Tree Traversals

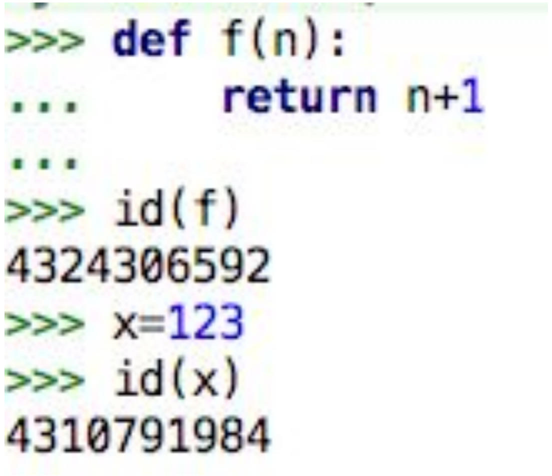
Arnamoy Bhattacharyya

Agenda

1. Passing function as argument of a function
2. Tree traversals
    a. Preorder
    b. Postorder
    c. Level Order

A function is an “object”

A set of *instructions* (code) as its *value*



```
>>> def f(n):
...     return n+1
...
>>> id(f)
4324306592
>>> x=123
>>> id(x)
4310791984
```

Passing a function as an argument

```
def count_if(t: Tree, p: Callable[[object], bool]) -> int:
    """
    Return number of values in Tree t that satisfy predicate p(value).

    Assume predicate p is defined on t's values

    >>> def p(v): return v > 4
    >>> t = descendants_from_list(Tree(0),
                                  [1, 2, 3, 4, 5, 6, 7, 8], 3)
    >>> count_if(t, p)
    4
    >>> def p(v): return v % 2 == 0
    >>> count_if(t, p)
    5
    """
```

Tree Traversal

- So far, ordering did not matter
- Sometimes you do care about the order of traversal
  - In Minimax sometimes the order you visit the nodes may give rise to different solutions
  - More examples End of the Day

Putting order in the traversal of a Tree

1. Preorder
    a. *Act* on the current node
    b. In a preorder fashion, visit its children (and act on them)
    c. Act is a function act() that does some action on the node
        i. e.g printing the value of node
        ii. comparing the value of node with something

Putting order in the traversal of a Tree

2. Postorder

    a. In a postorder fashion, visit its children (and act on them)
    b. *Act* on the current node
    c. Act is a function act() that does some action on the node
        i. e.g printing the value of node
        ii. comparing the value of node with something

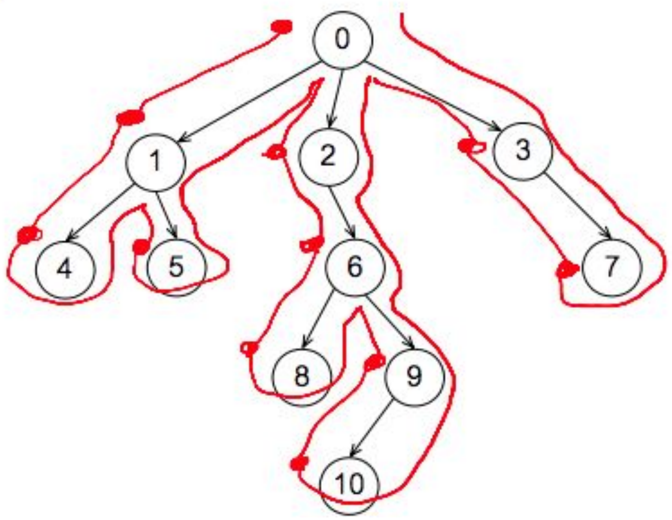
Putting order in the traversal of a Tree

2. Levelorder
    a. Visit every node and act() at a particular level
    b. Keep doing it until no more levels

Exercise (find the preorder, post order and level order):

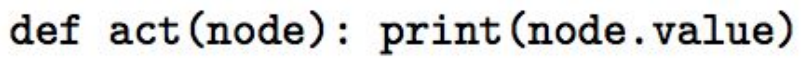
```
def act(node): print(node.value)
```

- ...

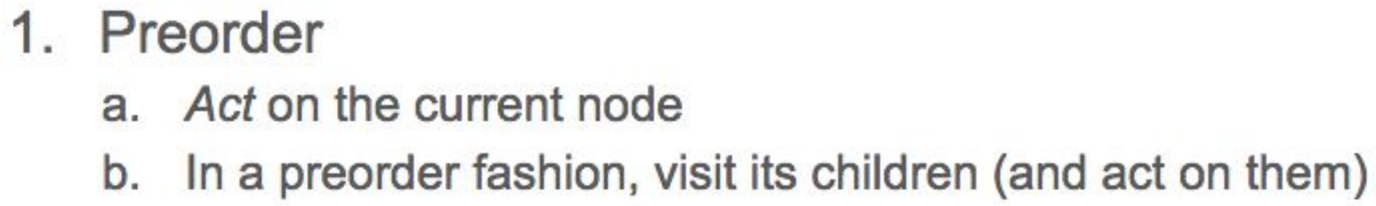


Implementation preorder

Advice: DO NOT use comprehension with code that has side effects (print)



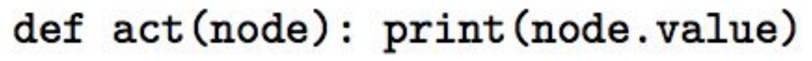
```
def act(node): print(node.value)
```



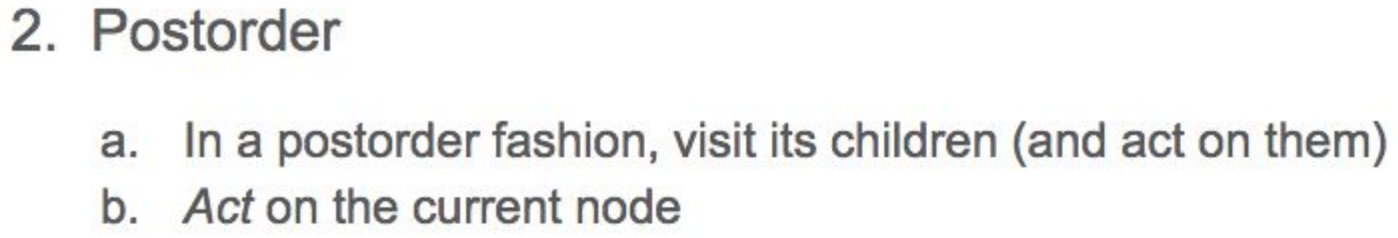
1. Preorder
   a. *Act* on the current node
   b. In a preorder fashion, visit its children (and act on them)

Implementation postorder

Advice: DO NOT use comprehension with code that has side effects (print)



```python
def act(node): print(node.value)
```



2. Postorder
   a. In a postorder fashion, visit its children (and act on them)
   b. *Act* on the current node

Implementation levelorder

There are two possible implementation

1. Use a queue() and no recursion
2. Use a recursive version which tracks the level

Why these orders?

Example postorder:

Deleting nodes from a tree: act on (delete) children first

Example level order:

In a game state, you want to know what are other states close to you