EECS 2011 M: Fundamentals of Data Structures

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/2011M`
Also on Moodle

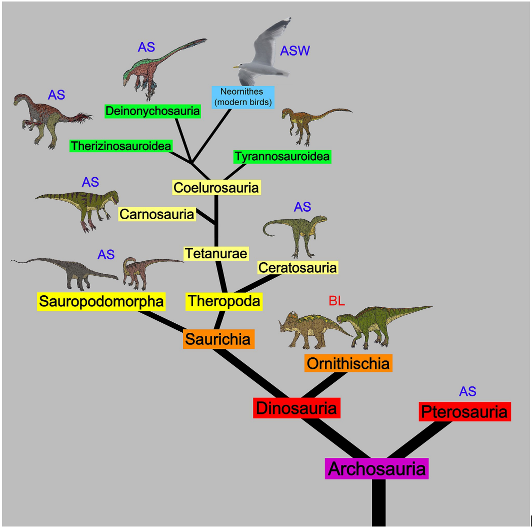
Trees

Ch. 8

- General and Binary Trees
- Tree Traversal
- Related topics:
  - Heaps Ch 9.3
  - Search Trees Ch 11.1
  - Height Balanced Search Trees Ch 11.2 - 11.6

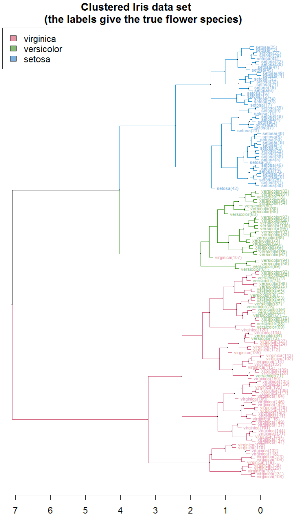
Note: Some slides in this presentation have been adapted from the authors' slides.

Trees



From https://upload.wikimedia.org/wikipedia/commons/f/f3/Phylogenetic_tree_of_Theropods_respiratory_system_01.JPG

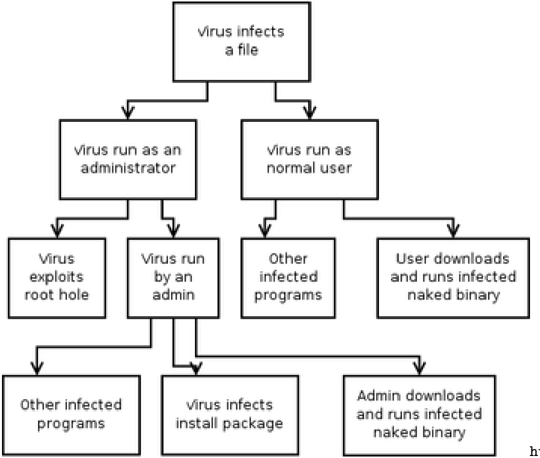
Trees - 2



By Talgalili - Own work, CC BY-SA 4.0,

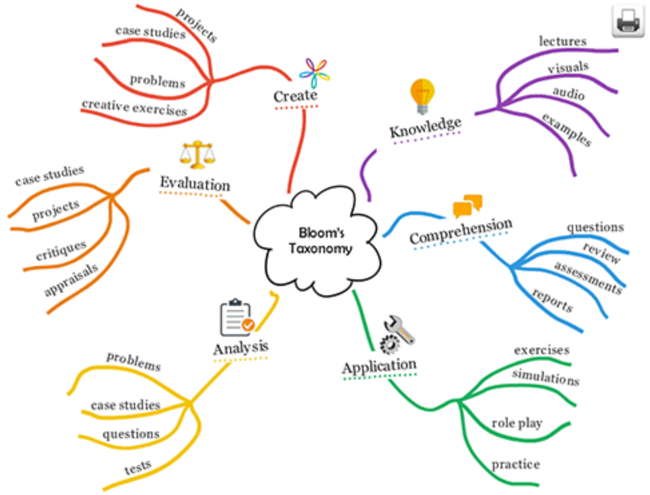
https://commons.wikimedia.org/w/index.php?curid=47743417

Trees - 3



https://upload.wikimedia.org/wikipedia/commons/thumb/c/c6/Attack_tree_virus.png/350px-Attack_tree_virus.png

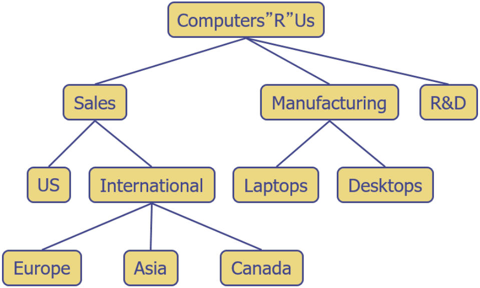
Trees - 4



https://articulate-heroes.s3.amazonaws.com/uploads/rte/jfdncskl_Liliana-Cotoara-Blooms-Taxonomy.png

Trees - 5

- An abstract model of a hierarchical structure
- consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments
- May be rooted or unrooted



Rooted Trees - Terminology

- Root: node without parent
- Internal node: node with at least one child
- External node (a.k.a. leaf): node without children
- Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- Depth of a node: no. of ancestors (depth(root) $= 0$)
- Height of a tree: maximum depth of any node
- Descendant of a node: child, grandchild, grand-grandchild, etc.
- Subtree: tree consisting of a node and its descendants

Position - ADT

- Models the notion of place within a data structure where a single object is stored
- It gives a unified view of diverse ways of storing data, such as
  - a cell of an array
  - a node of a linked list
  - a node of a tree

  Just one method:

  object p.element(): returns the element stored at the position p.

Trees - ADT

Uses positions to abstract nodes

- Generic methods:
  - integer size()
  - boolean isEmpty()
  - Iterator iterator()
  - Iterable positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - Iterable children(p)
  - integer numChildren(p)
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Update method: set(p, e)
  - replaces the element at position p with element e
  - returns the previously stored element.

Binary Trees

- Each internal node has at most two children (exactly two for proper binary trees)
- The children of a node are an ordered pair: left child and right child
- Alternative recursive definition: a binary tree
  - consists of a single node, or
  - has a root with an ordered pair of children, each of which is a binary tree

Applications:

- arithmetic expressions
- decision processes
- searching

Properties of Proper Binary Trees

Each node has 0 or 2 children.

- n: number of nodes
- e: number of external nodes (leaves)
- i: number of internal nodes
- h: height

- $e = i + 1$
- $n = 2e - 1$
- $h \leq i$
- $h \leq (n-1)/2$
- $e \leq 2^h$
- $h \geq \log_2 e$
- $h \geq \log_2(n+1) - 1$

Binary Tree ADT

- Extends the Tree ADT,
- Additional methods:
  - Position left(p)
  - Position right(p)
  - boolean hasLeft(p)
  - boolean hasRight(p)
- Update methods may be defined by data structures implementing the BinaryTree ADT

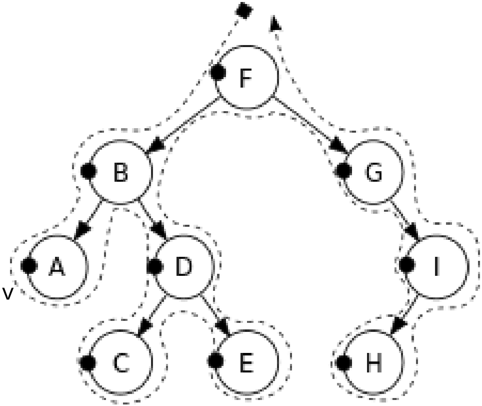
Tree Traversals

- Different ways of exploring and enumerating the nodes
- Each traversal is useful in some applications

Pre-order Traversal

- a node is visited before its descendants

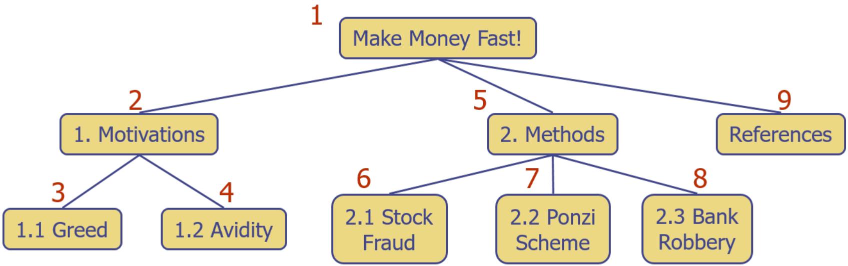
```
Algorithm preOrder(v)
  if (v != null)
      visit(v)
      for each child w of v
          preOrder (w)
```



From https://commons.wikimedia.org/w/index.php?curid=10616003

Pre-order Traversal - Application

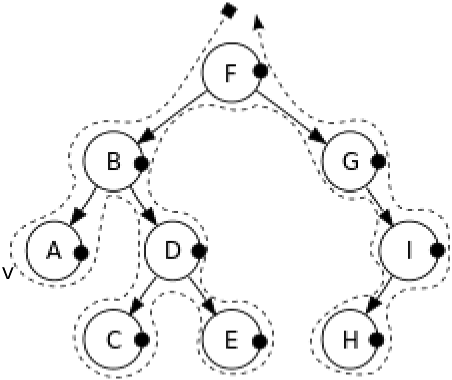
Print a structured document



Post-order Traversal

- a node is visited after its descendants

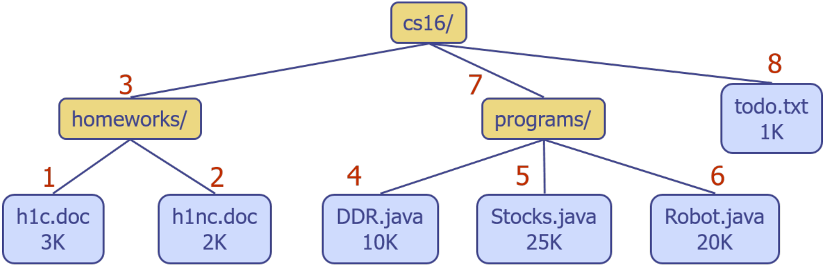
```
Algorithm postOrder(v)
  if (v != null)
      for each child w of v
          postOrder (w)
      visit(v)
```



https://commons.wikimedia.org/w/index.php?curid=10616033

Post-order Traversal - Application

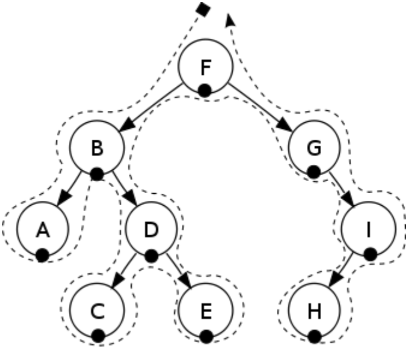
Compute space used by files in a directory and its subdirectories



In-order Traversal (Binary trees only)

- a node is visited after its left subtree and before its right subtree

```
Algorithm inOrder(v)
  if (v != null)
      inOrder (left (v))
      visit(v)
      inOrder (right (v))
```



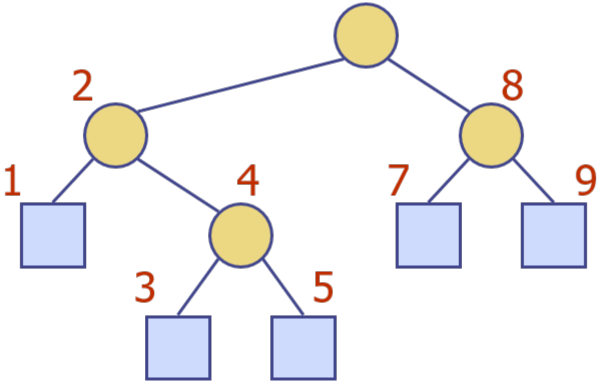
From https://commons.wikimedia.org/w/index.php?curid=10616018

In-order Traversal - Application

Draw a binary tree:

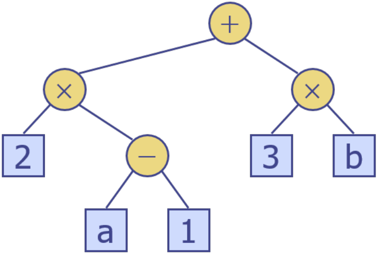
$x(v)$ = in-order rank of v

$y(v)$ = depth of v



Arithmetic Expression Tree

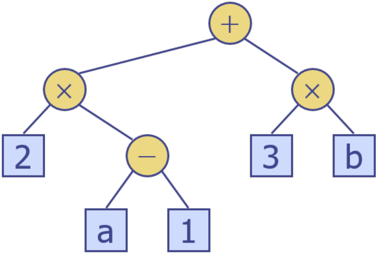
- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



Printing an Arithmetic Expression Tree

Specialization of an in-order traversal

```
Algorithm printExpression(v)
   if left (v) != null
      print("(")
      inOrder (left(v))
   print(v.element ())
   if right(v) != null
      inOrder (right(v))
  print(")")
```



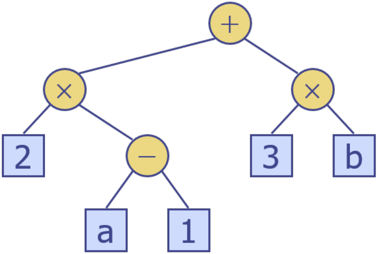
tree for the expression
$((2 \times (a - 1)) + (3 \times b))$

Evaluating an Arithmetic Expression Tree

Specialization of a post-order traversal

- recursively evaluate subtrees, by combining the values of the subtrees

```
Algorithm evalExpr(v)
  if isExternal (v)
      return v.element ()
  else
      x = evalExpr(left(v))
      y = evalExpr(right(v))
      op = operator at v
  return x op y
```

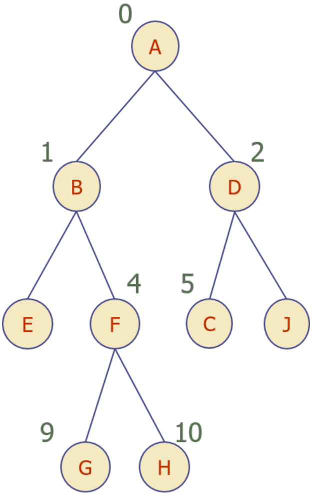


tree for the expression $((2 \times (a - 1)) + (3 \times b))$

Array-based Tree Implementation

Nodes are stored in an array A, e.g., v is stored at $A[rank(v)]$

- $\text{rank}(\text{root}) = 0$
- rank of left child of node i is $2i + 1$
- rank of right child of node i is $2i + 2$



Tree Implementation

Array-based

- Lower memory requirements: Parent and children are implicitly represented
- Memory requirements determined by tree height - very inefficient for sparse trees

Linked structure

- Requires explicit representation of 3 links per position: parent, left child, right child
- Data structure grows as needed – no wasted space.