CSC148 winter 2014

recursive structures

week 6

Danny Heap / Dustin Wehr
heap@cs.toronto.edu / dustin.wehr@utoronto.ca
BA4270 / SF4306D
http://www.cdf.toronto.edu/~heap/148/F13/

February 12, 2014

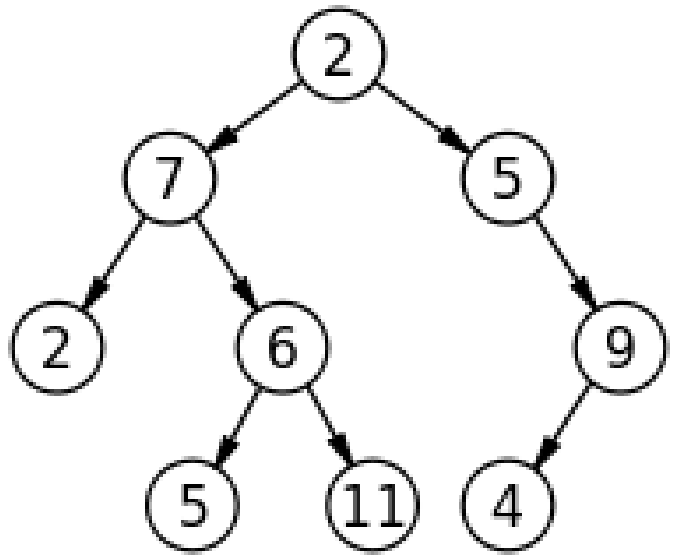
Outline

recursive structures: mathematical definition of trees

binary tree traversals

recursive tree class

recursion, natural and otherwise



terminology

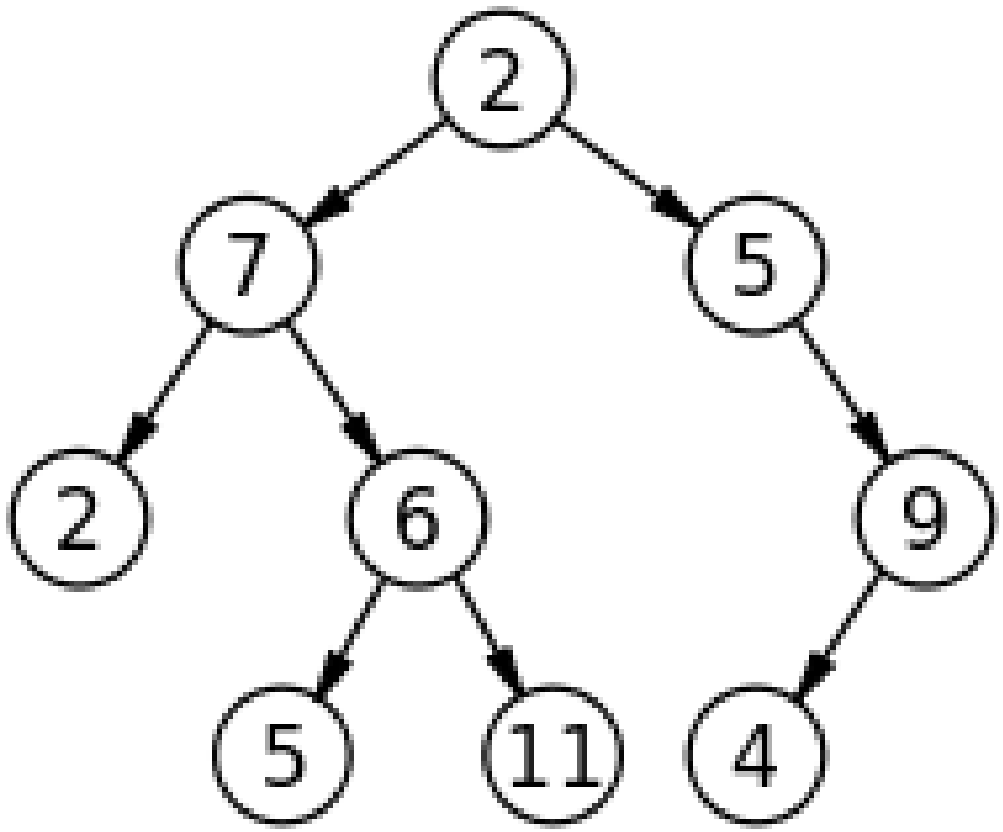
- set of **nodes** (possibly with values or labels), with directed **edges** between some pairs of nodes
- One node is distinguished as **root**
- Each non-root node has exactly one parent.
- A **path** is a sequence of nodes $n_1, n_2, \ldots, n_k$, where there is an edge from n_i to n_{i+1}. The **length** of a path is the number of edges in it
- There is a unique path from the root to each node. In the case of the root itself this is just n_1, if the root is node n_1.
- There are no **cycles** — no paths that form loops.

more terminology

- **leaf**: node with no children
- **internal node**: node with one or more children
- **subtree**: tree formed by any tree node together with its descendants and the edges leading to them.
- **height**: Maximum path length from a leaf to the root. A node also defines a height, which is the maximum path length of the tree rooted at that node
- **arity or branching factor**: maximum number of children for any node.

pre-order traversal

Visit root, then pre-order traverse left subtree, then pre-order traverse right subtree



exercise: code for preorder traversal

```
"""
A TreeList is either None or a Python list with 3 elements, where
  --- element 0 is a value
  --- element 1 is a TreeList
  --- element 2 is a TreeList
"""

def preorder(tl: 'TreeList') -> list:
    """
    Return list of values in tl in preorder

    >>> T = [5, [4, None, None], [3, [2, None, None], [1, None, None]]]
    >>> preorder(T)
    [5, 4, 3, 2, 1]
    """
```

exercise: code for preorder traversal

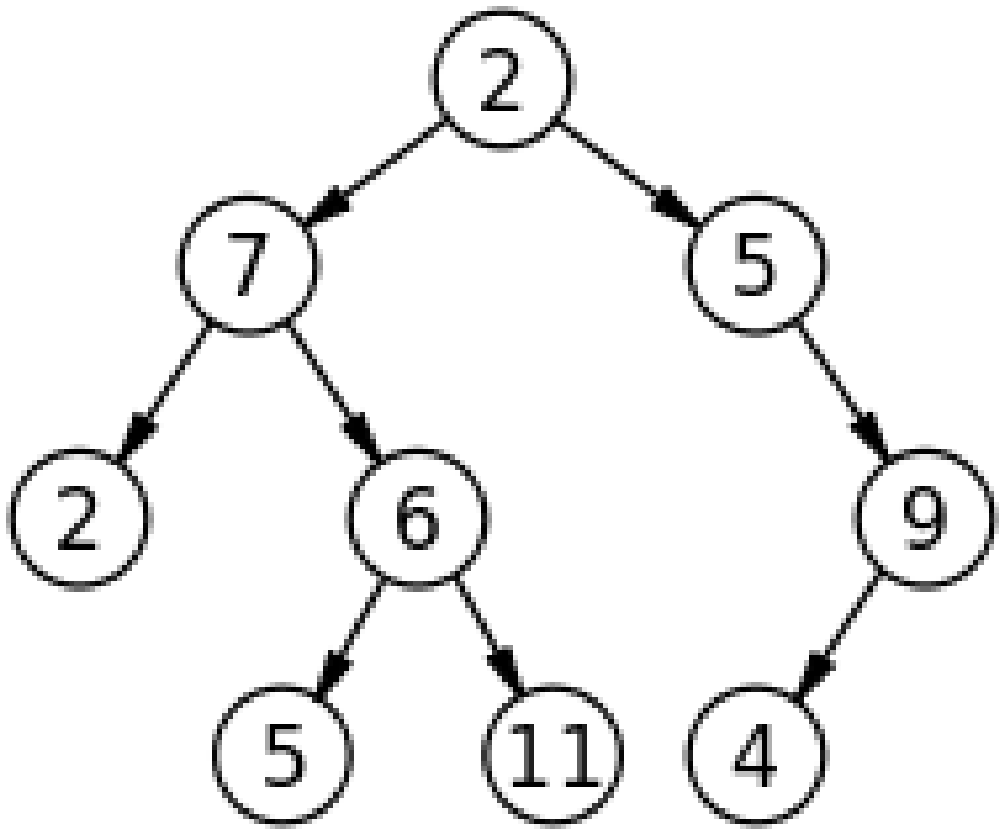
```
"""
A TreeList is either None or a Python list with 3 elements, where
  --- element 0 is a value
  --- element 1 is a TreeList
  --- element 2 is a TreeList
"""

def preorder(tl: 'TreeList') -> list:
    """
    Return list of values in tl in preorder

    >>> T = [5, [4, None, None], [3, [2, None, None], [1, None, None]]]
    >>> preorder(T)
    [5, 4, 3, 2, 1]
    """
    if tl is None:
        return []
    else:
        return [tl[0]] + preorder(tl[1]) + preorder(tl[2])
```

in-order traversal

In-order traverse left subtree, then visit root, then in-order traverse right subtree



exercise: code for inorder traversal

```
"""
A TreeList is either None or a Python list with 3 elements, where
  --- element 0 is a value
  --- element 1 is a TreeList
  --- element 2 is a TreeList
"""

def inorder(tl: 'TreeList') -> list:
    """
    Return list of values in tl in order

    >>> T = [5, [4, None, None], [3, [2, None, None], [1, None, None]]]
    >>> inorder(T)
    [4, 5, 2, 3, 1]
    """
```

exercise: code for inorder traversal

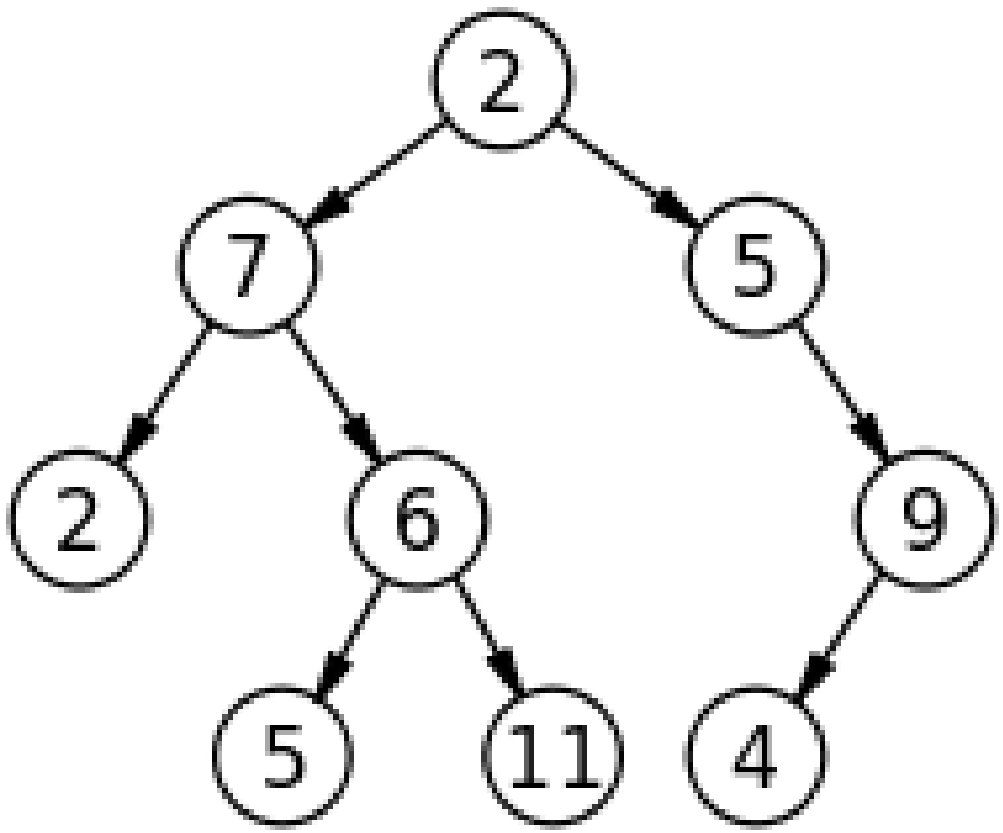
```
"""
A TreeList is either None or a Python list with 3 elements, where
  --- element 0 is a value
  --- element 1 is a TreeList
  --- element 2 is a TreeList
"""

def inorder(tl: 'TreeList') -> list:
    """
    Return list of values in tl in order

    >>> T = [5, [4, None, None], [3, [2, None, None], [1, None, None]]]
    >>> inorder(T)
    [4, 5, 2, 3, 1]
    """
    if tl is None:
        return []
    else:
        return  inorder(tl[1]) + [tl[0]] + inorder(tl[2])
```

post-order traversal

Post-order traverse left subtree, then post-order traverse right subtree, then visit root



exercise: code for postorder traversal

```
"""
A TreeList is either None or a Python list with 3 elements, where
  --- element 0 is a value
  --- element 1 is a TreeList
  --- element 2 is a TreeList
"""

def postorder(tl: 'TreeList') -> list:
    """
    Return list of values in tl in postorder

    >>> T = [5, [4, None, None], [3, [2, None, None], [1, None, None]]]
    >>> postorder(T)
    [4, 2, 1, 3, 5]
    """
```

exercise: code for postorder traversal

```
"""
A TreeList is either None or a Python list with 3 elements, where
  --- element 0 is a value
  --- element 1 is a TreeList
  --- element 2 is a TreeList
"""

def postorder(tl: 'TreeList') -> list:
    """
    Return list of values in tl in postorder

    >>> T = [5, [4, None, None], [3, [2, None, None], [1, None, None]]]
    >>> postorder(T)
    [4, 2, 1, 3, 5]
    """
    if tl is None:
        return []
    else:
        return  postorder(tl[1]) + postorder(tl[2]) + [tl[0]]
```

general tree implementation

Python **list** class has way more methods and attributes than needed. Let's specialize on **Tree** ADT.

```
class Tree:
    def __init__(self: 'Tree',
                 value: object =None, children: list =None):
        """Create a node with value and any number of children"""

        self.value = value
        if not children:
            self.children = []
        else:
            self.children = children[:] # quick-n-dirty copy of list

    def __contains__(self: 'Tree' , value: object) -> bool:
        """True if Tree has a node with value
        """
        return (self.value == value or
                any([t.__contains__(value) for t in self.children]))
```