SUMMARY SHEET – DECISION MATHS

Algorithms

The main ideas are covered in

AQA	Edexcel	MEI	OCR
D1	D1	D1	D1

The main ideas in this topic are

Understanding and implementing a variety of algorithms expressed as lists of instructions, flow charts or in pseudo code.

What is an algorithm?

An algorithm must have the following properties

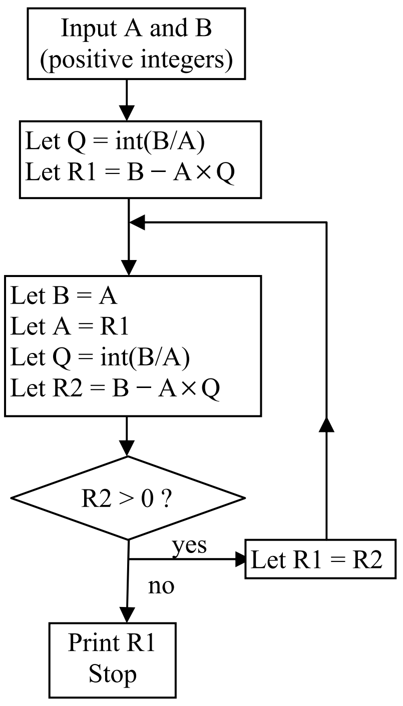
- it is a set of precisely defined instructions.
- it has generality: it will work for all valid inputs.
- it is finite: it has a stopping condition.
- it may be an iterative process: you may need to follow the procedure a number of times in order to reach the best solution.

Before the exam you should know:

- The three bin packing algorithms. These are the Full-Bin Algorithm, the First-Fit Algorithm and the First-Fit Decreasing Algorithm.
- The sorting algorithms. Make sure you know which of these algorithms you need to learn by heart.
- How to count the umber of comparisons and swaps in each pass and know the maximum number of passes that are required for a list of a given length.
- The different ways algorithms are presented and make sure you practice following unfamiliar algorithms.
- What is meant by efficiency of an algorithm.

Presenting and Implementing Algorithms

An algorithm is a well-defined, finite sequence of instructions to solve a problem. They can be communicated in various ways, including written English, pseudo code and flowcharts. Make sure you are experienced in all possible formats.



Example

a) What is the output of the algorithm when A = 84 and B = 660?

b) What does the algorithm achieve?

Solution

a)

A	84	72	12
B	660	84	72
Q	7	1	6
R1	72		12
R2		12	0

PRINT 12

b) It finds the highest common factor of A and B.

Bin Packing

These are examples of HEURISTIC algorithms. This means that none of these algorithms necessarily lead you to the best or optimal solution of the problem.

1. Full-Bin Algorithm

Look for combinations of boxes to fill bins. Pack these boxes. For the remainder, place the next box to be packed in the first available slot that can take that box.

Note – the full bin algorithm does not always lead to the same solution of the problem. In other words, two people could apply the full bin algorithm perfectly correctly and end up with their boxes packed differently.

2. First-Fit Algorithm

Taking the boxes in the order listed, place the next box to be packed in the first available slot that can take that box.

3. First-Fit Decreasing Algorithm

i) Re-order the boxes in order of decreasing size.

ii) Apply the First-Fit algorithm to this reordered list.

You should be able to form a judgement about the relative efficiency of these algorithms. The First-Fit Decreasing Algorithm requires a sort to be made before applying the First-Fit Algorithm so, in terms of computation, it requires more resources than the First-Fit Algorithm alone.

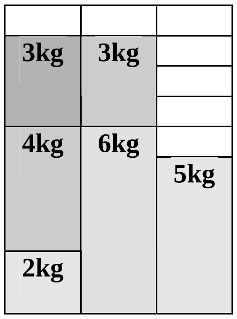
Example: Show how the following items are to be packed into boxes each of which has a capacity of 10Kg.

Item	A	B	C	D	E	F
Weight (kg)	2	4	6	3	3	5

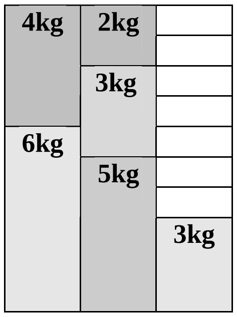
1. Full Bin

6+4=10, 5+3+2=10, 3 3 bins needed

2. First-Fit



3. First-Fit Decreasing



Notice that in this example the First-Fit Decreasing Algorithm gives the same result as the Full Bin Algorithm. This will not always be the case.

Sorting Algorithms

There are many sorting algorithms, so you must check carefully to see which, if any, you need to memorise for the examination.

Questions often ask about the relative efficiency of sorting algorithms by comparing the number of comparisons (c) and swaps that are made to sort the same list of numbers, as seen in this example:

List	1st pass	2nd pass	3rd pass
6	1	1	1
1	3	3	3
3	6	5	5
7	5	**6**	6
5	**7**	**7**	7
c	4	3	2
s	3	1	0

total number of comparisons: **9**

total number of swaps: **4**

Bubble Sort

First pass: the first number in the list is compared with the second and whichever is smaller assumes the first position. The second number is then compared with the third and the smaller is placed in the second position, and so on. At the end of the first pass, the largest number will be at the bottom. For the list of five numbers on the right, this involves 4 comparisons and 3 swaps.

Second pass: repeat first pass but exclude the last number (on the third pass the last two numbers are excluded and so on).

The list is repeatedly processed in this way until **no swaps take place in a pass**.

For a list of 5 numbers, the list will definitely be sorted after the 4th pass (why?), so this is the maximum number of passes. The maximum number of comparisons is 4+3+2+1=10 and the maximum number of swaps is 10. You should be able to generalise this to a list of *n* numbers.

list	1st pass	2nd pass	3rd pass	4th pass
6	**1**	**1**	**1**	**1**
1	**3**	**3**	**3**	**3**
3	6	**6**	**5**	**5**
7	**7**	5	**6**	**6**
5	5	**7**	**7**	**7**
c	4	2	1	0
s	1	1	1	0

total number of comparisons: **7**
total number of swaps: **3**

Quick Sort

Select a pivot – usually the middle item in the list

First pass: numbers are sorted into two sub lists, those smaller than the pivot element and those greater than the pivot element. The pivot element is now fixed in its correct position I the list.

Second pass: choose a pivot element in each of the two sub lists and repeat the sorting procedure.

Continue this process until all numbers are fixed and the list is sorted.

In this case the quick sort takes fewer comparisons and swaps than the bubble sort, though it does take one more pass to achieve the sort. It is worth noting that the relative efficiency of the different types of algorithm will vary depending on how "mixed up" the list is.

SUMMARY SHEET – DECISION MATHS

CRITICAL PATH ANALYSIS

The main ideas are covered in

AQA	Edexcel	MEI	OCR
D2	D1	D1	D2

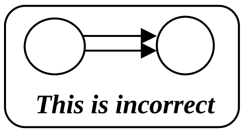
The main ideas in this topic are

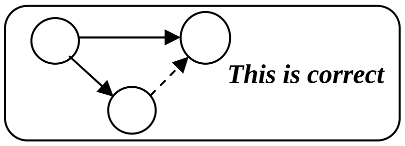
- Drawing Activity or Precedence Networks
- Performing Forward and Backward Passes and Identifying Critical Activities
- Drawing Cascade Charts and Resource Levelling

Before the exam you should know

- How to draw precedence networks. as you possibly can.
- When you need to use dummy activities.
- How to perform forward and backward passes on a precedence network to calculate early and late start times.
- How to find the critical activities.
- How to calculate the various types of float.
- How to draw a cascade chart and construct a resource histogram.
- Where resource levelling is required and how to make effective use of float to improve efficiency.
- What is meant by crashing a network.

Terminology
An activity is a task which needs to be done and takes an amount of time/resources to complete.
Precedence tables show the activities that need to be done together with their duration and their immediate predecessors.
Precedence networks show the sequence of the activities**.** The network must have one start node and one end node.
An **event** is the start/finish of one or more activities.
Dummy activities are used to keep the correct logic and to ensure each activity is **uniquely** defined by (i, j) where i is its starting event and j is the finishing event.



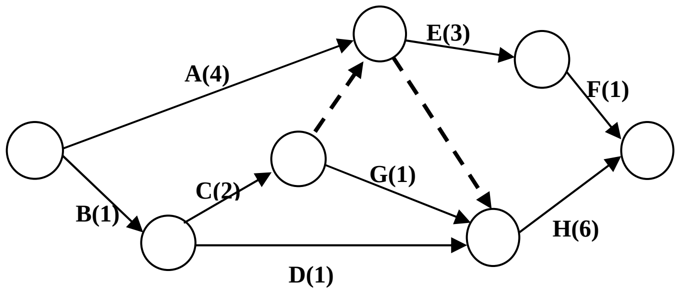


It can be a good idea to do an initial sketch as it's often possible to make your diagram clearer by repositioning activities to avoid them crossing over one another.
Forward pass establishes the earliest times that events can happen.
Backward pass establishes the latest time that an event can happen.
Critical activites are those whose timing is critical if the project is to be completed in the minimum time. The critical activities will form a path through the network
Float is the amount of time by which an activity can be delayed or extended.
Independent float does not affect other activities.
Interfering float is shared between two or more activities.

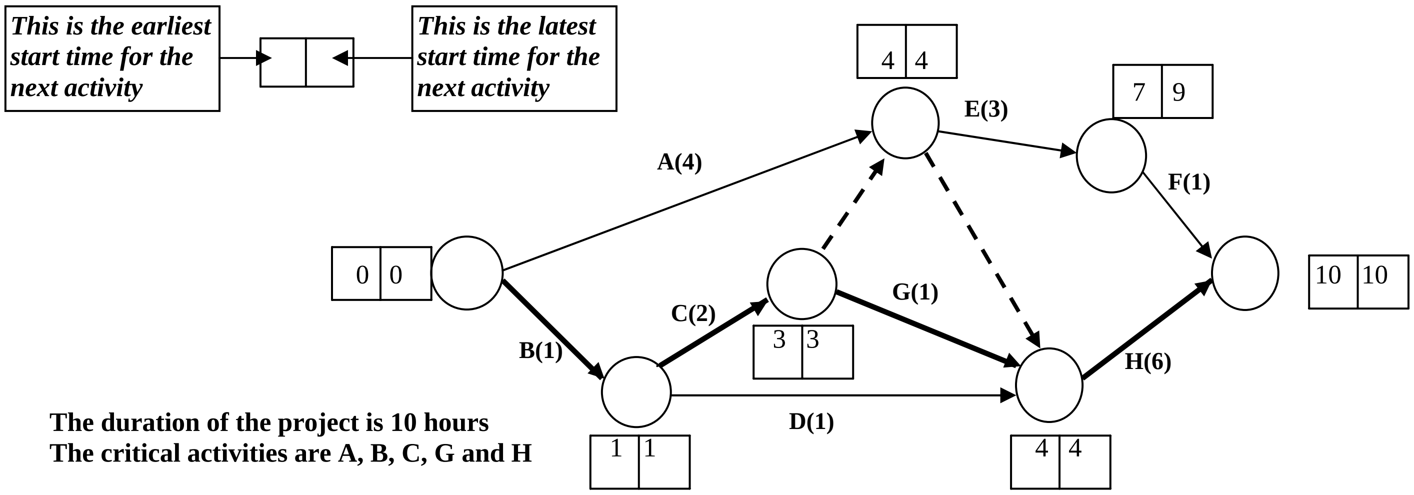
Example:
The table shows the activities involved in creating a small patio in a garden.

Activity Name	Task	Time (hrs)	Preceding Activities
A	Clear Garden	4	
B	Measure area	1	
C	Design Patio	2	B
D	Purchase fencing	1	B
E	Buy pots and plants	3	A,C
F	Plant all pots	1	E
G	Purchase paving	1	C
H	Construct Garden	6	A, D,G

The network for this precedence table



The forward and backward pass


This is the earliest start time for the next activity
This is the latest start time for the next activity
A(4)
B(1)
C(2)
D(1)
E(3)
F(1)
G(1)
H(6)
0 0
1 1
3 3
4 4
4 4
7 9
10 10

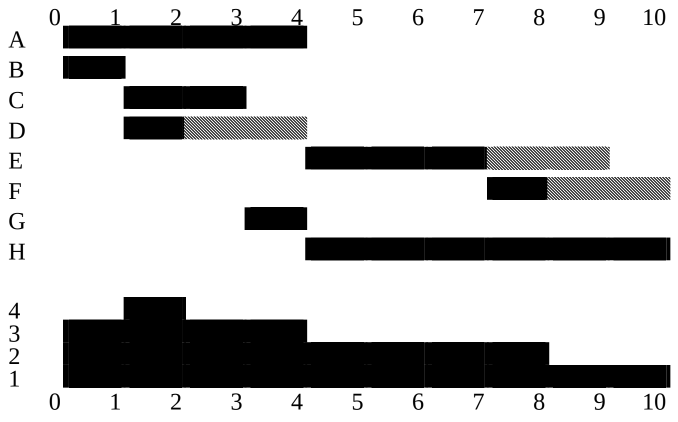
The duration of the project is 10 hours
The critical activities are A, B, C, G and H

Float

activity	float	type
D	2 hours	independent
E	2 hours	Interfering (with F)
F	2 hour	Interfering (with E)

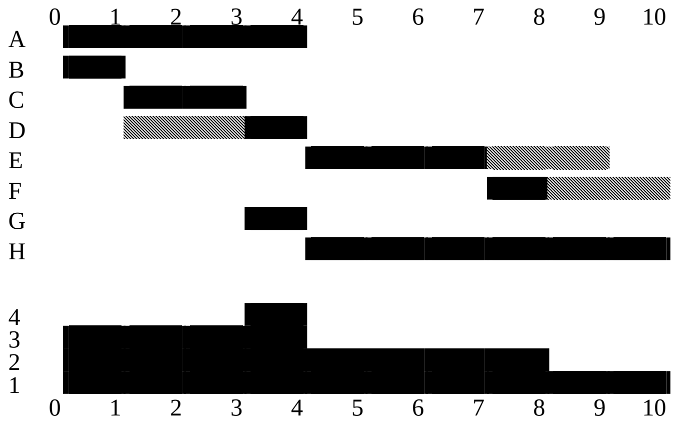
In this example there are two hours of float shared between activities E and F

Cascade Chart and Resources levelling



A Cascade Chart shows each activity set against a time line.
Float time is sometimes shown by using shading.
Dependencies are shown by vertical lines.
The cascade chart can be adjusted by using the float times to make use of resources more efficient.

If activity A needs two people and all the rest can be done by one person, then the resource histogram looks like this (note that 4 people are needed in the second hour).



If only three people are available for the first three hours, but a fourth friend can then come and help for an hour, we could move activity D within its float time to make this possible.

This would make the cascade chart look like this

The resource histogram would now look like this

SUMMARY SHEET – DECISION MATHS

Graph Theory

The main ideas are covered in

AQA	Edexcel	MEI	OCR
D1	D1	D1	D1

The main ideas in this topic are

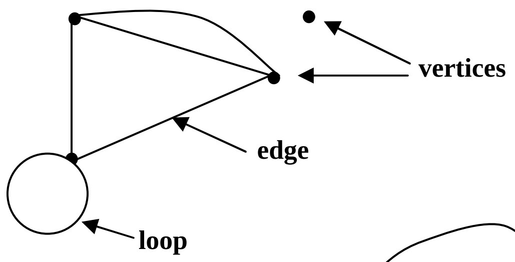
- The definition of a graph and the associated vocabulary.
- Mathematical modeling with graphs.

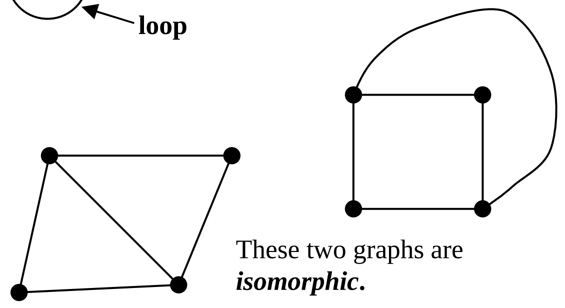
Terminology for Graph Theory

- **Graph** – collection of vertices & edges.
- **Vertex/Node –** the dots in a graph (usually where 2 or more edges meet, but not necessarily).
- **Edge/Arc** – a line between two vertices.
- **Tree** – a graph with no cycles.
- **Order (degree) of a vertex –** the number of edges starting or finishing at that vertex.
- **Simple graph** – a graph with no loops or multiple edges.
- **A path** – a route from one vertex to another which does not repeat any edge.
- **A cycle** – a route starting and finishing at the same vertex.
- **Connected graph** – a graph in which there is a route from each vertex to any other vertex (i.e. the graph is in one part).
- **Complete graph** – a simple graph in which every pair of vertices is connected by an edge.
- **Bipartite graph –** one in which the vertices are in two sets and each edge has a vertex from each set.
- **Planar graph –** one which can be drawn with no edges crossing.
- **Sub graph** – any set of edges & vertices taken from a graph is a sub-graph.
- **Hamiltonian cycle –** a cycle that visits every vertex of the graph.
- **Eulerian cycle** – a cycle that travels along every edge of the graph.
- **Eulerian graph** – a graph with no odd vertices.
- **Di-graph** – a graph in which the edges indicate direction.
- **Incidence matrix** – a matrix representing the edges in a graph.

Before the exam you should know:

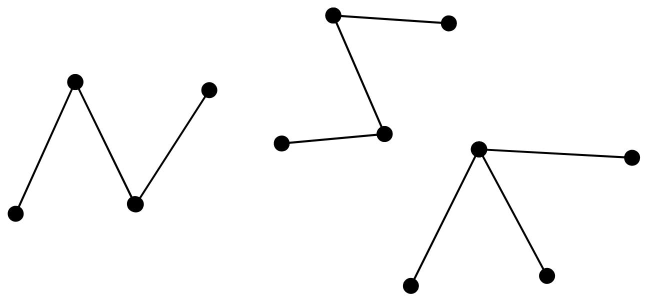
- The terms vertices (nodes), edges (arcs), digraphs, trees and paths.
- All the other vocabulary used to describe ideas in graph theory.
- How to draw a graph from an incidence matrix.
- How to model problems using graphs (e.g. Konigsberg Bridges).
- What is meant by a tree.
- How to recognise isomorphic graphs.
- What is meant by a Hamiltonian cycle.
- What is meant by an Euler cycle.



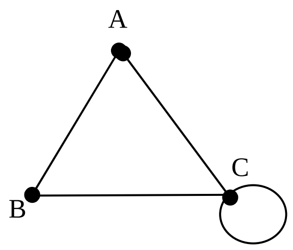


These two graphs are ***isomorphic.***

These diagrams all show trees of the graph above

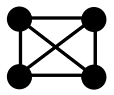


This shows a graph and its Incidence matrix.

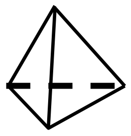


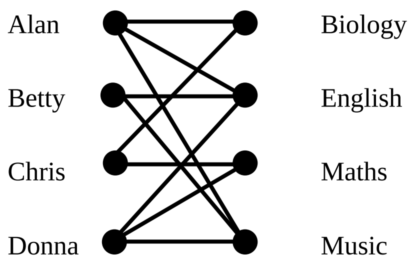
$$\begin{array}{c|c|ccc} & & & \text{To} & \\ & & A & B & C \\ \hline & A & - & 1 & 1 \\ \text{From} & B & 1 & - & 1 \\ & C & 1 & 1 & 2 \end{array}$$

Graphs can be used to represent many different things



This graph represents a tetrahedron





This ***bipartite graph*** shows which subjects four students study.

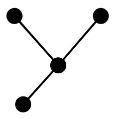
Example

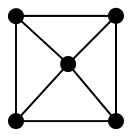
The table shows the number of vertices of degree 1, 2, 3 and 4 for three different graphs.
Draw an example of each of these graphs.

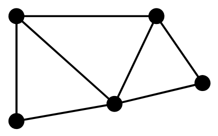
Order of vertex	**1**	**2**	**3**	**4**
Graph 1	3	0	1	0
Graph 2	0	0	4	1
Graph 3	0	2	2	1

solution

graph 1 graph 2 graph 3







Find the number of edges and the sum of the degrees of all the vertices of the graphs. What do you notice?

Graph 1: number of edges 3 sum of degrees of vertices 1+1+1+3 = 6
Graph 2: number of edges 8 sum of degrees of vertices 3+3+3+3+4=16
Graph 3: number of edges 7 sum of degrees of vertices 2+2+3+3+4 = 14

The sum of the degrees of the vertices is always twice the number of edges.

Also note that there are always an even number of odd vertices.

SUMMARY SHEET – DECISION MATHS

GRAPHICAL LINEAR PROGRAMMING

The main ideas are covered in

AQA	Edexcel	MEI	OCR
D1	D1	D1	D1

The main ideas in this chapter are

Formulating a problem as a linear programming problem, solving a Linear Programming Problem (maximisation and minimisation) and Integer Programming.

Before the exam you should:

- Practice formulating linear programming problems. This can often be the trickiest part of the problem. Remember to be consistent with units.
- Learn the terminology – the OBJECTIVE FUNCTION is what you have to maximise or minimise subject to a number of CONSTRAINTS.
- Make sure you are able to draw straight line graphs quickly from the constraints by considering where they cross the x and y axes.
- You must be able to find the solution to problems from the graph. Make sure you can draw graphs accurately.
- Remember to shade OUT the unacceptable region to keep the feasible region clear and easy to identify.
- You must be able to find correct solutions to problems where the answer must be an integer.

Formulating a problem as a Linear Programming Problem

First: identify the variables about which a decision is to be made. These are sometimes called the decision variables. For example if your problem is to decide how many chairs to make and how many tables to make to maximise profit, begin with a statement like – let *x* be the number of chairs and let *y* be the number of tables. If your problem is to work out how many grams of wheatgerm and how grams of oat flour there should be in a new food product to meet nutritional requirements and minimise cost then let *x* be the number of grams of wheatgerm and let *y* be the number of grams of oat flour.

Next: Decide what the objective function is (this is the value you are trying to maximise or minimise) and what the constraints are as inequalities involving *x* and *y*.

Be careful to use the same units consistently. For example it's possible that some distances appearing in a problem are given in metres and some are given in centimetres. Or some times they could be given in seconds with some given in minutes. Choose one type of units and convert everything into those units.

Example:
A clothing retailer needs to order at least 200 jackets to satisfy demand over the next sales period. He stocks two types of jacket which cost him £10 and £30 to purchase. He sells them at 20 pounds and 50 pounds respectively. He has 2700 pounds to spend on jackets.

The cheaper jackets are bulky and each need 20cm of hanging space. The expensive jackets need only 10cm each. He has 40m of hanging space for jackets.

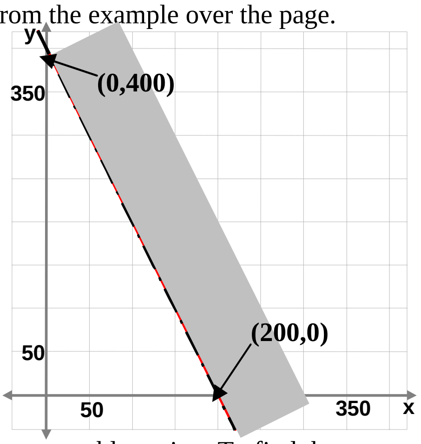
The retailer wishes to maximise profit. Assuming that all jackets will be sold, formulate a linear program, the solution of which will indicate how many jackets of each type should be ordered.

Formulation as a linear program
The decision is about how many of two types of jacket need to be ordered.
Let ***x* = number of cheaper jackets ordered**
Let ***y* = number of expensive jackets ordered**
The profit, *P*, given by selling all of these, is $P = 10x + 20y$, since the profit made on a cheaper jacket is 10 pounds and the profit made on an expensive one is 20 pounds.
The constraints are:
1. "needs to order at least 200" giving $x + y \geq 200$
2. "cost him 10 pounds and 30 pounds" and "has 2700 pounds to spend" giving $10x + 30y \leq 2700$
3. "20cm of hanging space" and "10cm" and "has 40m of hanging space" giving $0.2x + 0.1y \leq 40$

Solving a Linear Programming Problem

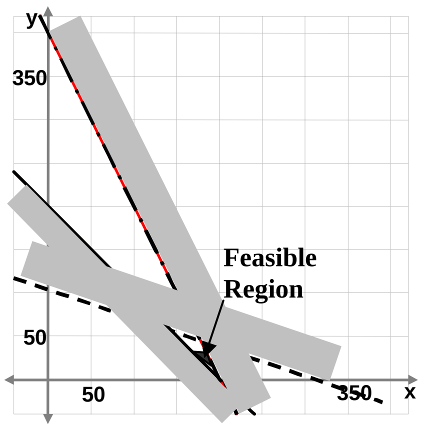
Draw a graph in which each constraint is represented by a line with shading. The unacceptable side of the line should be shaded. This leaves a "feasible region". The solution of the problem will be one of the vertices of the feasible region. These can be checked to find the best. We do this below for the example introduced over the page.

Drawing the line representing a constraint.
As an example, take the constraint
$0.2x + 0.1y \leq 40$ from the example over the page.
The initial aim is to draw the line $0.2x + 0.1y = 40$. We know this is a straight line so it's enough to find two points on the line and join them. When x = 0, y = 400 and when y = 0, x = 200. So the points (0, 400) and (200, 0) are on the line.



Then shade out, the unacceptable region. To find the unacceptable region just test a point to see if it satisfies the constraint or not. For example, in this case (10, 10) clearly satisfies the constraint and so is in the acceptable region.

The feasible region.
Once you have drawn all the constraints, the feasible region is the intersection of the acceptable regions for all of them.

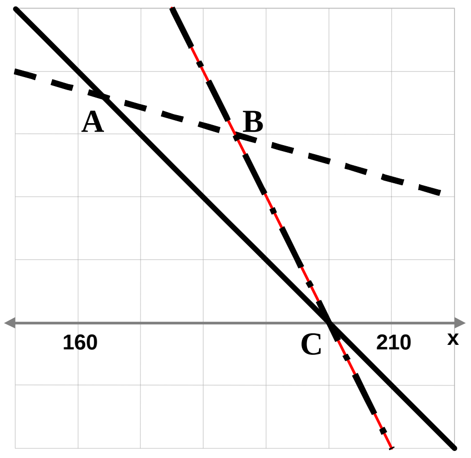


Constraint 1
$x + y \geq 200$

Constraint 2
$10x + 30y \leq 2700$

Constraint 3
$0.2x + 0.1y \leq 40$

Finding the solution
The solution of the problem will be at one of the vertices of the feasible region. You will need to solve simultaneous equations to find the co-ordinates of these vertices. Then each vertex must be checked to find the best. For example in the above we have a feasible region as in the diagram on the right. The coordinates of point A are found by solving $x + y = 200$ and $10x + 30y = 2700$ simultaneously. The solutions are x = 165 and y = 35. So the point is (165, 35) and the profit at that point is $P = 10x + 20y = 1650 + 700 = 2350$. Similarly it can be seen point B is (186, 28) giving a profit of 2420. Point C is (200, 0) giving a profit of 2000. So the best profit that can be made is by buying 165 cheap coats and 35 expensive coats.



Considering Gradients.
By calculating the gradients of each of the constraints and the gradient of the objective function, it's possible to predict in advance which vertex will give the optimal solution.

Minimisation problems are solved in exactly the same way. Just remember that this time you are looking for the vertex which makes the objective function the lowest.

Integer Programming

If the solution to the problem has to have integer values then points with integer value coordinates, close to the optimal point can be checked. This is likely to reveal the optimal solution but it is not guaranteed to. For example suppose the Objective Function is $2x + 3y$ and that this should be maximised. The optimal point may be (30.6, 40. 8) but do not assume that (30.40) will give the best solution; you must look at all the points with integer coordinates that are nearby: (31, 40), (30, 41), (30, 40) and (31, 41).

However (31, 41) and (31, 40) are not in the feasible region. You can check this by substituting in the values into the constraints. Of the two points nearby which are in the feasible region, namely (30, 41) and (30, 40), it can be seen that (30, 41) provides the best profit.

SUMMARY SHEET – DECISION MATHS 1

NETWORKS – Minimum spanning tree and shortest path

The main ideas are covered in

AQA	Edexcel	MEI	OCR
D1	D1	D1	D1

The main ideas in this topic are

- Appling Kruskal's and Prim's Algorithms to find the minimum spanning tree of a network.
- Applying Dijkstra's Algorithm to find the shortest (or least value path from one vertex to any other vertex in the network.

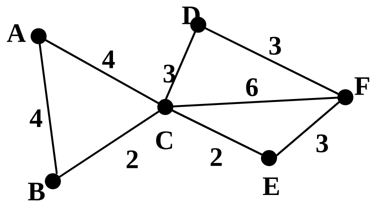
Before the exam you should know:

- How to show all the working clearly, there are more marks for the working than for getting the right answer.
- The distinction between Kruskal's and Prim's algorithms.
- How to apply Prim's algorithm to both a network and a table correctly.
- That Prim's and Kruskal's algorithms will usually give the same MST but often select the edges in a different order. Make sure you show sufficient working so that the examiner can see which algorithm you have used.
- How to work with networks or tables and be able to convert between the two.
- That you must always show all the working values as well as the permanent labels when using Dijkstra's algorithm.

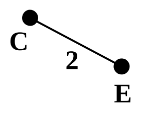
Minimum Spanning Tree

The minimum connector problem is to make a selection of the available edges so that any one vertex can be reached from any other, and the total length of the chosen edges is as small as possible. A connected set of edges with no loops is called a *tree* and the set which solves the minimum connector problem is the *minimum spanning tree* for the network.

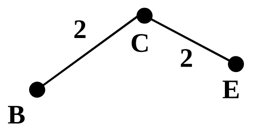
Kruskal's Algorithm



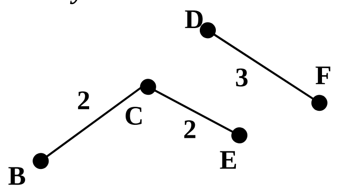
1. Choose the shortest edge (if there is more than one, choose any of the shortest)…



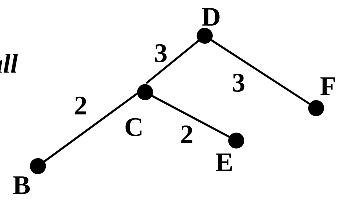
2. Choose the next shortest edge in the network (it doesn't have to be joined to the edges already…

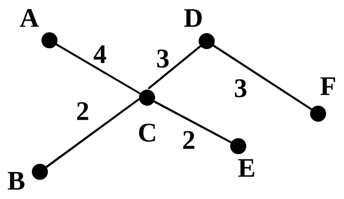


3. Choose the next shortest edge which does not create a cycle and add it…



4. Repeat step 3 until all the vertices are connected then stop.





Length of minimum spanning tree: 14

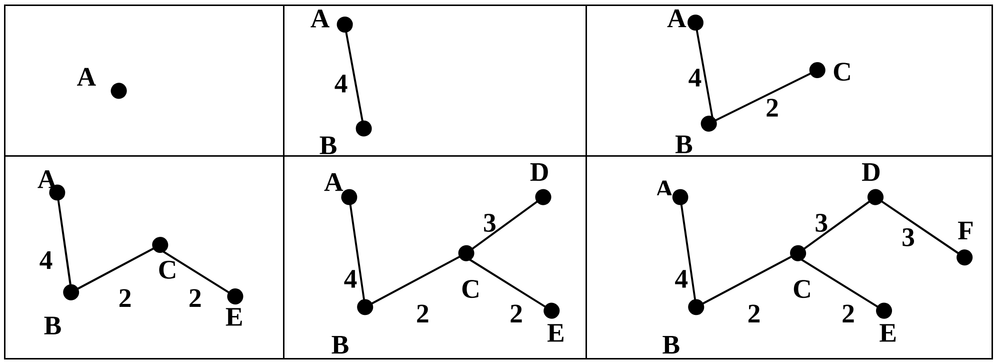
Prim's Algorithm on a network

1. Choose a vertex…

2. Choose the shortest edge from this vertex to any vertex connected directly to it…

3. Choose the nearest vertex not yet in the solution which is connected to any vertex which is in the solution and which does not create a cycle…

4. Repeat step 3 until all the vertices are connected then stop.



Prim's Algorithm on a Table

1. *Choose a column and cross out its row. Here D has been chosen. Delete row D.*
2. *Choose the smallest number in the column D and circle it. If there is a choice, choose either.*
3. *For the number you have just circled, cross out its row and put an arrow above its row at the top of the table.*
4. *Choose the smallest number not already crossed out from the arrowed columns and circle it.*
5. *For the number you have just circled, cross out its row*
6. *and put an arrow above it's row at the top of the table.*
7. *Continue till all vertices have been included in the tree.*

1

	A	B	C	D	E	F
A	-	4	4			
B	4	-	2			
C	4	2	-	3	2	6
~~D~~			~~3~~	~~-~~		~~3~~
E			2		-	3
F			6	3	3	-

2 1

	A	B	C	D	E	F
A	-	4	4			
B	4	-	2			
~~C~~	~~4~~	~~2~~	~~-~~	~~3~~	~~2~~	~~6~~
~~D~~			~~3~~	~~-~~		~~3~~
E			2		-	3
F			6	3	3	-

3 2 1

	A	B	C	D	E	F
A	-	4	4			
~~B~~	~~4~~	~~-~~	~~2~~			
~~C~~	~~4~~	~~2~~	~~-~~	~~3~~	~~2~~	~~6~~
~~D~~			~~3~~	~~-~~		~~3~~
E			2		-	3
F			6	3	3	-

3 2 1 4

	A	B	C	D	E	F
A	-	4	4			
~~B~~	~~4~~	~~-~~	~~2~~			
~~C~~	~~4~~	~~2~~	~~-~~	~~3~~	~~2~~	~~6~~
~~D~~			~~3~~	~~-~~		~~3~~
~~E~~			~~2~~		~~-~~	~~3~~
F			6	3	3	-

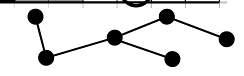
3 2 1 4 5

	A	B	C	D	E	F
A	-	4	4			
~~B~~	~~4~~	~~-~~	~~2~~			
~~C~~	~~4~~	~~2~~	~~-~~	~~3~~	~~2~~	~~6~~
~~D~~			~~3~~	~~-~~		~~3~~
~~E~~			~~2~~		~~-~~	~~3~~
~~F~~			~~6~~	~~3~~	~~3~~	~~-~~

6 3 2 1 4 5

	A	B	C	D	E	F
~~A~~	~~-~~	~~4~~	~~4~~			
~~B~~	~~4~~	~~-~~	~~2~~			
~~C~~	~~4~~	~~2~~	~~-~~	~~3~~	~~2~~	~~6~~
~~D~~			~~3~~	~~-~~		~~3~~
~~E~~			~~2~~		~~-~~	~~3~~
~~F~~			~~6~~	~~3~~	~~3~~	~~-~~

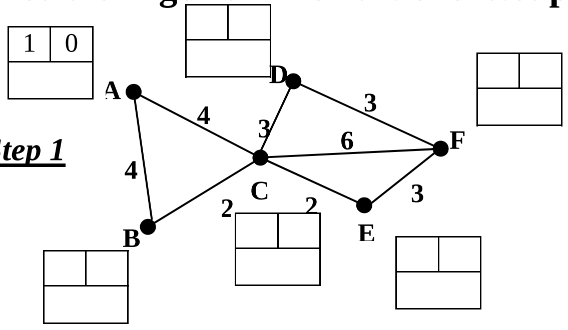
Length of minimum spanning tree 14



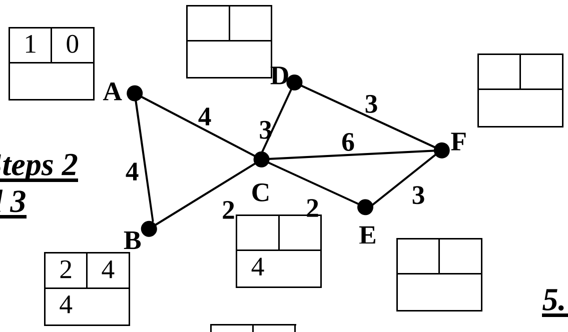
Dijkstra's Algorithm for the shortest path

1. *Label the start vertex with permanent label 0 and order label 1.*
2. *Assign temporary labels to all the vertices that can be reached directly from the start.*
3. *Select the vertex with the smallest temporary label and make its label permanent. Add the correct order label.*
4. *Put temporary labels on each vertex that can be reached directly from the vertex you have just made permanent. The temporary label must be equal to the sum of the permanent label and the direct distance from it. If there is an existing temporary label at a vertex, it should be replaced only if the new sum is smaller.*
5. *Repeat steps 3 and 4 until the finishing vertex has a permanent label.*
6. *To find the shortest paths(s), trace back from the end vertex to the start vertex. Write the route forwards and state the length.*

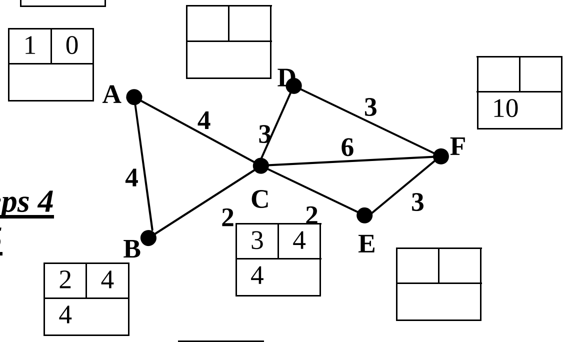
1. Step 1



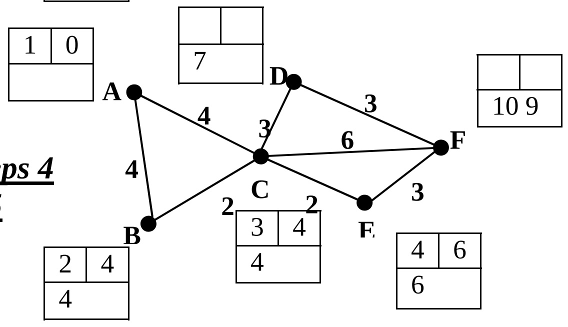
2. Steps 2 and 3



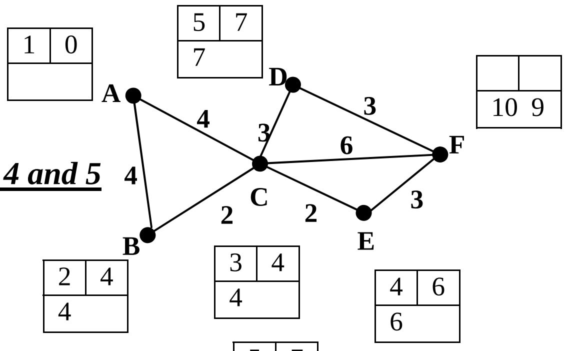
3. Steps 4 and 5



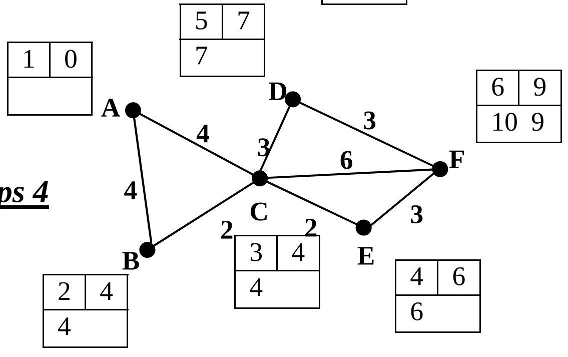
4. Steps 4 and 5



5. Steps 4 and 5



6. Steps 4 and 5



7. Step 6

Solution:
Shortest path ACEF
Length 9

REVISION SHEET – DECISION MATHS 1

SIMULATION

The main ideas are covered in

AQA	Edexcel	MEI	OCR
		D1	

The main ideas in this chapter are

Using random devices, especially random number generators to simulate events which are affected by chance.

Simulating queues.

Before the exam you should:

- Make sure you are able to allocate random numbers to events correctly, to generate events with the correct probabilities.
- Make sure you know how to correct rules, which result in some numbers being ignored.
- Make sure you understand service times, arrival times and inter-arrival times.
- Make sure you are happy with the idea of queueing disciplines.
- Be aware of how to allocate random numbers efficiently.
- Re-do Exercise 6D to ensure that you are fluent with exam style questions.

Simulating a Queue

First: Data must be gathered from observing the real queue and recording arrival times and service times.

Next: This data must be used to produce a frequency table of arrival times or inter-arrival times and service times. The wider the intervals, the less accurate will be the simulation.

Then: Use relative frequencies to allocate probabilities to the different arrival times, inter-arrival times or service times.

Then: Allocate random numbers to these probabilities.

Then: Run the simulation to see how the queue behaves. If the simulated queue behaves in a similar way to the real queue, the simulation may be adequate for its purpose. If it does not, the model may need improving, perhaps by shortening inter-arrival intervals or having more service intervals.

Example

The following data has been collected for a queue at a ticket office:

Inter-arrival time (seconds)	frequency
20	12
30	20
60	28

Service time (seconds)	frequency
30	30
50	15
70	15

Formulate rules to simulate these times using two-digit random numbers.

Inter-arrival time (seconds)	probability
20	$^{12}/_{60} = {}^{1}/_{5}$
30	$^{20}/_{60} = {}^{1}/_{3}$
60	$^{28}/_{60} = {}^{7}/_{15}$

Service time (seconds)	probability
30	$^{30}/_{60} = {}^{1}/_{2}$
50	$^{15}/_{60} = {}^{1}/_{4}$
70	$^{15}/_{60} = {}^{1}/_{4}$

The probabilities for the times are given by their relative frequencies. We use these probabilities to assign random numbers.

Allocating the random numbers

The lowest common denominator of $\frac{1}{5}$, $\frac{1}{3}$ and $\frac{7}{15}$ is 15. There are 100 two-digit random numbers, 00 – 99 inclusive. $\frac{100}{15} = 6$, remainder 10 and $6 \times 15 = 90$, so we use the ninety random numbers, 00 – 89. A probability of $\frac{1}{5}$ will require $\frac{90}{5} = 18$ numbers, 00 – 17. Check you can see how the others are arrived at.

Inter-arrival time (seconds)	Random numbers
20	00 - 17
30	18 - 47
60	48 - 89

To simulate the probabilities correctly, the 10 numbers 90 – 99 inclusive must be ignored if they come up.

It's much easier to allocate the random numbers for service times because the lowest common denominator of the probabilities divides exactly into 100, so all of the numbers 00 – 99 are allocated.

Service time (seconds)	Random numbers
30	00 - 49
50	50 - 74
70	75 - 99

Check that you can see where these came from.

Running the simulation

Use the random numbers below, reading from left to right and top to bottom, to simulate the arrival and service of the first 10 people in the queue.

Random numbers for arrivals

54, 63, 92, 67, 79,
55, 33, 21, 62, 88,
12, 45, 46, 28, 81

Random numbers for service

98, 49, 40, 22, 62,
61, 80, 77, 46, 19,
26, 23, 34, 09, 58

person	rand	Arrival time	Service start	rand	Service end	Wait time
1	54	60	60	98	130	0
2	63	120	130	49	160	10
3	*67	180	180	40	210	0
4	79	240	240	22	270	0
5	55	300	300	62	350	0
6	33	330	350	61	400	20
7	21	360	400	80	470	40
8	62	420	470	77	540	50
9	88	480	540	46	570	60
10	12	500	570	19	600	70

*92 had to be rejected – see rule in box on left.
Check that you can follow how this table was produced.

Mean queueing time

This is the mean of the times for which each customer queues. For the above example this is $\frac{10+20+40+50+60+70}{10} = 25$ seconds.

Mean length of queue

This is the total time spent queueing divided by the total elapsed time. For the above example this is $\frac{250}{600}$ = 0.417 people.

Server utilization

This is the percentage of elapsed time for which the server is busy. For the above example this is $\frac{500}{600} \times 100$ = 83.3%.

The 500 comes from the fact that the server is idle between 160 and 180 seconds, between 190 and 240 seconds and between 270 and 300 seconds, giving a total of 100 seconds idle time, so 500 seconds actually serving.

How do you improve the accuracy of a simulation?

If asked this in the exam, you should always say run more simulations

Other things that might make a simulation more reliable include:

- do longer runs of the simulation.
- have shorter arrival intervals and service times.
- collect more data on which to base probabilities.
- examine the assumptions made in the simulation and seeing if they can be removed by making the simulation more sophisticated e.g. in the above example it is assumed that as soon as one customer has finished being served, the next starts.

What is simulation used for?

- Simulation is used extensively in medical research, for example simulating the spread of highly contagious diseases.
- Often we need to be able to predict the effect of change without actually carrying out the change first, for example changing road layouts by building roundabouts, bypasses etc. is very expensive and disruptive, so it is a good idea to first simulate the affect of the proposed change, to help us decide whether it will work. To do this we first need to simulate the current situation accurately. Once this is done, we can adapt the simulation to incorporate the proposed changes and see whether they have the desired effect. We must always be on the lookout for unexpected consequences though.
- Weather forecasting is an example of simulation. It uses probabilities in a very complex weather model to try to predict the future.