Trees

Mongi BLEL

King Saud University

August 30, 2019

Table of contents

Definition

A tree is a connected simple undirected graph with no simple circuits (دورات).

A forest (غابة) is a (not necessarily connected) simple undirected graph with no simple circuits.

Remark

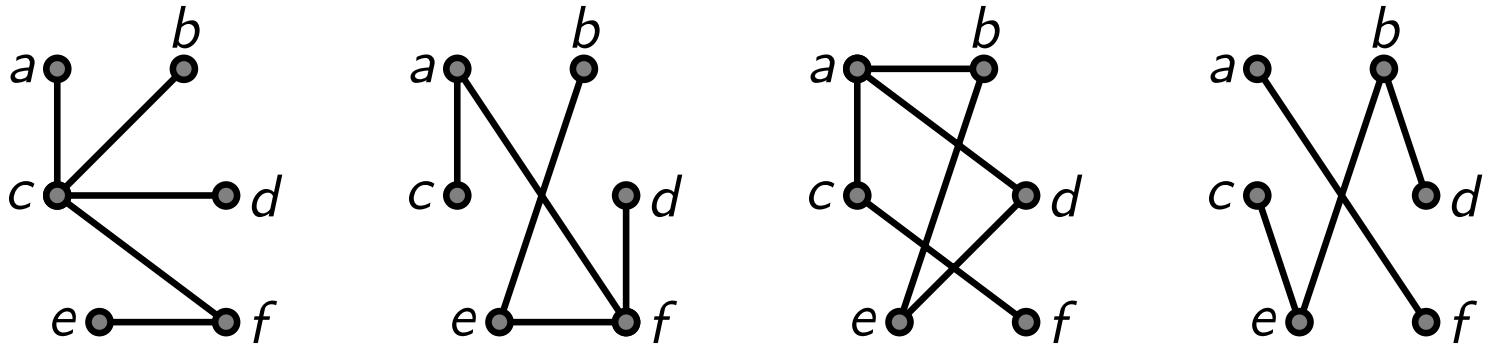
A graph G is a tree if and only if it contains no cycles, but adding any new edge creates a cycle.

Properties

If T is a tree:

1. There is a unique simple path between any 2 of its vertices.
2. No loops.
3. No multiple edges.

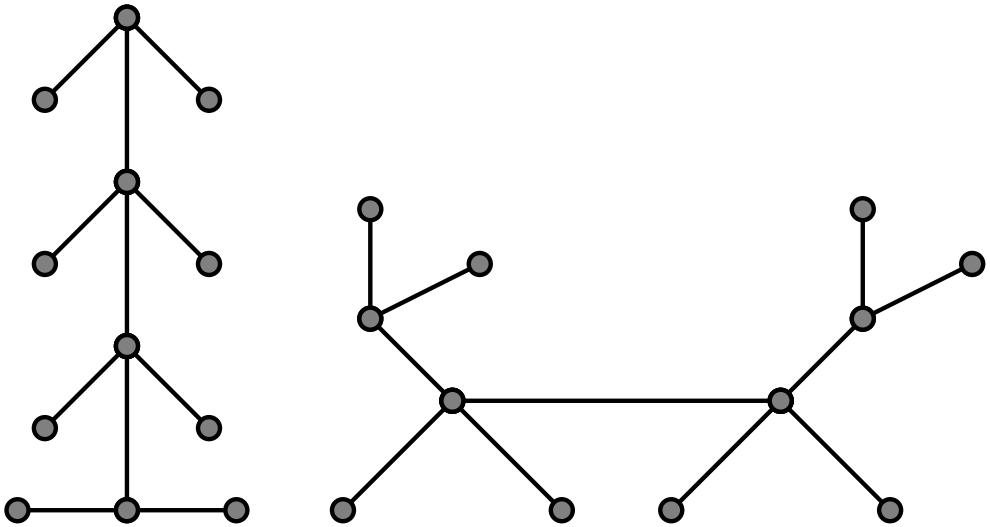
Examples



The first and the second are trees. The third contains a circuit a, b, e, d. The forth is not connected.

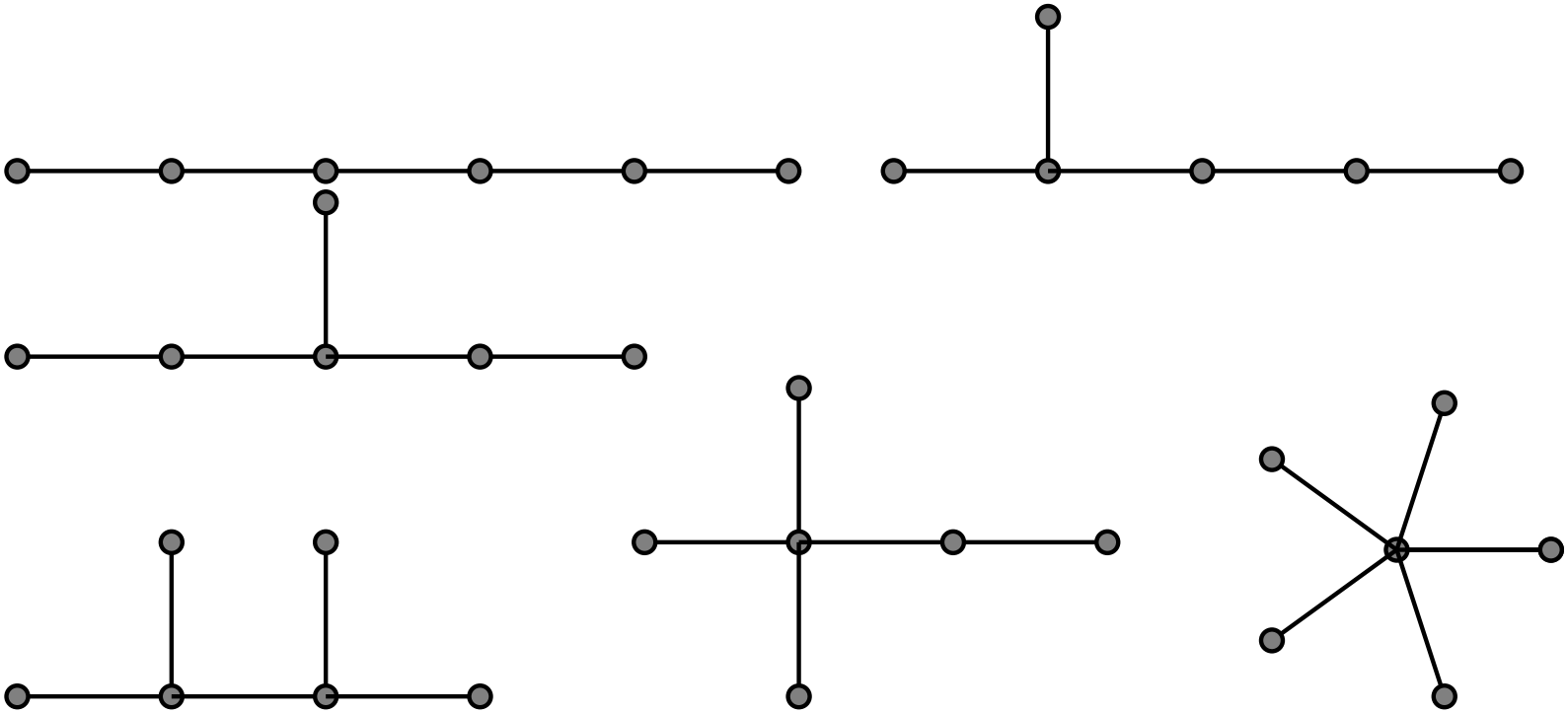
Examples

Example of forest:



Examples

The non isomorphic trees on 6 vertices



Theorem

Any undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

Theorem

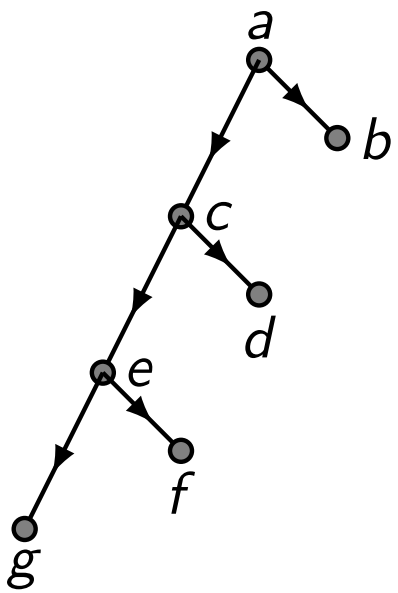
Every tree is a bipartite graph.

Rooted (Directed) Trees

Definition

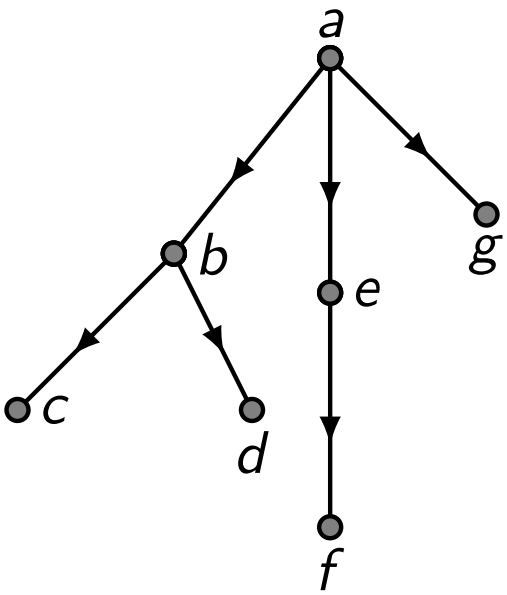
A rooted tree is a tree in which one vertex has been designed as the root and every edge is directed away from the root.

Example



The root is a vertex with in-degree 0. [Node *a* is the root]

Example

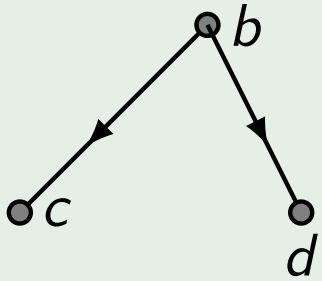


Definition

1. Parent: a vertex u is a parent of v if there is a directed edge from u to v.
2. Child: If u is parent of v, then v is child of u. [c and d are children of b]
3. Siblings: Vertices with the same parents. [c and d]
4. Ancestors: Vertices in path from the root to vertex v, excluding v itself, including the root. [Ancestors of d : b, a]
5. Descendents: All vertices that have v as ancestors. [Descendents of b : c, d]

Definition

6. Leaf: Vertex with no children. [c, d, f, g]
7. Internal vertices: Vertices that have children. [a, b, e].
8. Subtree: Subgraphs consisting of v and its descendents and their incident edges. (Subtree rooted at b)



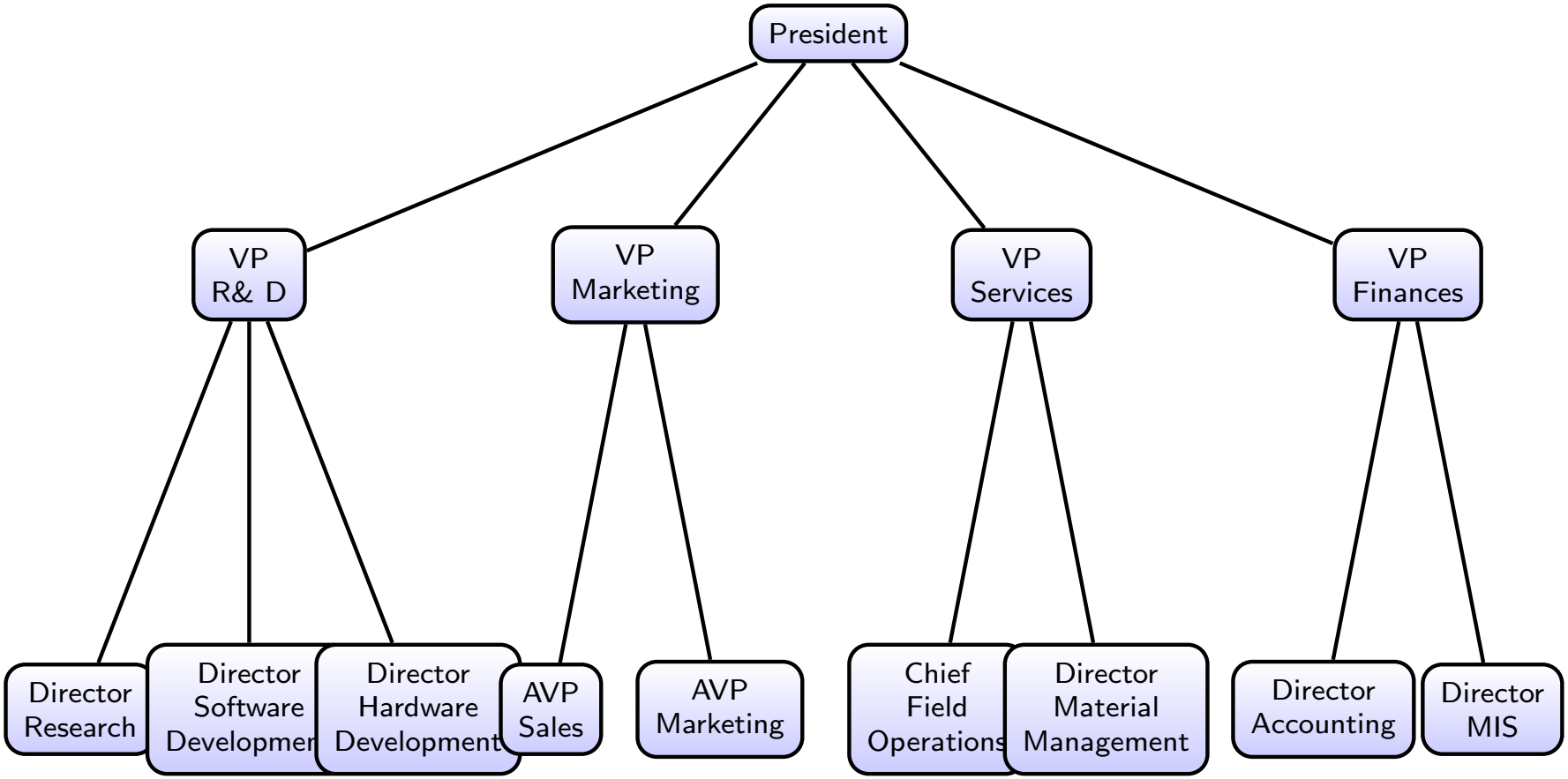
Definition

9. Level (of a vertex v) is the length of the unique path from root to v. [level of root $= 0$, level of $b = 1$, level of $c = 2$]
10. Height is maximum of vertices levels. [Height $= 2$]

Trees as Models

Trees are used as models in such diverse areas as computer science, chemistry, geology, botany, and psychology.We will describe a variety of such models based on trees. For example the representation of organizations. Each vertex in this tree represents a position in the organization. An edge from one vertex to another indicates that the person represented by the initial vertex is the (direct) boss of the person represented by the terminal vertex.

Example


President
VP R& D
VP Marketing
VP Services
VP Finances
Director Research
Director Software Developmer
Director Hardware Development
AVP Sales
AVP Marketing
Chief Field Operations
Director Material Management
Director Accounting
Director MIS

Theorem

Every tree, $T = (V, E)$ with $|V| \geq 2$, has at least two vertices that have degree equal 1.

Proof

Take any longest simple path $x_1, \ldots, x_m$ in T. Both x_1 and x_m must have degree 1: otherwise there is a longer path in T.

Theorem

Every tree with n vertices has exactly $n-1$ edges.

The proof is by induction on n. For $n=1$, there is no edges. Suppose the result for n and take a tree $T=(V,E)$ with $n+1$ vertices. There is $x \in V$ with degree 1. Since T is a tree, $T-x$ is a tree with n vertices. Then $|E|-1=|V-\{x\}|-1$ and $|E|=|V|-1$.

Definition

If a graph G is connected and e is an edge such that $G - e$ is not connected, then e is said to be a bridge or a cut edge.

Theorem

Any edge in a tree is a bridge.

Theorem

Let $T = (V, E)$ be a connected graph such that $|V| = n$ and $|E| = n - 1$, then T is a tree.

Proof

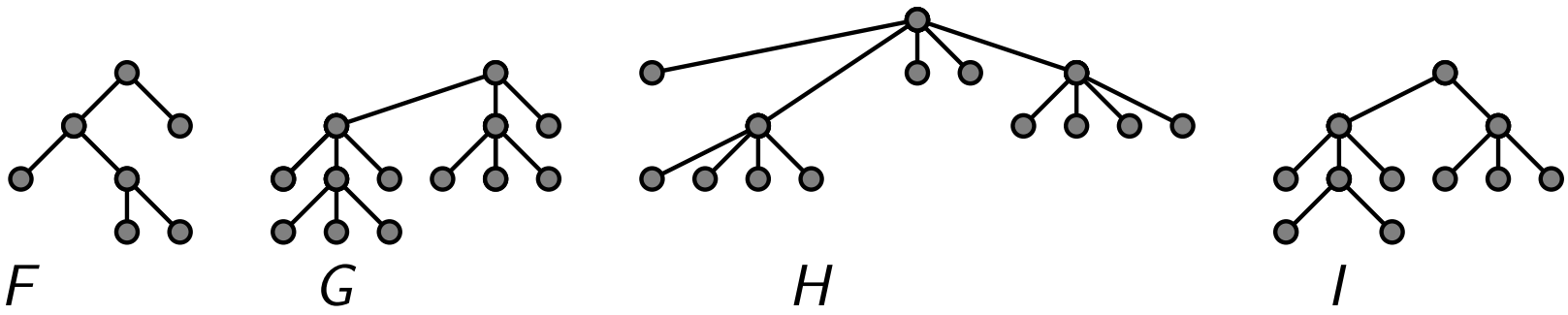
To prove that T is a tree it suffices to prove that do not contains circuits. If $x_1, \ldots, x_m$ is a circuit from a to a. Then x_2 is not a bridge and $T - x_2$ is a connected graph with n vertices and $n - 2$ edges, which is impossible. Then T does not contain circuits.

$m-$ary Trees

Definition

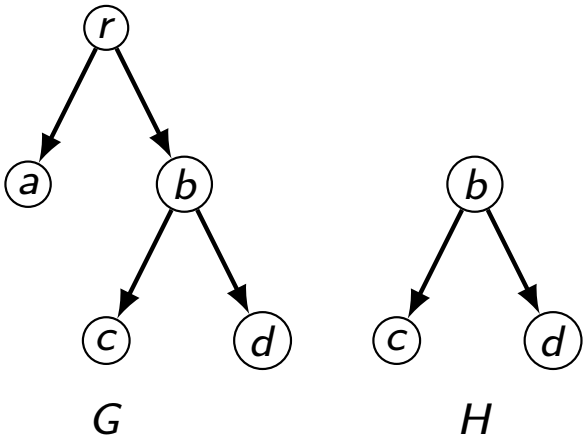
Let $m \geq 1$

1. A rooted tree is called a $m-$ary tree if every internal node has at most m children.
2. A rooted tree is called a full $m-$ary tree if every internal node has exactly m children.
3. An $m-$ary tree with $m = 2$ is called a binary tree.



F is a binary tree, G is a full $3-$ary, H is a $5-$ary, I is $3-$ary.

1. A rooted ordered tree is a rooted tree $(T, \leq)$ where in addition the children of each internal vertex v are linearly ordered according to some ordering $\leq$.
2. When drawing the tree, we usually write ordered children (from least to greatest) from left to right.
3. If the rooted ordered tree is a binary tree, then the first child is called left child and the second child is called right child.
4. The tree rooted at the left child of a vertex is called the left subtree of this vertex, and the tree rooted at the right child of a vertex is called the right subtree of the vertex.



In the tree G, a is the left child of r, b is the right child of r and the tree H is the right sub-tree of r.

Counting vertices in a full $m-$ary trees

Theorem

For all $m \geq 1$, every full $m-$ary tree with i internal vertices has exactly $n = m.i + 1$ vertices.

Proof Every vertex other than the root is a child of an internal vertex. There are thus $m.i$ such children, plus 1 root.

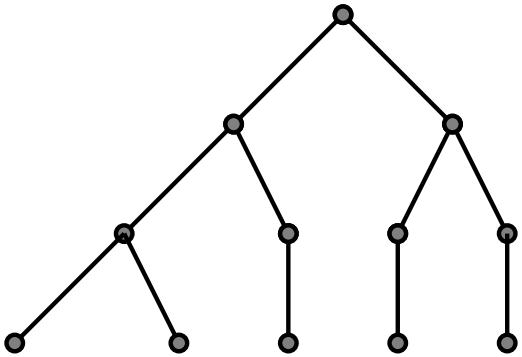
Theorem

For all $m \geq 1$, a full $m-$ary tree with:

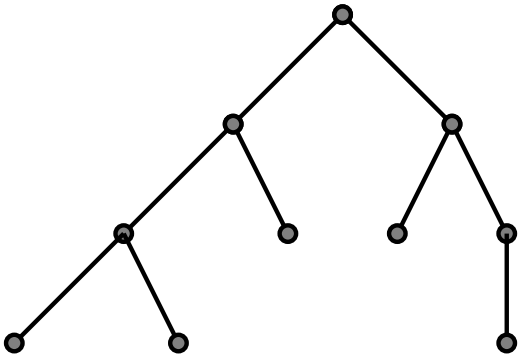
1. n vertices has $i = \dfrac{n-1}{m}$ internal vertices and $\ell = \dfrac{(m-1)n+1}{m}$ leaves.
2. i internal vertices has $n = m.i + 1$ vertices and $\ell = (m-1)i + 1$ leaves.
3. if $m \geq 2$, then if the $m-$ary tree has ℓ leaves then it has $n = m\ell - 1m - 1$ vertices and $i = \dfrac{\ell - 1}{m-1}$ internal vertices.

Definition

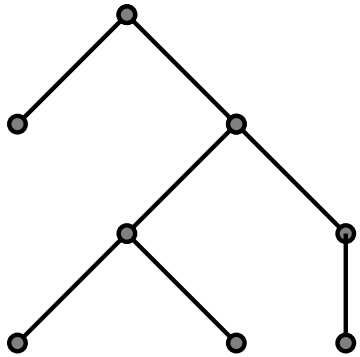
1. The height of the rooted tree is the maximum of the levels of vertices (length of the longest path from the root to any vertex)
2. **Balanced Tree** A rooted $m-$ary tree of height h is balanced if all leaves are at levels h or $h-1$.



Balanced



Balanced



not Balanced

Applications of Trees

We will discuss three problems that can be studied using trees.

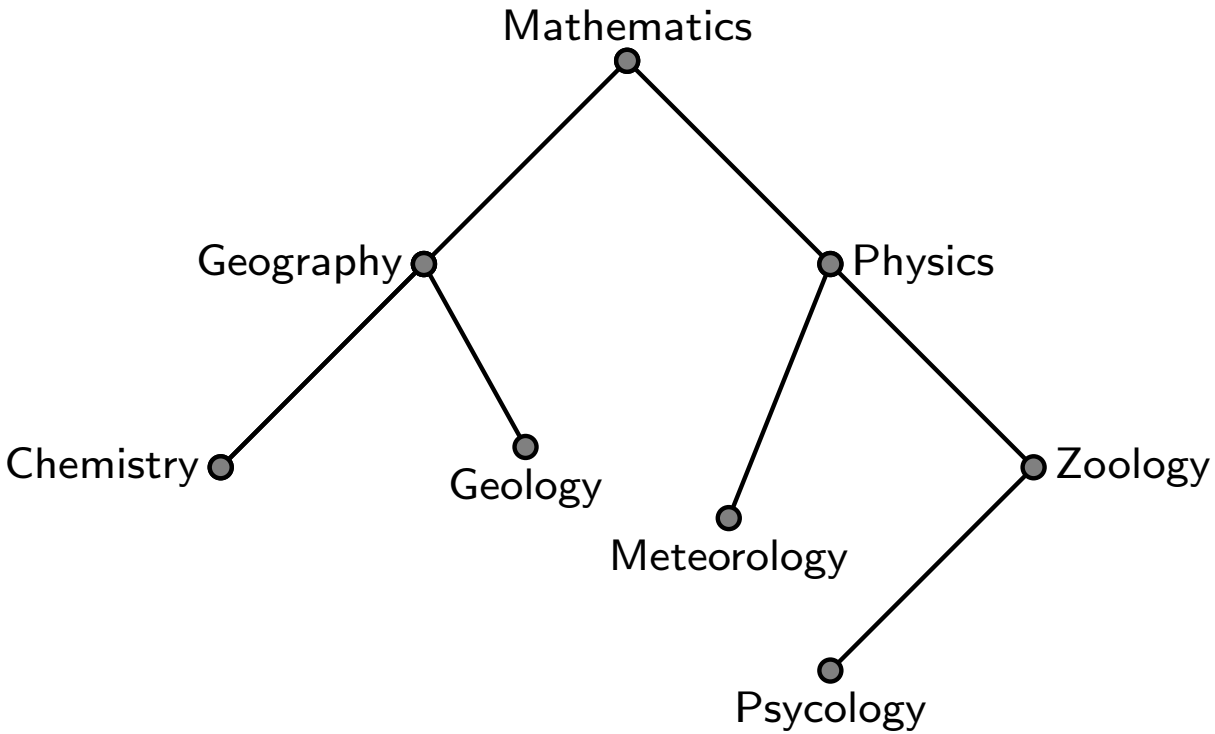
1. The first problem is: How should items in a list be stored so that an item can be easily located?
2. The second problem is: What series of decisions should be made to find an object with a certain property in a collection of objects of a certain type?
3. The third problem is: How should a set of characters be efficiently coded by bit strings?

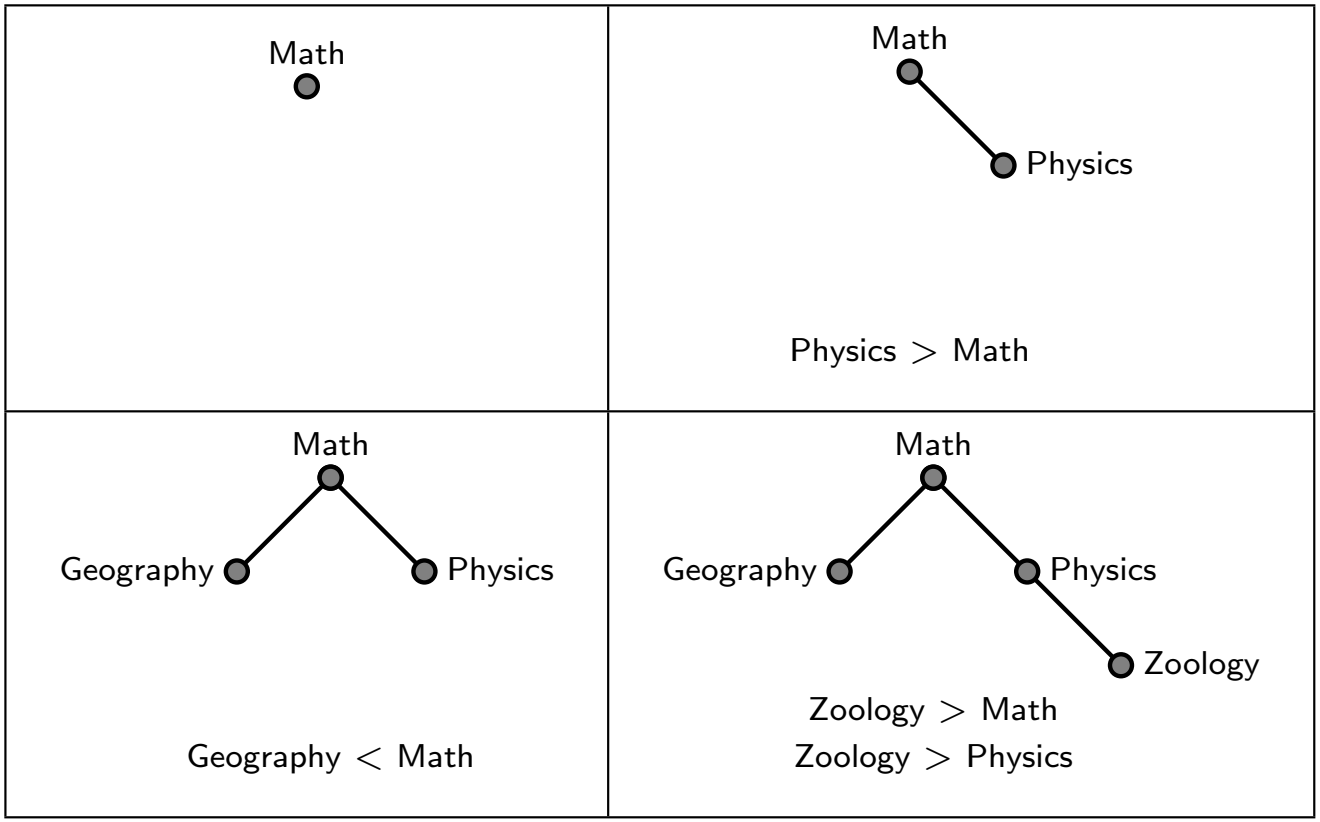
Binary Search Trees

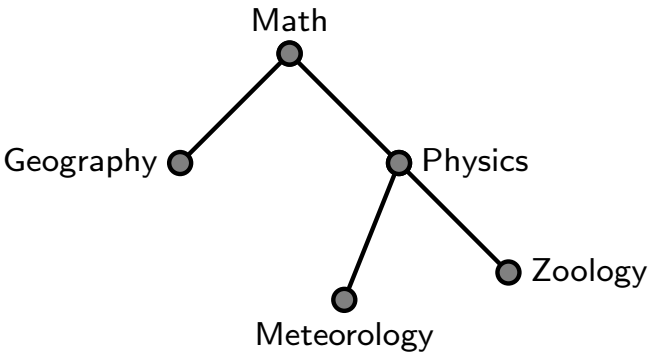
1. Searching for items in a list is one of the most important tasks that arises in computer science. Our primary goal is to implement a searching algorithm that finds items efficiently when the items are totally ordered. This can be accomplished through the use of a binary search tree.
2. A binary search tree is a binary tree in which each child of a vertex is designated as a right or left child, no vertex has more than one right child or left child, and each vertex is labeled with a key, which is one of the items. Furthermore, vertices are assigned keys so that the key of a vertex is both larger than the keys of all vertices in its left subtree and smaller than the keys of all vertices in its right subtree.

Example

Form a binary search tree for the words: mathematics, Physics, Geography, Zoology, Meteorology, Geology, Psychology, and Chemistry (using alphabetical order).

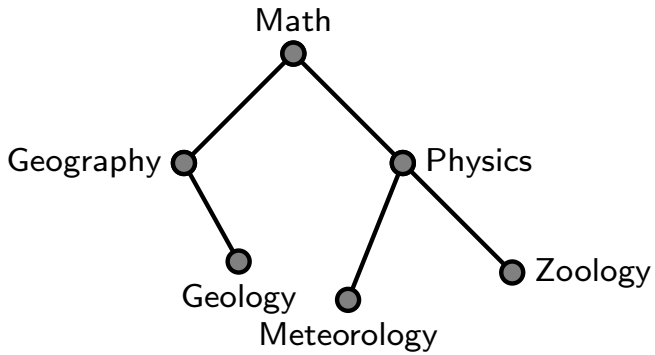



Math
Math
Physics
Physics > Math
Math
Geography
Physics
Geography < Math
Math
Geography
Physics
Zoology
Zoology > Math
Zoology > Physics



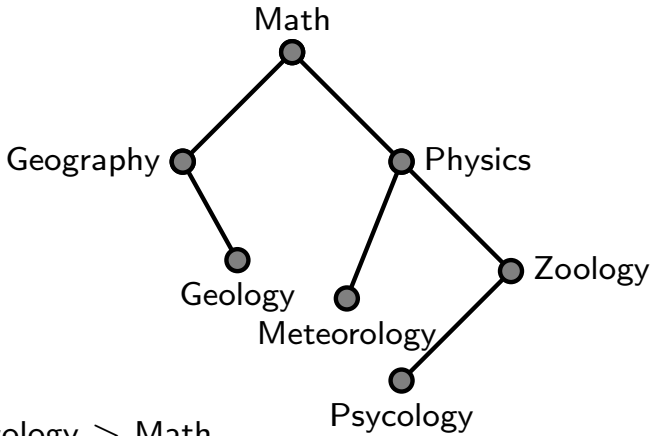
Meteorology $>$ Math

Meteorology $<$ Physics



Geology $<$ Math

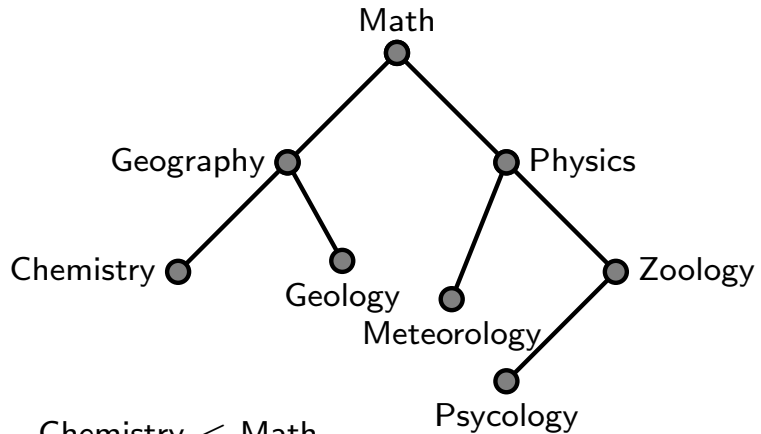
Geology $>$ Geography



Psycology $>$ Math

Psycology $>$ Physics

Psycology $<$ Zoology

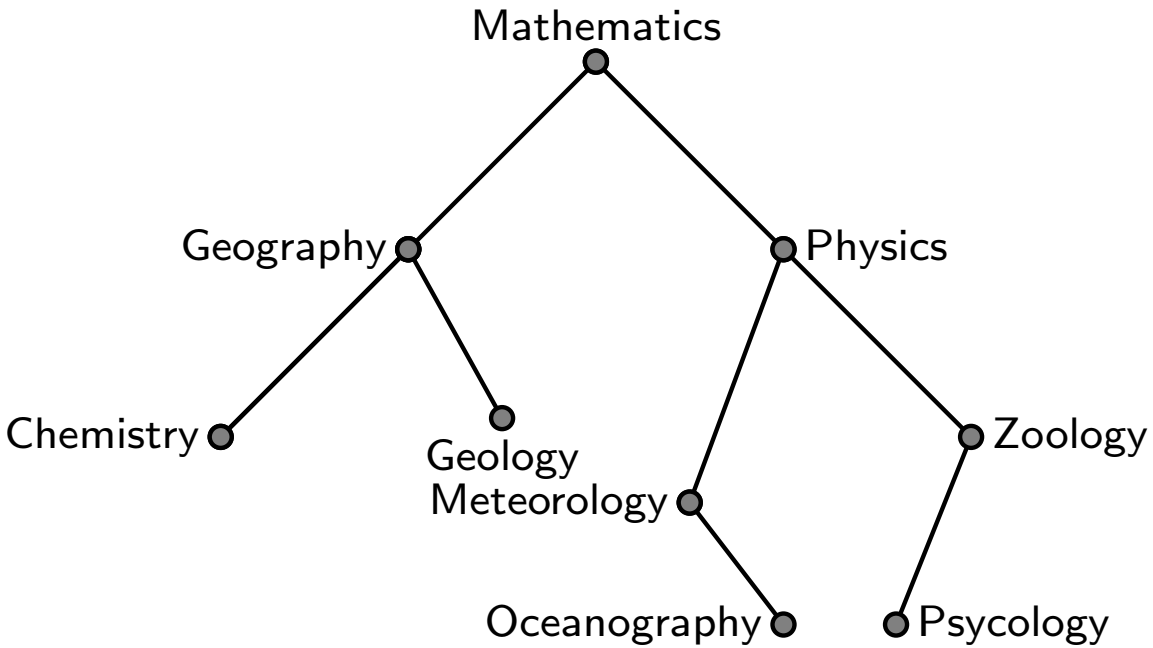


Chemistry $<$ Math

Chemistry $<$ Geography

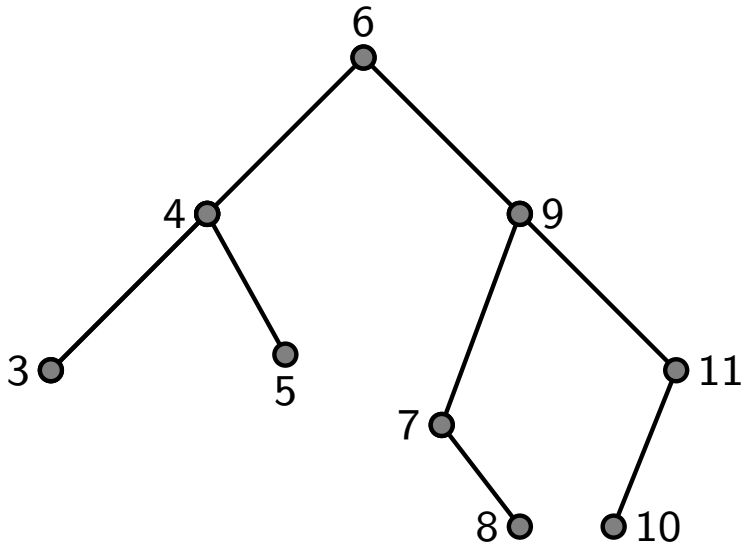
Example

Insert the word Oceanography into the binary search tree in the previous example.
mathematics $<$ oceanography, physics $>$ oceanography,
meteorology $<$ oceanography.



Example

Form a binary search tree for the numbers: 6, 9, 4, 11, 7, 5, 10, 3 and 8 (using the order on $\mathbb{N}$).



Decision Trees

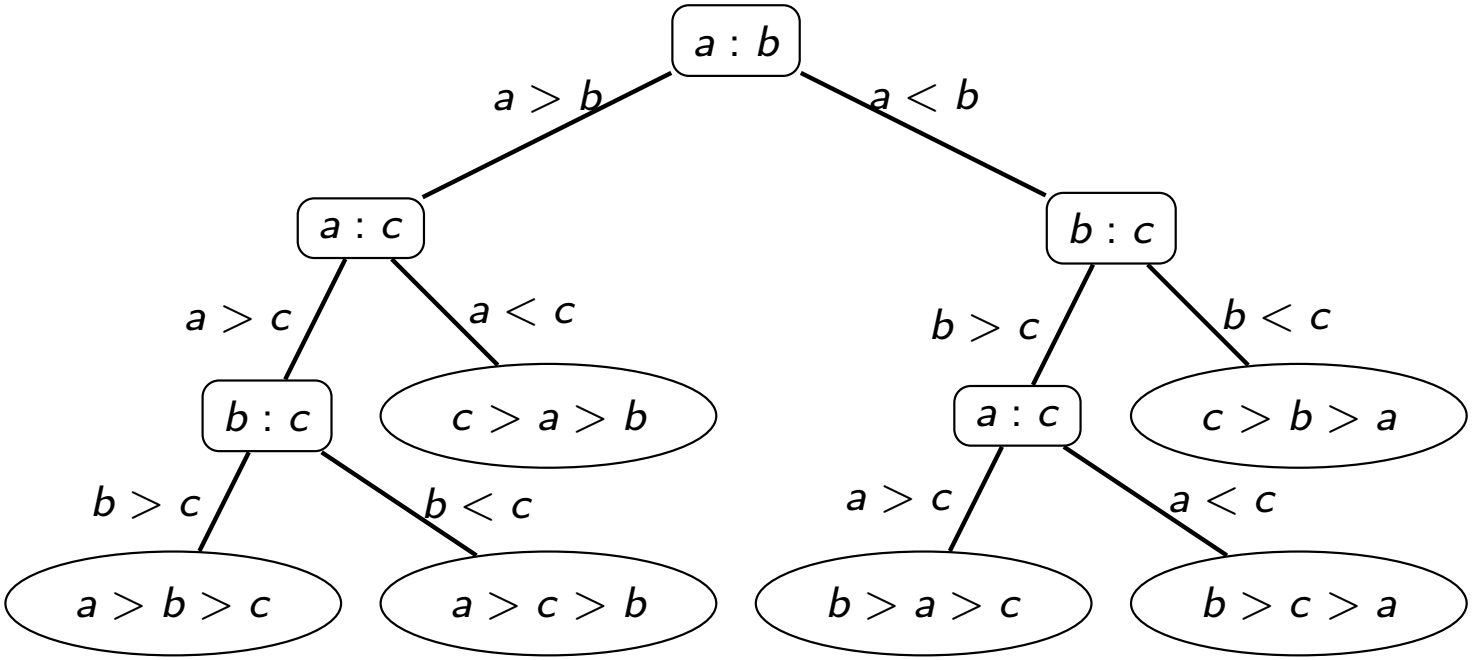
1. Rooted trees can be used to model problems in which a series of decisions leads to a solution.
2. For instance, a binary search tree can be used to locate items based on a series of comparisons, where each comparison tells us whether we have located the item, or whether we should go right or left in a subtree.
3. A rooted tree in which each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision, is called a **decision tree**.
4. The possible solutions of the problem correspond to the paths to the leaves of this rooted tree.

Example

Suppose there are seven coins, all with the same weight, and a counterfeit coin that weights less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one?
We do a weighing of the coins $(1, 2, 3)$ and $(3, 4, 5)$. If they have the same weight, we do an other weighing of the coins $(7, 8)$ and we have done. If $(1, 2, 3)$ is lighter than $(3, 4, 5)$, we do an other weighing of the coins $(1, 2)$ and we have done.

Example

A decision tree that orders the elements of the list a, b, c.



Prefix Codes

1. Consider using bit strings of different lengths to encode letters.
2. When letters are encoded using varying numbers of bits, some method must be used to determine where the bits for each character start and end.
3. One way to ensure that no bit string corresponds to more than one sequence of letters is to encode letters so that the bit string for a letter never occurs as the first part of the bit string for another letter. Codes with this property are called **prefix codes**.

4. A prefix code can be represented using a binary tree, where the characters are the labels of the leaves in the tree. The edges of the tree are labeled so that an edge leading to a left child is assigned a 0 and an edge leading to a right child is assigned a 1.

1. The bit string used to encode a character is the sequence of labels of the edges in the unique path from the root to the leaf that has this character as its label.
2. For instance, the tree in Figure (??) represents the encoding of e by 0, a by 10, t by 110, n by 1110, and s by 1111.

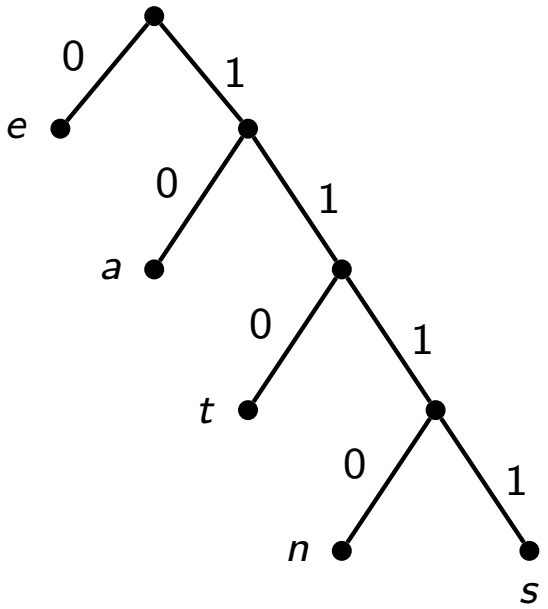


figure 1: A Binary Tree with a Prefix Code.

1. The tree representing a code can be used to decode a bit string. For instance, consider the word encoded by 11111011100 using the code in Figure (??). This bit string can be decoded by starting at the root, using the sequence of bits to form a path that stops when a leaf is reached.

Tree Traversal

1. Ordered rooted trees are often used to store information.
2. We need procedures for visiting each vertex of an ordered rooted tree to access data.
3. We will describe several important algorithms for visiting all the vertices of an ordered rooted tree.
4. Ordered rooted trees can also be used to represent various types of expressions, such as arithmetic expressions involving numbers, variables, and operations.

Traversal Algorithms

Procedures for systematically visiting every vertex of an ordered rooted tree are called traversal algorithms. We will describe three of the most commonly used such algorithms.

1. **preorder traversal**,**R**oot, **L**eft, **R**ight.
2. **inorder traversal**, **L**eft, **R**oot, **R**ight.
3. **postorder traversal**, **L**eft, **R**ight, **R**oot.

Preorder Traversal

Definition

Let T be an ordered rooted tree with root r. If T consists only of r, then r is the preorder traversal of T . Otherwise, suppose that $T_1, T_2, \cdots, T_n$ are the subtrees at r from left to right in T . The preorder traversal begins by visiting r. It continues by traversing T_1 in preorder, then T_2 in preorder, and so on, until T_n is traversed in preorder.

Example

In which order does a preorder traversal visit the vertices in the ordered rooted tree T shown in Figure (??) below

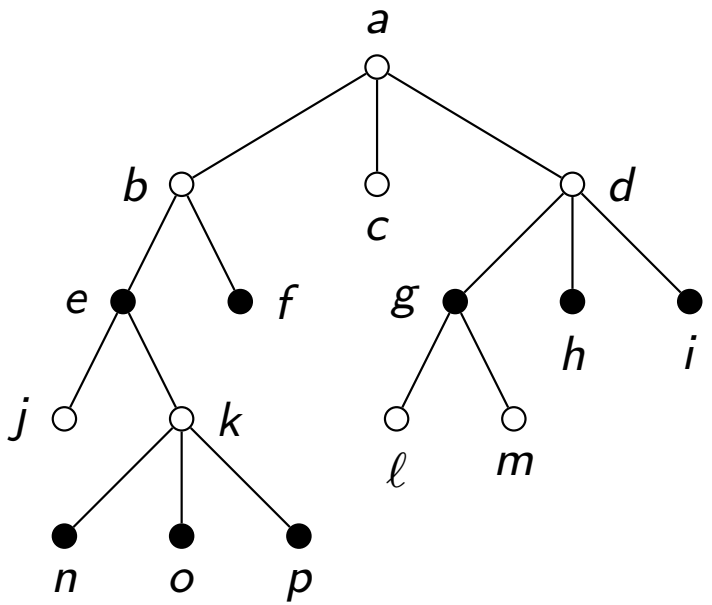
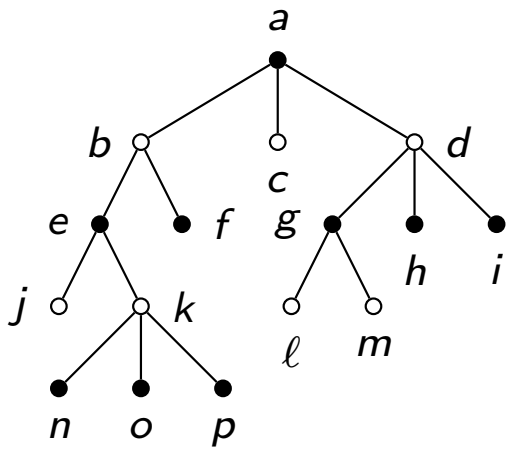
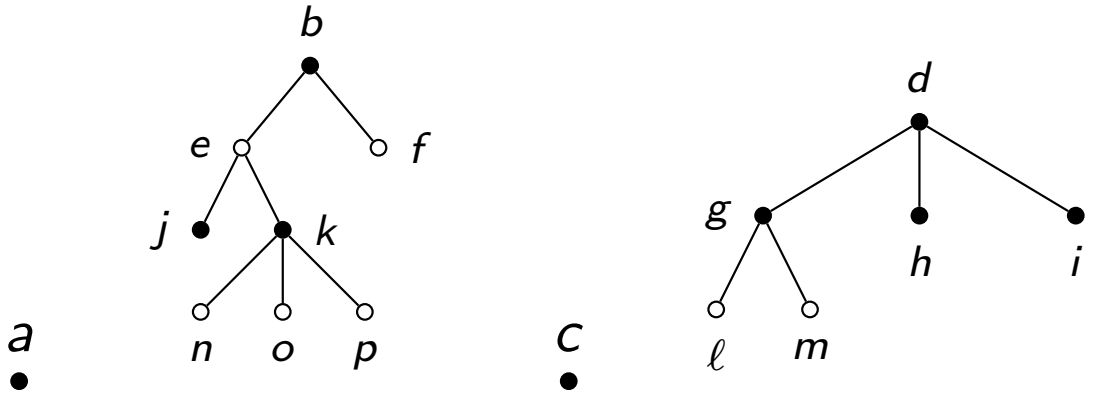


figure 2: The Ordered Rooted Tree T.

The steps of the preorder traversal of T are shown in Figure (??) below.





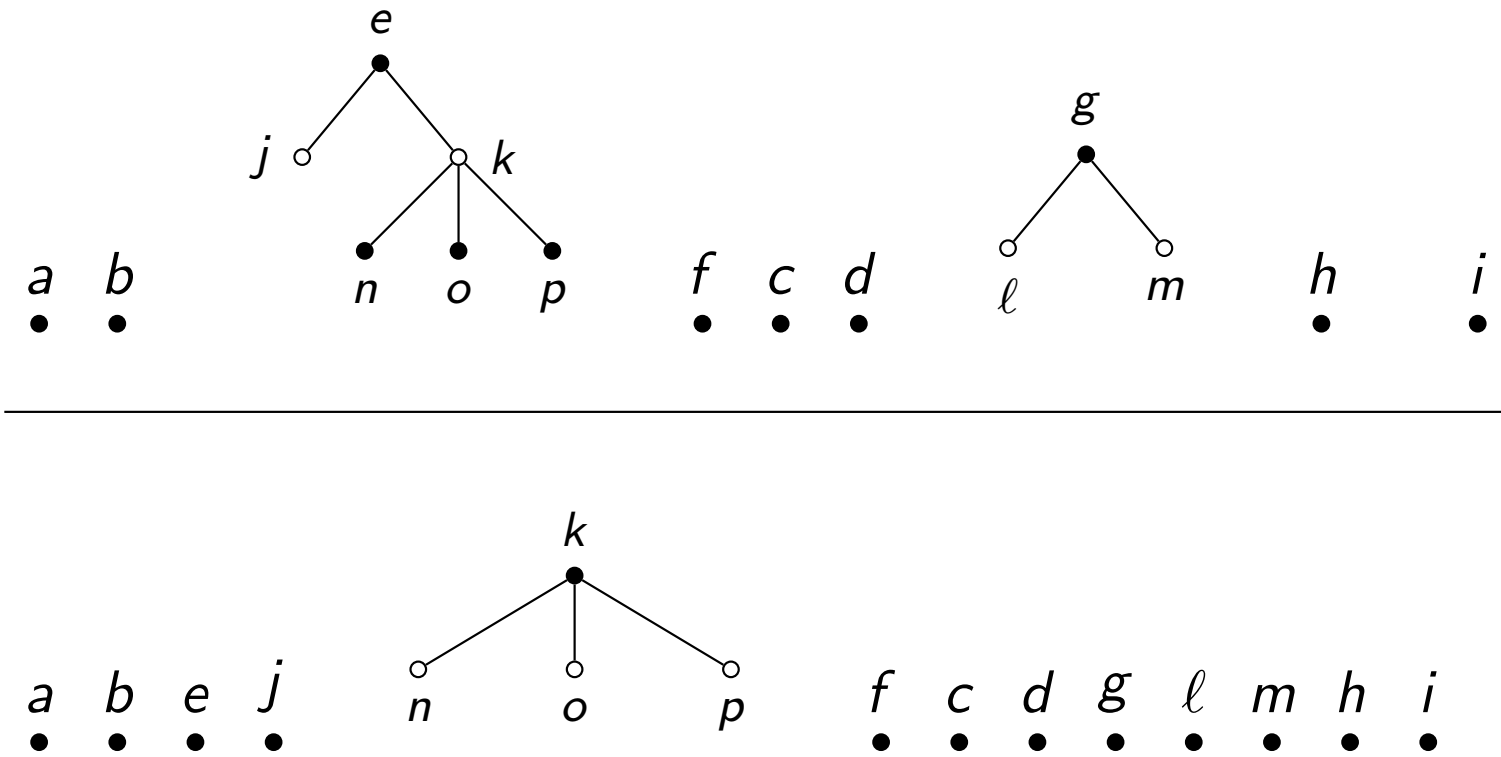


figure 3: The Preorder Traversal of T.

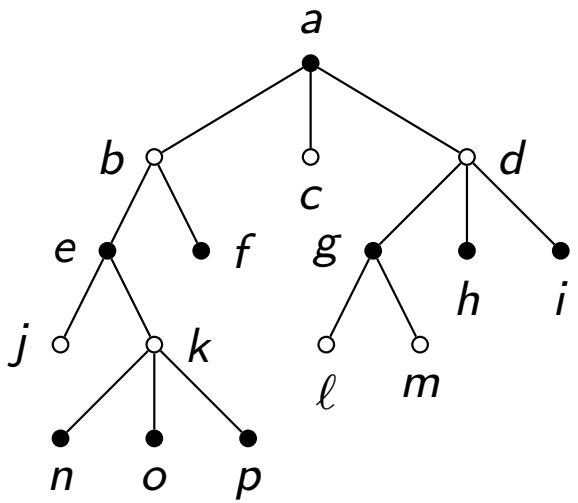
Inorder Traversal

Definition

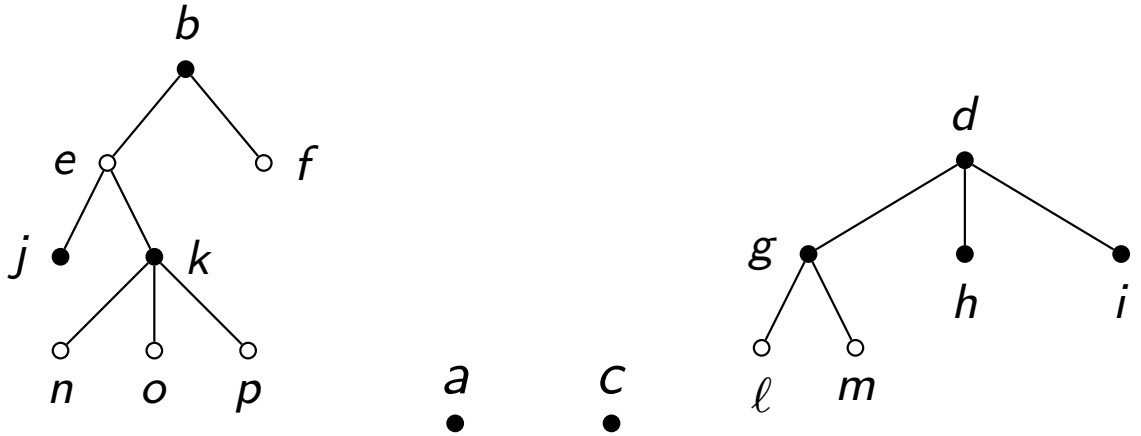
Let T be an ordered rooted tree with root r. If T consists only of r, then r is the inorder traversal of T . Otherwise, suppose that $T_1, T_2, \cdots, T_n$ are the subtrees at r from left to right. The inorder traversal begins by traversing T_1 in inorder, then visiting r. It continues by traversing T_2 in inorder, then T_3 in inorder, ... , and finally T_n in inorder.

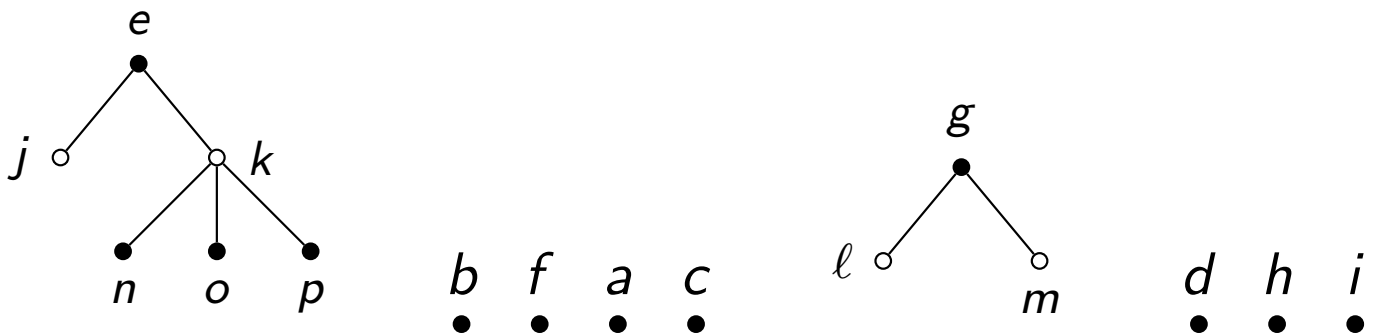
Example

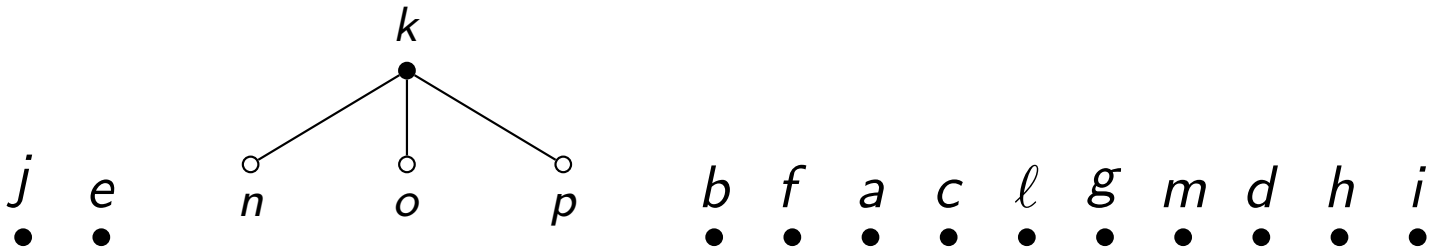
In which order does an inorder traversal visit the vertices of the ordered rooted tree T below?

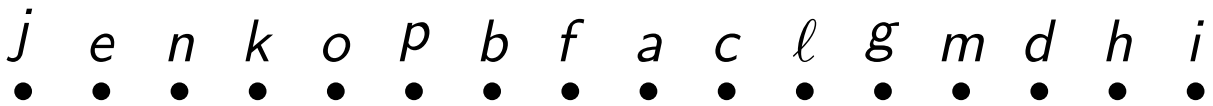


The steps of the inorder traversal of T are as fololows:









---

figure 4: The Inorder Traversal of T.

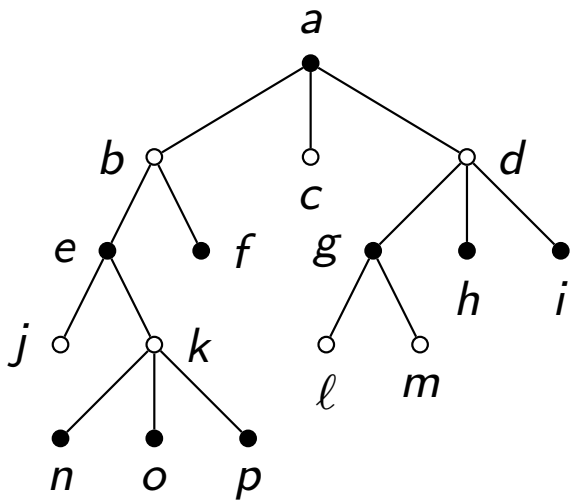
Postorder Traversal

Definition

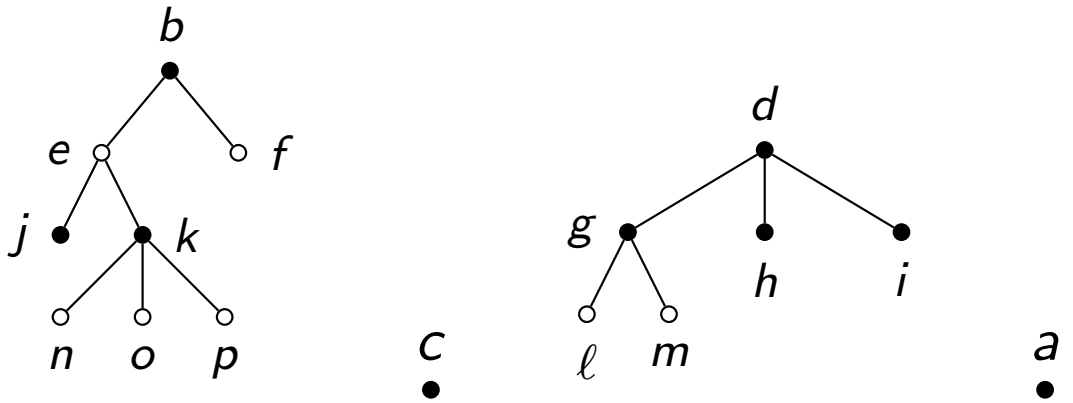
Let T be an ordered rooted tree with root r. If T consists only of r, then r is the postorder traversal of T . Otherwise, suppose that $T_1, T_2, \cdots, T_n$ are the subtrees at r from left to right. The postorder traversal begins by traversing T_1 in postorder, then T_2 in postorder, $\cdots$, then T_n in postorder, and ends by visiting r.

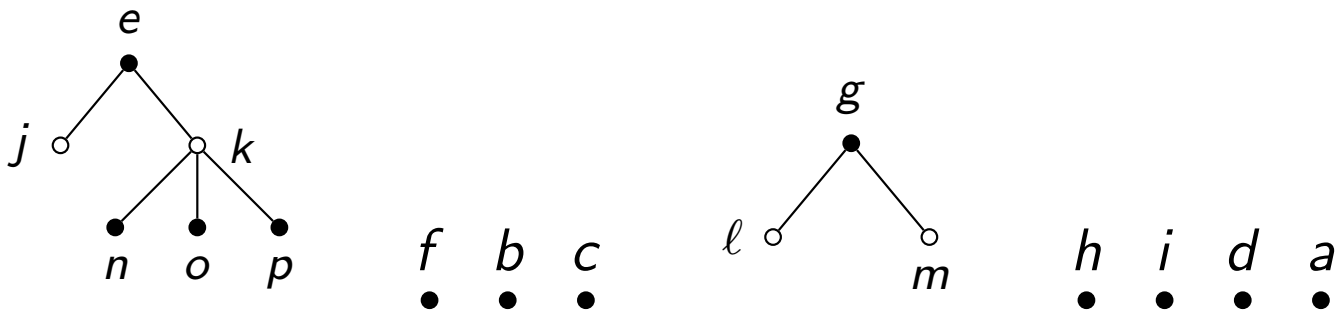
Example

In which order does a postorder traversal visit the vertices of the ordered rooted tree T shown below?



The steps of the postorder traversal of the ordered rooted tree T is as follows:





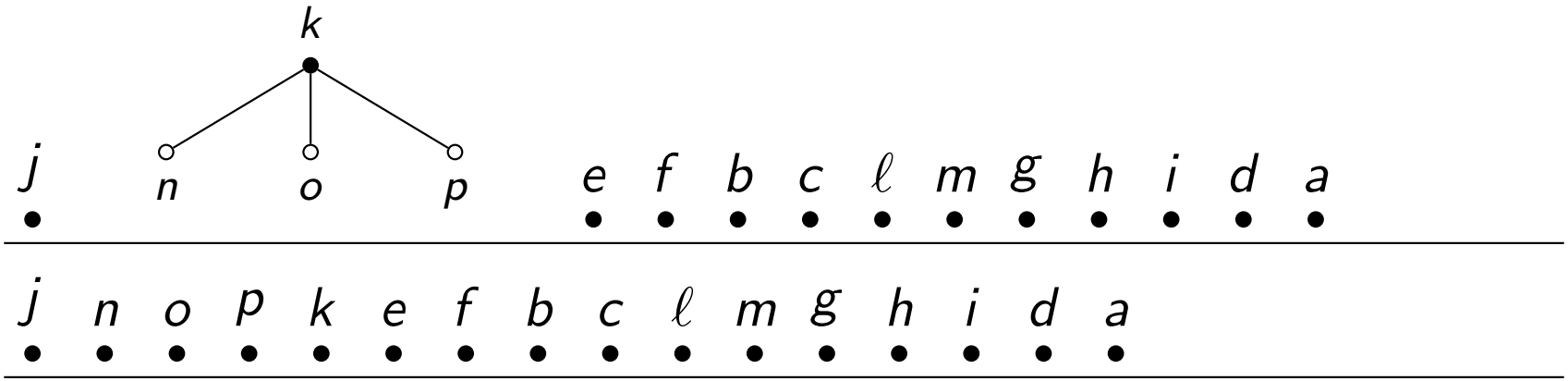


figure 5: The Postorder Traversal of T.

Infix, Prefix, and Postfix Notation

We can represent complicated expressions, such as compound propositions, combinations of sets, and arithmetic expressions using ordered rooted trees. For instance, consider the representation of an arithmetic expression involving the operators $+$ (addition), $-$ (subtraction), $*$ (multiplication), $/$ (division), and $\uparrow$ (exponentiation). We will use parentheses to indicate the order of the operations.An ordered rooted tree can be used to represent such expressions, where the internal vertices represent operations, and the leaves represent the variables or numbers. Each operation operates on its left and right subtrees (in that order).

Example

What is the ordered rooted tree that represents the expression $((x + y) \uparrow 2) + ((x - 4)/3)$?
The binary tree for this expression can be built from the bottom up. First, a subtree for the expression $x + y$ is constructed. Then this is incorporated as part of the larger subtree representing $(x + y) \uparrow 2$. Also, a subtree for $x - 4$ is constructed, and then this is incorporated into a subtree representing $(x - 4)/3$. Finally the subtrees representing $((x + y) \uparrow 2)$ and $((x - 4)/3)$ are combined to form the ordered rooted tree representing $((x + y) \uparrow 2) + ((x - 4)/3)$. These steps are shown as follows (Figure **??**).

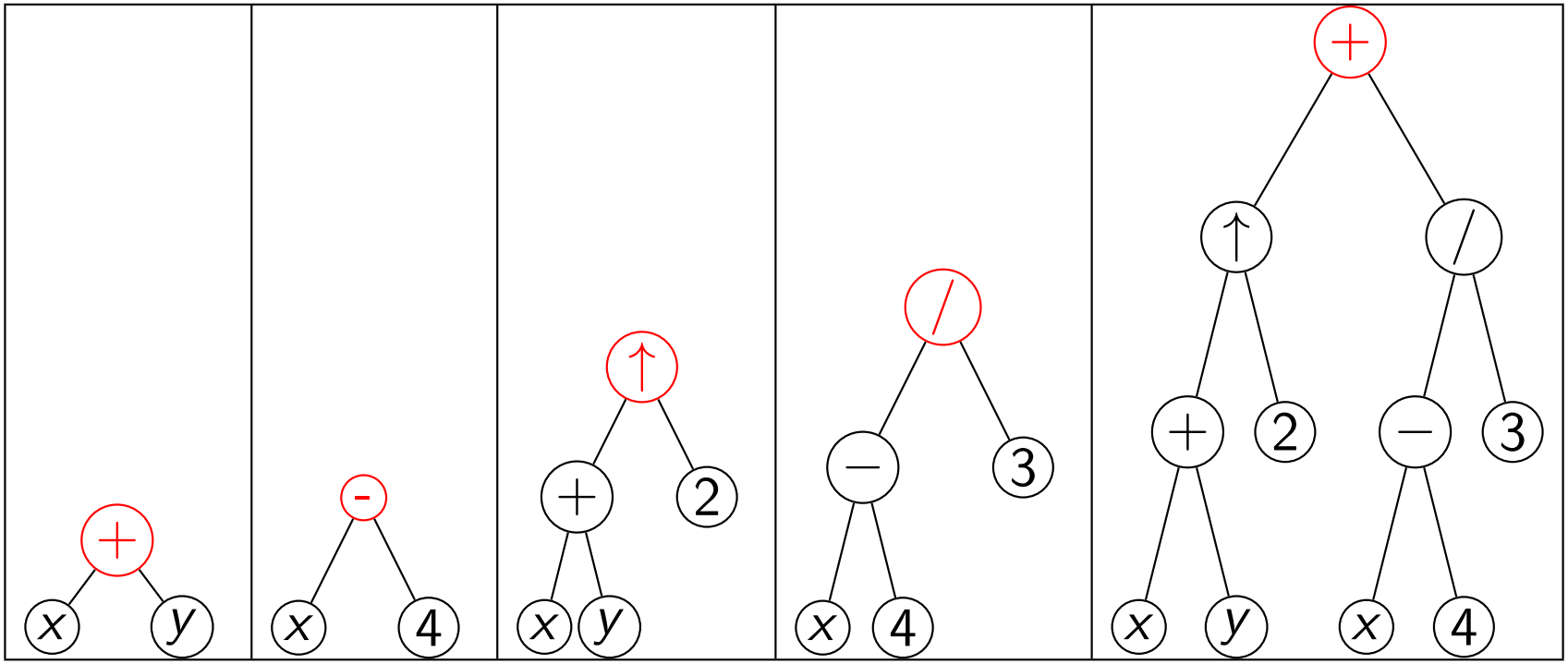


figure 6: A Binary Tree Representing $((x+y)^2)+(\frac{x-4}{3})$.

An inorder traversal of the binary tree representing an expression produces the original expression with the elements and operations in the same order as they originally occurred, except for unary operations, which instead immediately follow their operands. For instance, inorder traversals of the binary trees in Figure ??, which represent the expressions $\frac{x+y}{x+3}$, $(x+(\frac{y}{x}))+3$, and $x+(\frac{y}{x+3})$, all lead to the infix expression $\frac{x+y}{x+3}$.

To make such expressions unambiguous it is necessary to include parentheses in the inorder traversal whenever we encounter an operation.

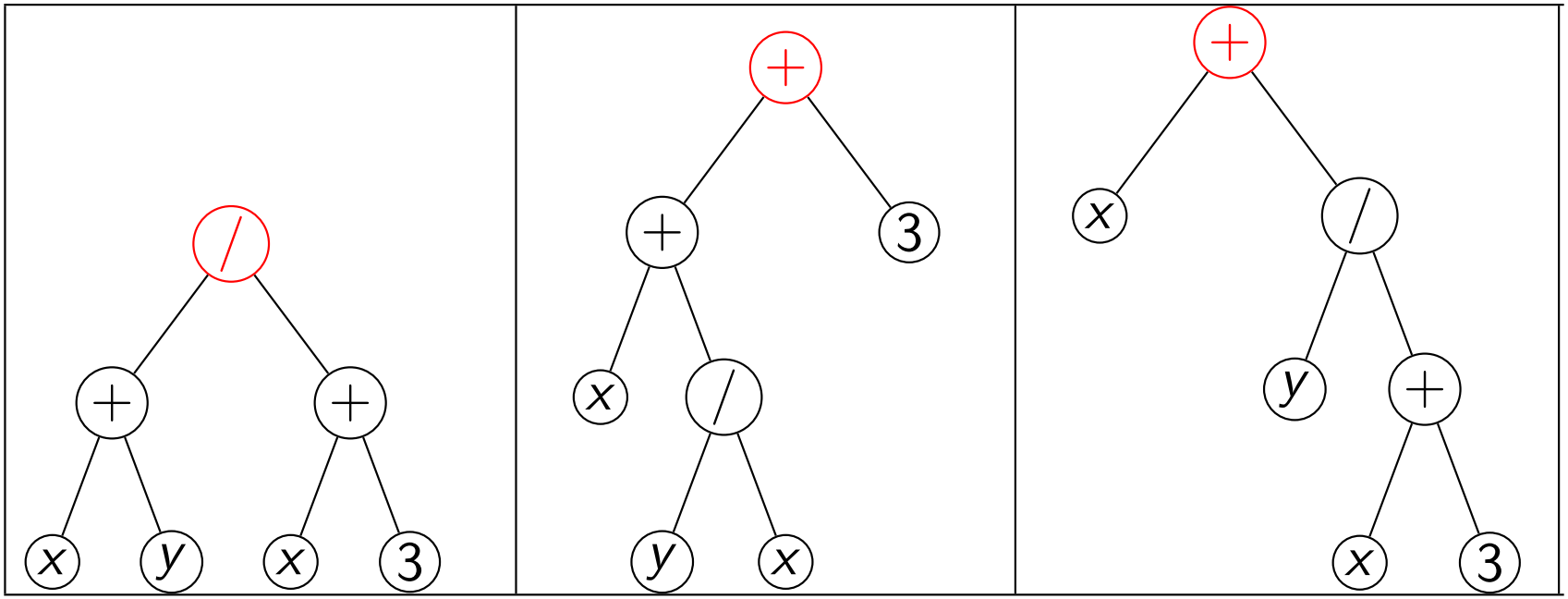
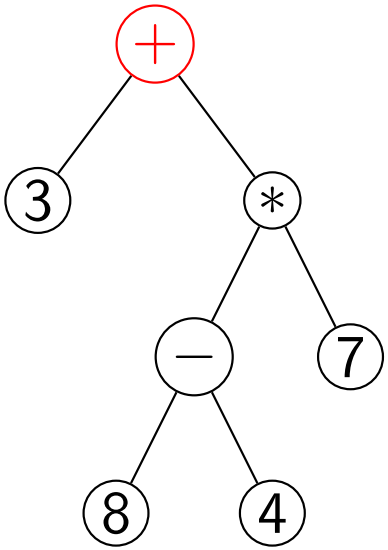


figure 7: Rooted Trees Representing $\frac{x+y}{x+3}$, $(x+\frac{y}{x})+3$, and $x+(\frac{y}{x+3})$.

- The fully parenthesized expression obtained in this way is said to be in **infix form**.
- We obtain the **prefix form** of an expression when we traverse its rooted tree in preorder.
- We obtain the **postfix form** of an expression by traversing its binary tree in postorder.

Order of evaluation



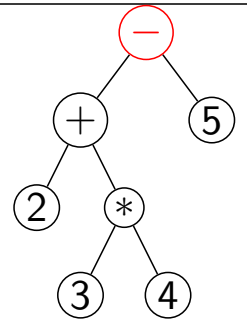
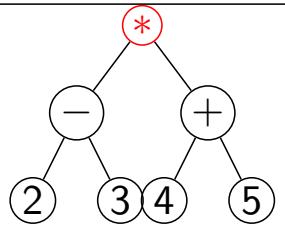
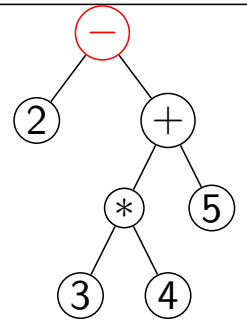
1. Prefix form: $+3 * -8\ 4\ 7$ (lecture)
   $+3 * (8-4)7 \rightarrow +3(4*7) \rightarrow +3\ 28 = 31$
2. Infix form: $3 + 8 - 4 * 7$ (lecture) $3 + ((8-4) * 7) = 31$.
3. Postfix form: $3\ 8\ 4 - 7 * +$ (lecture)
   $3(8-4)7*+ \rightarrow 347*+ \rightarrow 3(4*7)+ \rightarrow 3\ 28+ = 28+3 = 31$.

Evaluations of Arithmetic Expressions

Expression	Prefix forms	Infix forms	Postfix forms
(a + b)	+ a b	a + b	a b +
a - (b * c)	- a * b c	a - b * c	a b c * -

Remarks

1. A prefix or postfix form corresponds to exactly one expression tree.
2. An infix form may correspond to more than one expression tree. It is therefore not suitable for expression evaluation.

Expression	Expression Tree	Infix form
$2 - 3 * 4 + 5$		$2 - 3 * 4 + 5$
$(2 - 3) * (4 + 5)$		$2 - 3 * 4 + 5$
$2 - (3 * 4 + 5)$		$2 - 3 * 4 + 5$

Spanning Trees of undirected graphs

Definition

For a simple undirected graph G, a spanning tree of G is a subgraph T of G such that T is a tree and T contains every vertex of G.

Theorem

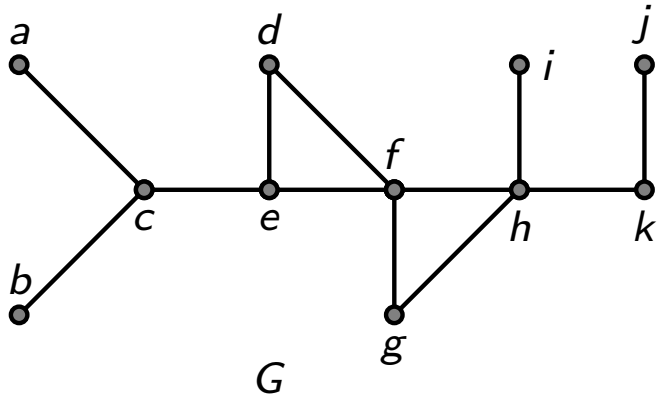
A simple graph is connected if and only if it has a spanning tree.

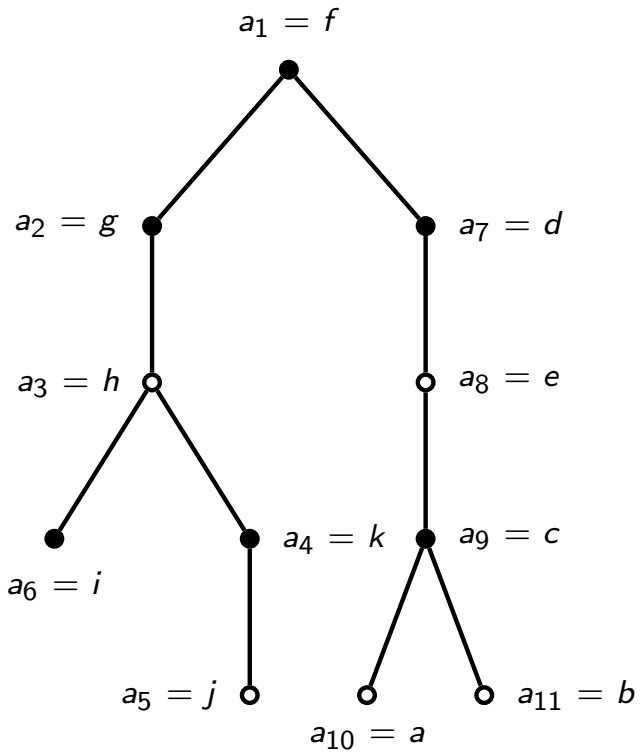
Depth-First Search

We can build a spanning tree for a connected simple graph using **depth-first search**. We will form a rooted tree, and the spanning tree will be the underlying undirected graph of this rooted tree. Arbitrarily choose a vertex of the graph as the root. Form a path starting at this vertex by successively adding vertices and edges, where each new edge is incident with the last vertex in the path and a vertex not already in the path. Continue adding vertices and edges to this path as long as possible. If the path goes through all vertices of the graph, the tree consisting of this path is a spanning tree. However, if the path does not go through all vertices, more vertices and edges must be added. Move back to the next to last vertex in the path, and, if possible, form a new path starting at this vertex passing through vertices that were not already visited.

If this cannot be done, move back another vertex in the path, that is, two vertices back in the path, and try again.
Repeat this procedure, beginning at the last vertex visited, moving back up the path one vertex at a time, forming new paths that are as long as possible until no more edges can be added. Because the graph has a finite number of edges and is connected, this process ends with the production of a spanning tree. Each vertex that ends a path at a stage of the algorithm will be a leaf in the rooted tree, and each vertex where a path is constructed starting at this vertex will be an internal vertex.
Depth-first search is also called backtracking, because the algorithm returns to vertices previously visited to add paths.

Example



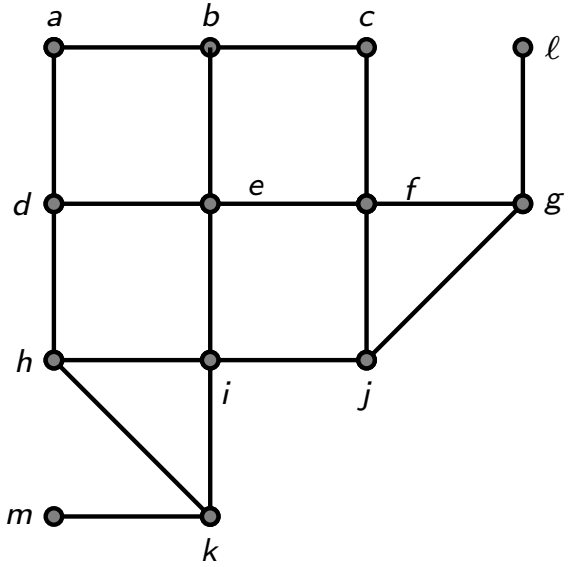


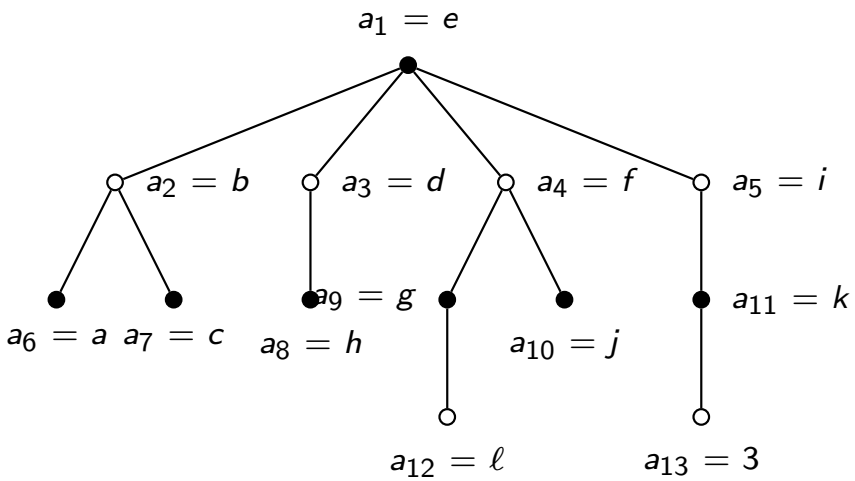
Breadth-First Search

We can also produce a spanning tree of a simple graph by the use of **breadth-first search.**
Arbitrarily choose a root from the vertices of the graph. Then add all edges incident to this vertex. The new vertices added at this stage become the vertices at level 1 in the spanning tree.
Arbitrarily order them. Next, for each vertex at level 1, visited in order, add each edge incident to this vertex to the tree as long as it does not produce a simple circuit. Arbitrarily order the children of each vertex at level 1.

This produces the vertices at level 2 in the tree. Follow the same procedure until all the vertices in the tree have been added. The procedure ends because there are only a finite number of edges in the graph. A spanning tree is produced because we have produced a tree containing every vertex of the graph.

Example


a
b
c
ℓ
d
e
f
g
h
i
j
m
k


$a_1 = e$
$a_2 = b$
$a_3 = d$
$a_4 = f$
$a_5 = i$
$a_6 = a$
$a_7 = c$
$a_8 = h$
$a_9 = g$
$a_{10} = j$
$a_{11} = k$
$a_{12} = \ell$
$a_{13} = 3$