CSC148 winter 2015

recursive structures

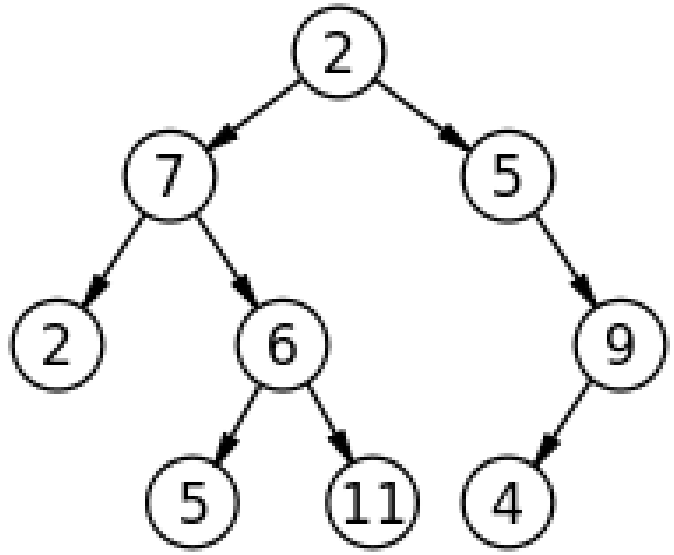
week 6

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~csc148h/winter/
416-978-5899

February 17, 2015

Outline

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**: 1 + the maximum path length in a tree. A node also has a height, which is 1 + the maximum path length of the tree rooted at that node
- **depth**: Height of the entire tree minus the height of a node is the depth of the node.
- **arity, branching factor**: maximum number of children for any node.

general tree implementation

```
class Tree:
    ''' Represent a Bare-bones Tree ADT'''

    def __init__(self, value=None, children=None):
        ''' (Tree, object, list) -> NoneType

        Create Tree(self) with root node value
        and 0 or more children.
        '''
        self.value = value
        # copy children if not None
        # Notice the functional 'if'
        self.children = children.copy() if children else []
```

height of this tree?

```
def height(t):
    ''' (Tree) -> int

    Return 1 + length of longest path in Tree t.

    >>> tn2 = Tree(2, [Tree(4), Tree(4.5), Tree(5), Tree(5.75)])
    >>> tn3 = Tree(3, [Tree(6), Tree(7)])
    >>> tn1 = Tree(1, [tn2, tn3])
    >>> height(tn1)
    3
    '''
    # 1 more node than the maximum height of a child, except
    # what happens if there are no children?
```

general form of recursion:

if ⟨condition to detect a base case⟩:

    ⟨do something without recursion⟩

else: # ⟨general case⟩

    ⟨do something that involves recursive call(s)⟩

how many leaves?

```
def leaf_count(t):
    ''' (Tree) -> int:

    Return the number of leaves in Tree t.
    >>> t = Tree(5)
    >>> leaf_count(t)
    1
    >>> tn2 = Tree(2, [Tree(4), Tree(4.5), Tree(5), Tree(5.75)])
    >>> tn3 = Tree(3, [Tree(6), Tree(7)])
    >>> tn1 = Tree(1, [tn2, tn3])
    >>> leaf_count(tn1)
    6
    '''
```

arity, or branching factor

```
def arity(t: Tree) -> int:
    '''Maximum branching factor of tree T

    >>> t = Tree(15)
    >>> arity(t)
    0
    >>> tn2 = Tree(2, [Tree(4), Tree(4.5), Tree(5), Tree(5.75)])
    >>> tn3 = Tree(3, [Tree(6), Tree(7)])
    >>> tn1 = Tree(1, [tn2, tn3])
    >>> arity(tn1)
    4
    '''
```