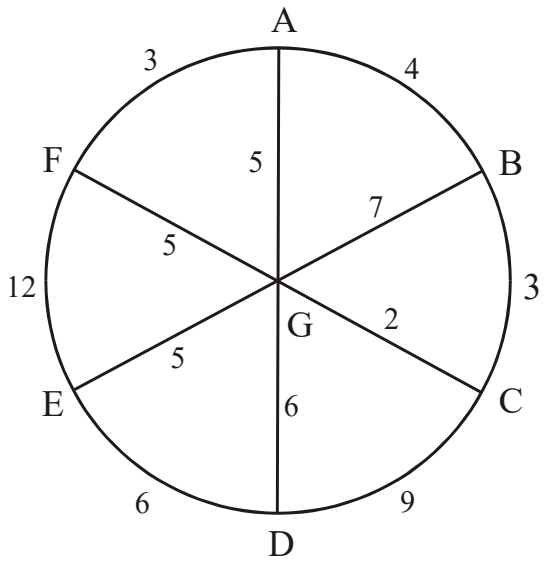
Minimum connector problem

The minimum connector of a network

A graph is a collection of vertices (also called nodes) and edges (also called arcs) which join vertices to one another. When these edges also carry a number (a weight), the graph is called a weighted graph. The edges can also be directed – in other words, represent a route from, say, vertex A to B, but not from B to A. A directed graph is also called a digraph.

A network is a weighted graph or weighted digraph. In this unit we consider only networks that are weighted graphs. We also only consider graphs that are fully connected – in other words, graphs where every vertex is connected to every other by some path or other.

Graphs can be used to represent (or model) many practical situations. For example, a network could show the possible connections between pumping stations in a projected water main system, and the weights along the edges could indicate the costs in £1000s of installing those connections.



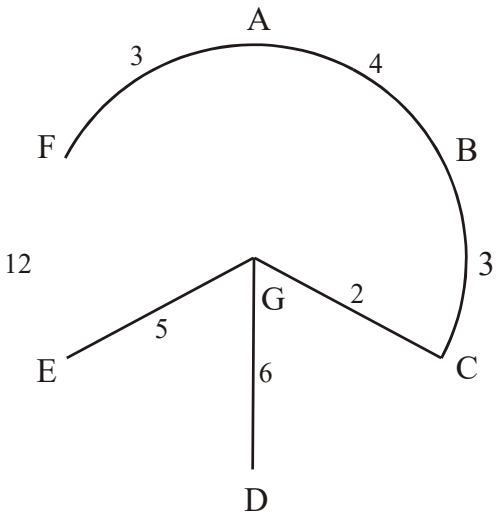
One practical question could be: given that no two pumping stations need to be connected more than once, and that every pumping station must be connected to every other, not necessarily directly, what is the least cost of installing the network?

An answer to this questions requires one to find the ***minimum connector*** **–** also called the *minimum spanning tree* or *minimum weight spanning tree.*

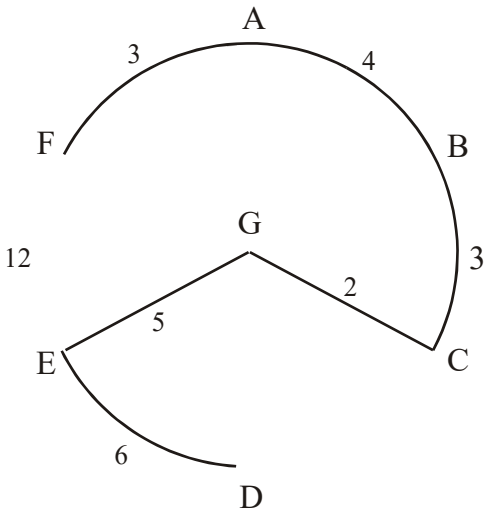
A *spanning tree* is a weighted graph that connects all the vertices of the graph to one another, either directly or through other vertices, such that no vertex is connected to the graph more than once.

A minimum connected is a spanning tree that has the least total weight. In other words, it is a minimum connector is a weighted graph (or digraph) in which all the vertices are connected to all the others either directly or through other vertices, and which has the least weight possible.

A solution to this problem is



There can be more than one solution to such a problem, since there can be sometimes more than one way to construct a minimum connector. For example, in this case the vertex D can be connected to the tree by the edge DG, as above, or by the edge ED, since both edges have the same weight, and there is no edge connecting D to the tree which has a lower weight.



The problem of finding a minimum connector, (or minimum spanning tree), is one that should be programmable – in other words, there should be an algorithm for finding such a tree.

This is indeed the case.

The algorithms that find the minimum connector are called **greedy algorithms**. The thinking behind this term is that in finding the minimum connector you choose the easiest path. Since you are trying to link nodes in the shortest possible way, or costing the least money, it is jokingly thought that this is "greedy".

There are two main greedy algorithms used for solving the minimum connector problem. These are Prim's algorithm and Kruskal's algorithm.

Before we describe these algorithms, we note the following theorem:

Theorem

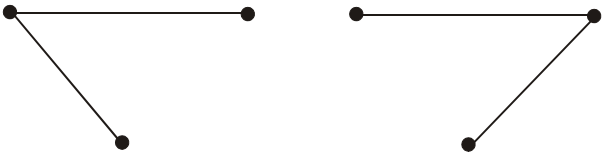
If a network has n vertices, then there are $n - 1$ edges in any spanning tree.

The proof of this theorem would be an easy application of the technique of mathematical induction. However, here we only explain it.

If we have two vertices, then we require only one edge to connect them.

So there are 2 vertices and 1 edge.

If we then add one more vertex, we require one more edge to connect that vertex to the existing tree. This edge could connect the vertex to the tree in one of two ways, but there will be just one edge required.



So there would be 3 vertices and 2 edges – or n vertices and $n - 1$ edges. In general, adding one vertex adds one edge, and as we started with one less edge than vertices, we have one less edge in all.

Prim's algorithm

STEP 1 Choose any vertex to be the starting point of the spanning tree, T.

STEP 2 Add to the tree, T, the shortest edge that has one vertex in T and the other vertex not in T. If there is a choice of two or more such edges, choose one of them at random.

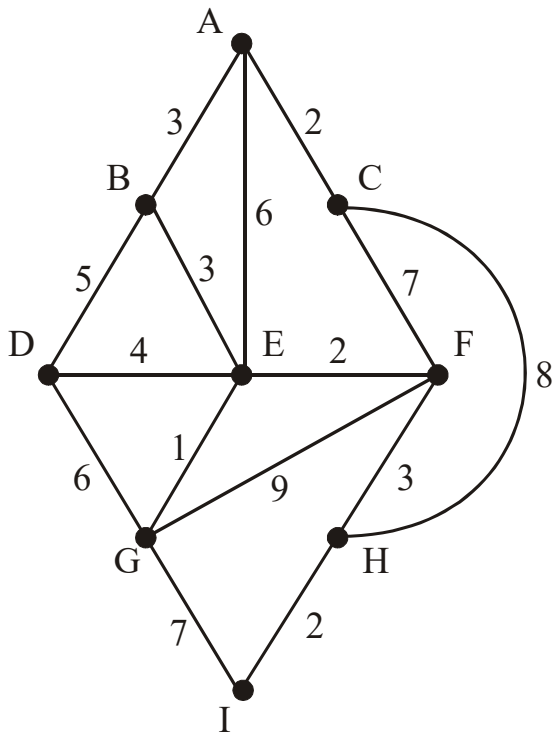
STEP 3 Repeat step 2 until all the vertices of the graph are added. This means that there will be $n - 1$ edges, so this step could also be written: repeat step 2 until there are $n - 1$ edges in the tree.

It is a theorem that this algorithm does indeed find the minimum connector. One proof of such a theorem could be by contradiction. In other words, you would start by assuming that there was a minimum connector of a network, but that Prim's algorithm did not find it, and then show that this lead to a contradiction.

Prim's algorithm is best illustrated by worked examples.

Example (1)

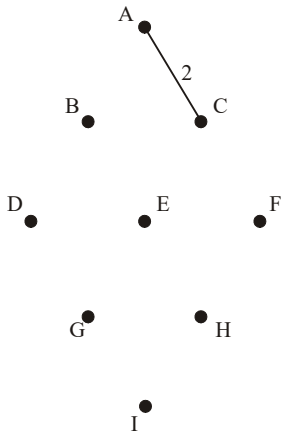
In the following network the weights along the edges represent distances in kilometres between towns.



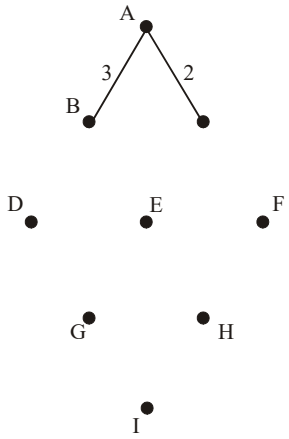
Using Prim's algorithm find a minimum spanning tree.

Solution

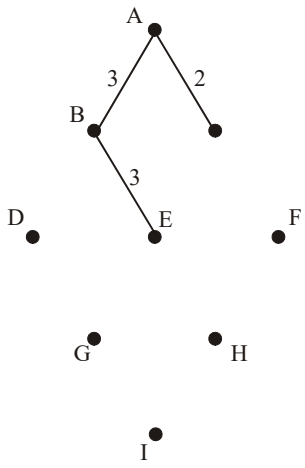
We can choose any vertex as the starting point of the minimum spanning tree, so let us choose vertex A. We add to the vertex A the shortest edge that is connected to A. This is the edge AC with length 2.



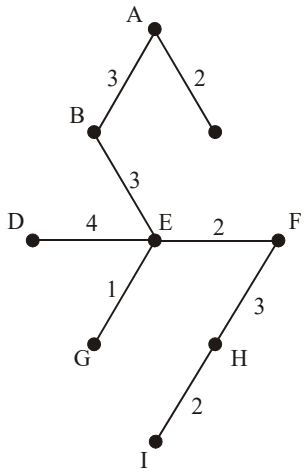
The tree *T* consists of just the edge AC. We add to *T* the shortest remaining edge that connects with *T*. This is the edge, AB. The tree becomes



We iterate (repeat) this process until all the vertices are connected. The next step is



The end result is



With length

$$\text{length} = 3+2+3+4+2+1+2+3 = 20km$$

Matrix representation of Prim's algorithm

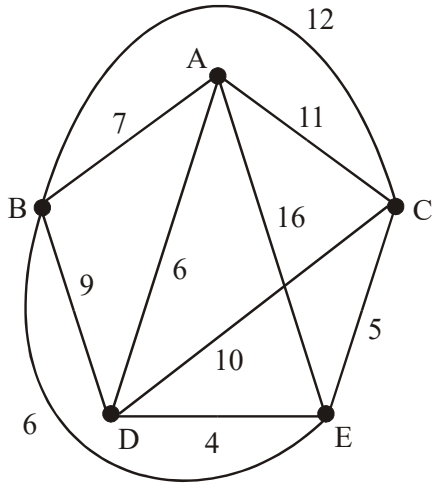
A matrix is a rectangular array of numbers.

We require a matrix representation of Prim's algorithm. Once we have the matrix, we can use operations on the matrix to find the minimum connector.

Such operations would be suitable for programming for a computer, which is one reason why we are interested in a matrix representation.

Example (2)

The following weighted graph shows the costs in £1000 of connecting five towns telephone cables. Using Prim's algorithm in matrix form, find a minimum spanning tree for the telephone network, and its cost.



Solution

Firstly, we have to construct the matrix representation of the weighted graph. This is

To \ From	A	B	C	D	E
A	-	7	11	6	16
B	7	-	12	9	6
C	11	12	-	10	5
D	6	9	10	-	4
E	16	6	5	4	-

The matrix shows the weight of connecting one vertex to another. The row represents the starting vertex; the column the ending vertex. Thus the edge connecting C to D is shown by the entries in the matrix

To \ From	A	B	C	D	E
A	-	7	11	6	16
B	7	-	12	9	6
C	11	12	-	(10)	5
D	6	9	(10)	-	4
E	16	6	5	4	-

There are two entries in the matrix for this one edge. This is because the edge from C to D is the same as the edge from D to C. In a directed graph (a digraph) the entries could be different, because the edge from C to D could have a different weight from the edge running in the opposite direction from D to C. However, here we are not concerned with directed graphs.

In Prim's algorithm we start with any randomly chosen vertex, so we may as well start with A. We chose the shortest edge in the row, which is the edge AD with length 6.

To \ From	A	B	C	D	E
A	-	7	11	(6)	16
B	7	-	12	9	6
C	11	12	-	10	5
D	6	9	10	-	4
E	16	6	5	4	-

Tree	Length
AD	6

This connects A and D to the tree so we delete columns A and D to indicate that they are now connected to the tree. On the other hand, since only A and D are connected to the tree, only rows A and D can be used for further connections.

To \ From	A	B	C	D	E
A	-	7	11	(6)	16
B	7	-	12	9	6
C	11	12	-	10	5
D	6	9	10	-	4
E	16	6	5	4	-

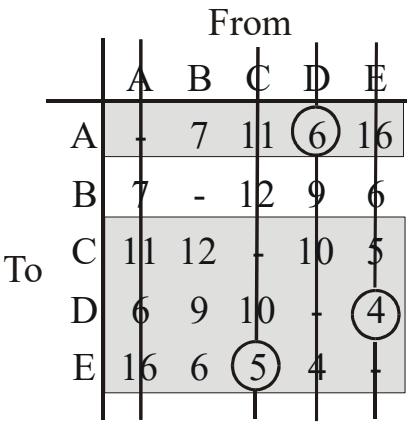
Tree	Length
AD	6

We choose the next edge from rows A and D with the least weight. We cannot choose from columns A and D. This gives the edge DE. We add this to our tree. We delete the column E, which is now connected to the tree. Since E is connected, we can use rows A, D and E in future iterations.

To \ From	A	B	C	D	E
A	-	7	11	(6)	16
B	7	-	12	9	6
C	11	12	-	10	5
D	6	9	10	-	(4)
E	16	6	5	4	-

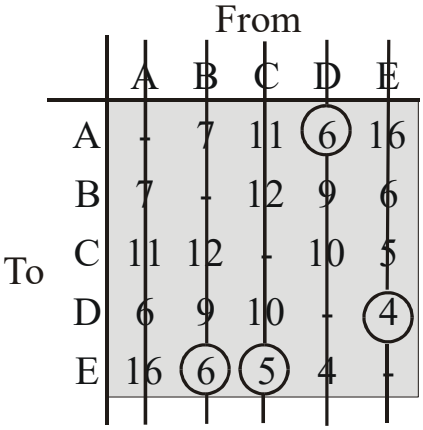
Tree	Length
AD	6
DE	4

The least weighted edge is now EC with weight 5. We add EC, delete the column C, and enable the row C.



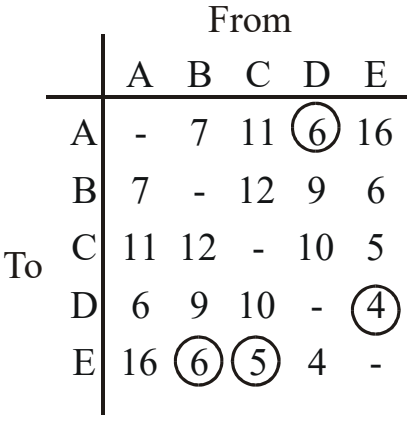
Tree	Length
AD	6
DE	4
EC	5

The final vertex to connect is B. The least edge is EB with weight 6. We add this edge, delete the column B, and we are done.



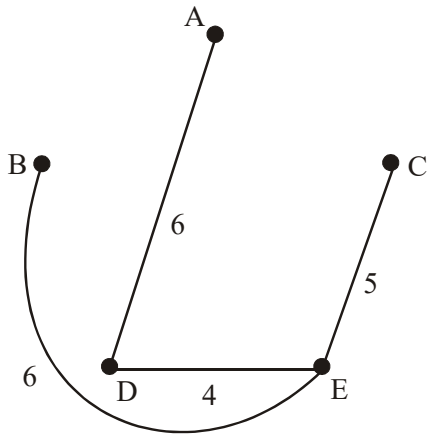
Tree	Length
AD	6
DE	4
EC	5
EB	6

The following diagram shows the solution, without the clutter of the deleted columns.



Tree	Length
AD	6
DE	4
EC	5
EB	6

The graph is



Kruskal's algorithm

Kruskal's algorithm is another algorithm used for finding minimum spanning trees. It works in terms of edge lengths rather than by going from point to point.

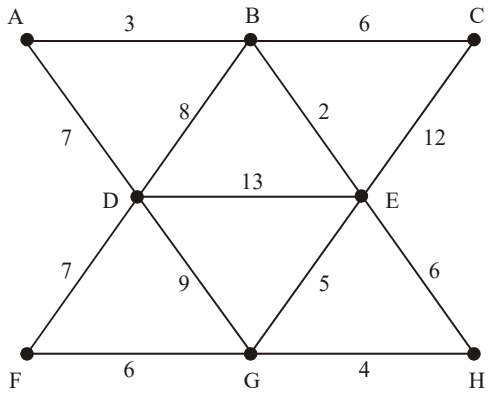
STEP 1 Choose the shortest edge.

STEP 2 Choose the next shortest edge that connects any previously unconnected vertex to any other vertex, not necessarily on the tree. Do not choose an edge if it creates a closed path. If there are two such edges, choose one of these at random.

STEP 3. Repeat step 2 until all the vertices are connected, that is until there are $n - 1$ edges, where there are n vertices.

A closed path will be one that creates a loop. If closed paths were allowed vertices could be connected by more than one path. This would mean that the tree could not be a minimum spanning tree.

Example (3)



(*i*) Use Kruskal's algorithm to construct a minimum spanning tree for the above network. The weights of the edges are in minutes.

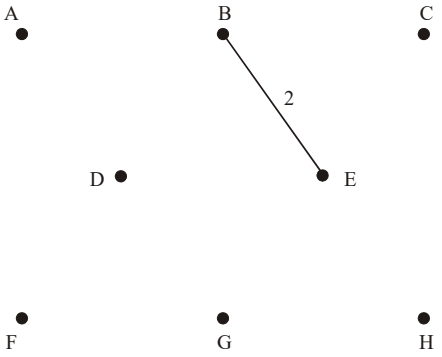
(*ii*) Find a route that, starting and finishing at A, passes through every node at least once, and takes exactly 52 minutes.

(*iii*) Find a route that, starting and finishing at A, passes through every node at least once, and takes less than 52 minutes. State its time.

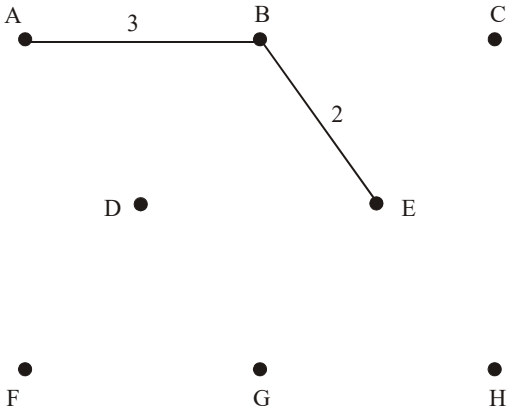
Solution

(*i*)

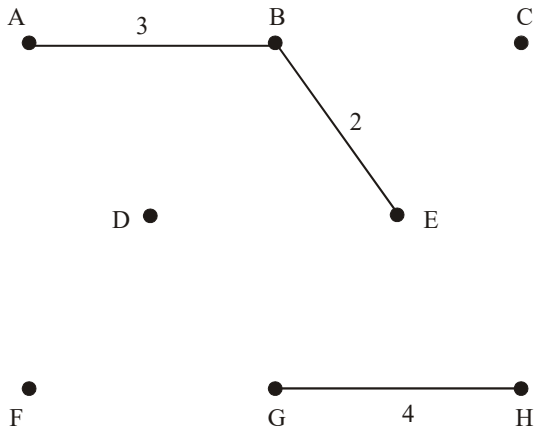
The least weighted edge is BE, so we begin with that



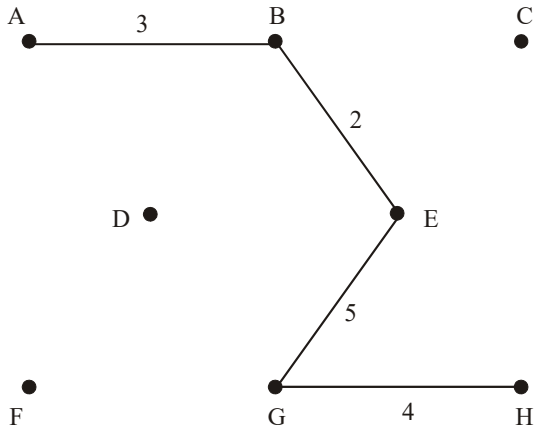
The next is AB, so that is added to the network.



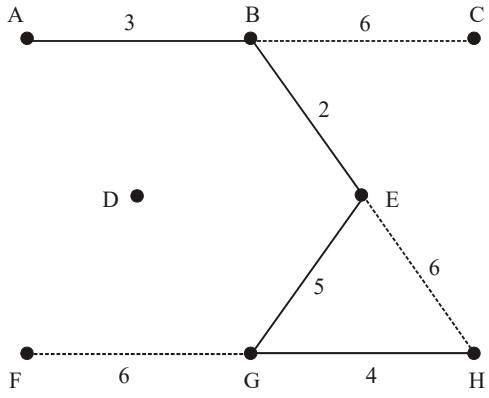
The next shortest path is GH



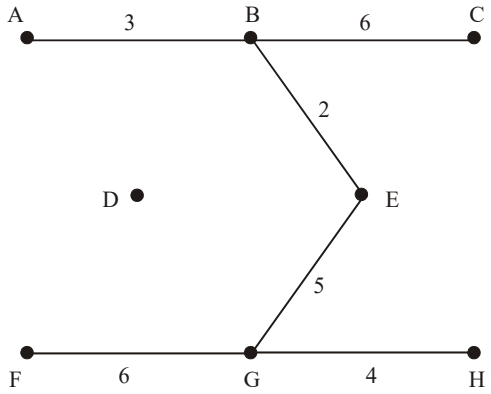
At this point the next possible shortest edge is EG



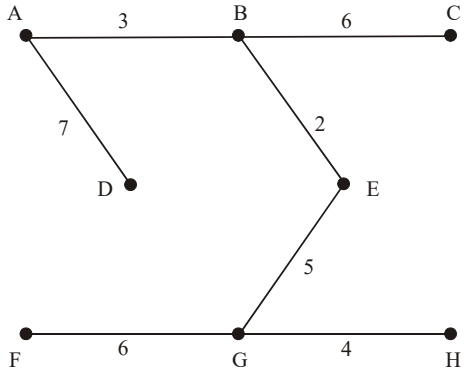
At this point there are three edges with the next minimum weight – BC, FG and EH – all of which have weight 6.



However, EH cannot be added, since it forms a closed path and connects E and H twice to the spanning tree. Neither BC nor FG do this, and both of these are added.



We have one more vertex to connect, and there are two edges with the next least weight, AD and DF. We choose one of these at random, say AD and connect that to the tree. The final spanning tree is

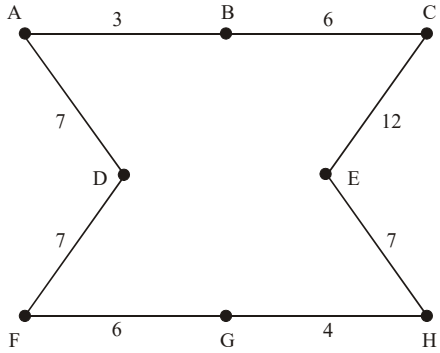


The length of the minimum spanning tree is

$$7 + 3 + 6 + 2 + 5 + 6 + 4 = 33$$

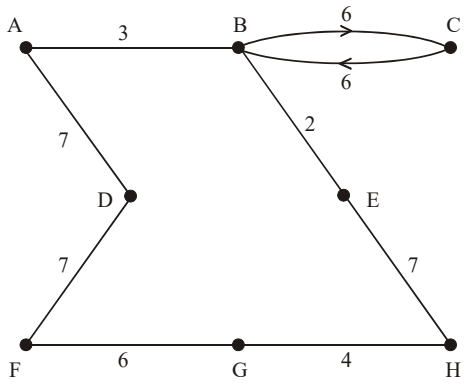
(*ii*)

This type of problem is solved by "inspection" – in other words, you just try one solution after another until you "see" the answer.



(*iii*)

The answer to the first part of the question should help here. We need to replace at least one edge in the answer to part (*ii*) by a shorter path. The edge EC with weight 12 is the longest, and by doubling up on the edge BC and adding BE we replace this by a path of length 2 + 6 = 8, which is a saving of 4.



The length is 52 – 4 = 48 minutes.