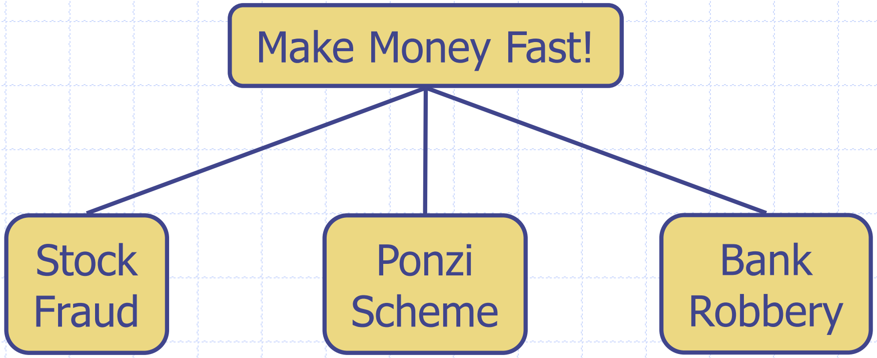
Trees



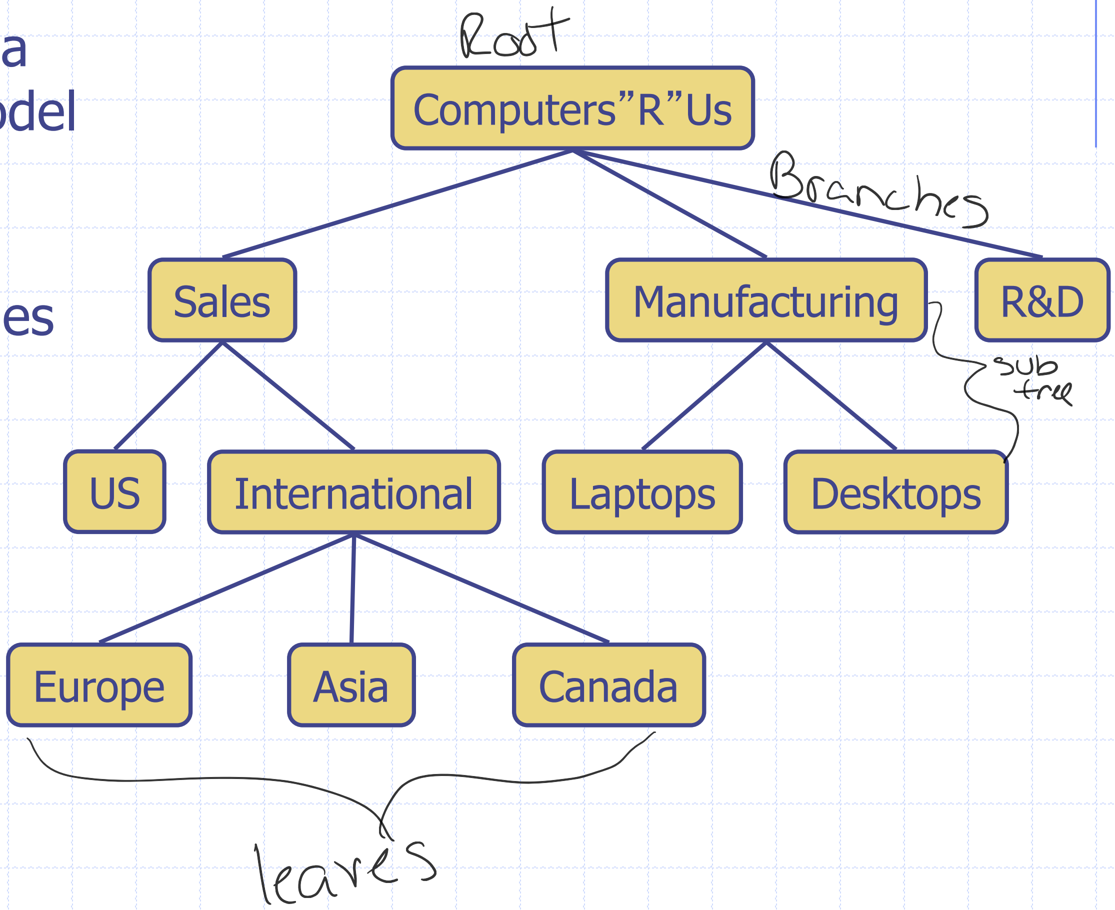
Sareh Taebi
COP3410 – Florida Atlantic University

Nonlinear Data Structures

- Linear data structures: array-based lists or linked lists
  - A 'before' and 'after' relationship between objects in a sequence ↳related to neighbors
- Nonlinear data structures: Trees
  - Relationships are 'hierarchical'
  - Some objects being 'above' and some 'below' others
    ↳family tree
    ↳hierarchy in company

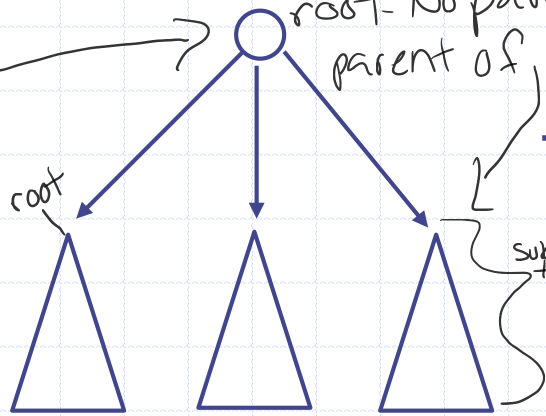
What is a Tree

- 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



Tree Definition

- we define a ***tree T*** as a set of ***nodes*** storing elements such that the nodes have a ***parent-child*** relationship that satisfies the following properties:
  - If T is nonempty, it has a special node, called the root of T, that has no parent.
  - Each node v of T different from the root has a unique parent node w; every node with parent w is a child of w.



Trees can be defined recursively

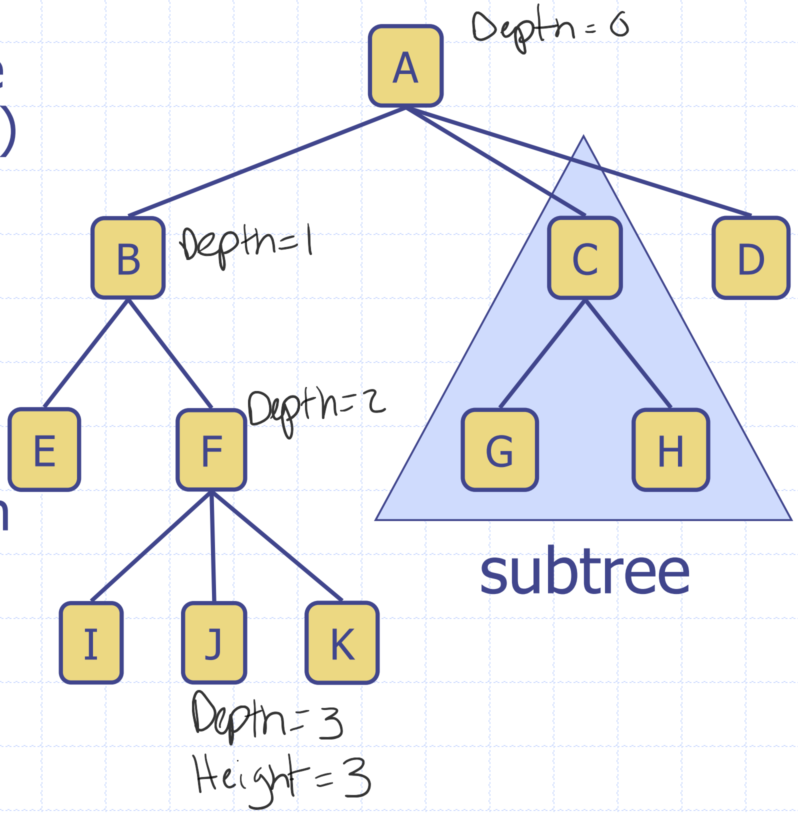
Tree is either empty → no root
OR
Not empty
↳has root
↳root connected to sub tree

Tree Terminology

- **Root**: node without parent (A)
- **Internal node**: node with at least one child (A, B, C, F)
- **External node** (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- **Ancestors of a node**: parent, grandparent, grand-grandparent, etc.
- **Depth of a node**: number of ancestors
- **Height of a tree**: maximum depth of any node (3)
- **Descendant of a node**: child, grandchild, grand-grandchild, etc.

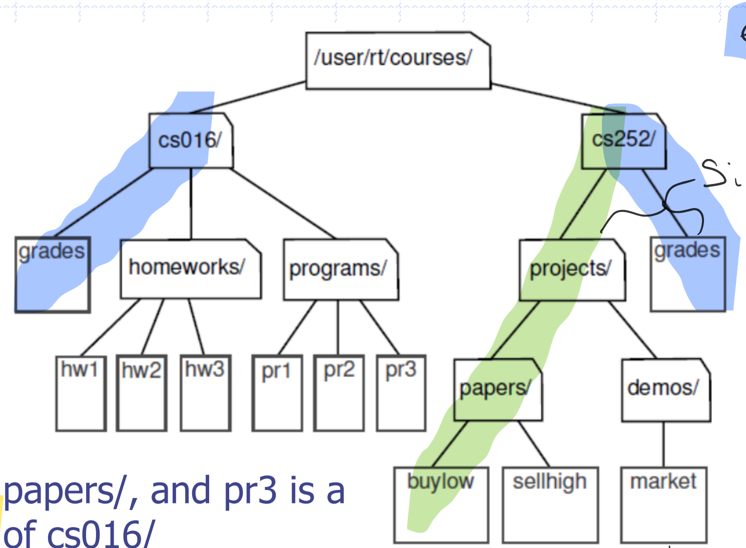
pg 307/308

- **Subtree**: tree consisting of a node and its descendants



Other Node Relationships

- Two nodes that are children of the same parent are *siblings*.
- A node *v* is *external* if v has no children.
- A node *v* is *internal* if it has one or more children.
- External nodes are also known as leaves.



edges

siblings

path

leaves/external

s252/ is an ancestor of papers/, and pr3 is a descendant of cs016/

Edges and Paths in Trees

- An *edge* of tree T is a pair of nodes (u,v) such that u is the parent of v, or vice versa.
- A *path* of T is a sequence of nodes such that any two consecutive nodes in the sequence form an edge.

(cs252/, projects/, demos/, market).

Ordered Tree

can be ordered
↳children left → right : ages

- A tree is ***ordered*** if there is a meaningful linear order among the children of each node;
- We purposefully identify the children of a node as being the first, second, third, and so on.
- Such an order is usually visualized by arranging siblings left to right, according to their order.

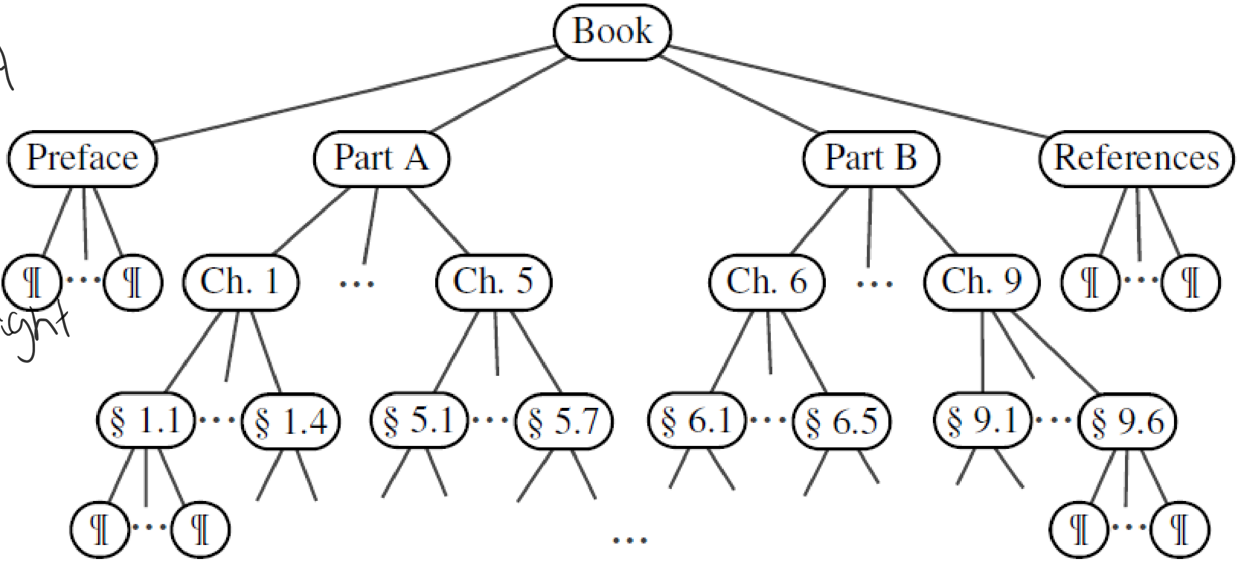
Ordered Tree Example

The components of a structured document, such as a book, are hierarchically organized as a tree whose internal nodes are parts, chapters, and sections, and whose leaves are paragraphs, tables, figures, and so on.

hierarchy
↳levels

Order
↳children
↳left→right

binary tree
↳at most 2 children per node



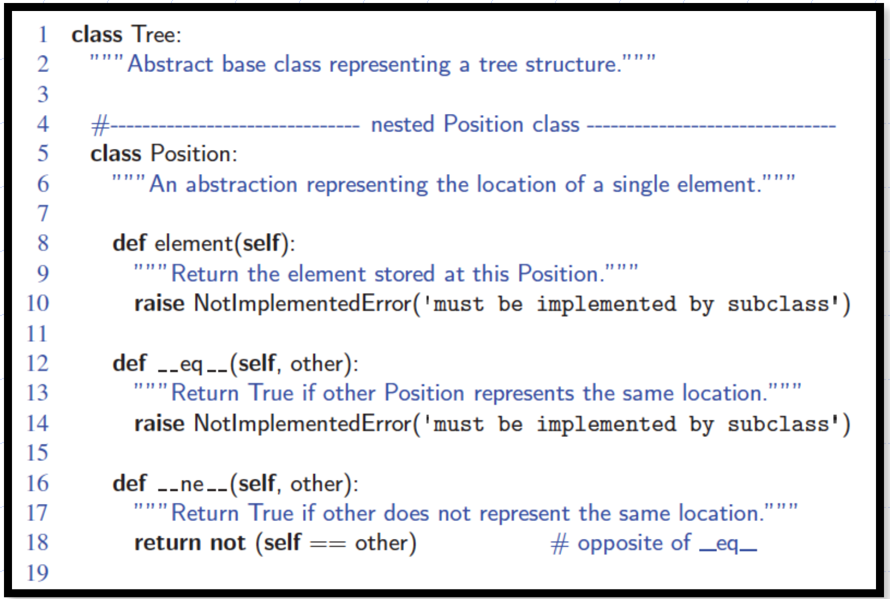
Basic functions Tree should support

Tree ADT

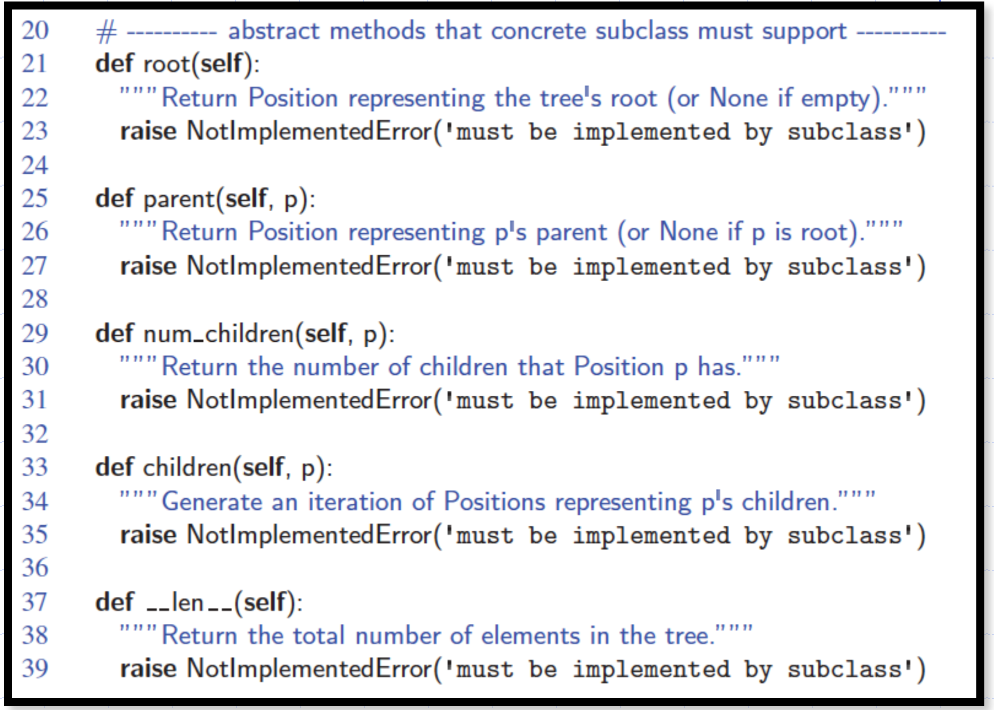
- We use *positions* to abstract nodes
- Let p be the position of a node of a tree T
- Generic methods:
  - Integer len()
  - Boolean is_empty()
  - Iterator positions()
  - Iterator iter()
- Accessor methods:
  - position root()
  - position parent(p)
  - Iterator children(p)
  - Integer num_children(p)

- 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

Abstract Tree Class in Python

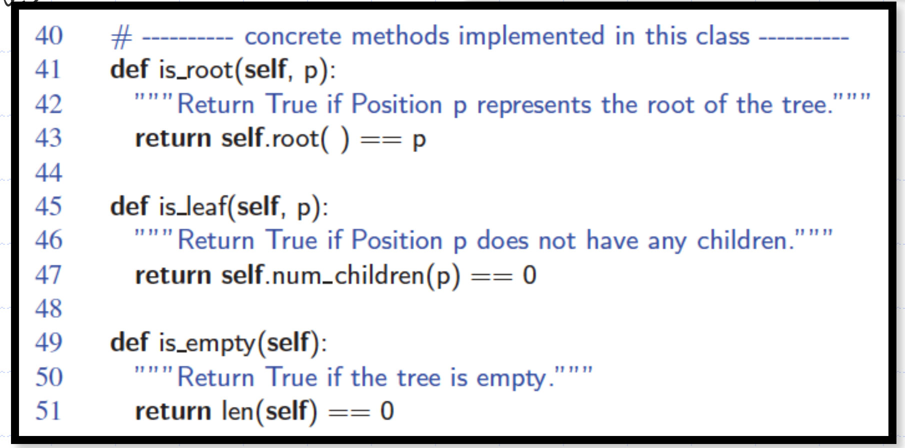


```
class Tree:
    """Abstract base class representing a tree structure."""

    #------------------------------- 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')
```

Queries ↳



```
    # ---------- 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
```

Only inside class tree can access position