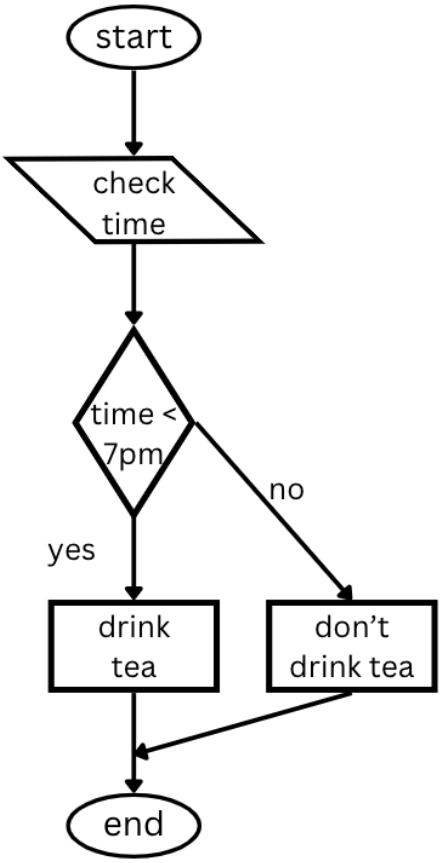
Algorithms and graph theory

Algorithms

An algorithm is a set of instructions which when followed will result in a task being carried out. Nowadays it is normally used in relation to social media where an algorithm takes an input of data about you and then follows steps to output a selection of content from that app that you might like.
Algorithms are most easily shown through flow charts or written out in pseudocode.

Flow charts

A flow chart is a collection of symbols and arrows which show an algorithm without actually writing much.
Oval - used at the start and end to clearly define where to begin and where you should stop.
Rectangle - contains text telling you to do something like add two numbers.
Diamond - this is a decision with a yes/no answer, you follow the arrow out of it with the correct label (yes/no).
Parallelogram - this takes an input such as a number or day of the week. It can also show output though.
Arrows - these are followed throughout to take you from one point to the next.



Here is an example, from the start you follow the arrow to the input where you input the current time. Then you follow the arrow to the decision diamond where you are asked if the time is less than (earlier than) 7pm. If it is you follow the yes arrow, otherwise you follow the no arrow. Depending on which arrow you followed you take the action described in the rectangle you get to. You then follow the arrow to the end. The arrows meet before the end to avoid having multiple arrows going into the end, just like we can only have one arrow come out of the start.

Pseudocode

Pseudocode is code written to be read out and followed by a human, not a computer. There are still a few conventions to be followed though. (These are questionable but used by the exam board so it is best to use them)

1. Start with the word start and end with the word stop.
2. No more than one statement per line.
3. Use the words “input”, “output”, “if” and “else” where they apply.
4. Use equations rather than writing out ‘increase n by 2’.

Here is an example from 2022:

1. Start
2. Input the value of n
3. Let a = 1
4. Let b = n - 2
5. Let c = b
6. Let a = a + 1
7. Let b = b - 1
8. Let c = c + (a x b) + (a - 1)
9. If b > 0 go to step 6
10. Output c
11. Stop

For pseudocode you may be asked to fill out a table to show how you follow this algorithm. The table will look like this and is completed for n = 5.

n	a	b	c	
5	1	3	3	Input 5, ‘a’ is given, ‘b’ is calculated and ‘c’ is ‘b’
	2	2	8	New line to redeclare ‘a’
	3	1	13	‘b’ is greater than 0 so now redeclaring ‘a’ again
	4	0	16	‘a’ is increased, ‘b’ is decreased and ‘c’ is calculated

‘b’ isn’t greater than 0 so ‘c’ is outputted and it is finished.
Note that we don’t have to put anything else in the ‘n’ column as it never changes.

Order of an algorithm

Each time a comparison or operation occurs it will take the computer some time, we can often find the number of comparisons for an algorithm in terms of the inputs given. For example, for the pseudocode above each of ‘a’, ‘b’ and ’c’ are declared at the start and then the values are changed 2 less times than the value of ‘n’ (the input). Therefore the number of comparisons is equal to 3 + 3 x (n - 2) or 3 x (n - 1) or 3n - 3.
Given the highest power of ‘n’ is 1 the order of the algorithm is 1, if the expression for the number of comparisons was a quadratic then the order would be 2 as there would be an ‘n’ to the power 2 involved.

Bin packing and sorting

One problem algorithms try to solve is bin packing. Bin packing is when you have a list of numbers which you need to sort into the minimum number of groups without the sum of a group exceeding a certain number. This is commonly visualised in terms of packing bins.

For example:
Each bin can bold a maximum of 10kg
You have items of weights (in kg): 2, 5, 4, 7, 1, 3 and 8

There are three algorithms to solve this: first-fit, first-fit decreasing and full-bin.

First-fit

Going from left to right in the list you put each item in the first available bin.
The first item goes in the first bin, the second item also fits in the first bin. There is now 7kg in the first bin so the 4kg item has to go in the second bin, the 7kg item can't go in the first or second so it goes in the third bin, the 1kg item next fits in the first bin, then the 3kg in the second bin and finally the 8kg in the fourth bin.

Bin1: 2, 5, 1
Bin2: 4, 3
Bin3: 7
Bin4: 8

First-fit decreasing

This is the same as first-fit but the items are sorted into descending order beforehand. Now the 8kg item is first so goes in the first bin. Then the 7kg in the second bin, the 5kg in the third bin, the 4kg in the third bin, the 3kg in the second bin and the 2kg in the first bin. The 1kg then fills the third bin.

Bin1: 8, 2
Bin2: 7, 3
Bin3: 5, 4, 1

Full-bin

Here, the items are grouped into 10kg groups before being put in. The 8kg and 2kg form a 10kg group, the 7kg and the 3kg form a 10kg group and the 5, 4 and 1 kgs form a 10kg group.

Bin1: 8, 2
Bin2: 7, 3
Bin3: 5, 4, 1

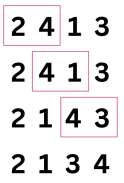
In this case either of full-bin or first-fit decreasing is optimal as it is impossible to do in 2 bins as the sum of the items is 30kg which can't fit into 2 bins of 10kg capacity. This isn't always the case though.

We may also use algorithms to sort a list into ascending or descending or alphabetical order. This can be done with bubble sort or quick sort.

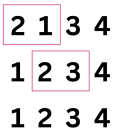
Bubble sort

Bubble sort repeatedly steps through the list to be sorted, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

For example for the list 2, 4, 1, 3 into ascending order:

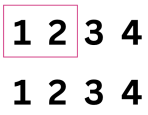


The 2 is less than the 4 so isn't swapped, the 4 is greater than the one so is swapped, the 4 is greater than the 3 so is swapped. That is the first pass. It is guaranteed that the final item is now correct.



The 2 is greater than the one so they are swapped, the 2 is less than the 3 so they aren't swapped. Now the last two are guaranteed to be correct. We still continue as it is possible that if in a different order at the start the 1 and 2 would need swapping.

The 1 is less than the 1 so they aren't swapped, now they are all in order.



The number of comparisons in bubble sort for a list of 'n' numbers is (n - 1) + (n - 2) + (n - 3) + … + 3 + 2 + 1 which simplifies to n x (n - 1) / 2. For n = 4 this gives us 6 comparisons which is correct as it is equal to the number of pink boxes above. Since the number of comparisons can be written as $0.5n^2 - 0.5n$ bubble sort is a second order algorithm as the highest power is 2.

Quicksort

For quicksort you first choose the middle (or right of middle) value as a pivot of the list and compare it to all the others to see which ones are less than it and which are greater than it. You then order them equivalently. Then repeat this for both sides of the original pivot and then repeat for each sides of those pivots until it is definitely sorted.

For example to get the list 30, 12, 5, 2, 23, 18, 36, 10, 15 and 24 into descending order:

There are an even number of items so choose the right of the middle as the first pivot (6th value).

30	12	5	2	23	18	36	10	15	24
30	23	36	24	18	12	5	2	10	15

Then choose a pivot for the block of white on each side, on the left the 36 is chosen as right of middle and on the right the 2 is chosen as it is in the middle.

30	23	36	24	18	12	5	2	10	15
36	30	23	24	18	12	5	10	15	2

For the first block of white the 23 is chosen as the midpoint and for the right one we can choose the 10 as the pivot as being right of middle.

36	30	23	24	18	12	5	10	15	2
36	30	24	23	18	12	15	10	5	2

For the left block of white the 24 is chosen as pivot as being right of middle. For the second block of white the 15 is chosen and for the third the 5 is chosen.

36	30	24	23	18	12	15	10	5	2
36	30	24	23	18	15	12	10	5	2

The 30 and 12 are the new pivots as they are the only ones left.

36	30	24	23	18	15	12	10	5	2
36	30	24	23	18	15	12	10	5	2

Now all values have been chosen as pivots so we have sorted the list into descending order.

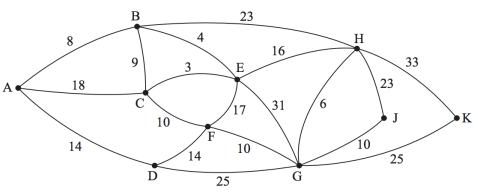
36	30	24	23	18	15	12	10	5	2

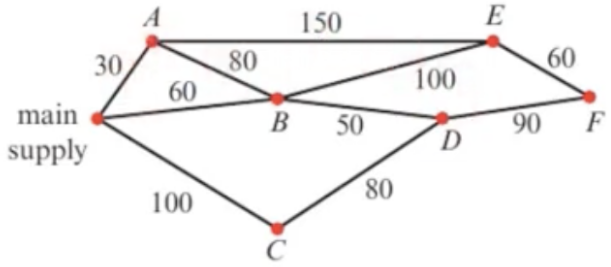
In an exam it looks like the image below. Bold are pivots and underlined are past pivots, you can change this notation just make it clear with a key.

30	12	5	2	23	**18**	36	10	15	24
30	23	**36**	24	<u>18</u>	12	5	**2**	10	15
<u>36</u>	30	**23**	24	<u>18</u>	12	5	**10**	15	<u>2</u>
<u>36</u>	30	**24**	<u>23</u>	<u>18</u>	12	**15**	<u>10</u>	5	<u>2</u>
<u>36</u>	30	<u>24</u>	<u>23</u>	<u>18</u>	<u>15</u>	12	<u>10</u>	5	<u>2</u>

Graph theory

In this context a graph is a connection of nodes (points) and arcs (connecting lines) which show connections. Arcs can also have numbers (weights) connected to them which can add more meaning to a graph. Here there are labelled nodes with arcs connecting them with weights beside the arc. This is called a weighted graph or network (no weights makes it just a graph).

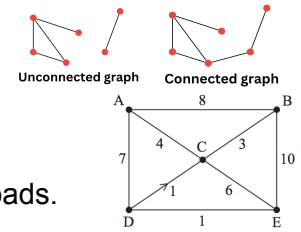




For this network the nodes represent places and the arcs the connection of pipes between them. The weights represent the distance of each pipe between places.

Key terms

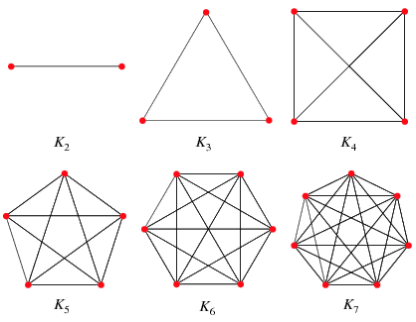
1. Points are called nodes or vertices.
2. Lines are called arcs or edges.
3. A subgraph is a graph which fits within a bigger graph. If I took A, B and E from the example above that could form a subgraph of that example.
4. The degree, valency or order of a node is how many arcs attach to it. For example the valency of the main supply is 3.
5. A route from one node to another like B, D, C, D is a walk.
6. A trail is a walk where no edge is travelled more than once.
7. A path is a walk where no vertex is visited more than once.
8. A cycle is a walk where the start node is the same as the finish node and no other vertex is used more than once.
9. A hamiltonian cycle is a cycle which includes all the nodes such as A, main supply, C, D, F, E, B, A.
10. A connected graph has all nodes connected.
11. A loop is an arc that starts and ends at the same vertex.
12. A simple graph has no loops and no more than one arc between two nodes.
13. Directed graphs have arcs with arrows which serve like one way roads.



In an undirected graph the sum of the valencies is equal to twice the number of arcs and as a result there is an even number of nodes with an odd degree, or none at all. This is called Euler's handshaking lemma

Types of graph

1. Tree - A graph with no cycles or loops.
2. Spanning tree - A subgraph that is a tree containing all the vertices of the original graph.
3. Complete graphs have every node connected to every other node. They are shown and named on the right.
4. Isomorphic graphs are the same graph drawn in different ways. (same nodes and arcs but drawn in different positions on a page)
5. A Eulerian graph has a cycle which travels every arc exactly once.

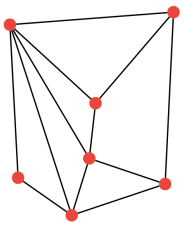


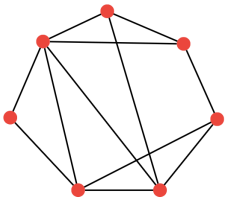
6. A semi-Eulerian graph has a Trail which travels every arc exactly once and the start is different from the end (otherwise it would be Eulerian)
7. A planar graph can be drawn without any arcs overlapping.

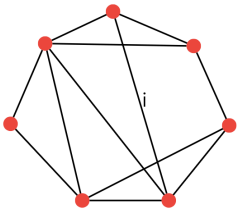
The planarity algorithm

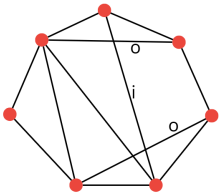
To find out if a graph is planar:

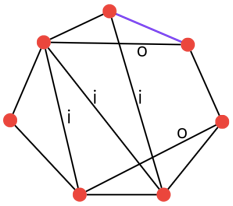
1. Find a hamiltonian cycle and draw the graph with the nodes in a circle.
2. Label one of the internal arcs ‘i’
3. Any arcs that cross that arc label ‘o’
4. Any arcs crossing an ‘o’ label with an ‘i’
5. Continue until all arcs are labelled, if an arc doesn’t cross any others label it ‘i’
6. If two ‘i’ arcs cross or two ‘o’ arcs cross it isn’t planar.
7. If it is planar, redraw it with the ‘i’ arcs inside the hamiltonian cycle and ‘o’ arcs outside it.

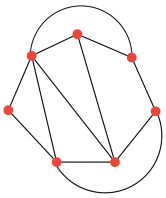
1. 

2. 

3. 

4. 

5. 

7. 

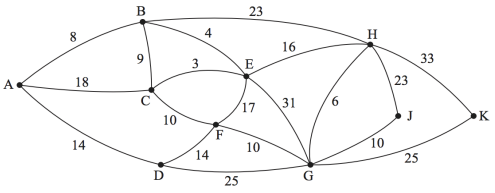
Algorithms on graphs

Minimum spanning tree

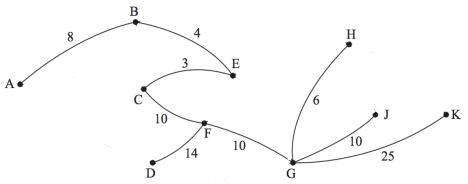
A minimum spanning tree (minimum connector) is the subgraph with the lowest total weight containing all the vertices of the original graph. There are two ways to find this: prim’s and kruskal’s.

Prim’s algorithm

1. Choose a random node to start from.
2. Join this up to the closest node (smallest weight away).
3. Continue step 2 connecting unconnected nodes to any already connected ones. If two nodes are equal weight away choose between them randomly.
4. Do this until all nodes are connected.



Becomes:



You need to be able to do this through a matrix as well.

1. Convert the network into a matrix format:
    a. Given the nodes already having letters or names, have a grid with a column for each node and a row for each node as shown for a graph with 4 nodes:

	A	B	C	D
A	-			
B		-		
C			-	
D				-

    b. For the first row fill in the weight to get from A to B, A to C and A to D. If a direct link isn't there, put a -.

	A	B	C	D
A	-	11	-	-
B		-		
C			-	
D				-

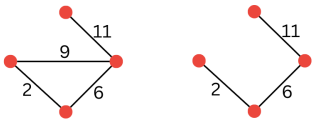
    c. Repeat for all rows.

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

2. Label with a 1 the column for the node we randomly selected to start with and cross out that node's row.
3. Circle the smallest value in that column and read across to see which node it links to.
4. Label that node's column with a 2 and cross out its row.

5. Look down all labelled columns and circle the smallest value in that column and read across to see which node it links to.
6. Repeat steps 4 and 5 until all rows are crossed out.

In this very simple example the left image becomes the right image.



Kruskal's algorithm

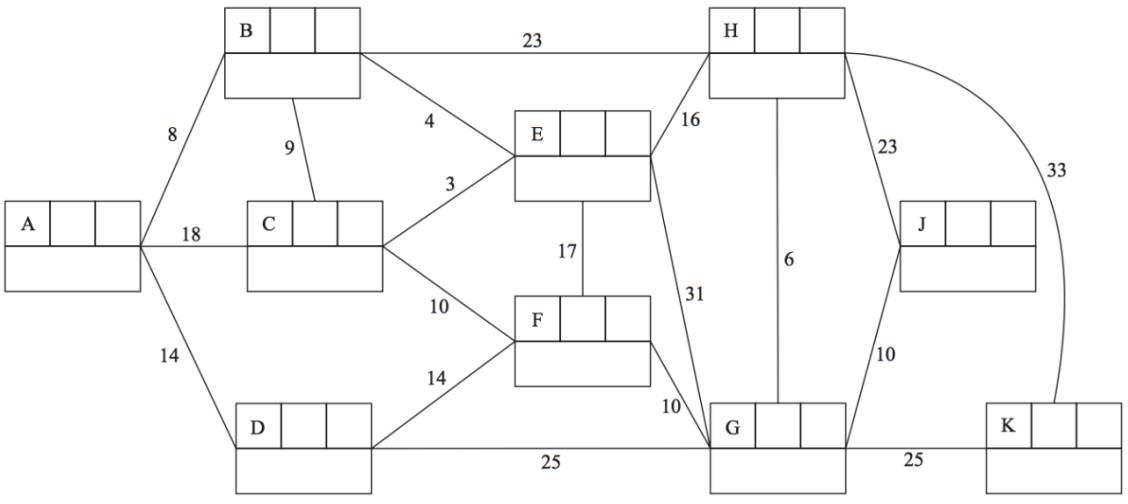
1. Order a list of the weights in increasing size.
2. Choose the arc with the smallest weight to start the minimum spanning tree.
3. Add the next smallest weighted arc unless it will form a cycle.
4. Repeat step 3 until all nodes are connected.

Finding the shortest path

It is often useful to find the quickest way (smallest weight) to get between two nodes. For this we can use one of two algorithms:

Dijkstra's algorithm

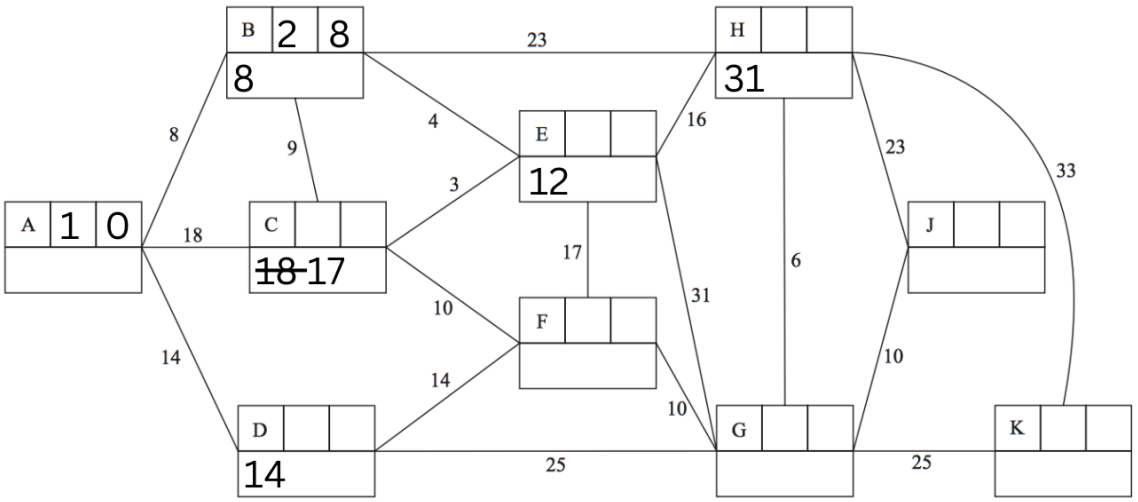
This algorithm builds outward from one starting node to find the shortest path to all other nodes. It is normally carried out on a specially drawn network with nodes replaced with spaces for workings as shown.



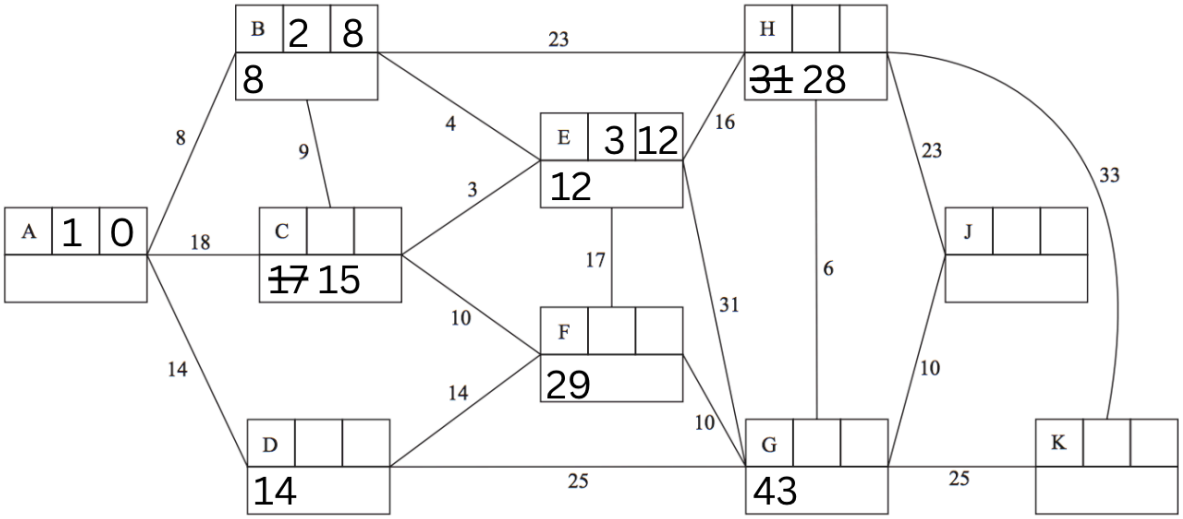
Key:

Vertex	Order of labelling	Final value
Working value		

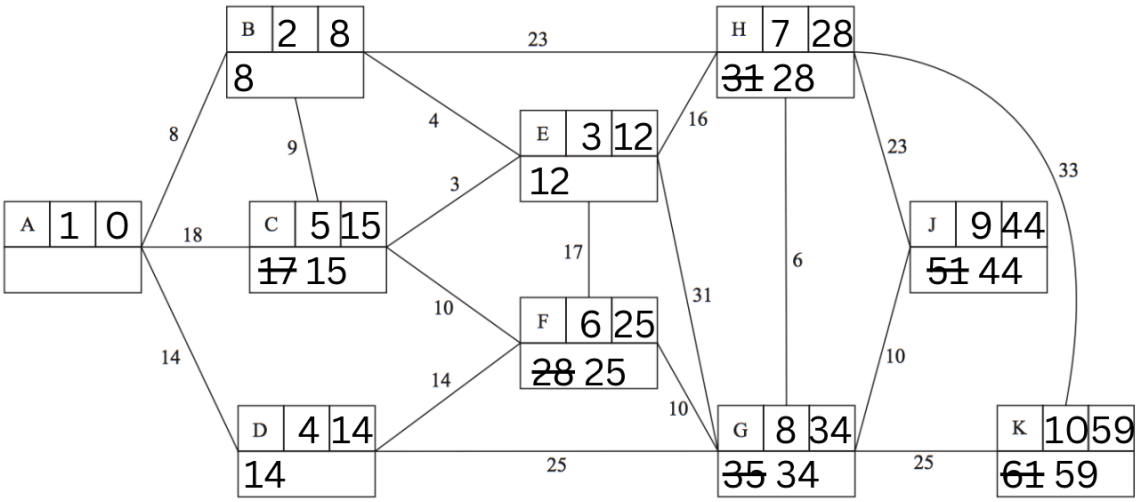
1. For the starting node put an order of labelling of 1 as it is the first labelled and put a final value of 0 as it takes nothing to get there.
2. For all nodes connected directly to that one, put a working value of the node it is connecting to's final value + the weight of the arc to get there.
3. For the node with the lowest working value, put its final value as its working value and write in its order of labelling.
4. For all nodes with arcs linked to the last labelled node, put a new working value of the last labelled node's final value + the weight of the arc between them only if it is lower than the previous working value.



5. Then find the new lowest working value for an unlabeled node and make it a final value. Also write in the order of labelling for this node and work out to nodes around it.



6. Repeat step 5 until all nodes have final values. These final values are the distances to each node from the starting node (A).



Key:

Vertex	Order of labelling	Final value
Working value		

Floyd's algorithm

This algorithm also finds the minimum sum of weights to get from any one node to any of the others. However it does it through matrix format.

1. Convert the network into a matrix like shown previously but if a path between nodes is unavailable put infinity unless it would connect a node to itself. Also setup a routes table of the correct size as shown on the right.

	A	B	C	D
A	-	2	∞	12
B	2	-	8	7
C	∞	8	-	∞
D	12	∞	∞	-

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

2. Starting with 'A' as the pivot, highlight its row and column. If any number is greater than the sum of the highlighted value in the same row as it and the highlighted number in the same column as it, change it to the sum of the two highlighted values and change the corresponding part of the route matrix to an 'A'.

	A	B	C	D
A	-	2	∞	12
B	2	-	8	7
C	∞	8	-	∞
D	12	14	∞	-

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

3. Repeat with the ‘B’ column.

	A	B	C	D
A	-	2	10	9
B	2	-	8	7
C	10	8	-	15
D	12	14	24	-

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

4. Repeat with the other columns one at a time.

	A	B	C	D
A	-	2	10	9
B	2	-	8	7
C	10	8	-	15
D	12	14	24	-

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

	A	B	C	D
A	-	2	10	9
B	2	-	8	7
C	10	8	-	15
D	12	14	24	-

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

Final:

	A	B	C	D
A	-	2	10	9
B	2	-	8	7
C	10	8	-	15
D	12	14	24	-

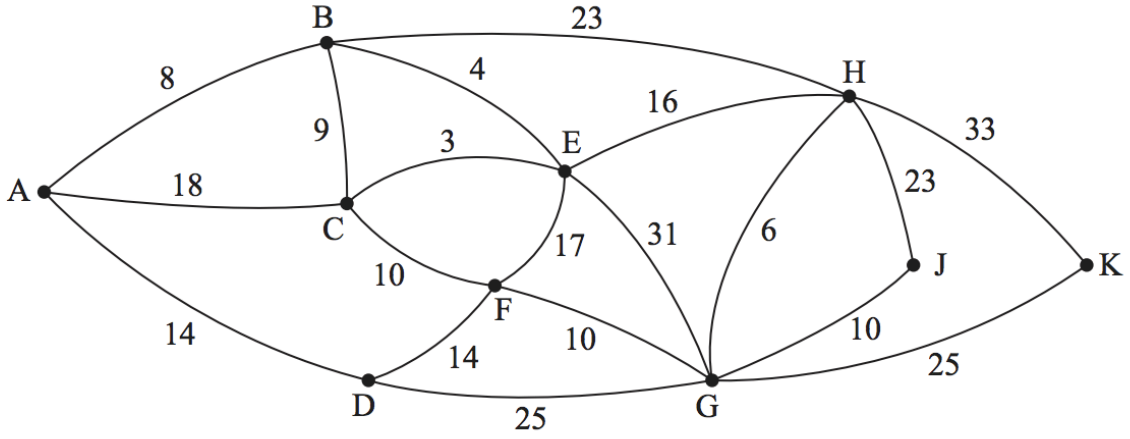
	A	B	C	D
A	A	B	B	B
B	A	B	C	D
C	B	B	C	B
D	A	A	B	D

Algorithms on graphs 2!

Route inspection algorithm

The goal of route inspection is, if the weights represent distances, to find the shortest walk that travels all arcs and ends at the vertex it started at.
The lower bound for solving this is a value that must be equal to or less than the answer. For this problem it is the sum of all the weights in a network as in an ideal solution they are all travelled once and you end up where you started.
As a worst case scenario you will need to backtrack every move you make so the upper bound will be twice the sum of the weights in the network.

When solving this problem you need to consider the order of each node involved. This is as if they are odd you will need to traverse one of its arcs multiple times as you will go to it the same number of times as you leave it.



In this example the odd degree nodes are:
A, D, E and H
We now need to consider pairings of nodes which will give us as little extra distance to cover as possible:
A->D and E->H = 14 + 16 = 30
A->E and D->H = 12 + 30 = 42
A->H and D->E = 28 + 26 = 54
So AD and EH gives us the smallest distance to travel twice, the distance needed to be travelled is therefore the sum of all the weights plus 30 which is 329.

If we were to start at A and end at F then we would want these to have an odd order as to start at A we will leave it one more time than we visit it and the opposite for F. That would make our odd nodes D, E, F and H. We would then compare different pairs of these as before.

The travelling salesman

The classical travelling salesman problem focuses on finding the shortest possible route for a salesman to visit each city in a list **once** and return to the starting city. We represent this on a network by having the nodes as cities and the time to travel between them the weights on the arcs.
The practical version of the problem changes the goal so that each city has to be visited **at least** once.

The classical travelling salesman on a complete graph satisfying the triangle inequality is a version of the classical problem done on a complete graph so that every city is connected to every other city and the triangle inequality means the time to get from A to B can't be greater than the time to get from A to C to B.

For the travelling salesman the upper bound as a worst case scenario is when you get to all the vertices and then have to return the way you came. This is twice the sum of the weights in the minimum spanning tree.
For the lower bound:

1. Remove a vertex and find a minimum spanning tree for the rest of the nodes
2. Find the total weight for the spanning tree then add on the two smallest weighted arcs that connected the removed vertex.
3. You may have to repeat this for all nodes and take the smallest value but more often you are told which node to remove.

Nearest neighbour algorithm

This is a way of approximating a solution to the travelling salesman problem on a complete graph, if the graph is incomplete use floyd's algorithm.

1. Choose a random city to start at and note it down.
2. Visit the closest unvisited city and add it to the list.
3. Continue step 2 until all cities are visited
4. Return to the start node to complete the cycle.

By summing up the weights of the arcs taken you can find a reasonably good solution to the travelling salesman. It isn't always optimal though.

Critical path analysis

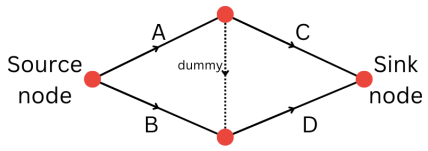
Modelling a project

Normally in life projects are able to be split up into individual tasks that will need completing. We call these activities and can represent them in an activity network. This is able to be constructed from an activity precedence table such as below.

Activity	Immediately preceding activities
A	
B	
C	A
D	A, B

Here A and B can start immediately but C can only be started after A is finished and D can only be started after A and B have finished.

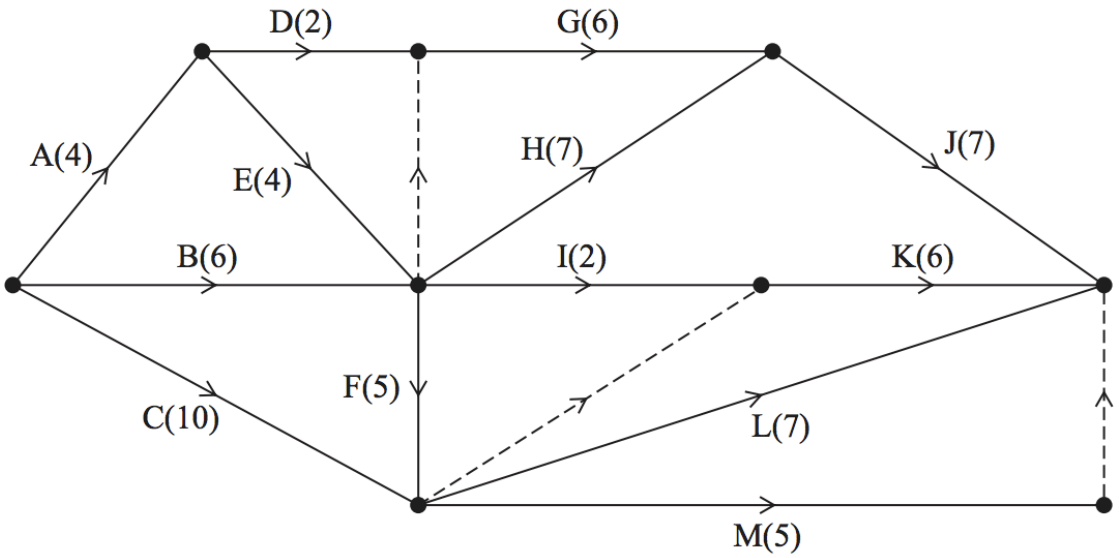
On an activity network the activities are represented by arcs and the nodes are only points of transition between them.From the precedence table above we can draw the following activity network:



You start at the source node and can do activities A and B, once A is done C and the dummy activity can start and once B and the dummy activity are done D can start. Once C and D are done the project is done. The dummy activity is used so that A can be used independently of B as well as with it. Here it also avoids A starting and finishing at the same nodes as B. Dummy activities have no duration and are used only for drawing the activity network.

Precedence table from network

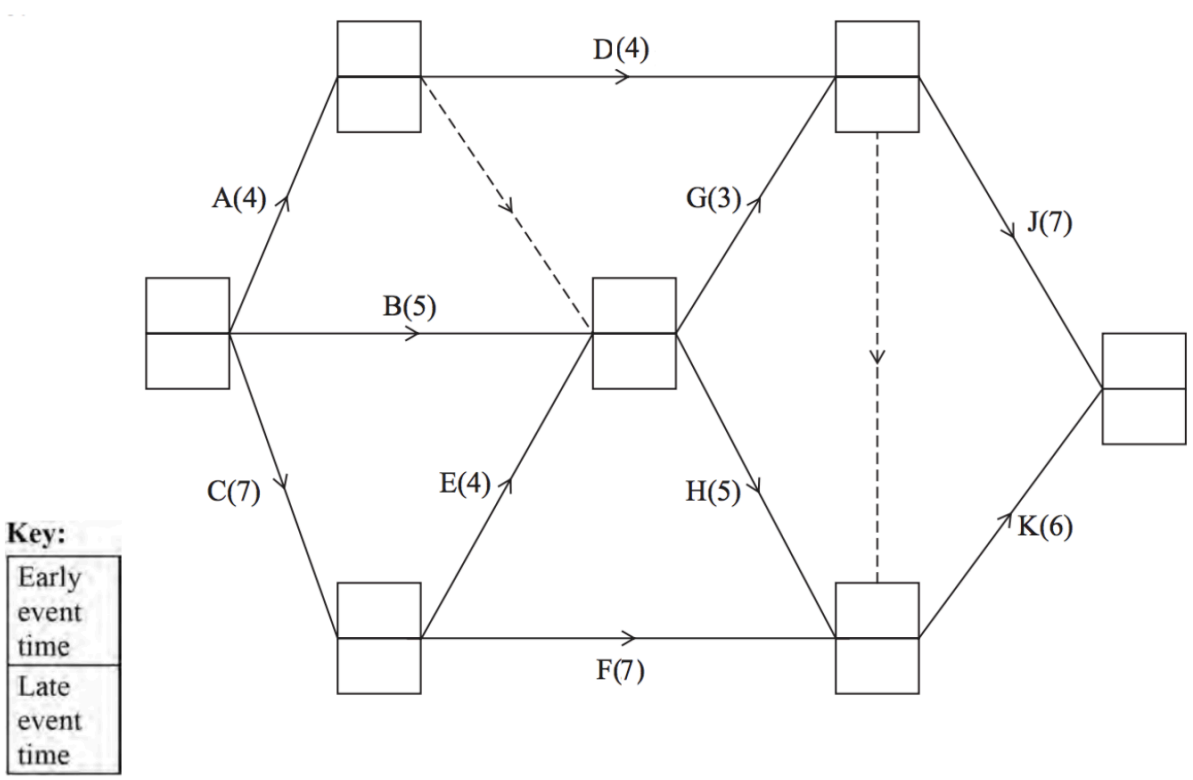
You also need to form an activity precedence table from a given activity network. On the activity network below the time taken to complete each event is also shown in brackets next to each activity. Note that in the precedence table only immediately preceding activities are shown, not all that went before it.



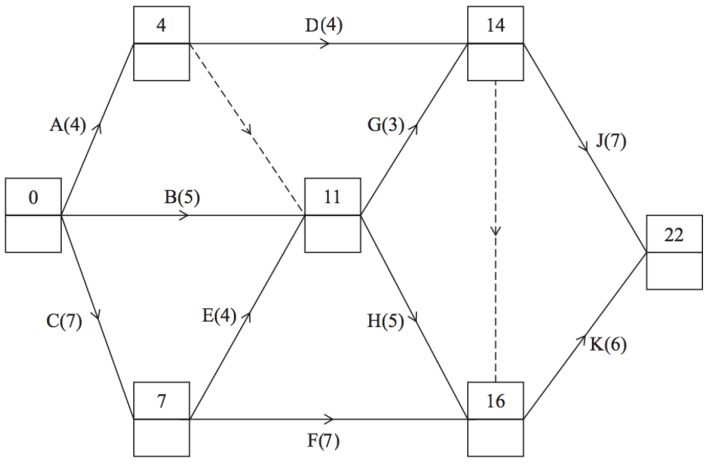
Activity	Immediately preceding activities	
A		This starts at the sink node with nothing before it
B		This starts at the sink node with nothing before it
C		This starts at the sink node with nothing before it
D	A	A is the only activity going into the node
E	A	A is the only activity going into the node
F	B, E	Both B and E enter it's start node
G	D, E, B	Dummy activity relies on E and B
H	B, E	Both B and E enter it's start node
I	B, E	Both B and E enter it's start node
J	G, H	Both G and H enter it's start node
K	I, F, C	Dummy activity relies on F and C
L	F, C	Both C and F enter it's start node
M	F, C	Both C and F enter it's start node

Early and late event times

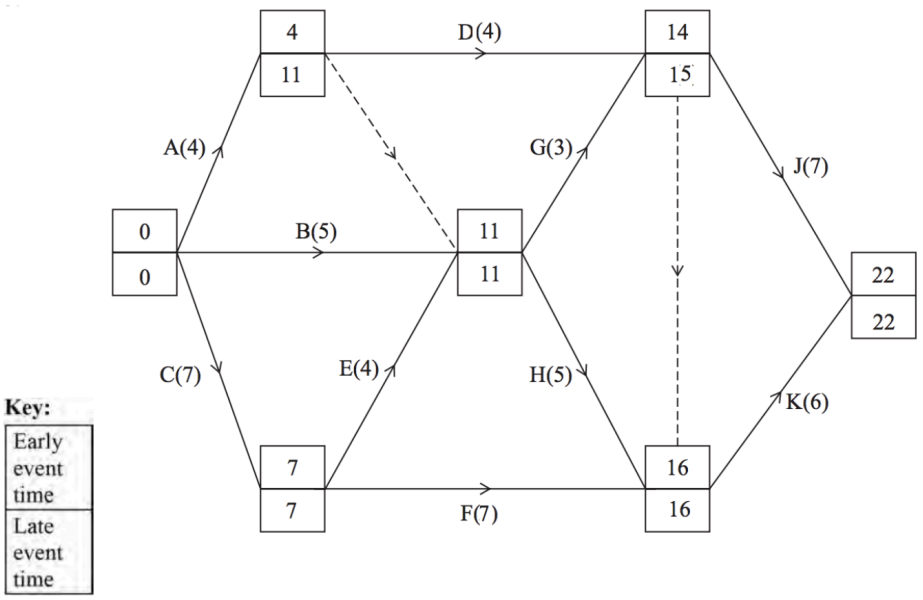
With all of the activities going on, some at the same time, it is crucial to be able to give a minimum completion time for the project and be able to identify what could hold it up. To do this we can use the activity network and label each node with an earliest time we can reach it by and a latest time we can reach it without delaying the project.



For the early event times Start at the source node by putting a zero and then work out the earliest you can reach each node by going out from the source node. For example, for the top left node only A needs to be done so the early event time is 4 and the middle node relies on A, B and E which relies on C. As a result it will take 11 for the early event time. This continues to give us:



We can then put 22 as the late event time for the sink node and 0 as the late event time for the source node and do the process backwards. For example the bottom right node only has K after it so 22 - 6 = 16 for its late event time. Then for the top right node only K and J are after it, since J takes the longest to complete the latest we can reach that node is at time 15. For the middle node G and H come after it. The latest G can end we found to be 15, 15 - 3 gives us a latest start time of 12 for G. The latest H can end we found to be 16, 16 - 5 = 11. The earliest of these is 11 so the middle node's late event time is 11.

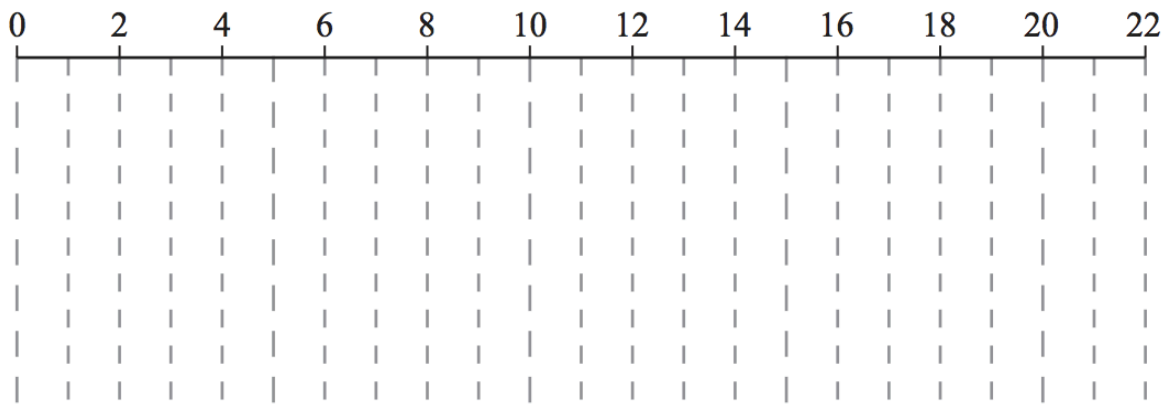


Critical activities are activities which if they take longer than expected will impact the time taken to complete the project (currently 22). The nodes they start and end at will have the same early and late event times. The difference between the start time and end time is equal to the length of the activity. Here the critical activities are C, E, H and K. B and F may look critical but their time is different to the difference between start and end times.

As a result the critical path is C, E, H, K. This is the path of critical activities going from the source node to the sink node.

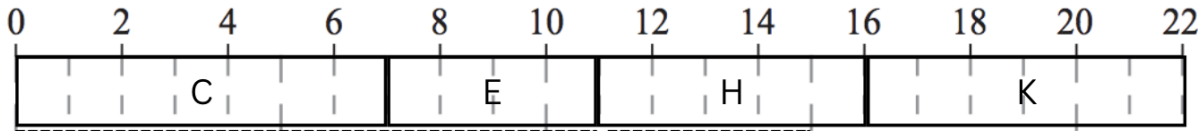
With multiple activities happening at the same time more than one person is needed to complete the project. Presuming each activity requires one person we can get a lower bound for the number of workers needed by totaling the activity times and dividing by our completion time. (4 + 5 + 7 + 4 + 4 + 3 + 5 + 7 + 6 + 7) / 22 = 2.364 Our lower bound is 3 workers.

Gantt charts

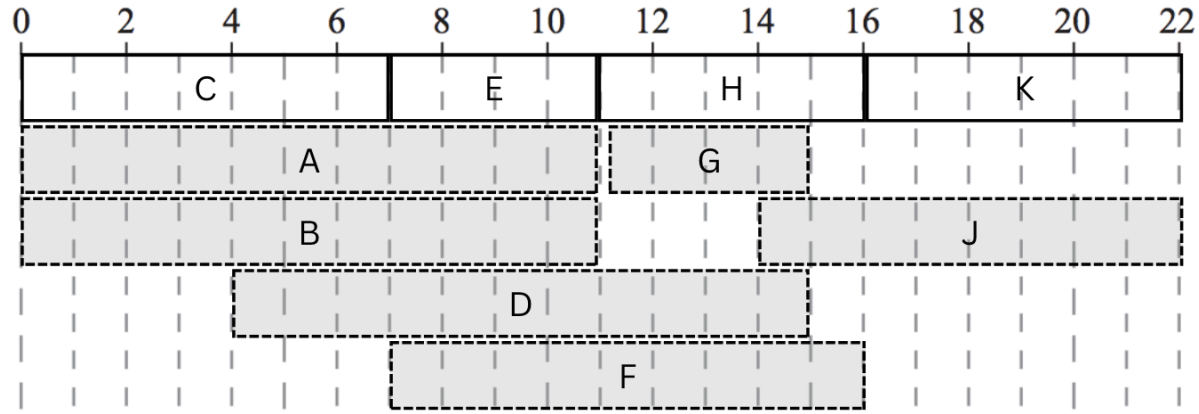


This is a Gantt chart, it is used to schedule activities. Each activity is drawn as a solid rectangle within a dashed rectangle within which it can be moved based on the early and late event times found earlier.

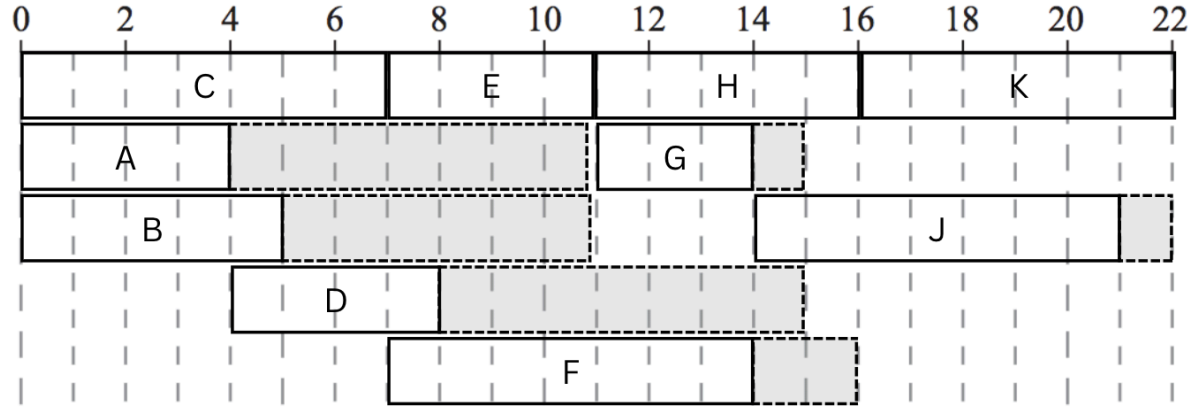
1. Put the critical path across the top. This has no space to move as they are critical.



2. Shade in for every other activity from its early start time to its late finish time.



3. Put in the activities at the start of their dashed boxes using their durations and early start times.



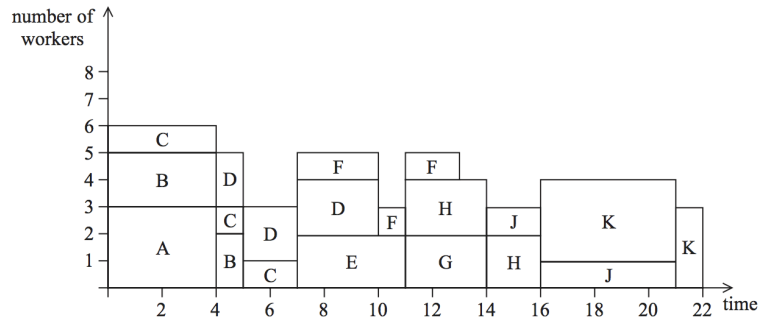
The remaining shaded area is called the float, F has a float of 2 and this corresponds to how late you can start it without affecting the finish time of the project.

You can see from the finished gantt chart that B can be put back by 4 and fit into the second row. J also fits into the second row leaving D and F which can fit in a third row together. This way we can show for certain only 3 workers are needed if each activity only requires one worker.

Resource histograms

In practice not every activity will only require one worker, as a result we use resource histograms to determine the number of workers needed for a project to be finished as quickly

as possible. A resource histogram has time into the project on the x-axis and the number of workers required on the y-axis as this changes depending on the ongoing activities.



Here we can see that although a maximum of 3 activities happen at any one time, up to 6 workers are required during the project.
To draw one, you need to have arranged activities so they overlap as little as possible on a gantt chart and also know how many workers each activity requires. Then you can add the activities at the correct time with the correct height. It is helpful to break up the activities into separate parts like with C so that it forms a conventional histogram.

Scheduling based on using the minimum number of workers

Critical activities are unable to change without putting back the project. As a result everything else must work around them. Using a Gantt chart and visualising the boxes being able to slide back and forth in their shaded area use trial and error to be able to avoid activities overlapping. This normally requires having one activity start as soon as one ends. If more than one worker is required for an activity feel free to draw it as twice as tall if it helps you visualise it. Most of the time when scheduling the activities will only require one worker each.

Linear programming

Forming linear programming problems

Linear programming problems include being able to optimise an equation in terms of a number of variables which are subject to certain constraints. The best format for solving the problem is as follows:

Maximise P where $P = x + 5y$
Subject to:
$y - 2x \leq 1$
$2y + x \leq 12$
$y + x \leq 8$
Where $x, y \geq 0$

We can solve this graphically or algebraically, to do so algebraically we need to turn the inequalities and equalities by introducing slack variables. These are positive real numbers which are equal to the difference between the sides of the inequality.

$y - 2x \leq 1$ => $-2x + y + s_1 = 1$
$2y + x \leq 12$ => $x + 2y + s_2 = 12$
$y + x \leq 8$ => $x + y + s_3 = 8$

Where s_1, s_2 and $s_3 \geq 0$

If instead of y being less than or equal to a constant, y is greater than or equal to a constant we also use artificial variables like below:

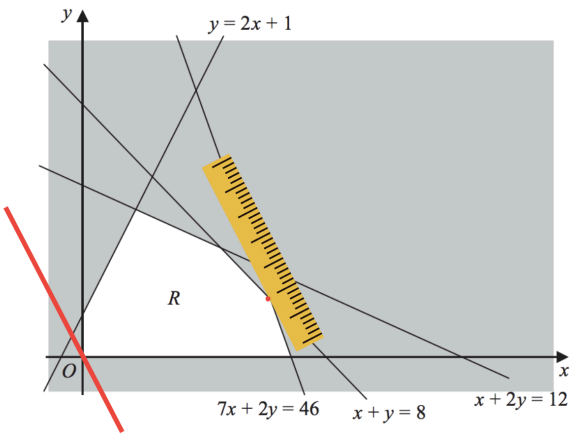
$y + 2x \geq 7$ => $y + 2x - s_1 + a_1 = 7$

The artificial variable is also ≥ 0 and we will only accept solutions where it is equal to 0 so we know that the original inequality is satisfied. Here s_1 is a surplus, not a slack, variable. These do a very similar job but surplus is used when there is an artificial variable and slack when there isn't.
Slack variables are used when there is slack in the system, surplus variables remove the surplus.

Solving graphically

When only two basic variables (like x and y) are used we can solve the problem on a graph as it has two axes.

1. You don't need to use slack, surplus or artificial variables. Just show the inequalities on a graph and shade in the area not satisfied by them.
2. Shade in the areas where x, y < 0 as x, y ≥ 0.
3. The area not shaded in is the feasible region, any points in here will satisfy the objective.
4. There are two ways to find the optimal point within the feasible region:
    a. Set the objective function equal to zero and draw it on the graph. Then hold a ruler against it and (keeping the gradient the same) move it across the region until it only touches the feasible region by one point or a line. This point (or anywhere on this line) is the optimal solution.



    b. Try all corners of the feasible region in the objective function to find which has the highest value. The one that gives the highest value is the optimal solution.

c. If integer solutions are required using graph paper you can find the optimal solution like above then put into the objective function the four coordinates surrounding it as one of them is likely to be the optimal solution. You should also check that they satisfy the constraints to ensure that your answer does. This only produces an approximation of the integer solution.

Simplex algorithm

To solve algebraically one method is the simplex algorithm. This is done once the problem is imputed into a table like so:

Maximise P

$P = x + 3y + z$	=>	$P - x - 3y - z = 0$
$x + 4y \leq 12$	=>	$x + 4y + s_1 = 12$
$3x + 6y + 4z \leq 48$	=>	$3x + 6y + 4z + s_2 = 48$
$y + z \leq 8$	=>	$y + z + s_3 = 8$

Where $x, y, z \geq 0$

B.V	x	y	z	s_1	s_2	s_3	Value
s_1	1	4	0	1	0	0	12
s_2	3	6	4	0	1	0	48
s_3	0	1	1	0	0	1	8
P	-1	-3	-1	0	0	0	0

In the first column goes the artificial variable if present or the slack variable if present. These are the variables that take the numbers in the values column.

1. Find the pivot column. This is the column which has the most negative value in the objective (bottom) row. In this case the ‘y’ column.
2. Divide the values for all except the objective row by the number in that row’s pivot column.

B.V	x	y	z	s_1	s_2	s_3	Value	θ
s_1	1	4	0	1	0	0	12	12 ÷ 4 = 3
s_2	3	6	4	0	1	0	48	48 ÷ 6 = 8
s_3	0	1	1	0	0	1	8	8 ÷ 1 = 8
P	-1	-3	-1	0	0	0	0	

3. The row with the smallest non-negative θ is the pivot row which intersects with the pivot column to give us the pivot, in this case 4.

4. The variable in the pivot column (y) is moved to the far left of the pivot row (where s_1 was). Then divide all numbers in the pivot row by the pivot.

B.V	x	y	z	s_1	s_2	s_3	Value	Row operation
y	¼	1	0	¼	0	0	3	$Row_1 \div 4$
s_2	3	6	4	0	1	0	48	
s_3	0	1	1	0	0	1	8	
P	-1	-3	-1	0	0	0	0	

5. Use the pivot row to cancel out all instances of y except in the pivot row.

B.V	x	y	z	s_1	s_2	s_3	Value	Row operation
y	¼	1	0	¼	0	0	3	$Row_1 \div 4$
s_2	3/2	0	4	-3/2	1	0	30	$Row_2 - 6 \times Row_1$
s_3	-¼	0	1	-¼	0	1	5	$Row_3 - Row_1$
P	-¼	0	-1	¾	0	0	9	$Row_4 + 3 \times Row_1$

6. Find a new pivot column and repeat until there are no negative variables in the objective row. Here we would continue with the z column in the next iteration.

Once you have done the method enough times to have no negative variables in the objective function you can use the first column and value column to assign values to variables. If we were finished here P would be 9, y would be 3, s_2 would be 30 and s_3 would be 5.

If it is instead asking for you to minimise the variable P then one way to do it is to say that C = -P and to maximise C. Then it can be solved through the simplex method. For example:
Minimise P where

$P = -2x - 3y \quad \Rightarrow \quad C - 2x - 3y = 0$
$5x + 4y \leq 32 \quad \Rightarrow \quad 5x + 4y + s_1 = 32$
$x + 2y \leq 10 \quad \Rightarrow \quad x + 2y + s_2 = 10$

Basic variables	x	y	s_1	s_2	Value	θ
s_1	5	4	1	0	32	32/4 = 8
s_2	1	2	0	1	10	10/2 = 5
C	-2	-3	0	0	0	

Basic variables	x	y	s_1	s_2	Value	Row operations
s_1	3	0	1	-2	12	$Row_1 - 4 \times Row_2$
y	½	1	0	½	5	$Row_2 / 2$
C	-½	0	0	3/2	15	$Row_3 + 3 \times Row_2$

Basic variables	x	y	s_1	s_2	Value	θ
s_1	3	0	1	-2	12	12 ÷ 3 = 4
y	½	1	0	½	5	5 ÷ ½ = 10
C	-½	0	0	3/2	15	

Basic variables	x	y	s_1	s_2	Value	Row operations
x	1	0	⅓	-⅔	4	$Row_1 \div 3$
y	0	1	-⅙	⅚	3	$Row_2 - ½ Row_1$
C	0	0	⅙	7/6	17	$Row_3 + ½ Row_1$

Given there are no negative values in the objective row this is the optimal solution.
x = 4
y = 3
P = -17

Two stage simplex and big M methods

Two stage simplex

When one or more of the constraints are greater than or equal to and require artificial variables instead of only including less than or equal to inequalities then it is best to use the two stage simplex method.

1. Formulate the linear programming problem in terms of the objective function, slack, surplus and artificial variables.
2. To have a valid solution the artificial variables must sum to 0, to ensure we have this we will create an auxiliary objective function to minimise the artificial variables and solve that using simplex.
3. If it is found that there is a basic feasible solution we now solve an extract from the previous simplex tableau excluding the artificial variables and the auxiliary objective function.

For example:

Maximise P

$P = 2x + y + 3z$	=>	$P - 2x - y - 3z = 0$
$x + 2y + 3z \leq 45$	=>	$x + 2y + 3z + s_1 = 45$
$3x + 2y \geq 9$	=>	$3x + 2y - s_2 + a_1 = 9$
$-x + 4z \geq 4$	=>	$-x + 4z - s_3 + a_2 = 4$

Where $x, y, z \geq 0$

The auxiliary objective function is $W = a_1 + a_2$ which we must first minimise which is the same as maximising C where $C = -W$.

$C + a_1 + a_2 = 0$

$C + 9 + s_2 - 3x - 2y + 4 + s_3 + x - 4z = 0$

$C - 2x - 2y - 4z + s_2 + s_3 = -13$

B.V	x	y	z	s_1	s_2	s_3	a_1	a_2	Value	θ
s_1	1	2	3	1	0	0	0	0	45	45/3 = 15
a_1	3	2	0	0	-1	0	1	0	9	9/0 = NaN
a_2	-1	0	4	0	0	-1	0	1	4	4/4 = 1
P	-2	-1	-3	0	0	0	0	0	0	0/-3 = 0
C	-2	-2	-4	0	1	1	0	0	-13	

B.V	x	y	z	s_1	s_2	s_3	a_1	a_2	Value	Row operations
s_1	7/4	2	0	1	0	¾	0	0	42	$Row_1 - 3 \times Row_3$
a_1	3	2	0	0	-1	0	1	0	9	Row_2
a_2	-¼	0	1	0	0	-¼	0	¼	1	$Row_3 / 4$
P	-11/4	-1	0	0	0	-¾	0	¾	3	$Row_4 + 3 \times Row_3$
C	-3	-2	0	0	1	0	0	1	-9	$Row_5 + 4 \times Row_3$

B.V	x	y	z	s_1	s_2	s_3	a_1	a_2	Value	θ
s_1	7/4	2	0	1	0	¾	0	0	42	42 ÷ 7/4 = 24
a_1	3	2	0	0	-1	0	1	0	9	9 ÷ 3 = 3
z	-¼	0	1	0	0	-¼	0	¼	1	1 ÷ -¼ = -4
P	-11/4	-1	0	0	0	-¾	0	¾	3	3 ÷ -11/4 = -12/11
C	-3	-2	0	0	1	0	0	1	-9	

B.V	x	y	z	s_1	s_2	s_3	a_1	a_2	Value	Row operations

s_1	0	-⅓	0	1	7/6	¾	-7/6	0	63/2	Row_1 - 7/4 x Row_2
x	1	⅔	0	0	-⅓	0	⅓	0	3	Row_2 ÷ 3
z	0	⅙	1	0	-1/12	-¼	1/12	¼	7/4	Row_3 + 1/4 x Row_2
P	0	⅚	0	0	-11/12	-¾	11/12	¾	45/4	Row_4 + 11/4 x Row_2
C	0	0	0	0	0	0	1	1	0	Row_5 + 3 x Row_2

This solves to give C = 0 when s_1 = 63/2, x = 3, z = 7/4 and P = 45/4. This is valid as C + a_1 + a_2 = 0 so when C is 0 so are a_1 and a_2 since they are both greater than or equal to 0.

We can then solve this extract from the above table to get our answer, this is the second part of the two stage simplex and is now the same as the simplex method.

B.V	x	y	z	s_1	s_2	s_3	Value
s_1	0	-⅓	0	1	7/6	¾	63/2
x	1	⅔	0	0	-⅓	0	3
z	0	⅙	1	0	-1/12	-¼	7/4
P	0	⅚	0	0	-11/12	-¾	45/4

You can also minimise using the two stage simplex using the same method as before.

The big M method

The big M method is another way of using the simplex method when at least one ≥ is involved (outside of non negativity constraints). Instead of first minimising the artificial variables, a large punishment for having them is introduced in the objective function. This is done by subtracting the sum of all artificial variables multiplied by M, an arbitrarily large number.

For example:

Maximise: $P = 2x + y$ => $P = 2x + y - M(a_1)$

=> $P = 2x + y - M(13 + s_2 - x - 3y)$

=> $P = (2 + M)x + (1 + 3M)y - 13M - Ms_2$

=> $P - (2 + M)x - (1 + 3M)y + 13M + Ms_2 = 0$

=> $P - (2 + M)x - (1 + 3M)y + Ms_2 = -13M$

Subject to: $x - y \leq 11$ => $x - y + s_1 = 11$

$x + 3y \geq 13$ => $x + 3y - s_2 + a_1 = 13$

$x, y \geq 0$

It is then able to be solved using the normal simple method as shown below.

B.V	x	y	s_1	s_2	a_1	Value	θ
s_1	1	-1	1	0	0	11	11/-1 = -11
a_1	1	3	0	-1	1	13	13/3 = 4.3
P	-2 - M	-1 - 3M	0	M	0	-13M	

The y column is chosen as the pivot column since M is so large that -1 -3M is made the most negative in the objective function as the Value column doesn't count. 3 is the pivot since 4.3 is the smallest non negative θ.

B.V	x	y	s_1	s_2	a_1	Value	Row operations
s_1	1.333	0	1	-⅓	⅓	15.333	$Row_1 + Row_2$
y	⅓	1	0	-⅓	⅓	4.333	$Row_2 / 3$
P	-1.666	0	0	-⅓	⅓ + M	4.333	$Row_3 + (1 + 3M) Row_2$

'y' is now added as a basic variable

B.V	x	y	s_1	s_2	a_1	Value	θ
s_1	1.333	0	1	-⅓	⅓	15.333	15.333/1.333 = 11.5
y	⅓	1	0	-⅓	⅓	4.333	4.333 / ⅓ = 15
P	-1.666	0	0	-⅓	⅓ + M	4.333	

B.V	x	y	s_1	s_2	a_1	Value	Row operations
s_1	1	0	0.75	-¼	¼	11.5	$Row_1 / 1.333$
y	0	1	-¼	-¼	¼	0.5	$Row_2 - Row_1 / 3$
P	0	0	1.25	-¾	¾ + M	23.5	$Row_3 + 5/3\ \ Row_1$

B.V	x	y	s_1	s_2	a_1	Value	θ
x	1	0	0.75	-¼	¼	11.5	11.5 / -¼ = -46
y	0	1	-¼	-¼	¼	0.5	0.5 / -¼ = -2
P	0	0	1.25	-¾	¾ + M	23.5	

Since both θs are negative there is no smallest positive value so we can't continue. This means we have reached our optimal solution.
P = 23.5 when x = 11.5 and y = 0.5 and s_1, s_2 and a_1 are 0.

When doing the big M method make sure to choose the right pivot (12 - M is more negative than -20) and always make sure to update the basic variables in the simplex method.