

CHAPTER 1 TREES

TREES

Let T be a relation on a finite set A . T is called a tree if there is a vertex v_0 such that there is only one path from v_0 to every other vertex, but no path from v_0 to v_0 .
 v_0 is called the root of the tree.

Eg (1)

Sketch the digraphs of some relations which are not trees and some which are trees.

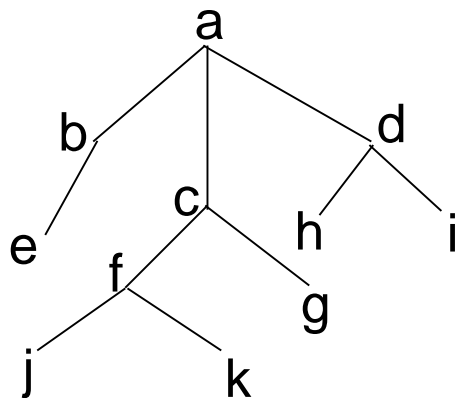
Theorem 1

Let (T, v_o) be a rooted tree with root v_o . Then

- (a) There are no cycles in T .
- (b) v_o is the only root in T .
- (c) v_o has in-degree __, and every other vertex has in-degree __.

Eg (2)

A typical tree looks like this:



With reference to the tree above, illustrate the meaning of parent, offspring, siblings, and descendants.

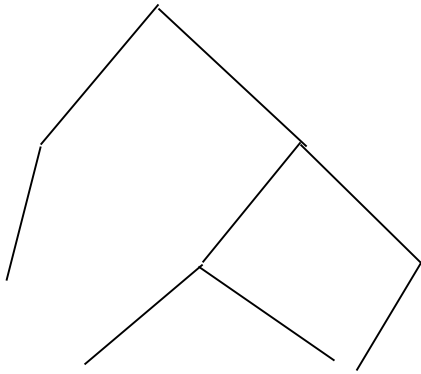
Theorem 2

Let (T, v_o) be a rooted tree. Then

- (a) T is irreflexive.
- (b) T is asymmetric.
- (c) If $(a, b) \in T$ and $(b, c) \in T$, then $(a, c) \notin T$.

Example

Eg (3)



Binary tree (need not have both offspring present)

Labelled tree

The vertices or edges of a tree may be labelled to indicate the use.

Eg (4)

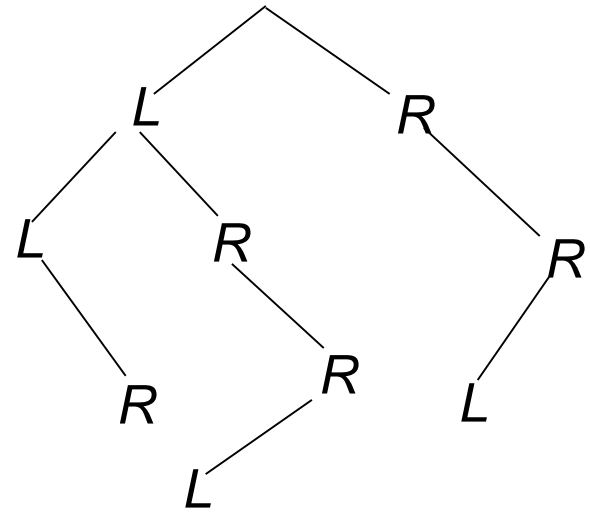
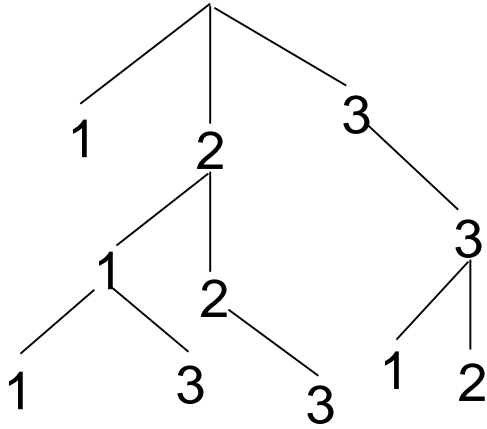
Draw a labelled tree to represent this fully parenthesised algebraic expression: $(3 - (2 \times y)) + ((y - 2) - (3 + y))$
(The central operation $+$ forms the root of the tree.)

Positional tree

Consider an n -tree. Suppose the offsprings of a vertex can be arranged in some order (ordered tree). If the vertices are labelled according to their position, we have a positional tree.

A positional binary tree may be labelled by L (for left offspring) and R (for right offspring).

Eg (5)



Tree searching

Performing a certain task (such as printing the label, compilation, etc.) at a vertex is described as visiting the vertex.

We wish to visit each vertex of a tree exactly once in a specific order. This process is called searching the tree.

Pre-order search of binary positional tree

In a binary positional tree, each vertex has 2 potential (not necessarily present) offsprings: v_L , v_R . The vertex v_L together with all its descendants (vertices connected below v_L) form the left subtree. The vertex v_R together with all its descendants form the right subtree.

Pre-order search consists of these steps:

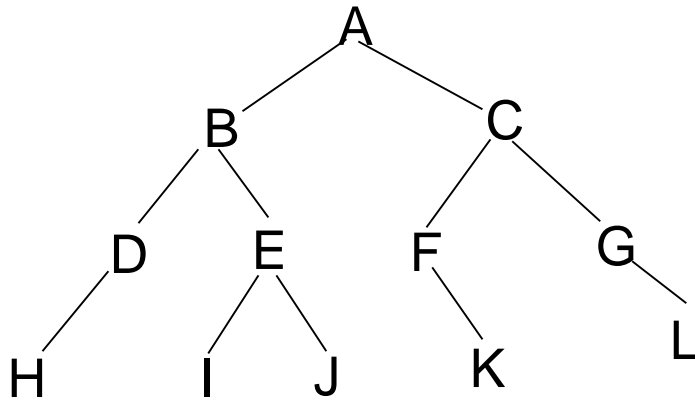
1. Visit the root.
2. Search the left subtree if it exists.
3. Search the right subtree if it exists.

(This algorithm is recursive.)

Eg (6)

Consider this binary tree:

Suppose "visiting a vertex means "printing its label". By the pre-order search, what sequence of labels will be printed ?



Solution:

Eg (7)

Consider this completely parenthesised expression:

$$(a - (b + c)) + ((d \times e) \div (f + g))$$

Draw a binary tree and apply preorder search.

By preorder search, the vertices are visited in this sequence:

Solution:

This is called prefix form, or Polish form of the given algebraic expression

To evaluate an expression in Polish form, start searching from the left. When we see an operator followed by 2 operands, this part can be evaluated, and replaced by the result of the operation. Repeat until the final result is obtained.

Eg (8)

Evaluate $+ - 2 + 3 \ 4 \div \times 8 \ 6 + 5 \ 1$

Solution:

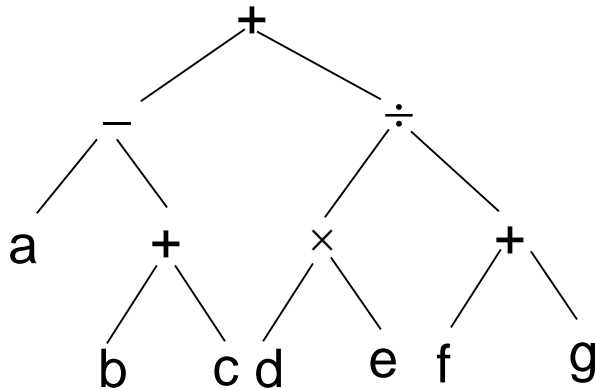
Draw a binary tree for this expression

The fully parenthesised form of the expression is

Inorder search

1. Search the left subtree if it exists.
2. Visit the root.
3. Search the right subtree if it exists.

Eg (9): Apply inorder search to the tree of eg(7).



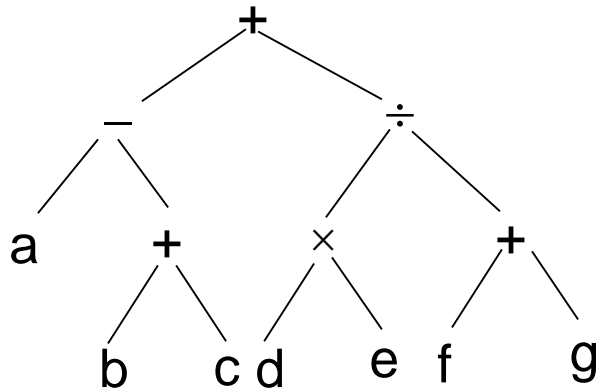
Solution:

The result is the same as the given expression with all parentheses removed. This is not very useful as the order of evaluation cannot be determined.

Postorder search

1. Search the left subtree if it exists.
2. Search the right subtree if it exists.
3. Visit the root.

Eg (10): Apply postorder search to the tree of eg(7).



Solution:

Binary Search Tree

3 problems can be studied using trees.

How should items in a list be stored so that an item can be easily located?

What series of decisions should be made to find an object with a certain property in a collection of objects of a certain type?

How should a set of characters be efficiently coded by bit strings?

Searching for items in a list is one of the most important tasks that arises in computer science.

Primary goal is to implement a searching algorithm that finds items efficiently when the items are totally ordered.

This can be accomplished through the use of a binary search tree.

- Binary search tree is a binary tree in which each child of a vertex is designated as a right or left child.
- No vertex has more than one right child or left child.
- Each vertex is labelled with a key, which is one of the items.
- Vertices are assigned keys so that the key of a vertex is both larger than the keys of all vertices in its left subtree and smaller than the keys of all vertices in its right subtree.
- Apply an inorder search on the tree, we obtain a sorted list of the items.

Eg(11)

Form a binary search tree for the words mathematics, physics, geography, zoology, meteorology, geology, psychology and chemistry (using alphabetical order)

Example (12):

Construct a binary search tree for the following list of numbers, sorted according to numerical value. Assume data are entered in the order as listed:

45, 55, 15, 35, 65, 40, 30, 10, 75, 20, 37, 25, 60, 23.

Write the sorted list by applying in-order search.

Supposing all the items are equally likely to be searched for, calculate the expected number of comparisons to locate an item.

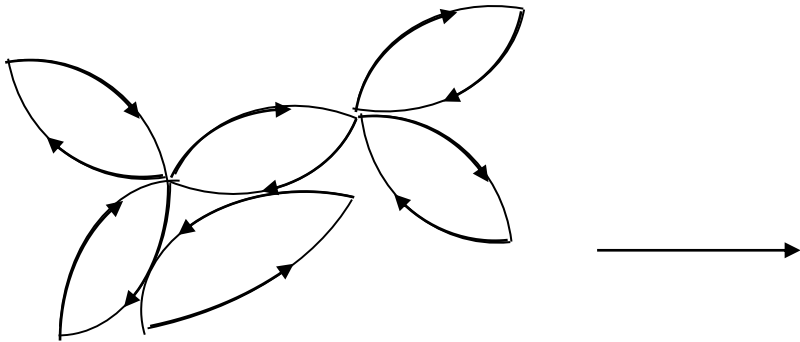
Inorder search:

Expected number of comparisons:

Undirected tree

A symmetric relation R may be represented by a graph, in which all the edges are "2-way streets". These edges are called undirected edges.

Eg (13):



Symmetric relation

A path from vertex a to vertex b is said to be a simple path if no edges are repeated.

eg. In the graph drawn above,

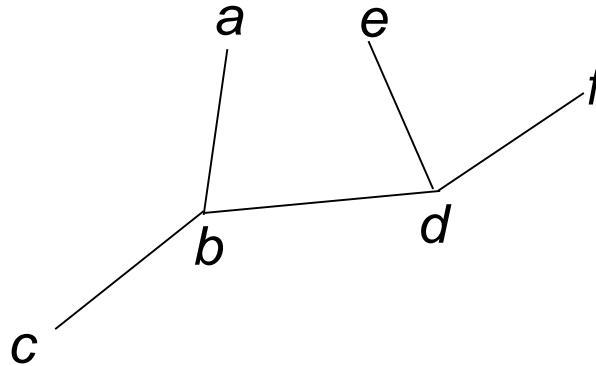
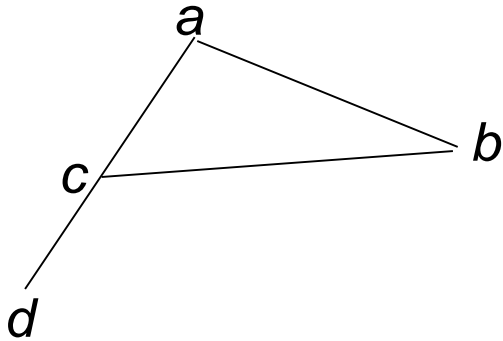
_____ is a simple path from a to d .

_____ is a path from a to d , but is not simple.

A simple cycle is a simple path that starts and ends at the same vertex. eg. In the graph just now, _____ is a simple cycle.

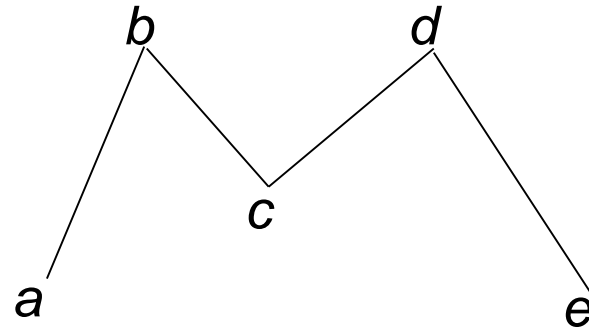
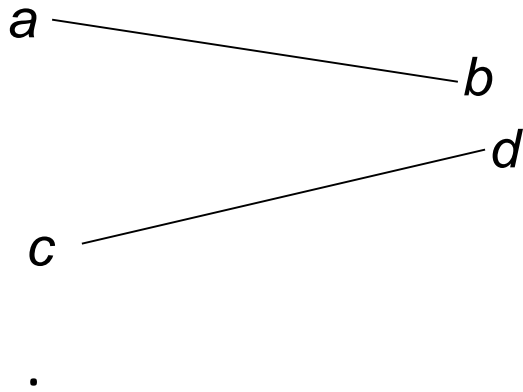
A graph is described as acyclic if there is no simple cycles in the graph.

Eg (14)



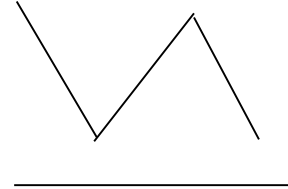
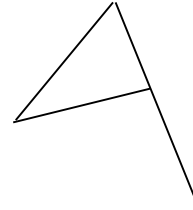
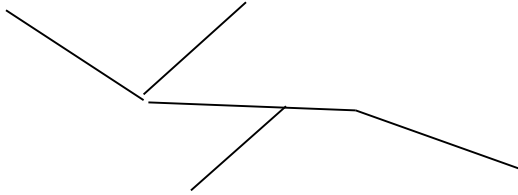
A graph is said to be connected if there is a path from any vertex to every other vertex.

Eg (15)



An undirected tree is a connected, acyclic graph.

Eg (16)

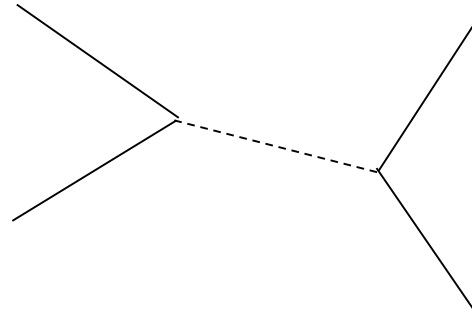
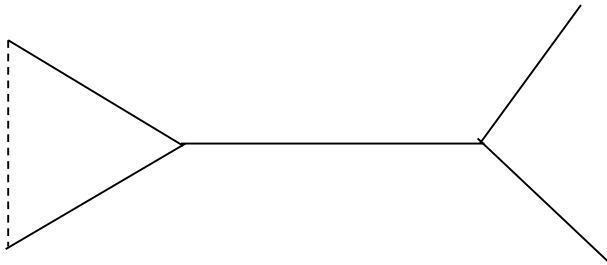


Theorem

- (1) A graph, G , is a tree if and only if G is acyclic and becomes not acyclic if an edge is added to it.
- (2) A graph, G , is a tree if and only if G is connected and becomes not connected if an edge is removed from it.

Eg (17)

Illustrate the 2 statements above using some sketch diagrams.

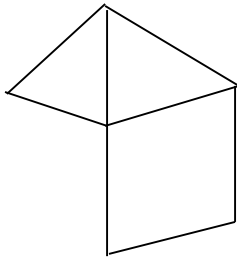


Spanning tree

Suppose R is a symmetric relation represented by a connected graph, G . Suppose some edges of R are removed until the resulting graph is a tree, T . T is called a spanning tree of R .

Eg (18)

For the symmetric relation R whose graph is drawn below, obtain some spanning trees.

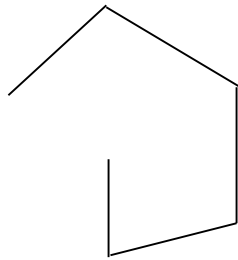


Solution:

If for each edge, both directions are included, we have an undirected spanning tree.

Eg (19):

We can build a spanning tree for a connected simple graph using a depth-first search and breadth-first search.

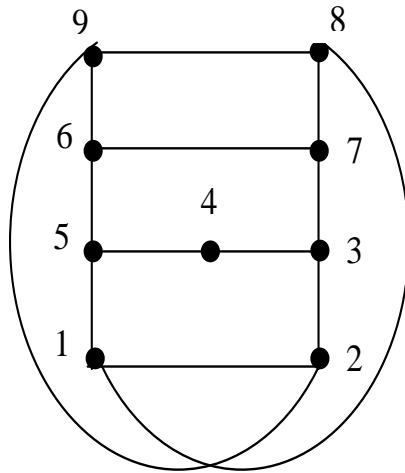


Depth-first Search

- Arbitrarily choose a vertex of the graph as the root.
- Form a path starting at this vertex by successively adding edges, where each new edge is incident with the last vertex in the path and a vertex not already in the path.
- Continue adding edges to this path as long as possible.
- If the path goes through all vertices of the graph, the tree consisting of this path is a spanning tree.
- If the path does not go through all vertices, more edges must be added.
- Move back to the next to last vertex in the path, and, if possible, form a new path starting at this vertex passing through vertices that were not already visited.
- Repeat this procedure, beginning at the last vertex visited, moving back up the path one vertex at a time, forming new paths that are as long as possible until no more edges can be added.
- Since the graph has a finite number of edges and is connected, this process ends with the production of a spanning tree.

Eg (20):

Construct the adjacency lists for the graph below by listing the adjacencies of each vertex in increasing order. Hence, use depth first search to produce a spanning tree by choosing 1 as the root of the spanning tree.



Solution:

Vertices	List of adjacency
1	
2	
3	
4	
5	
6	
7	
8	
9	

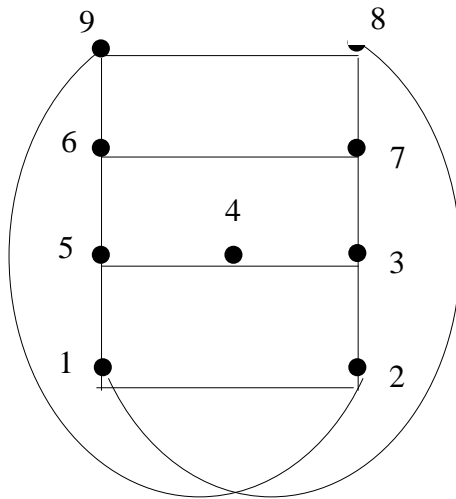
Depth first search spanning tree

Breadth-first Search

- Arbitrarily choose a root from the vertices of the graph. Add all edges incident to this vertex.
- The new vertices added at this stage become the vertices at level 1 in the spanning tree.
- For each vertex at level 1, visited in order, add each edge incident to this vertex to the tree as long as it does not produce a simple circuit.
- Arbitrarily order the children of each vertex at level 1.
- This produce the vertices at level 2 in the tree.
- Repeat the same proceduce until all the vertices in the tree have been added.
- A spanning tree is produced since we have produced a tree containing every vertex of the graph.

Eg (21):

Construct the adjacency lists for the graph below by listing the adjacencies of each vertex in increasing order. Hence, use breadth-first search to produce a spanning tree by choosing 1 as the root of the spanning tree.



Solution:

Vertices	List of adjacency
1	2 5 8
2	1 3 9
3	2 4 7
4	3 5
5	1 4 6
6	5 7 9
7	3 6 8
8	1 7 9
9	2 6 8

Breadth first search spanning tree

Solution:

Vertices	List of adjacency
1	2 5 8
2	1 3 9
3	2 4 7
4	3 5
5	1 4 6
6	5 7 9
7	3 6 8
8	1 7 9
9	2 6 8

Minimum Spanning Tree

A weighted graph is a graph for which each edge has an associated real number weight. The sum of the weights of all the edges is the total weight of the graph. A minimum spanning tree for a connected weighted graph is a spanning tree that has the least possible total weight compared to all other spanning trees for the graph. Prim's algorithm and Kruskal's algorithm are two efficient algorithms to construct minimum spanning trees.

Prim's algorithm

A vertex u is called a nearest neighbour of vertex v if u and v are adjacent (connected directly) and no other vertex is joined to v by an edge of lesser weight than (u, v) .

Let V represent the set of vertices that are already connected at a certain stage of the algorithm, and E represent the set of edges that are included in the tree at that stage.

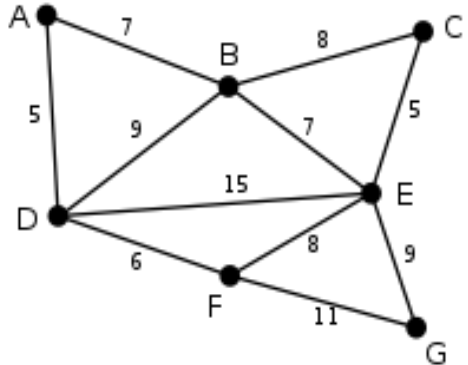
Step 1: Choose any vertex v_1 to start with. Let $V = \{v_1\}$ and $E = \{ \}$ initially.

Step 2: Choose a nearest neighbour v_i of V , which means v_i is adjacent to some member v_j of V and no other vertex is joined to a member of V by an edge of lesser weight. Add v_i to V and add (v_j, v_i) to E .

The v_i to be added to V should be chosen from outside V so that the newly added edge (v_j, v_i) does not form a cycle with edges that are already in E .

Repeat step 2 until $|E| = n - 1$, where n is the total number of vertices. Then all the vertices are connected and we have a minimal spanning tree.

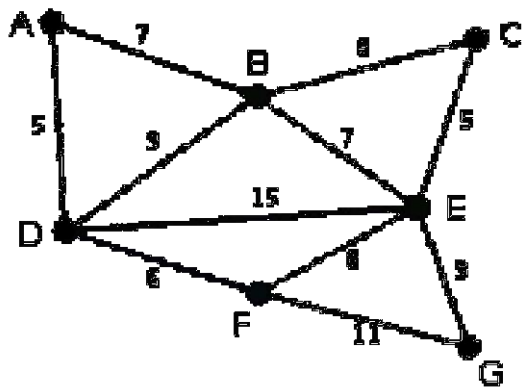
Example (22):

Image	Description
	<p>This is our original weighted graph. The numbers near the edges indicate their weight.</p>

	A	B	C	E	F	G
D						

Vertex **D** has been arbitrarily chosen as a starting point. Vertices **A**, **B**, **E** and **F** are connected to **D** through a single edge. **A** is the vertex nearest to **D** and will be chosen as the second vertex along with the edge **AD**

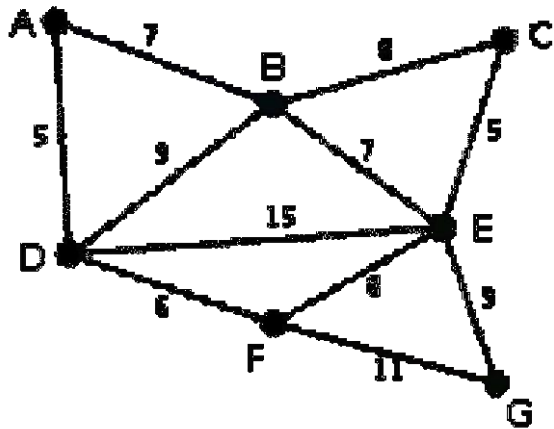
Weight AD =



	B	C	E	F	G
D					
A					

The next vertex chosen is the vertex nearest to *either* **D** or **A**. **B** is 9 away from **D** and 7 away from **A**, **E** is 15, and **F** is 6. **F** is the smallest distance away, so we highlight the vertex **F** and the arc **DF**.

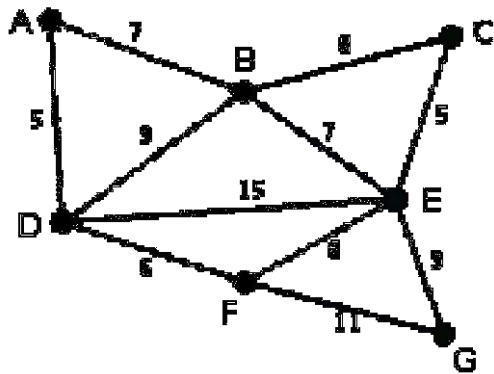
Weight DF =



	B	C	E	G
D				
A				
F				

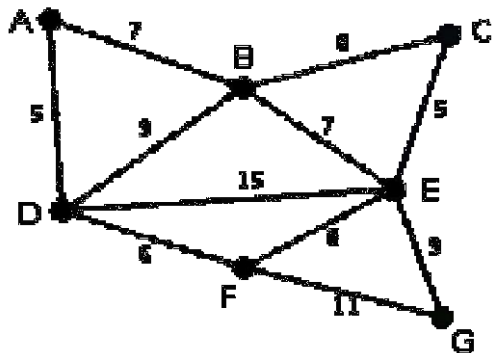
The algorithm carries on as above. Vertex **B**, which is 7 away from **A**, is highlighted.

Weight AB =



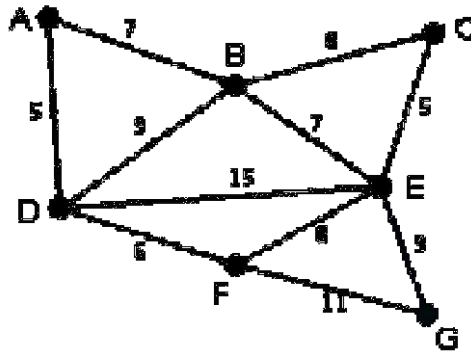
	C	E	G
D			
A			
F			
B			

In this case, we can choose between **C**, **E**, and **G**. **C** is 8 away from **B**, **E** is 7 away from **B**, and **G** is 11 away from **F**. **E** is nearest, so we highlight the vertex **E** and the arc **BE**.
Weight BE =



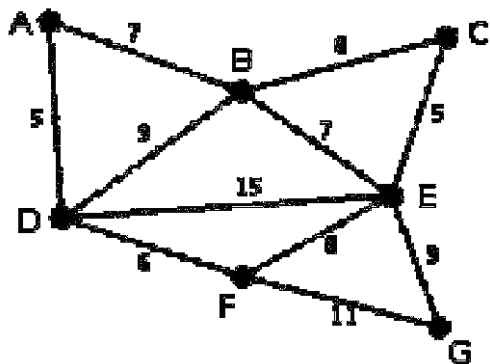
	C	G
D		
A		
F		
B		
E		

Here, the only vertices available are **C** and **G**. **C** is 5 away from **E**, and **G** is 9 away from **E**. **C** is chosen, so it is highlighted along with the arc **EC**.
Weight EC =



	G
D	
A	
F	
B	
E	
C	

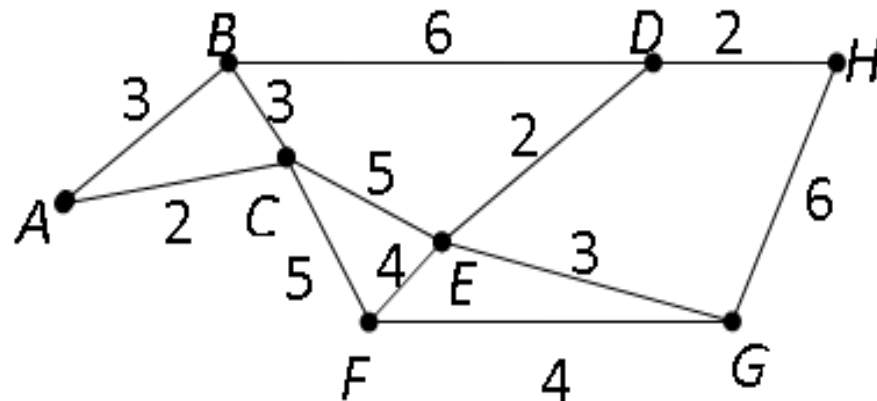
Vertex **G** is the only remaining vertex. It is 11 away from **F**, and 9 away from **E**. **E** is nearer, so we highlight it and the arc **EG**.
Weight EG =



Now all the vertices have been selected and the minimum spanning tree is shown in green. In this case, it has weight _____

Example (23):

A small town plans to pave some of the walking trails become bicycle paths. The town wants to link all the recreational areas with bicycle paths as cheaply as possible. Assuming that construction costs are the same on all parts of the system. The system is modelled by the weighted graph below, where the weights represent the distances in kilometres between sites. Use Prim's algorithm to find a plan for the town's paving.



Solution

Prim's algorithm

- The Prim's algorithm is used for obtaining a spanning tree with a specified root, v_o .
- Suppose a symmetric connected relation R is given. Represent R by a Boolean matrix. The intended root, v_o , is chosen to be at the first row and the first column.
- Let each diagonal element be 0. (There is no edge from a vertex to itself in a tree.)
- Look for a vertex connected to v_o , say v_j . Merge v_o and v_j (combine them and treat them as 1 vertex.)
- Rows and columns representing v_o and v_j are joined according to:

$$0 \vee 0 = \quad 0 \vee 1 = \quad 1 \vee 0 = \quad 1 \vee 1 =$$

and the result becomes the new v_o row & column.

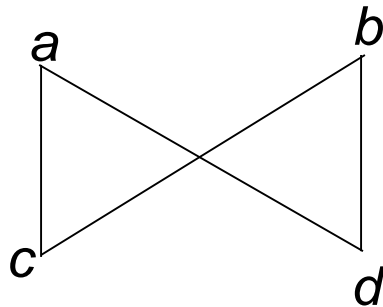
Delete row & column for v_j . Restore diagonal elements to 0.

Keep an edge of R that connects one of the vertices previously merged to v_o , to the newly merged v_j .

Repeat until all the vertices are merged into a single vertex.

Eg (24):

The following connected graph is given. Apply Prim's algorithm to obtain a spanning tree with vertex a as the root.



Solution:

$$\begin{matrix} & a & b & c & d \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \left(\begin{array}{cccc} & & & \end{array} \right) \end{matrix}$$

$$\begin{matrix} & \{a,c\} & b & d \\ \begin{matrix} \{a,c\} \\ b \\ d \end{matrix} & \left(\begin{array}{ccc} & & \end{array} \right) \end{matrix}$$

$$\begin{matrix} & \{a,c,b\} & d \\ \{a,c,b\} & \left(\right. & \\ d & & \end{matrix}$$

Matrix of spanning tree.

$$\begin{matrix} & a & b & c & d \\ a & \left(\right. & & & \\ b & & \left(\right. & & \\ c & & & \left(\right. & \\ d & & & & \left(\right. \end{matrix}$$

OR

$$\begin{matrix} & a & b & c & d \\ a & \left(\right. & & & \\ b & & \left(\right. & & \\ c & & & \left(\right. & \\ d & & & & \left(\right. \end{matrix}$$

Kruskal's Algorithm

Let R be a symmetric, connected relation with n vertices and let $S = \{e_1, e_2, \dots, e_k\}$ be the set of weighted edges of R .

Step 1 Choose an edge e_1 in S of least weight. Let $E = \{e_1\}$. Replace S with $S - \{e_1\}$.

Step 2 Select an edge e_i in S of least weight that will not make a cycle with members of E . Replace E with $E \cup \{e_i\}$ and S with $S - \{e_i\}$.

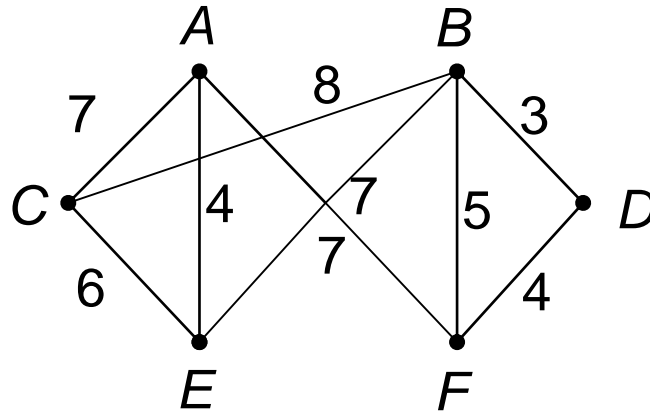
Step 3 Repeat Step 2 until $|E| = n - 1$.

Since R has n vertices, the $n - 1$ edges in E will form a spanning tree. The selection process in Step 2 guarantees that this is a minimum spanning tree.

The weight of a minimum spanning tree is unique, but the minimum spanning tree itself is not. Different minimum spanning trees can occur when two or more edges have the same weight.

Example (25):

Find a minimum spanning tree of the weighted graph below by using Kruskal's Algorithm.



Note that the graph above has six vertices, so a minimum spanning tree will have five edges.

Arrange the edges by increasing weights, and then successively add edges without forming any cycles, five edges are included. This yields the following data:

Edges	<i>BD</i>	<i>AE</i>	<i>DF</i>	<i>BF</i>	<i>CE</i>	<i>AC</i>	<i>AF</i>	<i>BE</i>	<i>BC</i>
Weight	3	4	4	5	6	7	7	7	8
Delete?									

Thus the minimum spanning tree which is obtained contains the edges *BD*, *AE*, *DF*, *CE*, *AF*.