10 Trees

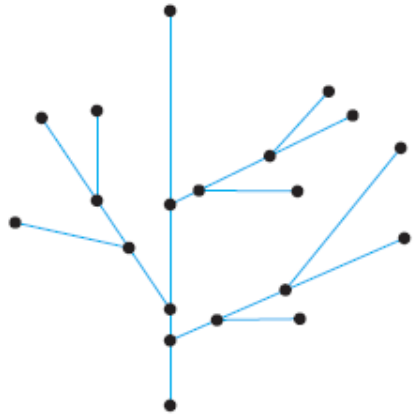
The core nonlinear abstract data structures of computer science

Trees

• Definition

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.

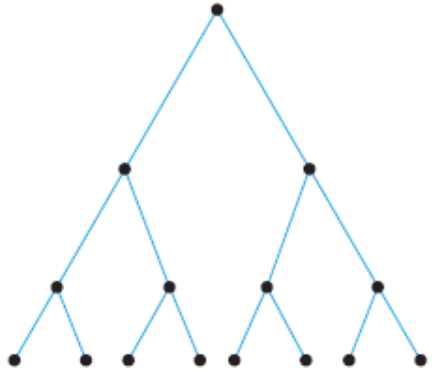
Trees: All are connected and circuit free



(a)

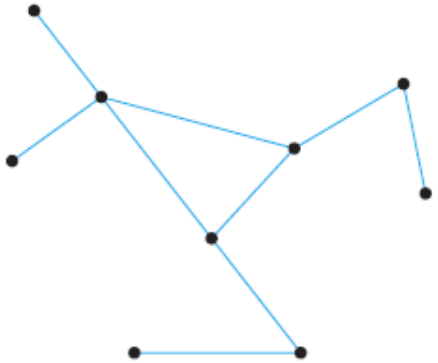
(b)

(c)

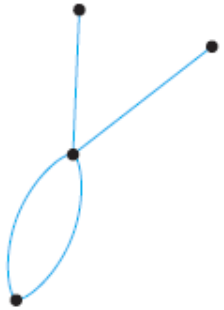


(d)

Non-Trees: (a), (b),(c) have circuits and (d) is not connected

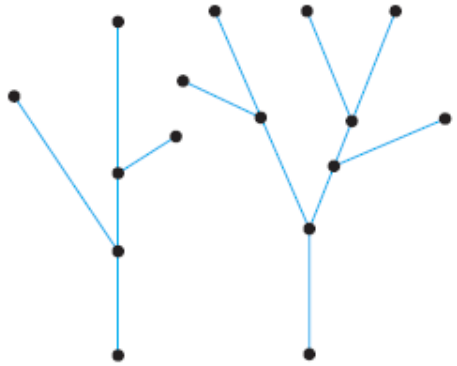


(a)



(b)

(c)



(d)

Example: Decision Tree

During orientation week, a college administers an exam to all entering students to determine placement in the mathematics curriculum. The exam consists of two parts, and placement recommendations are made as indicated by the tree shown in Figure 10.5.3. 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.

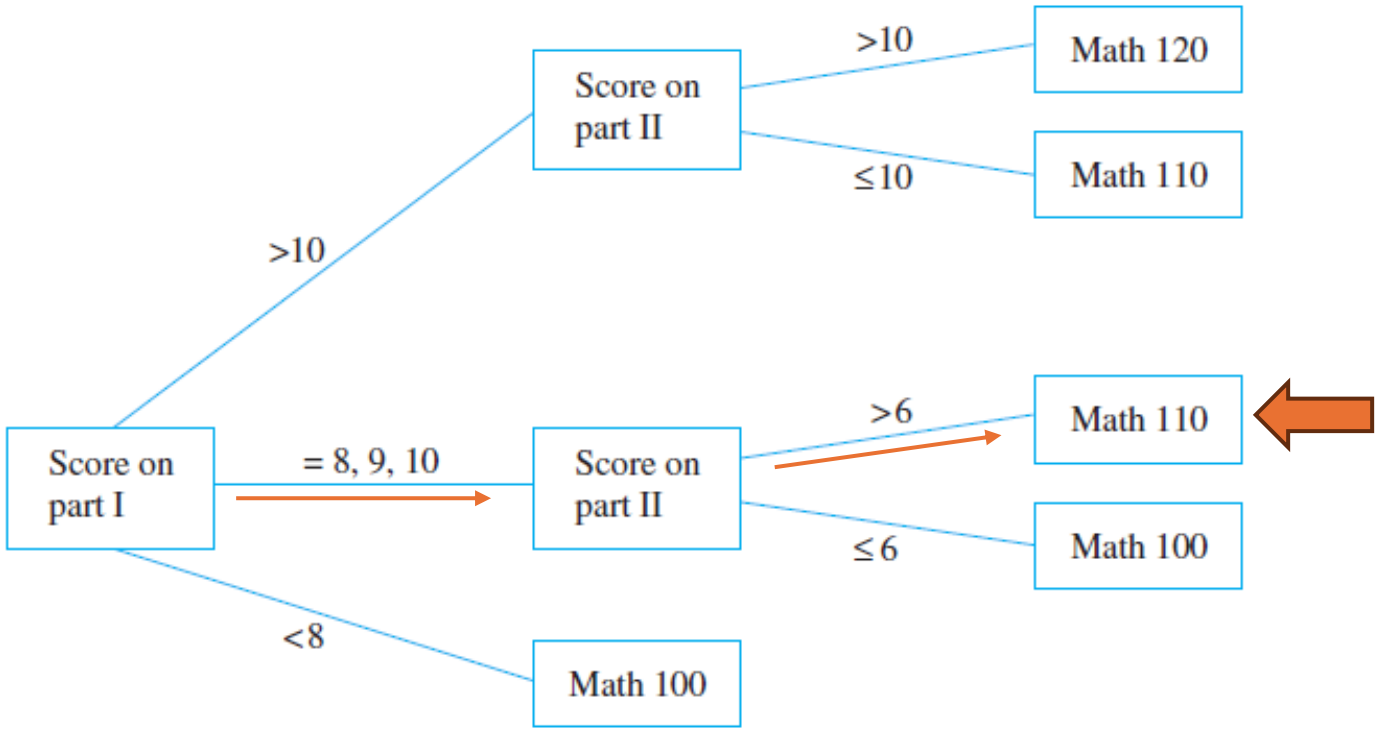


Figure 10.5.3

Example: Parse Tree

- In the study of grammars, trees are often used to show the derivation of grammatically correct sentences from certain basic rules.
- Such trees are called **syntactic derivation trees** or **parse trees.**
- In the study of linguistics, **syntax** refers to the grammatical structure of sentences, and **semantics** refers to the meanings of words and their interrelations.
- The rules of grammar are called **productions.**
- Production rules are then written into **shorthand notation using OR and Angle Brackets .**
- Useful in constructing compilers for high level computer languages.

A very small subset of English grammar, for example, specifies that

1. a sentence can be produced by writing first a noun phrase and then a verb phrase;
2. a noun phrase can be produced by writing an article and then a noun;
3. a noun phrase can also be produced by writing an article, then an adjective, and then a noun;
4. a verb phrase can be produced by writing a verb and then a noun phrase;
5. one article is “the”;
6. one adjective is “young”;
7. one verb is “caught”;
8. one noun is “man”;
9. one (other) noun is “ball.”

1. ⟨sentence⟩ → ⟨noun phrase⟩⟨verb phrase⟩

2., 3. ⟨noun phrase⟩ → ⟨article⟩⟨noun⟩ | ⟨article⟩⟨adjective⟩⟨noun⟩

4. ⟨verb phrase⟩ → ⟨verb⟩⟨noun phrase⟩

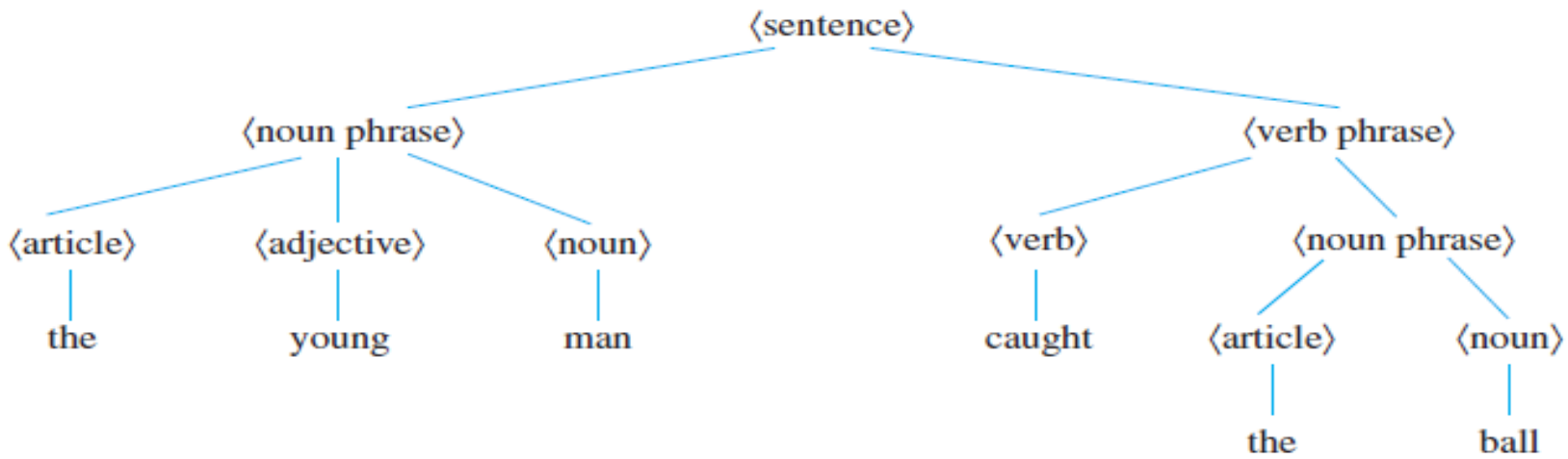
5. ⟨article⟩ → the

6. ⟨adjective⟩ → young

7, 8. ⟨noun⟩ → man | ball

9. ⟨verb⟩ → caught

The derivation of the sentence "The young man caught the ball" from the above rules is described by the tree shown below.



- A sentence can be syntactically correct but semantically incorrect, as the non sensical sentence.
- As the sentence derived from the rules previously discussed: **“The young ball caught the man”.**
- A sentence can be contain a syntactic errors but not semantic errors.
- As the sentence derived from the rules previously discussed: when a two-year-old child says, “Me hungry!”

Characterizing Trees

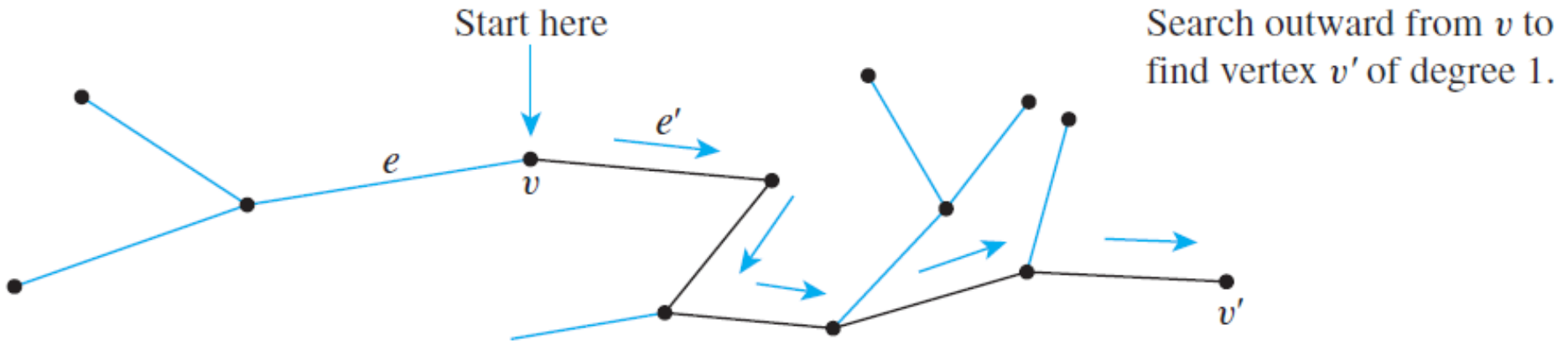
1. There is a somewhat surprising relation between the number of vertices and the number of edges of a tree.
2. It turns out that if n is a positive integer, then any tree with n vertices (no matter what its shape) has $n - 1$ edges.
3. Perhaps even more surprisingly, a partial converse to this fact is also true—namely, any connected graph with n vertices and $n - 1$ edges is a tree.
4. 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.
5. 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.

6. It follows that if n is a positive integer, any graph with n vertices and *fewer* than $n - 1$ edges is not connected.
7. A small but very important fact necessary to derive the first main theorem about trees is that any nontrivial tree must have at least one vertex of degree 1.
8. If we begin our study with the definition of a tree as a connected graph with no simple circuits, then a tree is a simple graph, and has a vertex of degree one.
9. The first theorem in this unit shows a simple graph with no circuits had at least one vertex of degree 1.

Characterizing Trees-Theorem 10.5.1

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

A constructive way to understand this lemma is to imagine being given a tree T with more than one vertex. You pick a vertex v at random and then search outward along a path from v looking for a vertex of degree 1. As you reach each new vertex, you check whether it has degree 1. If it does, you are finished. If it does not, you exit from the vertex along a different edge from the one you entered on. Because T is circuit-free, the vertices included in the path never repeat. And since the number of vertices of T is finite, the process of building a path must eventually terminate. When that happens, the final vertex v' of the path must have degree 1. This process is illustrated below.



Proof:

Let T be a particular but arbitrarily chosen tree that has more than one vertex, and consider the following algorithm:

Step 1: Pick a vertex v of T and let e be an edge incident on v.
[If there were no edge incident on v, then v would be an isolated vertex. But this would contradict the assumption that T is connected (since it is a tree) and has at least two vertices.]

Step 2a: Choose e' to be an edge incident on v such that $e' \neq e$. *[Such an edge exists because* $\deg(v) > 1$ *and so there are at least two edges incident on v.]*

Step 2b: Let v' be the vertex at the other end of e' from v. *[Since T is a tree, e' cannot be a loop and therefore e' has two distinct endpoints.]*

Step 2c: Let $e = e'$ and $v = v'$. *[This is just a renaming process in preparation for a repetition of step 2.]*

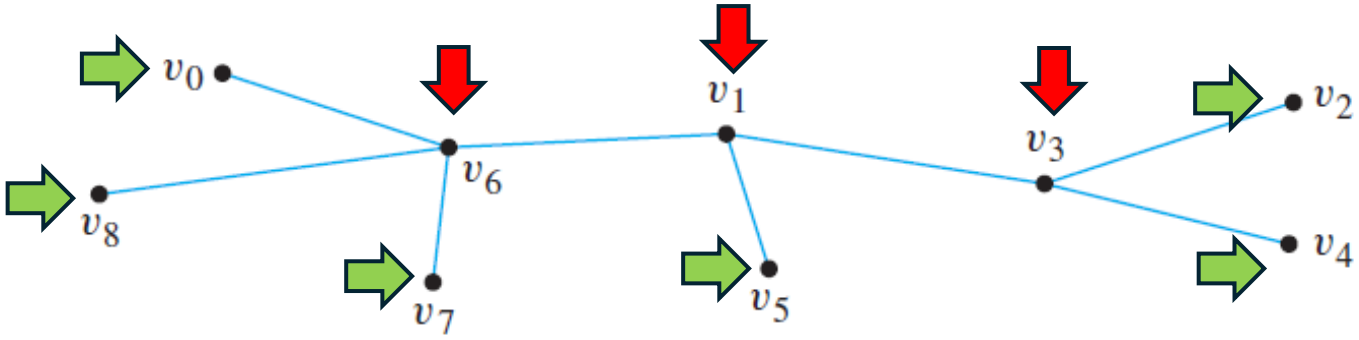
The algorithm just described must eventually terminate because the set of vertices of the tree T is finite and T is circuit-free. When it does, a vertex v of degree 1 will have been found.

Terminal and Internal Vertices

• Definition

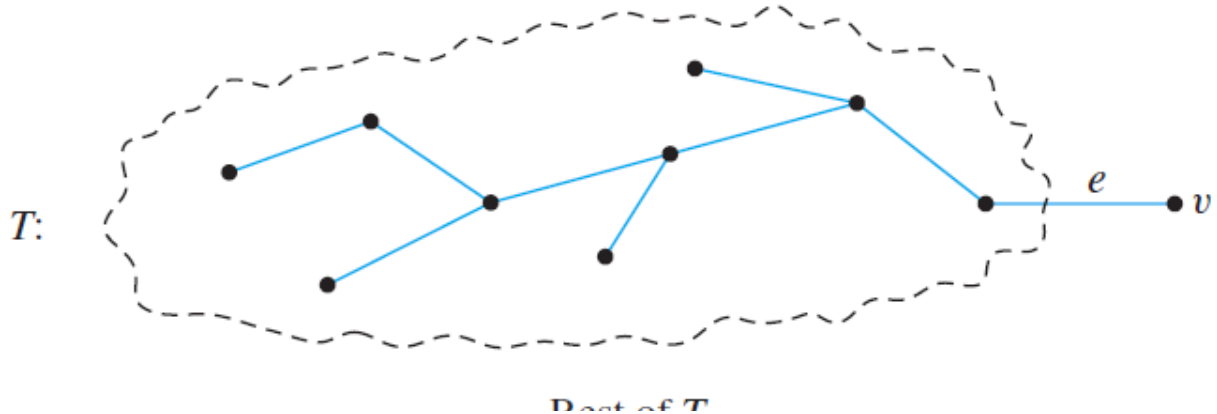
Let T be a tree. If T has only one or two vertices, then each is called a **terminal vertex.** If T has at least three vertices, then a vertex of degree 1 in T is called a **terminal vertex** (or a **leaf**), and a vertex of degree greater than 1 in T is called an **internal vertex** (or a **branch vertex**).

Terminal Vertices
Internal Vertices



Characterizing Trees-Theorem 10.5.2

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



Proof (by mathematical induction):

Let the property $P(n)$ be the sentence

Any tree with n vertices has $n - 1$ edges. $\leftarrow P(n)$

We use mathematical induction to show that this property is true for all integers $n \geq 1$.

Show that P(1) is true: Let T be any tree with one vertex. Then T has zero edges (since it contains no loops). But $0 = 1 - 1$, so $P(1)$ is true.

Show that for all integers $k \geq 1$, if $P(k)$ is true then $P(k+1)$ is true:
Suppose k is any positive integer for which $P(k)$ is true. In other words, suppose that

Any tree with k vertices has k - 1 edges. ← $P(k)$ inductive hypothesis

We must show that $P(k+1)$ is true. In other words, we must show that

Any tree with $k+1$ vertices has $(k+1)-1 = k$ edges. ← $P(k+1)$

Let T be a particular but arbitrarily chosen tree with $k+1$ vertices. *[We must show that T has k edges.]* Since k is a positive integer, $(k+1) \geq 2$, and so T has more than one vertex. Hence by Theorem 10.5.1, T has a vertex v of degree 1. Also, since T has more than one vertex, there is at least one other vertex in T besides v. Thus there is an edge e connecting v to the rest of T. Define a subgraph T' of T so that

$$V(T') = V(T) - \{v\}$$

Then

$$E(T') = E(T) - \{e\}.$$

1. The number of vertices of T' is $(k+1)-1=k$.
2. T' is circuit-free (since T is circuit-free, and removing an edge and a vertex cannot create a circuit).
3. T' is connected

Hence, by the definition of tree, T' is a tree. Since T' has k vertices, by inductive hypothesis

$$\begin{aligned}\text{the number of edges of } T' &= (\text{the number of vertices of } T') - 1\\ &= k-1.\end{aligned}$$

But then

$$\begin{aligned}\text{the number of edges of } T &= (\text{the number of edges of } T') + 1\\ &= (k-1)+1\\ &= k.\end{aligned}$$

[This is what was to be shown.]

Proof Summary

The proof is by mathematical induction. To do the inductive step, you assume the theorem is true for a positive integer k and then show it is true for $k + 1$. Thus you assume you have a tree T with $k + 1$ vertices, and you must show that T has $(k + 1) - 1 = k$ edges. As you do this, you are free to use the inductive hypothesis that *any* tree with k vertices has $k - 1$ edges. To make use of the inductive hypothesis, you need to reduce the tree T with $k + 1$ vertices to a tree with just k vertices. But by Theorem 10.5.1, T has a vertex v of degree 1, and since T is connected, v is attached to the rest of T by a single edge e as sketched in previous slide.

Now if e and v are removed from T, what remains is a tree T' with $(k + 1) - 1 = k$ vertices. By inductive hypothesis, then, T' has $k - 1$ edges. But the original tree T has one more vertex and one more edge than T'. Hence T must have $(k - 1) + 1 = k$ edges,

Characterizing Trees-Theorem 10.5.3

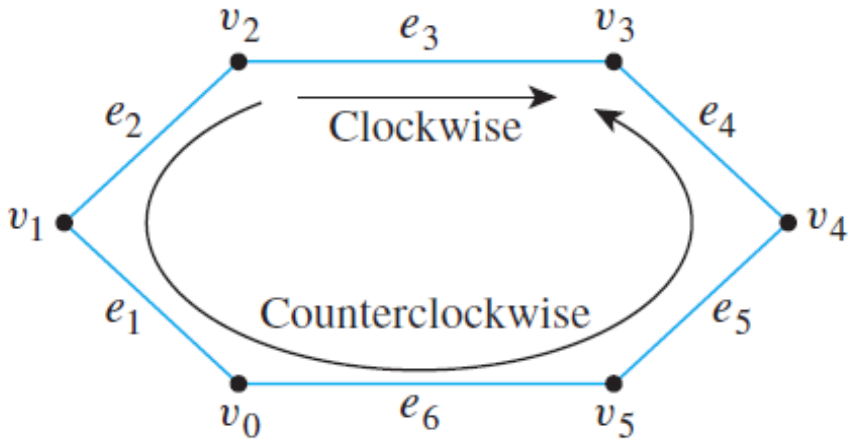
If G is any connected graph, C is any circuit in G, and any one of the edges of C is removed from G, then the graph that remains is connected.

Essentially, the reason why Theorem 10.5.3 is true is that any two vertices in a circuit are connected by two distinct paths. It is possible to draw the graph so that one of these goes "clockwise" and the other goes "counterclockwise" around the circuit. For example, in the circuit shown below , the clockwise path from v_2 to v_3 is

$$v_2 e_3 v_3$$

and the counterclockwise path from v_2 to v_3 is

$$v_2 e_2 v_1 e_1 v_0 e_6 v_5 e_5 v_4 e_4 v_3.$$



Proof:

Suppose G is a connected graph, C is a circuit in G, and e is an edge of C. Form a subgraph G' of G by removing e from G. Thus

$$V(G') = V(G)$$
$$E(G') = E(G) - \{e\}.$$

We must show that G' is connected. *[To show a graph is connected, we must show that if u and w are any vertices of the graph, then there exists a walk in G' from u to w.]* Suppose u and w are any two vertices of G'. *[We must find a walk from u to w.]* Since the vertex sets of G and G' are the same, u and w are both vertices of G, and since G is connected, there is a walk W in G from u to w.

Case 1 (e is not an edge of W): The only edge in G that is not in G' is e, so in this case W is also a walk in G'. Hence u is connected to w by a walk in G'.

Case 2 (e is an edge of W): In this case the walk W from u to w includes a section of the circuit C that contains e. Let C be denoted as follows:

$$C\colon v_0 e_1 v_1 e_2 v_2 \cdots e_n v_n\ (= v_0).$$

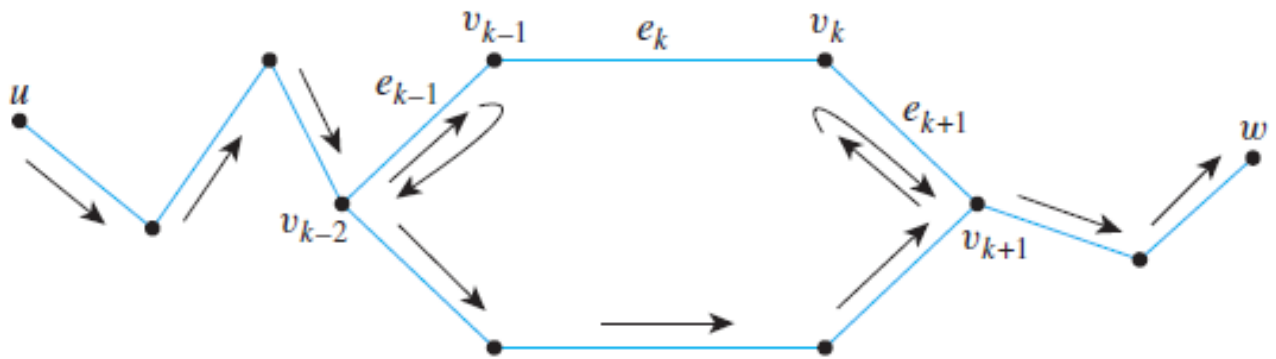
Now e is one of the edges of C, so, to be specific, let $e = e_k$. Then the walk W contains either the sequence

$$v_{k-1} e_k v_k \quad \text{or} \quad v_k e_k v_{k-1}.$$

If W contains $v_{k-1} e_k v_k$, connect v_{k-1} to v_k by taking the "counterclockwise" walk W' defined as follows:

$$W'\colon v_{k-1} e_{k-1} v_{k-2} \cdots v_0 e_n v_{n-1} \cdots e_{k+1} v_k.$$

An example showing how to go from u to w while avoiding e_k is given in Figure 10.5.4.



To go from u to w while avoiding e_k, go counterclockwise around the circuit.

Figure 10.5.4 An Example of a Walk from u to w That Does Not Include Edge e_k

If W contains $v_k e_k v_{k-1}$, connect v_k to v_{k-1} by taking the "clockwise" walk W'' defined as follows:

$$W'': v_k e_{k+1} v_{k+1} \cdots v_n e_1 v_1 e_2 \cdots e_{k-1} v_{k-1}.$$

Now patch either W' or W'' into W to form a new walk from u to w. For instance, to patch W' into W, start with the section of W from u to v_{k-1}, then take W' from v_{k-1} to v_k, and finally take the section of W from v_k to w. If this new walk still contains an occurrence of e, just repeat the process described previously until all occurrences are eliminated. *[This must happen eventually since the number of occurrences of e in C is finite.]* The result is a walk from u to w that does not contain e and hence is a walk in G'.

The previous arguments show that both in case 1 and in case 2 there is a walk in G' from u to w. Since the choice of u and w was arbitrary, G' is connected.

Characterizing Trees-Theorem 10.5.4

If any integer n, if G is a connected graph with n vertices and n-1 edges, then G is a tree.

Proof:

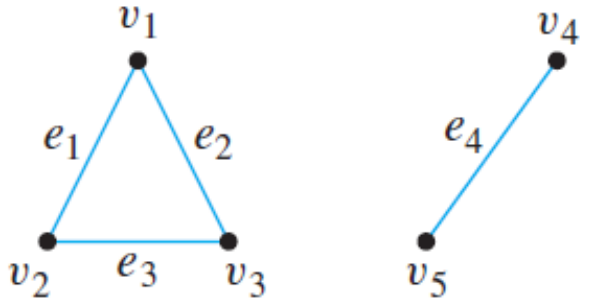
Let n be a positive integer and suppose G is a particular but arbitrarily chosen graph that is connected and has n vertices and $n - 1$ edges. *[We must show that G is a tree. Now a tree is a connected, circuit-free graph. Since we already know G is connected, it suffices to show that G is circuit-free.]* Suppose G is not circuit-free. That is, suppose G has a circuit C. *[We must derive a contradiction.]* By Lemma 10.5.3, an edge of C can be removed from G to obtain a graph G' that is connected. If G' has a circuit, then repeat this process: Remove an edge of the circuit from G' to form a new connected graph. Continue repeating the process of removing edges from circuits until eventually a graph G'' is obtained that is connected and is circuit-free. By definition, G'' is a tree. Since no vertices were removed from G to form G'', G'' has n vertices just as G does. Thus, by Theorem 10.5.2, G'' has $n - 1$ edges. But the supposition that G has a circuit implies that at least one edge of G is removed to form G''. Hence G'' has no more than $(n - 1) - 1 = n - 2$ edges, which contradicts its having $n - 1$ edges. So the supposition is false. Hence G is circuit-free, and therefore G is a tree *[as was to be shown]*.

- Theorem 10.5.4 is not a full converse of theorem 10.5.2
- Although it is true that every connected graph with n vertices and n − 1 edges (where n is a positive integer) is a tree, it is not true that every graph with n vertices and n − 1 edges is a tree.

3 A Graph with n Vertices and $n - 1$ Edges That Is Not a Tree

Give an example of a graph with five vertices and four edges that is not a tree.

Solution By Theorem 10.5.4, such a graph cannot be connected. One example of such an unconnected graph is shown below.

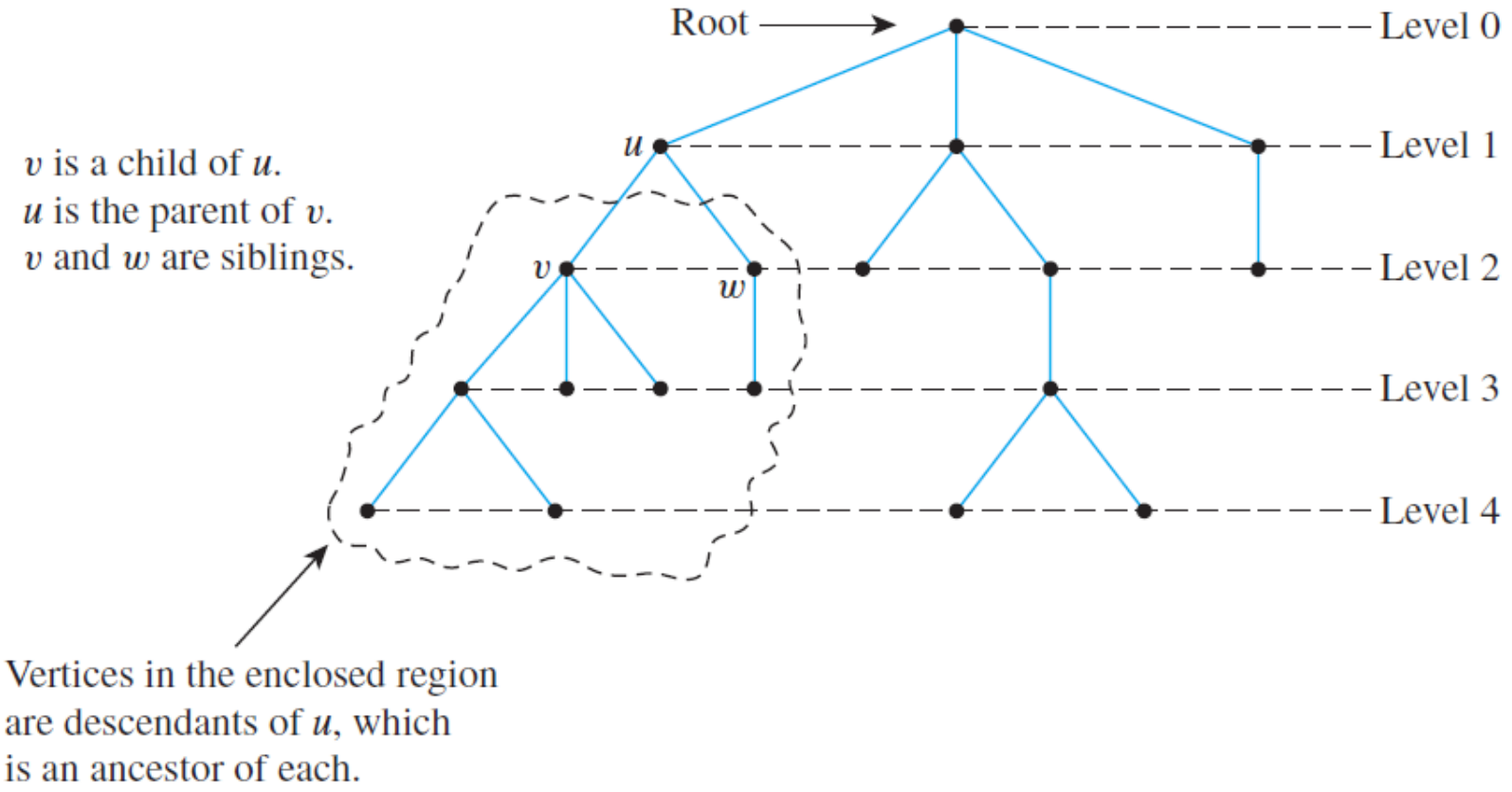


Rooted Trees

• Definition

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.

Rooted Tree –Labelled Figure



Example:

Consider the tree with root v_0 shown below.

a. What is the level of v_5?

b. What is the level of v_0?

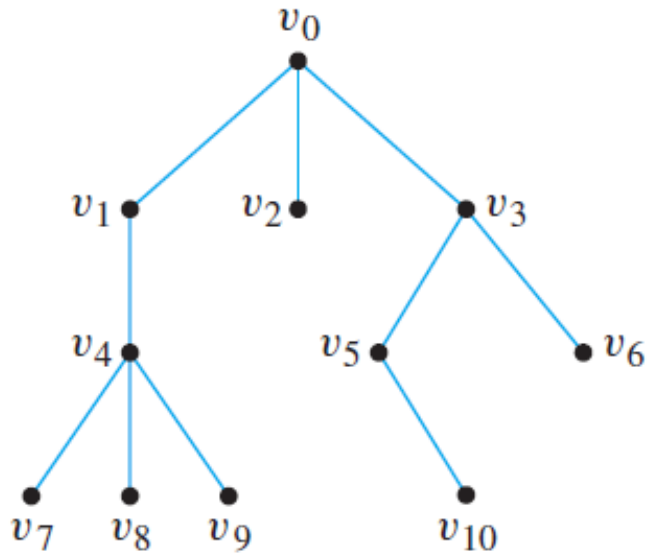
c. What is the height of this rooted tree?

d. What are the children of v_3?

e. What is the parent of v_2?

f. What are the siblings of v_8?

g. What are the descendants of v_3?



Solution

a. 2 b. 0 c. 3 d. v_5 and v_6 e. v_0 f. v_7 and v_9 g. v_5, v_6, v_{10}

Note that in the tree with root v_0 shown below, v_1 has level 1 and is the child of v_0, and both v_0 and v_1 are terminal vertices.

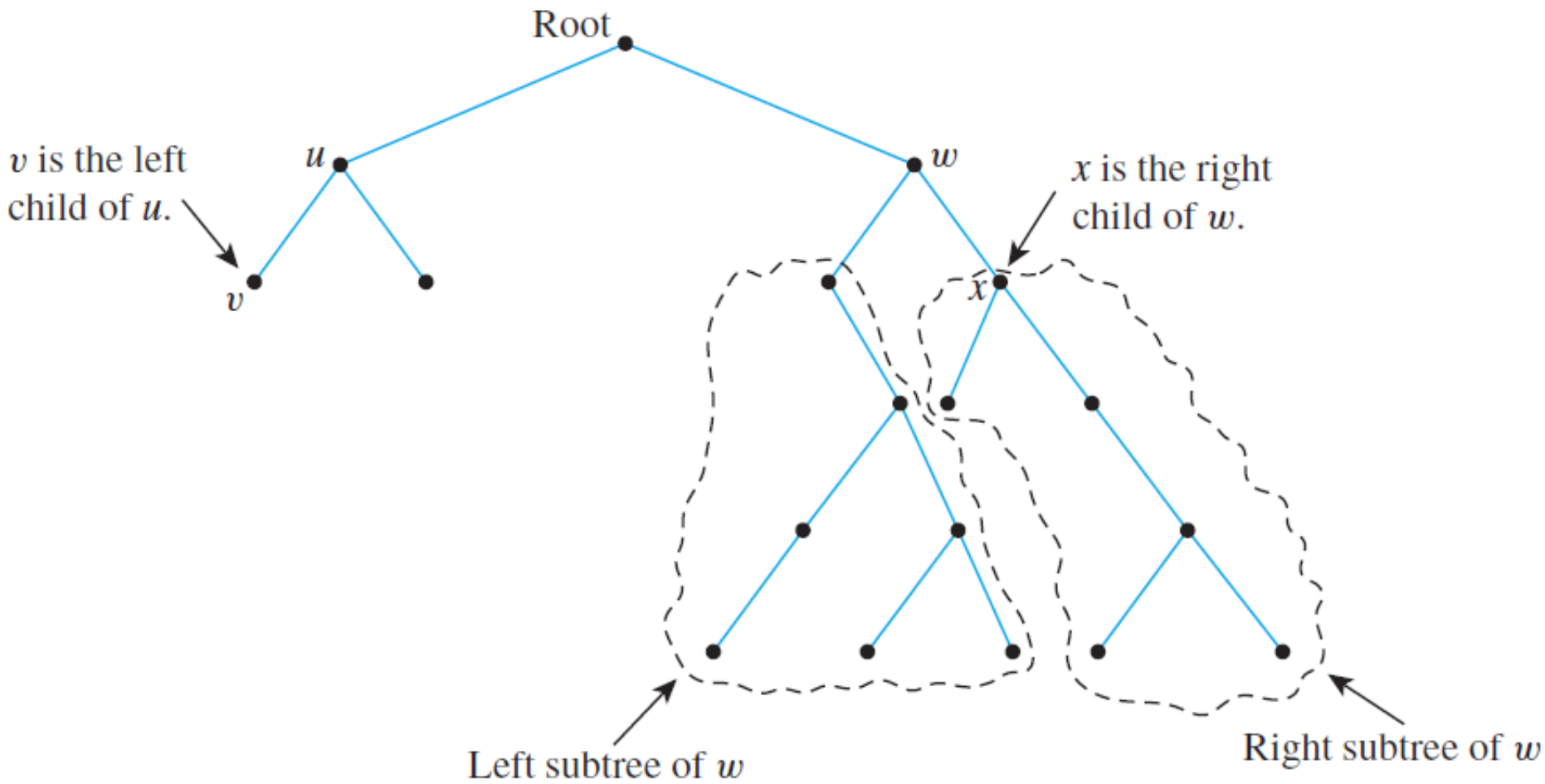
Types of Rooted Tree

- Binary Tree
- Full Binary Tree

• Definition

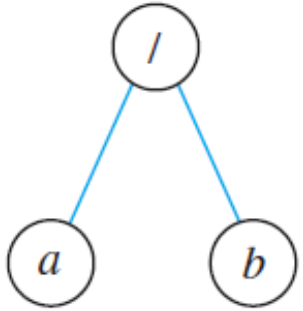
A **binary tree** is a rooted tree in which every parent has at most two children. Each child in a binary tree is designated either a **left child** or a **right child** (but not both), and every parent has at most one left child and one right child. A **full binary tree** is a binary tree in which each 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.

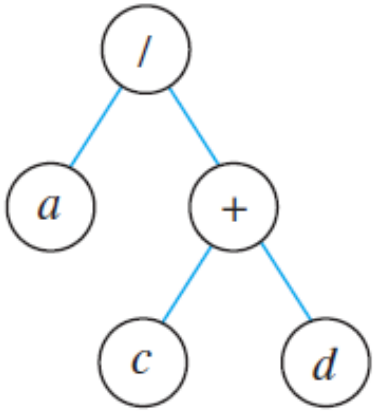

Root
v is the left
child of u.
u
w
x is the right
child of w.
v
x
Left subtree of w
Right subtree of w

Example: Representation of Algebraic Expression

Binary trees are used in many ways in computer science. One use is to represent algebraic expressions with arbitrary nesting of balanced parentheses. For instance, the following (labeled) binary tree represents the expression a/b: The operator is at the root and acts on the left and right children of the root in left-right order.

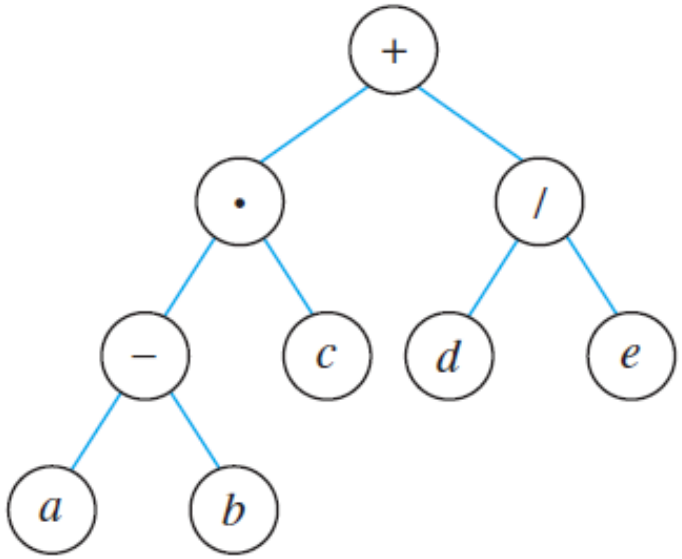


More generally, the binary tree shown below represents the expression $a/(c+d)$. In such a representation, the internal vertices are arithmetic operators, the terminal vertices are variables, and the operator at each vertex acts on its left and right subtrees in left-right order.



Draw a binary tree to represent the expression $((a-b)\cdot c)+(d/e)$.

Solution



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 terminal vertices, and conversely. More specifically, a full binary tree with k internal vertices has a total of $2k + 1$ vertices of which $k + 1$ are terminal vertices.

Theorem 10.6.1

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

Proof:

Suppose k is a positive integer and T is a full binary tree with k internal vertices. Observe that the set of all vertices of T can be partitioned into two disjoint subsets: the set of all vertices that have a parent and the set of all vertices that do not have a parent. Now there is just one vertex that does not have a parent, namely the root. Also, since every internal vertex of a full binary tree has exactly two children, the number of vertices that have a parent is twice the number of parents, or $2k$, since each parent is an internal vertex. Hence

$$\begin{bmatrix}\text{the total number}\\ \text{of vertices of } T\end{bmatrix} = \begin{bmatrix}\text{the number of}\\ \text{vertices that}\\ \text{have a parent}\end{bmatrix} + \begin{bmatrix}\text{the number of}\\ \text{vertices that do}\\ \text{not have a parent}\end{bmatrix}$$

$$= \quad 2k \quad + \quad 1.$$

But it is also true that the total number of vertices of T equals the number of internal vertices plus the number of terminal vertices. Thus

$$\begin{bmatrix}\text{the total number}\\ \text{of vertices of } T\end{bmatrix} = \begin{bmatrix}\text{the number of}\\ \text{internal vertices}\end{bmatrix} + \begin{bmatrix}\text{the number of}\\ \text{terminal vertices}\end{bmatrix}$$

$$= \quad k \quad + \begin{bmatrix}\text{the number of}\\ \text{terminal vertices}\end{bmatrix}$$

Now equate the two expressions for the total number of vertices of T:

$$2k + 1 = k + \begin{bmatrix}\text{the number of}\\ \text{terminal vertices}\end{bmatrix}$$

Solving this equation gives

$$\begin{bmatrix}\text{the number of}\\ \text{terminal vertices}\end{bmatrix} = (2k + 1) - k = k + 1.$$

Thus the total number of vertices is $2k + 1$ and the number of terminal vertices is $k + 1$ *[as was to be shown]*.

Example

Determining Whether a Certain Full Binary Tree Exists

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

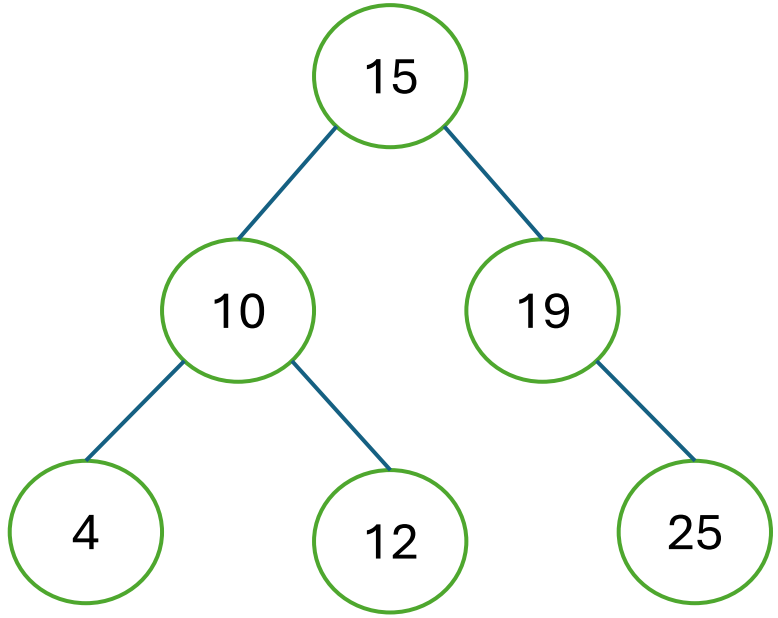
Solution No. By Theorem 10.6.1, a full binary tree with 10 internal vertices has $10 + 1 = 11$ terminal vertices, not 13. ■

Binary Search Trees (BST)

- It's a kind of binary tree in which data records, i.e; customer information can be stored, searched and processed effectively.
- To place records in BST it must be necessary to place records in a total order.
- If no natural order is there then a key must be introduced.
- **KEY:** An element of a totally ordered set, such as number or a word, may be added to each record.
  - Keys must be inserted into vertices of each tree and provide access to records to which they are attached.

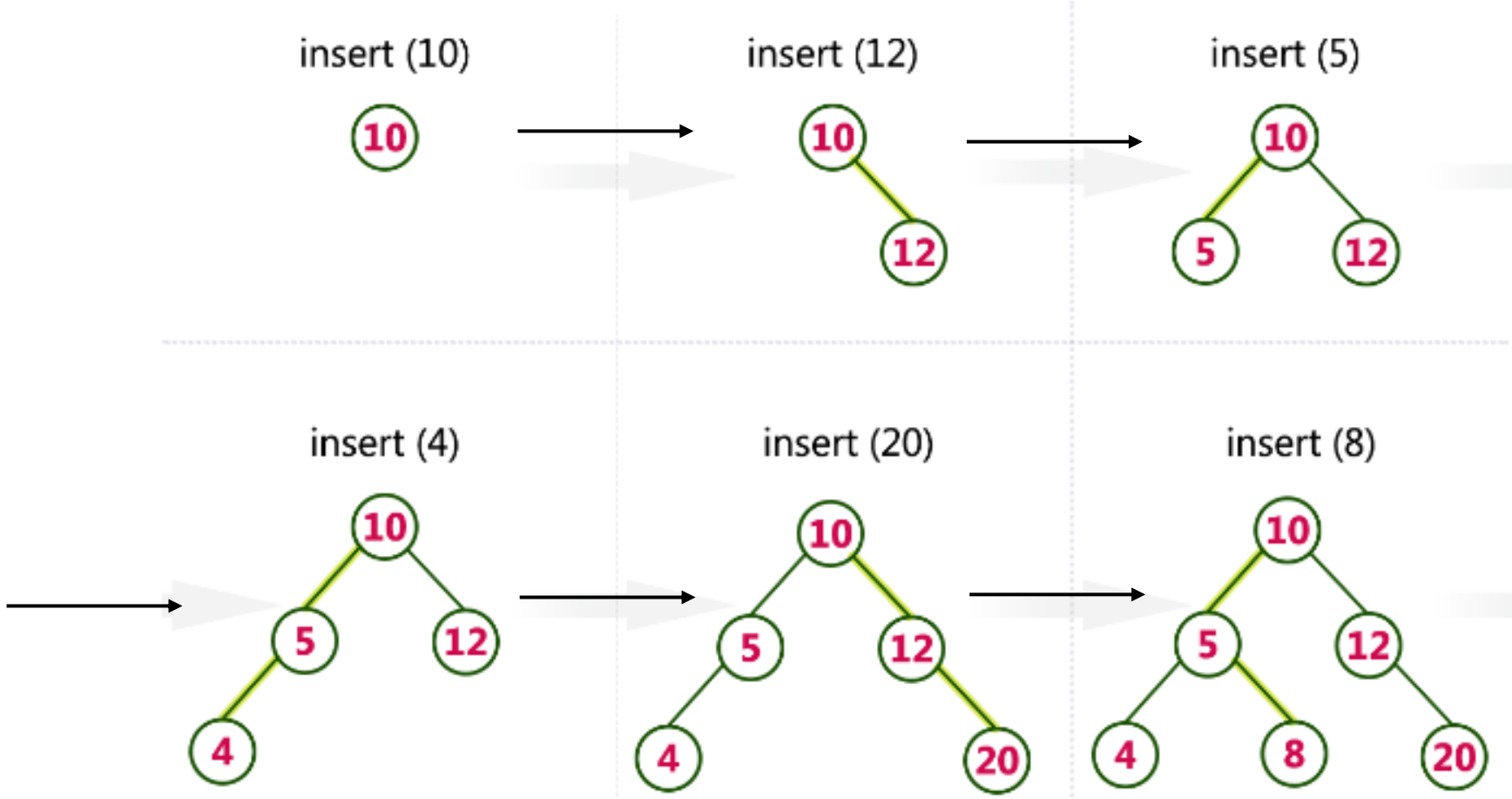
Properties-Binary Search Trees (BST)

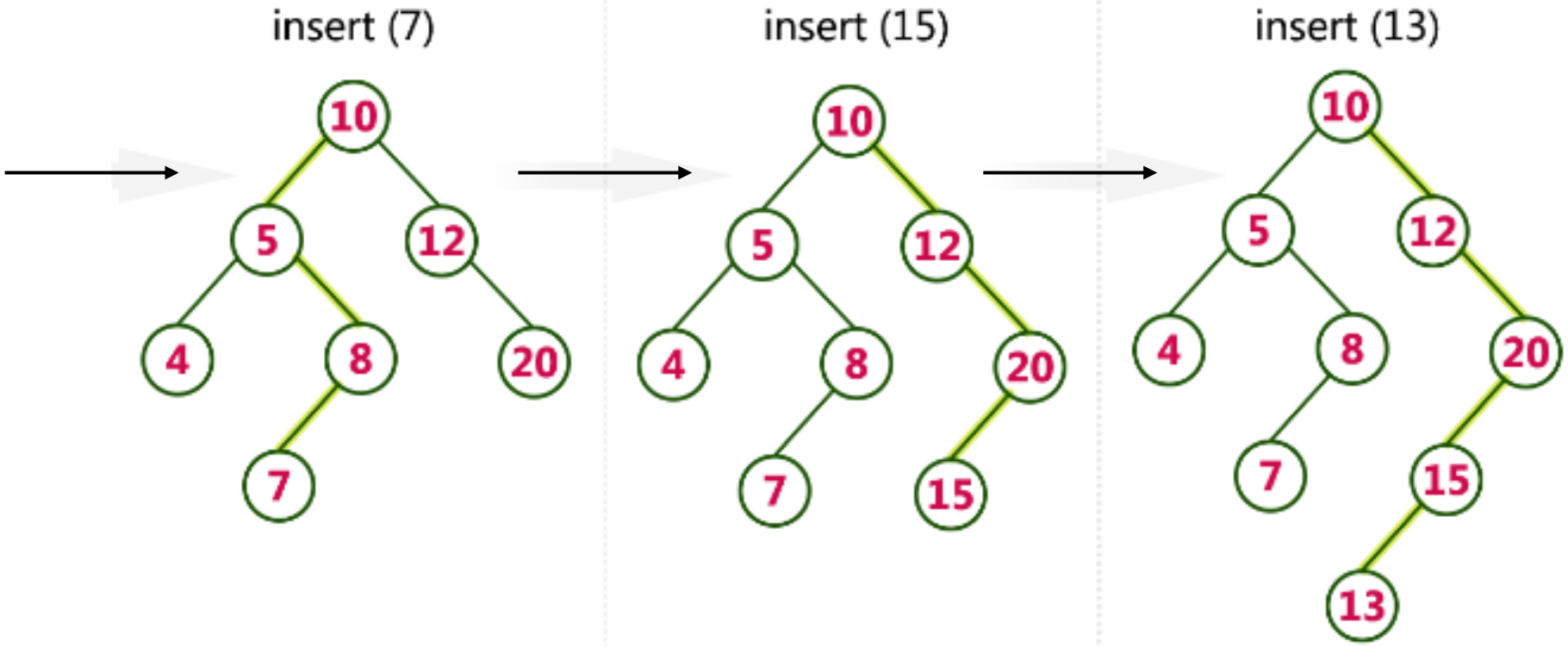
- For every internal vertex *v*,
  - all the keys in the left subtree of *v are less than the key in v,*
  - all the keys in the right subtree of *v are greater than the key in v,*
- ***Example:***
  - ***Keys: 15, 10, 19, 25, 12, 4***
  - ***Construct a Binary search tree of the mentioned keys records.***


15
10
19
4
12
25

10,12,5,4,20,8,7,15 and 13

Above elements are inserted into a Binary Search Tree as follows...

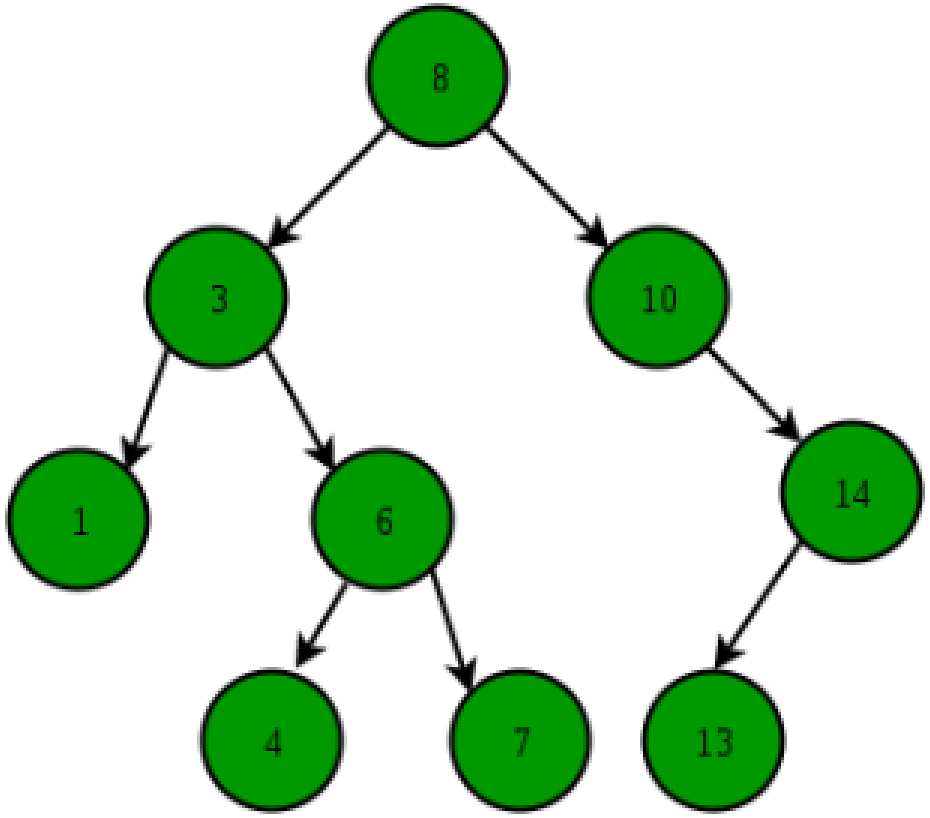


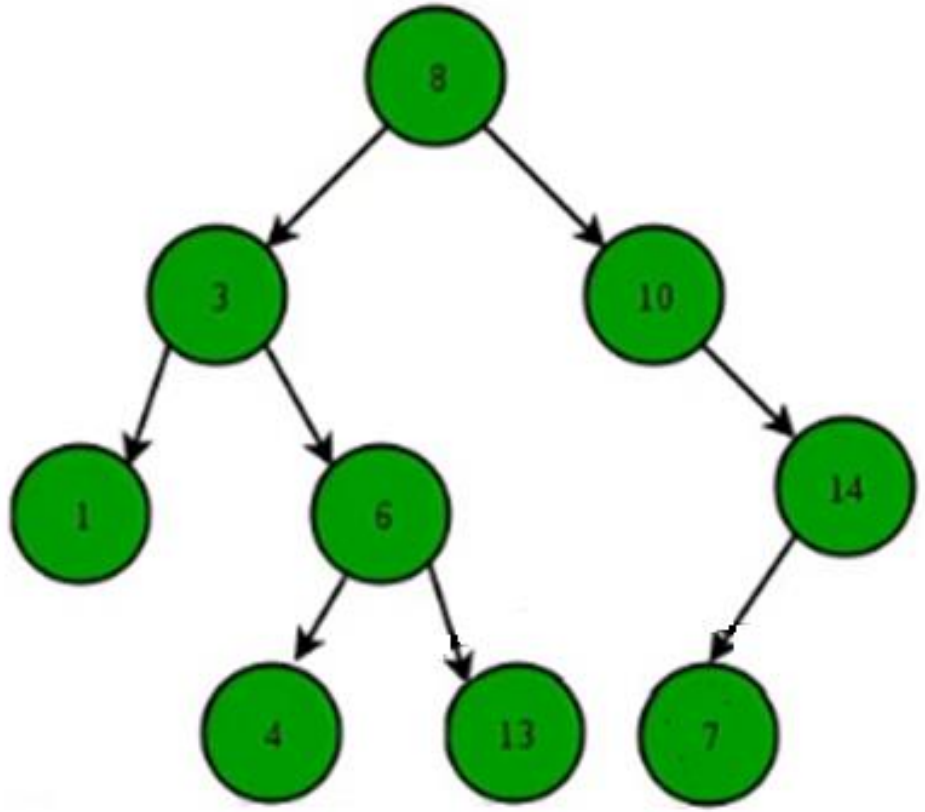

insert (7)
10
5
12
4
8
20
7
insert (15)
10
5
12
4
8
20
7
15
insert (13)
10
5
12
4
8
20
7
15
13

Handling approach for Duplicate values in the Binary Search tree

- You can not allow the duplicated values at all.
- We must follow a consistent process throughout i.e either store duplicate value at the left or store the duplicate value at the right of the root, but be consistent with your approach.
- We can keep the counter with the node and if we found the duplicate value, then we can increment the counter

Are these BSTs?





Algorithm

- Input: A totally ordered, non-empty 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**

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

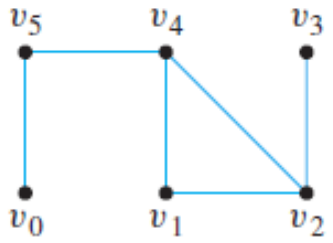
Spanning Trees and Shortest Path

- Spanning Trees
  - Minimum Spanning Tree(MST)
  - Kruskal's Algorithm
  - Prim's Algorithm
- Shortest Path
  - Dijkstra's Algorithm

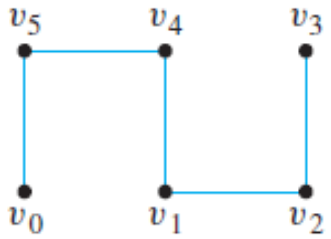
Spanning Tree

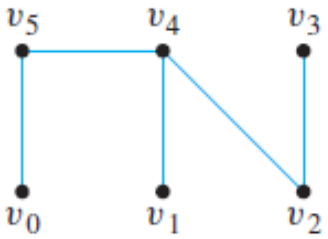
"A **spanning tree** for a graph G is a subgraph of G that contains every vertex of G and is a tree."

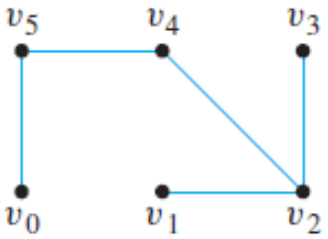
Find all spanning trees for the graph G pictured below.



Solution The graph G has one circuit $v_2v_1v_4v_2$, and removal of any edge of the circuit gives a tree. Thus, as shown below, there are three spanning trees for G.







Minimum Spanning Tree(MST)

• Definition

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 total weight compared to all other spanning trees for the graph.

If G is a weighed 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.

An East Coast airline company wants to expand service to the Midwest and has received permission from the Federal Aviation Authority to fly any of the routes shown in Figure 10.7.1.

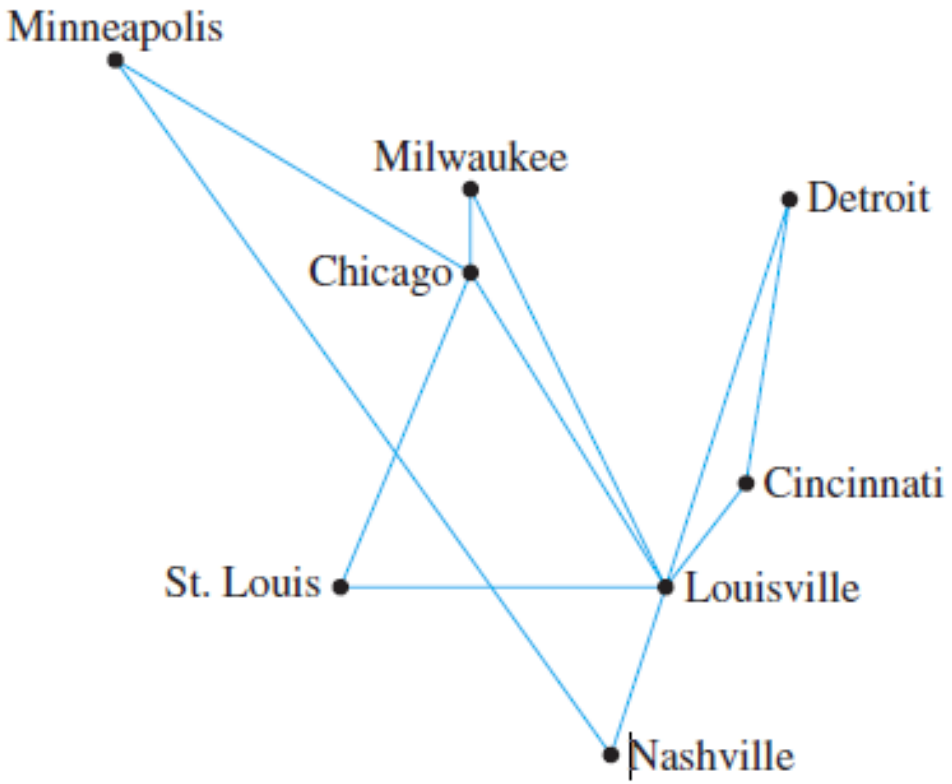
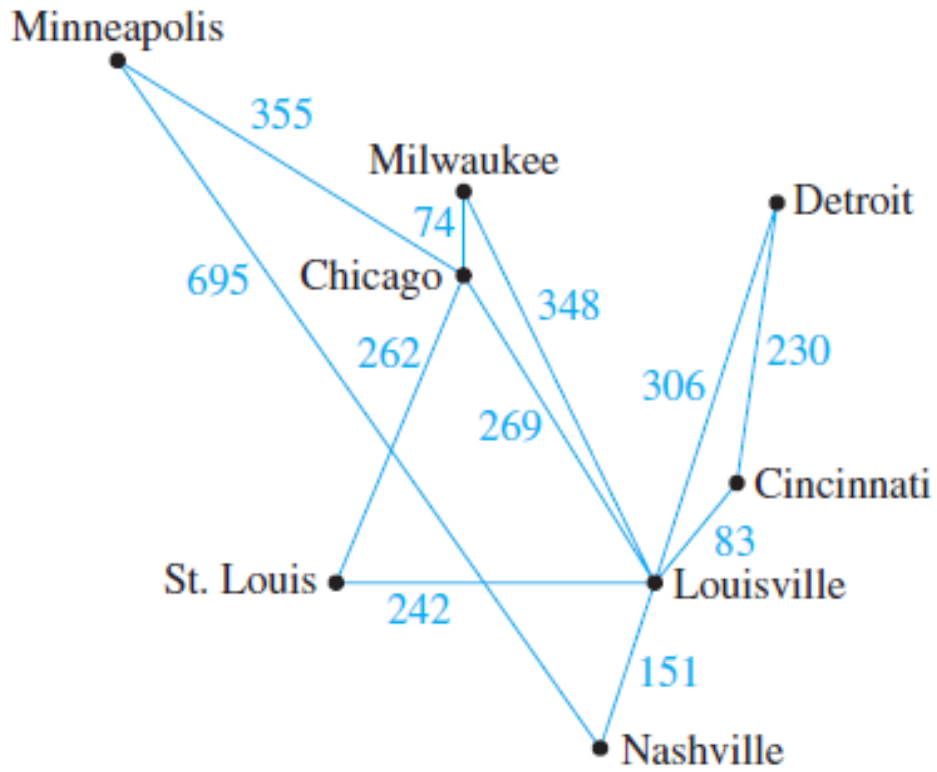


Figure 10.7.1

The graph of the routes allowed by the Federal Aviation Authority shown in Figure 10.7.1 can be annotated by adding the distances (in miles) between each pair of cities. This is done in Figure 10.7.3.



Now suppose the airline company wants to serve all the cities shown, but with a route system that minimizes the total mileage. 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 (again, by Lemma 10.5.3), 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 *weighed 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.

- The problem of finding a minimum spanning tree for a graph is certainly solvable.
- One solution is to list all spanning trees for the graph, compute the total weight of each, and choose one for which this total is a minimum. (Note that the well-ordering principle for the integers guarantees the existence of such a minimum total.) This solution, however, is inefficient in its use of computing time because the number of distinct spanning trees is so large.
- For instance, a complete graph with n vertices has n^{n-2} spanning trees.
- Even using the fastest computers available today, examining all such trees in a graph with approximately 100 vertices would require more time than is estimated to remain in the life of the universe.
- We have other algorithms to solve this problem.