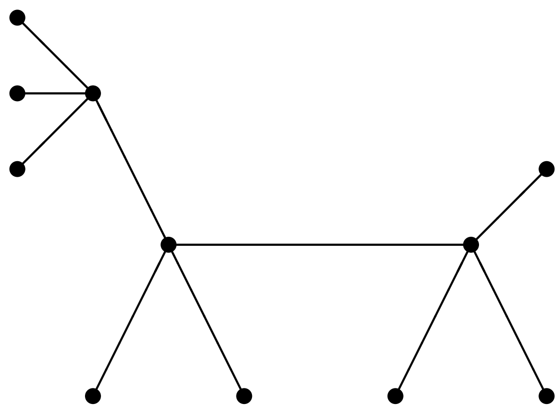
Special graphs: trees

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

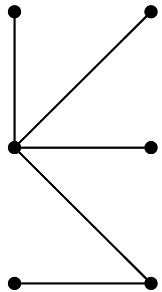


Some useful facts about trees:

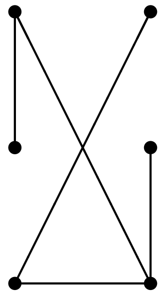
- In a tree there is a unique simple path between any two of its vertices.
- If we add an edge to a tree, it creates a cycle.
- If we remove an edge from a tree, it becomes not connected.

Trees: examples and non-examples

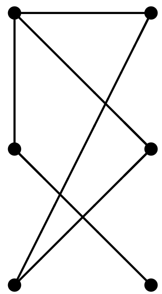
Trees:



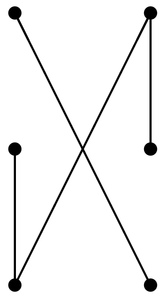
and



Not trees:

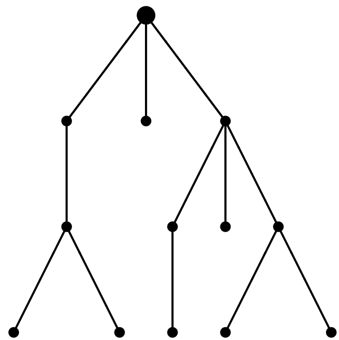


and



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.

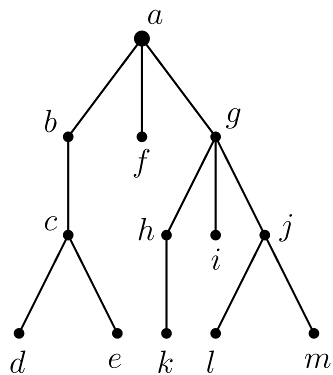
Rooted trees: basic terminology

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.

FOR EXAMPLE:

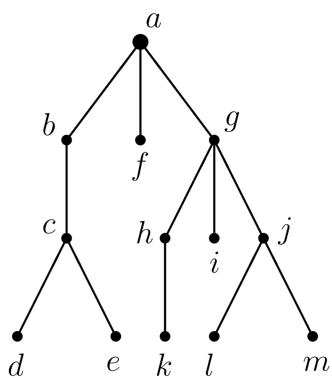
- family trees in genealogy
- representing organisations
- computer file systems
- constructing efficient methods for locating items in a list: *binary search trees*
- game trees to analyse winning strategies in games
- decision trees
- *decomposition trees* to parse arithmetical and logic formulas and expressions
- ...

Rooted trees: the level of a vertex

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

The level of the root is 0.

FOR EXAMPLE:



level of a is 0

level of f is 1

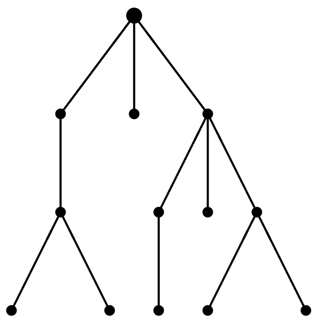
level of j is 2

level of e is 3

Rooted trees: height

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

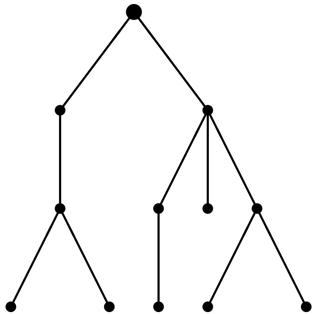
FOR EXAMPLE:

height of  is 3

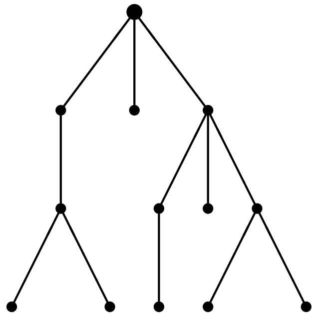
Balanced rooted trees

A rooted tree of height n is called **<u>balanced</u>**
if all its leaves are of level n or $n-1$.

FOR EXAMPLE:



balanced



not balanced

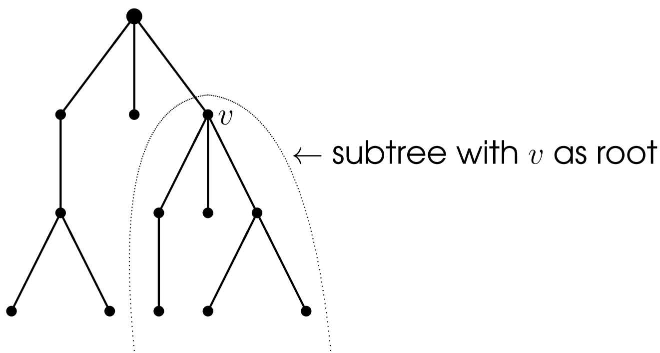
Rooted trees: subtrees

- If v is a vertex in a rooted tree T, the **subtree** with v as its root is the subgraph of T consisting of v,

  all its descendants,

  and all edges incident to these descendants.

FOR EXAMPLE:

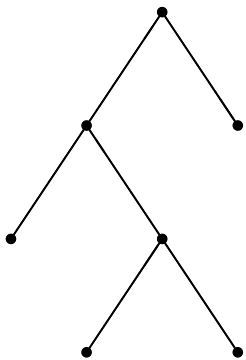


Special trees

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

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

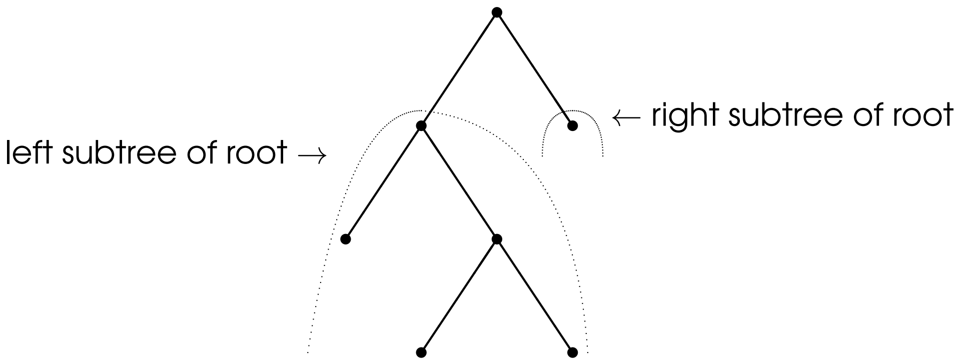
FOR EXAMPLE: A full binary tree:



Useful observations about full binary trees

Recall: A full binary tree is a rooted tree in which

every internal vertex has exactly 2 children.



When we have a full binary tree of height n then

- the left and right subtrees of the root are **both** full binary trees of height $\leq n-1$
- **at least one** of the left and right subtrees of the root is a full binary tree of height $n-1$ (but not necessarily both)

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.

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

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

Exercise 7.1

Prove by induction that, for every positive integer n, every full binary tree of height $\leq n$ has $\leq 2^n$ leaves.

$P(n)$: the number of leaves of any full binary tree of height $\leq n$ is $\leq 2^n$

SOLUTION: **Basis step:** We need to prove

$P(1)$: the number of leaves of any full binary tree of height ≤ 1 is $\leq 2^1$

So let's see. A full binary tree of height ≤ 1 is either of height 0, or of height 1.

- A full binary tree of height 0 is just a root, so it has 1 leaf and $1 \leq 2 = 2^1$.
- And a full binary tree of height 1 consists of a root and its two children, so it has 2 leaves and $2 \leq 2^1$.

Exercise 7.1 (cont.)

Inductive step: We need show that, for all positive integer k,

if $P(k)$ holds then $P(k+1)$ holds as well.

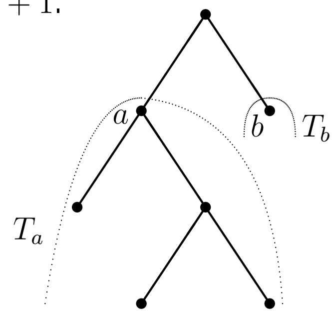
So suppose that for some positive integer k,

the number of leaves in any full binary tree of height $\leq k$ is $\leq 2^k$ **(IH).**

We need to show that

the number of leaves in any full binary tree of height $\leq k+1$ is $\leq 2^{k+1}$.

So let T be an arbitrary full binary tree of height $\leq k+1$.



- Take the two children of the root of T, say, a and b. Let T_a denote the subtree with root a, and T_b denote the subtree with root b. Then both T_a and T_b are full binary trees of height $\leq k$ (see lecture slide 197).
- Thus, by the IH, T_a has $\leq 2^k$ leaves, and T_b has $\leq 2^k$ leaves as well.
- As the leaves of T consists of all the leaves in T_a plus all the leaves in T_b, T has $\leq 2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$ leaves, as required.

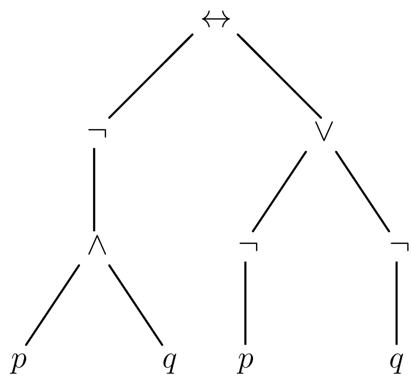
Example: decomposition tree of a logic formula

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

FOR EXAMPLE: The formula

$$\big(\neg(p \wedge q) \leftrightarrow (\neg p \vee \neg q)\big)$$

can be represented as



Binary search trees: a tool for sorting linearly ordered lists

Linearly ordered list: a sequence (list) 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 (which is a linear order, see lecture slide 92)
- $(\textit{mathematics}, \textit{physics}, \textit{geography}, \textit{geology}, \textit{psychology})$ is a list of words, words can be ordered by the *lexicographical order relation* $\prec$ (see lecture slide 203)

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 linearly ordered lists to binary search trees and back.

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 two things: a list $\boxed{L}$ of items, and a linear order $\boxed{\prec}$ on them.

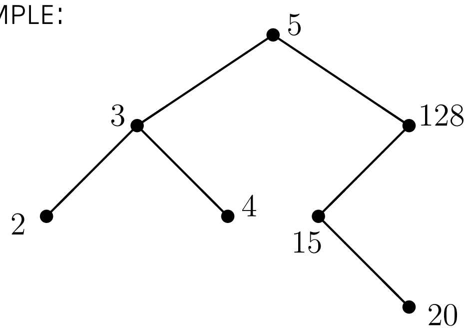
A <u>**binary search tree** for L and $\prec$</u> is a binary tree in which every vertex is labelled with an item from L such that:

(1) 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.

(2) 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 $\boxed{\prec}$.

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

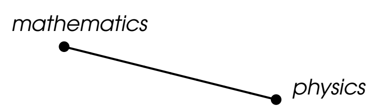
↓

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

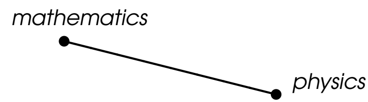
2ND STEP: We take *physics* and compare it with *mathematics*:

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

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



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



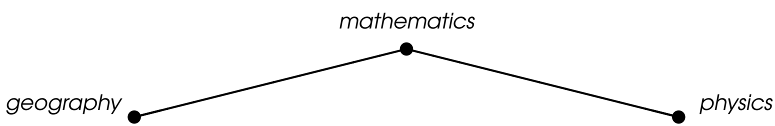
$\downarrow$

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

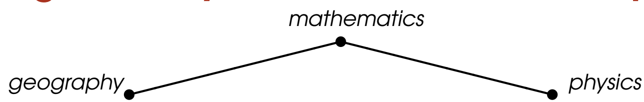
3RD STEP: We take *geography* and compare it with *mathematics*:

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

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



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



$\downarrow$

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

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

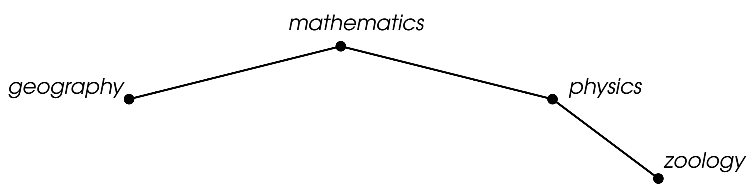
$$\textit{mathematics} \prec \textit{zoology},$$

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

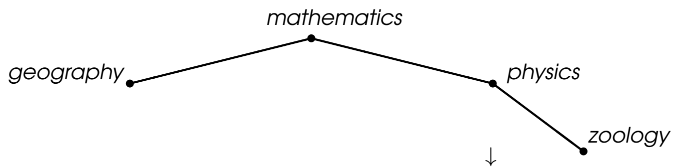
5TH STEP: We compare the new word *zoology* with *physics*:

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

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



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



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

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

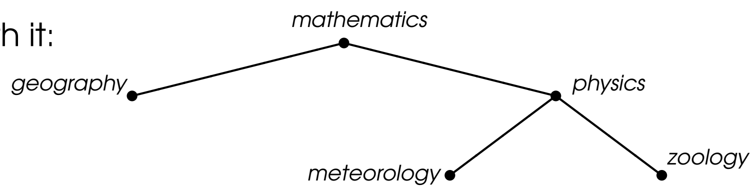
$$\textit{mathematics} \prec \textit{meteorology},$$

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

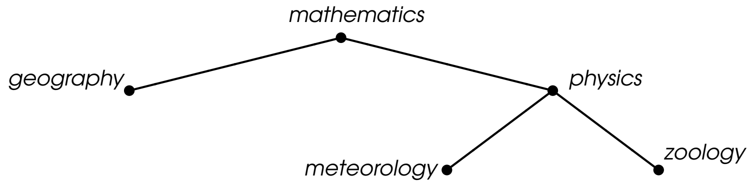
7TH STEP: We compare the new word *meteorology* with *physics*:

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

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



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



$$\boxed{\textit{mathematics},\ \textit{physics},\ \textit{geography},\ \textit{zoology},\ \textit{meteorology},\ \textit{geology},\ \textit{psychology}}$$

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

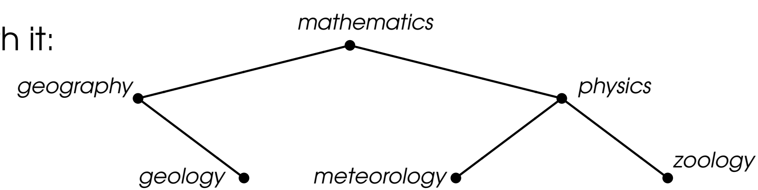
$$\textit{geology} \prec \textit{mathematics},$$

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

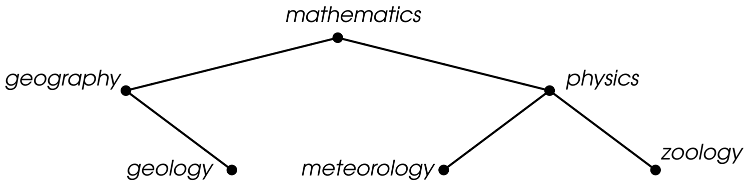
9TH STEP: We compare the new word *geology* with *geography*:

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

so we label a new right child with it:



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



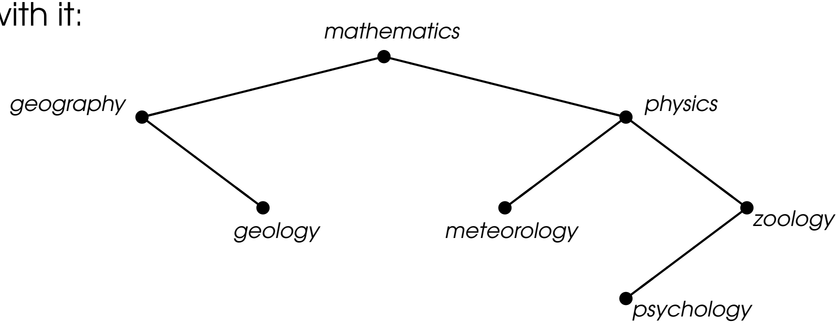
$\downarrow$

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

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

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

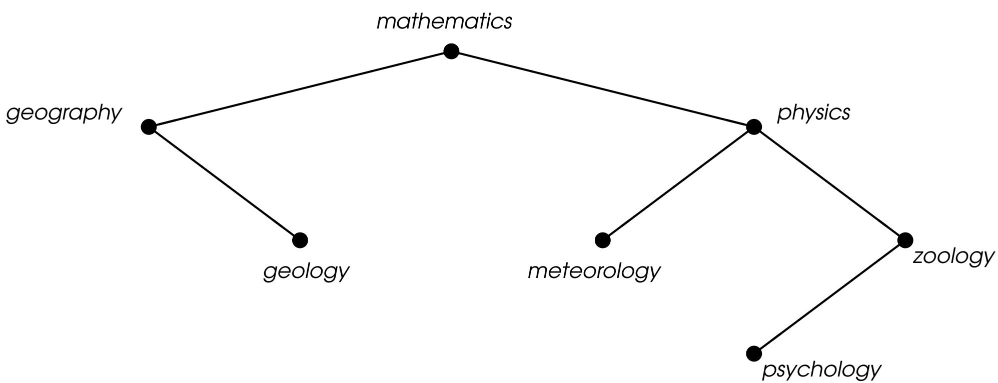
we label a <u>new left child</u> with it:



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.



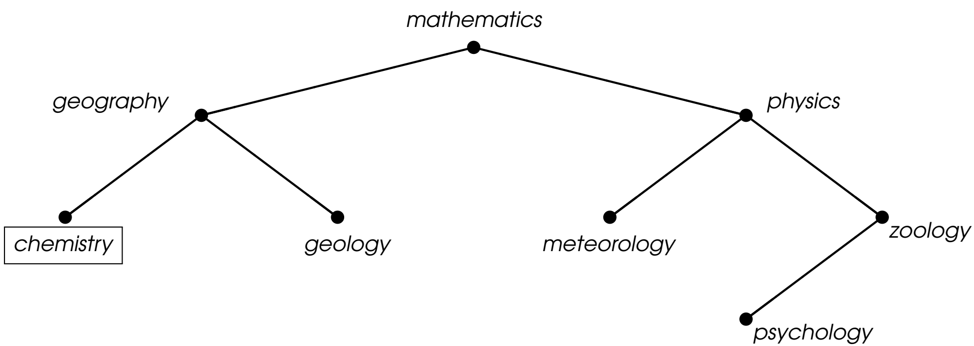
Binary search trees: locating or adding items II

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

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

and *geography* has no left child, we know at this point that *chemistry* is NOT in the tree. So we create a new left child and label it with *chemistry*:



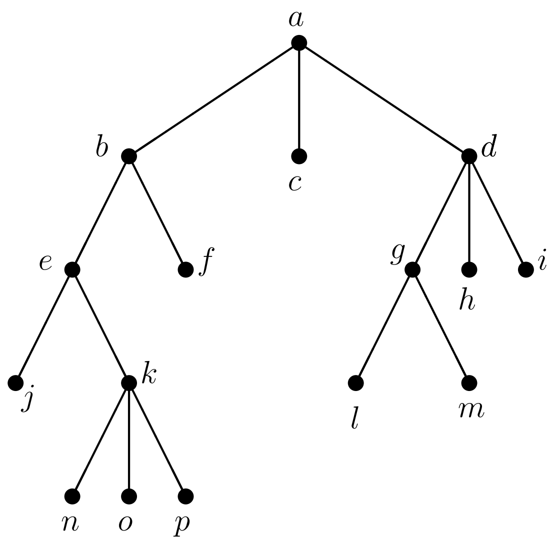
Tree traversal

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.

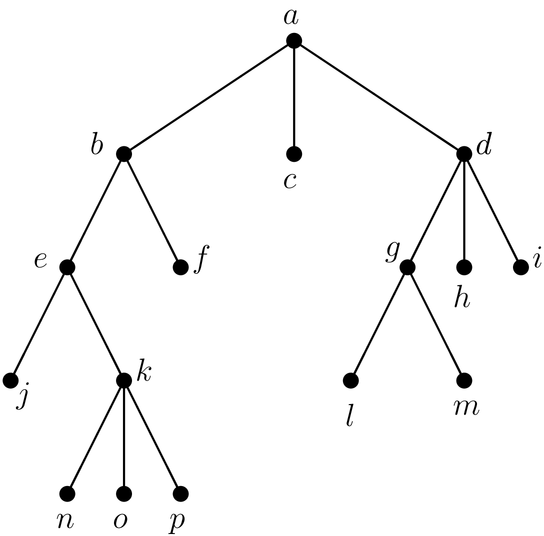
Preorder traversal



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

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

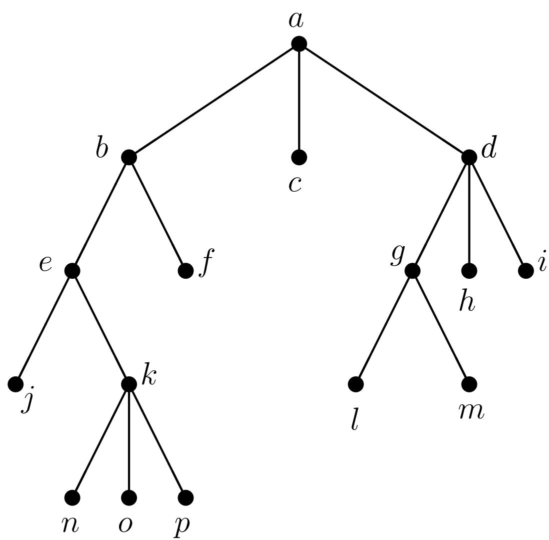
Inorder traversal



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:

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

