

A dense word cloud centered around computer science and programming concepts. The most prominent words are "ottimizzazione" (optimization), "algoritmi" (algorithms), "applicazioni" (applications), "dati" (data), and "strutture dati" (data structures). Other visible words include "problemi" (problems), "realizzazione" (implementation), "simulazione" (simulation), "java", "tecniche", "programmazione", "gestione", "corsi", "grado", "laboratorio", "solving", "risoluzione", "graf", "utilizzo", "divide", "accesso", and various terms related to software development, databases, and system design.

Trees

(in computer science)

Tree in Computer Science

- ▶ A tree is a widely used data structure that simulates a hierarchical tree structure with a set of linked nodes

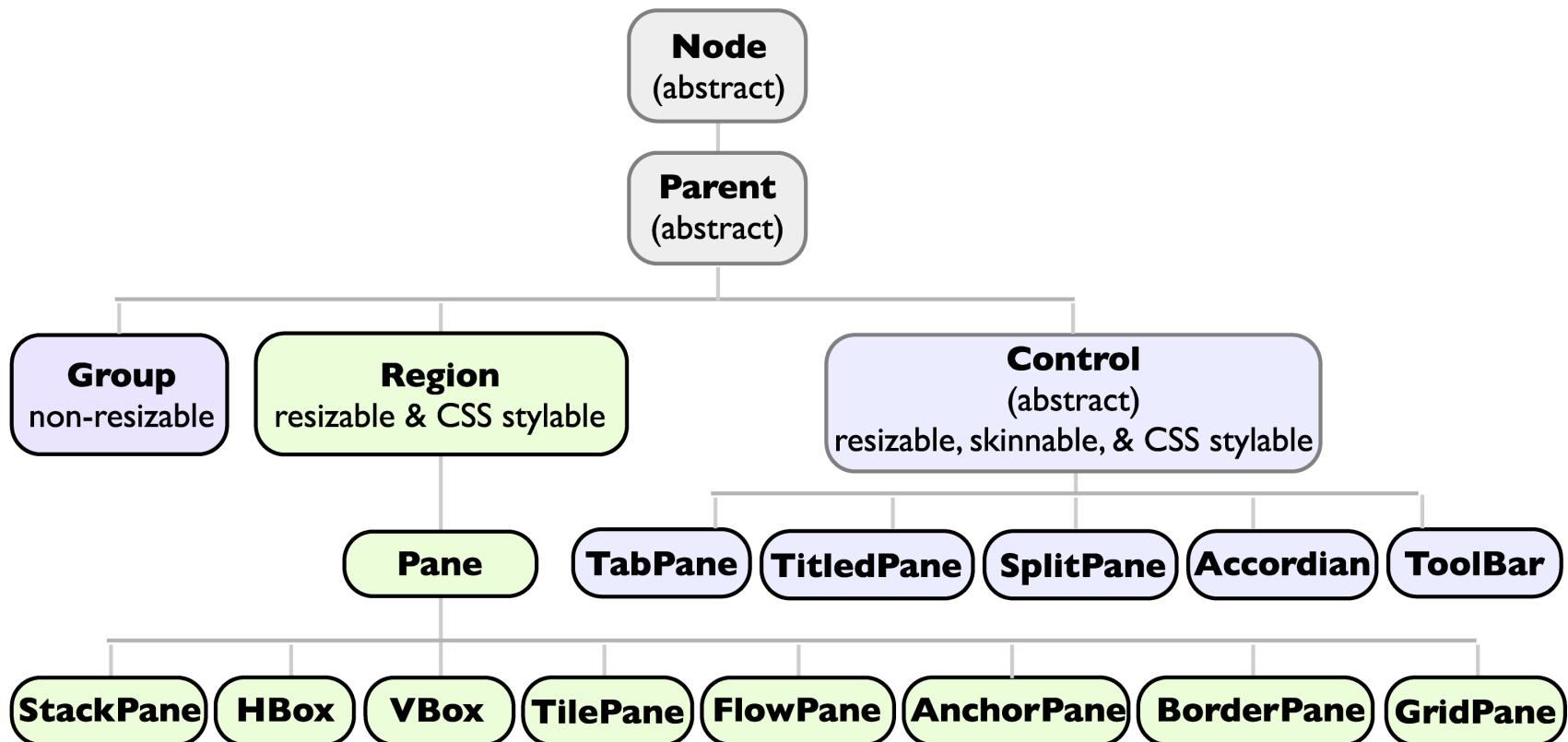


Tree in Computer Science

- ▶ **Fundamental** data storage structures used in programming
- ▶ Nonlinear structure
- ▶ Represents a *hierarchy*
- ▶ Items in a tree do not form a simple sequence
- ▶ Quite efficient for retrieving items (as arrays)
- ▶ Quite efficient for inserting/deleting items (as lists)



JavaFX 2.0 Layout Classes



Ordinamento dello Stato Italiano



Tree basics

- ▶ Consists of nodes connected by edges
- ▶ Nodes often represent entities (complex objects)
- ▶ Edges between the nodes represent the way the nodes are related
- ▶ The only way to get from node to node is to follow a path along the edges

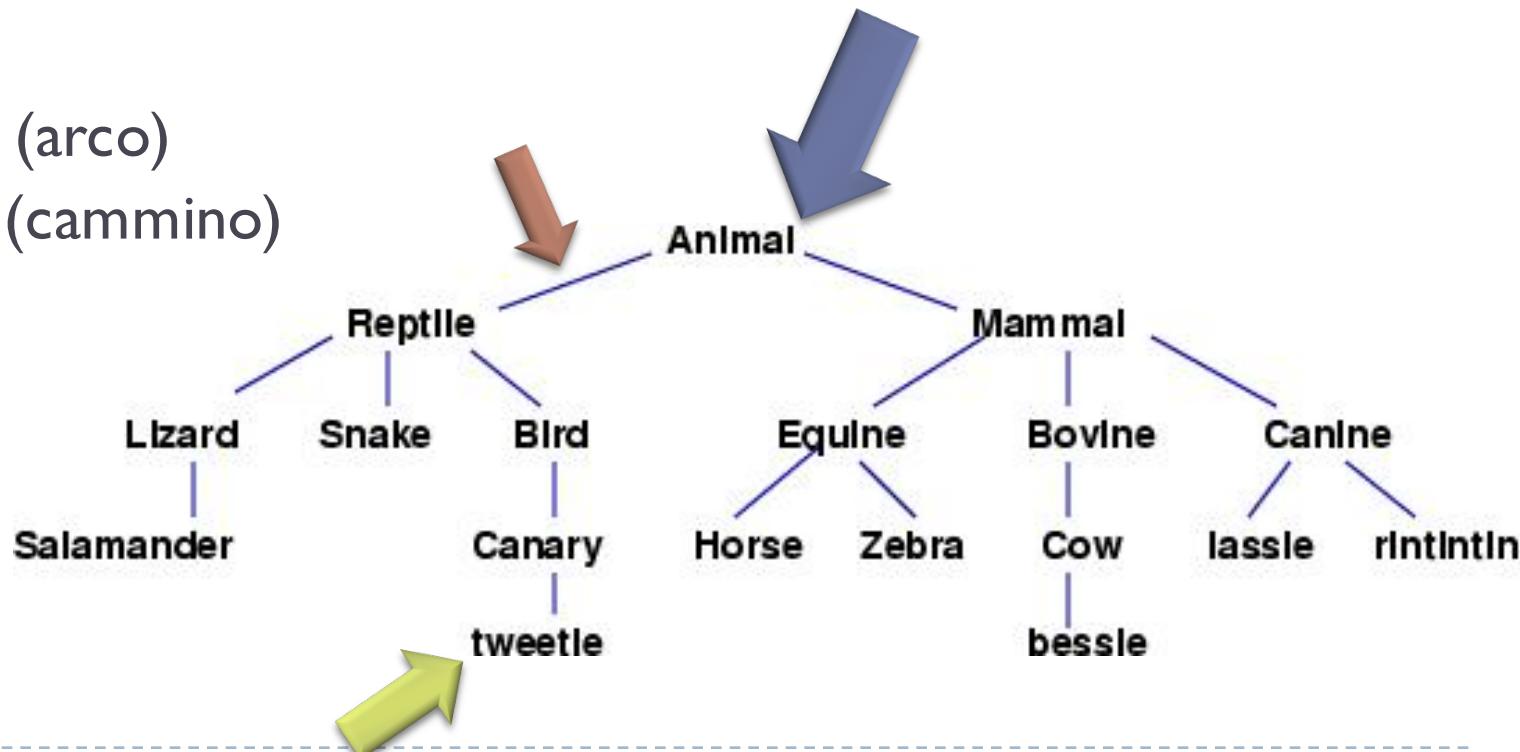
Tree Basics

▶ Node

- ▶ Root (radice)
- ▶ Leaf (foglia)
- ▶ Interior node/branch (nodo interno)

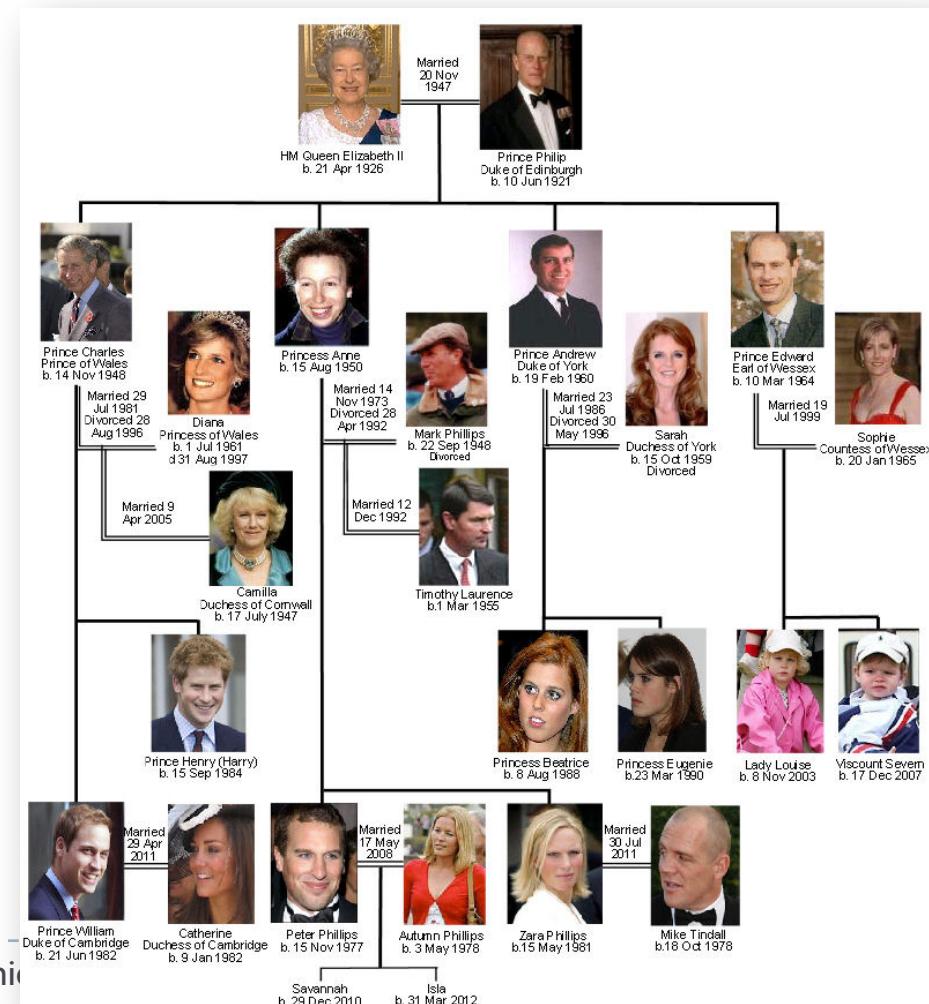
▶ Links

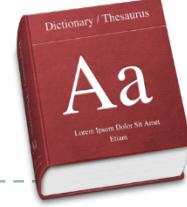
- ▶ Edge (arco)
- ▶ Path (cammino)



Tree Basics

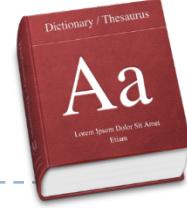
- ▶ Relationship
 - ▶ Parent (padre)
 - ▶ Child nodes (nodi figli)
 - ▶ Sibling (fratelli)
 - ▶ Descendant (discendente, successore)
 - ▶ Ancestor (antenato, predecessore)





Terminology

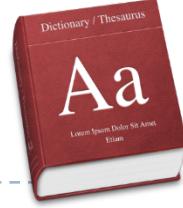
- ▶ **Visiting**
 - ▶ A node is visited when program control arrives at the node, usually for processing
- ▶ **Traversing**
 - ▶ To traverse a tree means to visit all the nodes in some specified order



Terminology

▶ Levels

- ▶ The level of a particular node refers to how many generations the node is from the root
- ▶ Root is assumed to be level 0

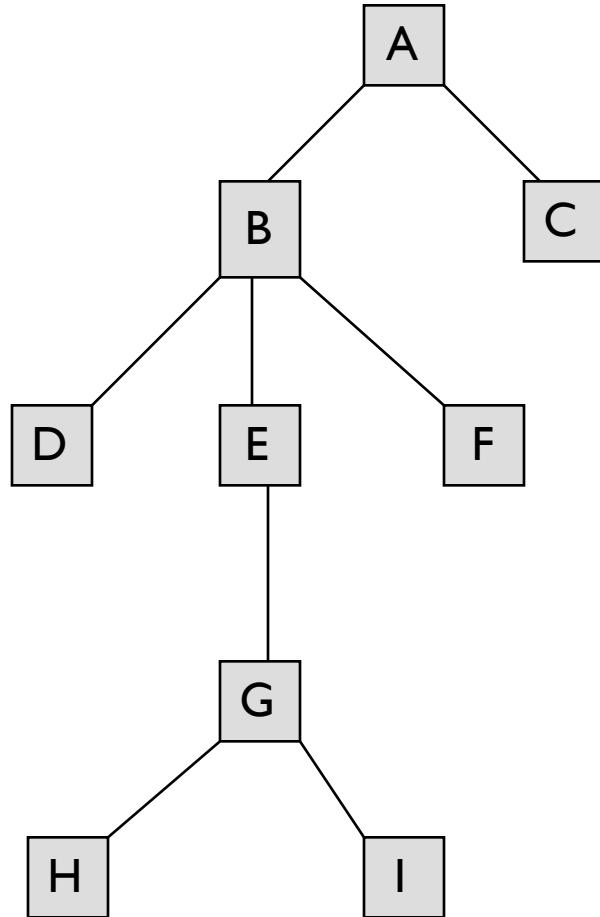


Terminology

▶ Height

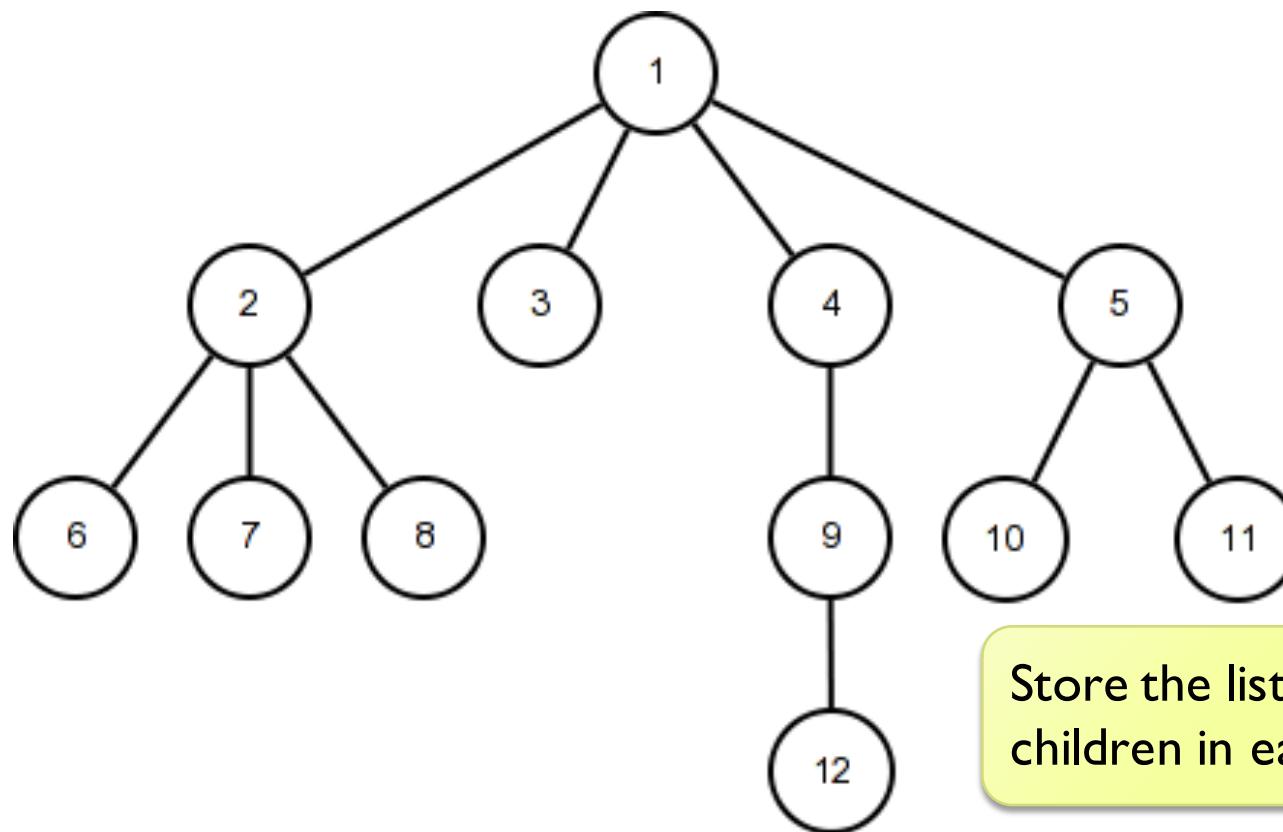
- ▶ The height of a node is the length of the path to its farthest descendant (i.e. farthest leaf node)
- ▶ The height of a tree is the height of the root
- ▶ A tree with only root node has height 0

Test!



- ▶ Number of nodes
- ▶ Height
- ▶ Root Node
- ▶ Leaves
- ▶ Levels
- ▶ Interior nodes
- ▶ Ancestors of H
- ▶ Descendants of B
- ▶ Siblings of E

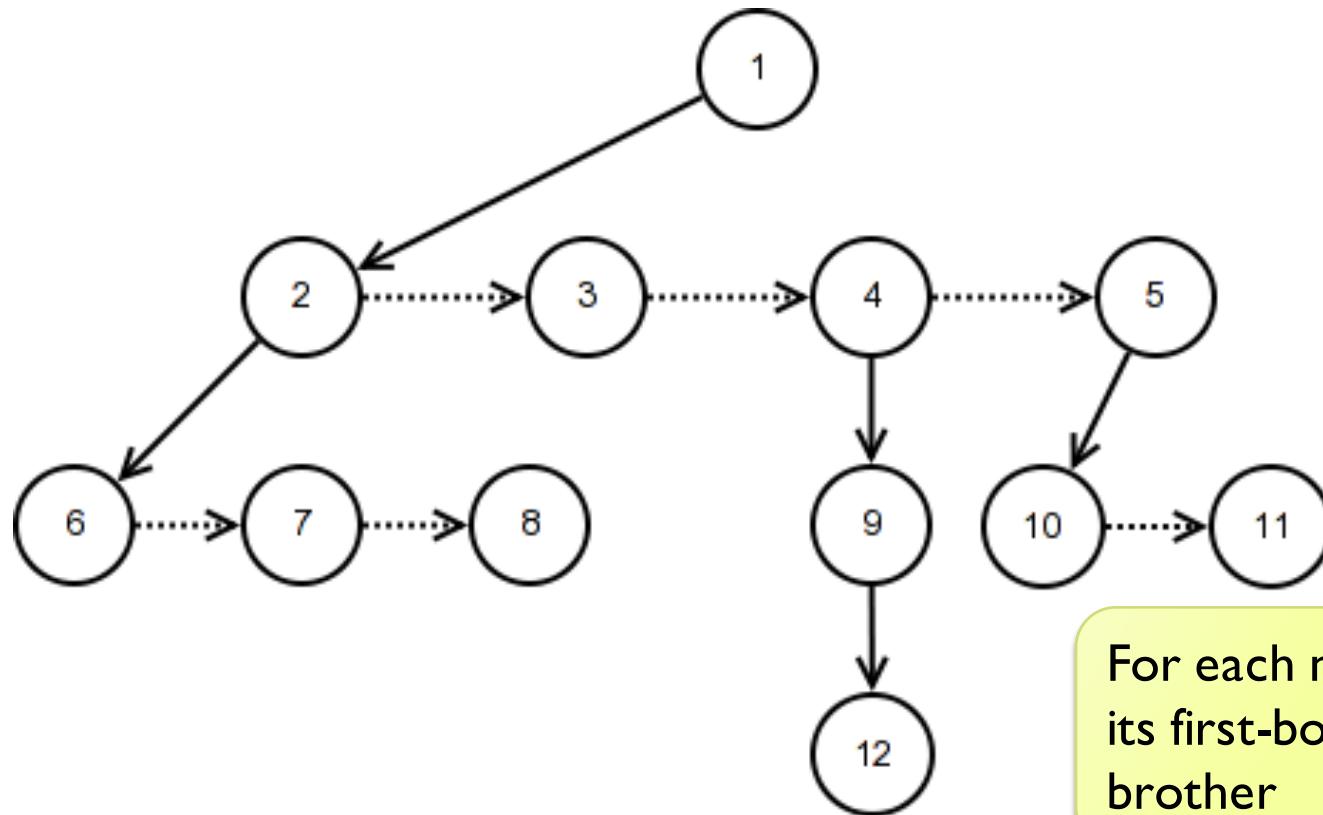
Tree representation



Store the list of
children in each node



Tree representation (alt)

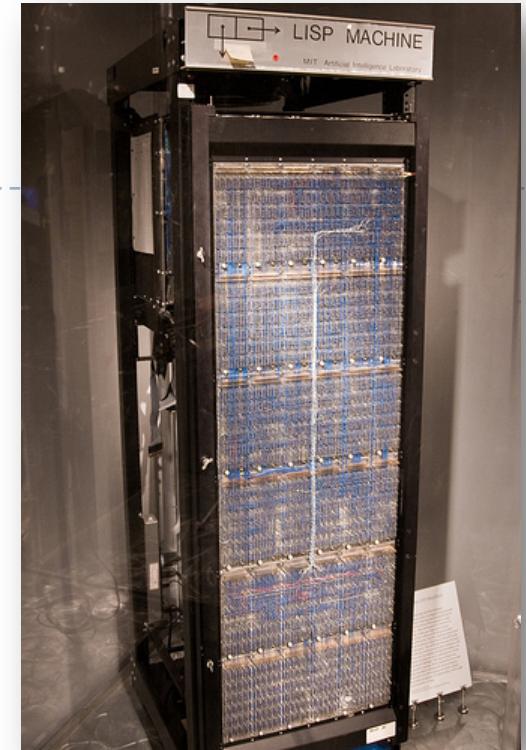
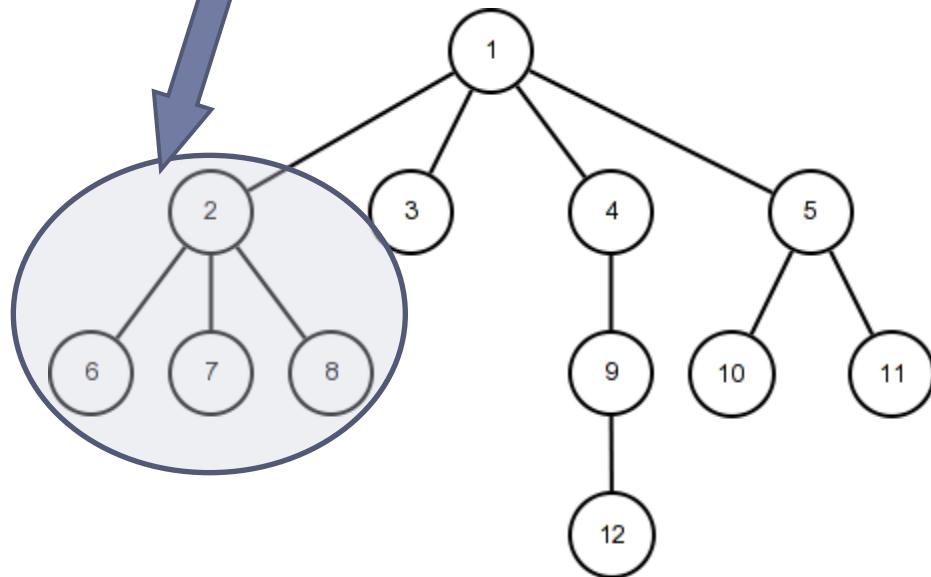


For each node store
its first-born and its
brother



Tree representation (alt)

- ▶ ()
- ▶ (1 2 3 4 5)
- ▶ ...
- ▶ (1 (2 (6 7 8) 3 4 (9 (12)) 5 (10 11)))



Store each sub-tree
as a separate object
in a list





An oversimplified tree

```
public class Tree<T> {  
    public Node<T> root;  
  
    public Tree() {  
        root = new Node<T>();  
    }  
  
    public Tree(T r) {  
        root = new Node<T>(r);  
    }  
}
```



An oversimplified tree

```
public class Node<T> {  
    T data;  
    Node<T> parent;  
    List<Node<T>> children;  
  
    public Node() {  
        data = null;  
        children = new ArrayList<Node<T>>();  
    }  
    public Node(T d) {  
        this();  
        data = d;  
    }  
    [...]
```

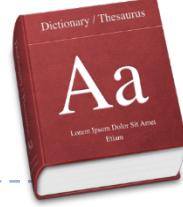


An oversimplified tree

[...]

```
public void addChild(Node<T> n) {  
    n.parent = this;  
    children.add(n);  
}  
  
public void removeChild(Node<T> n) {  
    children.remove(n);  
}
```

[...]



Terminology

▶ Visiting

- ▶ A node is visited when program control arrives at the node, usually for processing

▶ Traversing

- ▶ To traverse a tree means to visit all the nodes in some specified order





An oversimplified tree

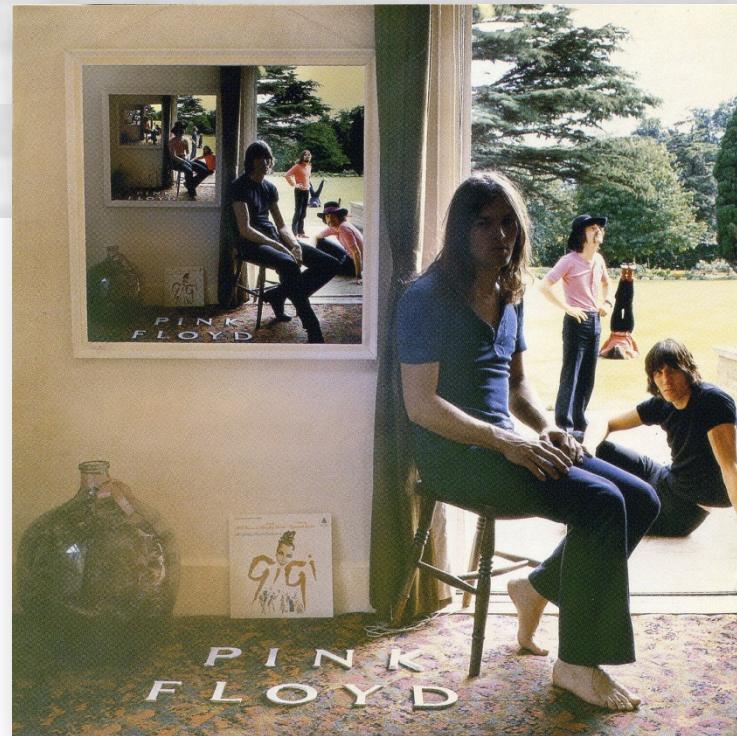
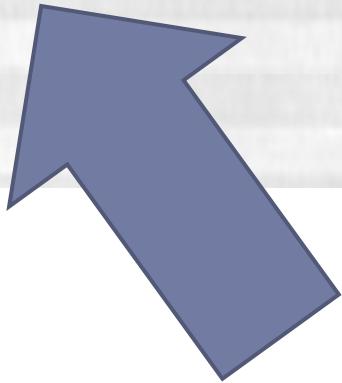
```
public class Tree<T> {  
    [...]  
  
    public void Visit() {  
        root.Visit();  
    }  
}
```





An oversimplified tree

```
public class Node<T> {  
    [...]  
    void Visit() {  
        // Do something on the node  
        for(Node<T> n : children) {  
            n.Visit();  
        }  
    }  
}
```





An oversimplified tree

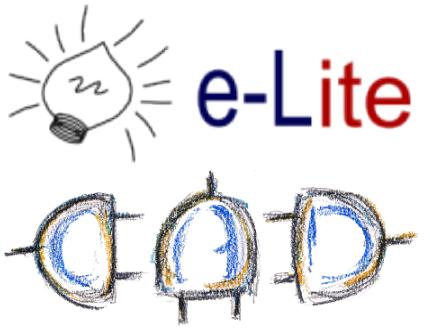
```
void Visit() {  
    if(children.size()>0)  
        System.out.print("(");  
    System.out.print(data);  
    for(Node<T> n : children) {  
        System.out.print(" ");  
        n.Visit();  
    }  
    if(children.size()>0)  
        System.out.print(")");  
}
```





An oversimplified tree

```
public class Node<T> {  
    [...]  
    void Visit() {  
        for(Node<T> n : children) {  
            n.Visit();  
        }  
        // Do something on the node  
    }  
}
```

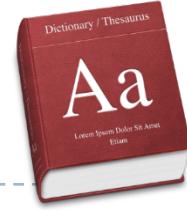


Binary Trees

Binary Tree

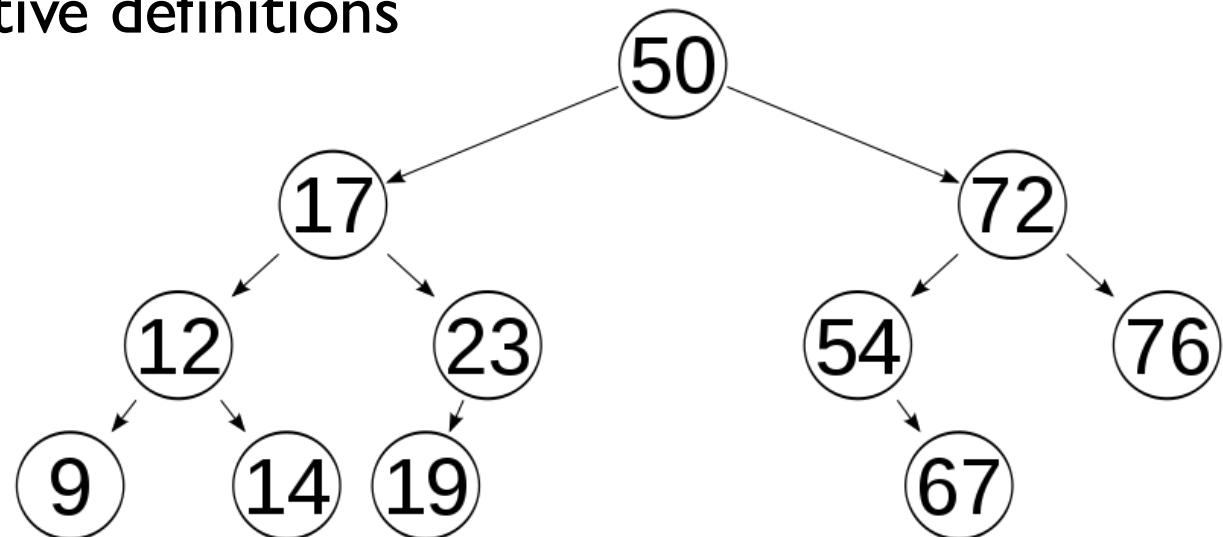
- ▶ A binary tree is a tree where each node has at most two children
- ▶ The two children are ordered (“left”, “right”)
 - ▶ Right sub-tree vs. Left sub-tree

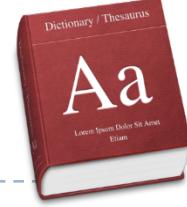




Balanced trees

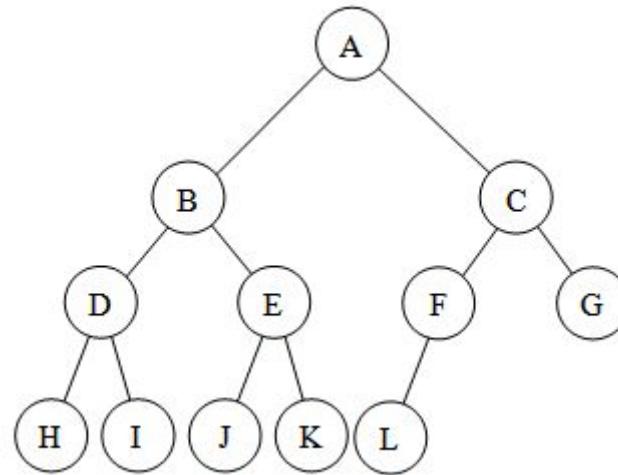
- ▶ (Height-)balanced trees
 - ▶ The left and right sub-trees' heights differ by at most one
 - ▶ The two sub-trees are (height-)balanced
- ▶ Perfectly balanced
 - ▶ $2^h - 1$ nodes
- ▶ Several alternative definitions



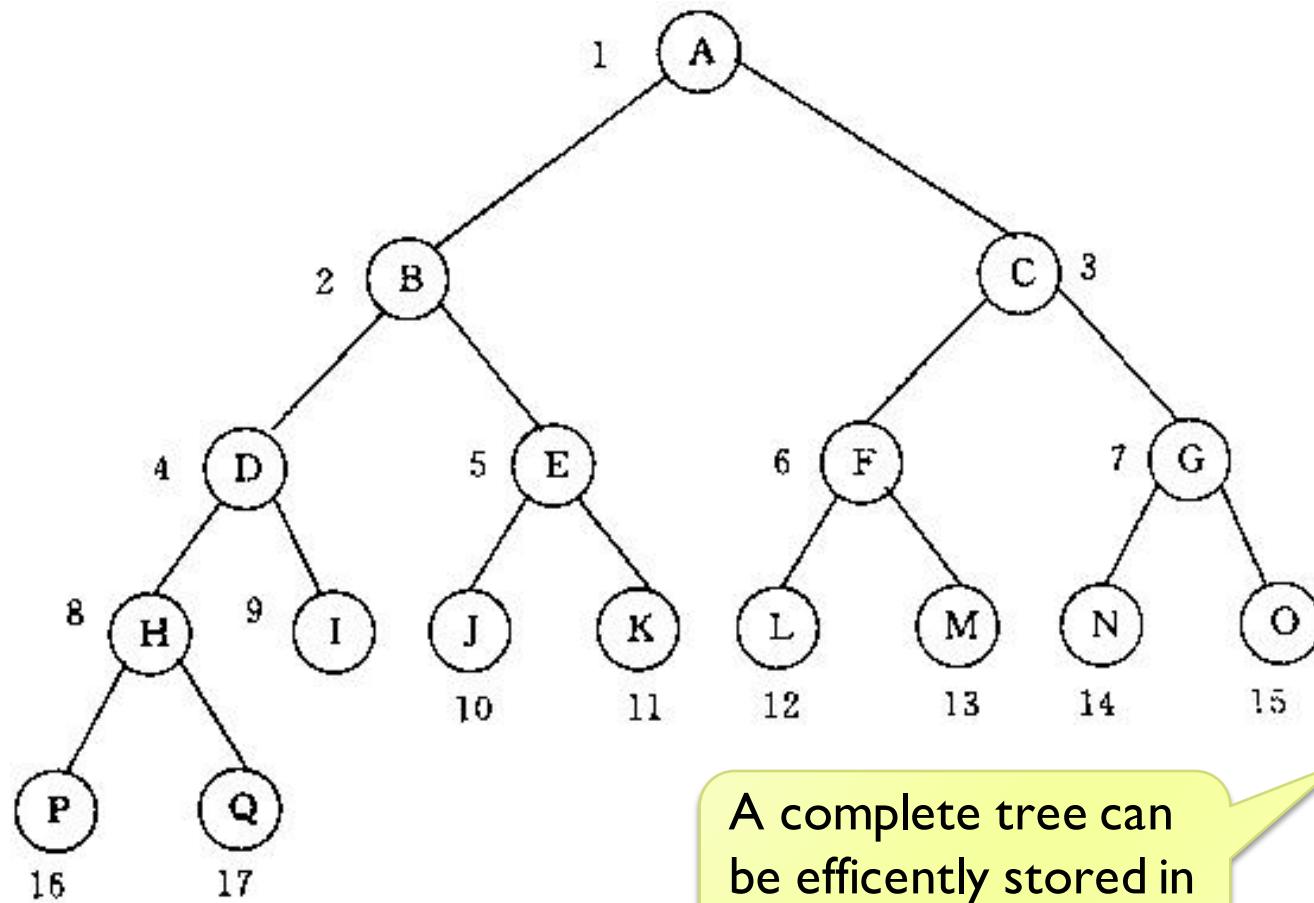


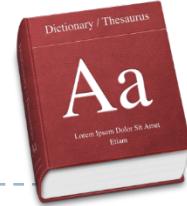
Complete trees

- ▶ Complete binary tree
 - ▶ Every level, except possibly the last, is completely filled, and all nodes are as far left as possible

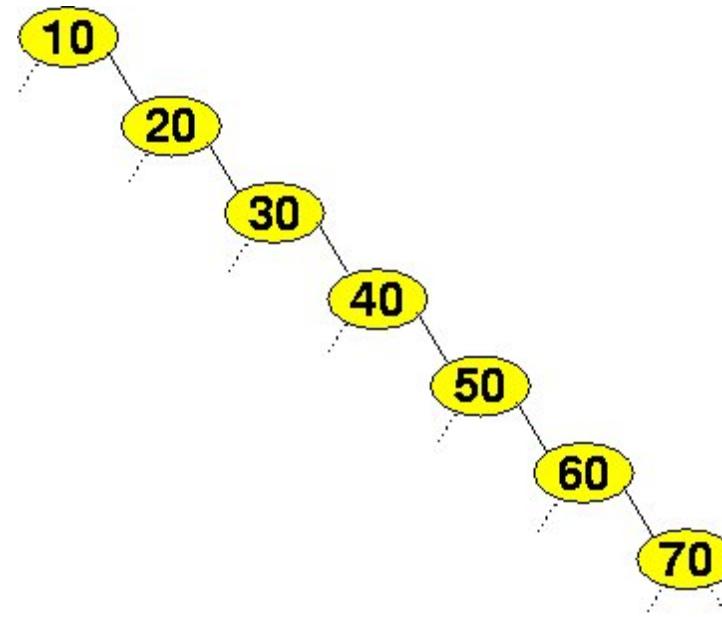


Tree representation



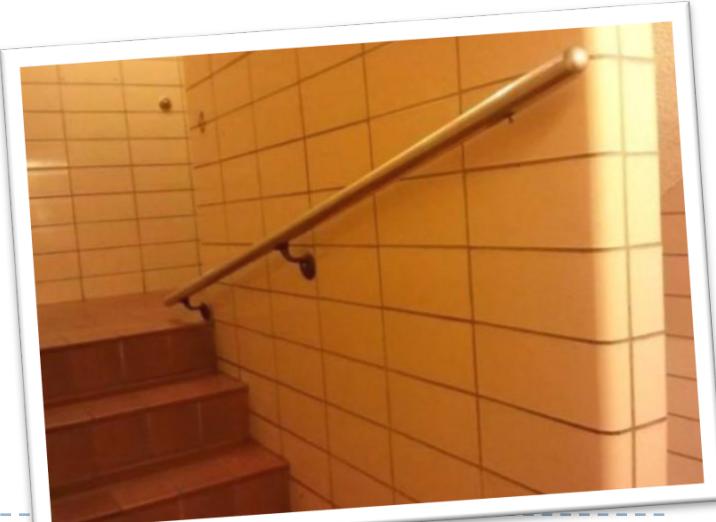


Degenerate trees

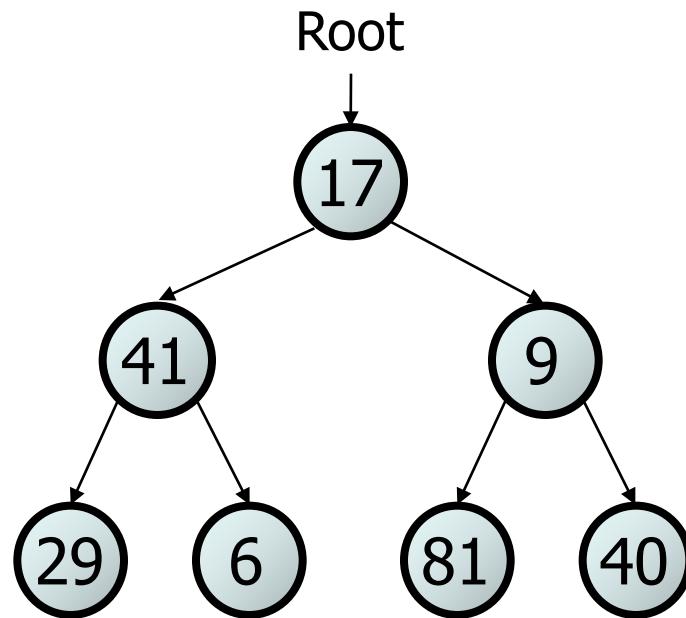


Traversal in binary trees

- ▶ **Pre-order**
 - ▶ process root node, then its left/right sub-trees
- ▶ **In-order**
 - ▶ process left sub-tree, then root node, then right
- ▶ **Post-order**
 - ▶ process left/right sub-trees, then root node



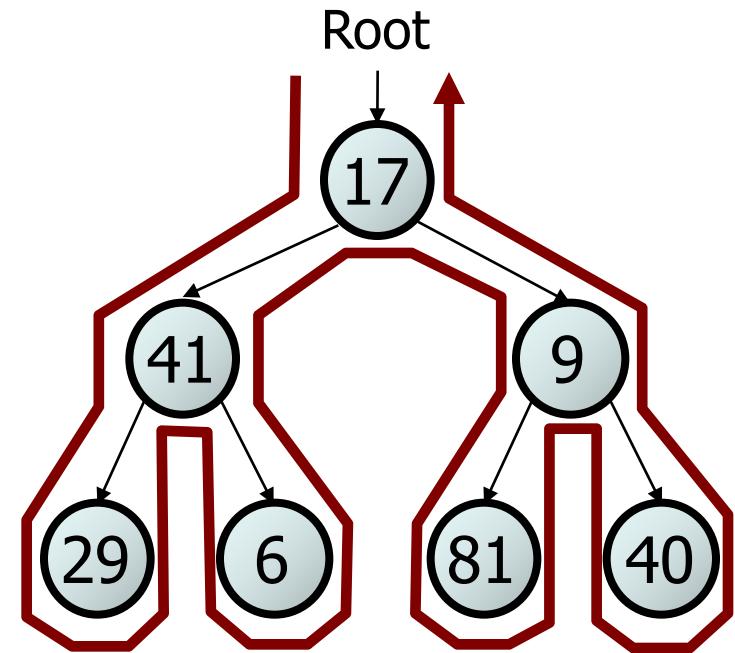
Traversal in binary trees



- ▶ **pre-order:**
- ▶ **in-order:**
- ▶ **post-order:**

Traversal trick

- ▶ To quickly generate a traversal:
 - ▶ Trace a path around the tree
 - ▶ As you pass a node on the proper **side**, process it
 - ▶ pre-order: left side
 - ▶ in-order: bottom
 - ▶ post-order: right side
- pre-order: 17 41 29 6 9 81 40
- in-order: 29 41 6 17 81 9 40
- post-order: 29 6 41 81 40 9 17



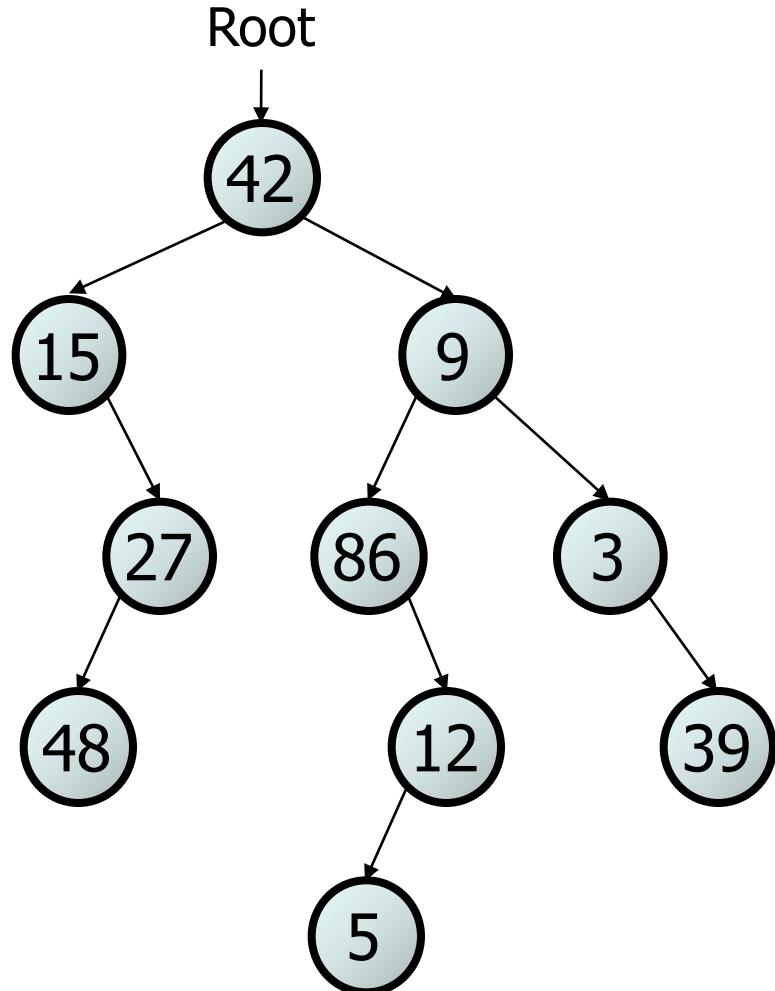
Exercise

- ▶ Give pre-, in-, and post-order traversals for the following tree:

pre:

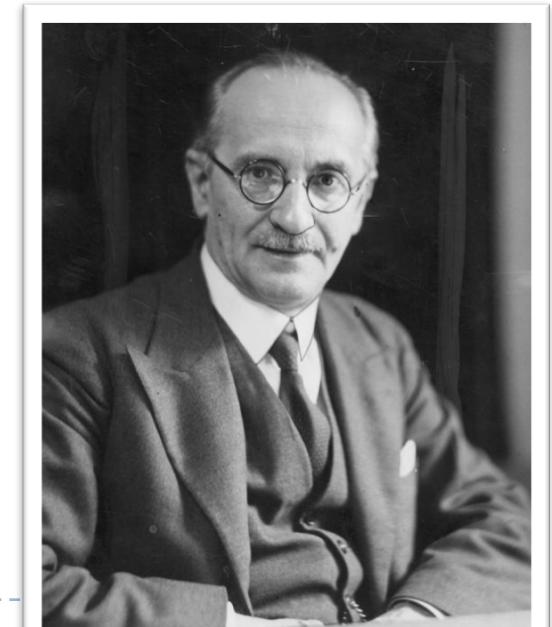
in:

post:



Polish prefix notation

- ▶ Akas: “Polish notation”, “prefix notation”
- ▶ Created in 1924 by the Polish logician Jan Łukasiewicz
- ▶ Operators are on the left of their operands
- ▶ If the arity of the operators is fixed
⇒ no need for parentheses or other brackets
- ▶ E.g.:
 - ▶ $3 * (2 + 7) \Rightarrow * 3 + 2 7$
 - ▶ $(x + y) / (2 - z) \Rightarrow / + x y - 2 z$



Reverse Polish notation

- ▶ (Re-)Invented by Bauer and Dijkstra in early 1960s to exploit stack for evaluating expressions
- ▶ Operator follows all of its operands
- ▶ If the arity of the operators is fixed
⇒ no need for parentheses or other brackets

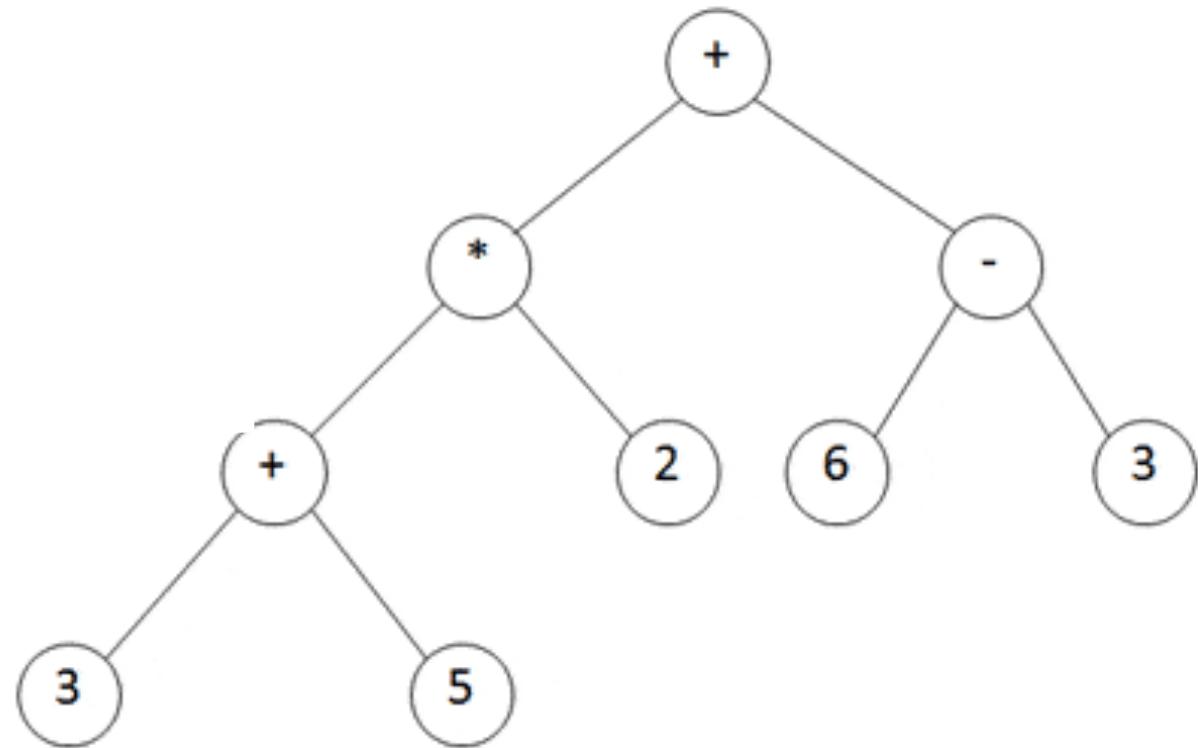


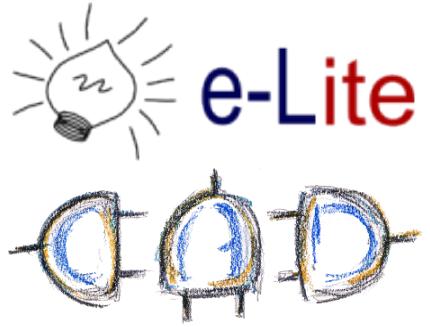
Traversals and notations

▶ In-order:

▶ Pre-order:

▶ Post-order:





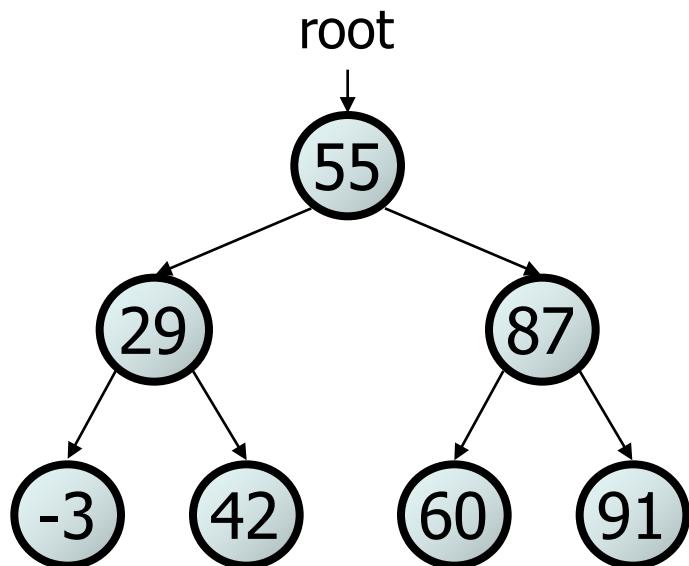
ottimizzazione applicazioni dati
realizzazione basi librerie tipo
soluzione risolvere comprensione capacità
listi grafiche svolgendo
risultato tipo oggetti sviluppo
incisive specifiche efficienza
dalle specifiche efficace
algoritmi problemi linguaggio
code software specifiche
dalle specifiche algoritmi
greedy interfacce
algoritmi problemi linguaggio
laboratorio
complementi ottimale
standard esercitazioni
java simulazione
programmazione filone
efficiente riconoscere min-max
mediante risoluzione
solving risoluzione
grafici programmi
abelli programmi
utilizzo divide e conquista
complese vista questo particolare
accesso

BST

Binary Search Tree

Binary search trees

- ▶ A binary tree where each non-empty node R has the following properties:
 - ▶ Elements of R's left sub-tree contain data “less than” R's data
 - ▶ Elements of R's right sub-tree contain data “greater than” R's data
 - ▶ R's left and right sub-trees are also binary search trees

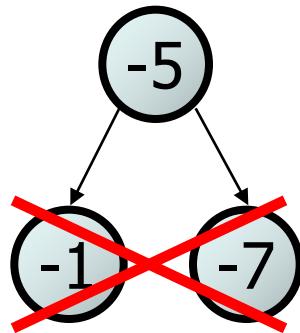


Binary search trees

- ▶ BSTs store their elements in sorted order, which is helpful for searching/sorting tasks

Exercise

- ▶ Is it a legal binary search tree?



Exercise

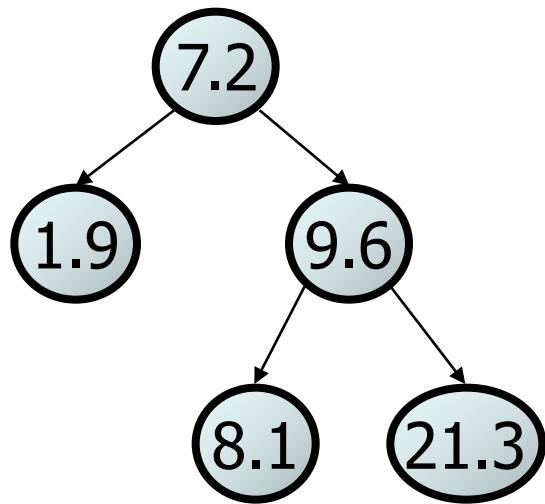
- ▶ Is it a legal binary search tree?



42

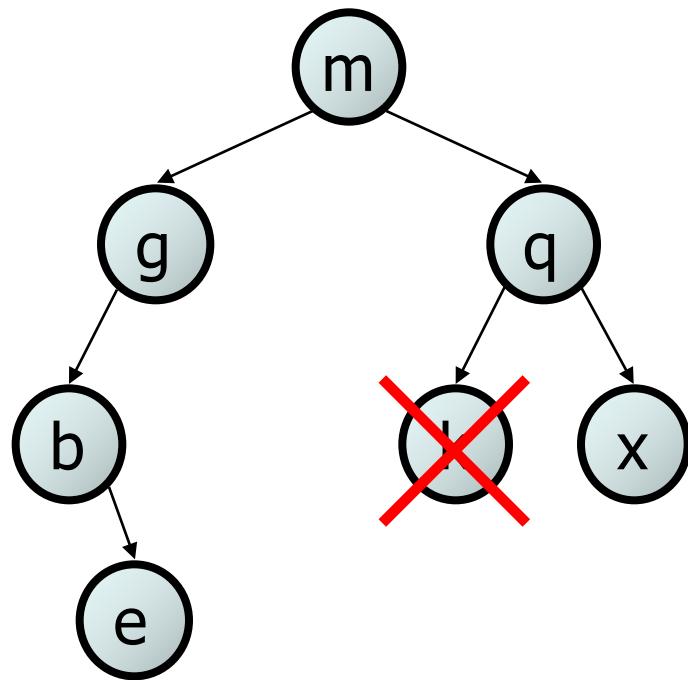
Exercise

- ▶ Is it a legal binary search tree?



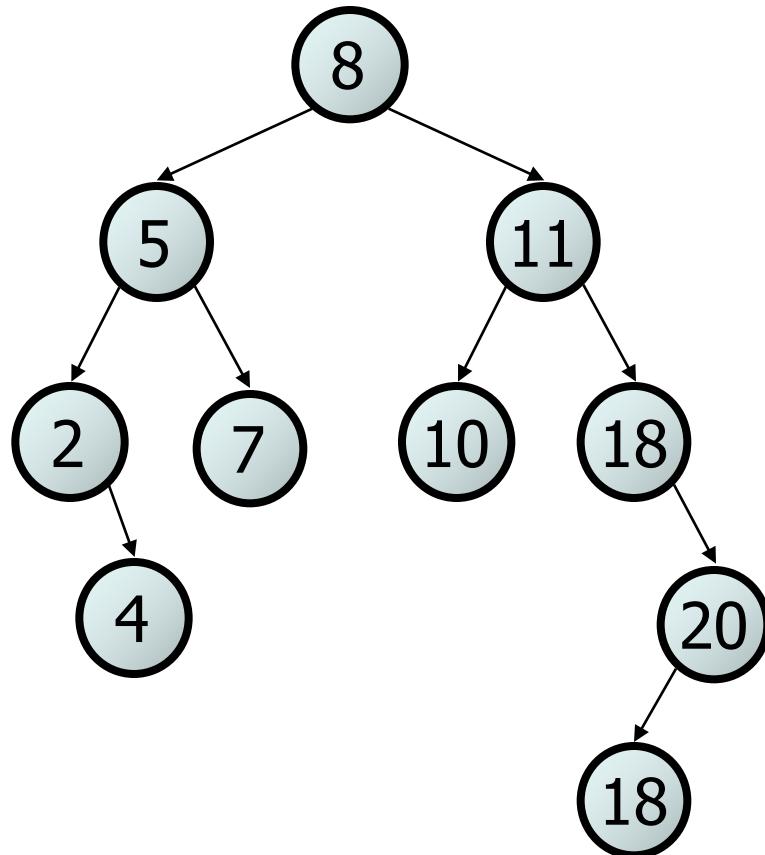
Excercise

- ▶ Is it a legal binary search tree?



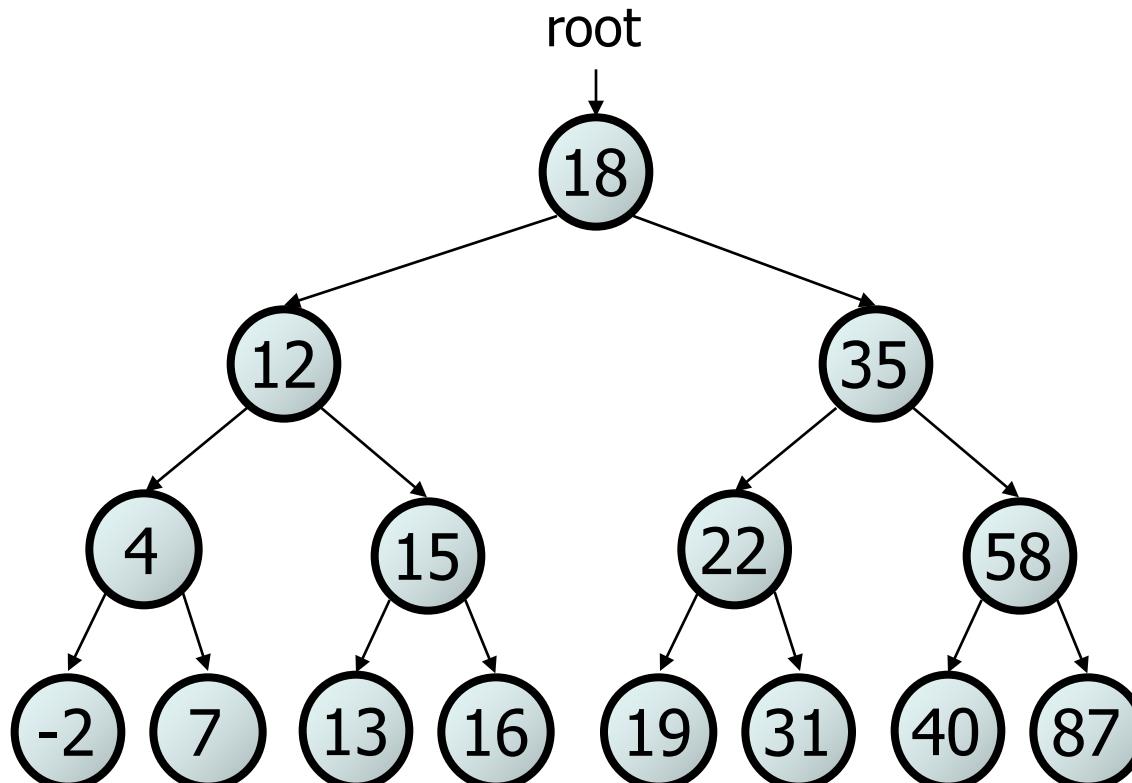
Exercise

- ▶ Is it a legal binary search tree?



Searching in a BST

- ▶ Describe an algorithm for searching a binary search tree
(try searching for 31, then 6)



Searching in a BST

- ▶ Searching in a BST is $O(h)$

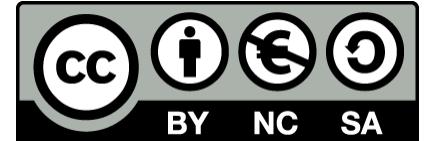
If the tree is balanced, then $h \cong \log_2 N$

⇒ Searching for an element is $O(\ln N)$



Showdown

	Array	List	Hash	BST
<code>add(element)</code>	$O(1)$	$O(1)$	$O(1)$	$O(\ln n)$
<code>remove(object)</code>	$O(n) + O(n)$	$O(n) + O(1)$	$O(1)$	$O(\ln n)$
<code>get(index)</code>	$O(1)$	$O(n)$	n.a.	n.a.
<code>set(index, element)</code>	$O(1)$	$O(n) + O(1)$	n.a.	n.a.
<code>add(index, element)</code>	$O(1) + O(n)$	$O(n) + O(1)$	n.a.	n.a.
<code>remove(index)</code>	$O(n)$	$O(n) + O(1)$	n.a.	n.a.
<code>contains(object)</code>	$O(n)$	$O(n)$	$O(1)$	$O(\ln n)$
<code>indexOf(object)</code>	$O(n)$	$O(n)$	n.a.	n.a.



Licenza d'uso

- ▶ Queste diapositive sono distribuite con licenza Creative Commons “Attribuzione - Non commerciale - Condividi allo stesso modo (CC BY-NC-SA)”
- ▶ Sei libero:
 - ▶ di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera
 - ▶ di modificare quest'opera
- ▶ Alle seguenti condizioni:
 - ▶ **Attribuzione** — Devi attribuire la paternità dell'opera agli autori originali e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera.
 - ▶ **Non commerciale** — Non puoi usare quest'opera per fini commerciali.
 - ▶ **Condividi allo stesso modo** — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.
- ▶ <http://creativecommons.org/licenses/by-nc-sa/3.0/>