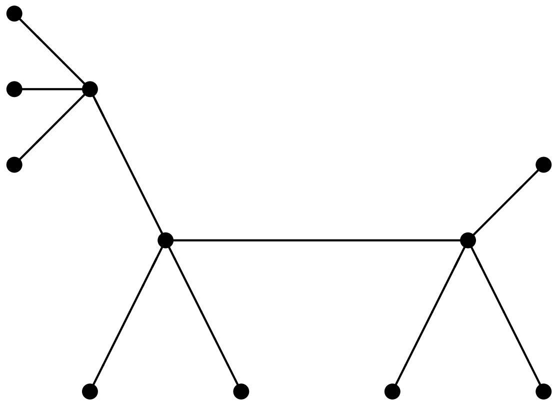
Special graphs: trees

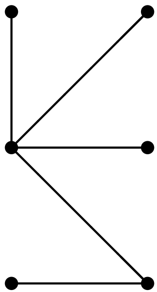
A **tree** is a connected simple graph with no simple cycles.



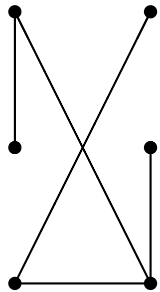
- In a tree there is a unique simple path between any two of its vertices.
- If we add an edge to a tree, it creates a cycle.
- If we remove an edge from a tree, it becomes not connected.

Trees: examples and non-examples

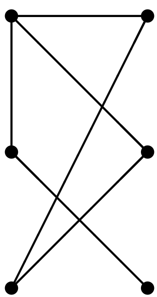
Trees:



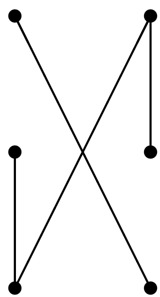
and



Not trees:

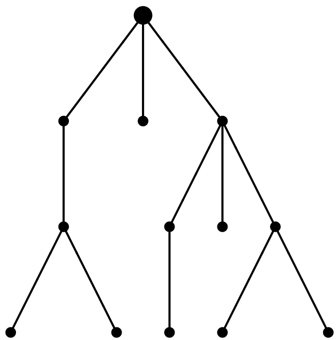


and



Rooted trees

A **rooted tree** is a tree in which one vertex has been designated as the root. We can change an unrooted tree to a rooted tree by choosing *any* vertex as the root. We usually draw a rooted tree with its root at the top:



Two rooted trees are **isomorphic** if there is a bijection between their vertices that

- takes the root to root, and
- takes edges to edges, and non-edges to non-edges.

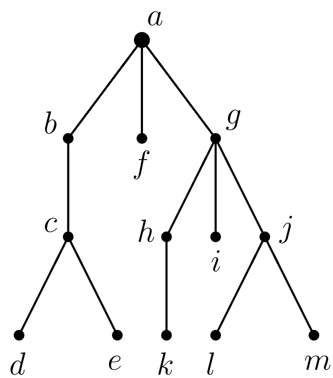
Rooted trees: basic terminology

The terminology for trees has botanical and genealogical origins.

- If vertices u and v are connected by an edge, and u is closer to the root than v (that is, above v), then
  - u is called the **parent** of v, and v is called a **child** of u.

  Vertices with the same parent are called **siblings**.
- A childless vertex is called a **leaf**.
- Vertices with at least one child are called **internal**.
- The **ancestors** of a non-root vertex v are the vertices in the (unique) simple path from the root to v.
- The **descendants** of vertex v are those vertices that have v as an ancestor.

Basic terminology: an example



- The root is a.
- The parent of c is b.
- The children of g are h, i, and j.
- The siblings of h are i and j.
- The ancestors of e are c, b, and a.
- The descendants of b are c, d, and e.
- The internal vertices are a, b, c, g, h, and j.
- The leaves are d, e, f, i, k, l, and m.

Applications of trees

Trees are used for modelling and problem solving in a wide variety of disciplines.

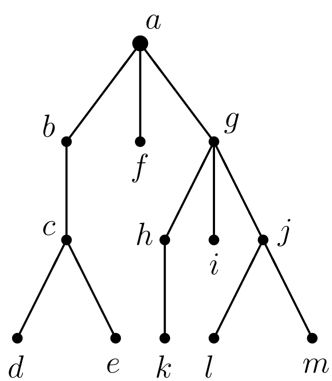
- Family trees in genealogy.
- Representing organisations.
- Computer file systems.
- Constructing efficient methods for locating items in a list: *binary search trees*.
- Game trees to analyse winning strategies in games.
- Decision trees.
- *Decomposition trees* to parse arithmetical and logic formulas and expressions.
- ...

Rooted trees: depth of a vertex

- The **depth** (or **level**) of a vertex v is the length of the (unique) path from the root to v.

  The depth of the root is 0.

FOR EXAMPLE:



depth of a is 0

depth of f is 1

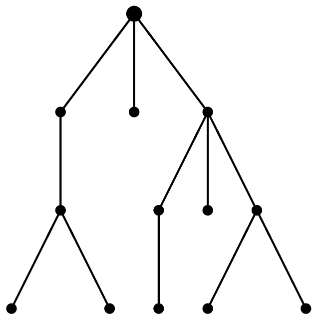
depth of j is 2

depth of e is 3

Rooted trees: height

- The **height** of a rooted tree is the maximum of the depths of its vertices.

FOR EXAMPLE:

height of  is 3

Rooted trees: subtrees

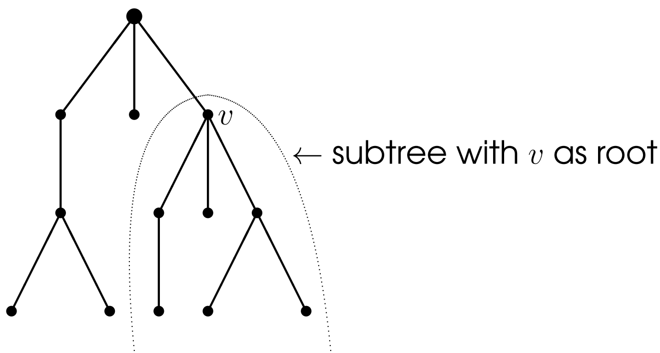
- If v is a vertex in a rooted tree T, the <u>**subtree** with v as its root</u>

  is the subgraph of T consisting of v,

  all its descendants,

  and all edges incident to these descendants.

FOR EXAMPLE:

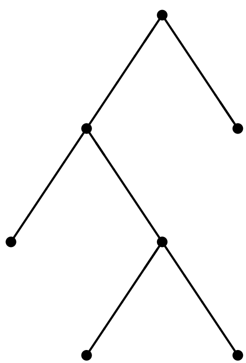


Special trees

- A rooted tree is called an <u>m-**ary tree**</u> if every internal vertex has no more than m children.
- A rooted tree is called a <u>**full** m-**ary tree**</u> if every internal vertex has exactly m children.

  A rooted tree is called a <u>**full binary tree**</u> if every internal vertex has exactly 2 children: a **left child** and a **right child.**

FOR EXAMPLE: A full binary tree:



Counting vertices and edges of trees

- A full binary tree with n internal vertices contains $2n+1$ vertices altogether.

  WHY? Every vertex, except the root, is the child of an internal vertex.

  Because each of the n internal vertices has 2 children, there are $2n$ vertices in the tree other than the root.

- A full m-ary tree with n internal vertices contains $m \cdot n + 1$ vertices altogether.

- A tree with m vertices has $m - 1$ edges.

  WHY? It can be proved by induction on m.

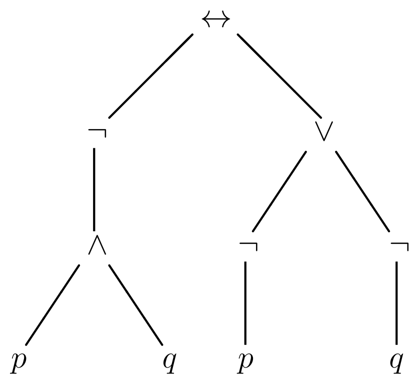
Example: decomposition tree of a logic formula

We can represent complicated expressions, like formulas of logic and arithmetical expressions, using rooted trees.

FOR EXAMPLE: The formula

$$\big(\neg(p \land q) \leftrightarrow (\neg p \lor \neg q)\big)$$

can be represented as



Binary search trees: a tool for sorting linearly ordered lists

Searching for items in a *linearly ordered list* is an important task. Binary search trees are particularly useful in representing elements in such a list. There are very efficient methods for

- *searching* data in binary search trees,
- *revising* data in binary search trees,
- converting lists to binary search trees and back.

Linearly ordered list: a sequence whose elements are linearly ordered (not necessarily in the order of listing)

FOR EXAMPLE:

- $(5, 128, 3, 2, 15, 4, 20)$ is a list of natural numbers, natural numbers can be ordered by the $\leq$ relation
- (*mathematics*, *physics*, *geography*, *geology*, *psychology*) is a list of words, words can be ordered by the *lexicographical order relation* $\prec$ (see next slide)

Example linear order: lexicographical order on words

First, we order the letters of the English alphabet as usual:

$$a \prec b \prec c \prec d \prec e \prec \ldots \prec x \prec y \prec z$$

Then, we can use this ordering of the letters to order longer words:

- Given two words w_1 and w_2, we compare them letter by letter, from left to right, passing equal letters.
- If at any point a letter in w_1 is $\prec$-smaller than the corresponding letter in w_2, then we put $w_1 \prec w_2$.
- If every letter in w_1 is equal to the corresponding letter in w_2, but w_2 is longer than w_1, then we also put $w_1 \prec w_2$.
- In any other case, we put $w_2 \prec w_1$.

FOR EXAMPLE: *discreet* $\prec$ *discreetness* $\prec$ *discrete* $\prec$ *discretion*

geography $\prec$ *geology* $\prec$ *mathematics* $\prec$ *physics* $\prec$ *psychology*

Binary search trees

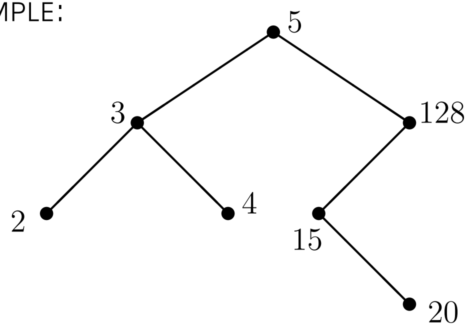
We are given a list L of items, and a linear order $\prec$ on them.

A **binary search tree** for L and $\prec$ is a binary tree in which every vertex is labelled with an item from L such that the label of each vertex:

- is $\prec$-greater than the labels of all vertices in its left subtree,
- is $\prec$-less than the labels of all vertices in its right subtree.

Also, every path in the tree is 'compatible with' the order of listing.

FOR EXAMPLE:



for the list $(5, 128, 3, 2, 15, 4, 20)$
and linear order $\leq$

How to build binary search trees from linearly ordered lists

We are given a list L of items, and a linear order $\prec$ on them.
We go through each member of the list, from left to right:

- **First item:** We assign it as the label of the root.
- **Comparing:** We take the next item on the list, and first we compare it with the labels of the 'old' vertices already in the tree, starting from the root and
  - moving to the left if the new item is $\prec$-less than the label of the respective 'old' vertex, if this 'old' vertex has a left child, or
  - moving to the right if the new item is $\prec$-greater than the label of the respective 'old' vertex, if this 'old' vertex has a right child.
- **Adding:**
  - When the new item is $\prec$-less than the label of an 'old' vertex and the vertex has no left child, then we insert a new left child to the 'old' vertex, and label it with the new item.
  - When the new item is $\prec$-larger than the label of an 'old' vertex and the vertex has no right child, then we insert a new right child to the 'old' vertex, and label it with the new item.

Building a binary search tree: an example

TASK: Build a binary search tree for the list of words

mathematics, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*,

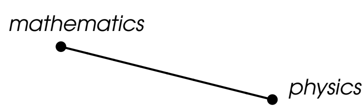
using lexicographical order $\prec$.

1ST STEP: We take *mathematics* and label the root with it:

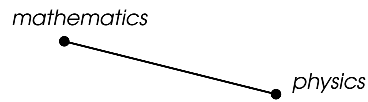
2ND STEP: We take *physics* and compare it with *mathematics*:

$$\textit{mathematics} \prec \textit{physics},$$

mathematics has no right child, so we label a new right child with *physics*:



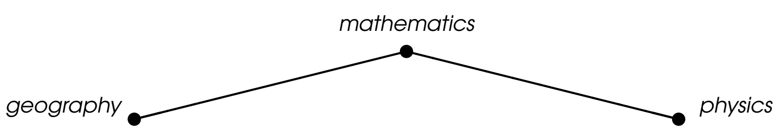
Building a binary search tree: an example (cont.)



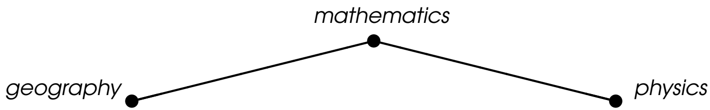
3RD STEP: We take *geography* and compare it with *mathematics*:

$$\textit{geography} \prec \textit{mathematics},$$

mathematics has no left child, so we label a new left child with *geography*:



Building a binary search tree: an example (cont.)



4TH STEP: We take *zoology* and compare it with *mathematics*:

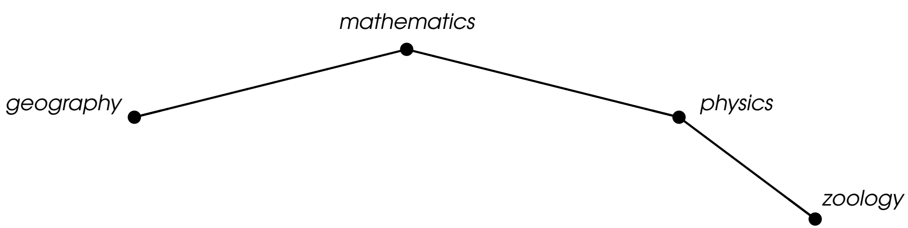
$$\textit{mathematics} \prec \textit{zoology},$$

so we move to the right child of the root and take its label, *physics*.

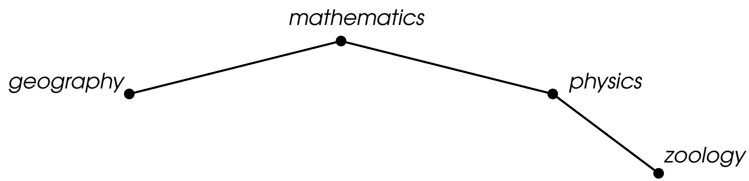
5TH STEP: We compare the new word *zoology* with *physics*:

$$\textit{physics} \prec \textit{zoology},$$

physics has no right child, so we label a new right child with *zoology*:



Building a binary search tree: an example (cont.)



6TH STEP: We take *meteorology* and compare it with *mathematics*:

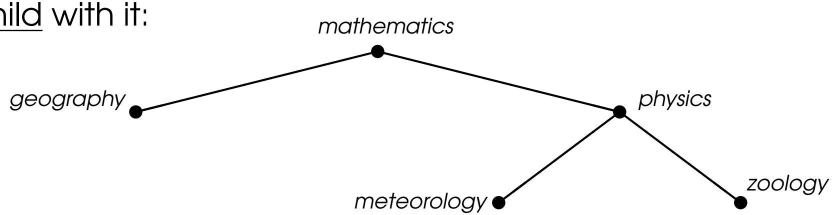
$$\textit{mathematics} \prec \textit{meteorology},$$

so we move to the right child of the root and take its label, *physics*.

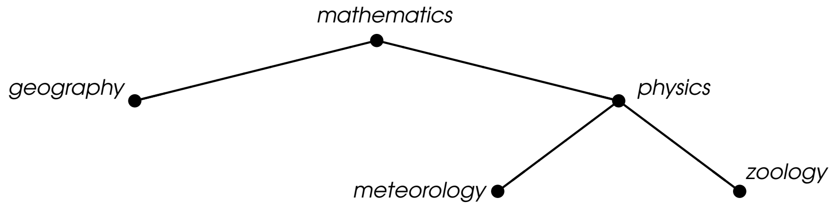
7TH STEP: We compare the new word *meteorology* with *physics*:

$$\textit{meteorology} \prec \textit{physics},$$

so we label a <u>new left child</u> with it:



Building a binary search tree: an example (cont.)



8TH STEP: We take *geology* and compare it with *mathematics*:

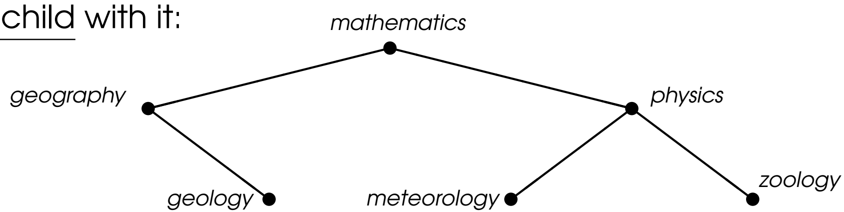
$$geology \prec mathematics,$$

so we move to the left child of the root and take its label, *geography*.

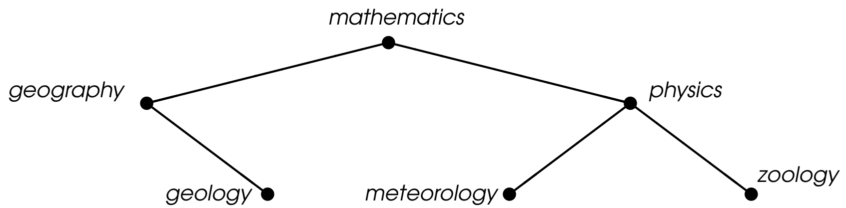
9TH STEP: We compare the new word *geology* with *geography*:

$$geography \prec geology,$$

so we label a <u>new right child</u> with it:



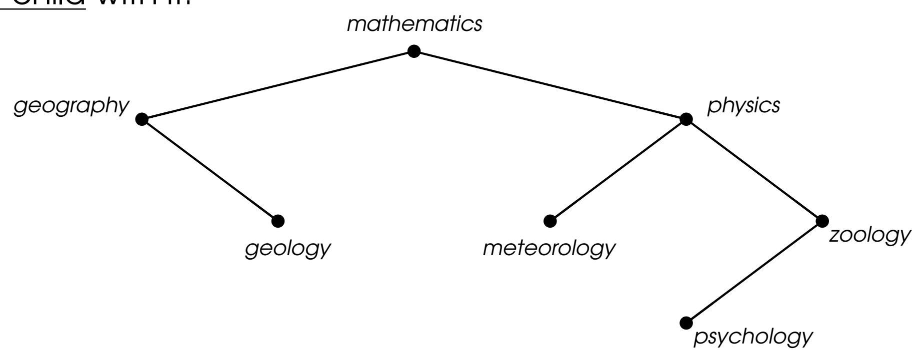
Building a binary search tree: an example (cont.)



FINALLY: We take *psychology*, then compare it with *mathematics*, move to the right, compare it with *physics*, move to the right, then compare it with *zoology*. As

$$\textit{psychology} \prec \textit{zoology},$$

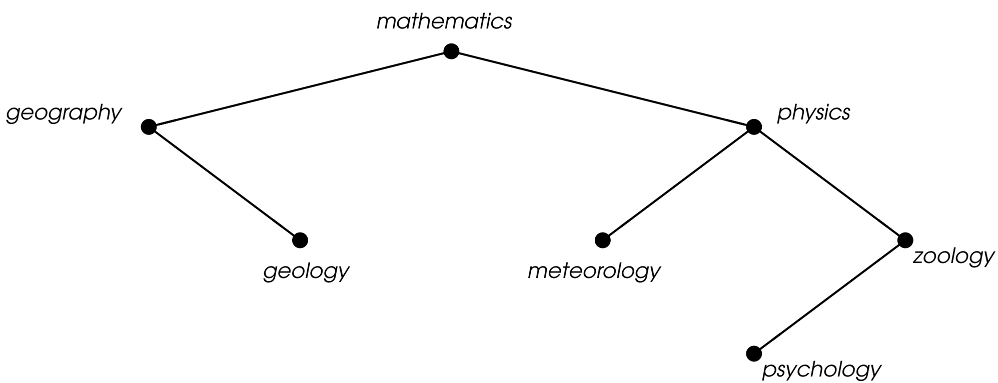
we label a new left child with it:



Binary search trees: locating or adding items I

TYPICAL TASK: We already have a binary search tree. We are given a word, *meteorology*. How many comparisons do we need to locate this word in our tree (if it is there), or to add to it (if it is new)?

SOLUTION: Just 3. We take the word. First compare it with *mathematics*, move to the right, then compare it with *physics*, move to the left, then compare it with *meteorology*: successfully located.



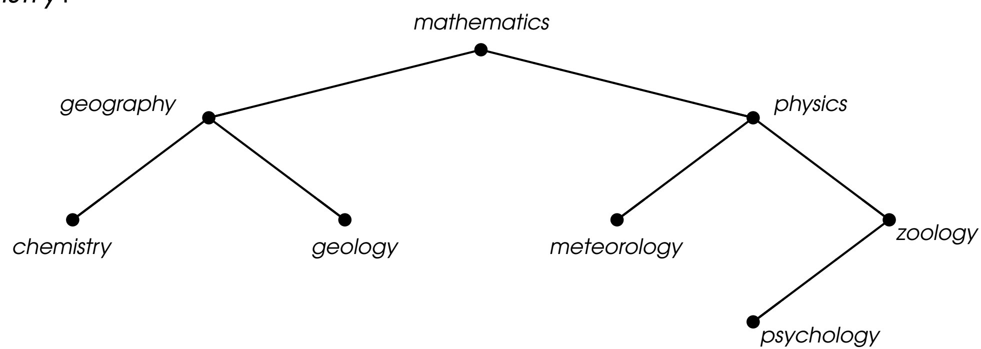
Binary search trees: locating or adding items II

TASK: We are given a word, *chemistry*. How many comparisons do we need to locate or add it?

SOLUTION: Just 2. We compare it with *mathematics*, move to the left, then compare it with *geography*. As

$$chemistry \prec geography,$$

and *geography* has no left child, we create a <u>new left child</u> and label it with *chemistry*:



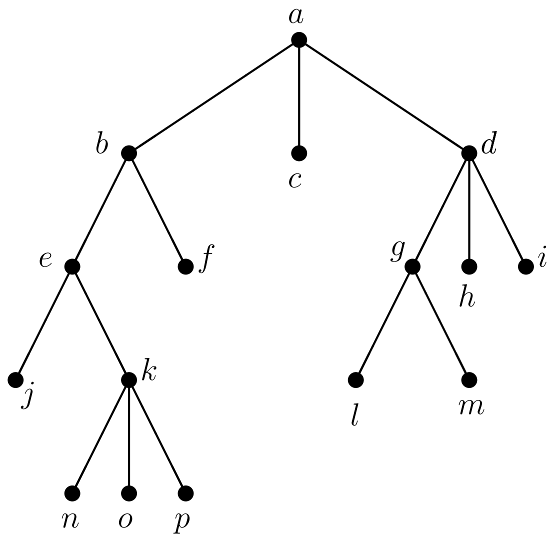
Tree traversal

Rooted trees are often used to store information. We need systematic procedures for visiting each vertex of a rooted tree to access data. Such procedures are called *traversal algorithms.*

Here are some important ones:

- **Preorder traversal:** Visit the root, then continue traversing subtrees in preorder, from left to right.
- **Inorder traversal:** Begin traversing leftmost subtree in inorder, then visit root, then continue traversing subtrees in inorder, from left to right.
- **Postorder traversal:** Begin traversing leftmost subtree in postorder, then continue traversing subtrees in postorder, from left to right, finally visit root.

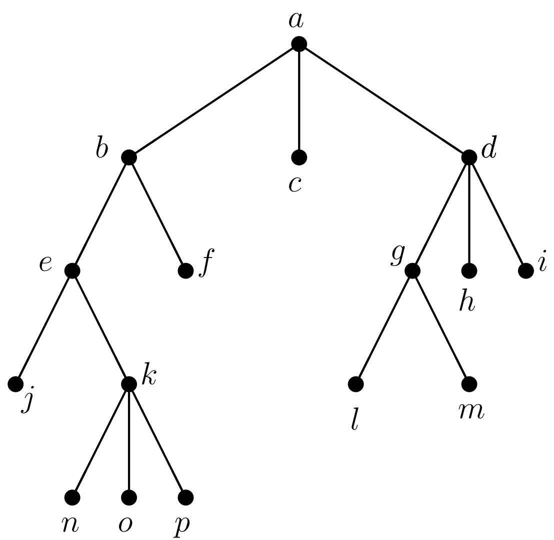
Preorder traversal



Visit the root, then continue traversing subtrees in preorder, from left to right:

$$a, \quad b, \quad e, \quad j, \quad k, \quad n, \quad o, \quad p, \quad f, \quad c, \quad d, \quad g, \quad l, \quad m, \quad h, \quad i$$

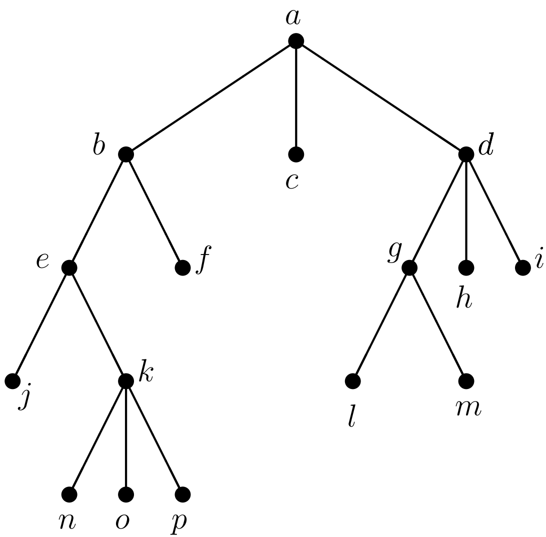
Inorder traversal



Begin traversing leftmost subtree in inorder, then visit root, then continue traversing subtrees in inorder, from left to right:

$$j, \quad e, \quad n, \quad k, \quad o, \quad p, \quad b, \quad f, \quad a, \quad c, \quad l, \quad g, \quad m, \quad d, \quad h, \quad i$$

Postorder traversal



Begin traversing leftmost subtree in postorder, then continue traversing subtrees in postorder, from left to right, finally visit root:

$$j, \quad n, \quad o, \quad p, \quad k, \quad e, \quad f, \quad b, \quad c, \quad l, \quad m, \quad g, \quad h, \quad i, \quad d, \quad a$$