CSC148-Section:L0301
Week#7-Friday

Instructed by
AbdulAziz Al-Helali
a.alhelali@mail.utoronto.ca
Office hours: Wednesday 11-1, BA2230.

Slides adapted from Professor Danny Heap course material
winter17
Outline

• Recursive structures
  • Trees
recursion, natural and otherwise
structure to organize information
**Terminology**

- **Tree**: 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 $n_1, n_2, ..., n_k$ where there is an edge from $n_i, n_{i+1}$

- 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.
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**: length of a path from root to a node is the node's depth.

• **arity, branching factor**: maximum number of children for any node.
**Tree ADT**

```python
class Tree:
    
    ""
    A bare-bones Tree ADT that identifies the root with the entire tree.
    ""
    
    def __init__(self, value: object=None, children: List['Tree']=None) -> None:
        ""
        Create Tree self with content value and 0 or more children
        ""
        self.value = value
        # copy children if not None
        self.children = children.copy() if children else []
```

*Do Not assign a parameter to the empty list [] in a method definition instead make it None and assign it to [] inside the method body*
Do Not assign a parameter to the empty list `[]` in a method definition instead make it `None` and assign it to `[]` inside the method body.

```python
>>> def fun(n:int, L=[]):
...     L.append(n)
...     return L
...     return L
... >>> fun(5)
[5]
>>> fun(7)
[5, 7]
```

Corrected version:

```python
>>> def fun(n:int, L=None):
...     L=[]
...     L.append(n)
...     return L
...     return L
... >>> fun(5)
[5]
>>> fun(7)
[7]
```
Template for Tree recursive functions

```python
if t.children == []: #Base case
    return some_result

else: #General case
    return helper_function([your_rec_func(x) for x in t.children])
```
how many leaves?

def leaf_count(t: Tree) -> int:
    """
    Return the number of leaves in Tree t.
    """

>>> t = Tree(7)
>>> leaf_count(t)
1

>>> t = descendants_from_list(Tree(7),
                                [0, 1, 3, 5, 7, 9, 11, 13], 3)

>>> leaf_count(t)
6
    """
def leaf_count(t: Tree) -> int:
    """
    Return the number of leaves in Tree t.
    >>> t = Tree(7)
    >>> leaf_count(t)
    1
    >>> t = descendants_from_list(Tree(7),[0, 1, 3, 5, 7, 9, 11,
    >>> leaf_count(t)
    6
    """

    if t.children==[]:
        return 1
    else:
        return sum([leaf_count(x) for x in t.children])
height of this tree?

def height(t: Tree):
    """
    Return 1 + length of longest path of t.
    """

>>> t = Tree(13)
>>> height(t)
1
>>> t = descendants_from_list(Tree(13),
                             [0, 1, 3, 5, 7, 9, 11, 13], 3)
>>> height(t)
3
    """
    # 1 more edge than the maximum height of a child, except
    # what do we do if there are no children?
```python
def height(t: Tree) -> int:
    """
    Return 1 + length of longest path of t.
    >>> t = Tree(13)
    >>> height(t)
    1
    >>> t = descendants_from_list(Tree(13),[0, 1, 3, 5, 7, 9, 11],
    >>> height(t)
    3
    """
    # 1 more edge than the maximum height of a child, except
    # what do we do if there are no children?
    # helpful helper function
    if t.children == []:
        return 1
    else:
        return 1 + max([height(x) for x in t.children])
```
arity, or branching factor

def arity(t: Tree) -> int:
    """
    Return the maximum branching factor (arity) of Tree t.
    """
    
    >>> t = Tree(23)
    >>> 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
    """
```python
def arity(t: Tree) -> int:
    
    """
    Return the maximum branching factor (arity) of Tree t.
    >>> t = Tree(23)
    >>> 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
    """

    if t.children == []:
        return 0
    else:
        y = [arity(x) for x in t.children]
        return max(y) if max(y) > len(y) else len(y)
```

See next slide for another way of doing the same thing
```python
def arity(t: Tree) -> int:
    """
    Return the maximum branching factor (arity) of Tree t.
    >>> t = Tree(23)
    >>> 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
    """
    if t.children == []:
        return 0
    else:
        return max(len(t.children), max([arity(x) for x in t.children]))
```