

Tree in Computer Science

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



▶ 3

Tecniche di programmazione A.A. 2015/2016

Tree in Computer Science

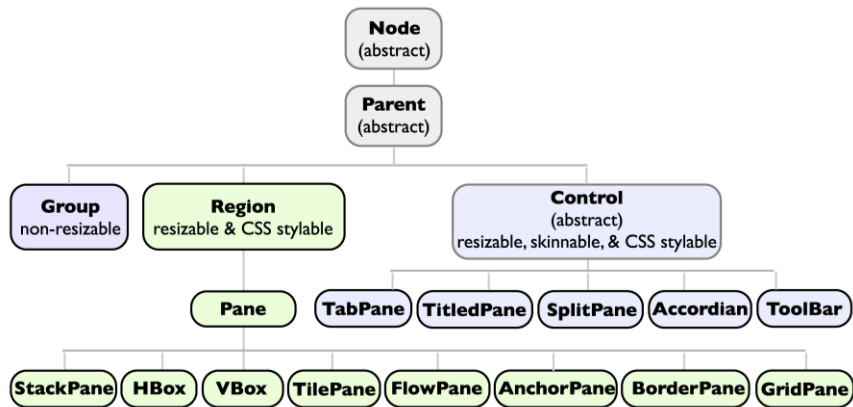
- ▶ **Fundamental** data storage structures used in programming
- ▶ Nonlinear structure
- ▶ Represents a *hierarchy*
- ▶ Items in a tree do not form a simple sequence
- ▶ Quite efficient for retrieving items (as arrays)
- ▶ Quite efficient for inserting/deleting items (as lists)



▶ 4

Tecniche di programmazione A.A. 2015/2016

JavaFX 2.0 Layout Classes



▶ 5

Tecniche di programmazione A.A. 2015/2016

Ordinamento dello Stato Italiano



▶ 6

Tecniche di programmazione A.A. 2015/2016

Tree basics

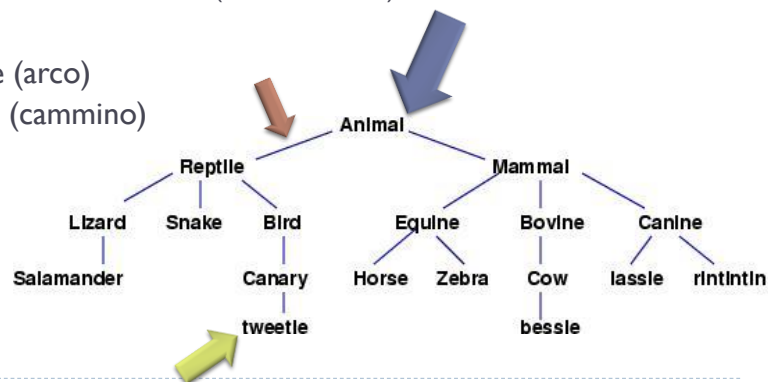
- ▶ Consists of nodes connected by edges
- ▶ Nodes often represent entities (complex objects)
- ▶ Edges between the nodes represent the way the nodes are related
- ▶ The only way to get from node to node is to follow a path along the edges

▶ 7

Tecniche di programmazione A.A. 2015/2016

Tree Basics

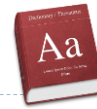
- ▶ **Node**
 - ▶ Root (radice)
 - ▶ Leaf (foglia)
 - ▶ Interior node/branch (nodo interno)
- ▶ **Links**
 - ▶ Edge (arco)
 - ▶ Path (cammino)



▶ 8

Tecniche di programmazione A.A. 2015/2016

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

▶ 11

Tecniche di programmazione A.A. 2015/2016

Terminology



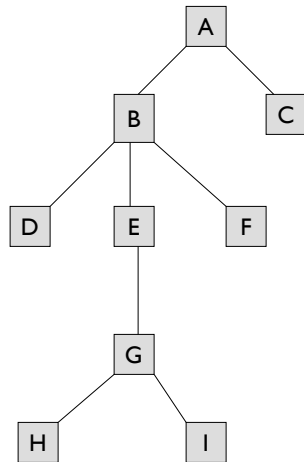
▶ Height

- ▶ 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

▶ 12

Tecniche di programmazione A.A. 2015/2016

Test!

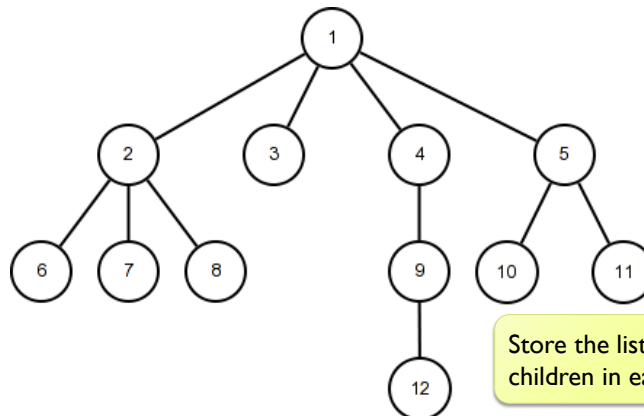


- ▶ Number of nodes
- ▶ Height
- ▶ Root Node
- ▶ Leaves
- ▶ Levels
- ▶ Interior nodes
- ▶ Ancestors of H
- ▶ Descendants of B
- ▶ Siblings of E

▶ 13

Tecniche di programmazione A.A. 2015/2016

Tree representation



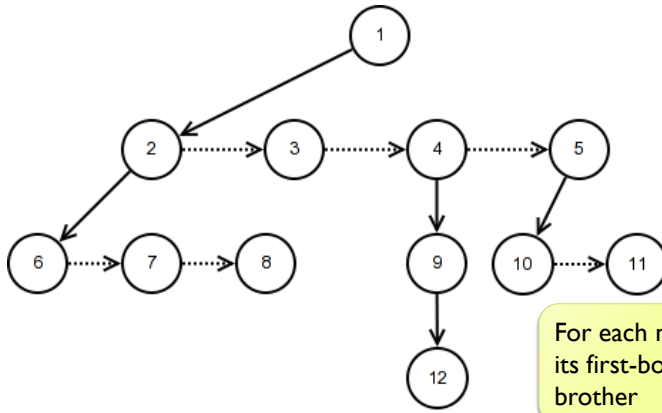
Store the list of children in each node



▶ 14

Tecniche di programmazione A.A. 2015/2016

Tree representation (alt)



For each node store its first-born and its brother

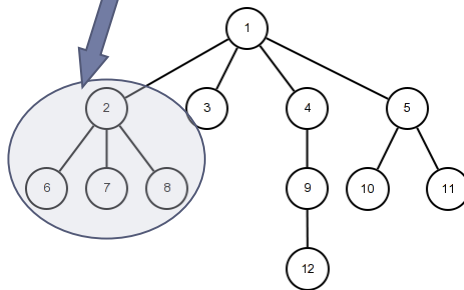


▶ 15

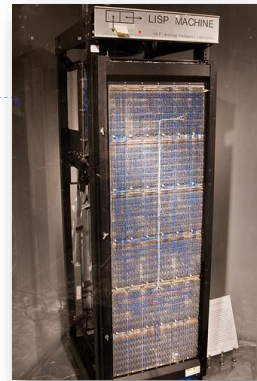
Tecniche di programmazione A.A. 2015/2016

Tree representation (alt)

- ▶ 1
- ▶ 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



▶ 16

Tecniche di programmazione A.A. 2015/2016

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);
    }
}
```

▶ 17

Tecniche di programmazione A.A. 2015/2016

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;
    }
}

[...]
```

▶ 18

Tecniche di programmazione A.A. 2015/2016

An oversimplified tree



[...]

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

public void removeChild(Node<T> n) {
    children.remove(n);
}
```

[...]

▶ 19

Tecniche di programmazione A.A. 2015/2016

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



▶ 20

Tecniche di programmazione A.A. 2015/2016

An oversimplified tree



```
public class Tree<T> {
  [...]

  public void Visit() {
    root.Visit();
  }
}
```



▶ 21

Tecniche di programmazione A.A. 2015/2016

An oversimplified tree



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



▶ 22

Tecniche di programmazione A.A. 2015/2016

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(")");
}
```



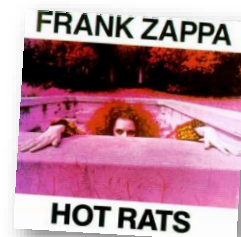
▶ 23

Tecniche di programmazione A.A. 2015/2016

An oversimplified tree

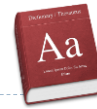


```
public class Node<T> {
    [...]
    void Visit() {
        for(Node<T> n : children) {
            n.Visit();
        }
        // Do something on the node
    }
}
```



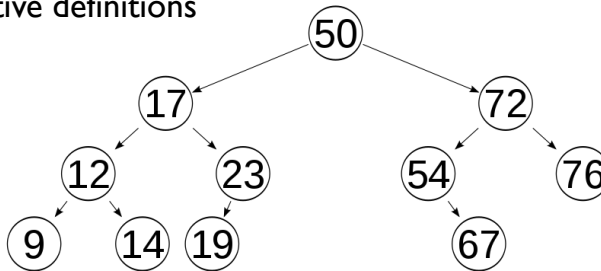
▶ 24

Tecniche di programmazione A.A. 2015/2016



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



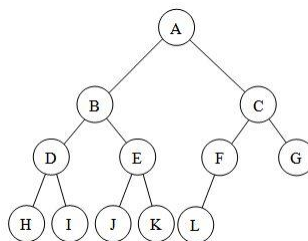
▶ 27

Tecniche di programmazione A.A. 2015/2016



Complete trees

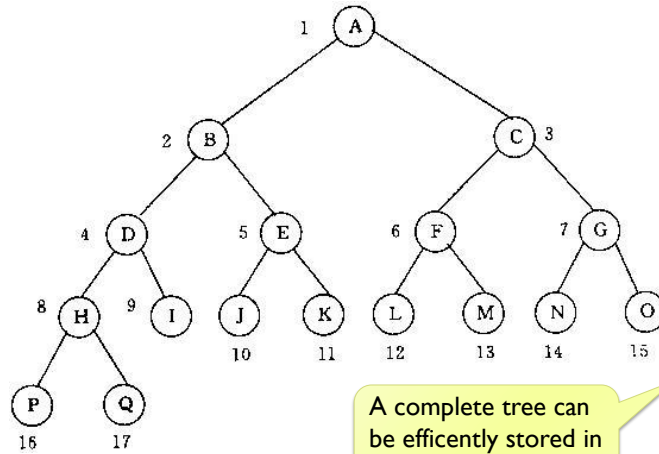
- ▶ Complete binary tree
 - ▶ Every level, except possibly the last, is completely filled, and all nodes are as far left as possible



▶ 28

Tecniche di programmazione A.A. 2015/2016

Tree representation



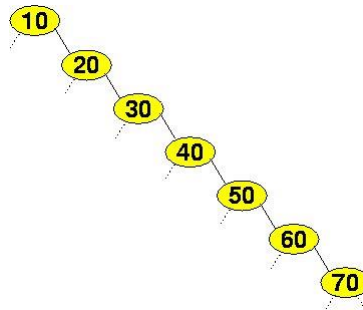
A complete tree can be efficiently stored in an array



▶ 29

Tecniche di programmazione A.A. 2015/2016

Degenerate trees



▶ 30

Tecniche di programmazione A.A. 2015/2016

Traversal in binary trees

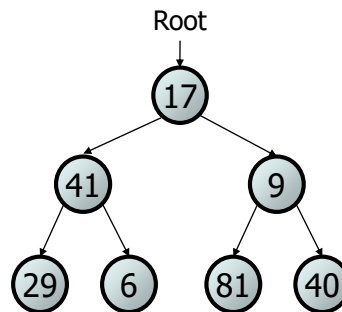
- ▶ **Pre-order**
 - ▶ process root node, then its left/right sub-trees
- ▶ **In-order**
 - ▶ process left sub-tree, then root node, then right
- ▶ **Post-order**
 - ▶ process left/right sub-trees, then root node



▶ 31

Tecniche di programmazione A.A. 2015/2016

Traversal in binary trees



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

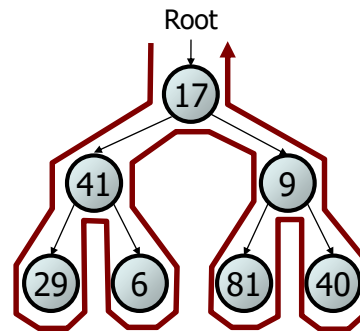
▶ 32

Tecniche di programmazione A.A. 2015/2016

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



Tecniche di programmazione A.A. 2015/2016

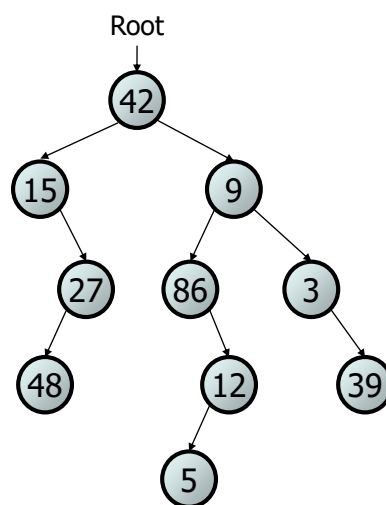
Exercise

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

pre:

in:

post:



▶ 34

Tecniche di programmazione A.A. 2015/2016

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.:

- ▶ $3 * (2 + 7) \Rightarrow * 3 + 2 7$
- ▶ $(x + y) / (2 - z) \Rightarrow / + x y - 2 z$



▶ 35

Tecniche di programmazione

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

- ▶ E.g.:

- ▶ $3 * (2 + 7) \Rightarrow 3 2 7 + *$
- ▶ $(x + y) / (2 - z) \Rightarrow x y + 2 z - /$

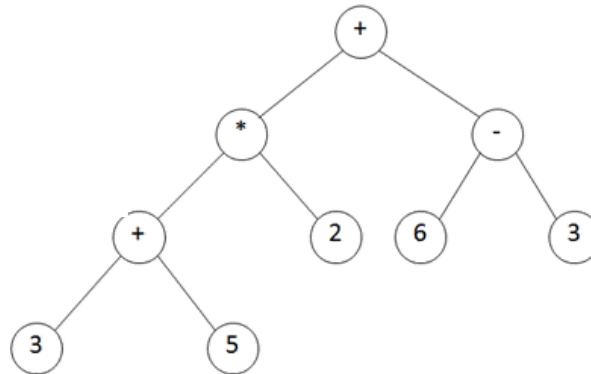


▶ 36

Tecniche di programmazione

Traversals and notations

- ▶ In-order:
- ▶ Pre-order:
- ▶ Post-order:



▶ 37

Tecniche di programmazione A.A. 2015/2016

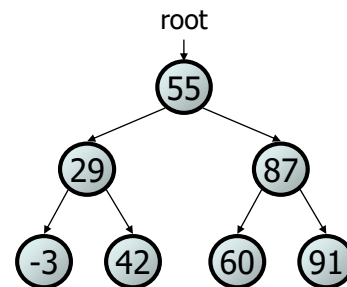


BST

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
 - ▶ R's left and right sub-trees are also binary search trees



▶ 39

Tecniche di programmazione A.A. 2015/2016

Binary search trees

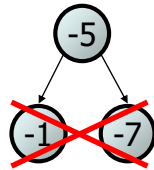
- ▶ BSTs store their elements in sorted order, which is helpful for searching/sorting tasks

▶ 40

Tecniche di programmazione A.A. 2015/2016

Exercise

- ▶ Is it a legal binary search tree?



▶ 41

Tecniche di programmazione A.A. 2015/2016

Exercise

- ▶ Is it a legal binary search tree?

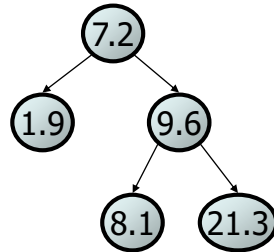
42

▶ 42

Tecniche di programmazione A.A. 2015/2016

Exercise

- ▶ Is it a legal binary search tree?

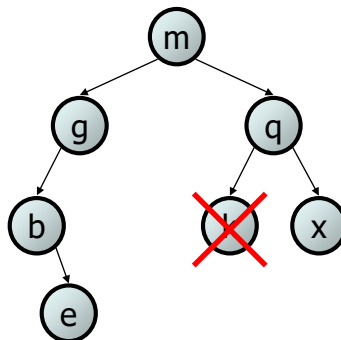


▶ 43

Tecniche di programmazione A.A. 2015/2016

Excercise

- ▶ Is it a legal binary search tree?

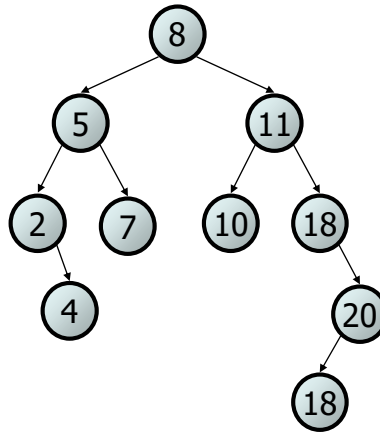


▶ 44

Tecniche di programmazione A.A. 2015/2016

Exercise

- ▶ Is it a legal binary search tree?

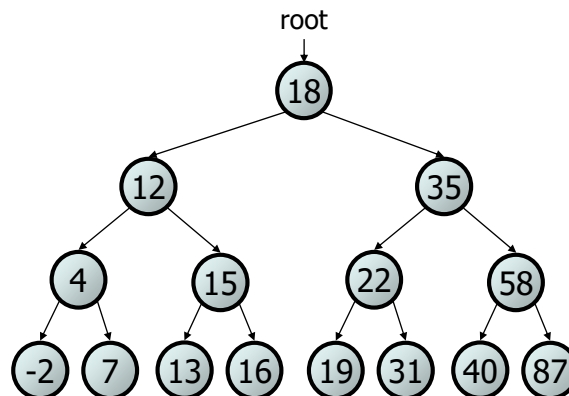


▶ 45

Tecniche di programmazione A.A. 2015/2016

Searching in a BST

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



▶ 46

Tecniche di programmazione A.A. 2015/2016

Searching in a BST

- ▶ Searching in a BST is $O(h)$
- ▶ If the tree is balanced, then $h \cong \log_2 N$
- ▶ \Rightarrow Searching for an element is $O(\ln N)$



▶ 47

Tecniche di programmazione A.A. 2015/2016

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.

▶ 48

Tecniche di programmazione A.A. 2015/2016

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/>