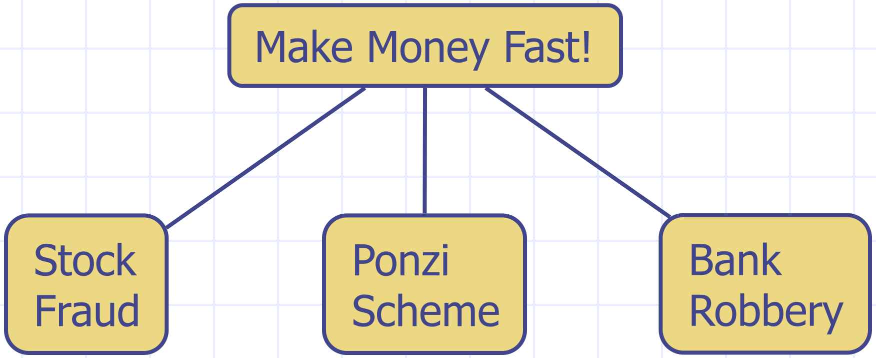
Trees



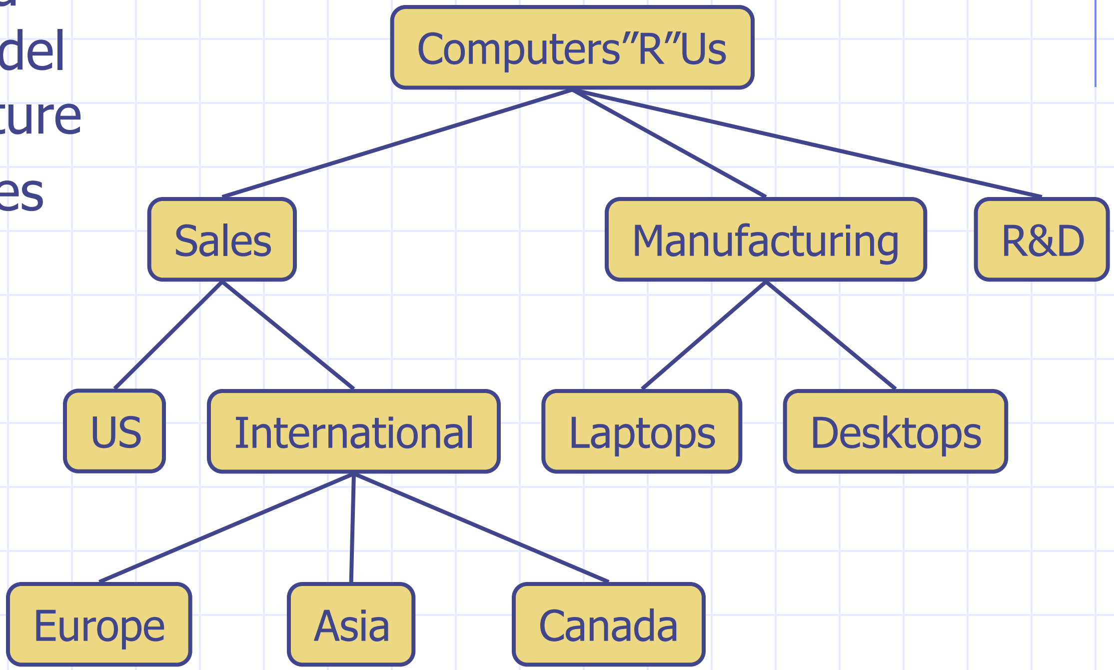
Trees

- Abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation

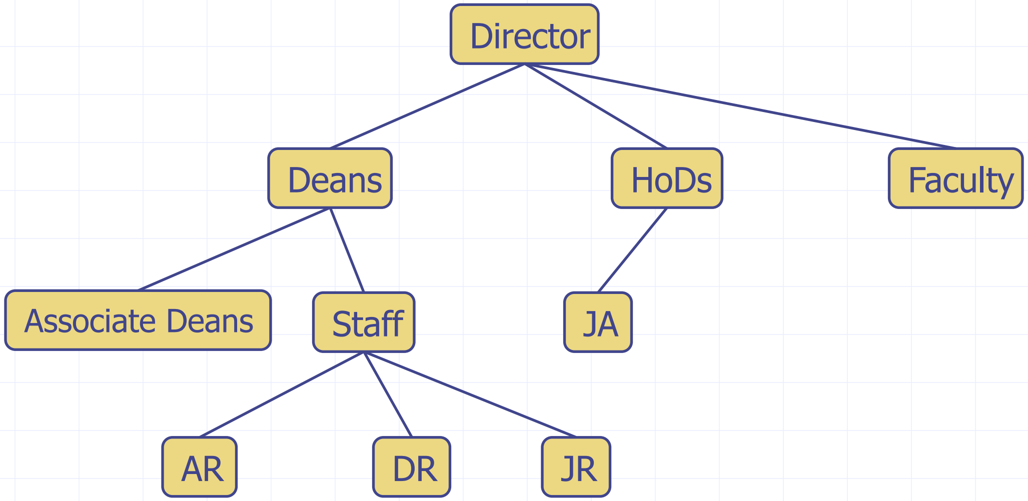
google images

What is a Tree

- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments

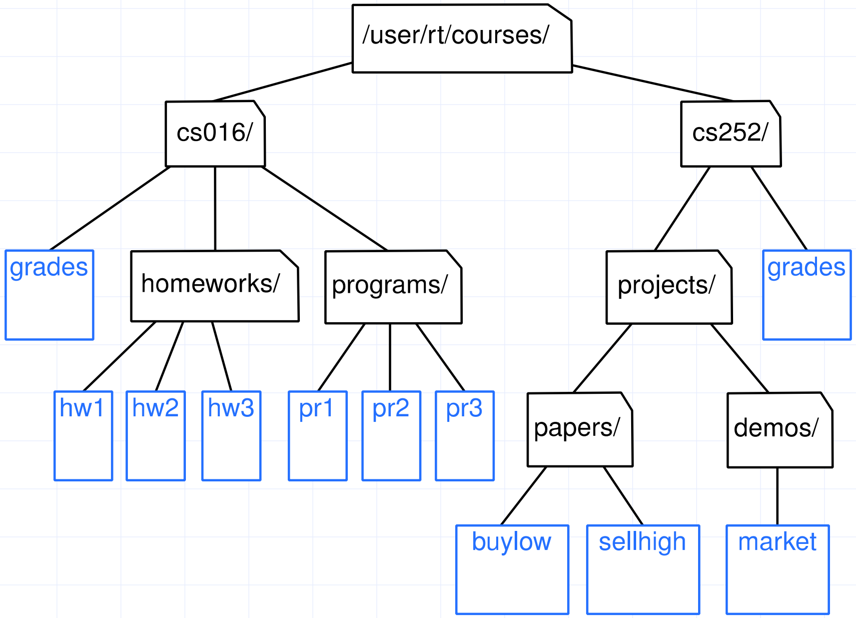


Trees - Examples



organization structure of a corporation

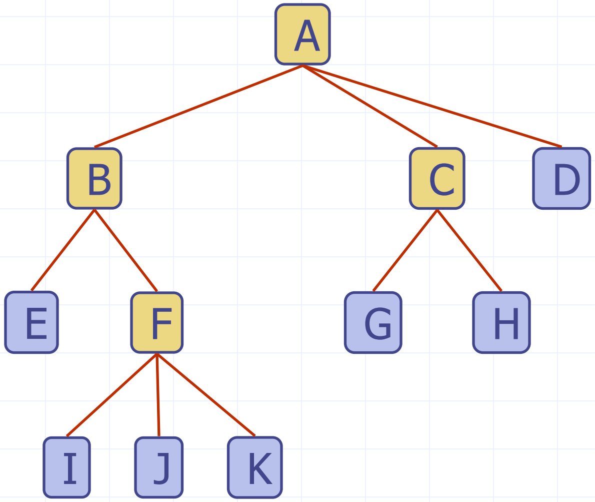
Trees - Examples (2)



Portion of a file system

Trees - Terminology

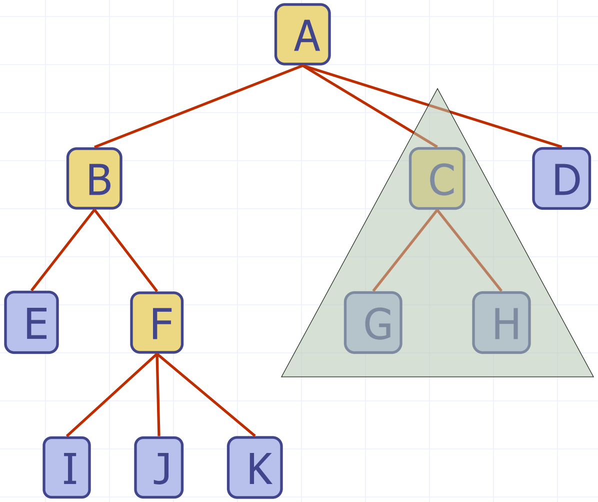
- A is the root node
- B is parent of E and F
- A is ancestor of E and F
- E and F are descendants of A
- C is the sibling of B
- E and F are children of B
- E, I, J, K, G, H, and D are leaves
- A, B, C, and F are internal nodes



Trees - Terminology (2)

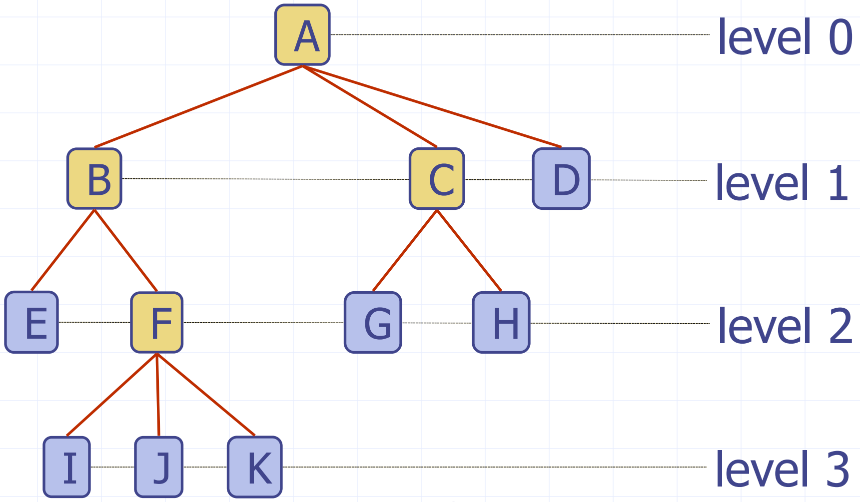
- A is the root node
- B is parent of E and F
- A is ancestor of E and F
- E and F are descendants of A
- C is the sibling of B
- E and F are children of B
- E, I, J, K, G, H, and D are leaves
- A, B, C, and F are internal nodes

- Subtree: tree consisting of node and its descendants



Trees - Terminology (3)

- The depth (level) of E is 2
- The height of the tree is 3
- The degree of node F is 3



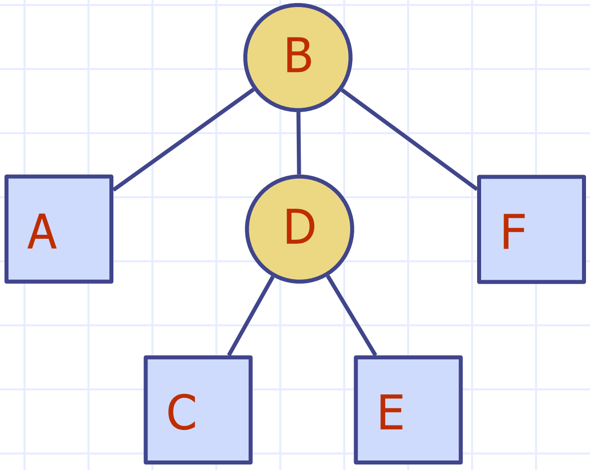
Tree ADT

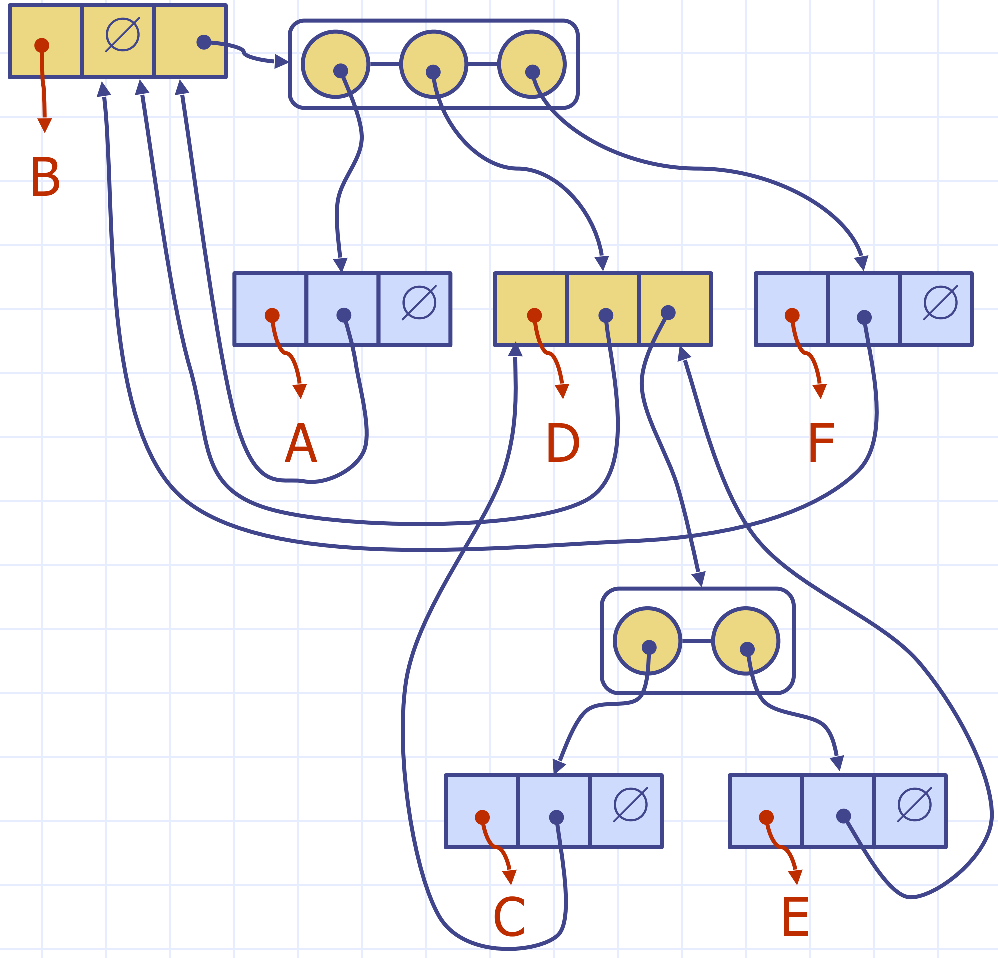
- We use positions to abstract nodes
- Generic methods:
  - Integer len()
  - Boolean is_empty()
  - Iterator positions()
  - Iterator iter()
- Accessor methods:
  - position root()
  - position parent(p)
  - Iterator children(p)
  - Integer num_children(p)

- Query methods:
  - Boolean is_leaf(p)
  - Boolean is_root(p)
- Update method:
  - element replace (p, o)
- Additional update methods may be defined by data structures implementing the Tree ADT

Linked Structure for Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes
- Node objects implement the Position ADT





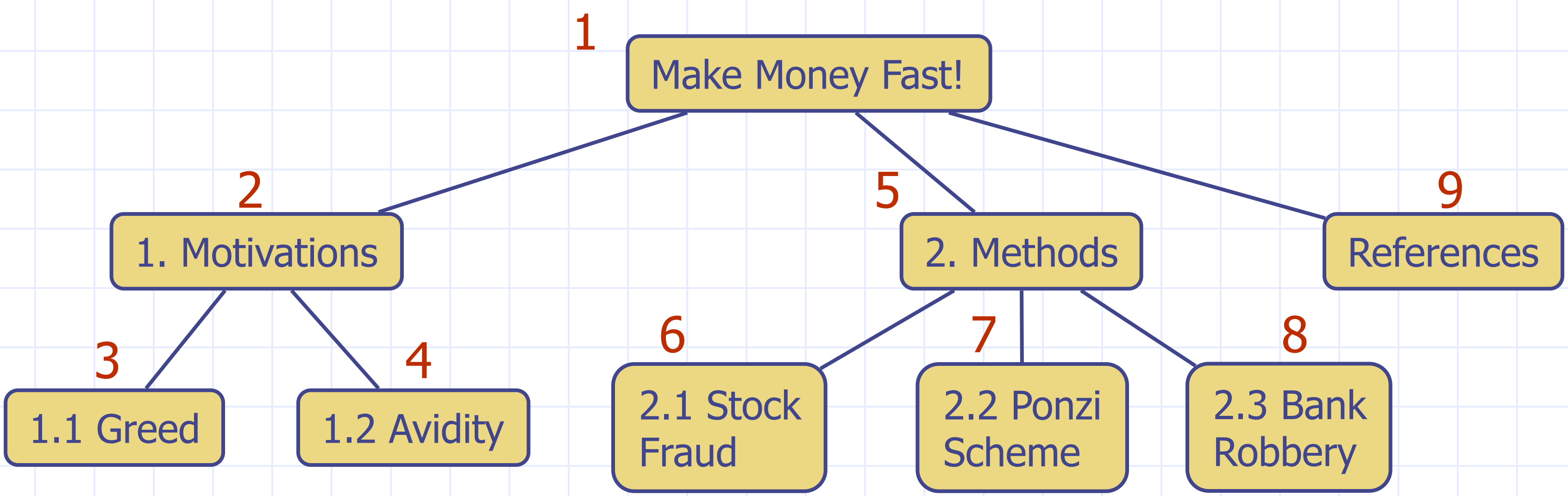
Abstract Tree Class in Python

```
class Tree:
  """Abstract base class representing a tree structure."""

  #------------------------------- nested Position class -------------------------------
  class Position:
    """An abstraction representing the location of a single element."""

    def element(self):
      """Return the element stored at this Position."""
      raise NotImplementedError('must be implemented by subclass')

    def __eq__(self, other):
      """Return True if other Position represents the same location."""
      raise NotImplementedError('must be implemented by subclass')

    def __ne__(self, other):
      """Return True if other does not represent the same location."""
      return not (self == other)            # opposite of __eq__

  # ---------- abstract methods that concrete subclass must support ----------
  def root(self):
    """Return Position representing the tree's root (or None if empty)."""
    raise NotImplementedError('must be implemented by subclass')

  def parent(self, p):
    """Return Position representing p's parent (or None if p is root)."""
    raise NotImplementedError('must be implemented by subclass')

  def num_children(self, p):
    """Return the number of children that Position p has."""
    raise NotImplementedError('must be implemented by subclass')

  def children(self, p):
    """Generate an iteration of Positions representing p's children."""
    raise NotImplementedError('must be implemented by subclass')

  def __len__(self):
    """Return the total number of elements in the tree."""
    raise NotImplementedError('must be implemented by subclass')

  # ---------- concrete methods implemented in this class ----------
  def is_root(self, p):
    """Return True if Position p represents the root of the tree."""
    return self.root( ) == p

  def is_leaf(self, p):
    """Return True if Position p does not have any children."""
    return self.num_children(p) == 0

  def is_empty(self):
    """Return True if the tree is empty."""
    return len(self) == 0
```

Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

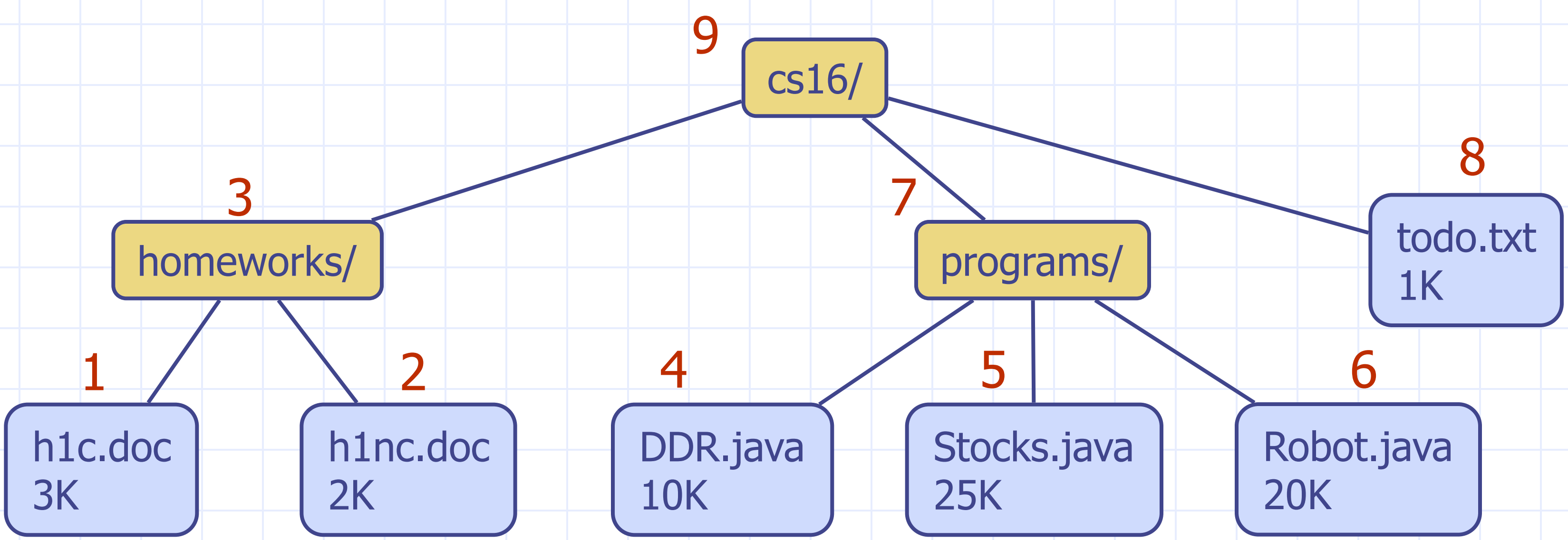
Algorithm *preOrder*(v)
    visit(v)
    for each child w of v
        preorder (w)



Postorder Traversal

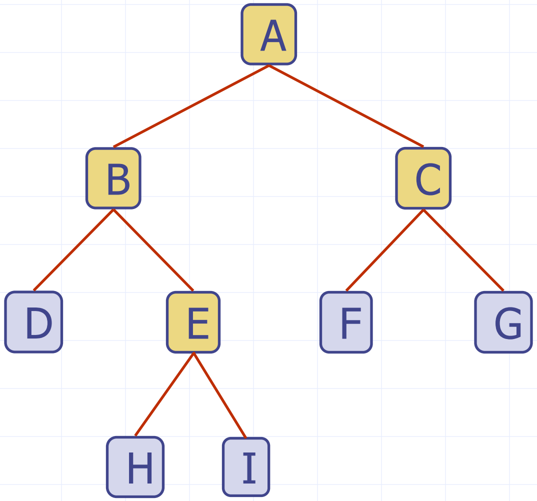
- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

```
Algorithm postOrder(v)
    for each child w of v
        postOrder (w)
    visit(v)
```



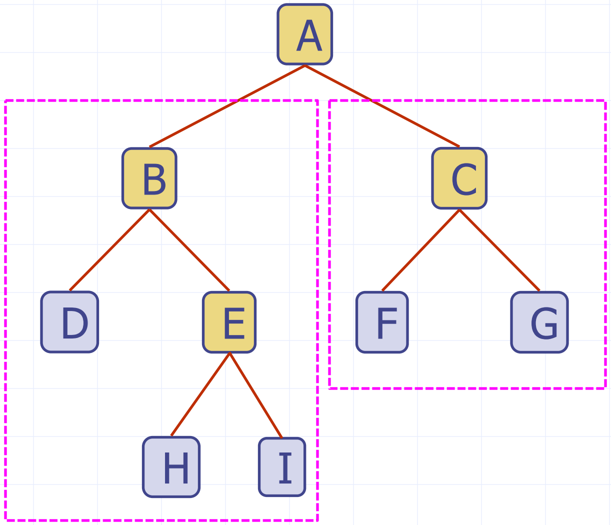
Binary Trees

- An **ordered tree** is one in which the children of each node are ordered
- **Binary tree**: ordered tree with all nodes having at most 2 children
  - left child and right child



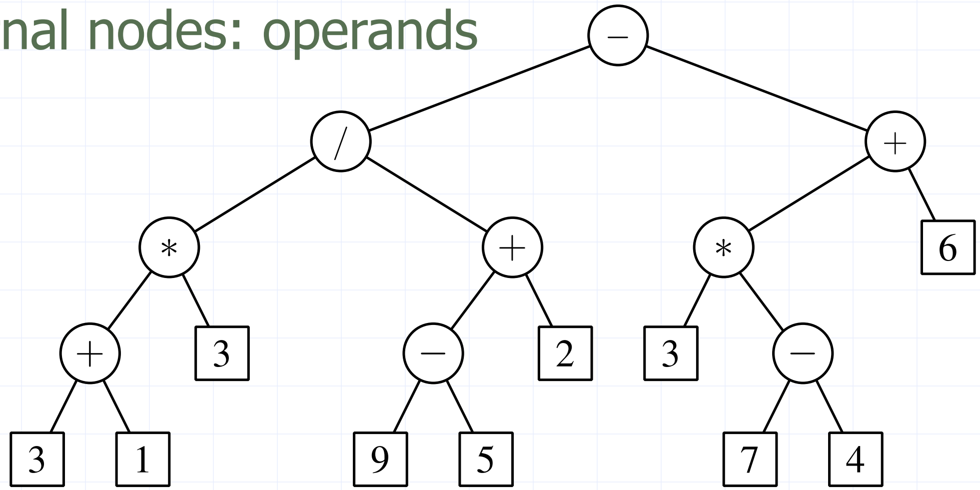
Binary Trees

- Recursive definition of binary tree
  - either a leaf or
  - an internal node (the root) and one/ two binary trees (left subtree and/or right subtree)



Example of Binary Trees - Arithmetic Expression Tree

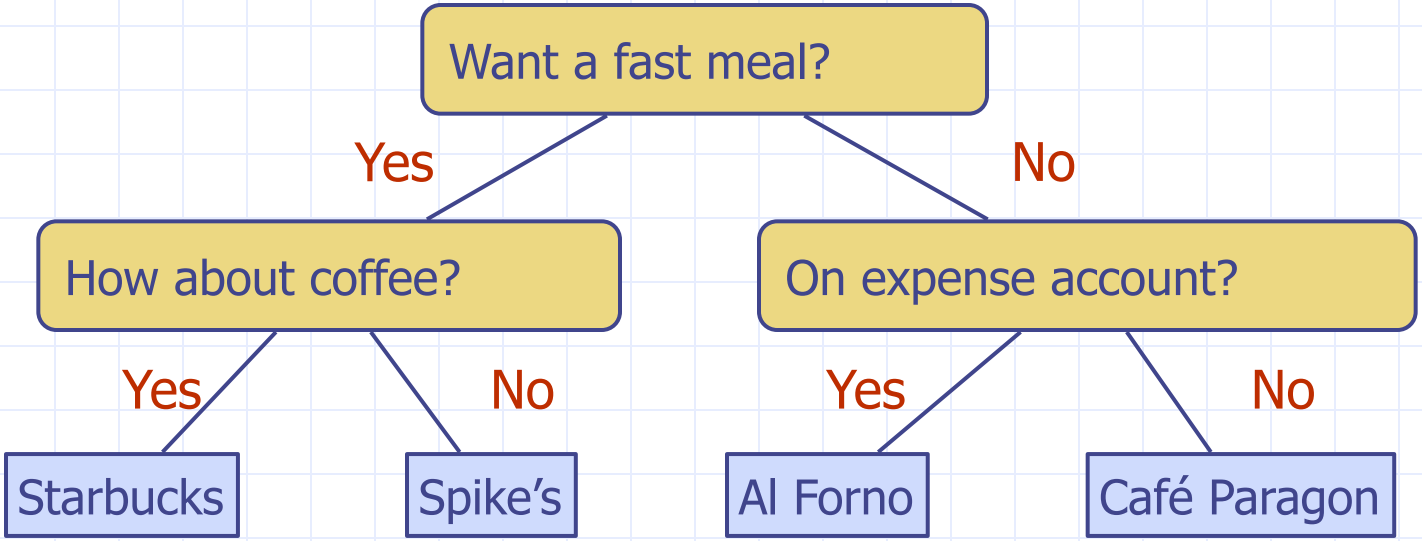
- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands



((((3 + 1) *3) / (9 - 5)+2))-((3*(7 - 4)) +6))

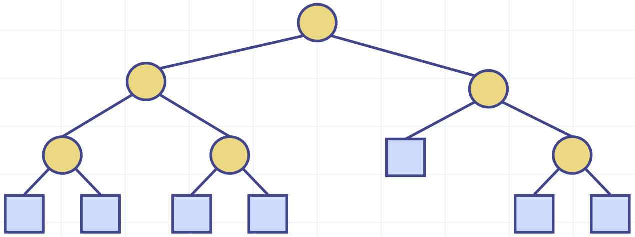
Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision

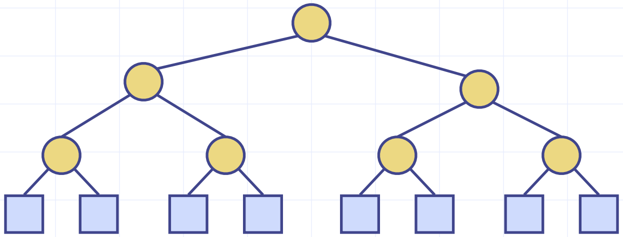


Proper, Full, Complete Binary Trees

- Proper/Full - Every node has either zero or two children

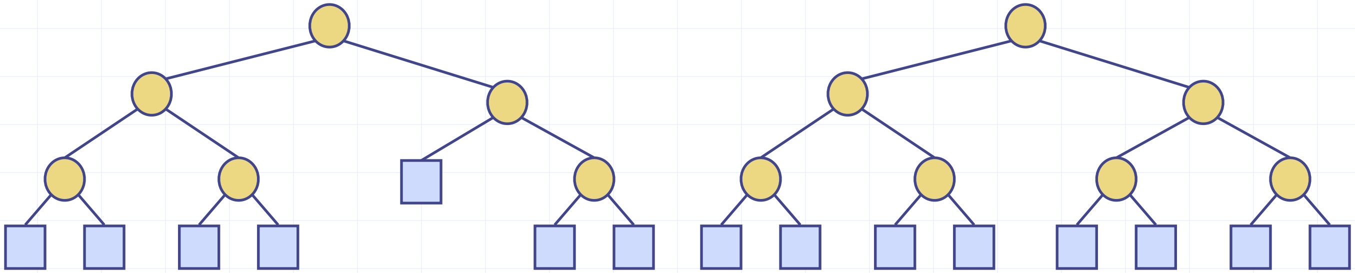


- Complete - every level except possibly the last is completely filled and all leaf nodes are as left as possible.



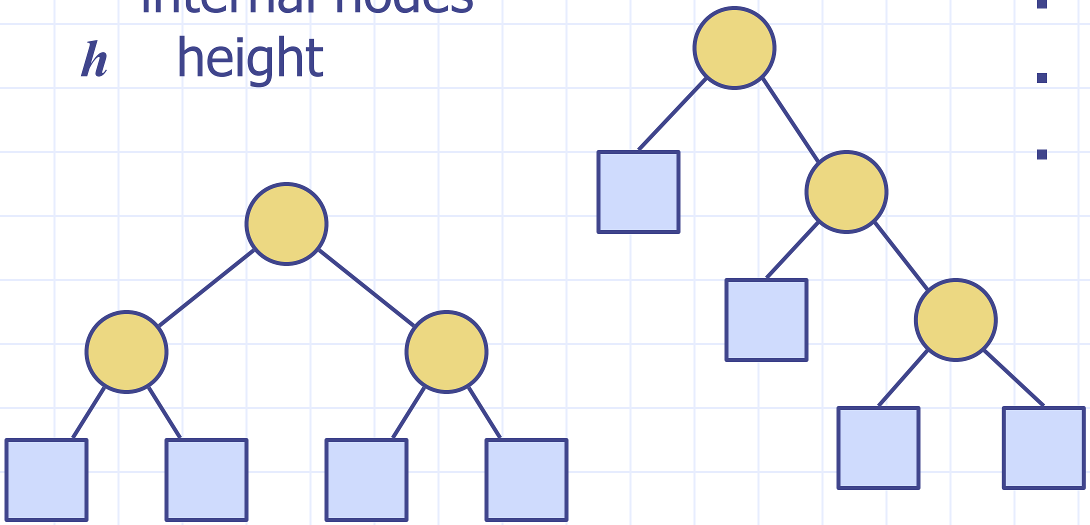
Binary tree from a complete binary tree

- A binary tree can be obtained from appropriate complete binary tree by pruning.



Properties of Proper Binary Trees

- Notation
  - n number of nodes
  - e number of external nodes
  - i number of internal nodes
  - h height



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

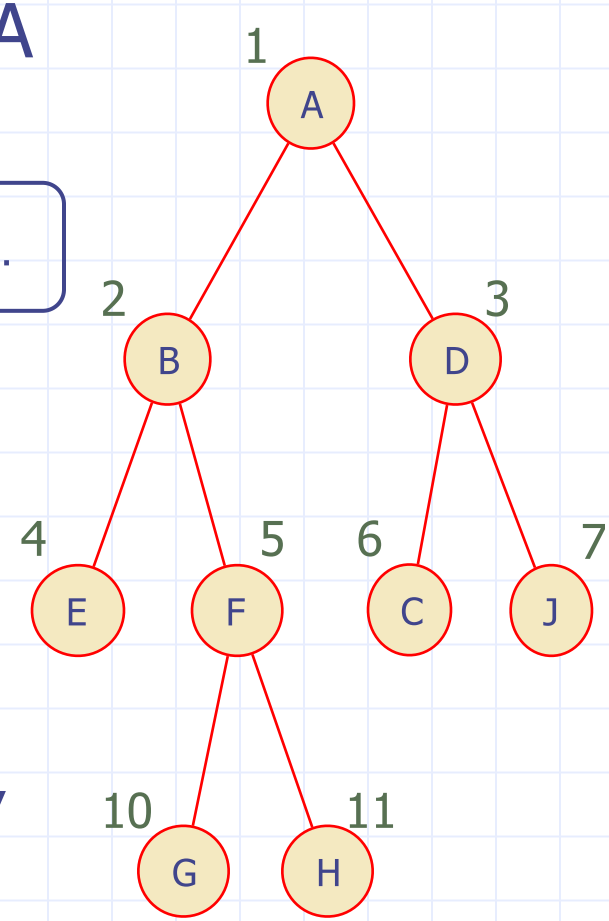
BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position left(p)
  - position right(p)
  - position sibling(p)
- Update methods may be defined by data structures implementing the BinaryTree ADT

Array-Based Representation of Binary Trees

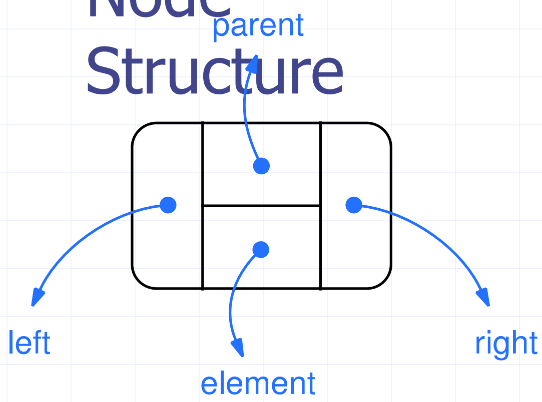
- Nodes are stored in an array A

- Node v is stored at A[rank(v)]
  - rank(root) = 1
  - if node is the left child of parent(node), rank(node) = 2 · rank(parent(node))
  - if node is the right child of parent(node), rank(node) = 2 · rank(parent(node)) + 1



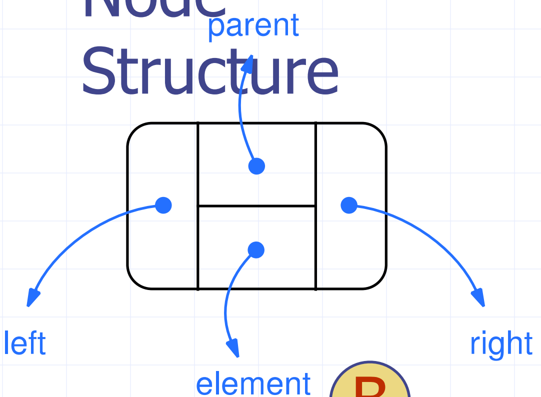
Linked Structure for Binary Trees

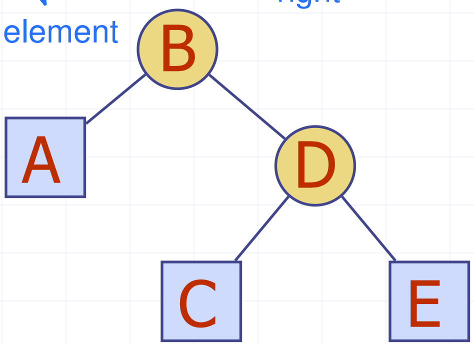
Node Structure

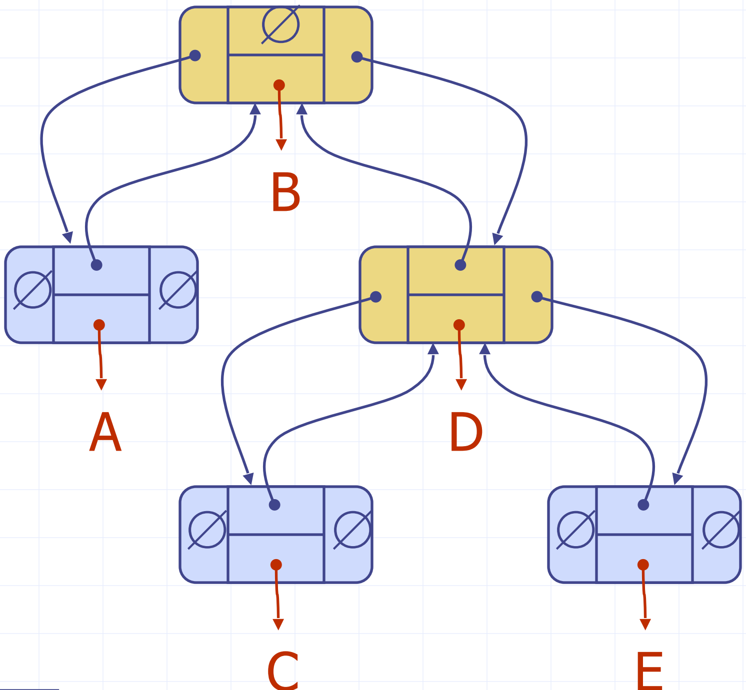


Linked Structure for Binary Trees

Node Structure



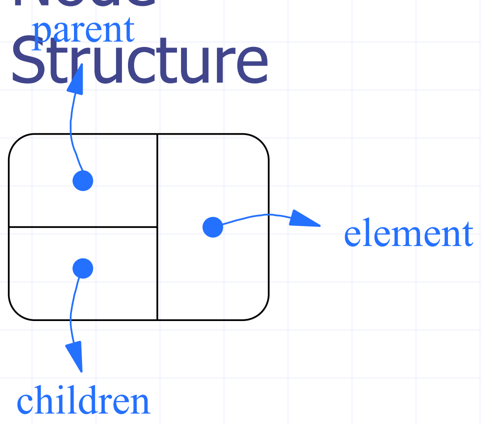




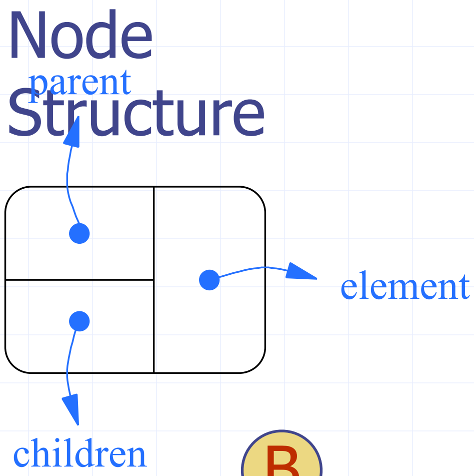
Code Fragments
8.8, 8.9, 8.10, 8.11

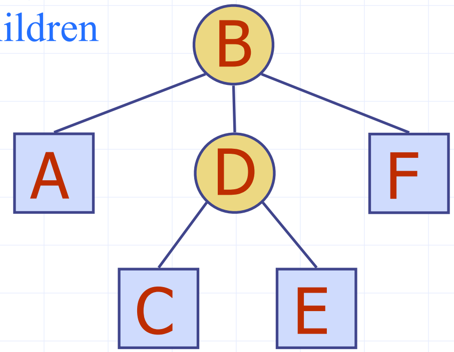
Linked Structure for General Trees

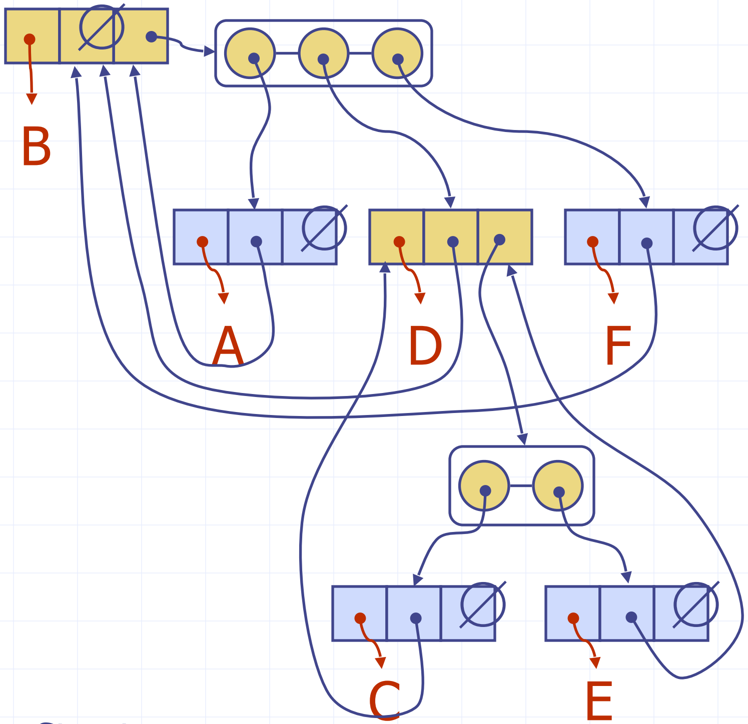
- Node Structure



Linked Structure for General Trees







Computing Depth

- p be a position within the tree T
- calculate depth(p)

```
def depth(self, p):
"""Return the number of levels separating
Position p from the root."""
     if self.is_root(p):
          return 0
     else:
          return 1 +
self.depth(self.parent(p))
```

Computing Height

```
def _height2(self, p): # time is linear in size of subtree
"""Return the height of the subtree rooted at Position p."""
if self.is_leaf(p):
return 0
else:
return 1 + max(self._height2(c) for c in self.children(p))
```

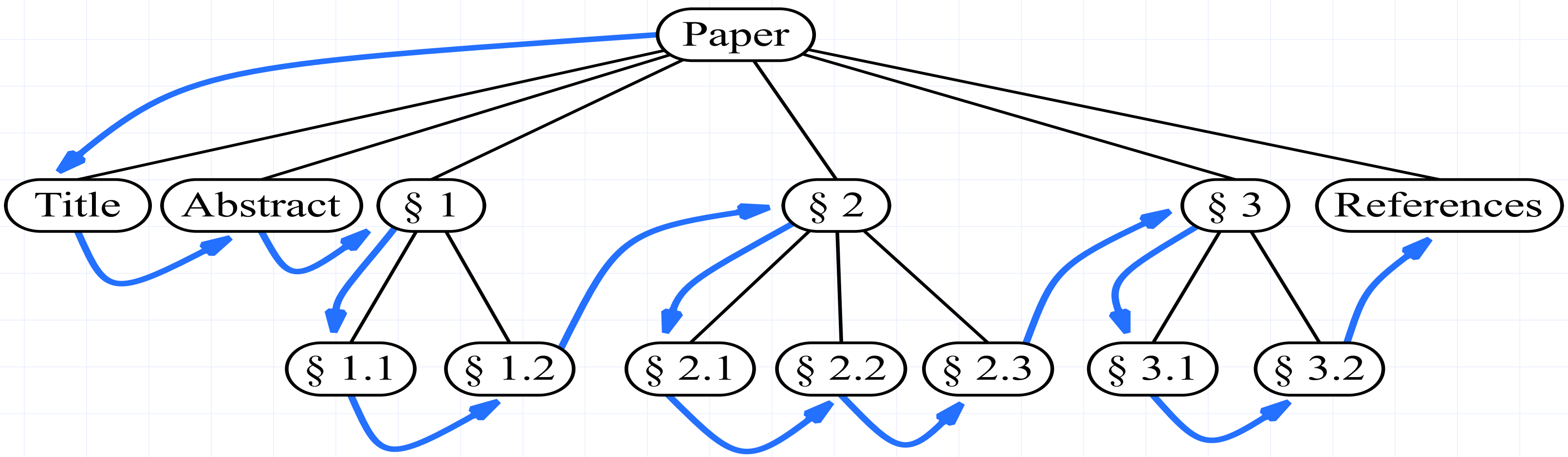
Analysis

$O(\Sigma_p(c_p+1))$ is $O(n)$

Tree Traversals

- Systematic way of visiting all nodes in a tree in a specified order
  - preorder - processes each node before processing its children
  - postorder - processes each node after processing its children

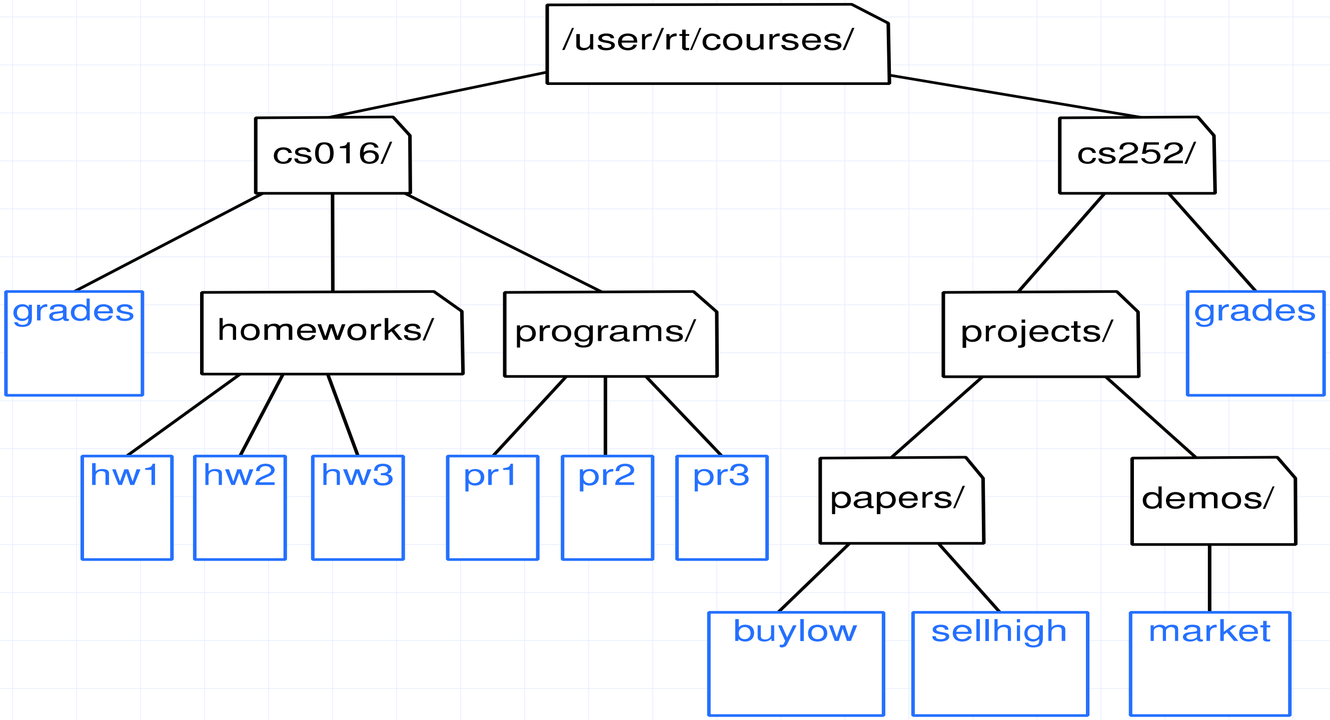
Preorder Traversal



Preorder Traversal - Algorithm

- Algorithm preorder(p)
  - perform the "visit" action for position p
  - for each child c in children(p) do
    - preorder(c)
- Example:
  - reading a document from beginning to end

Postorder Traversal



Postorder Traversal - Algorithm

- Algorithm postorder(p)
  - for each child c in children(p) do
    - postorder(c)
  - perform the "visit" action for position p
- Example
  - du - disk usage command in Unix

Traversals of Binary Trees

- preorder(v)
  - visit(v)
  - preorder(v.leftchild())
  - preorder(v.rightchild())
- postorder(v)
  - postorder(v.leftchild())
  - postorder(v.rightchild())
  - visit(v)

More Example of Traversals

- Visit - printing the data in the node
- Preorder traversal
  - a b d e h i c f g
- Postorder traversal
  - d h i e b f g c a