CSC148-Section:L0301 Week#7-Monday

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

Computer Science
UNIVERSITY OF TORONTO

Announcements

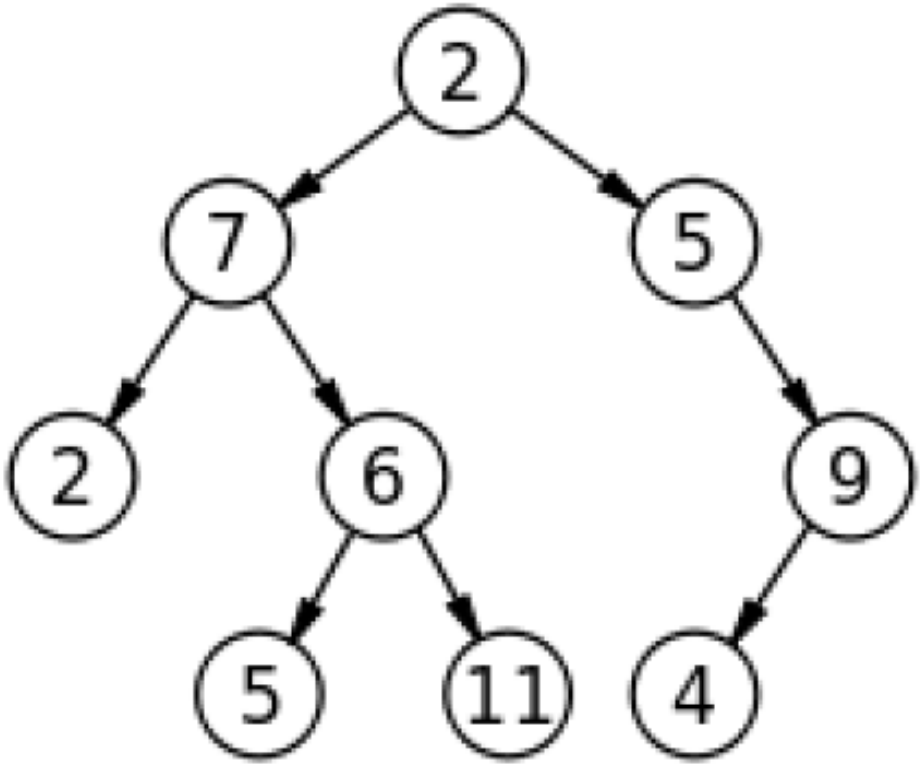
- Test 1 marks are released
  - Average: 76%
  - Median: 80%

- A2 due March 6th, 2018
  - Use office hours
  - Pizza

Outline

- Trees
  - Quick Review
  - Module function vs method
  - Implement methods
  - Example: filesystem using Trees

Quick Review



Tree ADT

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

```
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 []
```

Tree ADT

- __eq__
- __repr__
- __str__
- Are provided for you.

Module function vs method

- Indentation?
- Self attribute?
- inside a class or a module?
- Does it need an instance?

- Last lecture we implemented:
  `leaf_count/height/ arity` **as Module functions**

Computer Science
UNIVERSITY OF TORONTO

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?
```

Implementation as Module function

```
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, 1
    >>> 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])
```

Implementation as Method inside class Tree

```
def height(self) ->int:
    """
    Return length of longest path, + 1, in tree rooted at self.

    >>> t = Tree(5)
    >>> t.height()
    1
    >>> t = descendants_from_list(Tree(7), [0, 1, 3, 5, 7, 9,
           11, 13], 3)
    >>> t.height()
    3
    """
    if self.children==[]:
        return 1
    else:
        return 1 + max([c.height() for c in self.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
    """
```

Implementation as Module function

```
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])
```

Computer Science
UNIVERSITY OF TORONTO

Implementation as Method inside class Tree

```
def leaf_count(self) -> int:
    """
    Return the number of leaves in Tree t.
    >>> t = Tree(7)
    >>> t.leaf_count()
    1
    >>> t = descendants_from_list(Tree(7),[0, 1, 3, 5, 7, 9,
11, 13], 3)
    >>> t.leaf_count()
    6
    """
    if self.children == []:
        return 1
    else:
        return sum([x.leaf_count() for x in self.children])
```

Computer Science
UNIVERSITY OF TORONTO

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
    """
```

Implementation as Module function

```
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

Flatten a Tree

```
def flatten(self) -> List:
    """ Return a list of all values in tree rooted at self.
    >>> t = Tree(5)
    >>> t.flatten()
    [5]
    >>> t = descendants_from_list(Tree(7), [0, 1, 3, 5, 7, 9,
11, 13], 3)
    >>> L = t.flatten()
    >>> L.sort()
    >>> L == [0, 1, 3, 5, 7, 7, 9, 11, 13]
    True
    """
```

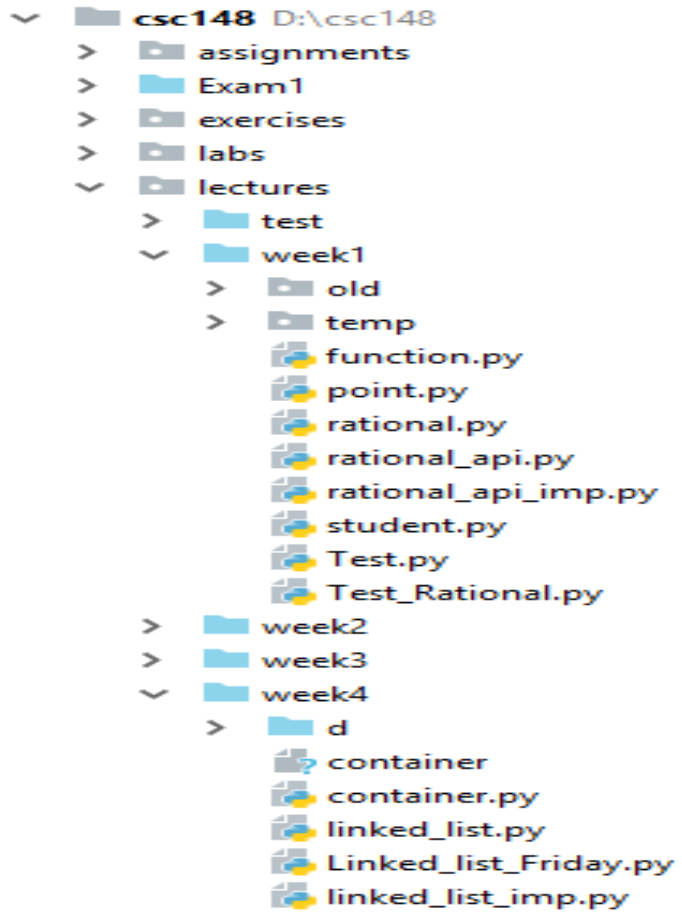
Computer Science
UNIVERSITY OF TORONTO

```
def flatten(self) -> List:
    """ Return a list of all values in tree rooted at self.
    >>> t = Tree(5)
    >>> t.flatten()
    [5]
    >>> t = descendants_from_list(Tree(7), [0, 1, 3, 5, 7, 9,
11, 13], 3)
    >>> L = t.flatten()
    >>> L.sort()
    >>> L == [0, 1, 3, 5, 7, 7, 9, 11, 13]
    True
    """
    if self.children==[]:
        return [self.value]
    else:
        return ([self.value]
                + sum([c.flatten()
                       for c in self.children], []))
```

Implementation as Method inside class Tree

```
def is_leaf(self):
    """Return whether Tree self is a leaf
    @param Tree self:
    @rtype: bool
    >>> Tree(5).is_leaf()
    True
    >>> Tree(5,[Tree(7)]).is_leaf()
    False
    """
    return len(self.children) == 0
```

Example: filesystem using Trees



Example: filesystem using Trees

```
from os import scandir, path
from trees_api_mond import *

def path_to_tree(path_name:str)-> Tree:
    """
    Returns a tree representation for filesystem starting frompath_
    """
    # print(path_name)
    return Tree((path_name,[f.name for f in scandir(path_name)]),
                [path_to_tree(path.join(path_name,f.name))
                 for f in scandir(path_name)
                 if f.is_dir()])
```

scandir

```
>>> from os import scandir,path
>>> cd=[f.name for f in scandir(".")]
>>> cd
['DLLs', 'Doc', 'include', 'Lib', 'libs', 'LICENSE.txt', 'man', 'NEWS.txt', 'pyt
hon.exe', 'python3.dll', 'python36.dll', 'pythonw.exe', 'Scripts', 'tcl', 'Tools
', 'vcruntime140.dll']
>>>
```

path.join

```
>>> path.join("dir1","dir2")
'dir1\\dir2'
>>>
```

Example: filesystem using Trees

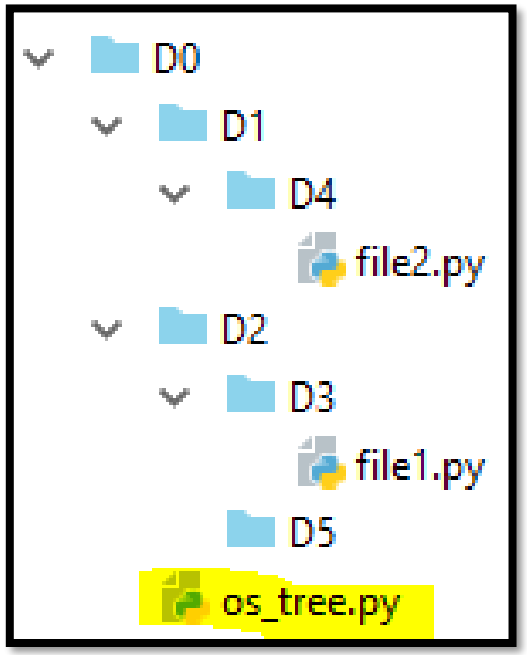
```
from os import scandir, path
from trees_api_mond import *

def path_to_tree(path_name:str)-> Tree:
    """
    Returns a tree representation for filesystem starting from
      path_name
    """
    # print(path_name)
    if  path.isdir(path_name):
        return Tree(path_name,
                    [path_to_tree(path.join(path_name, f.name))
                     for f in scandir(path_name)])
    else:
        return Tree(path_name)
```

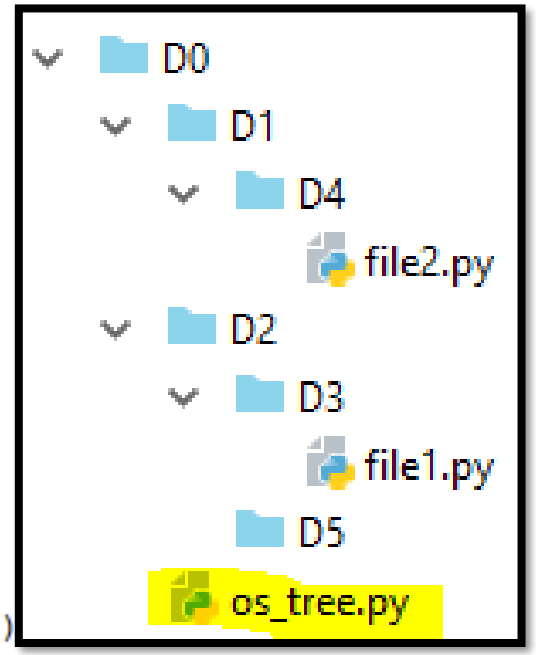
Computer Science
UNIVERSITY OF TORONTO

Example: filesystem using Trees

```
t=path_to_tree(".")
print("------height-------")
print("Height:",t.height())
print("------number for Files in ( folder D0 )-------")
print("Number for Files:",t.leaf_count())
print("------__repr__-------")
print(t.__repr__())
print("------flatten-------")
print(t.flatten())
print("------__str__-------")
print(t.__str__(13))
```



Sample Output




```
 D:/csc148/lectures/week7/D0/os_tree.py
------height-------
Height: 4
------number for Files in ( folder D0 )-------
Number for Files: 4
------__repr__-------
Tree('.', [Tree('.\\D1', [Tree('.\\D1\\D4', [Tree('.\\D1\\D4\\file2.py')])]), Tree('.\\D2',
 [Tree('.\\D2\\D3', [Tree('.\\D2\\D3\\file1.py')]), Tree('.\\D2\\D5')]), Tree('.\\os_tree.py')])
------flatten-------
['.', '.\\D1', '.\\D1\\D4', '.\\D1\\D4\\file2.py', '.\\D2', '.\\D2\\D3', '.\\D2\\D3\\file1.py',
 '.\\D2\\D5', '.\\os_tree.py']
------__str__-------
                .\D2\D5
            .\D2
                    .\D2\D3\file1.py
                .\D2\D3
            .\os_tree.py
        .
                    .\D1\D4\file2.py
                .\D1\D4
            .\D1
```