ABE(10)
	F	ACF(9)

Table 3.19

	Vertex	Tentative Paths
✓	A	(0)
✓	B	AB(4)
✓	C	AC(3)
✓	D	ABD(7)
	E	ABE(10)
✓	F	ACF(9)

Table 3.20

	Vertex	Tentative Paths
✓	A	(0)
✓	B	AB(4)
✓	C	AC(3)
✓	D	ABD(7)
✓	E	ABE(10)
✓	F	ACF(9)

Table 3.21

Notice how, from Table 3.11 to Table 3.12, we didn't add a tentative path to E even though C and E are adjacent. This is because the edge runs in the wrong direction, we can't go from C to E along this path.

Example 3.34 Use Dijkstra's Algorithm to find the shortest paths from all vertices to A in the graph below.

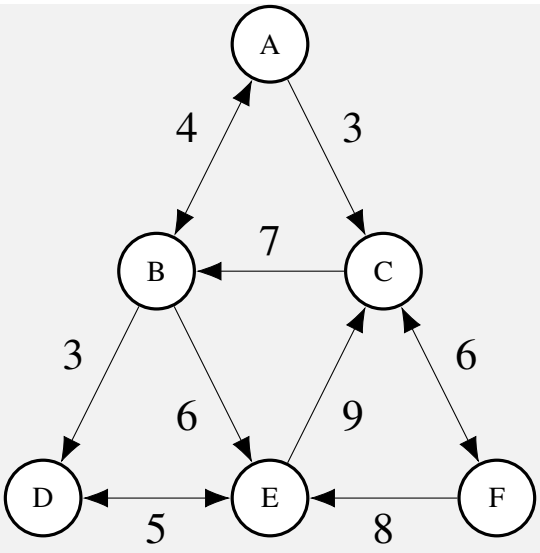


Figure 3.13

Notice how in this case the paths end at A and so when adding tentative paths the edge connecting the adjacent vertices comes first and the known shortest path comes after.

	Vertex	Tentative Paths
✓	A	(0)
	B	
	C	
	D	
	E	
	F	

Table 3.22

	Vertex	Tentative Paths
✓	A	(0)
✓	B	BA(4)
	C	
	D	
	E	
	F	

Table 3.23

	Vertex	Tentative Paths
✓	A	(0)
✓	B	BA(4)
✓	C	CBA(11)
	D	
	E	
	F	

Table 3.24

	Vertex	Tentative Paths
✓	A	(0)
✓	B	BA(4)
✓	C	CBA(11)
	D	
	E	ECBA(20)
✓	F	FCBA(17)

Table 3.25

	Vertex	Tentative Paths
✓	A	(0)
✓	B	BA(4)
✓	C	CBA(11)
	D	
✓	E	ECBA(20)
✓	F	FCBA(17)

Table 3.26

	Vertex	Tentative Paths
✓	A	(0)
✓	B	BA(4)
✓	C	CBA(11)
✓	D	DECBA(25)
✓	E	ECBA(20)
✓	F	FCBA(17)

Table 3.27

See that, from Table 3.22 to 3.23 we only add the path BA, as the edge between A and C goes the wrong way. On the topic of writing the paths, consider the paths we find between Tables 3.24 and 3.25. We combine the edge EC to the known shortest path CBA from C to A to get the path $ECBA$ from E to A, and similarly for F. Note finally how from Table 3.25 to 3.26 we don't add any more paths; this is fine and will happen occasionally, even with undirected graphs.

3.8 Notes on Dijkstra's Algorithm

- **Uniqueness:** There may be more than one shortest distance between two nodes. In Dijkstra's Algorithm you may have a choice of paths to a vertex when comparing distances which are equal. If this is the case then any of the shortest paths that contain this vertex will not be unique.
- **Distance to all nodes:** Dijkstra's Algorithm gives the shortest distance from the source node to every node.
- **Directed:** Dijkstra's Algorithm can be used on directed and undirected graphs.
- **Unweighted graphs:** Dijkstra's Algorithm can be used on unweighted graphs by giving

each edge a weight of 1. This gives the "shortest path" as the path with the fewest edges.

- **Simple:** It is stated that Dijkstra's Algorithm can only be used for simple graphs. One could use this algorithm on multigraphs by removing loops and multiple edges that are not the edge of minimum weight between two nodes, and apply the algorithm to the resulting subgraph. Because this is not an interesting generalisation it is not something we will study.
- **Greedy:** Dijkstra's algorithm is a greedy algorithm as at each step we decide what the optimal path for a new vertex is without considering what's about to happen next as we continue to explore the graph.
- **Visualisation:** Dijkstra's algorithm requires a picture of the graph.
- **Approach:** Dijkstra's Algorithm figures out the shortest path for a new vertex each step.
- **Distance between two nodes:** Some problems only ask for the shortest path between two nodes. This means, as a result of the approach, that sometimes you can finish Dijkstra's Algorithm early.

3.9 Exam Questions

3.9.1 Prim & Kruskal's Algorithm

Edexcel exam questions on Prim and Kruskal's algorithm are quite standard. They often ask if the minimum spanning trees found are unique. They also stipulate whether Prim or Kruskal's algorithm should be used and, in the case of Prim's algorithm, which vertex should be used as a starting point. Some questions ask the student to draw the minimum spanning tree. Occasionally questions stipulate that certain arcs have to be part of the minimum spanning tree; in this case Kruskal's algorithm must be used with the arcs placed on top of the list and included, regardless of their weights. Questions like this are unlikely to appear on the Leaving Cert exams but are an interesting application.

Question 3.35 — Old D1 January 2002 Q3(i).

(i)

	A	*B*	*C*	*D*	*E*	*F*
A	–	10	12	13	20	9
B	10	–	7	15	11	7
C	12	7	–	11	18	3
D	13	15	11	–	27	8
E	20	11	18	27	–	18
F	9	7	3	8	18	–

The table shows the distances, in metres, between six nodes *A*, *B*, *C*, *D*, *E*, and *F* of a network.

(*a*) Use Prim's algorithm, starting at *A*, to solve the minimum connector problem for this table of distances. Explain your method and indicate the order in which you selected the edges.

(4)

(*b*) Draw your minimum spanning tree and find its total length.

(2)

(*c*) State whether your minimum spanning tree is unique. Justify your answer.

(1)

Figure 3.14

Question 3.36 — Old D1 January 2004 Q6.

	A	B	C	D	E	F
A	–	7	3	–	8	11
B	7	–	4	2	–	7
C	3	4	–	5	9	–
D	–	2	5	–	6	3
E	8	–	9	6	–	–
F	11	7	–	3	–	–

The matrix represents a network of roads between six villages A, B, C, D, E and F. The value in each cell represents the distance, in km, along these roads.

(a) Show this information on the diagram in the answer book.

(2)

(b) Use Kruskal's algorithm to determine the minimum spanning tree. State the order in which you include the arcs and the length of the minimum spanning tree. Draw the minimum spanning tree.

(5)

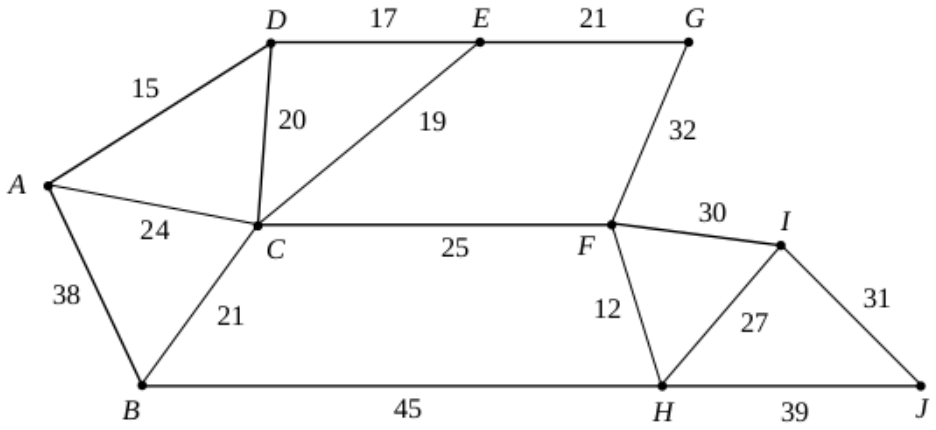
(c) Starting at D, use Prim's algorithm on the matrix given in the answer book to find the minimum spanning tree. State the order in which you include the arcs.

(3)

Figure 3.15

Question 3.37 — Old D1 January 2005 Q3.

Figure 2



The network in Figure 2 shows the distances, in metres, between 10 wildlife observation points. The observation points are to be linked by footpaths, to form a network along the arcs indicated, using the least possible total length.

(*a*) Find a minimum spanning tree for the network in Figure 2, showing clearly the order in which you selected the arcs for your tree, using

(i) Kruskal's algorithm,

(3)

(ii) Prim's algorithm, starting from *A*.

(3)

Given that footpaths are already in place along *AB* and *FI* and so should be included in the spanning tree,

(*b*) explain which algorithm you would choose to complete the tree, and how it should be adapted. (You do **not** need to find the tree.)

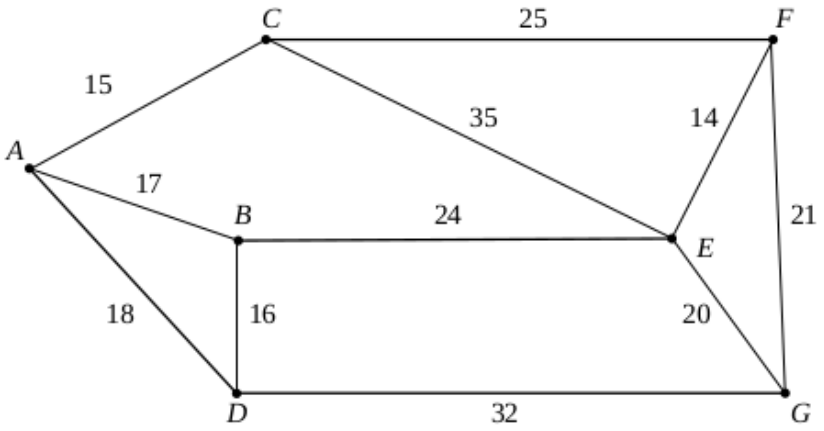
(2)

Figure 3.16

Question 3.38 — Old D1 June 2003 Q3.

(*a*) Describe the differences between Prim's algorithm and Kruskal's algorithm for finding a minimum connector of a network.

Figure 2



(*b*) Listing the **arcs** in the order that you select them, find a minimum connector for the network in Fig. 2, using

(i) Prim's algorithm,

(ii) Kruskal's algorithm.

(4)

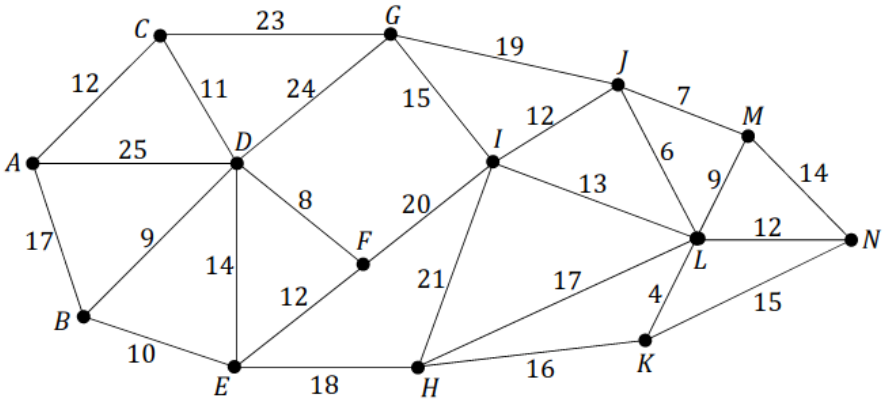
Figure 3.17

The questions on minimum spanning trees in the Applied Maths Sample paper and 2023 paper did not stipulate whether Prim or Kruskal's algorithm should be used. It was also less specific than the Edexcel papers in the form of the answer, not asking students to list the arcs in order of inclusion, simply saying "relevant supporting work must be shown". If applying Kruskal's algorithm showing our list of edges complete with ✓ and ✗ is sufficient. If applying either the graph or distance matrix form of Prim's algorithm it should be stated which vertex the student is starting with and listing the edges included in order of inclusion.

Example 3.39 — Sample Paper Q5(a)(ii).

Question 5

(a) In the network shown below, the edges represent roads and the nodes represent the junctions of two or more roads, labelled with the letters A to N. The weight of each edge represents the distance (in km) between a pair of junctions.



(ii) A group of engineers want to close down some of the roads to carry out maintenance work. They wish to close down as much of the road network as possible while still allowing a person to drive between any two junctions on the network.

Using an appropriate algorithm, find the minimum spanning tree for the network. Name the algorithm you used. Relevant supporting work must be shown.

Figure 3.18

We will show a solution for both methods.

Kruskal's Algorithm: The table on the right below is a sufficient answer; however students may find it helpful to highlight edges on the graph shown as they are included to make sure cycles aren't made. This should be done in pencil in case mistakes are made.

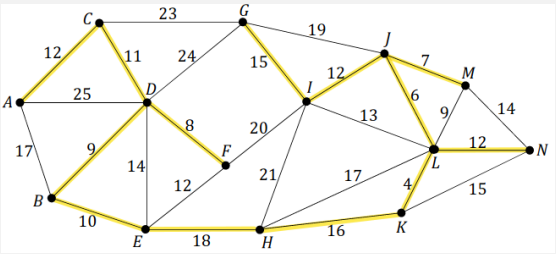


Figure 3.19

Edge	Weight	
KL	4	✓
JL	6	✓
JM	7	✓
DF	8	✓
BD	9	✓
LM	9	✗
BE	10	✓
CD	11	✓
EF	12	✗
IJ	12	✓
LN	12	✓
AC	12	✓
IL	13	✗
MN	14	✗
DE	14	✗
GI	15	✓
KN	15	✗
HK	16	✓
AB	17	✗
HL	17	✗
EH	18	✓
GJ	19	
FI	20	
HI	21	
CG	23	
DG	24	
AD	25	

Table 3.28

Prim's Algorithm: If we start with vertex A, we get the minimum spanning tree below, along with the order of inclusion.

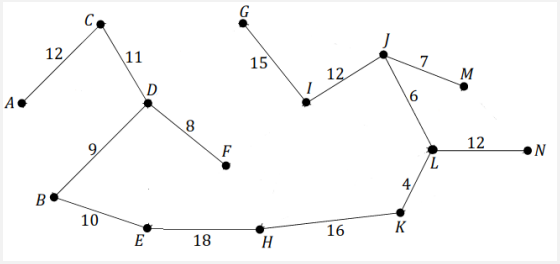


Figure 3.20

Edges in order of inclusion
AC
CD
DF
BD
BE
EH
HK
KL
JL
JM
LN
IJ
GI
EH

Table 3.29

3.9.2 Dijkstra's Algorithm

Edexcel exam questions on Dijkstra's Algorithm usually consist of a standard question asking students to apply Dijkstra's algorithm to a graph. Some questions then ask the student to find the shortest route that does or does not go through a given node, or asks a question about the shortest distance between another two nodes that requires applying Bellman's Principle of Optimality. The EdExcel papers had no questions on Dijkstra's algorithm for directed graphs.

Question 3.40 — Old D1 January 2002 Q4.

Figure 1

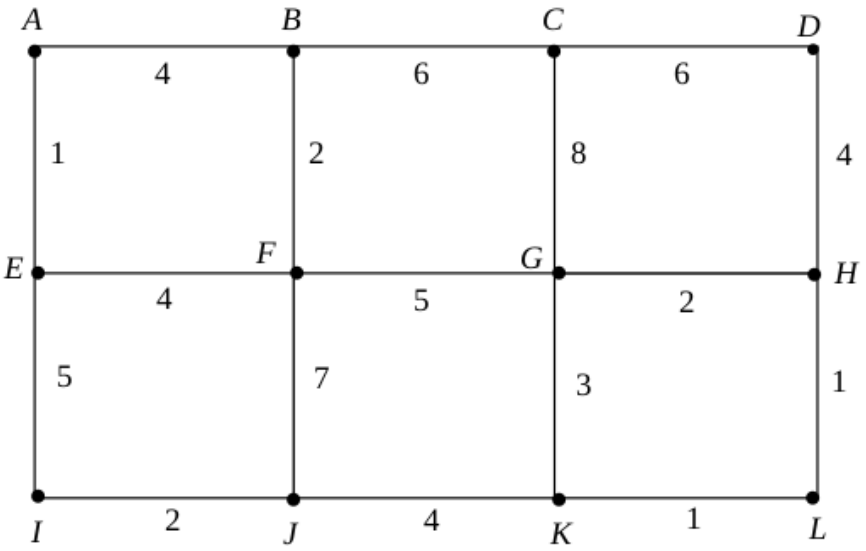


Figure 1 shows a network of roads. Erica wishes to travel from A to L as quickly as possible. The number on each edge gives the time, in minutes, to travel along that road.

(a) Use Dijkstra's algorithm to find a quickest route from A to L. Complete all the boxes on the answer sheet and explain clearly how you determined the quickest route from your labelling.

(7)

(b) Show that there is another route which also takes the minimum time

(1)

Figure 3.21

Question 3.41 — Old D1 January 2009 Q6.

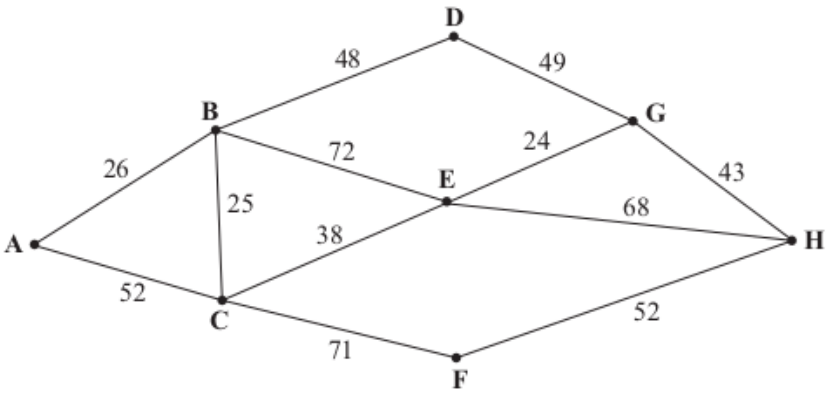


Figure 4

Figure 4 shows a network of roads through eight villages, A, B, C, D, E, F, G and H. The number on each arc is the length of that road in km.

(a) Use Dijkstra's algorithm to find the shortest route from A to H. State your shortest route and its length.

(5)

There is a fair in village C and you cannot drive through the village. A shortest route from A to H which avoids C needs to be found.

(b) State this new minimal route and its length.

(2)

Figure 3.22

Question 3.42 — Old D1 January 2011 Q1.

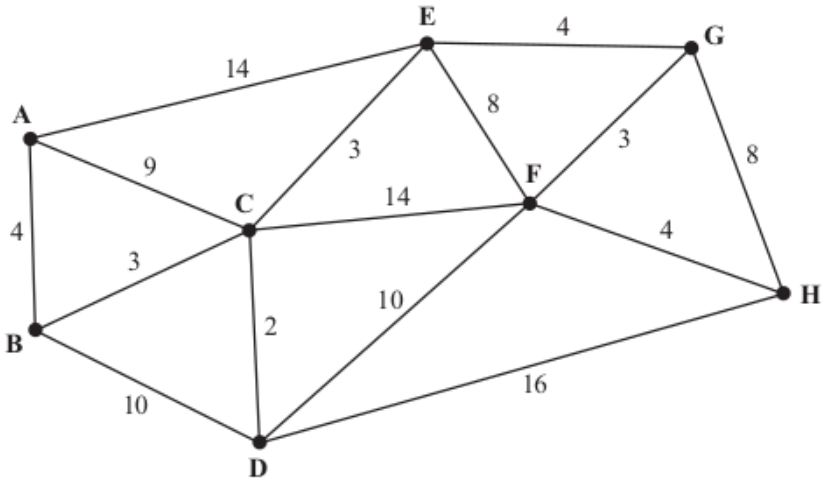


Figure 1

Figure 1 shows a network of roads between eight villages, A, B, C, D, E, F, G and H. The number on each arc gives the length, in miles, of the corresponding road.

(a) Use Dijkstra's algorithm to find the shortest distance from A to H.

(5)

(b) State your shortest route.

(1)

(c) Write down the shortest route from H to C and state its length.

(2)

Figure 3.23

Question 3.43 — Old D1 January 2012 Q4.

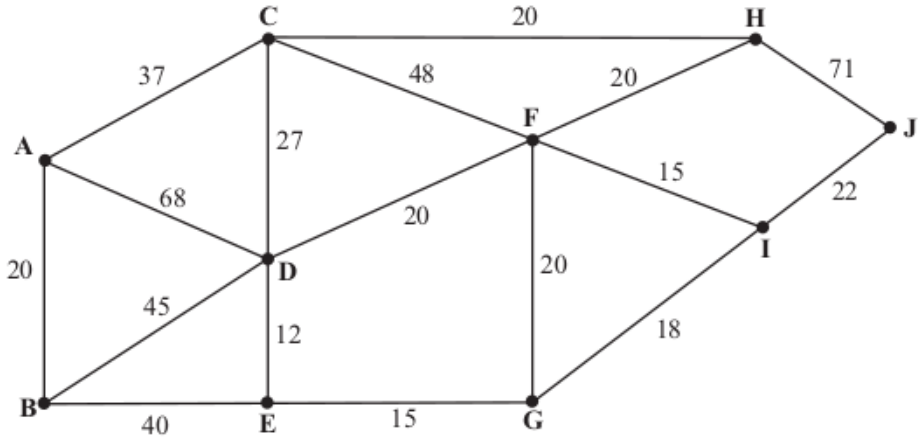


Figure 5

Figure 5 models a network of roads. The number on each edge gives the time, in minutes, taken to travel along that road. Olivia wishes to travel from A to J as quickly as possible.

(a) Use Dijkstra's algorithm to find the shortest time needed to travel from A to J. State the shortest route.

(7)

On a particular day Olivia must include G in her route.

(b) Find a route of minimal time from A to J that includes G, and state its length

(2)

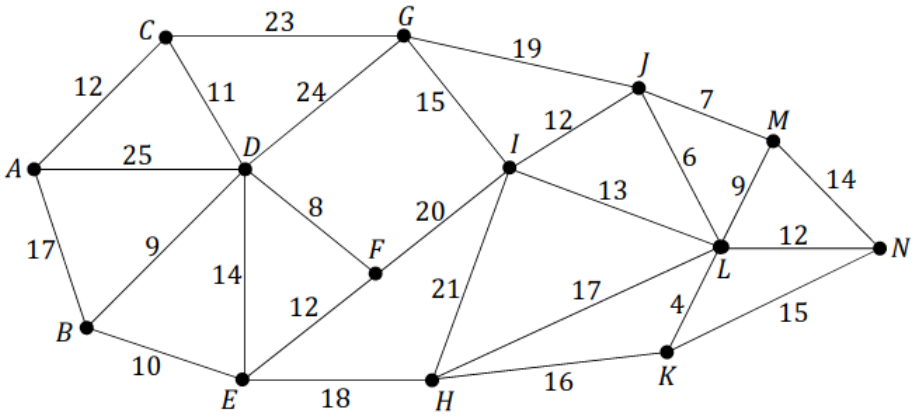
Figure 3.24

The sample paper and 2023 paper both had one question on Dijkstra's Algorithm which were completely standard, if a bit long.

Example 3.44 — Sample Paper Q5(a)(ii).

Question 5

(a) In the network shown below, the edges represent roads and the nodes represent the junctions of two or more roads, labelled with the letters A to N. The weight of each edge represents the distance (in km) between a pair of junctions.



(i) Use Dijkstra's algorithm to find the shortest path from junction A to junction N. Calculate the length of the shortest path. Relevant supporting work must be shown.

Figure 3.25

The solution without workings is shown below.

	Vertex	Tentative Paths
✓	A	(0)
✓	B	AB(17)
✓	C	AC(12)
✓	D	ACD(23)
✓	E	ABE(27)
✓	F	ACDF(31)
✓	G	ACG(35)
✓	H	ABEH(45)
✓	I	ACGI(50)
✓	J	ACGJ(54)
✓	K	ABEHK(61)
✓	L	ACGJL(60)
✓	M	ACGJM(61)
✓	N	ACGJLN(72)

Therefore the shortest path from A to N is $ACGJLN$ which is of length 72 km.

3.10 Summary

- Understand what an algorithm is.
- Be able to use Kruskal's Algorithm.
- Be able to use Prim's Algorithm using a graph.
- Be able to use Prim's Algorithm using a distance matrix.
- Know that Kruskal and Prim's Algorithm give a Minimum Spanning Tree for the original

graph.

- Know when Kruskal and Prim's Algorithm result in a non-unique Minimum spanning Tree.
- Understand that both Kruskal and Prim's Algorithm are for weighted, undirected, simple, connected graphs.
- Know that both Kruskal and Prim's Algorithm are greedy algorithms.
- Know that while Kruskal's Algorithm can only be applied using a graph, Prim's Algorithm can be applied to a graph or to a distance matrix.
- Understand how Kruskal and Prim's Algorithm vary in approach; **K**ruskal's Algorithm adds an ar**k** at each step, and may construct a disconnected subgraph at times before it is finished, whereas **P**rim's algorithm adds a **p**oint and so always constructs a tree at each step.
- Be able to use Dijkstra's Algorithm for both directed and undirected graphs.
- Understand that Dijkstra's Algorithm gives the shortest path between the source node and every other node, and so if you only want the shortest path between two nodes you may be able to finish the algorithm early.
- Understand Bellman's Principle of Optimality and its consequences.
- Know when Dijkstra's Algorithm results in a non-unique shortest path, and which paths are non-unique.
- Understand that Dijkstra's Algorithm can also be used on directed or undirected, and on weighted or unweighted graphs, but that the graph should be simple and connected.
- Know that Dijkstra's Algorithm is a greedy algorithm.
- Know that Dijkstra's Algorithm can only be applied using a graph.
- Understand the approach of Disjktra's Algorithm; it figures out the shortest path for a single new vertex at each step.
- As a final note, we have covered how to apply Kruskal's, Prim's and Dijkstra's Algorithm. However we did not study why they give the optimal paths or minimum spanning tree. This will be revisited in 6th Year.

3.11 Notes on the Exam, and Work Still to Cover

Exam questions on this topic are likely to mainly consist a straightforward application of Prim, Kruskal or Dijkstra's algorithm, although they may include some "curveball" questions on topics like Bellman's Principle of Optimality or similar, like the EdExcel past papers.

So far both past papers (the sample and 2023) gave students the choice between using Prim and Kruskal's algorithm and there have been no questions asking students to apply Dijkstra's algorithm to a directed graph.

3.12 Homework

Kruskal's Algorithm

1. Use Kruskal's Algorithm to find a Minimum Spanning Tree for each of the following graphs. State in each case whether or not the tree is unique. If there is more than one, draw the other tree(s) (it is not required to show your work for the additional tree(s)).

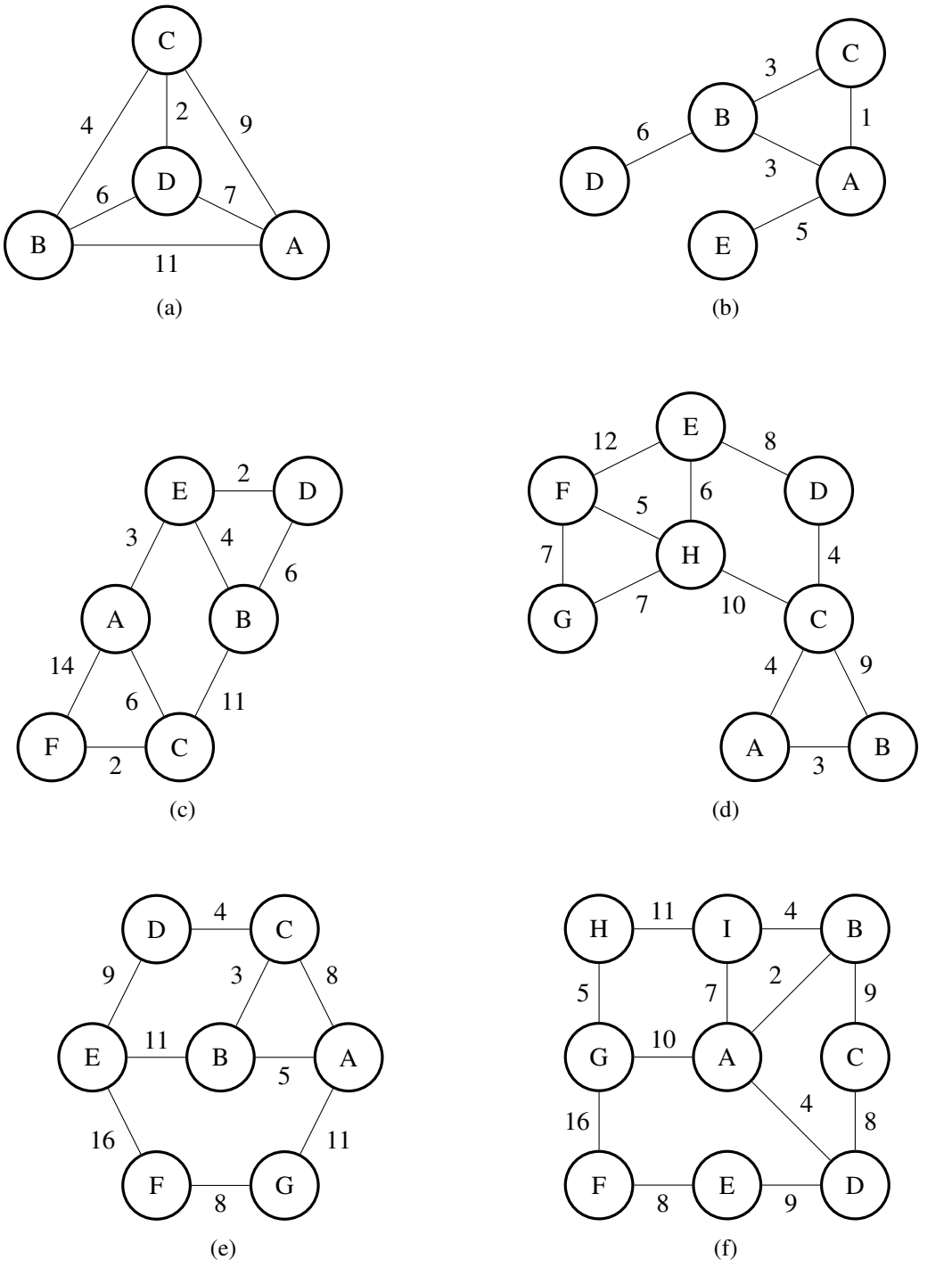


Figure 3.26

Prim's Algorithm

2. Use Prim's Algorithm to find a Minimum Spanning Tree for each of the graphs in Figure 3.26. State in each case whether or not the tree is unique. If there is more than one, draw the other tree(s) (it is not required to show your work for the additional tree(s)).

Prim's Algorithm: Matrix Form

3. For each of the subfigures in Figure 3.26, write down the distance matrix (no need to Apply Prim's Algorithm).

4. Apply the matrix form of Prim's Algorithm to each of the following distance matrices. State in each case whether or not the Minimum Spanning Tree you find is unique. If there is more than one, find the other tree(s) (it is not required to show your work for the additional tree(s)). Draw each of the trees, and calculate their total weight.

	A	B	C	D
A	-	3	7	-
B	3	-	5	5
C	7	5	-	4
D	-	5	4	-

Table 3.30

	A	B	C	D	E	F
A	-	14	11	-	8	12
B	14	-	-	-	9	-
C	11	-	-	3	2	7
D	-	-	3	-	4	6
E	8	9	2	4	-	16
F	12	-	7	6	16	-

Table 3.31

	A	B	C	D	E	F	G	H	I	J
A	-	6	-	-	-	13	-	-	-	-
B	6	-	18	8	17	-	-	-	-	-
C	-	18	-	-	-	-	15	-	-	-
D	-	8	-	-	14	20	-	19	-	-
E	-	17	-	14	-	-	9	-	7	-
F	13	-	-	20	-	-	-	7	-	-
G	-	-	15	-	9	-	-	-	5	-
H	-	-	-	19	-	7	-	-	12	11
I	-	-	-	-	7	-	5	12	-	10
J	-	-	-	-	-	-	-	11	10	-

Table 3.32

Dijkstra's Algorithm

5. For each of the graphs in Figure 3.26, apply Dijkstra's Algorithm to find the shortest distance from each node to node A.

6. In a certain undirected, weighted graph, the shortest path between A and F is $ACGBDF$. What is the shortest path between C and D?

Dijkstra's Algorithm for Directed Graphs

7. In each of the following digraphs, use Dijkstra's Algorithm to find

   (a) the shortest path from A to each node,

   (b) the shortest path from each node to A.

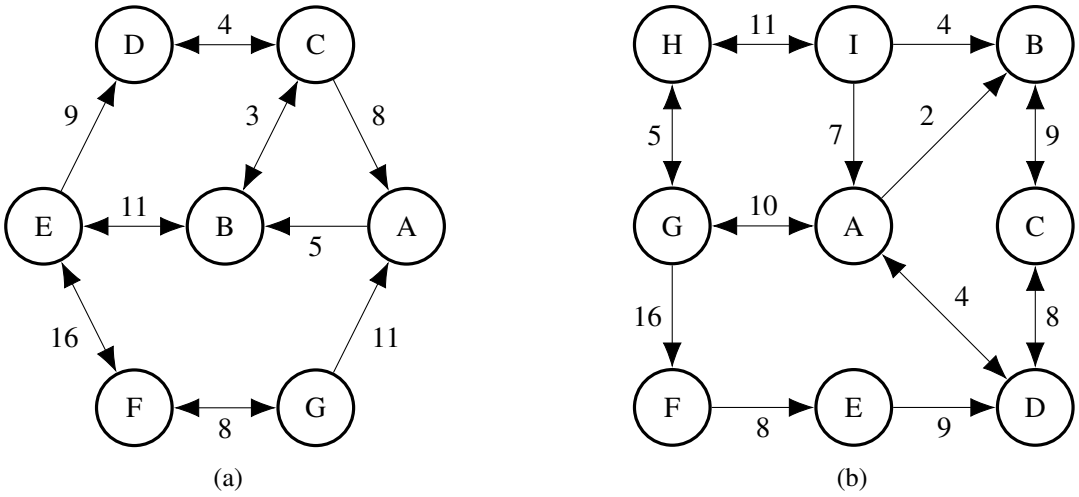
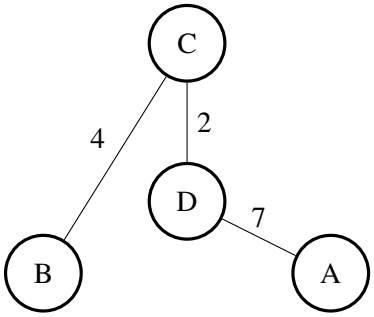


Figure 3.27

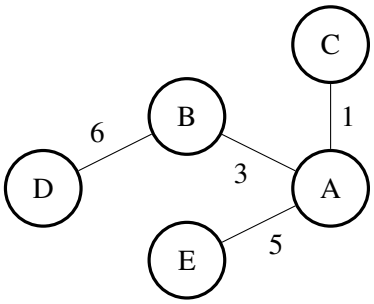
3.13 Homework Solutions

Kruskal's Algorithm

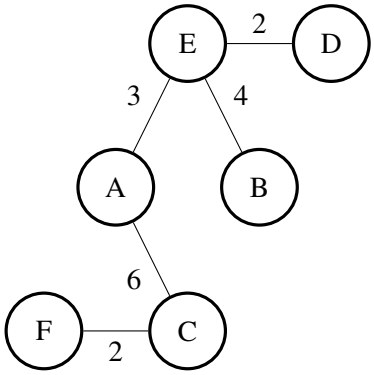
1.



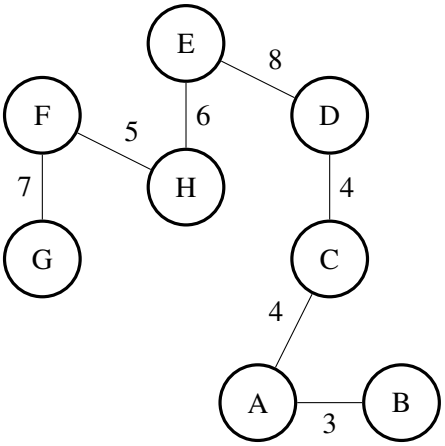
(a) Unique



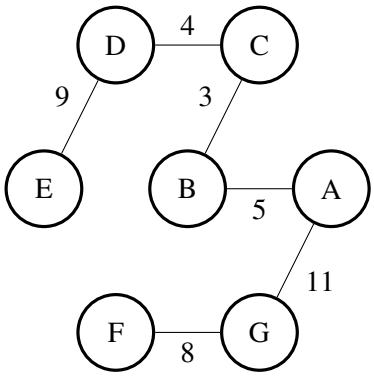
(b) Non-unique: can swap AB for BC



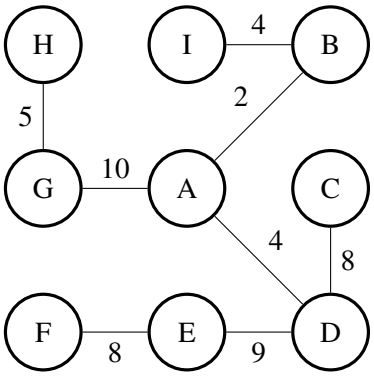
(c) Unique



(d) Non-unique: can swap FG for GH



(e) Unique



(f) Unique

Figure 3.28

Prim's Algorithm

2. Same as previous question.

Prim's Algorithm: Matrix Form

3.

	A	B	C	D
A	-	11	9	7
B	11	-	4	6
C	9	4	-	2
D	7	6	2	-

Table 3.33

	A	B	C	D	E
A	-	3	1	-	5
B	3	-	3	6	-
C	1	3	-	-	-
D	-	6	-	-	-
E	5	-	-	-	-

Table 3.34

	A	B	C	D	E	F
A	-	-	6	-	3	14
B	-	-	11	6	4	-
C	6	11	-	-	-	2
D	-	6	-	-	2	-
E	3	4	-	2	-	-
F	14	-	2	-	-	-

Table 3.35

	A	B	C	D	E	F	G	H
A	-	3	4	-	-	-	-	-
B	3	-	9	-	-	-	-	-
C	4	9	-	4	-	-	-	10
D	-	-	4	-	8	-	-	-
E	-	-	-	8	-	12	-	6
F	-	-	-	-	12	-	7	5
G	-	-	-	-	-	7	-	7
H	-	-	10	-	6	5	7	-

Table 3.36

	A	B	C	D	E	F	G
A	-	5	8	-	-	-	11
B	5	-	3	-	11	-	-
C	8	3	-	4	-	-	-
D	-	-	4	-	9	-	-
E	-	11	-	9	-	16	-
F	-	-	-	-	16	-	8
G	11	-	-	-	-	8	-

Table 3.37

	A	B	C	D	E	F	G	H	I
A	-	2	-	4	-	-	10	-	7
B	2	-	9	-	-	-	-	-	4
C	-	9	-	8	-	-	-	-	-
D	4	-	8	-	9	-	-	-	-
E	-	-	-	9	-	8	-	-	-
F	-	-	-	-	8	-	16	-	-
G	10	-	-	-	-	16	-	5	-
H	-	-	-	-	-	-	5	-	11
I	7	4	-	-	-	-	-	11	-

Table 3.38

4. There are many different final distance matrices with ~~strikethroughs~~ depending on what vertex/column you start with. These solutions instead list the edges of the minimum spanning tree, or trees if there is more than one.

(a) Tree is non-unique, with edges

$$\begin{array}{ccc} AB & & AB \\ CD & & CD \\ BC & or & BD \end{array}$$

(b) Tree is unique, with edges

$$\begin{array}{l} CE \\ CD \\ DF \\ AE \\ BE. \end{array}$$

(c) Tree is unique, with edges

$$\begin{array}{l} GI \\ EI \\ IJ \\ HJ \\ FH \\ AF \\ AB \\ BD \\ CG. \end{array}$$

Dijkstra's Algorithm

5. There is no need to have multiple answers here; they are written for students' benefit.

(a)

	Vertex	Tentative Paths
	A	(0)
	B	AB(11)
	C	AC(9) or ADC(9)
	D	AD(7)

(b)

	Vertex	Tentative Paths
	A	(0)
	B	AB(3)
	C	AC(1)
	D	ABD(9)
	E	AE(5)

(c)

	Vertex	Tentative Paths
	A	(0)
	B	AEB(7)
	C	AC(6)
	D	AED(5)
	E	AE(3)
	F	ACF(8)

(d)

	Vertex	Tentative Paths
	A	(0)
	B	AB(3)
	C	AC(4)
	D	ACD(8)
	E	ACDE(16)
	F	ACHF(19)
	G	ACHG(21)
	H	ACH(14)

(e)

	Vertex	Tentative Paths
	A	(0)
	B	AB(5)
	C	AC(8) or ABC(8)
	D	ACD(12) or ABCD(12)
	E	ABE(16)
	F	AGF(19)
	G	AG(11)

(f)

	Vertex	Tentative Paths
	A	(0)
	B	AB(2)
	C	ABC(11)
	D	AD(4)
	E	ADE(13)
	F	ADEF(21)
	G	AG(21)
	H	AGH(15)
	I	ABI(6)

6. *CGBD*.

Dijkstra's Algorithm for Directed Graphs

7. (a) (i)

	Vertex	Tentative Paths
	A	(0)
	B	AB(5)
	C	ABC(8)
	D	ABCD(12)
	E	ABE(16)
	F	ABEF(32)
	G	ABEFG(40)

(ii)

	Vertex	Tentative Paths
	A	(0)
	B	BCA(11)
	C	CA(8)
	D	DCA(12)
	E	EDCA(21)
	F	FGA(19)
	G	GA(11)

(b) (i)

	Vertex	Tentative Paths
	A	(0)
	B	AB(2)
	C	ABC(11)
	D	AD(4)
	E	AGFE(34)
	F	AGF(26)
	G	AG(10)
	H	AGH(15)
	I	AGHI(26)

(ii)

	Vertex	Tentative Paths
	A	(0)
	B	BCDA(21)
	C	CDA(12)
	D	DA(4)
	E	EDA(13)
	F	FEDA(21)
	G	GA(10)
	H	HGA(15)
	I	IA(7)

3.14 Revision

When revising this chapter, students are best off using the EdExcel papers as they are numerous and ask questions similar to those in the Leaving Cert exams.

As this is a new topic past exam questions on this chapter are only from the sample paper (Q5) and the 2023 paper (Q2 (a) and Q7 (a)).