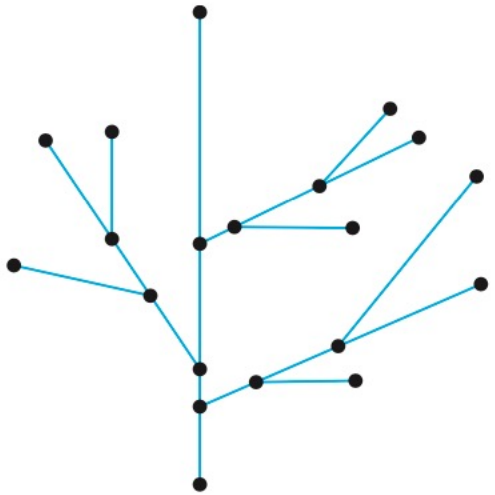
Trees

Definition

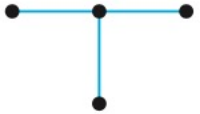
- A graph is said to be **circuit-free** if, and only if, it has no circuits.
- A graph is called a **tree** if, and only if, it is circuit-free and connected.
- A **trivial tree** is a graph that consists of a single vertex.
- A graph is called a **forest** if, and only if, it is circuit-free and not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

Example:

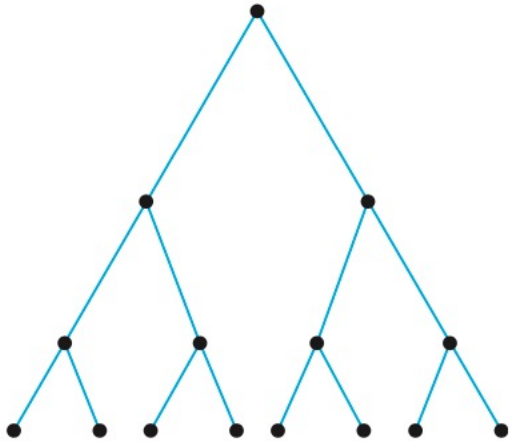


(a)

(b)

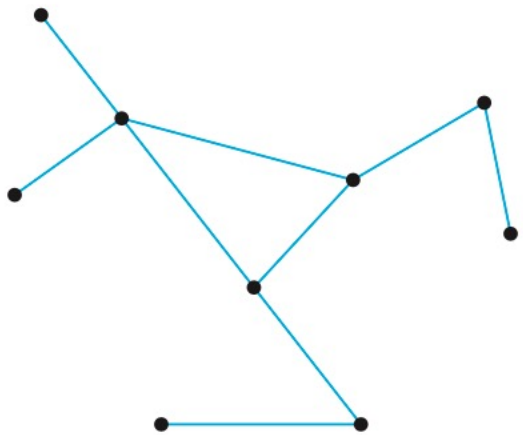


(c)

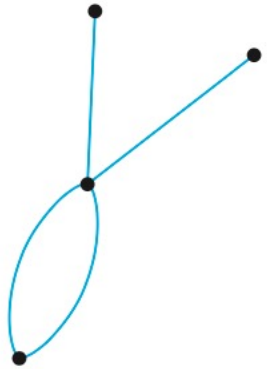


(d)

Example:

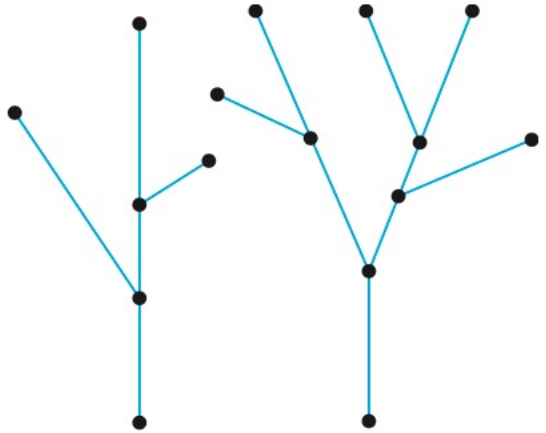


(a)



(b)

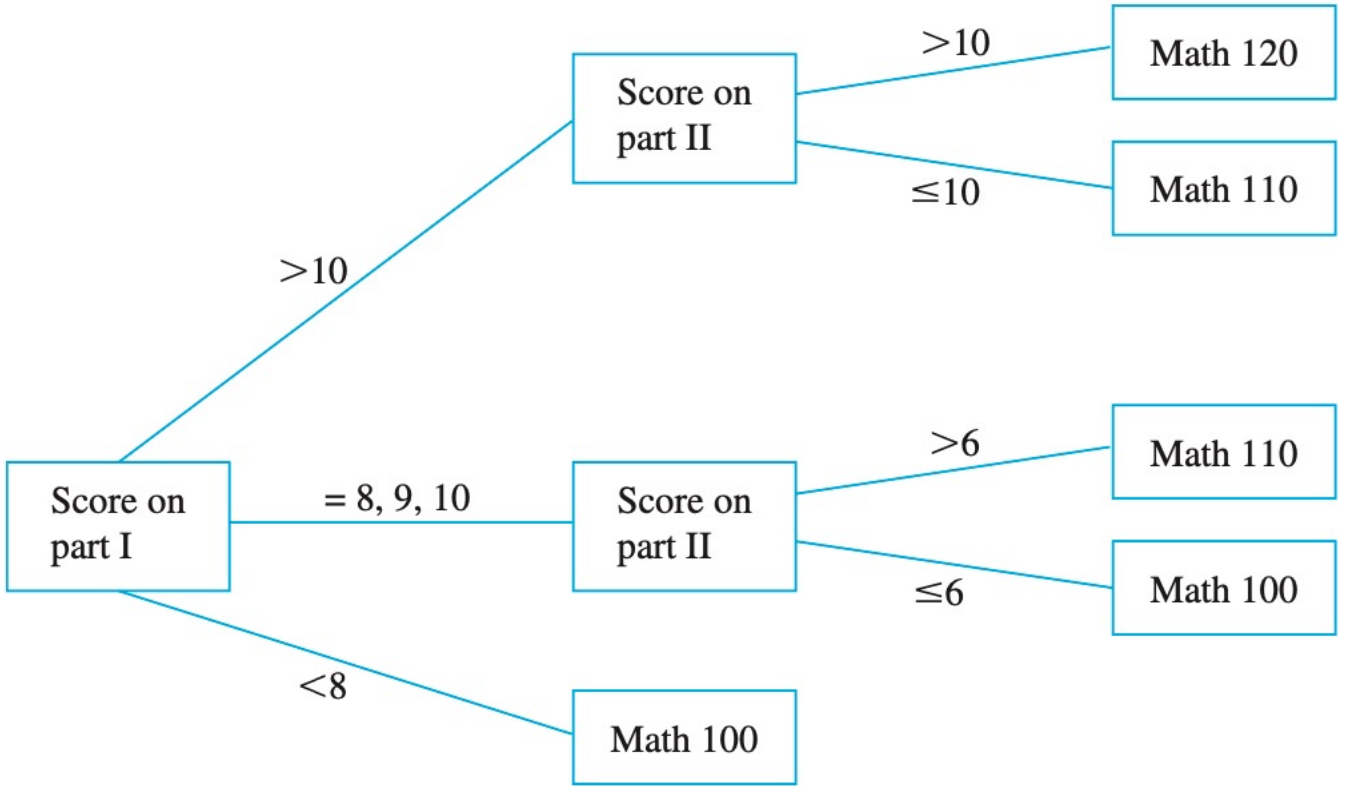
(c)



(d)

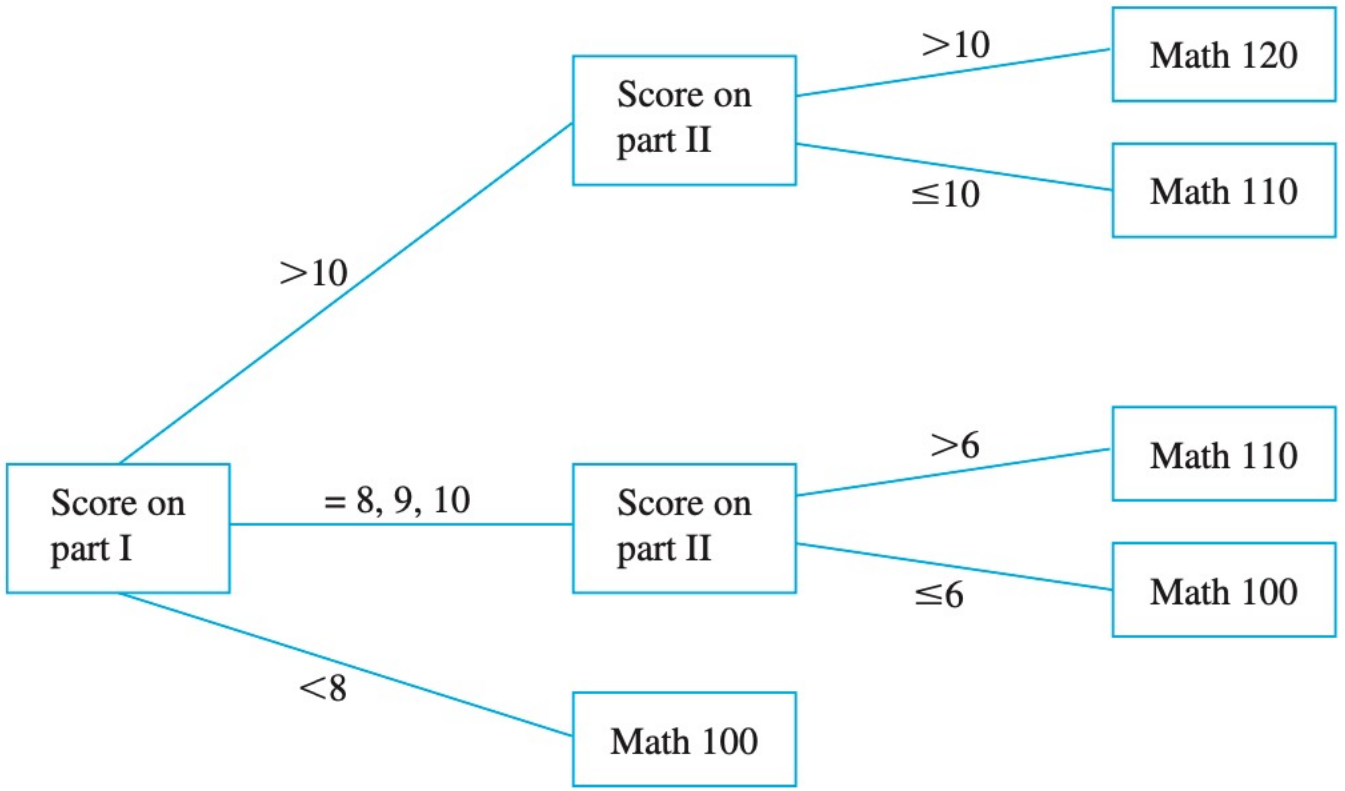
Example: Decision Tree

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.



Example: Decision Tree

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.



Example: A Parse Tree

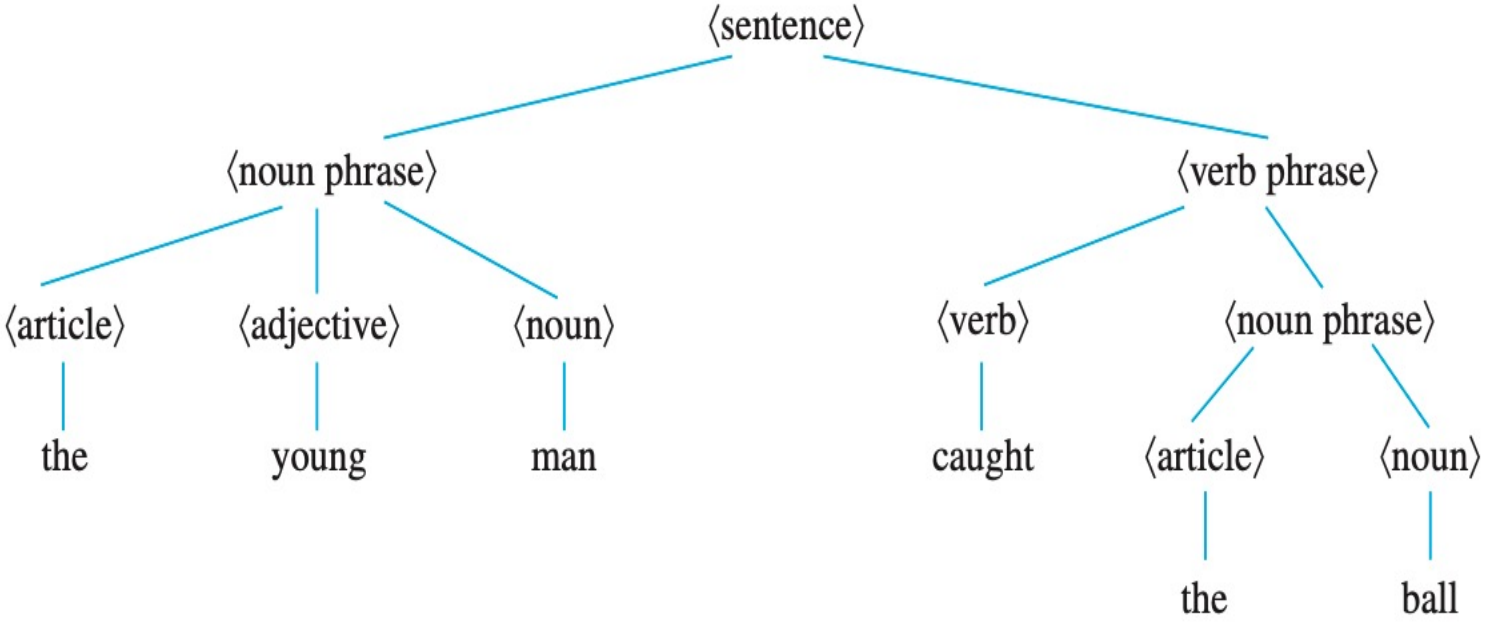
Noam Chomsky and others have developed new ways to describe the syntax (or grammatical structure) of natural languages such as English. In the study of grammars, trees are often used to show the derivation of grammatically correct sentences from certain basic rules. Such trees are called **syntactic derivation trees** or **parse trees**.

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

- a sentence can be produced by writing first a noun phrase and then a verb phrase;
- a noun phrase can be produced by writing an article and then a noun;
- A noun phrase can also be produced by writing an article, then an adjective, and then a noun;
- a verb phrase can be produced by writing a verb and then a noun phrase;

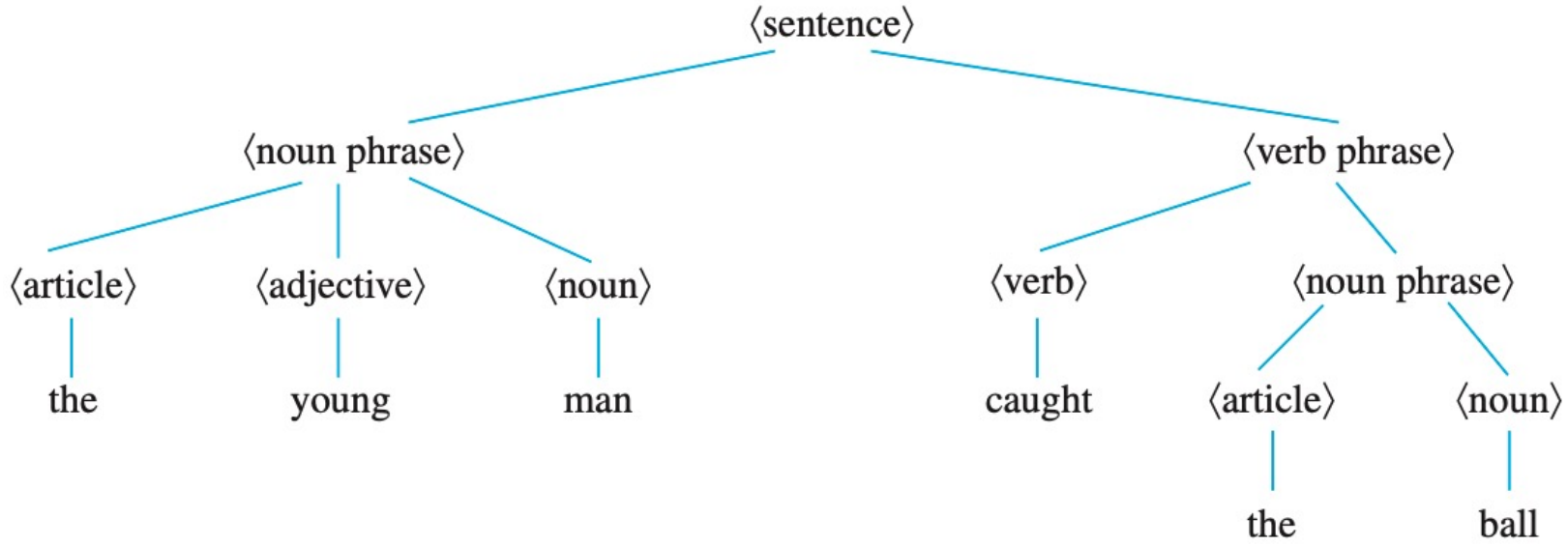
Example: A Parse Tree

- a sentence can be produced by writing first a noun phrase and then a verb phrase;
- a noun phrase can be produced by writing an article and then a noun;
- A noun phrase can also be produced by writing an article, then an adjective, and then a noun;
- a verb phrase can be produced by writing a verb and then a noun phrase;
- one article is “the”;
- one adjective is “young”;
- one verb is “caught”;
- one noun is “man”;
- one (other) noun is “ball.”



Example: A Parse Tree

In the study of linguistics, **syntax** refers to the grammatical structure of sentences, and **semantics** refers to the meanings of words and their interrelations. A sentence can be syntactically correct but semantically incorrect, as in the nonsensical sentence “The young ball caught the man,” which can be derived from the rules given above. Or a sentence can contain syntactic errors but not semantic ones, as, for instance, when a two-year-old child says, “Me hungry!”



Lemma

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

Definition

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**.

Theorem:

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

Example:

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

Example:

Find all non-isomorphic trees with four vertices.

Theorem:

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

Example:

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

Question:

What is the total degree of a tree with *n* vertices? Why?

Theorem:

If G is any graph with n vertices and m edges, where m and n are positive integers and $m \geq n$, then G has a circuit.

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*.

Definition:

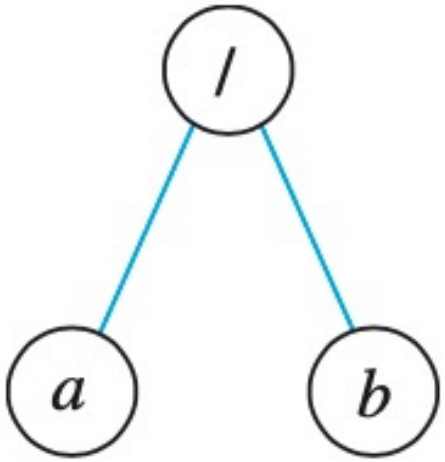
- A **binary tree** is a rooted tree in which every parent has at most two children.
- Each child in a binary tree is designated either a **left child** or a **right child** (but not both), and every parent has at most one left child and one right child.
- A **full binary tree** is a binary tree in which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

Representation of Algebraic Expressions

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-to-right order.

Representation of Algebraic Expressions

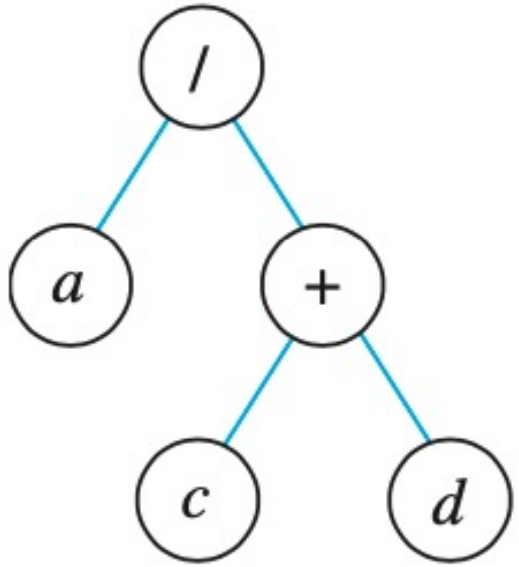
Binary trees are used in many ways in computer science. One use is to represent algebraic expressions with arbitrary nesting of balanced parentheses. For instance, the following (labeled) binary tree represents the expression *a/b*: The operator is at the root and acts on the left and right children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



Representation of Algebraic Expressions

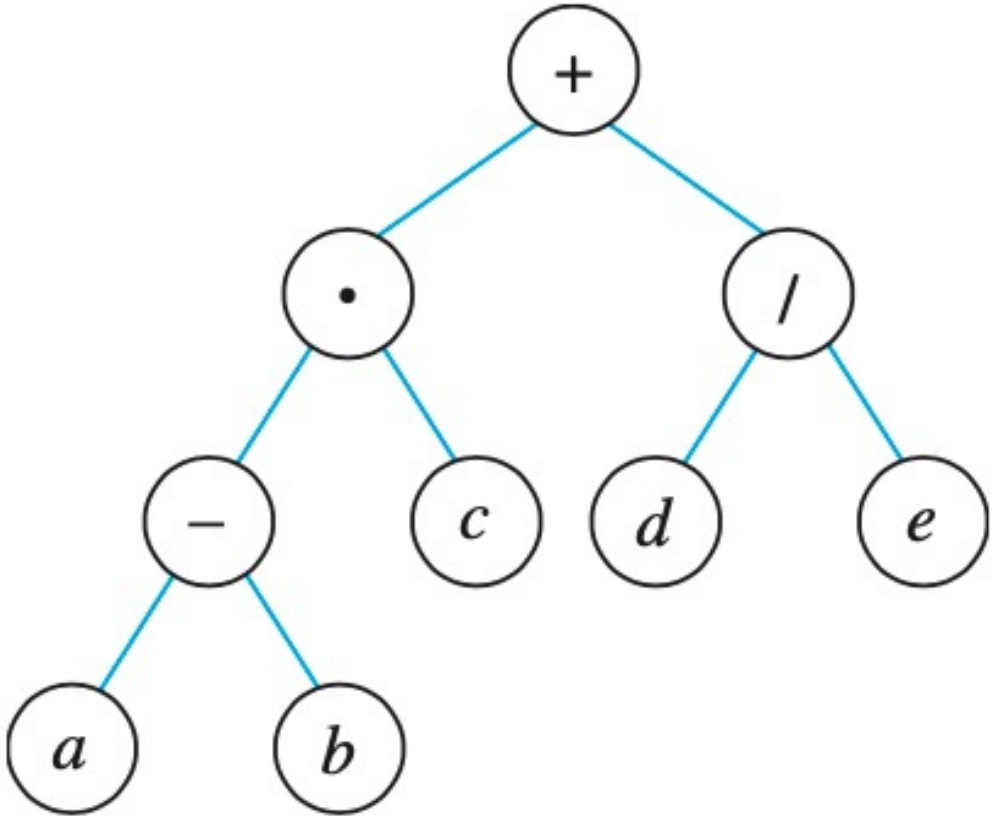
More generally, the binary tree shown below represents the expression *a*/(*c*+*d*). In such a representation,

- the internal vertices are arithmetic operators,
- the leaves are variables,
- and the operator at each vertex acts on its left and right subtrees in left-to-right order.



Example:

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



Theorem:

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.

Question:

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

Solution No. By Theorem 10.5.1, a full binary tree with 10 internal vertices has

$$10+1=11 \text{ leaves, not } 13.$$

Theorem

For every integer $h \geq 0$, if T is any binary tree with height h and t leaves, then $t \leq 2^h$.

Equivalently:

$$\mathrm{Log}_2\, t \leq h.$$

Corollary:

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

Binary Search Trees

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.

Binary Search Trees

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.

Binary Search Trees

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*.

Algorithm: 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* 5 0. **while** (*success* 5 0)

if (*newkey* , *key*(*v*))
then if (*v* has a left child), call the left child *vL* and let *v*:5 *vL* **else do**1. add a vertex *vL* to *T* as the left child for *v*

2. add an edge to *T* to join *v* to *vL* 3. insert *newkey* as the key for *vL* 4. let *success* :5 1 **end do**

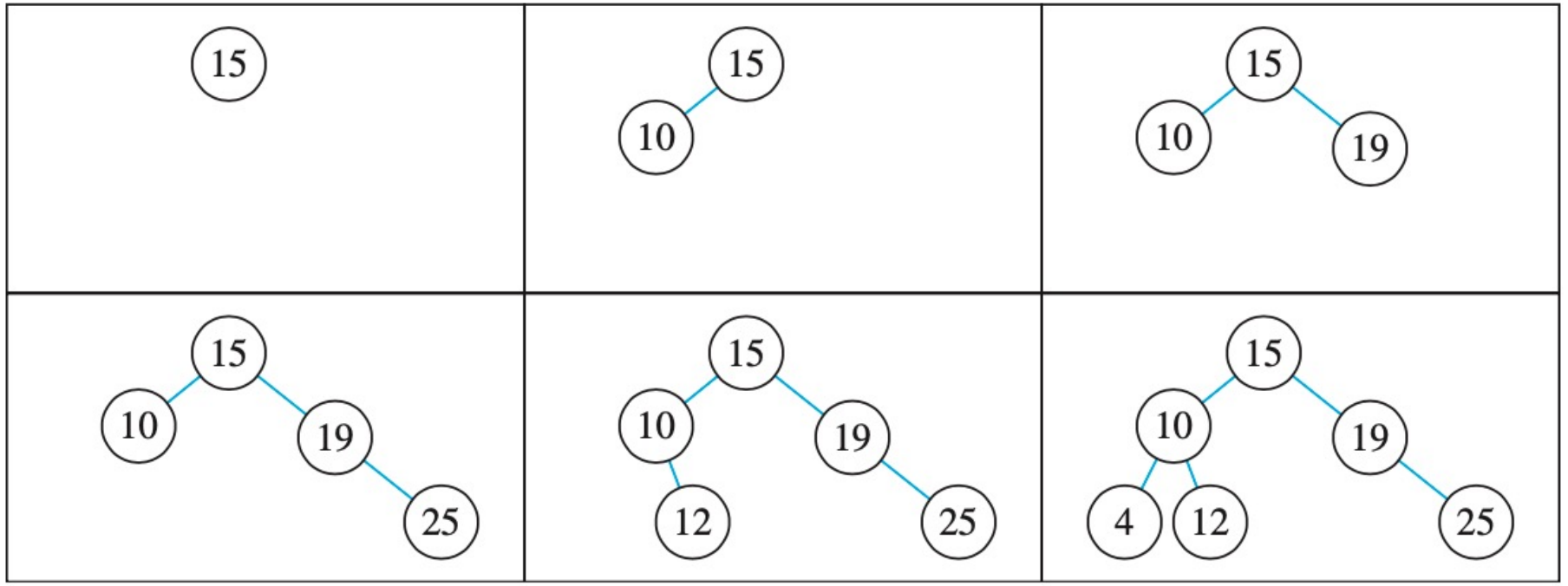
if (*newkey* . *key*(*v*))
then if *v* has a right child
then call the right child *vR*, and let *v* :5 *vR*
else do1. add a vertex *vR* to *T* as the right child for *v*
2. add an edge to *T* to join *v* to *vR* 3. insert *newkey* as the key for *vR* 4. let *success* :5 1 **end do**
end while end while
Output: A binary search tree *T* for the set *K* of keys

Example:

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.

Example:

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.



Example:

Note that the algorithm does not specify an order in which keys are to be added as the tree is being built. In fact, adding keys in a different order usually results in a different binary search tree for the given set of keys. For example, if the keys shown in the previous example are added in the order 19, 10, 25, 4, 15, 12, the result is the following binary search tree: