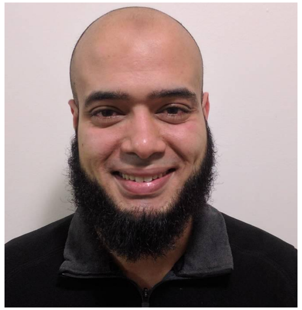
Data Structures

Binary Tree Serialization

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

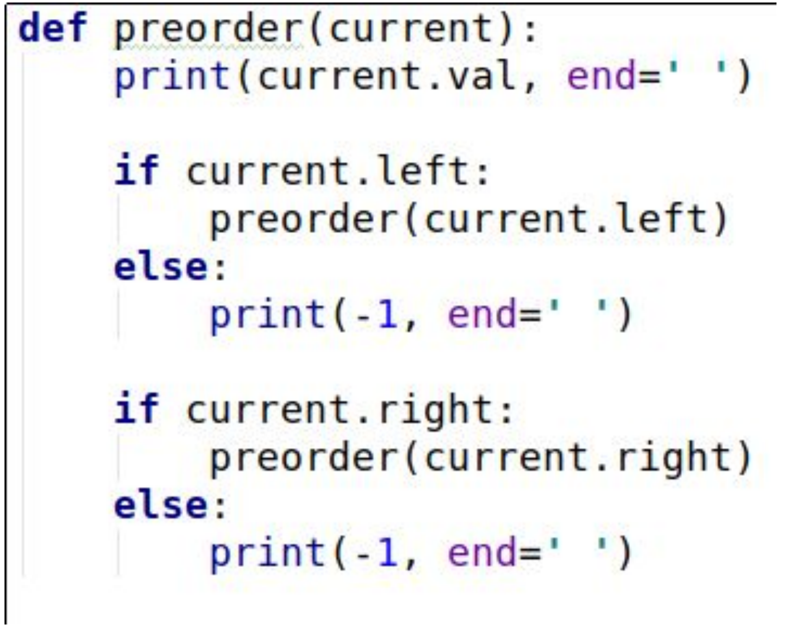


A unique tree representation

- We've learned how 2 representations can sometimes allow us to reconstruct a tree:
  - E.g. Inorder + preorder   or   Inorder + postorder   or   Inorder + level-order
- But this means we need 2 arrays to represent a tree!
- Why isn't one representation enough?
  - Because we don't know if these values are for left or right subtrees!
  - In other words, nothing clearly indicates the null subtrees!
- To have one unique representation, simply change it to indicate the null trees in a more explicit way!
- Try implementing: void print_preorder_complete()
  - Its preorder representation is uniquely a tree
  - Assume all tree values are >= 0

Full information preorder

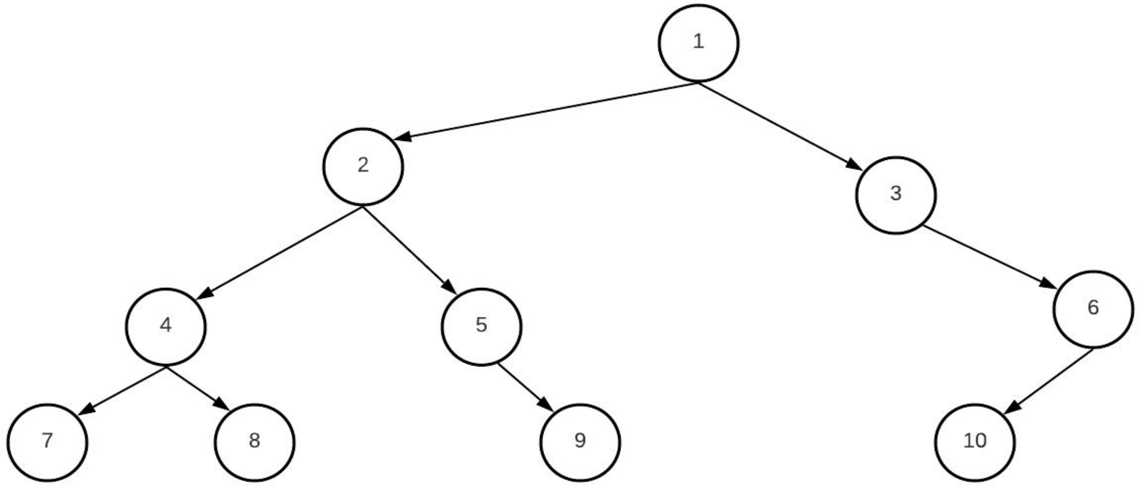
- Put simply: when we have a null node, print something that indicates this!
- e.g. -1, assuming that there is no -1 value held in the tree
- If this is the output, can you build the tree?
  1 2 4 7 -1 -1 8 -1 -1 5 -1 9 -1 -1 3 -1 6 10 -1 -1 -1



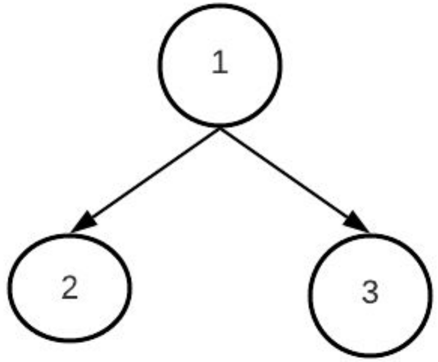
```python
def preorder(current):
    print(current.val, end=' ')

    if current.left:
        preorder(current.left)
    else:
        print(-1, end=' ')

    if current.right:
        preorder(current.right)
    else:
        print(-1, end=' ')
```

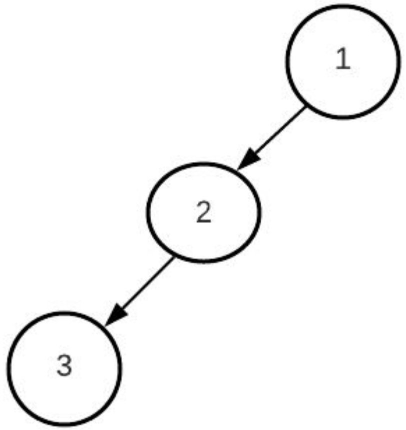
Full information preorder

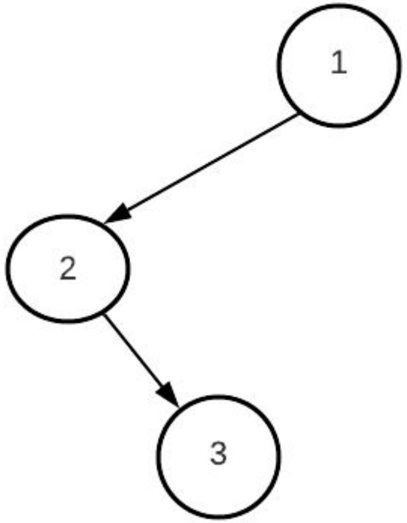
- 1 2 4 7 -1 -1 8 -1 -1 5 -1 9 -1 -1 3 -1 6 10 -1 -1 -1
    - We can also use None instead of -1



Full information preorder







1 2 **-1** **-1** 3 **-1** **-1**

1 2 3 **-1** **-1** -1 -1

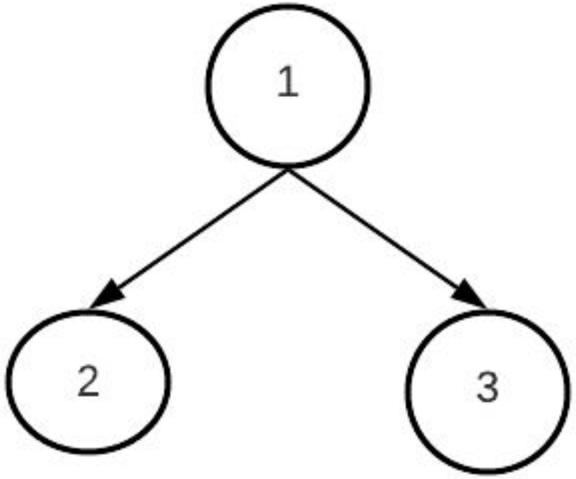
1 2 -1 3 -1 -1 -1

Seralization

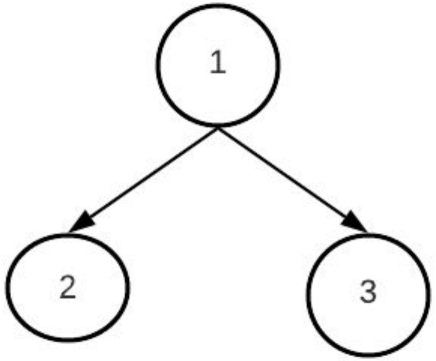
- Serialization is the process of converting a data structure to a representation that can easily be stored somewhere, for example, in a file
- We learned how to get a uniquely representative preorder representation
- Another interesting representation is to parenthesize the tree!
- Each tree representation is
  - (
  - Left sub-tree representation
  - Right sub-tree representation
  -)
- Then a **None child** is represented as ()

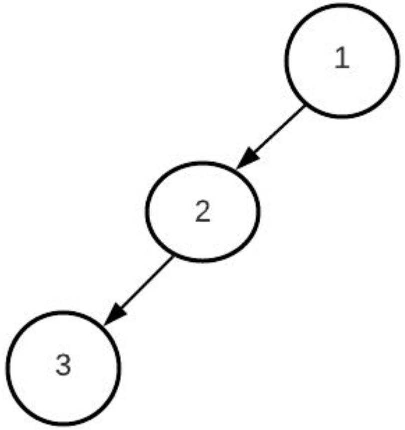
Parenthesizing a tree

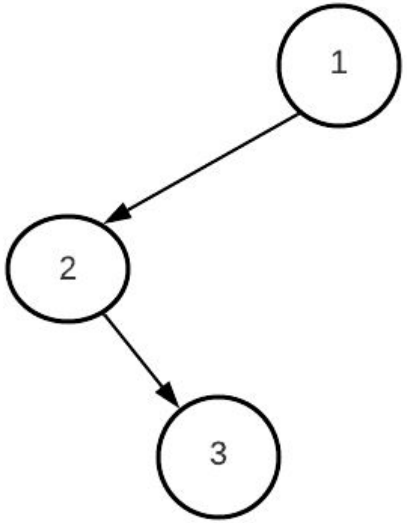
- Node 2 representation:
  - (2()())
- Node 3 representation:
  - (3()())
- Node 1 representation:
  - (1LR)
  - (1   (2()())   (3()())   )



Parenthesizing a tree







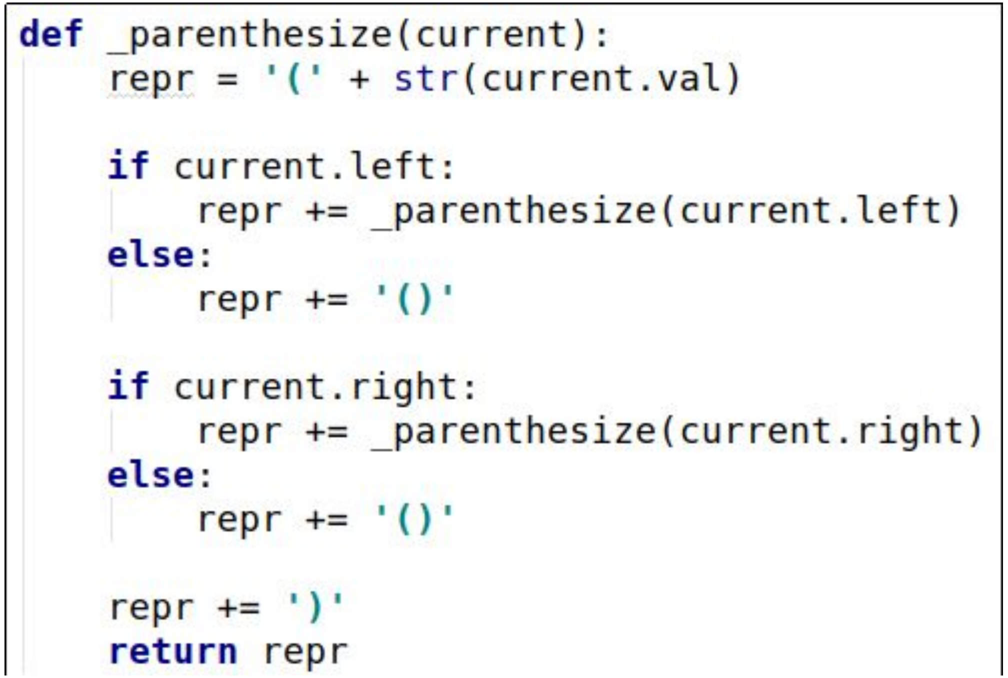
(1(**2()()**)(**3()()**))

(1(2(3()())())())

(1(2()(3**()()**))())

Parenthesizing a tree

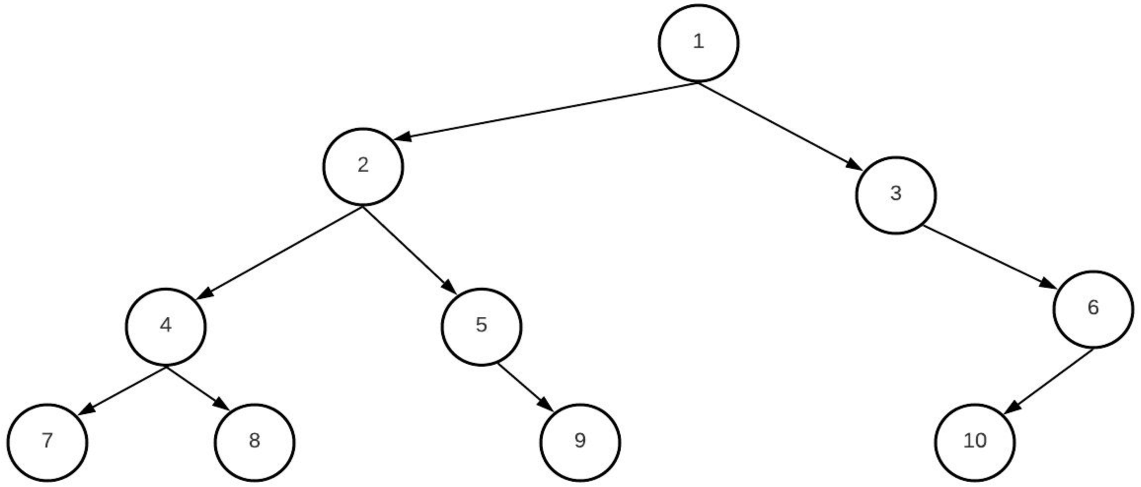
- Note, as += creates usually new memory, this additions making the code quadratic time



```python
def _parenthesize(current):
    repr = '(' + str(current.val)

    if current.left:
        repr += _parenthesize(current.left)
    else:
        repr += '()'

    if current.right:
        repr += _parenthesize(current.right)
    else:
        repr += '()'

    repr += ')'
    return repr
```

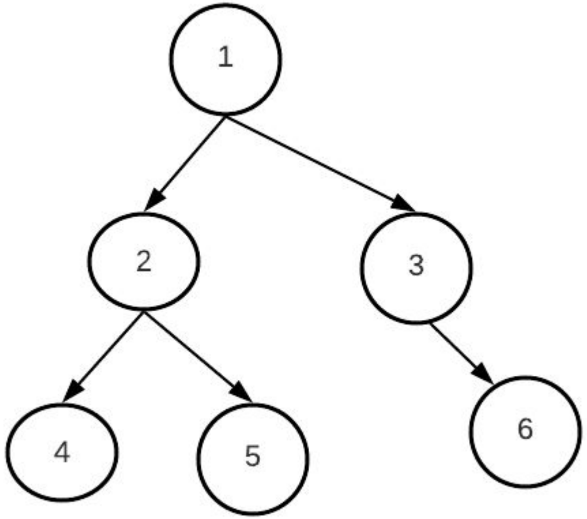
Parenthesizing a tree

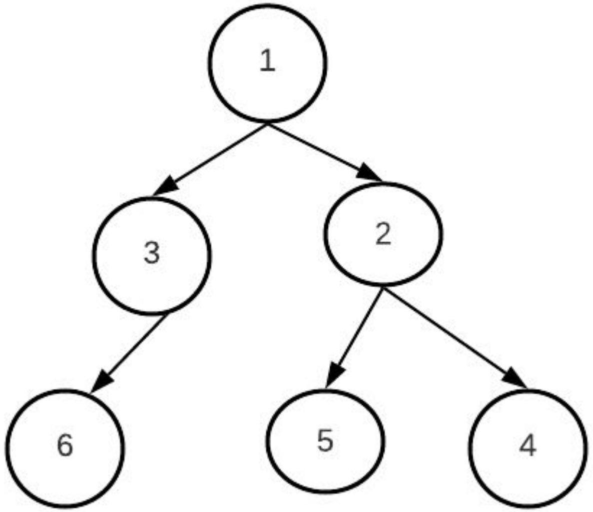
- (1(2(4(7()())(8()()))(5()(**9()()**)))(3()(6(**10()()**)())))

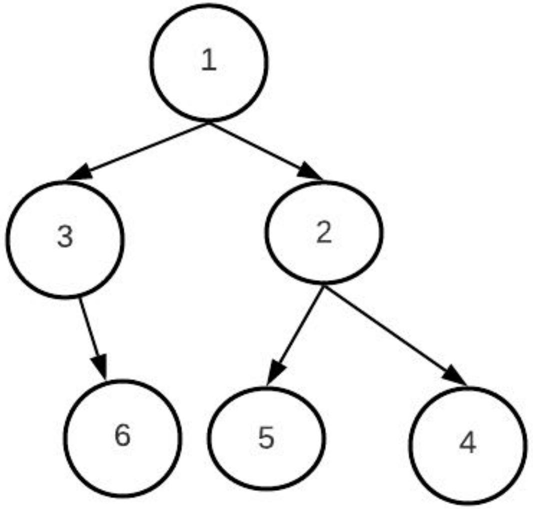


Canonicalizing a tree

- If you have several arrays: how can you check if they have the same values?
  - Sort each array and compare them.
- Below are very similar trees: how can we **sort a tree**? E.g. for comparisons
  - *Each subtree must still contain its old children with their relationships!*

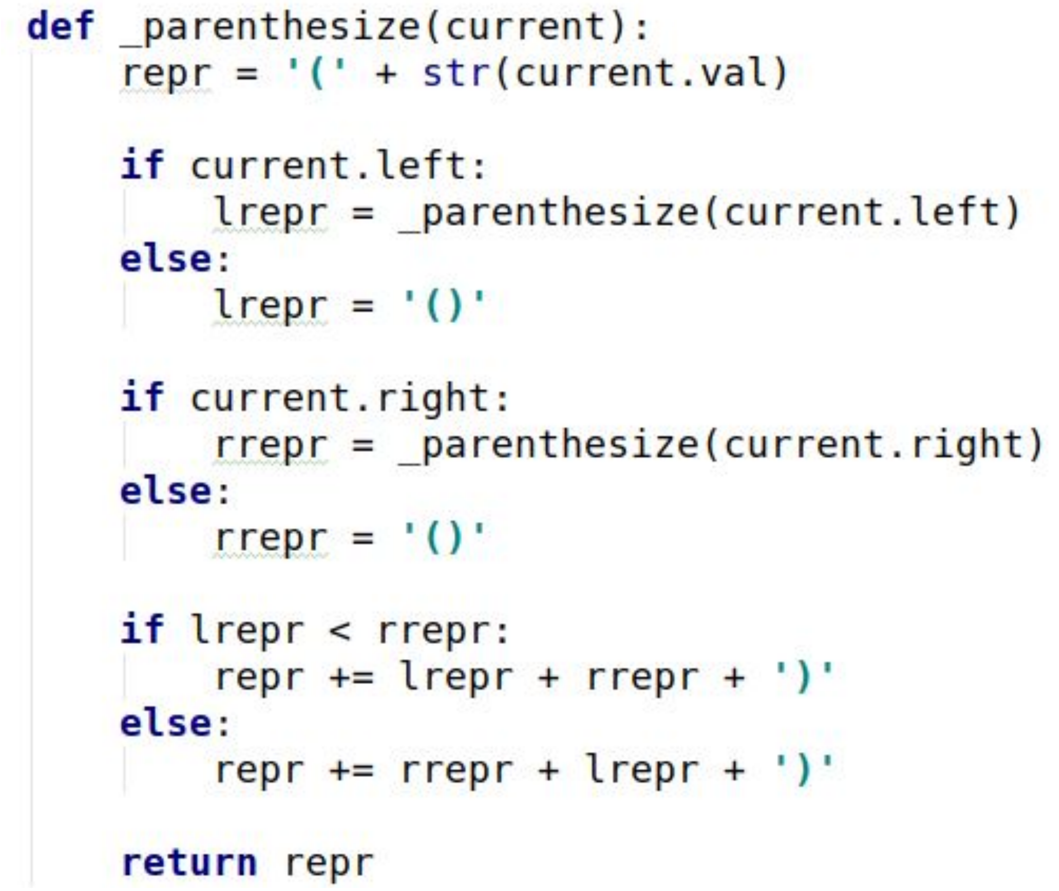






Canonicalizing a tree

- The core idea is simple
- Just build left and right representations and add the smaller one first
    - Notice: this is str comparison
- Notice: If values are unique, then we can compare left and right values



```python
def _parenthesize(current):
    repr = '(' + str(current.val)

    if current.left:
        lrepr = _parenthesize(current.left)
    else:
        lrepr = '()'

    if current.right:
        rrepr = _parenthesize(current.right)
    else:
        rrepr = '()'

    if lrepr < rrepr:
        repr += lrepr + rrepr + ')'
    else:
        repr += rrepr + lrepr + ')'

    return repr
```

Tackling Problems

- There are many problems that depend on tree unique representation
    - Check if 2 trees are identical
    - Check if 2 trees are mirrors
    - Check if a tree has duplicate subtrees
    - Check if a tree is a subtree of another
    - Find the largest identical 2 subtrees of a tree
- We might able to:
    - Develop a recursive technique that tries to answer them
    - Serialize each (sub)tree and easily compare them
        - This is usually less buggy and less thinking!

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”