Graph and Tree Theory Notes

Matt Warner

Contents

1 Trails, Paths, and Circuits

Definition

Let G be a graph, and let v and w be vertices in G.

A walk from v to w is a finite alternating sequence of adjacent vertices and edges of G. Thus a walk has the form

$$v_0 e_1 v_1 e_2 \cdots v_{n-1} e_n v_n$$

where the v 's represent vertices, the e 's represent edges, $v_0 = v, v_n = w$, and for each $i = 1, 2, \ldots n, v_{i-1}$ and v_i are the endpoints of e_i. The trivial walk from v to v consists of the single vertex v.

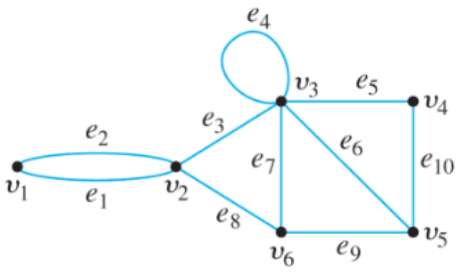
- A trail from v to w is a walk from v to w that does not contain a repeated edge.
- A path from v to w is a trail that does not contain a repeated vertex.
- A closed walk is a walk that starts and ends at the same vertex.
- A circuit is a closed walk that contains at least one edge and does not contain a repeated edge.
- A simple circuit is a circuit that does not have any other repeated vertex except the first and last.

	Repeated Edge?	**Repeated Vertex?**	**Starts and Ends at Same Point?**	**Must Contain at Least One Edge?**
Walk	allowed	allowed	allowed	no
Trail	no	allowed	allowed	no
Path	no	no	no	no
Closed Walk	allowed	allowed	yes	no
Circuit	no	allowed	yes	yes
Simple Circuit	no	first and last only	yes	yes

Example 1.1

Determine which of the following walks are trails, paths, circuits, or simple circuits.

1. $v_1, e_1, v_2, e_3, v_3, e_4, v_3, e_5, v_4$
2. e_1, e_3, e_5, e_5, e_6
3. $v_2 v_3 v_4 v_5 v_3 v_6 v_2$
4. $v_1 e_1 v_2 e_1 v_1$
5. $v_2, v_3, v_4, v_5, v_6, v_2$
6. v_1



Soultion: ☺

1. this walk is a Trail, since it does not contain any repeted edges.
2. This is just a walk and nothing else.
3. This walk is a Closed walk and is also a circuit, but not a simple circuit.
4. This is just a closed walk, it cant be a circuit since there is a repeated edge.
5. This is a closed walk, as well as a simple circuit.
6. This is a closed walk, as well as a trail, not a circuit becuase the walk does not contain any edges.

2 Subgraphs

Definition

A graph H is said to be a subgraph of a graph G if, and only if, every vertex in H is also a vertex in G, every edge in H is also an edge in G, and every edge in H has the same endpoints as it has in G.

Example 2.1

List all subgraphs of the graph G with vertex set $\{v_1, v_2\}$ and edge set $\{e_1, e_2, e_3\}$, where the endpoints of e_1 are v_1 and v_2, the endpoints of e_2 are v_1 and v_2, and e_3 is a loop at v_1.

G can be drawn as shown below.

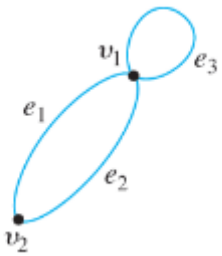


Figure 1: G

There are 11 subgraphs of G, which can be grouped according to those that do not have any edges, those that have one edge, those that have two edges, and those that have three edges.

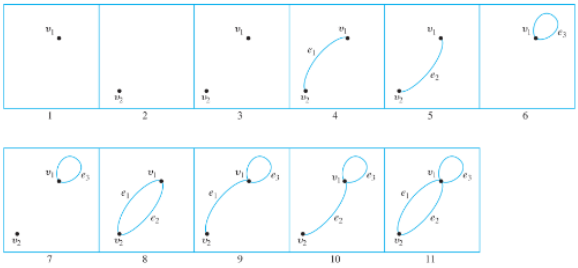


Figure 2: Subgraphs of G

3 Connectedness

Definition

Let G be a graph. Two **vertices, v and w of G are connected** if, and only if, there is a walk from v to w.

The **graph G is connected** if, and only if, given *any* two vertices v and w in G, there is a walk from v to w. Symbolically:

$$G \text{ is connected} \longleftrightarrow \forall \text{ vertices } v \text{ and } w \text{ in } G, \exists \text{ a walk from } v \text{ to } w.$$

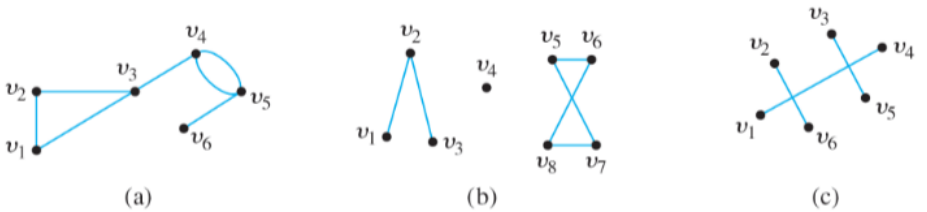
Note:-

If you take the negation of this definition, you will see that a graph G is *not connected* if, and only if, there exists two vertices of G that are not connected by any walk.

Example 3.1

Which of the following graphs are connected?

The graphs are listed below



Solution: ☺

Graph A is connected Graph B is not connected Graph C is also not connected

Some useful facts relating circuits and connectedness are collected in the following lenma.

Lenma 3.1

Let G be a graph.

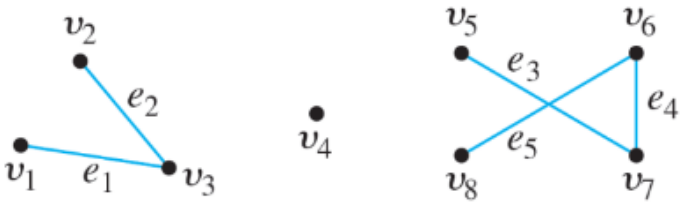
a. If G is connected, then any two distinct vertices of G can be connected by a path.

b. If vertices v and w are part of a circuit in G and one edge is removed from the circuit, then there still exists a trail from v to w in G.

c. If G is connected and G contains a circuit, then an edge of the circuit can be removed without disconnecting G.

Example 3.2

Find all connected components of the following graph G.



Solution: ☺

G has three connected components: H_1, H_2 and H_3 with vertex sets V_1, V_2, and V_3 and edge sets E_1, E_2, and E_3

$$\begin{aligned} V_1 &= \{v_1, v_2, v_3\}, & E_1 &= \{e_1, e_2\}, \\ V_2 &= \{v_4\}, & E_2 &= \varnothing, \\ V_3 &= \{v_5, v_6, v_7, v_8\}, & E_3 &= \{e_3, e_4, e_5\}. \end{aligned}$$

4 Euler Circuits

Definition

Let G be a graph. An Euler circuit for G is a circuit that contains every vertex and every edge of G. That is, an Euler circuit for G is a sequence of adjacent vertices and edges in G that has at least one edge, starts and ends at the same vertex, uses every vertex of G at least once, and uses every edge of G exactly once.

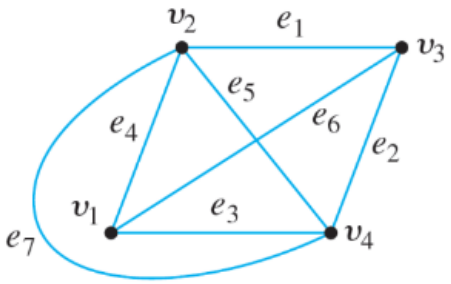
Theorem 4.1

If a graph has a Euler circuit, then every vertex of the graph has positive even degree.

If some vertex of a graph has odd degree, then the graph does not have a Euler circuit

Example 4.1

Show that the graph below does not have a Euler circuit.



Solution: ☺

The vertices v_1, and v_3 both have odd degrees (degree 3). So the graph cannot be a Euler circuit.

Note:-

If a graph G is connected and the degree of every vertex of G is a postive even integer, then G has a Euler circuit.

4.1 Euler Trails

Definition

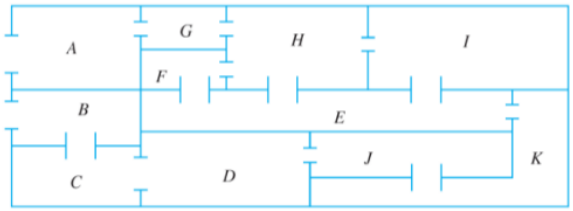
Let G be a graph, and let v and w be two distinct vertices of G. An Euler trail from v to w is a sequence of adjacent edges and vertices that starts at v, ends at w, passes through every vertex of G at least once, and traverses every edge of G exactly once.

Corollary 4.1

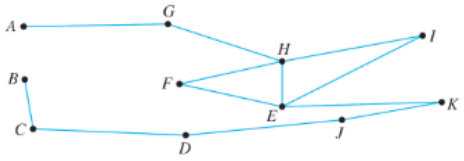
Let G be a graph, and let v and w be two distinct vertices of G. There is an Euler trail from v to w if, and only if, G is connected, v and w have odd degree, and all other vertices of G have positive even degree.

Example 4.2

The floor plan shown below is for a house that is open for public viewing. Is it possible to find a trail that starts in room A, ends in room B, and passes through every interior doorway of the house exactly once? If so, find such a trail.



We can represent this floor plan as a graph:



Each vertex of this graph has an even degree except for A and B, hence by Corollary 0.4.1 there is an Euler trail from A to B one such trail is

$$A - G - H - F - E - I - H - E - K - J - D - C - B$$

5 Hamiltonian Circuits

Definition

Given a graph G, a **Hamiltonian circuit** for G is a simple circuit that includes every vertex of G. That is, a Hamiltonian circuit for G is a sequence of adjacent vertices and distinct edges in which every vertex of G appears exactly once, except for the first and the last, which are the same.

Proposition 5.1

If a graph G has a Hamiltonian circuit, then G has a subgraph H with the following properties:

1. H contains every vertex of G.
2. H is connected.
3. H has the same number of edges as vertices.
4. Every vertex of H has degree 2 .

6 Matrices

Definition

Matrices are two-dimensional analogues of sequences.

They also are called two-dimensional arrays

An $m \times n$ (read " m by n ") matrix A over a set S is a rectangular array of elements of S arranged into m rows and n columns:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mj} & \cdots & a_{mn} \end{bmatrix} \leftarrow \text{ ith row of A}$$

We write $\mathbf{A} = (a_{ij})$

The ith row of A is

$$\begin{bmatrix} a_{i1} & a_{i2} & \ldots & a_{in} \end{bmatrix}$$

and the jth column of A is

$$\begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix}$$

A matrix for which the numbers of row and columns are equal is called a **square matrix**

If $\mathbf{A}$ is a square matrix of size $n \times n$, then the main diagonal of $\mathbf{A}$ consists of all the entries $a_{11}, a_{22}, \ldots, a_{nn}$

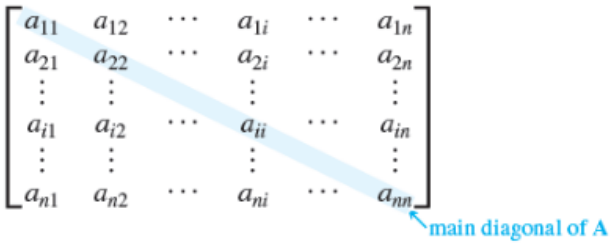


Figure 3: Square matrix diagonal

Example 6.1

The following is a 3 x 3 matrix over the set of integers.

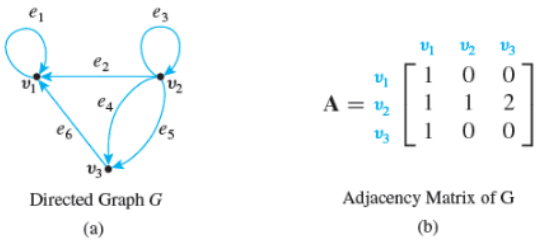
$$\mathbf{A} = \begin{bmatrix} 1 & 0 & -3 \\ 4 & -1 & 5 \\ -2 & 2 & 0 \end{bmatrix}$$

1. What is a_{23}, the entry in row 2 , column 3 ?
2. What is the second column of $\boldsymbol{A}$?
3. What are the entries in the main diagonal of $\mathbf{A}$?

Solution: ☺

1. $a_{23} = 5$
2. $\begin{bmatrix} 0 \\ -1 \\ 2 \end{bmatrix}$
3. $1, -1, 0$

6.1 Matrices and Directed Graphs



A Directed Graph and Its Adjacency Matrix

Figure 10.2.1

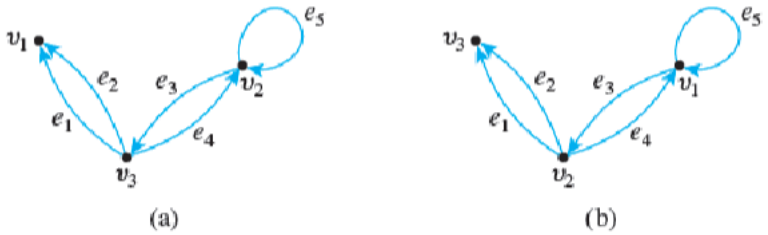
Note:-

As shown by the figure above, the adjacency matrix holds holds all the data for the directed graph, each entry in the matrix is either a 1 or 0, the entry is a 0 if the two vertices are not adjacent, and a 1 if they are adjacent, with the number increasing by the amount of edges that connect the vertices.

Example 6.2

The two graphs show below are identical and differ only in the ordering of their vertices.

Find their adjacency matrices



Solution: ☺

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 2 & 1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

6.2 Symmetric Matrices

Definition

An $n \times n$ square matrix $\mathbf{A} = (a_{ij})$ is called symmetric if, and only if, for every i and $j = 1, 2, \ldots, n$,

$$a_{ij} = a_{ji}$$

Example 6.3

Which of the following matrices are symmetric?

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & 0 \\ 2 & 0 & 3 \end{bmatrix} \qquad C = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Soultion: ☺

Only graph B is symmetric

In matrix A the entry in the first row and the second column differs from the entry in the second row and the first column; the matrix, C, is not even square.

6.3 Matrix Multiplication

The product of two matrices is built up of *scalar* or *dot* products of their individual rows and columns.

Definition

Suppose that all entries in matrices **A** and **B** are real numbers. If the number of element, n, in the i th row of **A** equals the number of elements in the j th column of **B**, then the scalar product or dot product of the i th row of **A** and j th column of **B** is the real number obtained as follows:

$$\begin{bmatrix} a_{i1} & a_{i2} & \cdots & a_{in} \end{bmatrix} \begin{bmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{nj} \end{bmatrix} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$$

Example 6.4

$$A = \begin{bmatrix} 2 & 0 & 3 \\ -1 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 4 & 3 \\ 2 & 2 \\ -2 & -1 \end{bmatrix}$$

solution: 😊

A has size 2×3 and **B** has size 3×2, so the number of columns of **A** equals the number of rows of **B** and the matrix product of **A** and **B** can be computed. Then

$$\begin{bmatrix} 2 & 0 & 3 \\ -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 4 & 3 \\ 2 & 2 \\ -2 & -1 \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

Where,

$$c_{11} = 2 \cdot 4 + 0 \cdot 2 + 3 \cdot (-2) = 2$$
$$c_{12} = 2 \cdot 3 + 0 \cdot 2 + 3 \cdot (-1) = 3$$
$$c_{21} = (-1) \cdot 4 + 1 \cdot 2 + 0 \cdot (-2) = -2$$
$$c_{22} = (-1) \cdot 3 + 1 \cdot 2 + 0 \cdot (-1) = -1$$

Hence,

$$AB = \begin{bmatrix} 2 & 3 \\ -2 & -1 \end{bmatrix}$$

7 Isomorphisms of Graphs

Definition

Two graphs that are the same except for the labeling of their vertices and edges are called *isomorphic*. The word isomorphism comes from the Greek, meaning "same form." Isomorphic graphs are those that have essentially the same form

Let G and G' be graphs with vertex sets $V(G)$ and $V(G')$ and edge sets $E(G)$ and $E(G')$, respectively. $\boldsymbol{G}$ is isomorphic to $\boldsymbol{G'}$ if, and only if, there exist one-to-one correspondences $g : V(G) \to V(G')$ and $h : E(G) \to E(G')$ that preserve the edge-endpoint functions of G and G' in the sense that for each $v \in V(G)$ and $e \in E(G)$, v is an endpoint of $e \Leftrightarrow g(v)$ is an endpoint of $h(e)$.

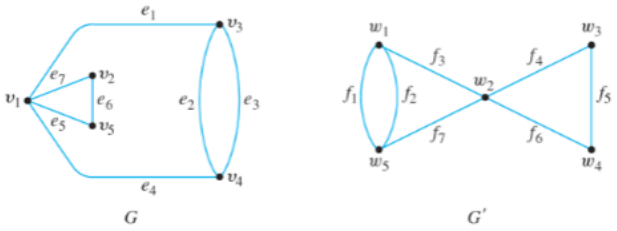
$$v \text{ is an endpoint of } e \Leftrightarrow g(v) \text{ is an endpoint of } h(e).$$

In other words,

G is isomorphic to G' if, and only if, the vertices and edges of G and G' can be matched up by one-to-one, onto functions in such a way that the edges between corresponding vertices correspond to each other.

Example 7.1

Show that the following two graphs are isomorphic.



Solution: 😄

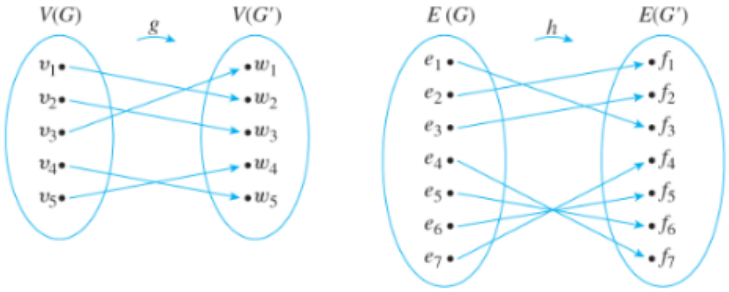
To solve this problem, you must find functions g: $V(G) \to V(G')$ and h: $E(G) \to E(G')$ such that for each $v \in V(G)$ and $e \in E(G)$, v is an endpoint of e if, and only if, $g(v)$ is an endpoint of $h(e)$

Note:-

Setting up such functions is partly a matter of trial and error and partly a matter of deduction.

For instance, since e_2 and e_3 are parallel (*have the same endpoint*), $h(e_2)$ and $h(e_3)$ must be parallel also.

One pair of functions for showing isomorphism between the two graphs is shown below



8 Trees

Definition

In mathematics, a tree is a connected graph that does not contain any circuits. Mathematical trees are similar in certain ways to their botanical namesakes.

A graph is said to be circuit-free if, and only if, it has no circuits. A graph is called a tree if, and only if, it is circuit-free and connected. A **trivial tree** is a graph that consists of a single vertex. A graph is called a forest if, and only if, it is circuit-free and not connected.

Example 8.1 (Trees and Non-trees)
All the graphs shown in figure 4 are trees, whereas those in figure 5 are not.

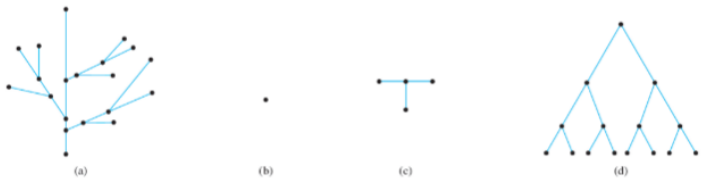


Figure 4: Trees

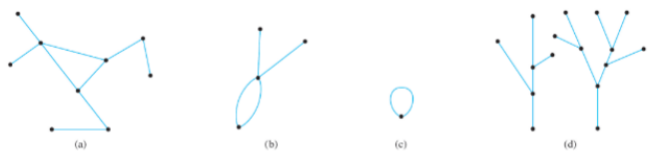


Figure 5: Non-Trees

Examples of Trees

Example 8.2 (A Decision Tree)

During orientation week, a college administers a mathematics placement exam to all entering students. The exam consists of two parts, and placement recommendations are made as indicated by the tree shown below in figure 6.

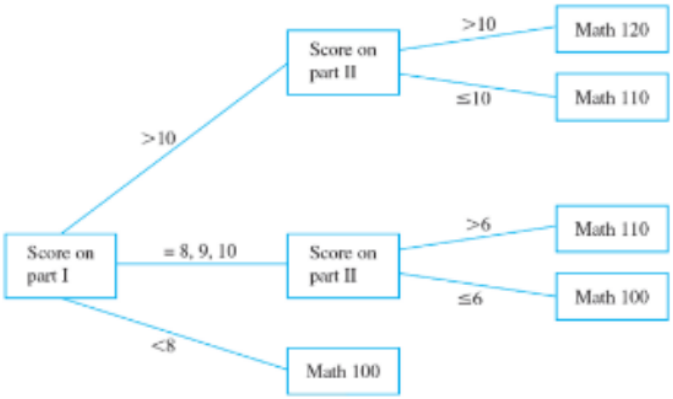


Figure 6

We read the tree from left to right to decide what course should be recommended for a student who scored 9 on part I and 7 on part II.

Since the student socred 9 on part I, the score on part II is checked.

Since it is greater than 6, the student should be advised to take Math 110.

8.1 Characterizing Trees

Defintion

There is a somewhat suprising relation between the number of vertices and the number of edges of a tree.

It turns out that if n is a postive integer, then any tree with n vertices (no matter what its shape) has $n-1$ edges.

Perhaps even more surprisingly, a partial converse to this fact is also true - namely, any connected graph with n vertices and $n-1$ edges is a tree.

It follows from these facts that if even one new edge (but no new vertex) is added to a tree, the resulting graph must contain a circuit.

Also, from the fact that removing an edge from a circuit does not disconnect a graph, it can be shown that every connected graph has a subgraph that is a tree.

It follows that if n is a positive integer, any graph with n vertices and *fewer* than $n-1$ edges is not connected.

Theorem 8.1

Any tree that has more than one vertex has at least one vertex of degree 1.

Theorem 8.2

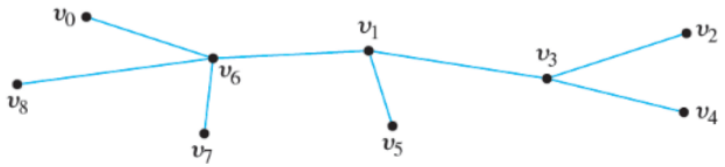
For any positive integer n, any tree with n vertices has $n-1$ edges.

Terminal Vertices

Let T be a tree. If T has at least two vertices, then a vertex of degree 1 in T is called a **leaf** (or a **terminal vertex**), and a vertex of degree greater than 1 in T is called an **internal vertex** (or a **branch vertex**). The unique vertex in a trivial tree is also called a **leaf** or **terminal vertex**.

Example 8.3

Find all leaves (or terminal vertices) and all internal (or branch) vertices in the following tree.



Solution: ☺

The terminal vertices are

$$v_0, v_2, v_4, v_5, v_7, v_8$$

The internal vertices are

$$v_6, v_1, v_3$$

Example 8.4

A graph G has ten vertices and twelve edges. Is it a tree?

Solution: ☺

No, By definition of Theorem 8.2, any tree with ten vertices has nine edges, not twelve.

8.2 Rooted Trees

Defintion

In mathematics, a rooted tree is a tree in which one vertex has been distinguished from the others and is designed the *root*. Given any other vertex v in the tree, there is a unique path from the root to v.

The number of edges in such a path is called the level of v, and the *height* of the tree is the length of the longest such path. It is traditional in drawing rooted trees to place the root at the top and show the branches descending from it.

A **rooted tree** is a tree in which there is one vertex that is distinguished from the others and is called the **root**.

The **level** of a vertex is the number of edges along the unique path between it and the root.

The **height** of a rooted tree is the maximum level of any vertex of the tree. Given the root or any internal vertex v of a rooted tree, the **children** of v are all those vertices that are adjacent to v and are one level farther away from the root than v.

If w is a child of v, then v is called the **parent** of w, and two distinct vertices that are both children of the same parent are called **siblings**. Given two distinct vertices v and w, if v lies on the unique path between w and the root, then v is an **ancestor** of w and w is a **descendant** of v.

These terms are illustrated in Figure 7

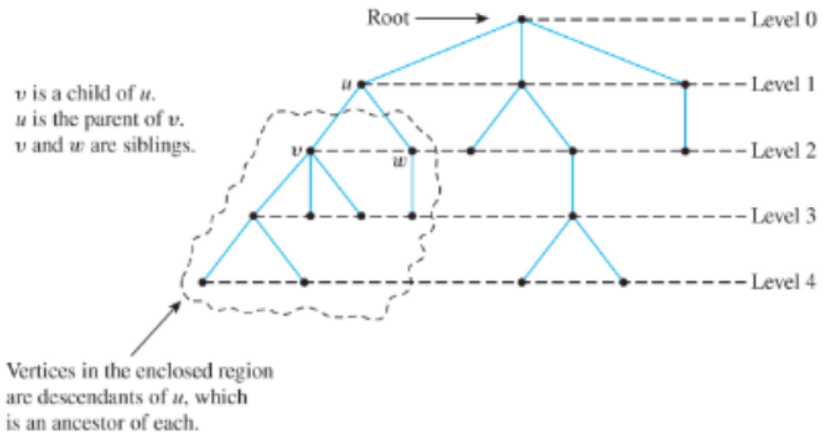
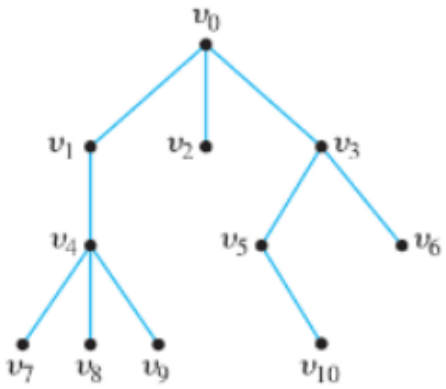


Figure 7: Rooted Tree

Example 8.5

Consider the tree with root v_0 shown below.

1. What is the level of v_5?
2. What is the level of v_0?
3. what is the height of this rooted tree?
4. What are the children of v_3
5. What is the parent of v_2
6. what are the siblings of v_8
7. What are the descendants of v_3?
8. How many terminal vertices are on the tree



Solution: ☺

The tree has a level of **2** since the number of edges from v_0 to v_5 is 2.

The level of v_0 is **0** since there are no edges between v_0 and itself.

the height of this rooted tree is **3** since the maximum level of the tree is 3.

The children of v_3 are v_5 and v_6

the parent of v_2 is v_0

the siblings of v_8 are v_7 and v_9

The descendants of v_3 are v_5, v_6, and v_10

There are 6 leaves (terminal vertices) on this rooted tree.

8.3 Binary Trees

Defintion

When every vertex in a rooted tree has at most two children and each child is designated either the (unique) left child or the (unique) right child, the result is a binary tree.

A **binary tree** is a rooted tree in which every parent has at most two children.

Each child in a binary tree is designated either a **left child** or a **right child** (but not both), and every parent has at most one left child and one right child.

A **full binary tree** is a binary tree in which each parent has exactly two children.

Given any parent v in a binary tree T, if v has a left child, then the **left subtree** of v is the binary tree whose root is the left child of v, whose vertices consist of the left child of v and all its descendants, and whose edges consist of all those edges of T that connect the vertices of the left subtree. The **right subtree** of v is defined analogously.

These terms are illustrated in Figure 8

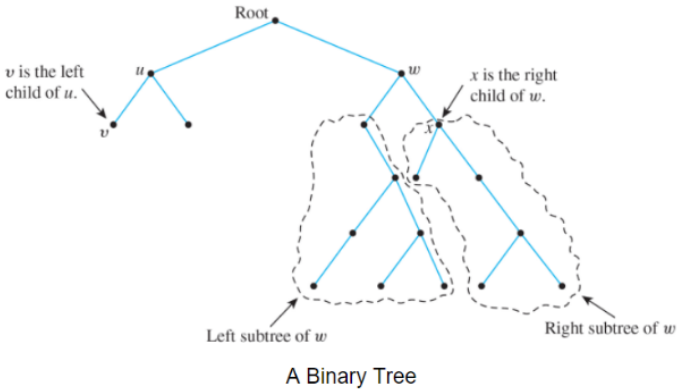


Figure 8: A Binary Tree

An interesting theorem about binary trees says that if you know the number of internal vertices of a full binary tree, then you can calculate both the total number of vertices and the number of leaves, and conversely. More specifically, a full binary tree with k internal vertices has a total of $2k + 1$ vertices of which $k + 1$ are leaves.

Theorem 8.3

If k is a positive integer and T is a full binary tree with k internal vertices, then (1) T has a total of $2k + 1$ vertices, and (2) T has $k + 1$ leaves.

Example 8.6 (Determining Whether a Certain Full binary Tree Exists)

Is there a full binary tree that has 10 internal vertices and 13 terminal vertices?

Solution: 😊

No. By Theorem 8.3, a full binary tree with 10 internal vertices has $10 + 1 = 11$ leaves, not 13.

Theorem 8.4

for every integer $h \geq 0$, if T is any binary tree with height h and t leaves, then

$$t \leq 2^h$$

Equivalently:

$$log_2 t \leq h$$

Example 8.7 (Determining whether a Certain Binary Tree Exists)

Is there a binary tree that has height 5 and 38 leaves?

Solution: 😊

No. By Theorem 8.4, a binary tree with height 5 and 38 leaves cannot exists because the height is greater than 2^5

Note:-

A full binary tree of height h has 2^h leaves.

8.4 Binary Search Trees

Defintion

A binary search tree is a kind of binary tree in which data records, such as customer information, can be stored, searched, and processed very efficiently. To place records into a binary search tree, it must be possible to arrange them in a total order.

In case they do not have a natural total order of their own, an element of a totally ordered set, such as a number or a word and called a **key**, may be added to each record. The keys are inserted into the vertices of the tree and provide access to the records to which they are attached.

Once it is built, a binary search tree has the following property:

for every internal vertex v, all the keys in the **left subtree** of v are **less** than the key in v, and all the keys in the **right subtree** of v are **greater** than the key in v.

For example, check that the following is a binary search tree for the set of records with the following keys:

$$15, 10, 19, 25, 12, 4$$

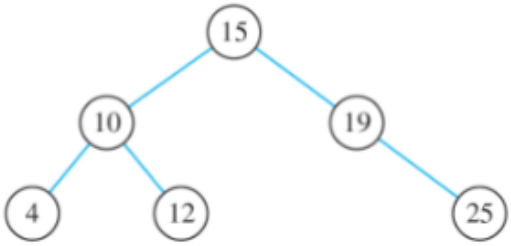


Figure 9: Binary Search Tree for the data listed above

To build a binary search tree, start by making a root and insert a key into it.

To add a new key, compare it to the key at the root. If the new key is less than the key at the root, give the root a left child and insert the new key into it.

If the key is greater than the key at the root, give the root a right child and insert the new key into it.

So to add a key at a subsequent stage, work down the tree to find a place to put the new key, starting at the root and either moving left or right depending on whether the new key is less or greater than the key at the vertex to which it is currently being compared.t

Algorithm 10.5.1 Building a Binary Search Tree

Input: A totally ordered, nonempty set K of keys

Algorithm Body:
Initialize T to have one vertex, the root, and no edges. Choose a key from K to insert into the root.

while (there are still keys to be added)
Choose a key, *newkey*, from K to add. Let the root be called v, let $key(v)$ be the key at the root, and let *success* = 0.
while (*success* = 0)
if (*newkey* < $key(v)$)
then if (v has a left child), call the left child v_L and let $v := v_L$
else do 1. add a vertex v_L to T as the left child for v
2. add an edge to T to join v to v_L
3. insert *newkey* as the key for v_L
4. let *success* := 1 **end do**

Algorithm 10.5.1 Building a Binary Search Tree

if (*newkey* > $key(v)$)
then if v has a right child
then call the right child v_R, and let $v := v_R$
else do 1. add a vertex v_R to T as the right child for v
2. add an edge to T to join v to v_R
3. insert *newkey* as the key for v_R
4. let *success* := 1 **end do**
end while
end while
Output: A binary search tree T for the set K of keys

Figure 10: Binary search tree algorithm

Example 8.8 (Building a Binary search tree)
Go through the steps to build a binary search tree for the keys $15, 10, 19, 25, 12, 4$, and insert the keys in the order in which they are listed. For simplicity, use the same names for the vertices and their associated keys.

Solution: ☺

Insert 15: Make 15 the root.

Insert 10: Compare 10 to 15.

Since $10 < 15$ and 15 does not have a left child, make 10 the left child of 15 and add an edge joining 15 and 10.

Insert 19: Compare 19 to 15.

Since $19 > 15$ and 15 does not have a right child, make 19 the right child of 15 and add an edge joining 15 and 19.

Insert 25: Compare 25 to 15.

Since $25 > 15$ and 15 has a right child, namely 19,

compare 25 to 19.

Since $25 > 19$ and 19 does not have a right child, make 25 the right child of 19 and add an edge joining 19 and 25.

Insert 12: Compare 12 to 15. Since $12 < 15$ and 15 has a left child, namely 10,

compare 12 to 10 . Since $12 > 10$ and 10 does not have a right child, make 12 the right child of 10 and add an edge joining 10 and 12.

Insert 4: Compare 4 to 15.

Since $4 < 15$ and 15 has a left child, namely 10,

compare 4 to 10.

Since $4 < 10$ and 10 does not have a left child, make 4 the left child of 10 and add an edge joining 10 and 4.

The sequence of steps is shown in the following diagrams

Figure 11: Steps for building the example tree

8.5 Spanning Trees

Defintion

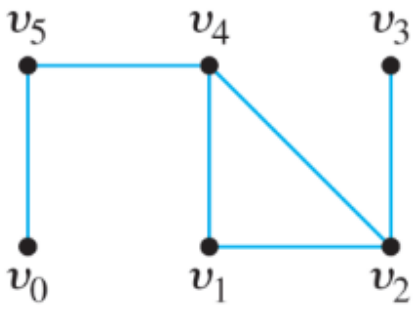
A **Spanning Tree** for a graph G is a subgraph of G that contains every vertex of G and is a tree.

Proposition 8.1

1. Every connected graph has a spanning tree.
2. Any two spanning trees for a graph have the same number of edges

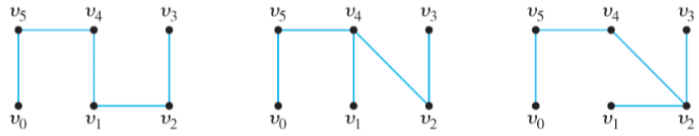
Example 8.9

Find all spanning trees for the graph G pictured below.



Solution: ☺

The Graph has one circuit (V_2, V_1, V_4, V_2), and removing any edge of the circuit gives a tree. Thus, as show below, there are three spanning trees for G.



8.6 Minimum Spanning Trees

The Graph of the routes allowed by the U.S Federal Aviation Authority shown in **figure 12** can be annotated by adding the distance (in miles) between each pair of cites.

Figure 12

Thus,

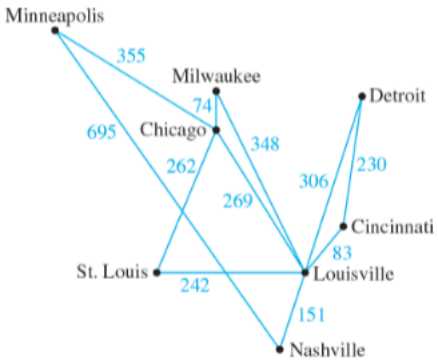


Figure 13
Graph with distances

Now suppose the airline company wants to serve all the cities shown, but with a route system that minimized the total mileage of the system as a whole.

Note:-

Note that such a system is a tree, because if the system contained a circuit, removal of an edge from the circuit would not affect a person's ability to reach every city in the system from every other, but it would reduce the total mileage of the system.

More generally, a graph whose edges are labeled with numbers (known as weights) is called a weighted graph. A minimum-weight spanning tree, or simply a minimum spanning tree, is a spanning tree for which the sum of the weights of all the edges is as small as possible.

Defintion

A **weighted graph** is a graph for which each edge has an associated positive 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 weight compared to all other spanning trees for the graph.

If G is a weighted graph and e is an edge of G, then $w(e)$ denotes the weight of e and $w(G)$ denotes the total weight of G

8.7 Kruskal's Algorithm

Defintion

In Kruskal's algorithm, the edges of a connected, weighted graph are examined one by one in order of increasing weight. At each stage the edge being examined is added to what will become the minimum spanning tree, provided that this addition does not create a circuit.

After $n-1$ edges have been added (where n is the number of vertices of the graph), these edges, together with the vertices of the graph, form a minimum spanning tree for the graph.

Algorithm 10.6.1 Kruskal

Input: *G [a connected, weighted graph with n vertices, where n is a positive integer]*
Algorithm Body:
[Build a subgraph T of G to consist of all the vertices of G with edges added in order of increasing weight. At each stage, let m be the number of edges of T.]

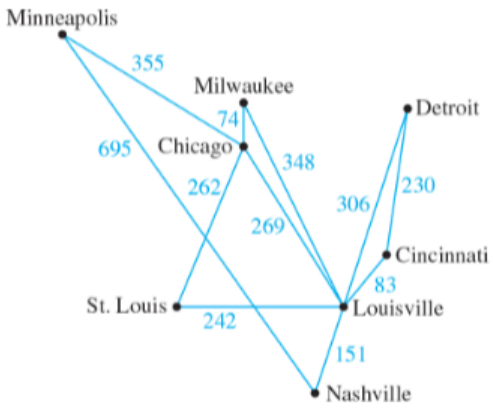
1. Initialize T to have all the vertices of G and no edges.
2. Let E be the set of all the edges of G, and let $m := 0$.
3. **while** $(m < n-1)$
   - 3a. Find an edge e in E of least weight.
   - 3b. Delete e from E.
   - 3c. **if** addition of e to the edge set of T does not produce a circuit **then** add e to the edge set of T and set $m := m+1$

   end while

Output: *T [T is a minimum spanning tree for G.]*

Figure 14
Kruskal's Algorithm

Example 8.10 (Actions of Kruskal's Algorithm)
Describe the action of Kruskal's algorithm on the graph shown in the figure below, where $n = 8$



Solution: ☺

Iteration Number	Edge Considered	Weight	Action Taken
1	Chicago-Milwaukee	74	added
2	Louisville-Cincinnati	83	added
3	Louisville-Nashville	151	added
4	Cincinnati-Detroit	230	added
5	St. Louis-Louisville	242	added
6	St. Louis-Chicago	262	added
7	Chicago-Louisville	269	not added
8	Louisville-Detroit	306	not added
9	Louisville-Milwaukee	348	not added
10	Minneapolis-Chicago	355	added

The tree produced by Kruskal's algorithm is shown in the figure below

Figure 15

Note:-

When a connected, weighted graph is input to Kruskal's algorithm, the output is a minimum spanning tree.

8.8 Prim's Algorithm

Defintion

Prim's algorithm works differently from Kruskal's. It builds a minimum spanning tree T by expanding outward in connected links from some vertex. One edge and one vertex are added at each stage.

The edge added is the one of least weight that connects the vertices already in T with those not in T, and the vertex is the endpoint of this edge that is not already in T.

Algorithm 10.6.1 Kruskal

Input: *G [a connected, weighted graph with n vertices, where n is a positive integer]*

Algorithm Body:

[Build a subgraph T of G to consist of all the vertices of G with edges added in order of increasing weight. At each stage, let m be the number of edges of T.]

1. Initialize T to have all the vertices of G and no edges.
2. Let E be the set of all the edges of G, and let $m := 0$.
3. **while** $(m < n - 1)$
   - 3a. Find an edge e in E of least weight.
   - 3b. Delete e from E.
   - 3c. **if** addition of e to the edge set of T does not produce a circuit **then** add e to the edge set of T and set $m := m + 1$

   end while

Output: *T [T is a minimum spanning tree for G.]*

Example 8.11 (Action of Prim's Algorithm)

Describe the action of Prim's algorithm for the graph in Figure using the Minneapolis vertex as a starting point.

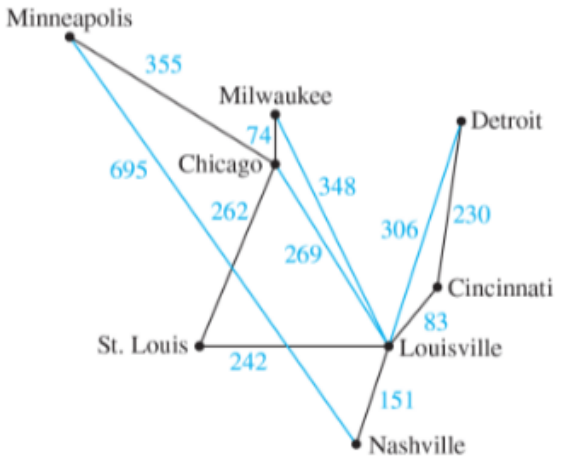


Figure 16: Prims Algorithm

Note:-

The black lines shown in the figure create the minimum spanning tree.

Steps for figure 16

Iteration Number	Vertex Added	Edge Added	Weight
0	Minneapolis		
1	Chicago	Minneapolis-Chicago	355
2	Milwaukee	Chicago-Milwaukee	74
3	St. Louis	Chicago-St. Louis	262
4	Louisville	St. Louis-Louisville	242
5	Cincinnati	Louisville-Cincinnati	83
6	Nashville	Louisville-Nashville	151
7	Detroit	Cincinnati-Detroit	230

8.9 Dijkstra's Shortest Path Algorithm

Defintion

Although the trees produced by Kruskal's and Prim's algorithms have the least possible total weight compared to all other spanning trees for the given graph, they do not always reveal the shortest distance between any two points on the graph.

In 1959 the computing pioneer, Edsger Dijkstra developed an algorithm to find the shortest path between a starting vertex and an ending vertex in a weighted graph in which all the weights are positive.

It is somewhat similar to Prim's algorithm in that it works outward from a starting vertex a, adding vertices and edges one by one to construct a tree T.

However, it differs from Prim's algorithm in the way it chooses the next vertex to add, ensuring that for each added vertex v, the length of the shortest path from a to v has been identified.

Algorithm 10.6.3 Dijkstra

Input: *G [a connected simple graph with a positive weight for every edge], ∞ [a number greater than the sum of the weights of all the edges in the graph], w(u, v) [the weight of edge {u, v}], a [the starting vertex], z [the ending vertex]*

Algorithm Body:

1. Initialize T to be the graph with vertex a and no edges. Let $V(T)$ be the set of vertices of T, and let $E(T)$ be the set of edges of T.
2. Let $L(a) = 0$, and for all vertices in G except a, let $L(u) = \infty$.
   [The number L(x) is called the label of x.]
3. Initialize v to equal a and F to be $\{a\}$.
   [The symbol v is used to denote the vertex most recently added to T.]

Figure 17: Dijkstra's Shortest Path Algorithm (1)

Algorithm 10.6.3 Dijkstra

4. while $(z \notin V(T))$

4a. $F := (F - \{v\}) \cup \{\text{vertices that are adjacent to } v \text{ and are not in } V(T)\}$
[The set F is called the fringe. Each time a vertex is added to T, it is removed from the fringe and the vertices adjacent to it are added to the fringe if they are not already in the fringe or the tree T.]

4b. For each vertex u that is adjacent to v and is not in $V(T)$,
if $L(v) + w(v, u) < L(u)$ **then**

$$L(u) := L(v) + w(v, u)$$
$$D(u) := v$$

[Note that adding v to T does not affect the labels of any vertices in the fringe F except those adjacent to v. Also, when L(u) is changed to a smaller value, the notation D(u) is introduced to keep track of which vertex in T gave rise to the smaller value.]

4c. Find a vertex x in F with the smallest label
Add vertex x to $V(T)$, and add edge $\{D(x), x\}$ to $E(T)$
$v := x$ *[This statement sets up the notation for the next iteration of the loop.]*

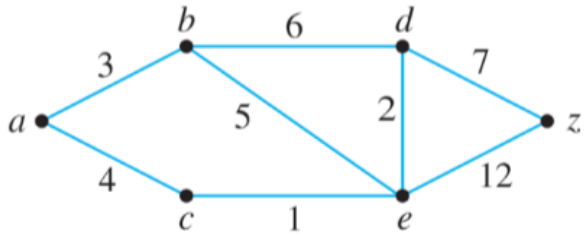
end while

Output: $L(z)$ *[L(z), a nonnegative integer, is the length of the shortest path from a to z.]*

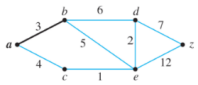
Figure 18: Dijkstra's Shortest Path Algorithm (2)

Example 8.12 (Actions of Dijkstra's Algorithm)

Show the steps in the execution of Dijktra's shortest path algorithm for the graph shown below with starting vertex a and ending vertex z

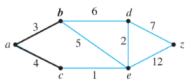


Step 1: Going into the **while** loop: $V(T) = \{a\}$, $E(T) = \emptyset$, and $F = \{a\}$



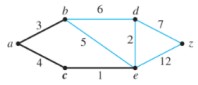
During iteration:
$F = \{b, c\}$, $L(b) = 3$, $L(c) = 4$.
Since $L(b) < L(c)$, b is added to $V(T)$, $D(b) = a$, and $\{a, b\}$ is added to $E(T)$.

Step 2: Going into the **while** loop: $V(T) = \{a, b\}$, $E(T) = \{\{a, b\}\}$



During iteration:
$F = \{c, d, e\}$, $L(c) = 4$, $L(d) = 9$, $L(e) = 8$.
Since $L(c) < L(d)$ and $L(c) < L(e)$, c is added to $V(T)$, $D(c) = a$, and $\{a, c\}$ is added to $E(T)$.

Step 3: Going into the **while** loop: $V(T) = \{a, b, c\}$, $E(T) = \{\{a, b\}, \{a, c\}\}$



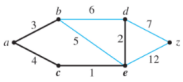
During iteration:

$F = \{d, e\}$, $L(d) = 9$, $L(e) = 5$

$L(e)$ becomes 5 because *ace*, which has length 5, is a shorter path to e than *abe*, which has length 8.

Since $L(e) < L(d)$, e is added to $V(T)$, $D(e) = c$, and $\{c, e\}$ is added to $E(T)$.

Step 4: Going into the **while** loop: $V(T) = \{a, b, c, e\}$, $E(T) = \{\{a, b\}, \{a, c\}, \{c, e\}\}$



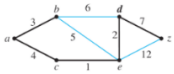
During iteration:

$F = \{d, z\}$, $L(d) = 7$, $L(z) = 17$

$L(d)$ becomes 7 because *aced*, which has length 7, is a shorter path to d than *abd*, which has length 9.

Since $L(d) < L(z)$, d is added to $V(T)$, $D(d) = e$, and $\{e, d\}$ is added to $E(T)$.

Step 5: Going into the **while** loop: $V(T) = \{a, b, c, e, d\}$, $E(T) = \{\{a, b\}, \{a, c\}, \{c, e\}, \{e, d\}\}$



During iteration:

$F = \{z\}$, $L(z) = 14$

$L(z)$ becomes 14 because *acedz*, which has length 14, is a shorter path to d than *abdz*, which has length 17.

Since z is the only vertex in F, its label is a minimum, and so z is added to $V(T)$, $D(z) = d$, and $\{d, z\}$ is added to $E(T)$.