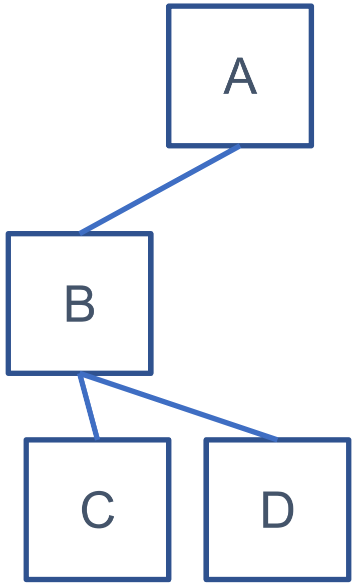
Review

- Tree
- Rooted tree
- Rooted binary tree
- Binary search tree



Computing Bootcamp

Trees

Lecture 17

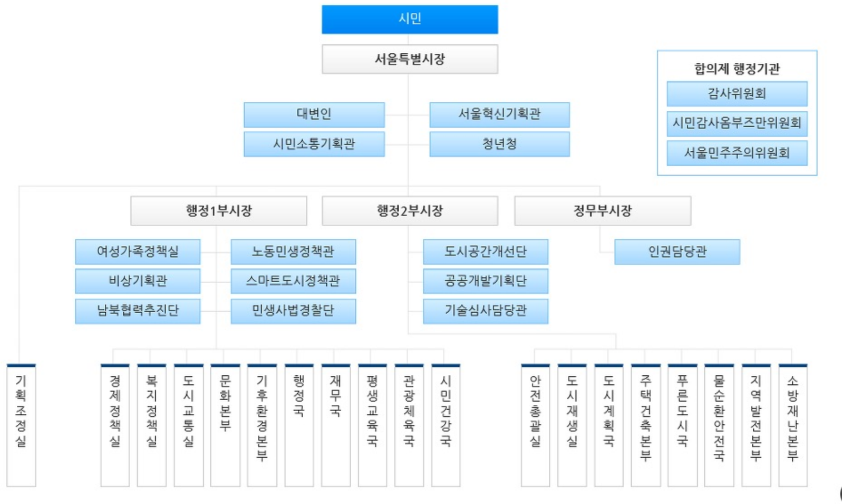
Hyung-Sin Kim

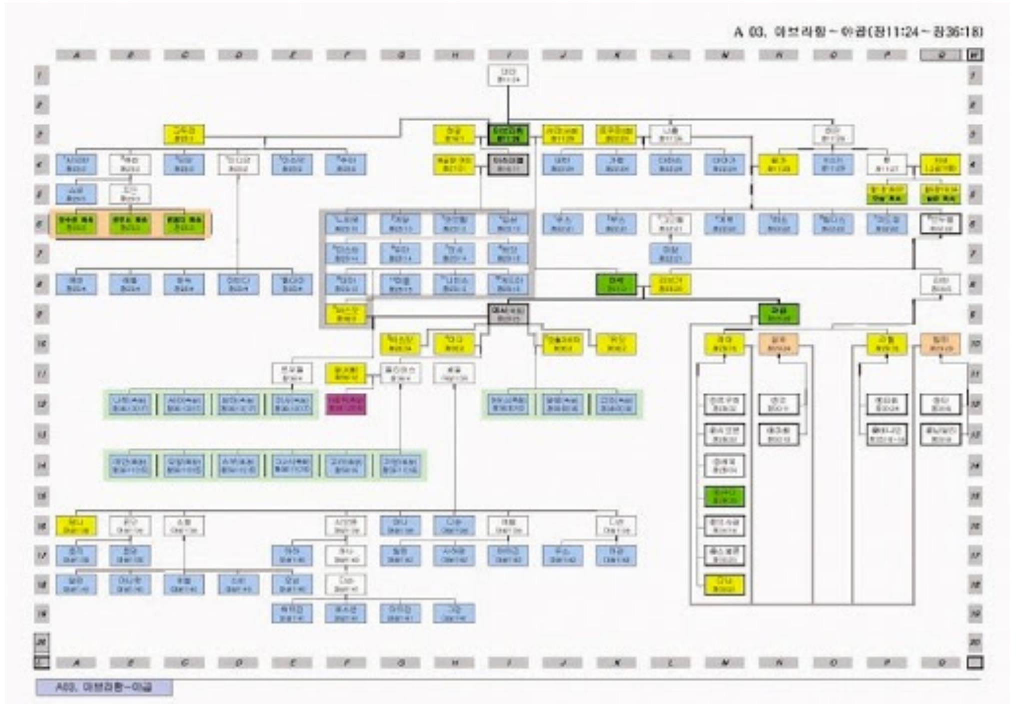
SNU Graduate School of Data Science

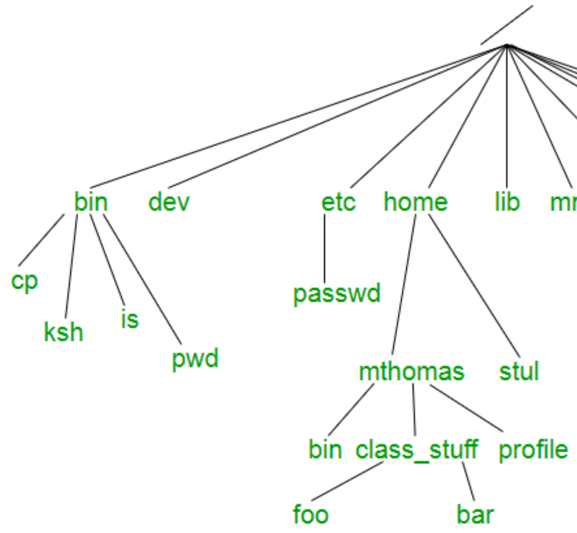
Contents

Trees are Everywhere

- Organization chart
- Genealogy (family tree)
- File system







K-ary Trees

- A general tree node does not have to have only two children nodes

- A tree that allows each node to have up to k children nodes is called **k-ary tree**

```
class TreeNode():
    def __init__(self, x: int, k: int) -> None:
        self.val = x
        self.arity = k
        self.child = [None]*k
```

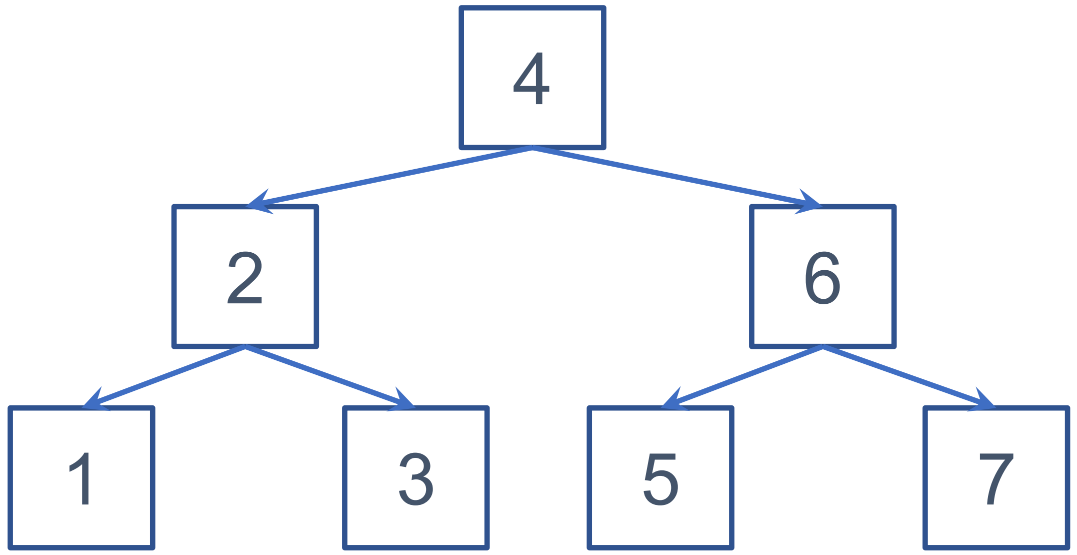
How to navigate the whole tree conveniently?

Breadth-First Traversal

Computing Bootcamp

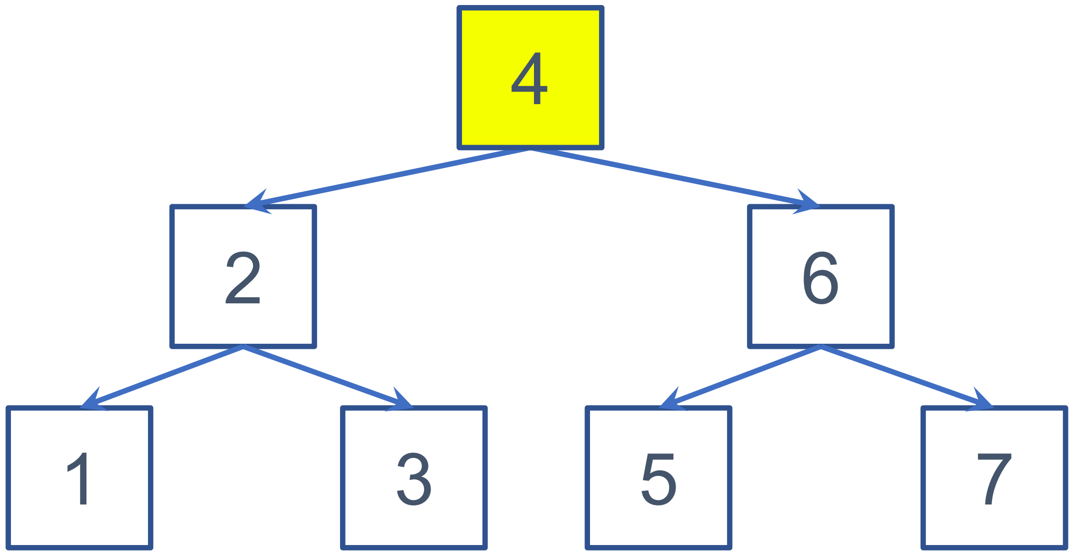
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom



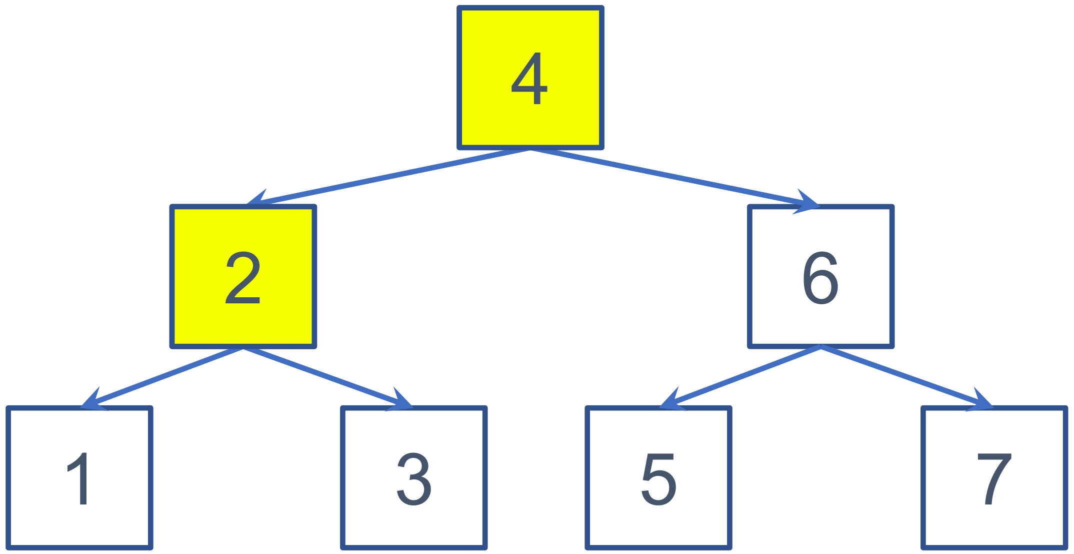
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom



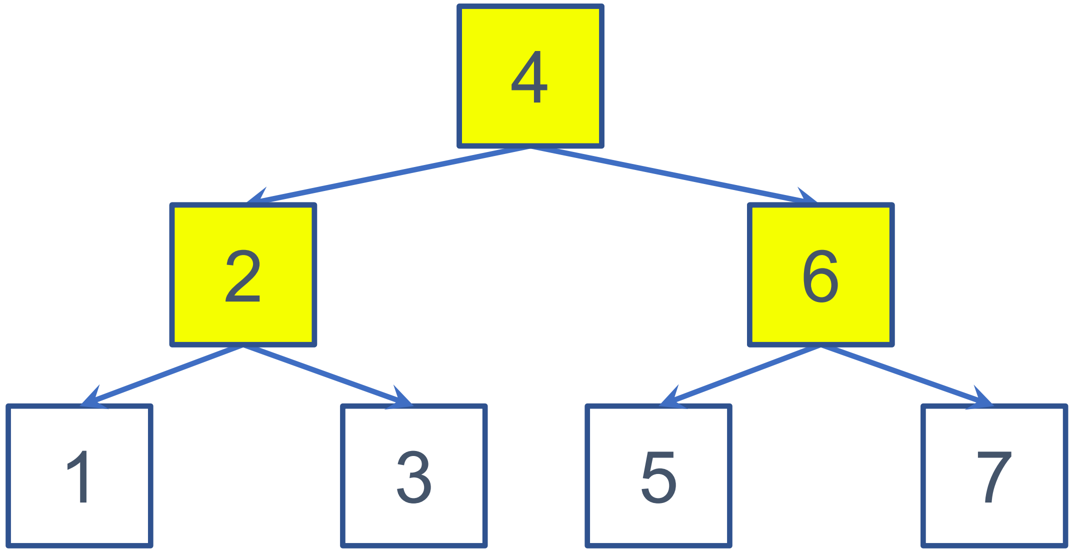
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom



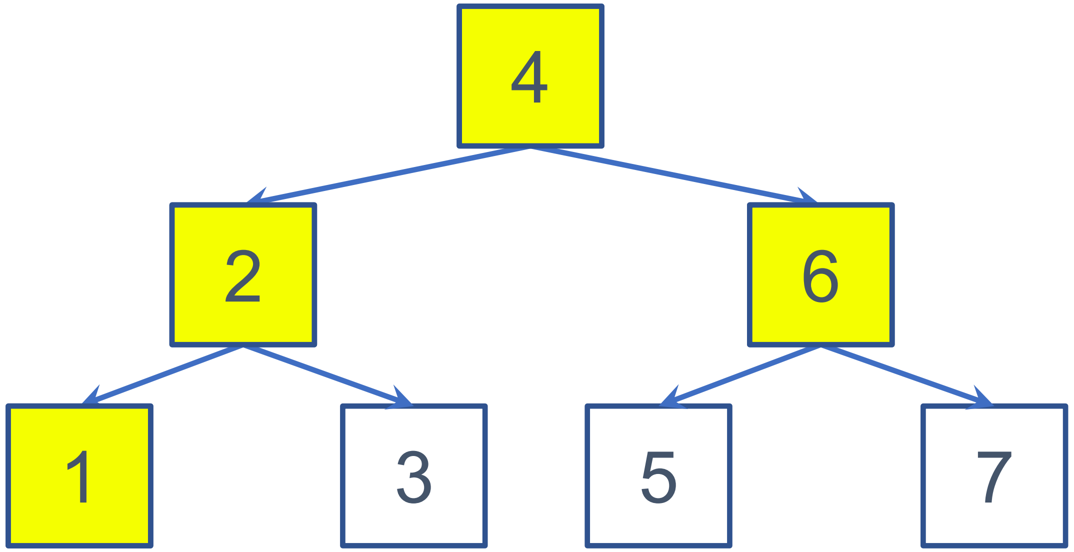
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom



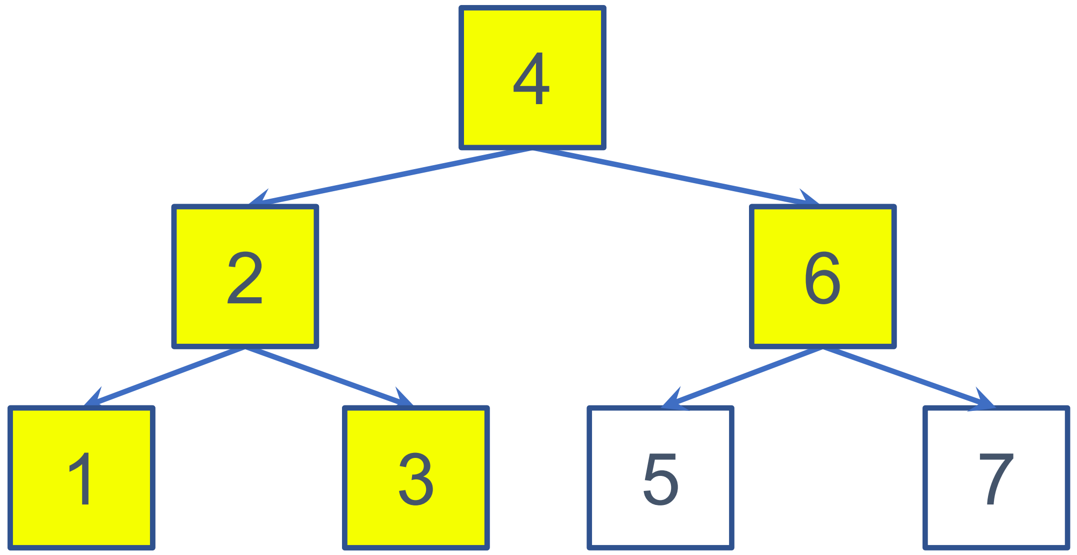
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom



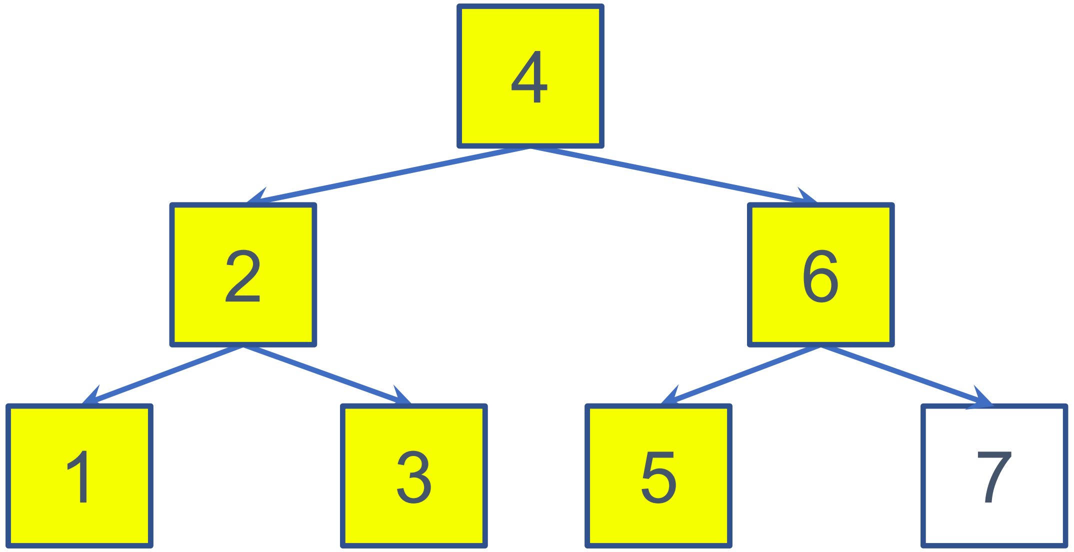
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom



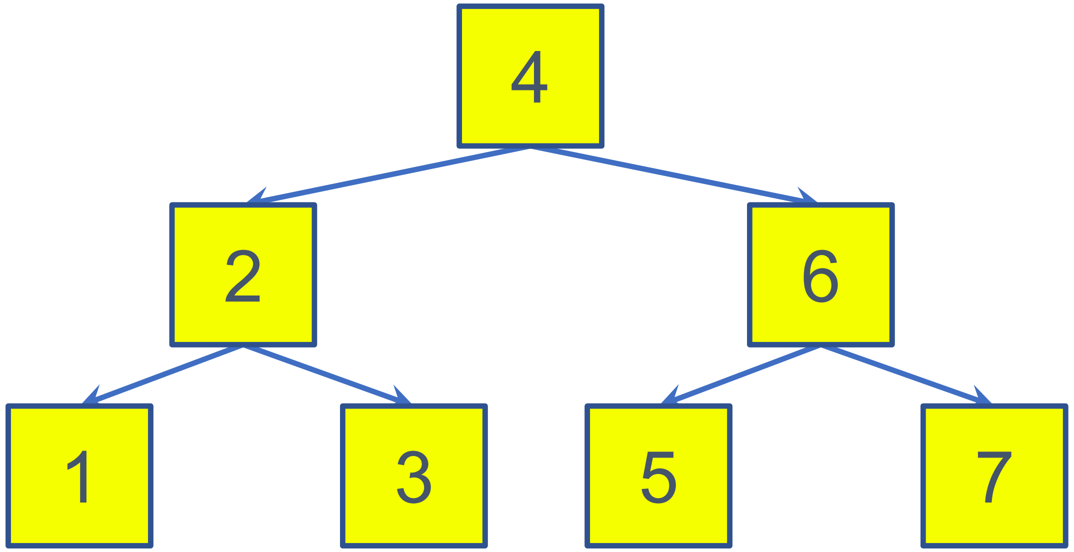
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom



Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom

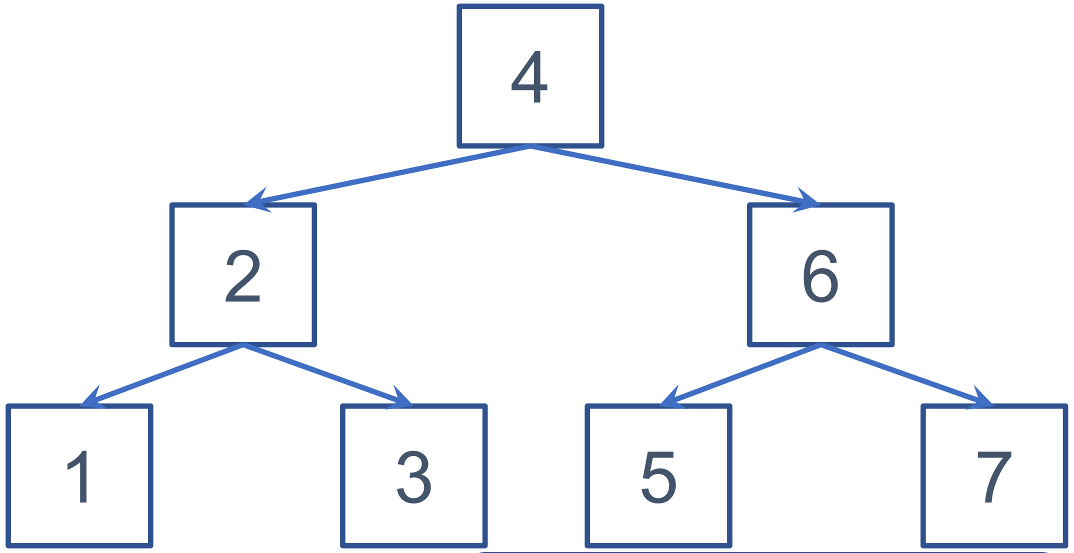


Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = [self.root]
        while q:
            curNode = q.pop(0)
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```



Out:

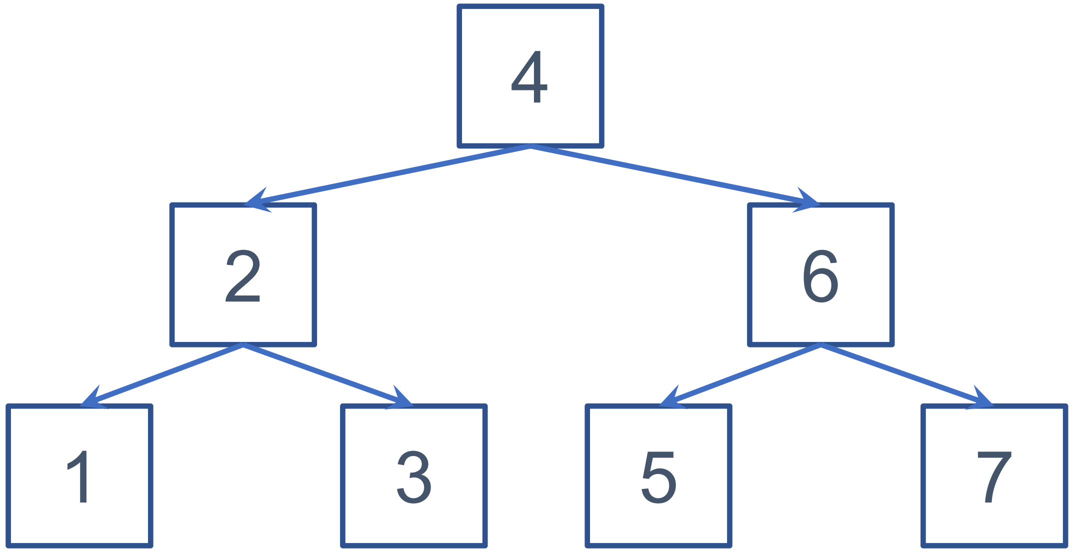
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = [self.root]
        while q:
            curNode = q.pop(0)
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```

q = [T(4)]



Out:

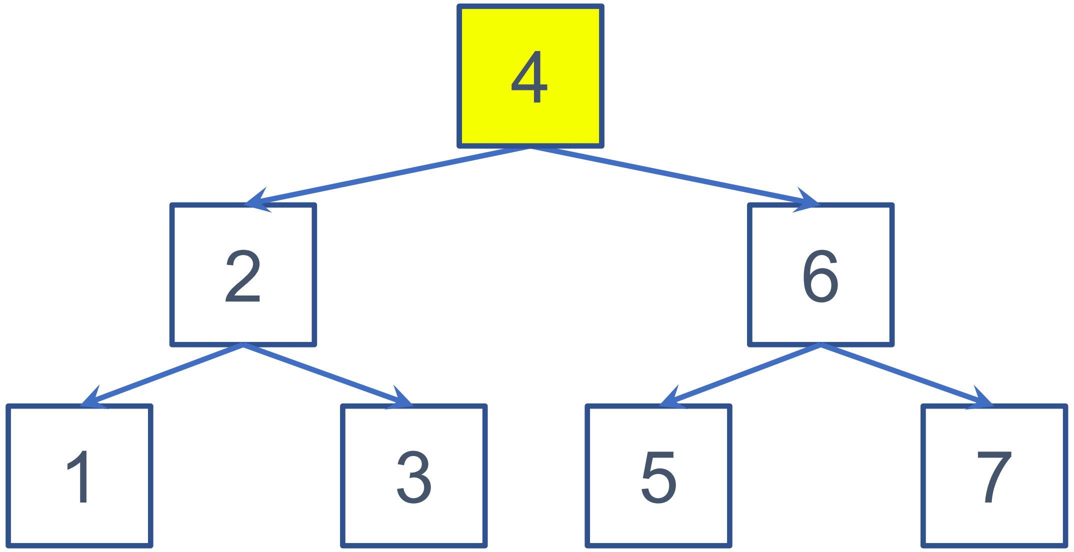
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = [self.root]
        while q:
            curNode = q.pop(0)
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```

q = []



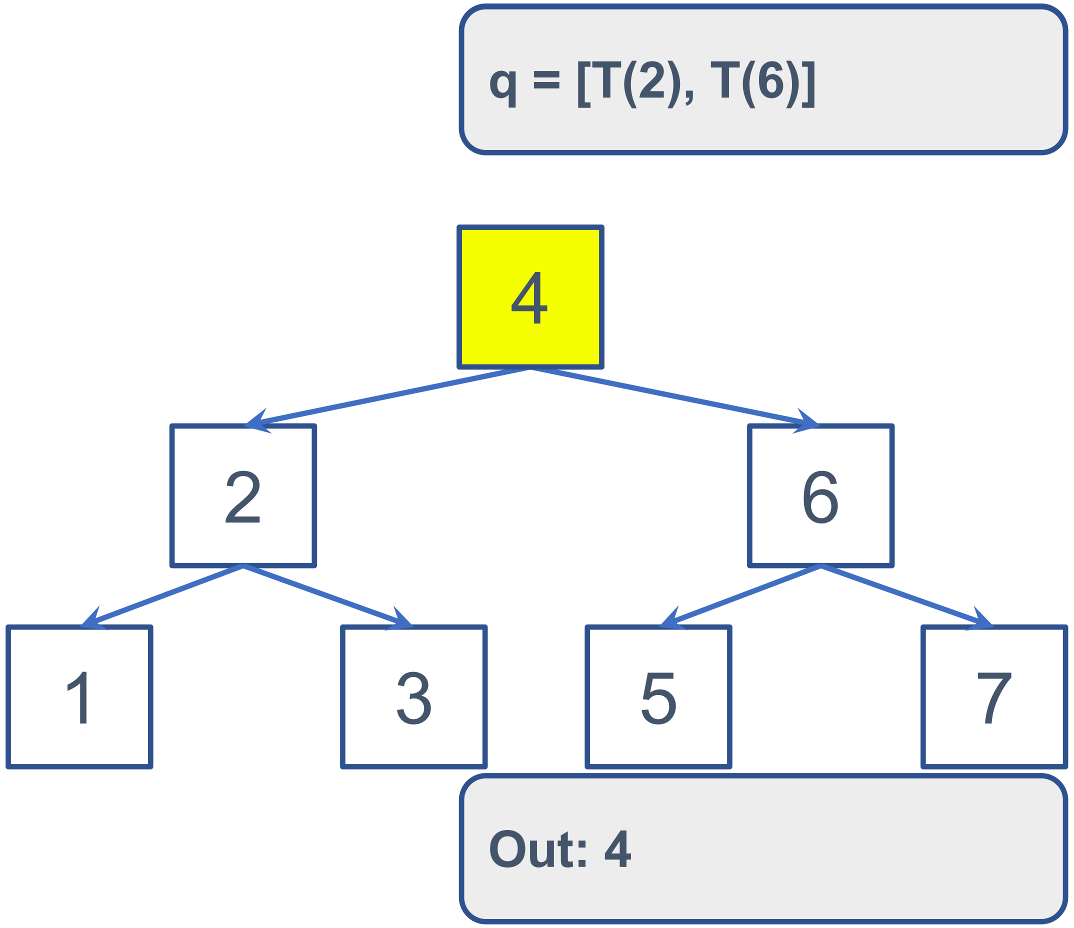
Out: 4

Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = [self.root]
        while q:
            curNode = q.pop(0)
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```



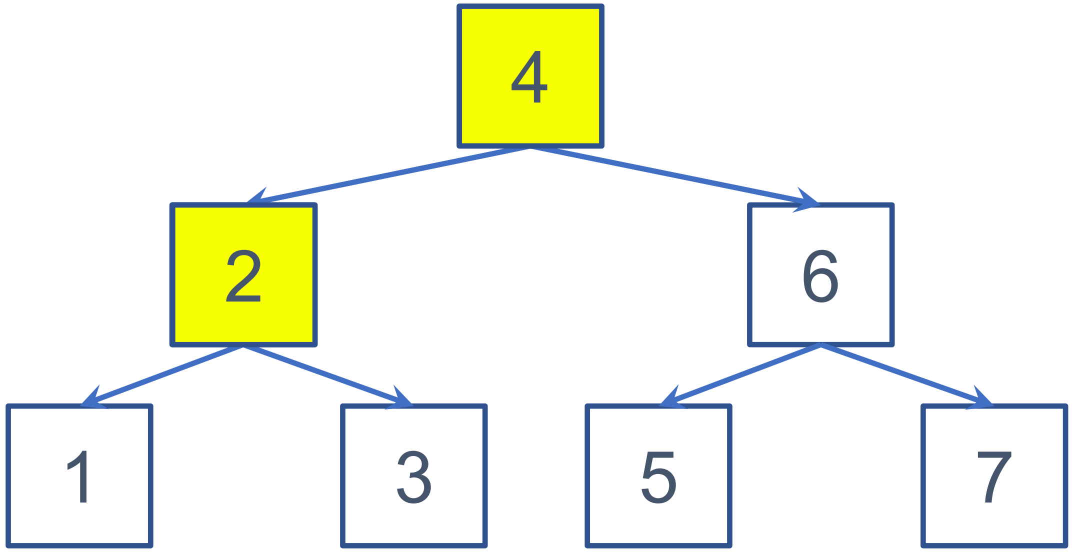
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = [self.root]
        while q:
            curNode = q.pop(0)
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```

q = [T(6)]



Out: 4 2

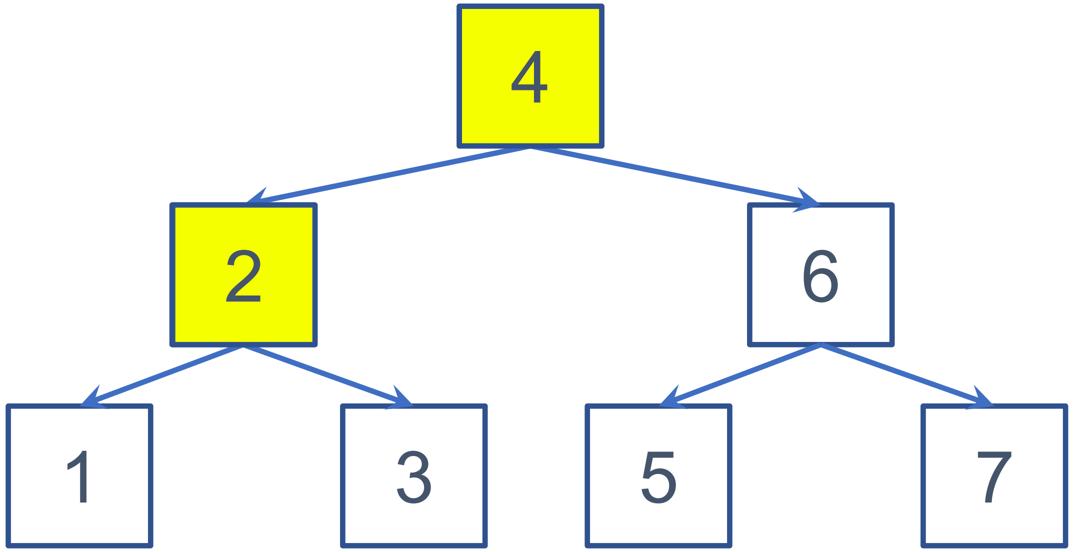
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = [self.root]
        while q:
            curNode = q.pop(0)
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```

q = [T(6), T(1), T(3)]



Out: 4 2

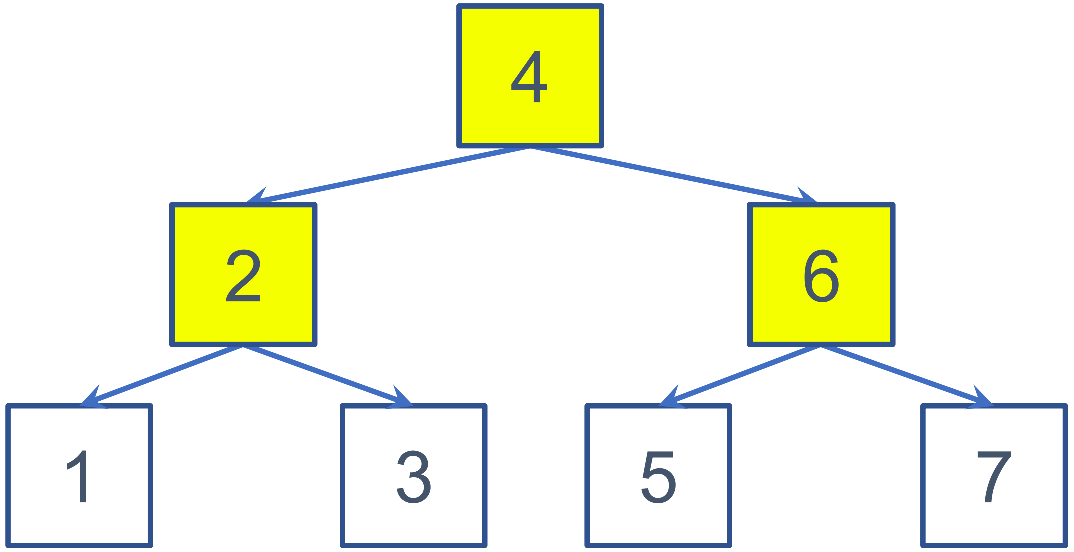
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = [self.root]
        while q:
            curNode = q.pop(0)
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```

q = [T(1), T(3)]



Out: 4 2 6

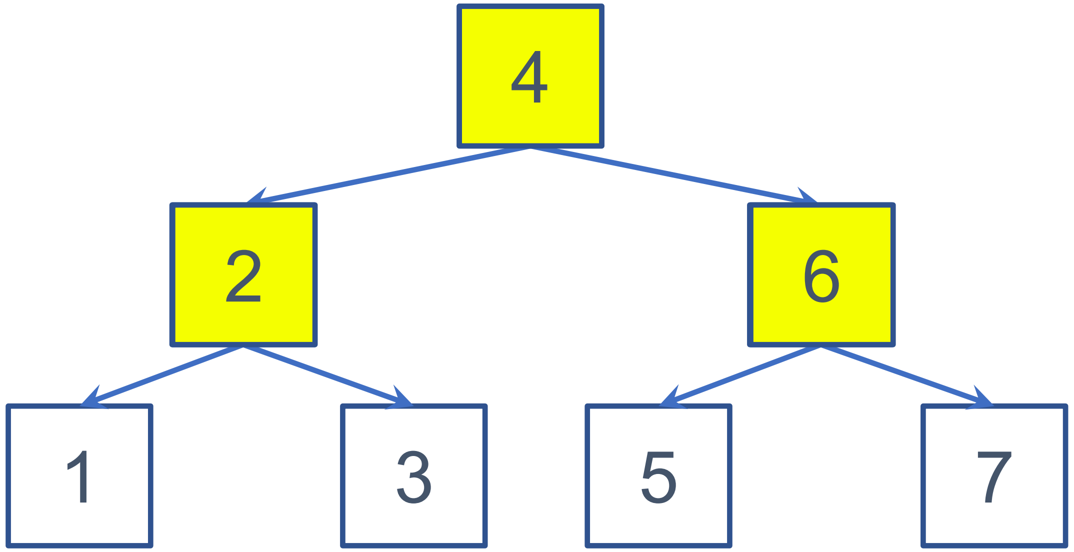
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = [self.root]
        while q:
            curNode = q.pop(0)
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```

q = [T(1), T(3), T(5), T(7)]



Out: 4 2 6

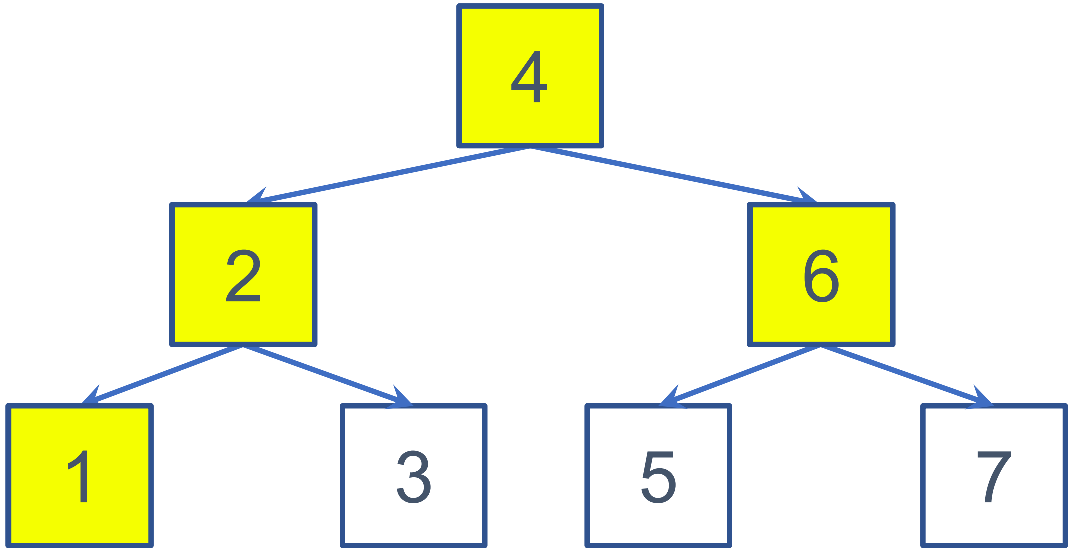
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = [self.root]
        while q:
            curNode = q.pop(0)
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```

q = [T(3), T(5), T(7)]



Out: 4 2 6 1

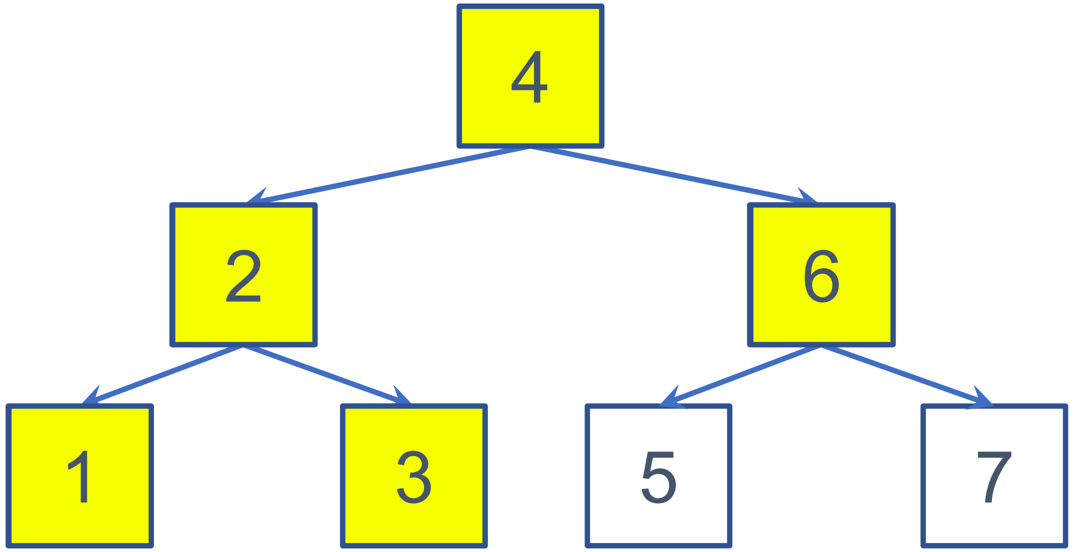
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = [self.root]
        while q:
            curNode = q.pop(0)
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```

q = [T(5), T(7)]



Out: 4 2 6 1 3

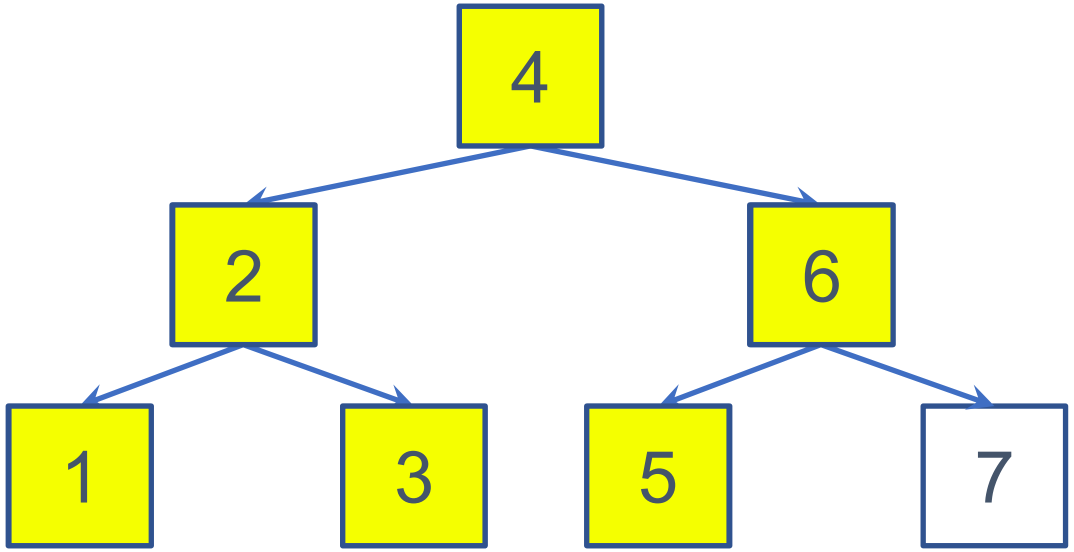
Level-order (Breadth-First) Traversal

- Visit nodes from left to right, and from top to bottom

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = [self.root]
        while q:
            curNode = q.pop(0)
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```

q = [T(7)]



Out: 4 2 6 1 3 5

Level-order (Breadth-First) Traversal

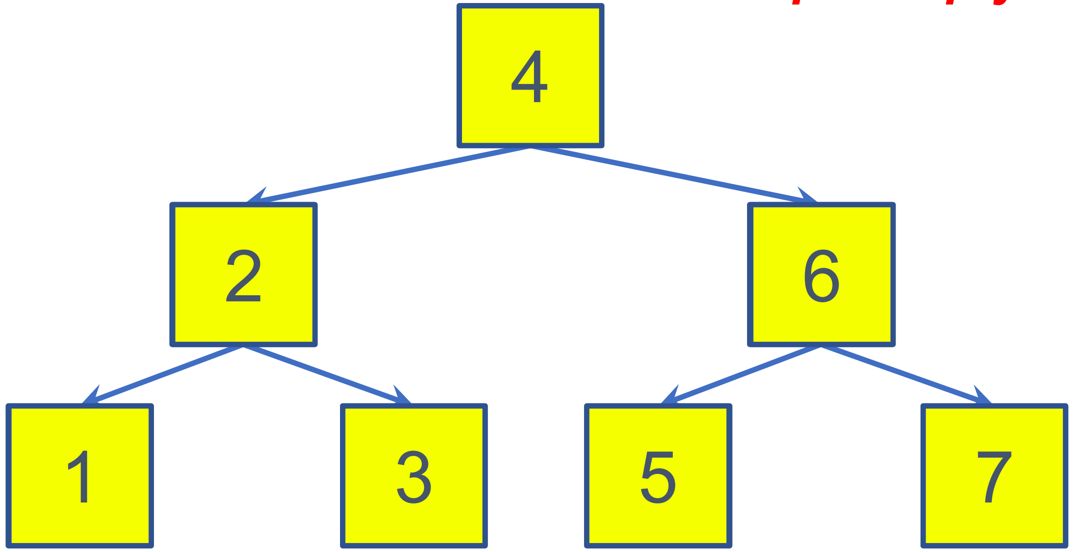
- Visit nodes from left to right, and from top to bottom

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = [self.root]
        while q:
            curNode = q.pop(0)
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```

q = []

Now q is empty!



Out: 4 2 6 1 3 5 7

Level-order (Breadth-First) Traversal – Deque

- Visit nodes from left to right, and from top to bottom

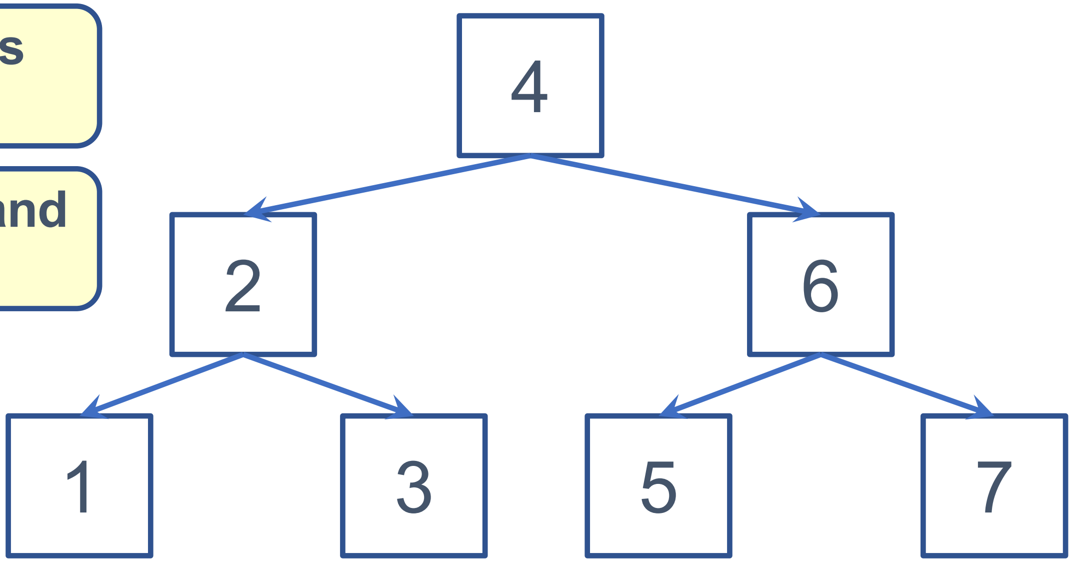
```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def BFT(self):
        if self.root == None:
            return
        q = deque([self.root])
        while q:
            curNode = q.popleft()
            self.visit(curNode)
            for childNode in curNode.child:
                if childNode:
                    q.append(childNode)
```

Doubly-linked list that provides
- append(x), appendleft(x),
- pop(), popleft()

from **collections** import **deque**

Faster pushing and popping!



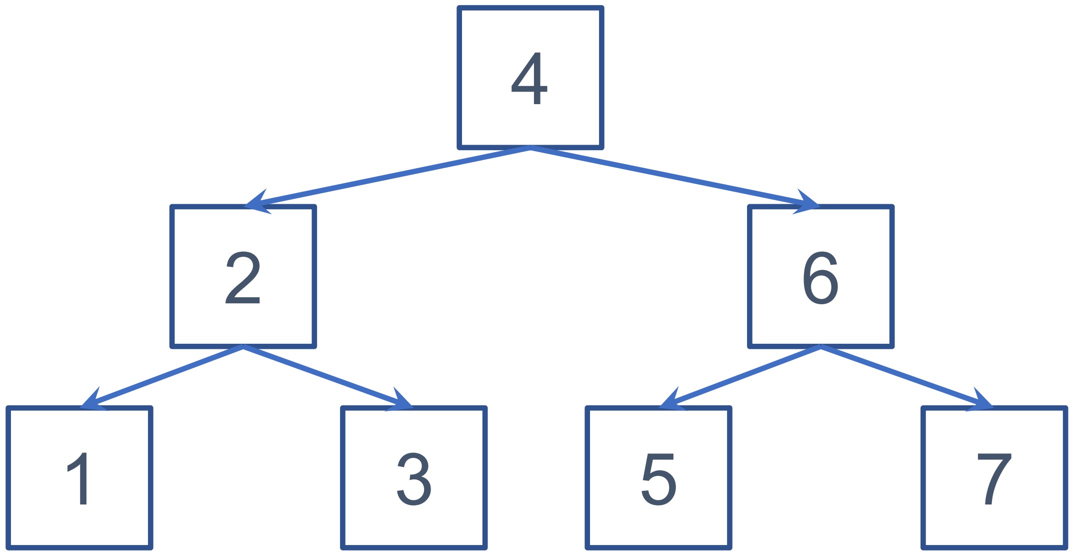
Depth-First Traversal

Depth-First Traversal

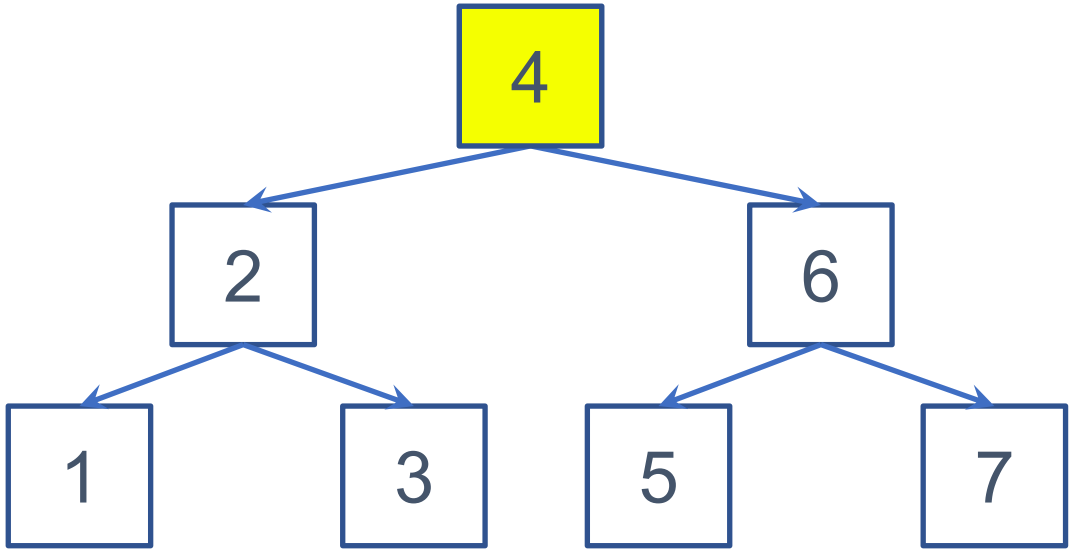
- **Depth-First Traversal**
- **Preorder**
- **Inorder**
- **Postorder**

Computing Bootcamp

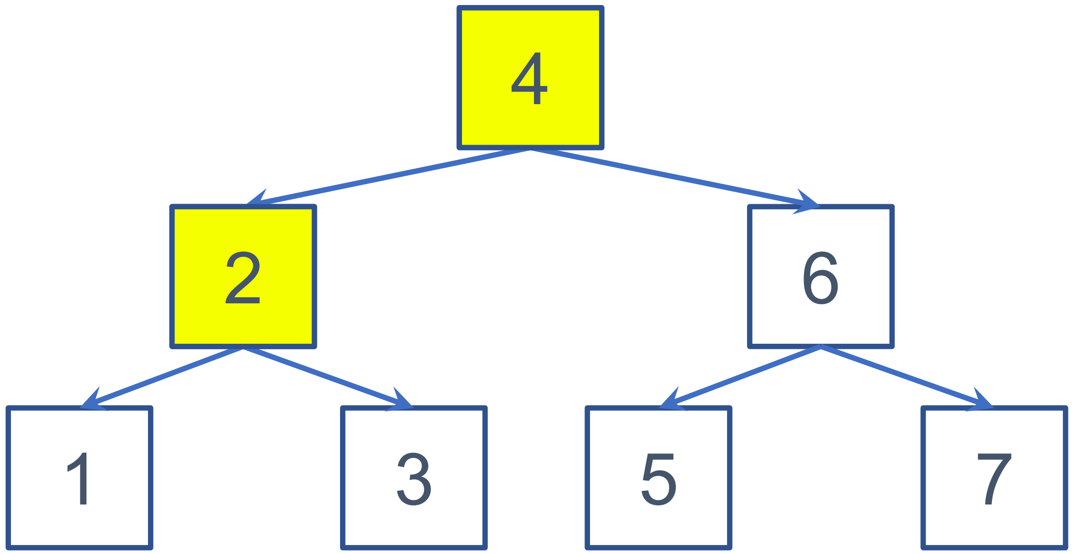
Depth First Traversals



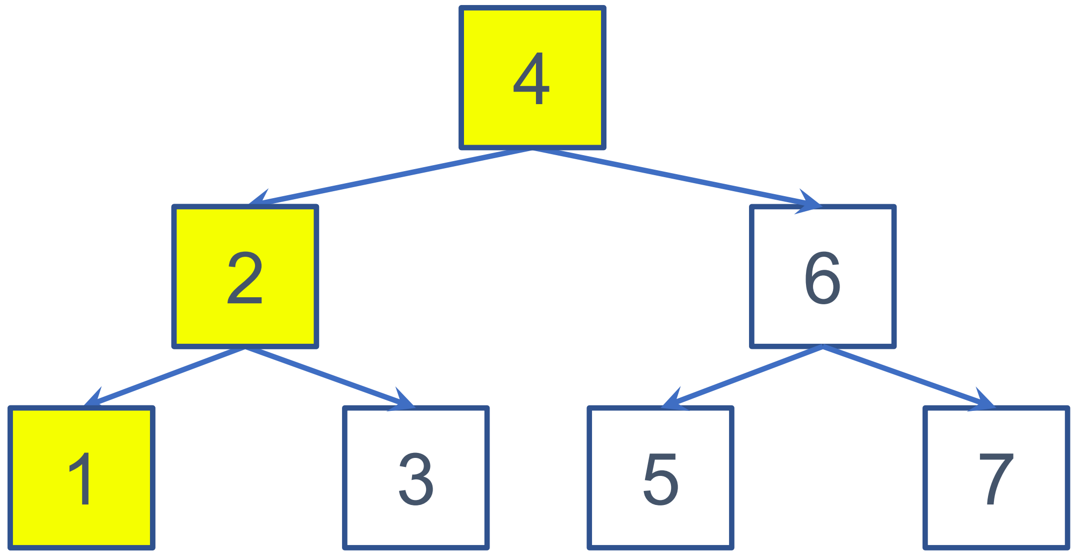
Depth First Traversals



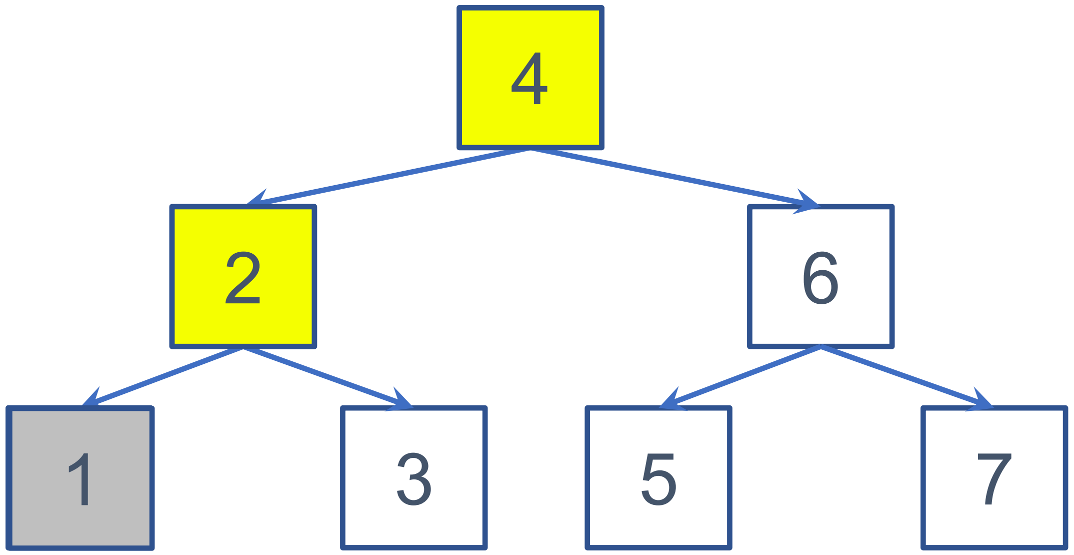
Depth First Traversals



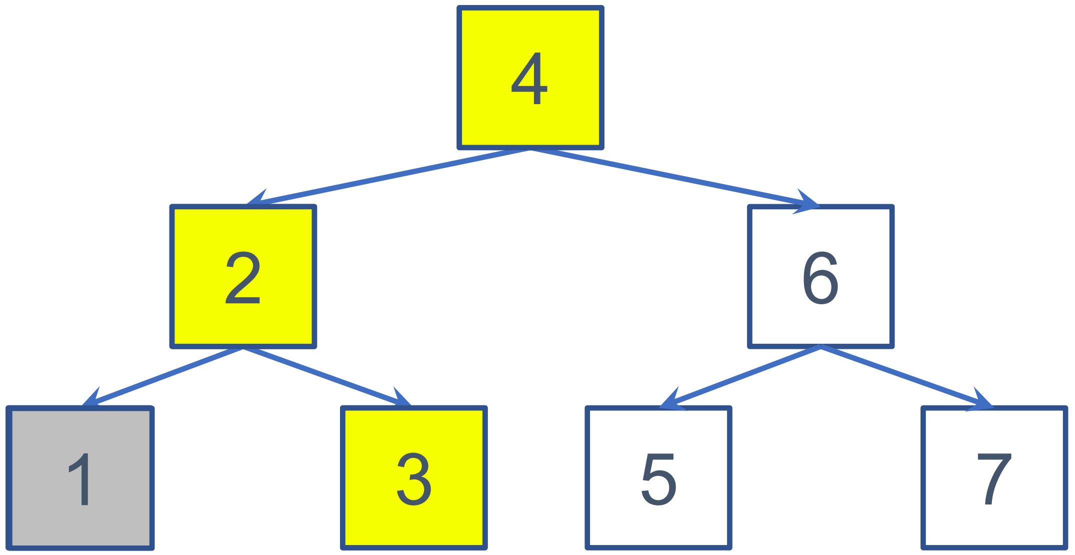
Depth First Traversals



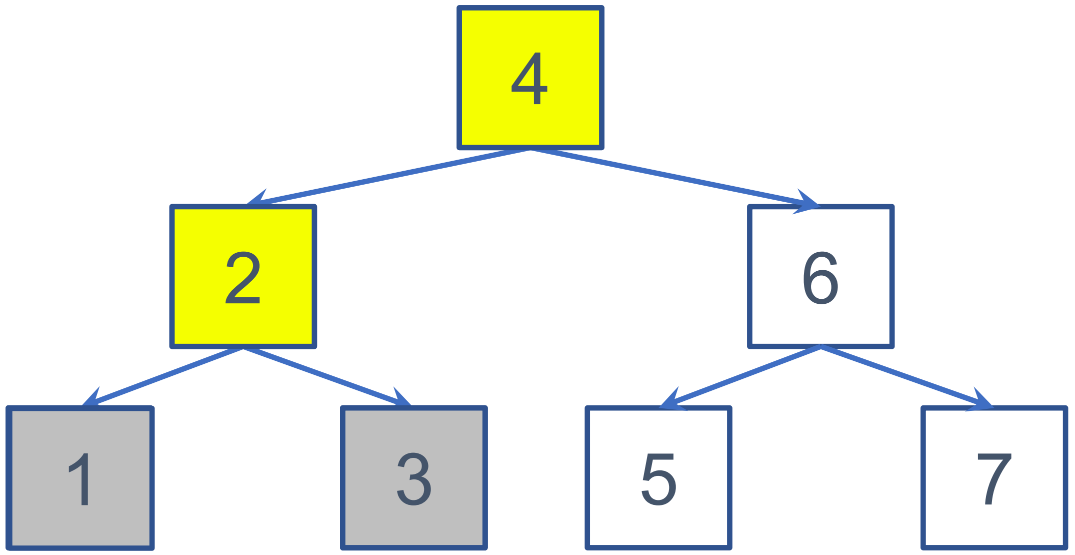
Depth First Traversals



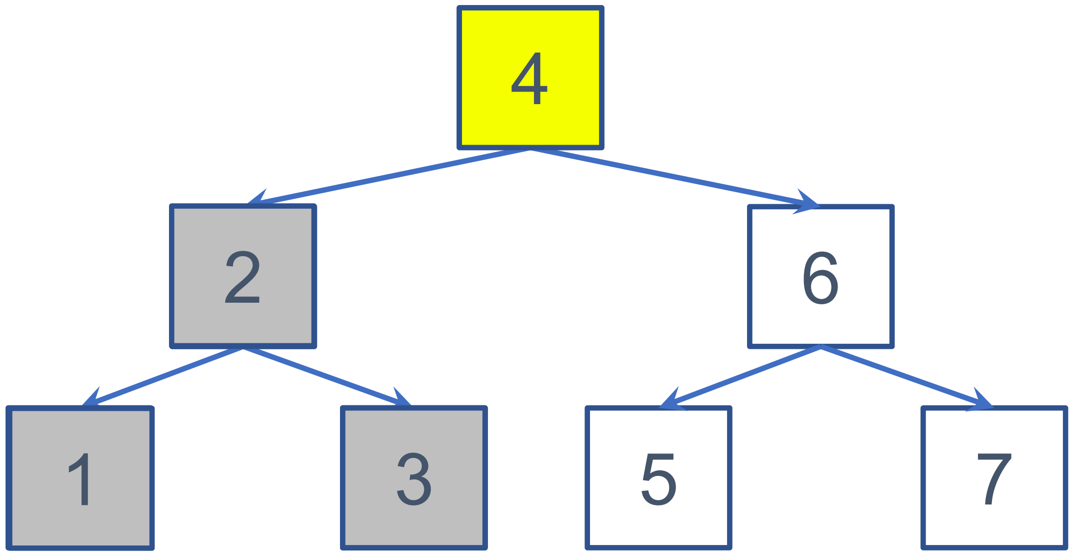
Depth First Traversals



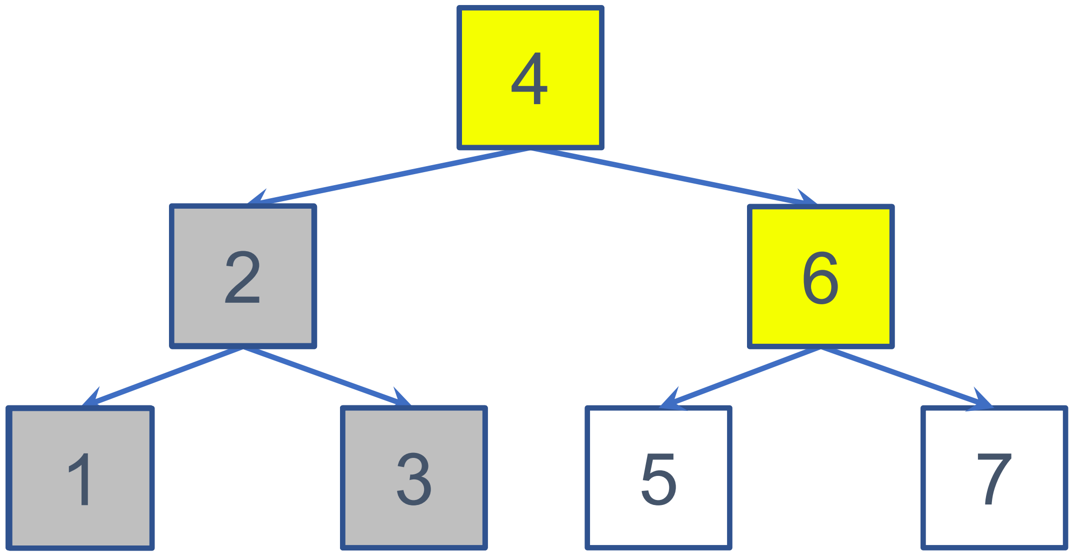
Depth First Traversals



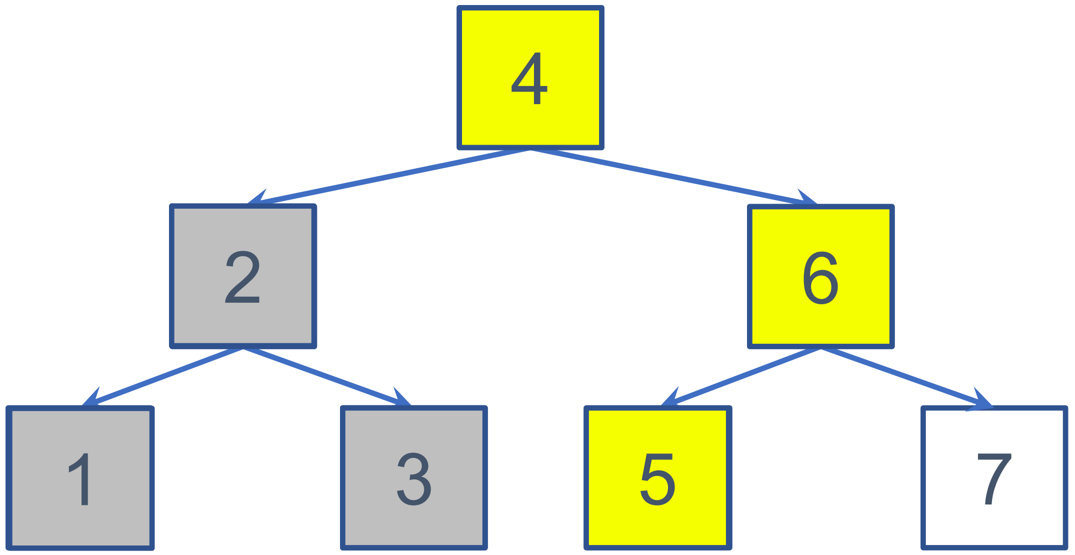
Depth First Traversals



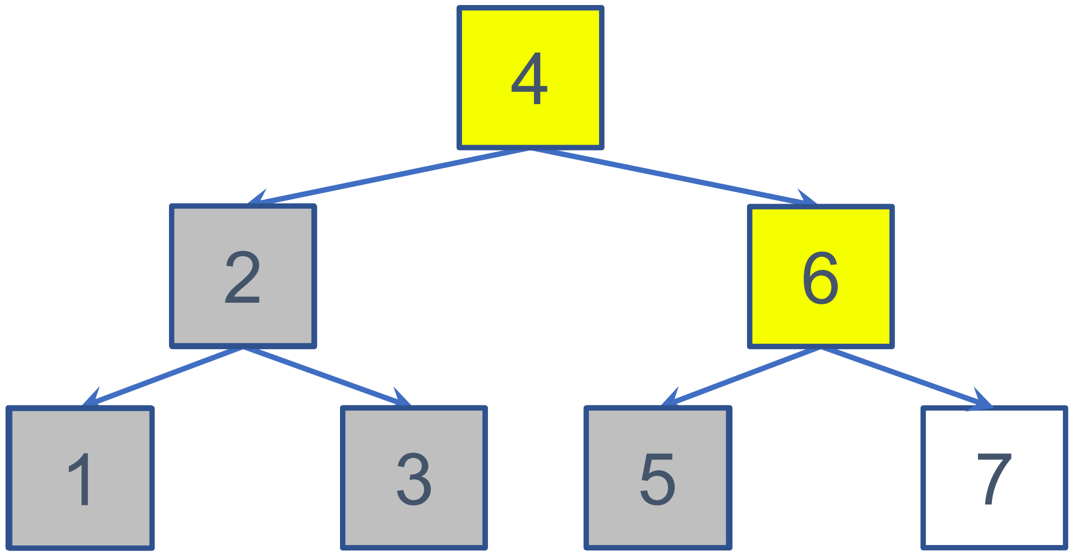
Depth First Traversals



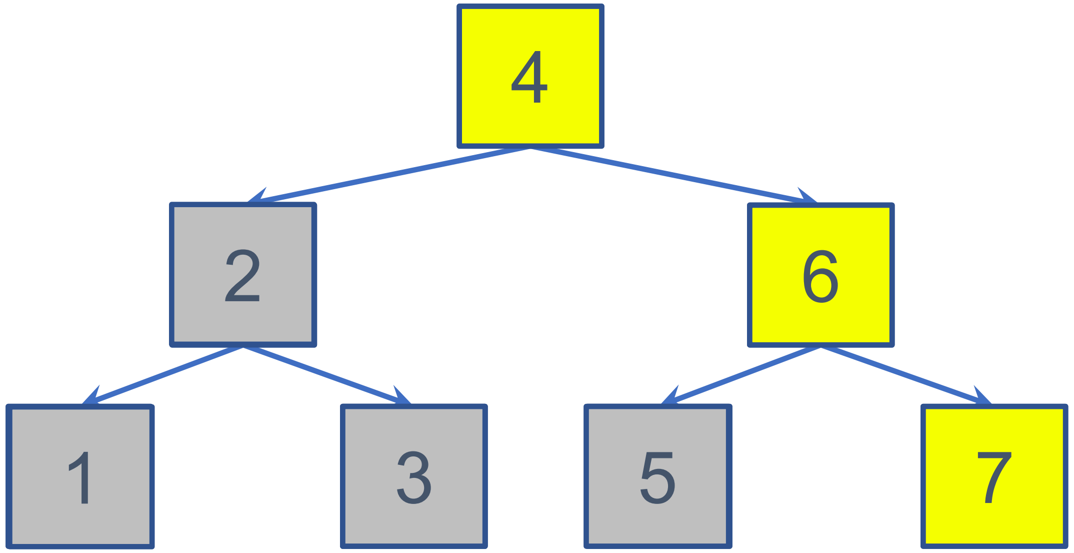
Depth First Traversals



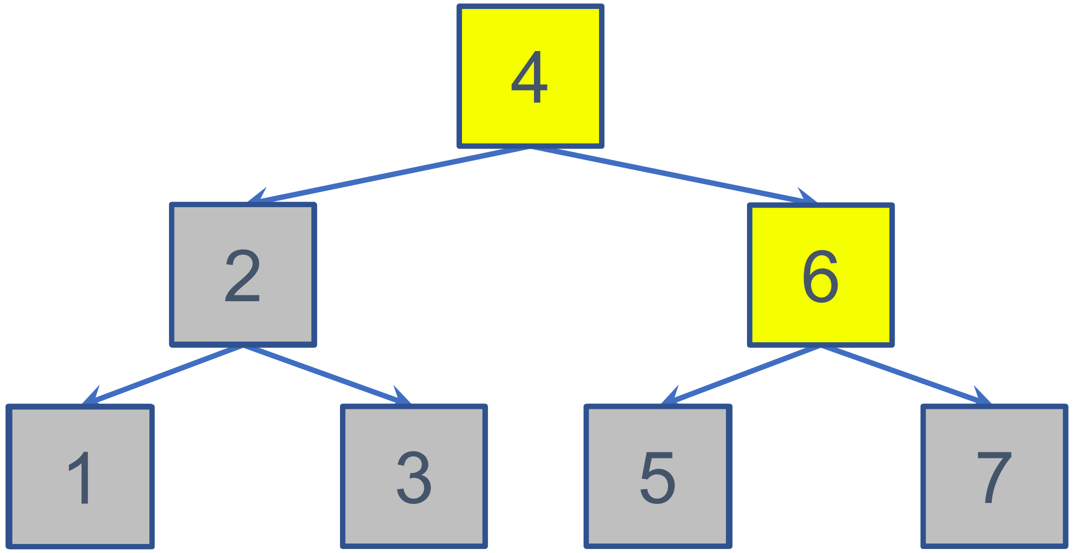
Depth First Traversals



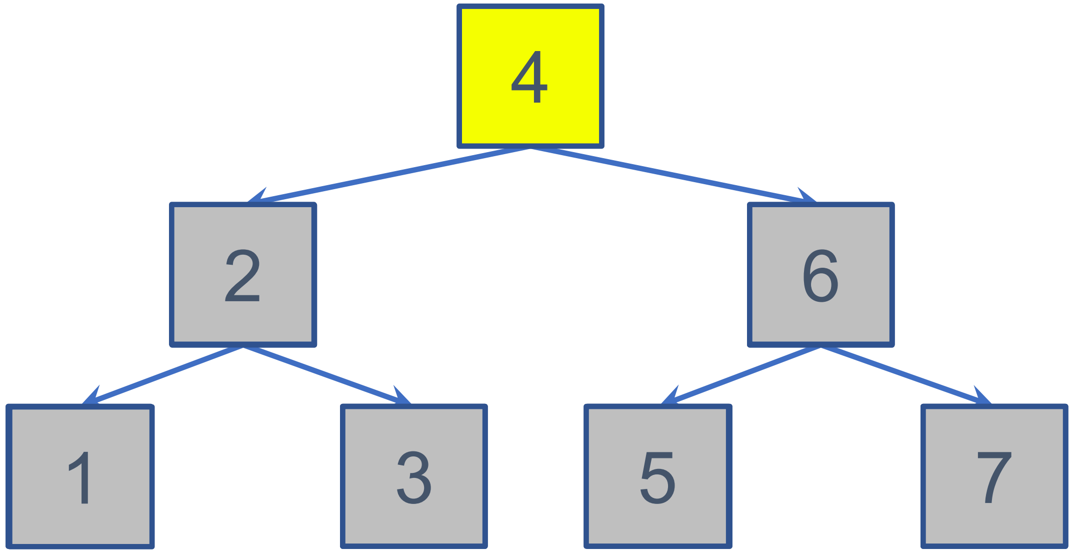
Depth First Traversals



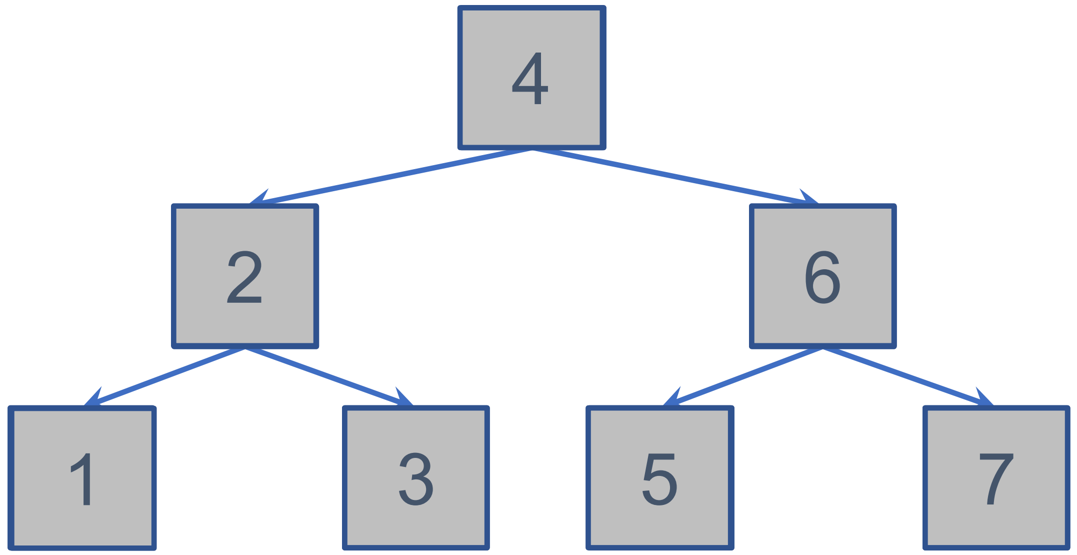
Depth First Traversals



Depth First Traversals

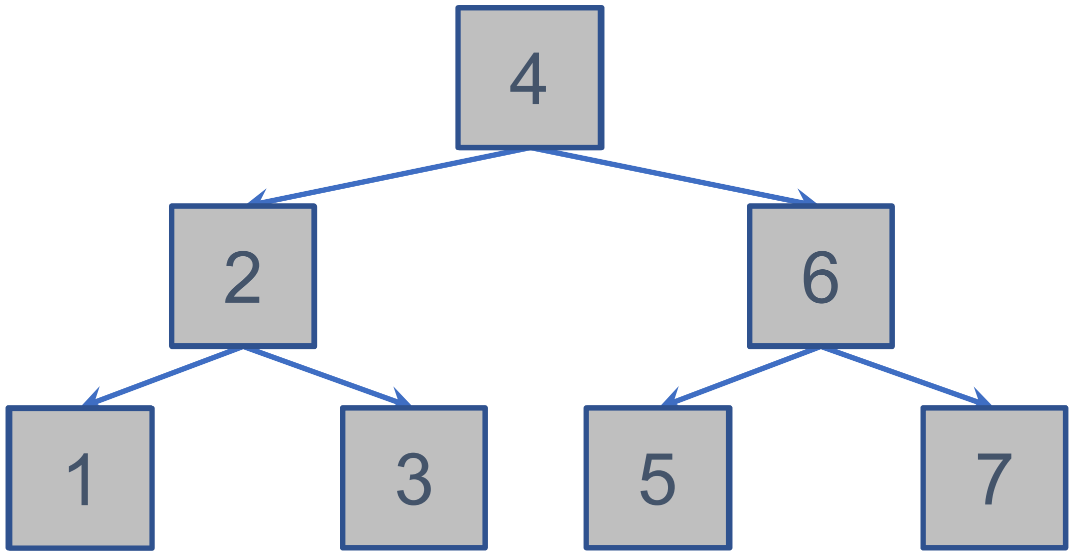


Depth First Traversals



Depth First Traversals

- Three types
  - Preorder, inorder, and postorder



Depth-First Traversal

- Depth-First Traversal
- **Preorder**
- Inorder
- Postorder

Computing Bootcamp

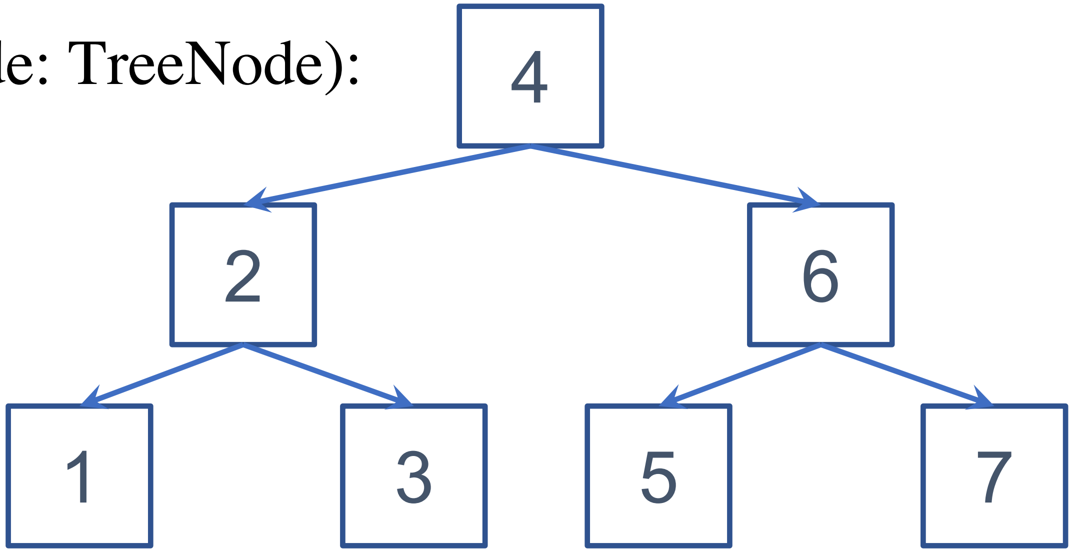
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



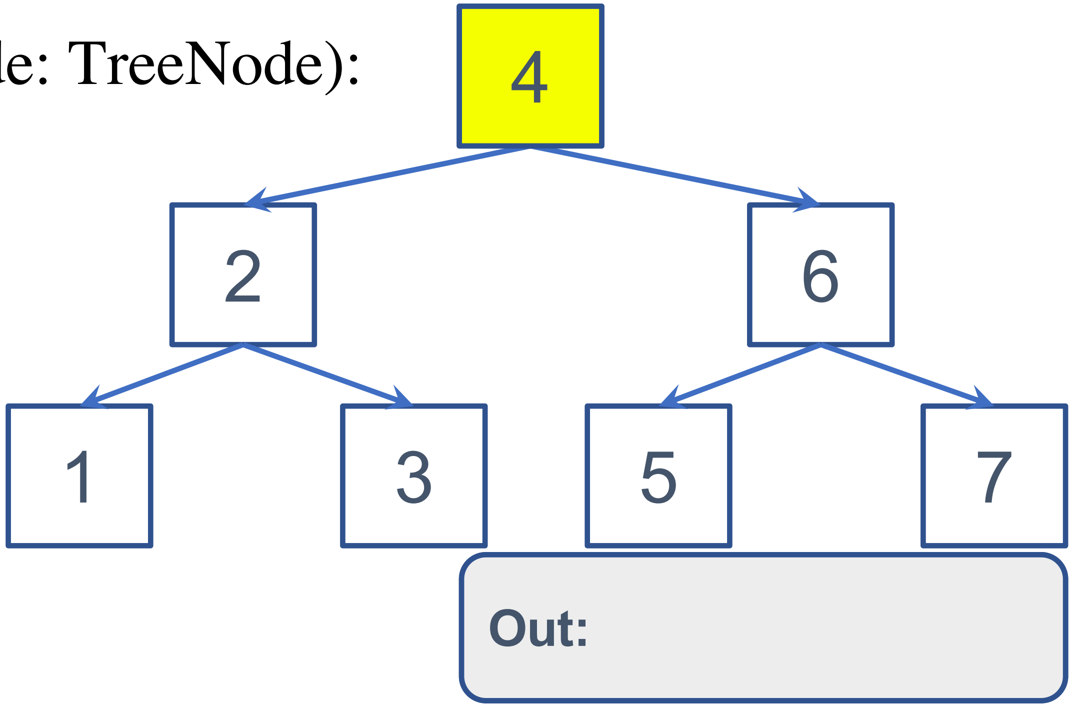
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



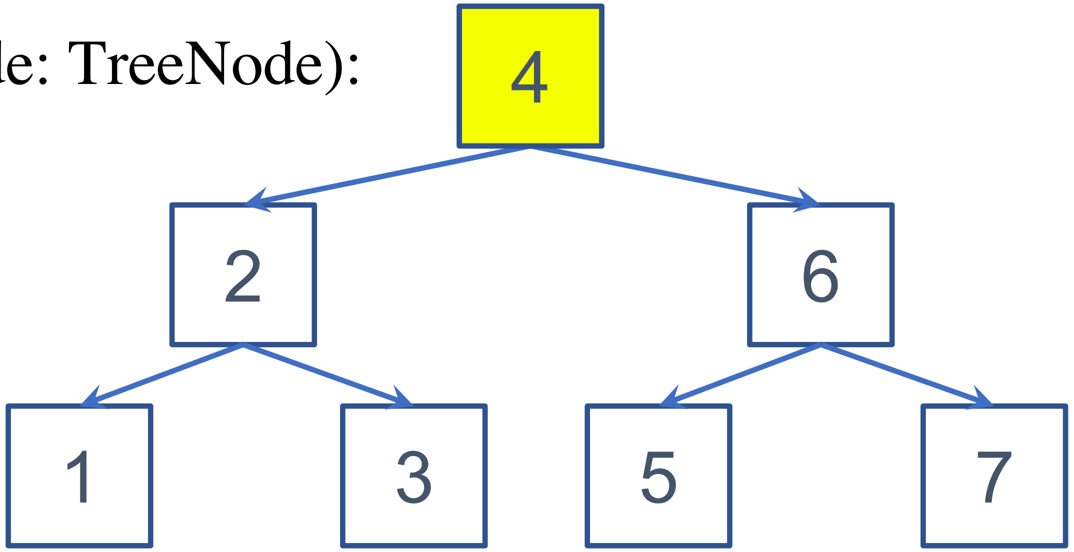
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



Out: 4

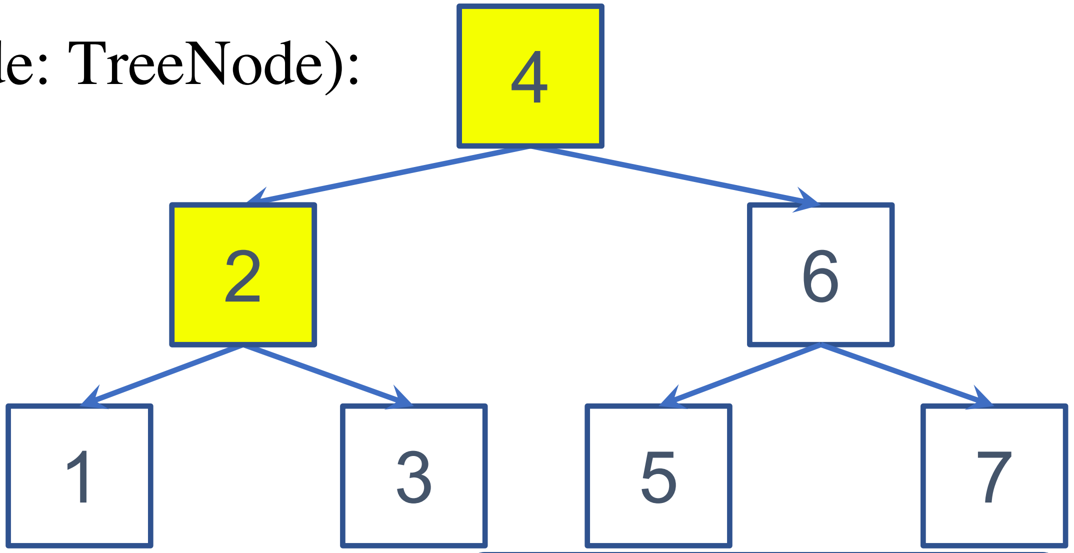
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



Out: 4

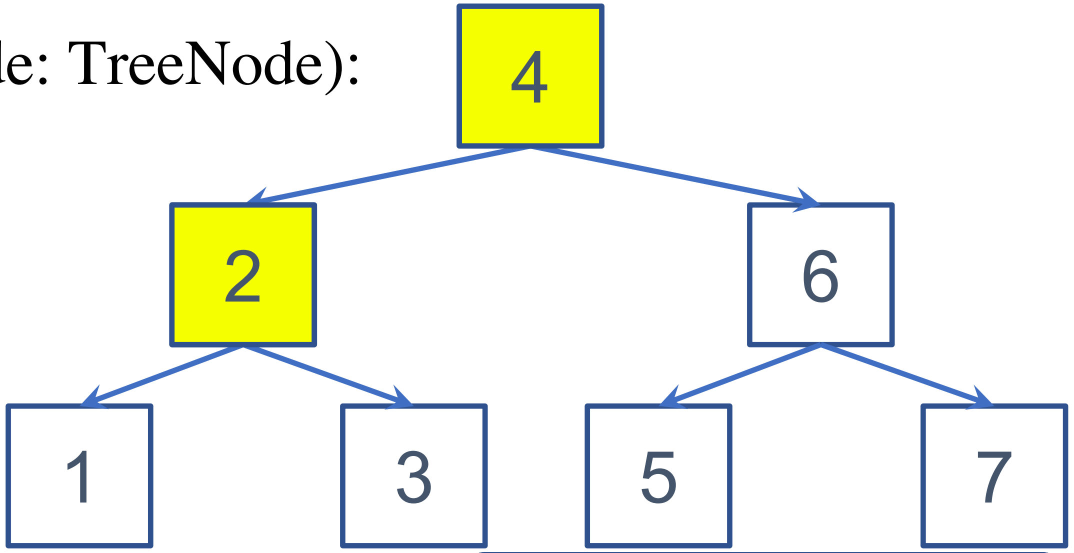
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



Out: 4 2

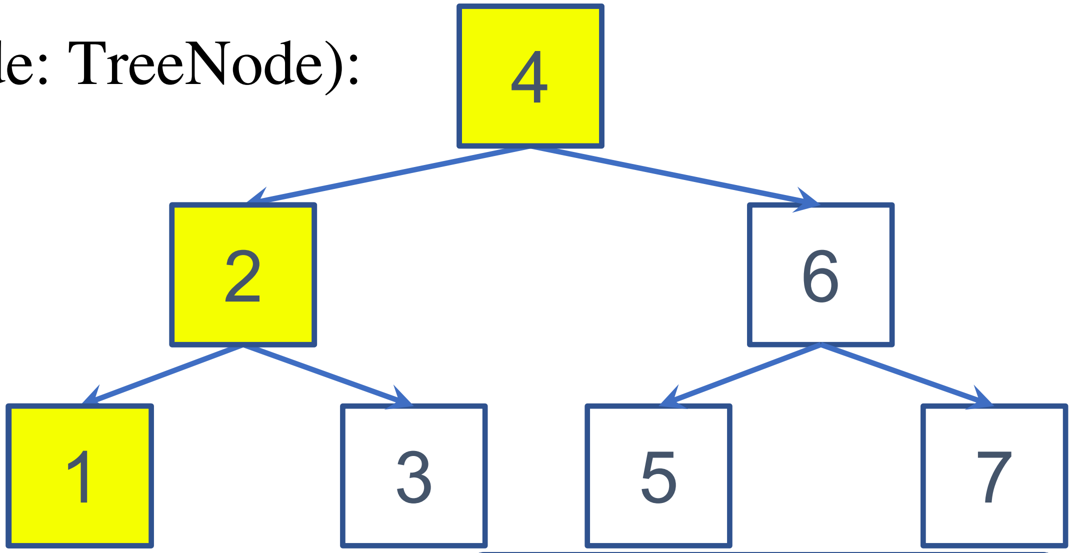
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



```
Out: 4 2
```

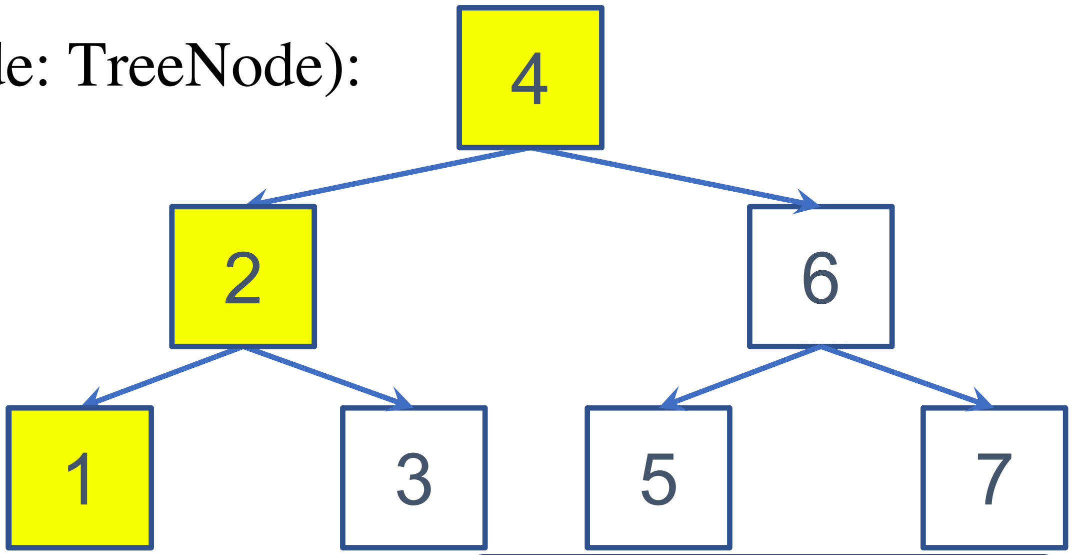
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



Out: 4 2 1

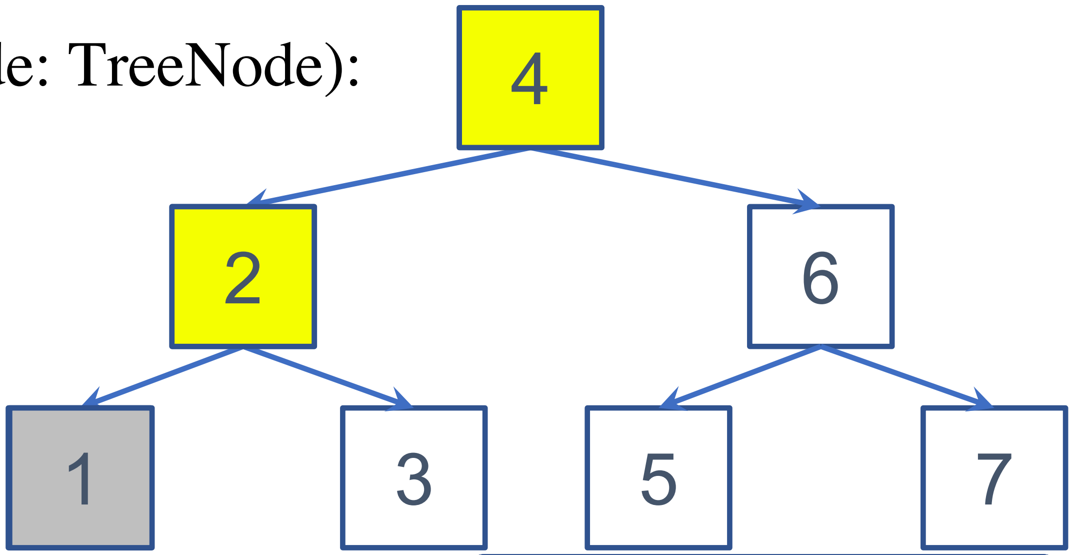
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



Out: 4 2 1

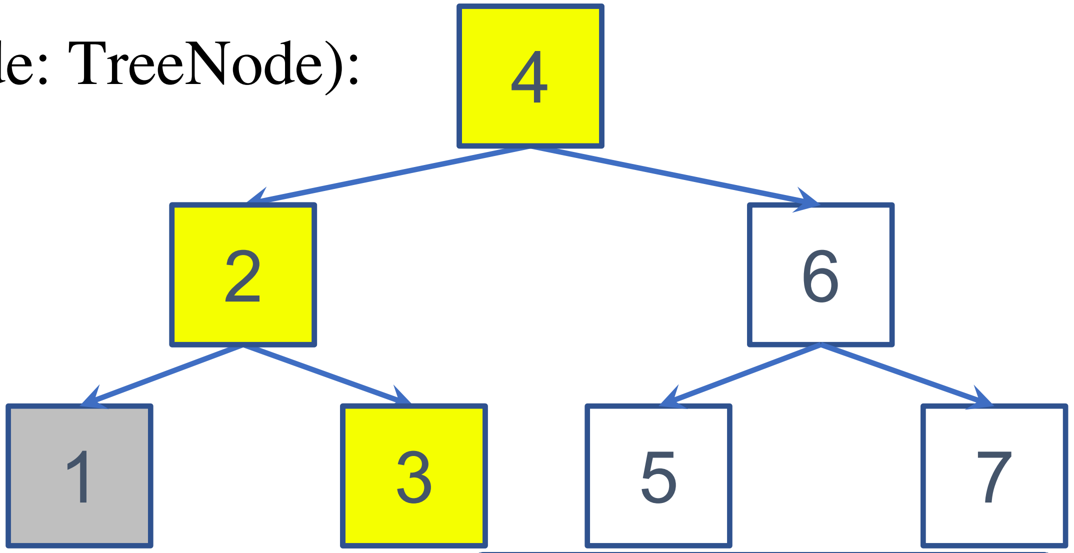
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



```
Out: 4 2 1
```

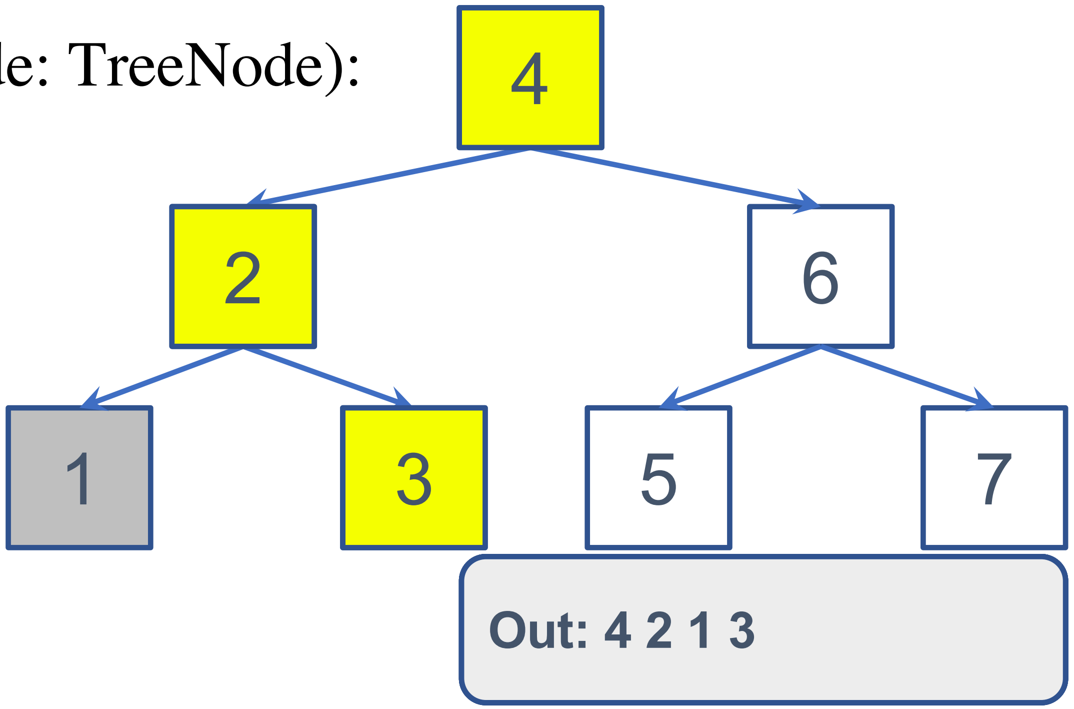
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



Out: 4 2 1 3

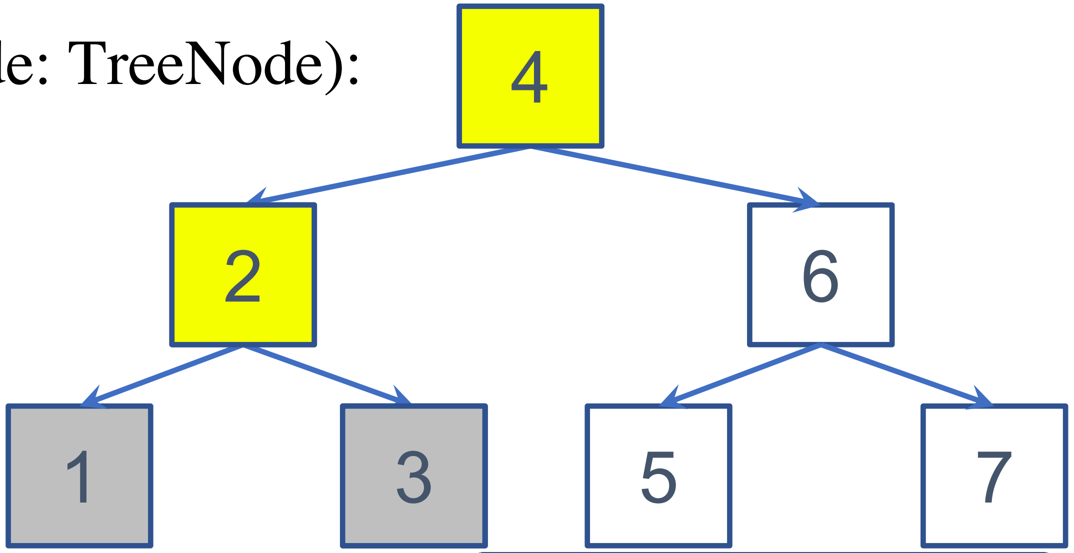
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



```
Out: 4 2 1 3
```

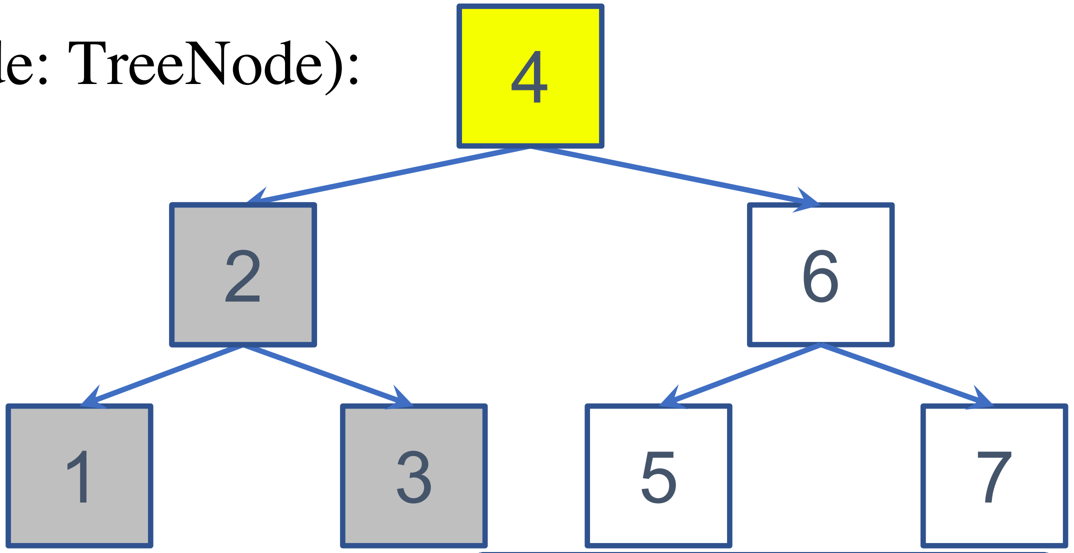
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



Out: 4 2 1 3

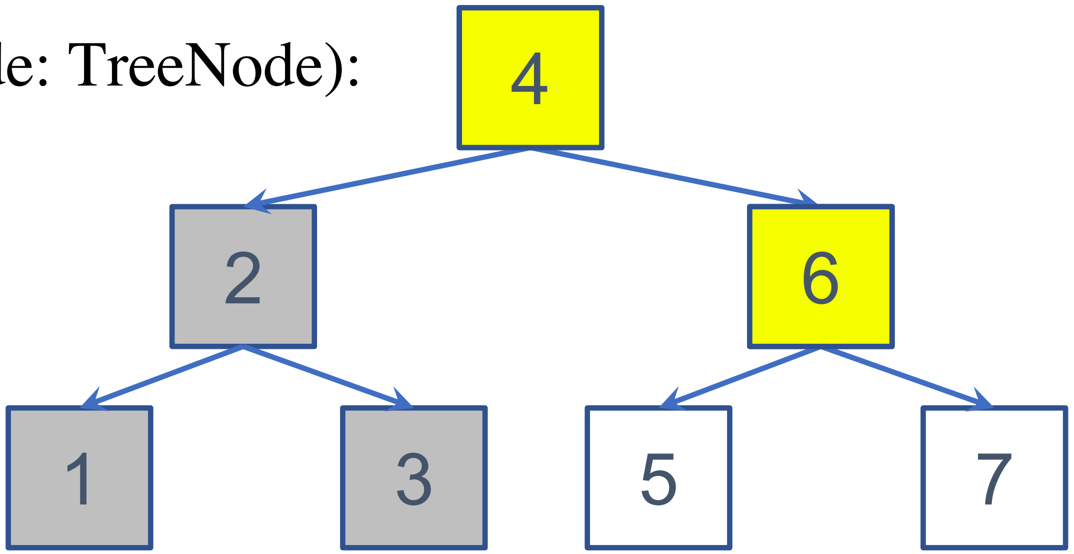
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



```
Out: 4 2 1 3
```

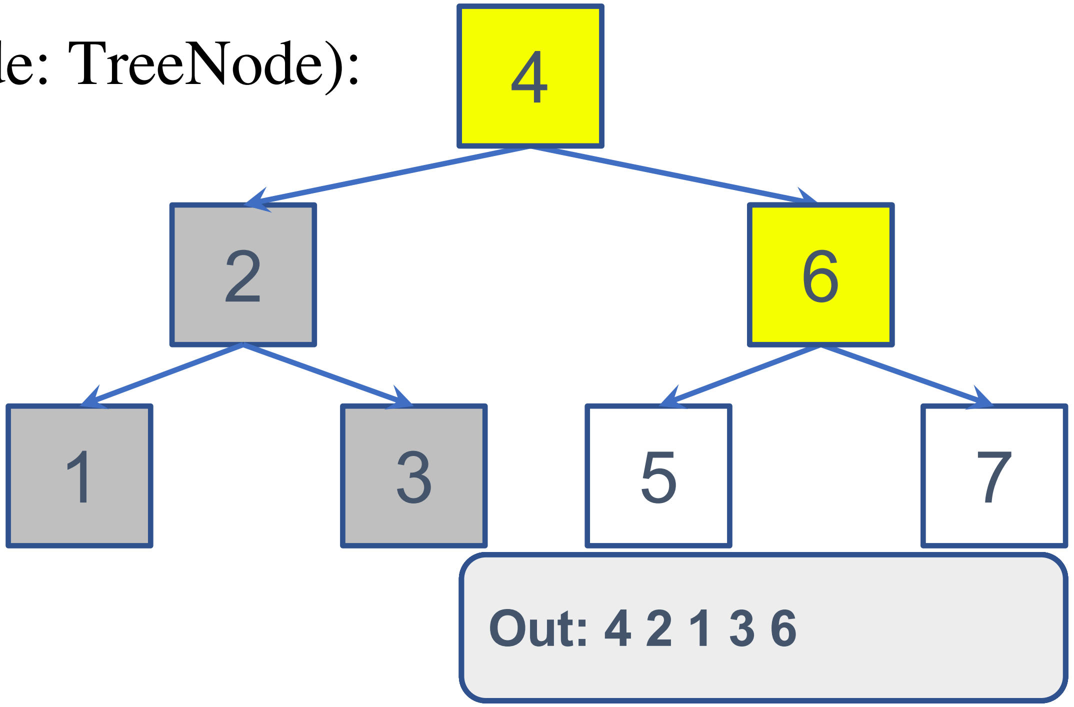
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



Out: 4 2 1 3 6

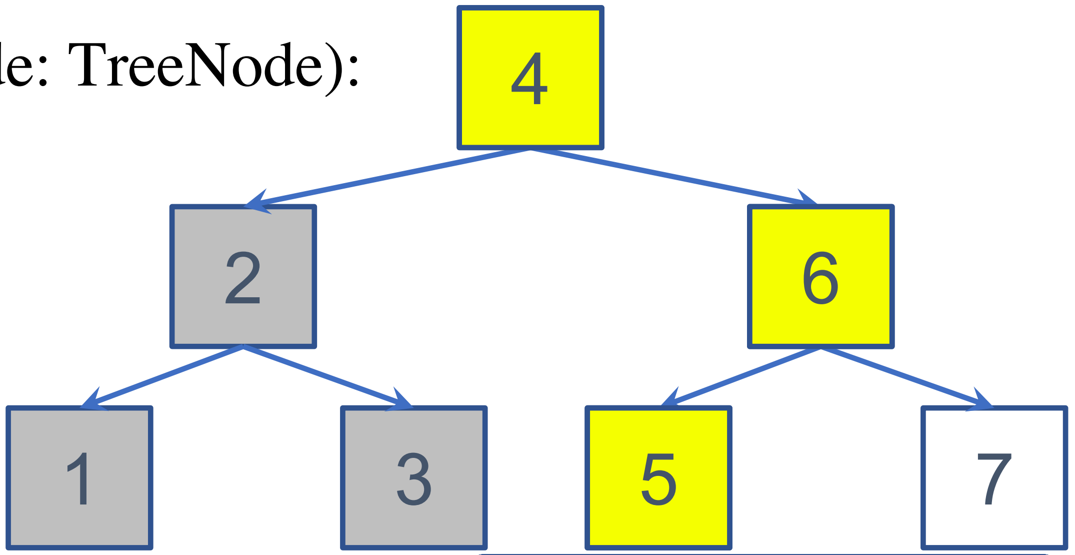
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



```
Out: 4 2 1 3 6
```

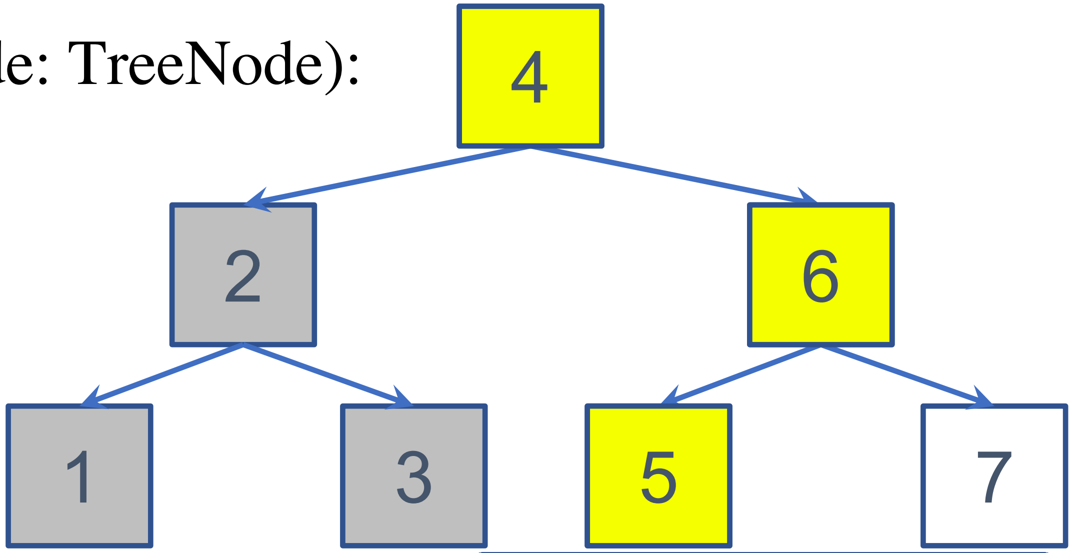
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



```
Out: 4 2 1 3 6 5
```

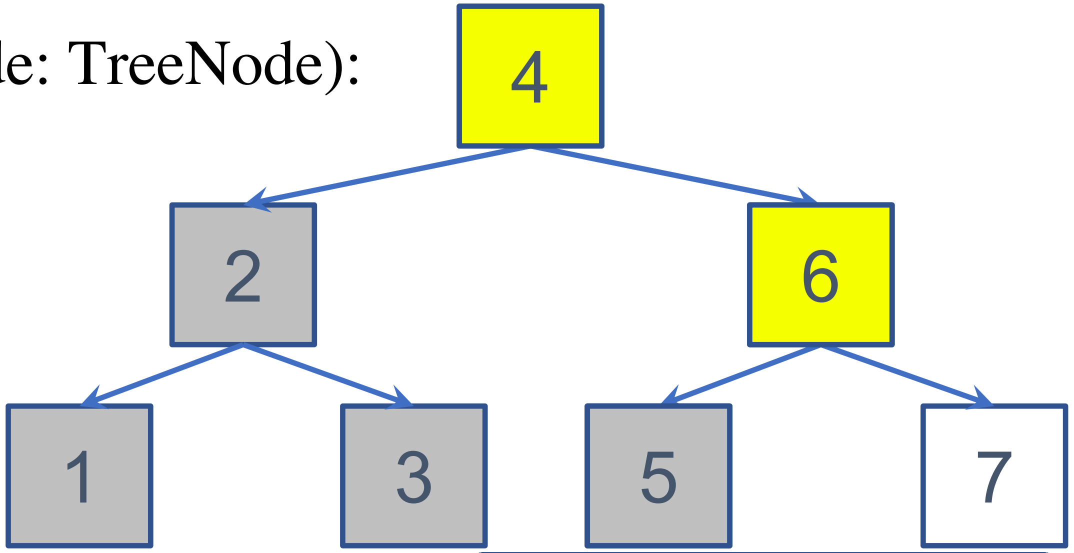
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



```
Out: 4 2 1 3 6 5
```

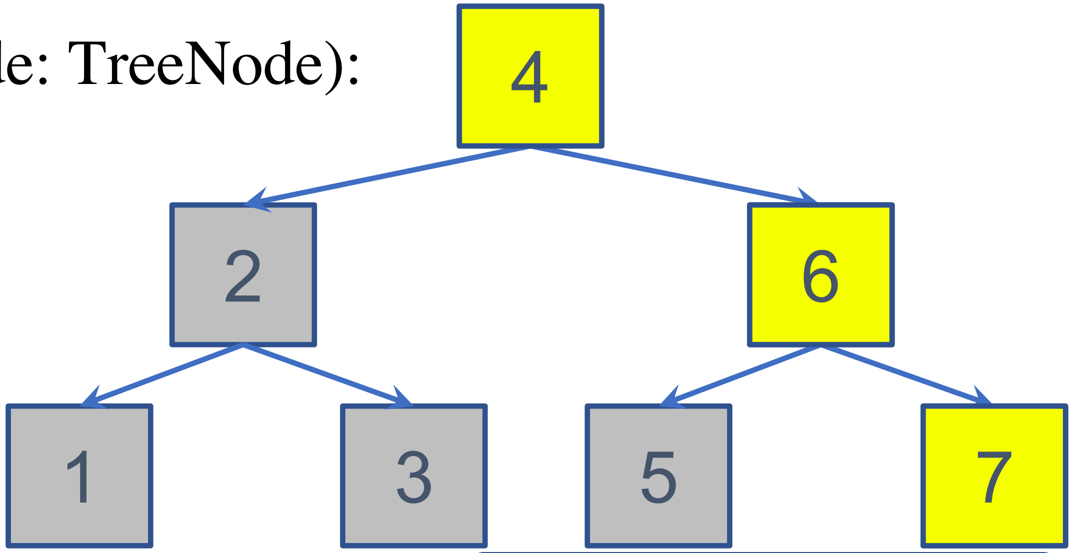
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



```
Out: 4 2 1 3 6 5
```

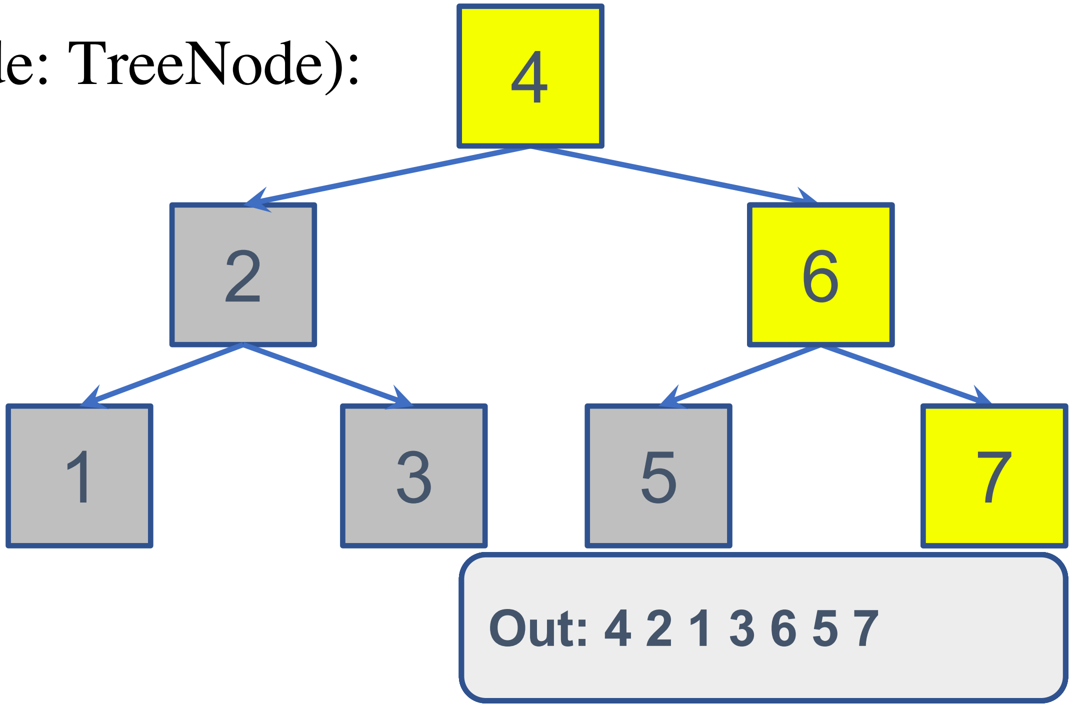
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



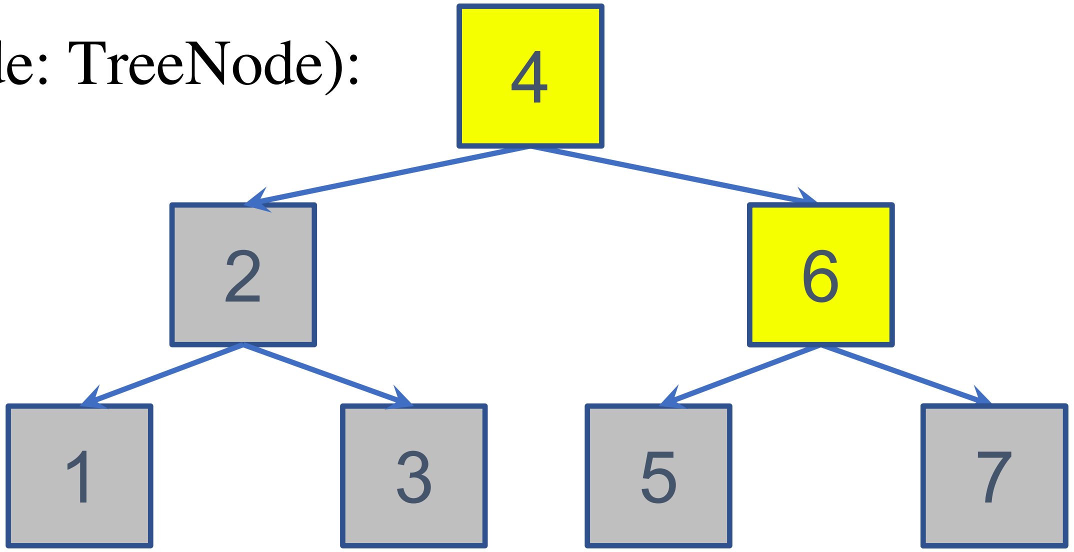
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



Out: 4 2 1 3 6 5 7

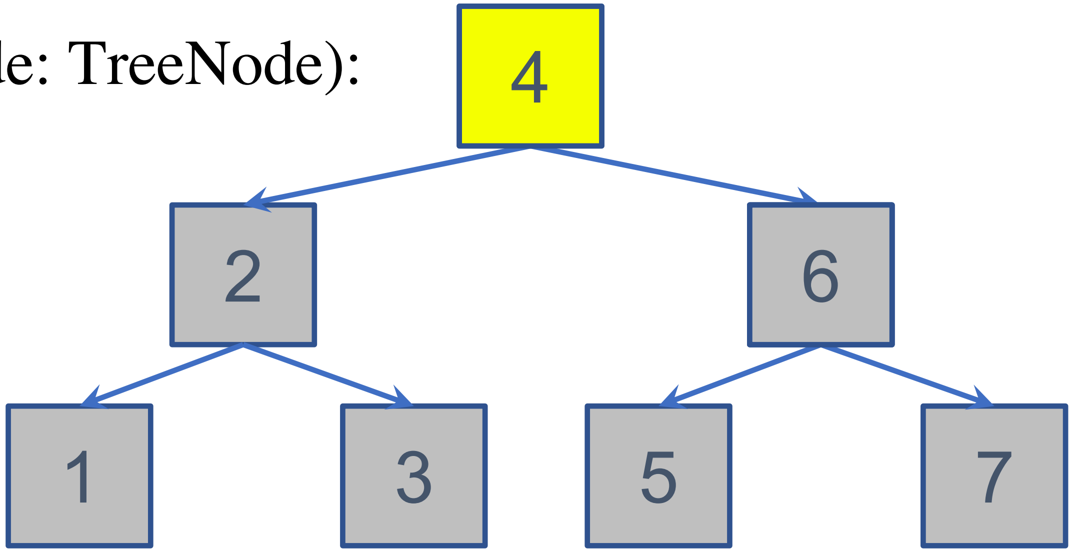
Depth First Traversals – Preorder

- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```



Out: 4 2 1 3 6 5 7

Depth First Traversals – Preorder

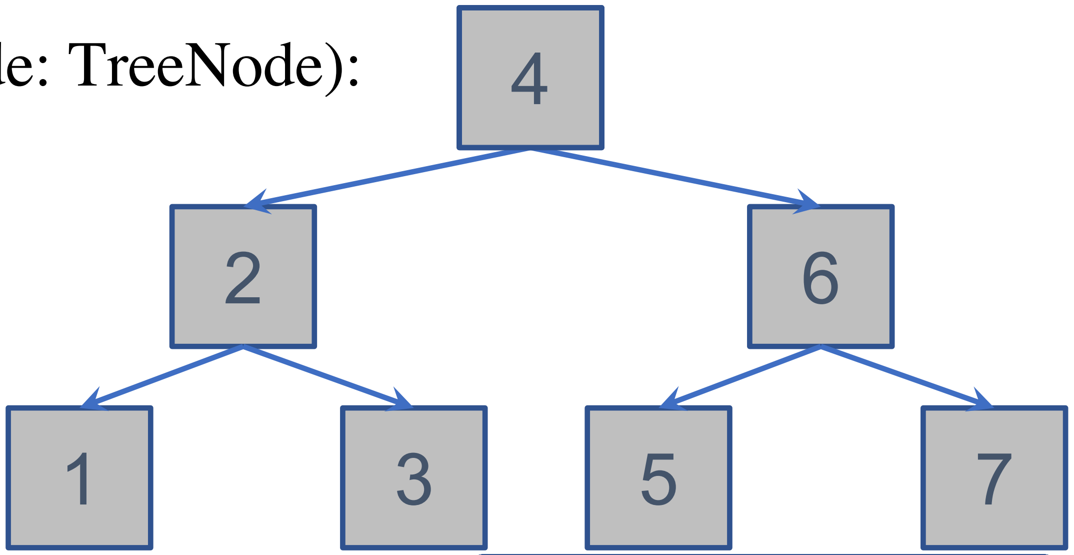
- Visit a node **before** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_preorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        self.visit(curNode)
        for childNode in curNode.child:
            self.__DFT_preorderHelp(childNode)

    def DFT_preorder(self):
        self.__DFT_preorderHelp(self.root)
```

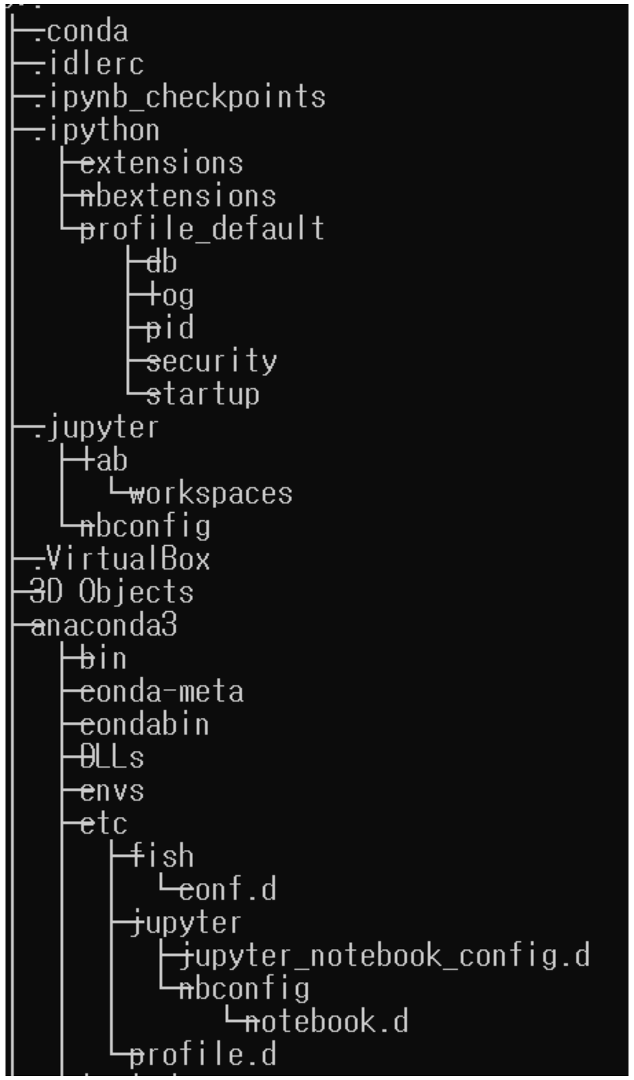
Done!



Out: 4 2 1 3 6 5 7

Depth First Traversals – Preorder

- **Application**: Directory listing (type "Tree" for fun)



```
├─.conda
├─.idlerc
├─.ipynb_checkpoints
├─.ipython
│  ├─extensions
│  ├─nbextensions
│  └─profile_default
│      ├─db
│      ├─log
│      ├─pid
│      ├─security
│      └─startup
├─.jupyter
│  ├─lab
│  │  └─workspaces
│  └─nbconfig
├─.VirtualBox
├─3D Objects
├─anaconda3
│  ├─bin
│  ├─conda-meta
│  ├─condabin
│  ├─DLLs
│  ├─envs
│  ├─etc
│  │  ├─fish
│  │  │  └─conf.d
│  │  ├─jupyter
│  │  │  ├─jupyter_notebook_config.d
│  │  │  └─nbconfig
│  │  │      └─notebook.d
│  │  └─profile.d
```

Depth-First Traversal

- Depth-First Traversal
- Preorder
- **Inorder**
- Postorder

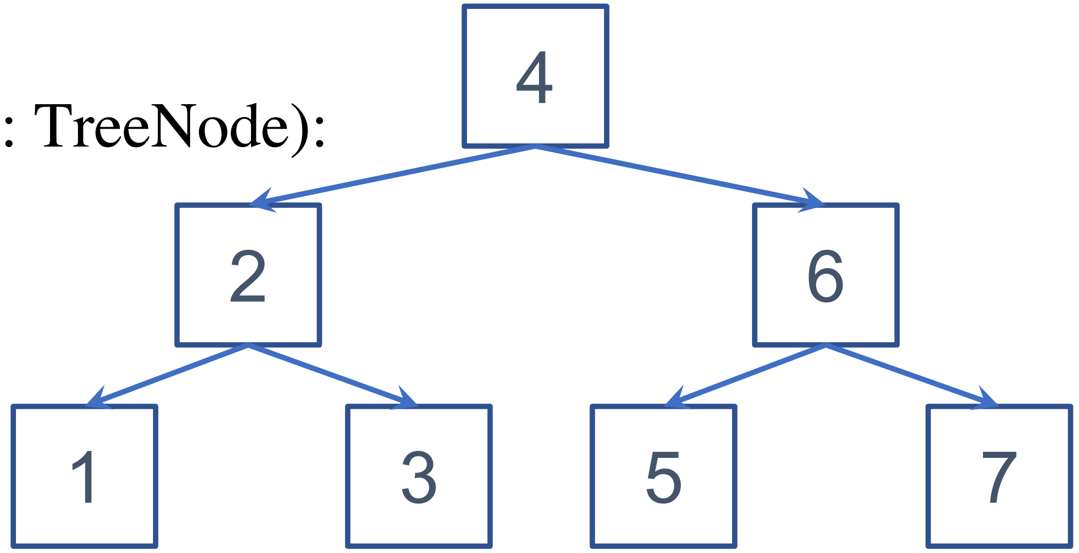
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out:

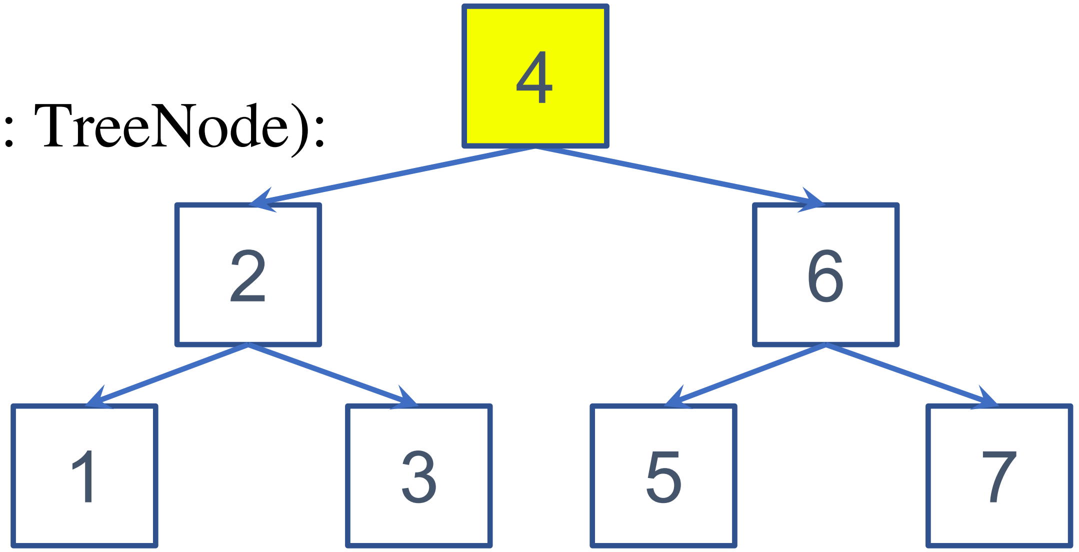
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out:

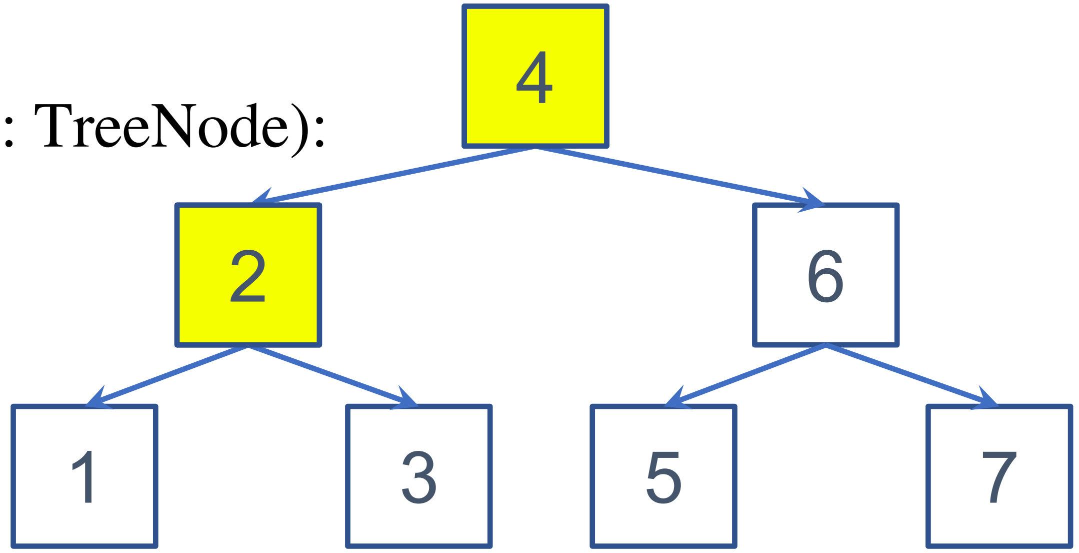
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out:

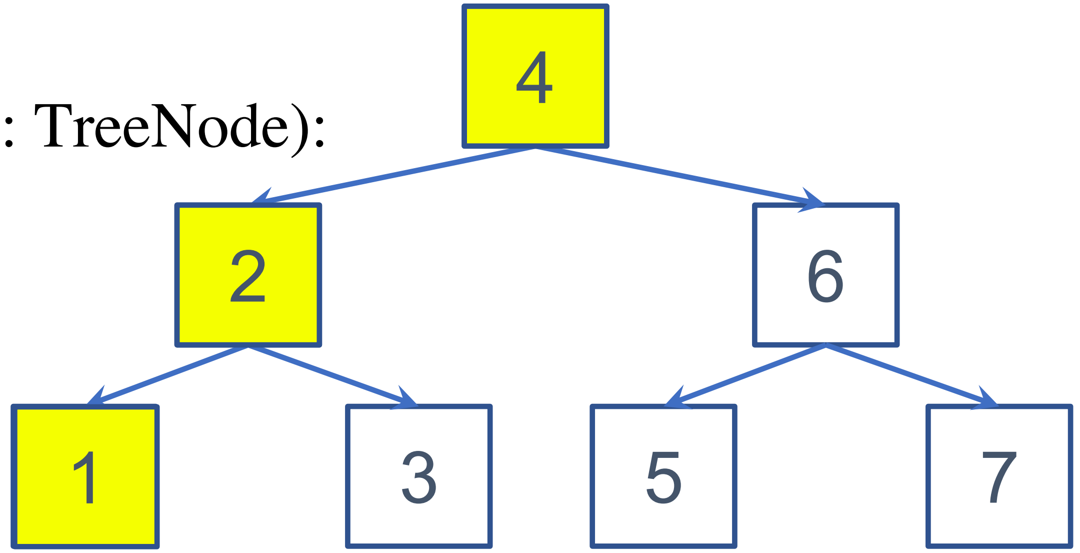
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out:

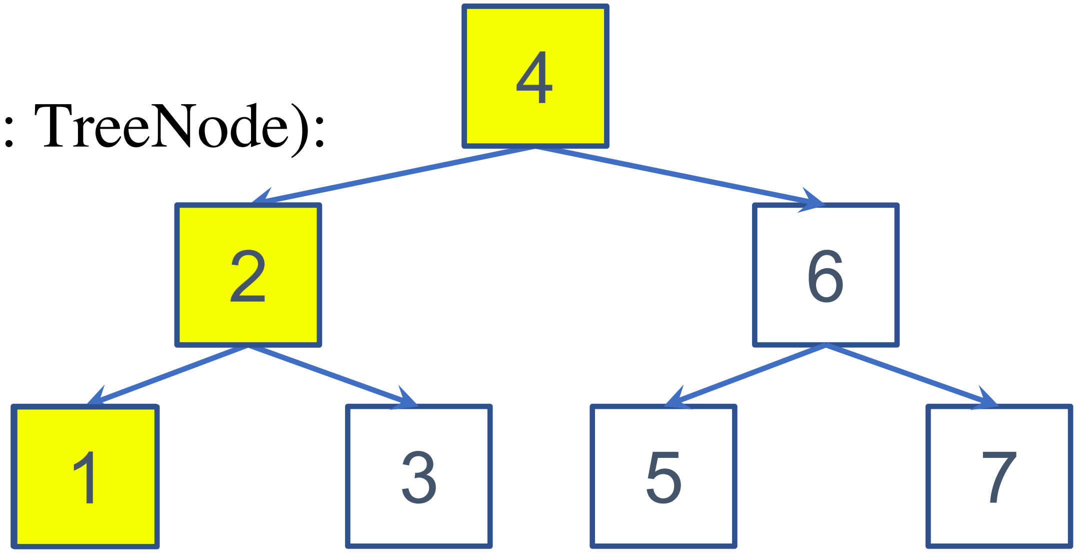
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1

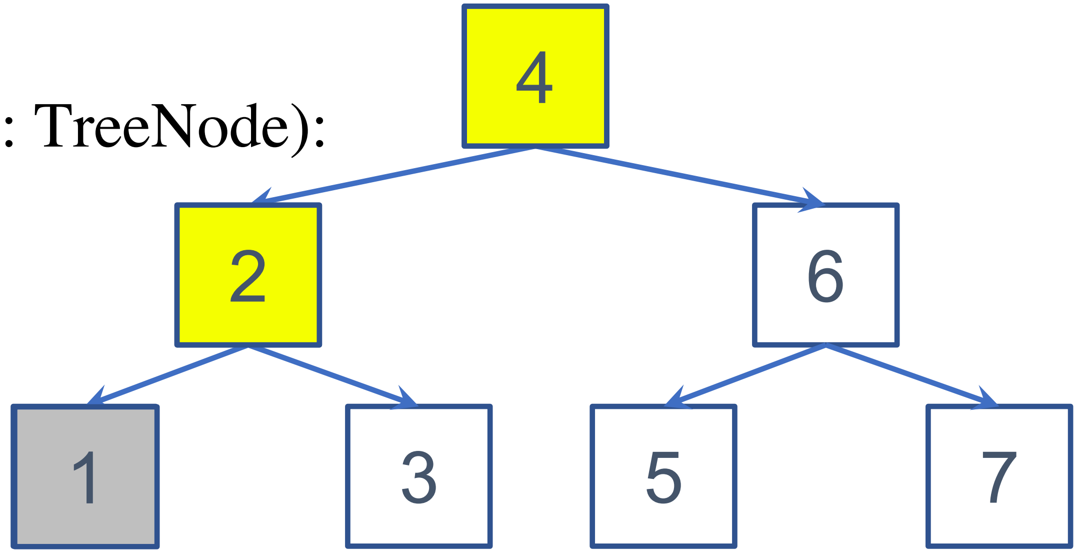
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1

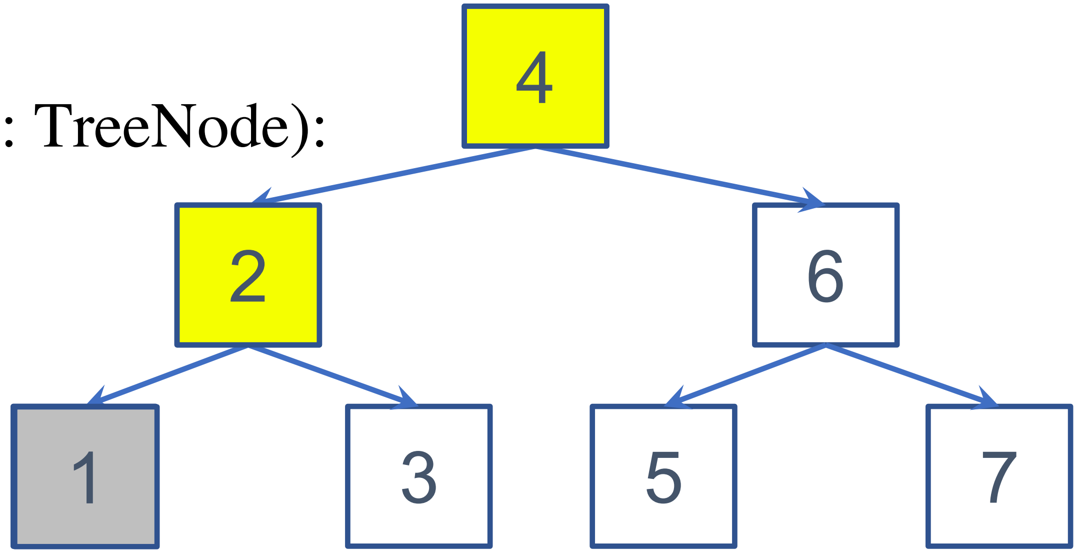
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1 2

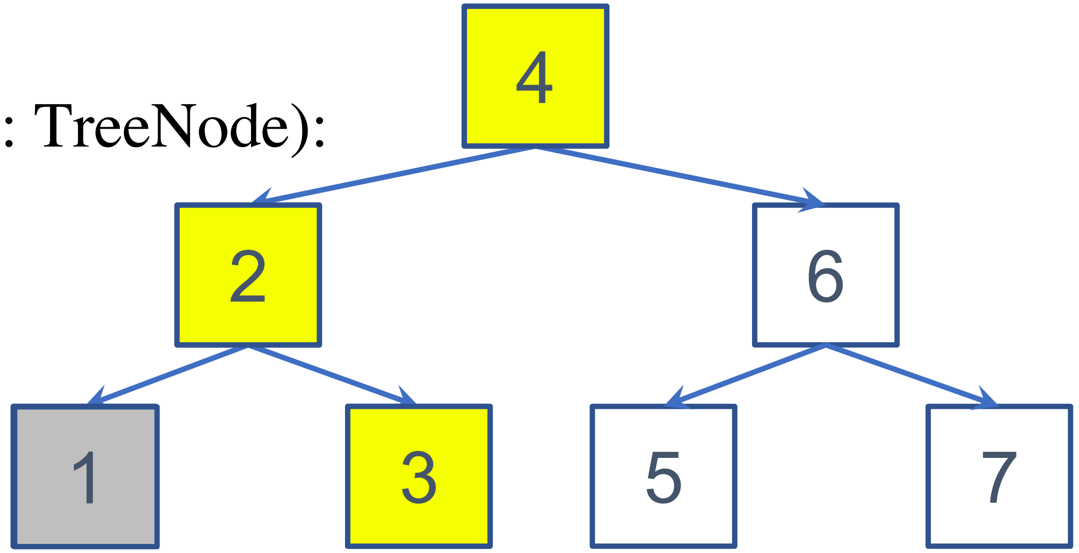
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1 2

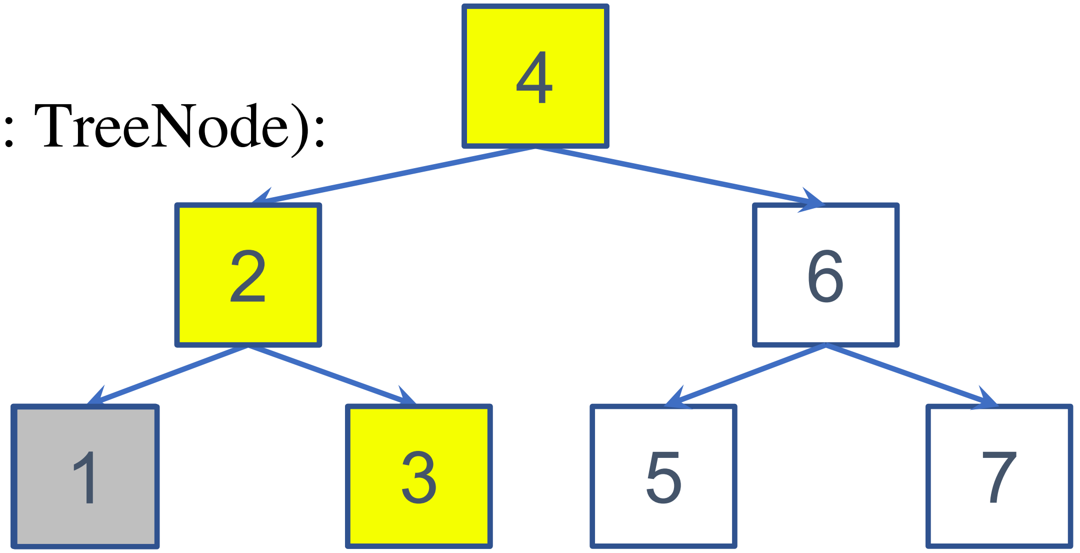
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1 2 3

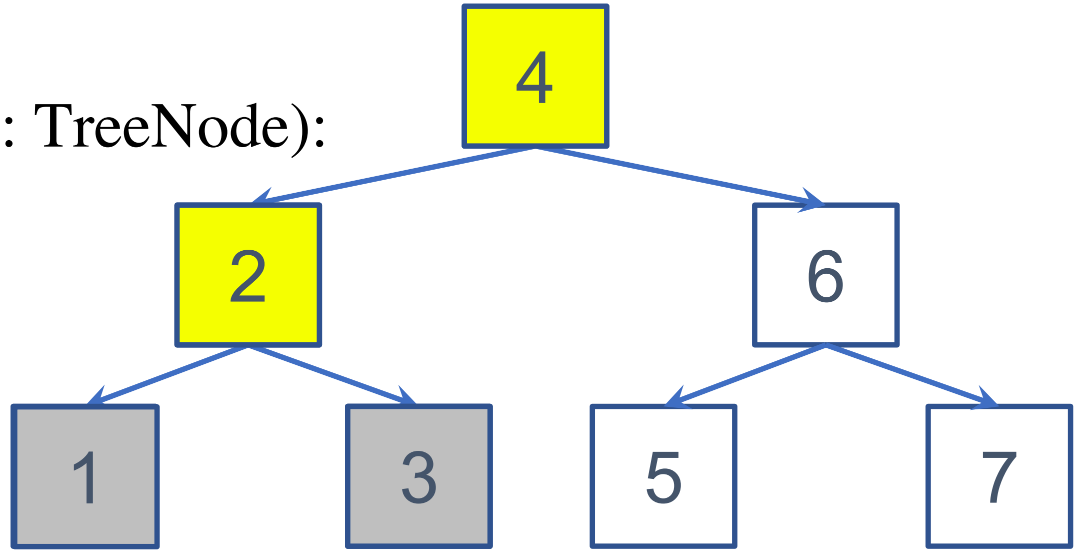
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1 2 3

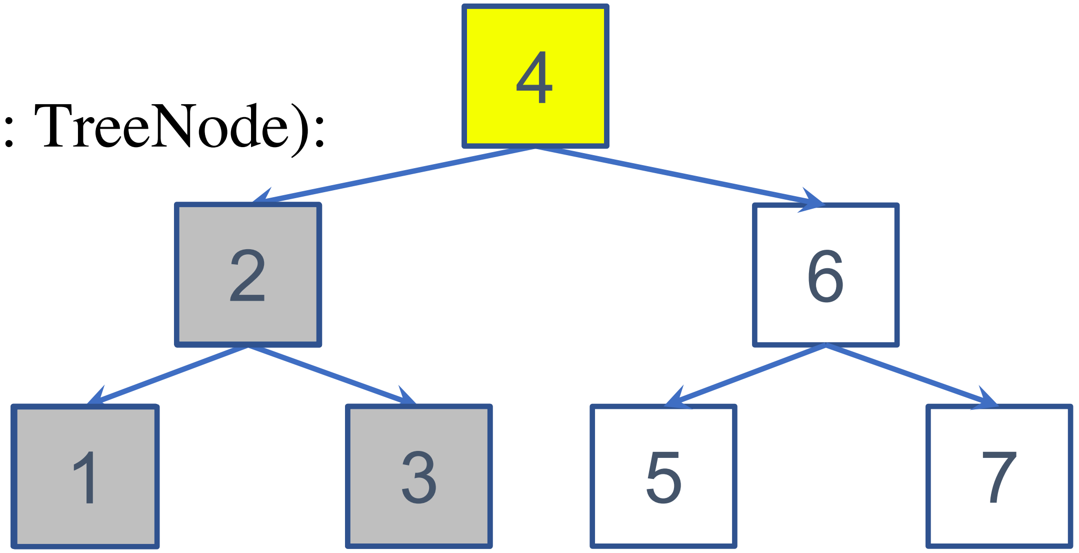
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1 2 3

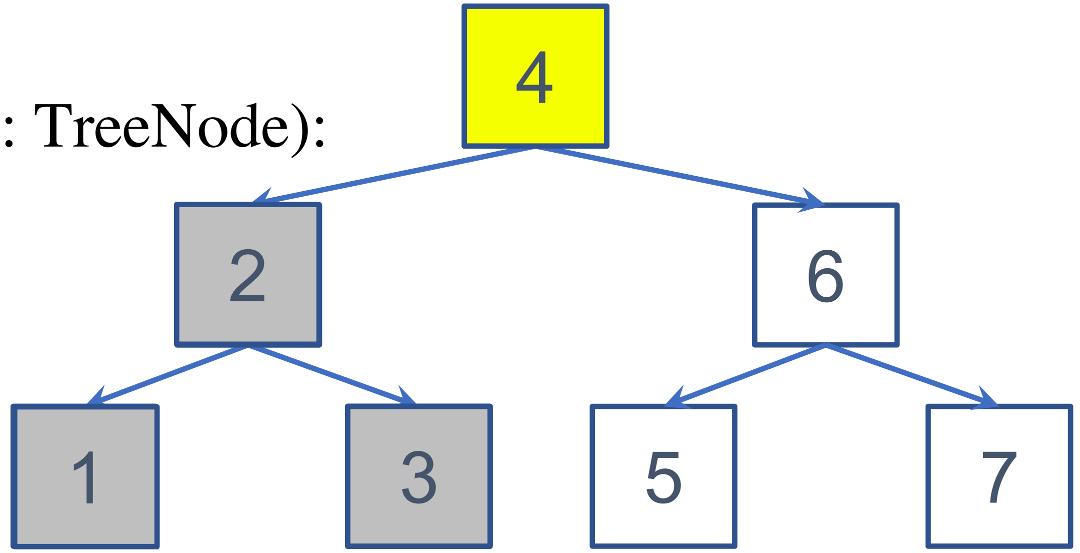
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



```
Out: 1 2 3 4
```

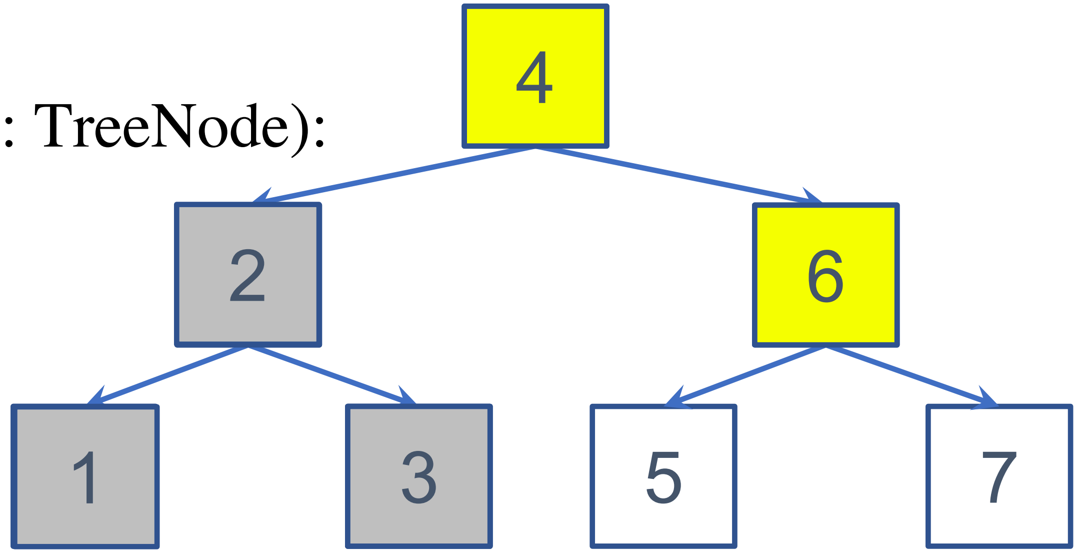
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



```
Out: 1 2 3 4
```

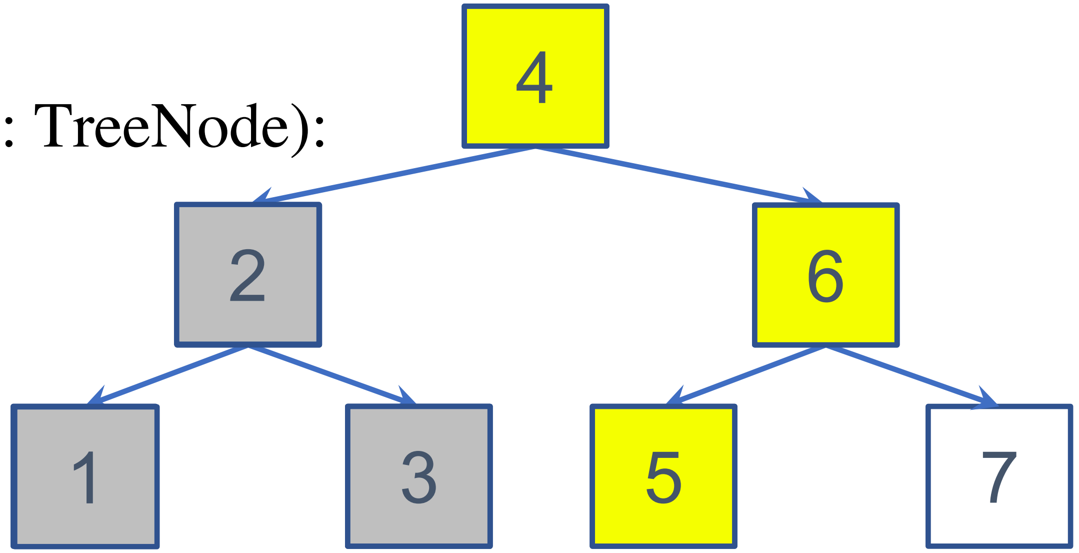
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1 2 3 4

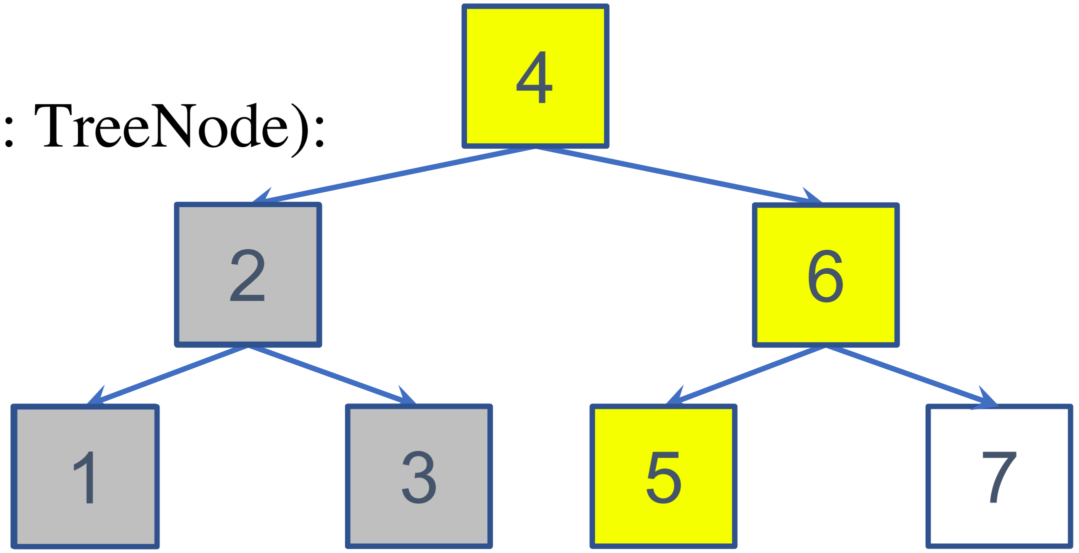
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1 2 3 4 5

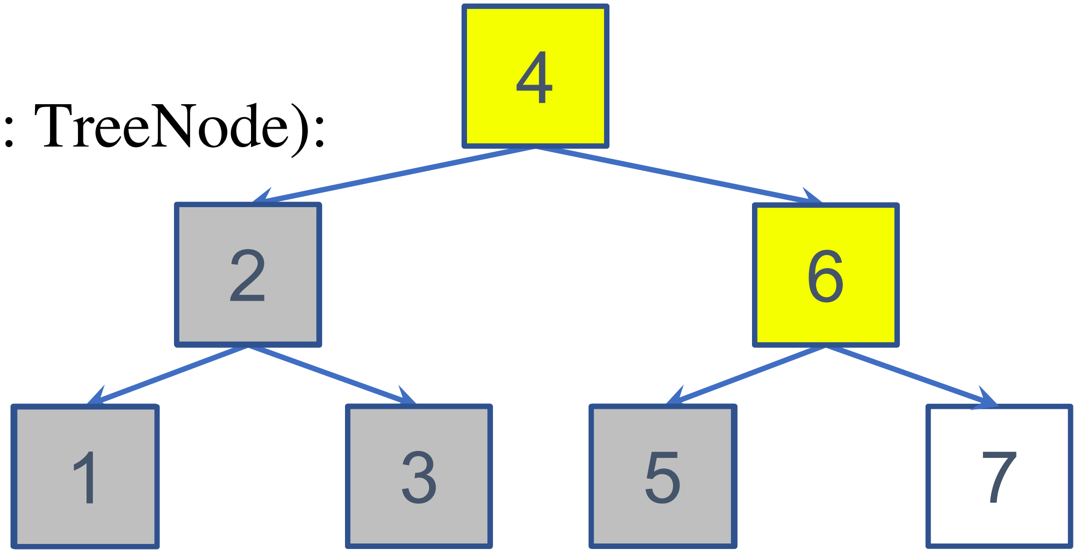
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1 2 3 4 5

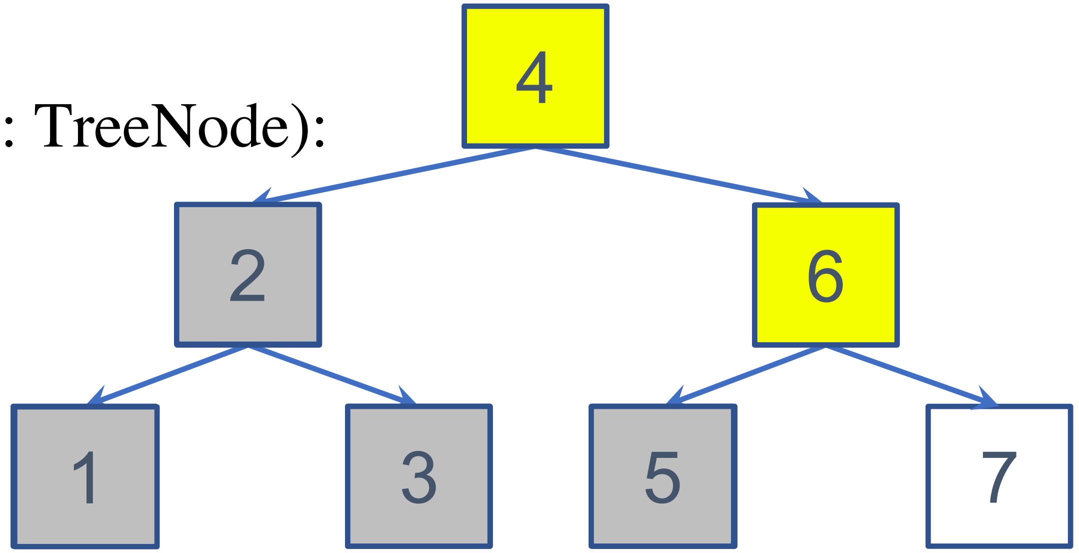
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1 2 3 4 5 6

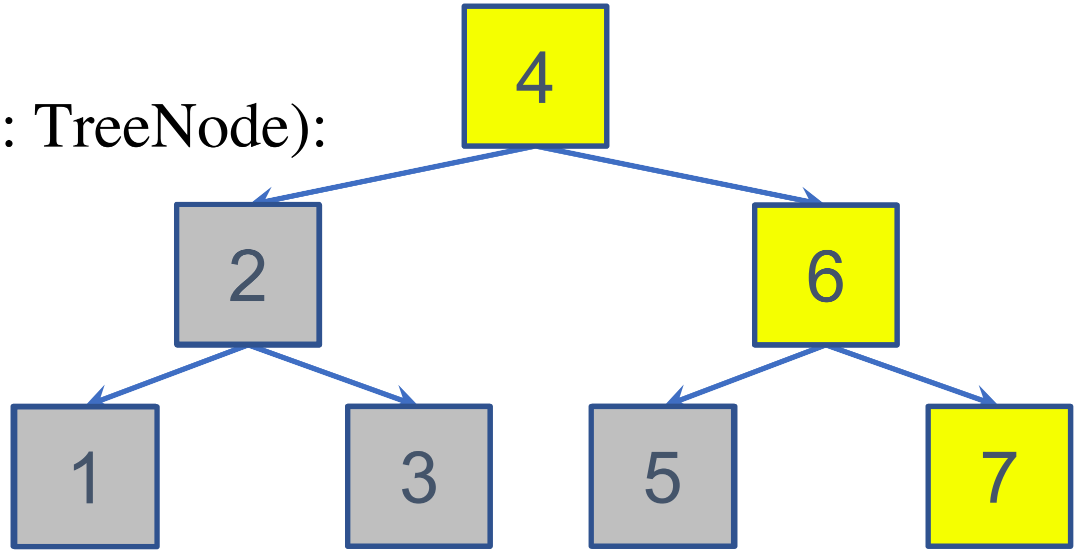
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



```
Out: 1 2 3 4 5 6
```

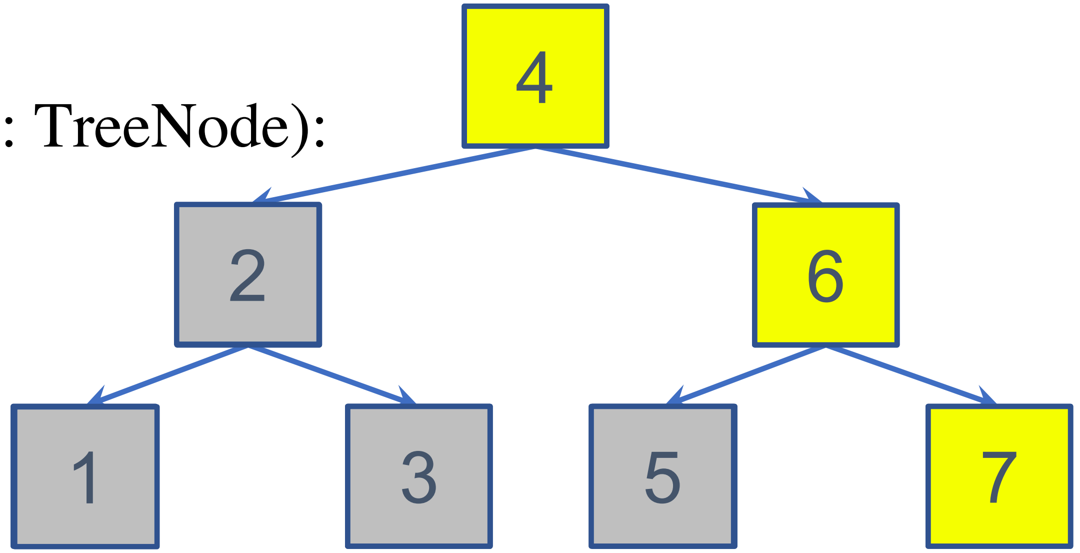
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1 2 3 4 5 6 7

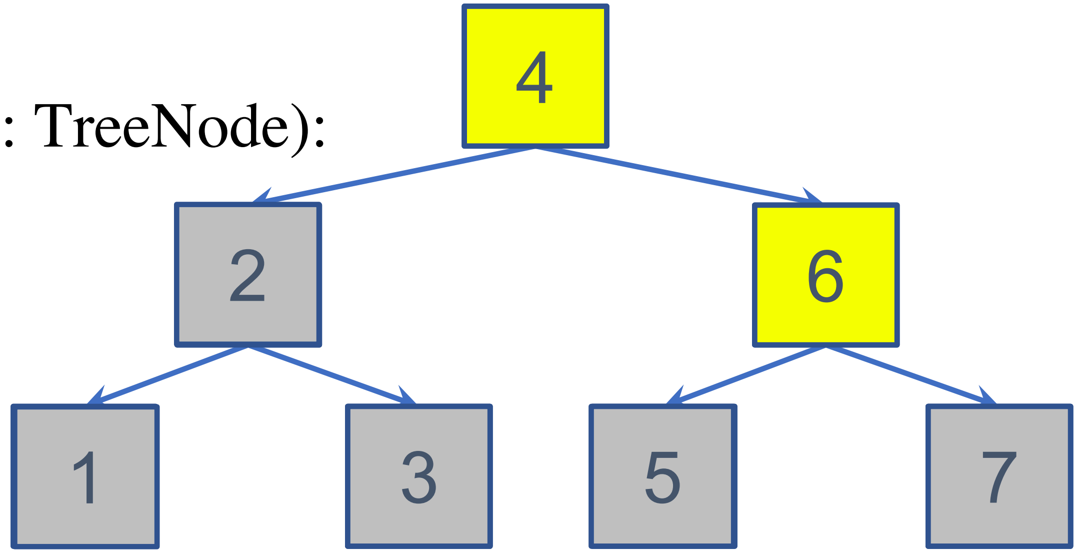
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1 2 3 4 5 6 7

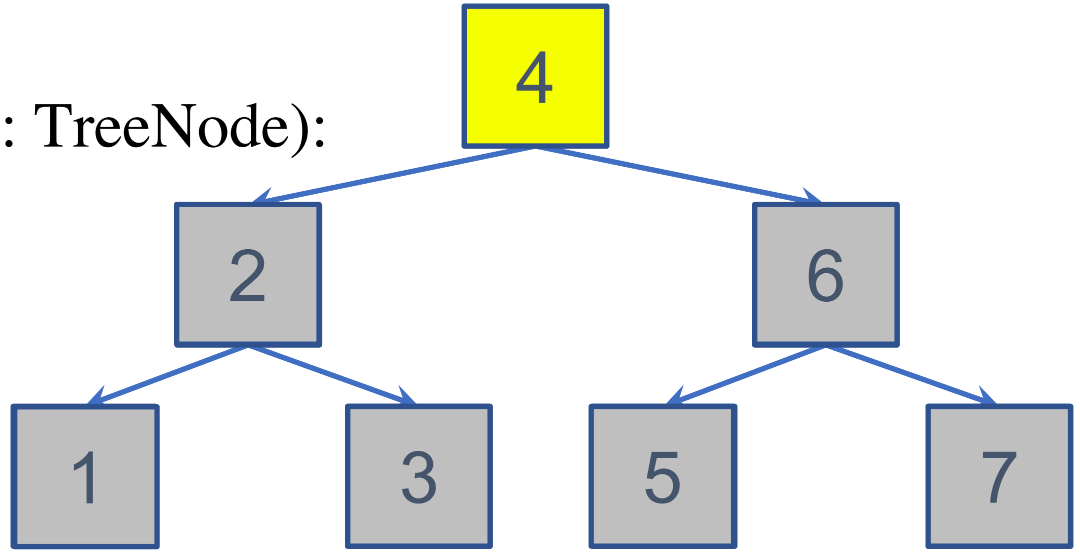
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



Out: 1 2 3 4 5 6 7

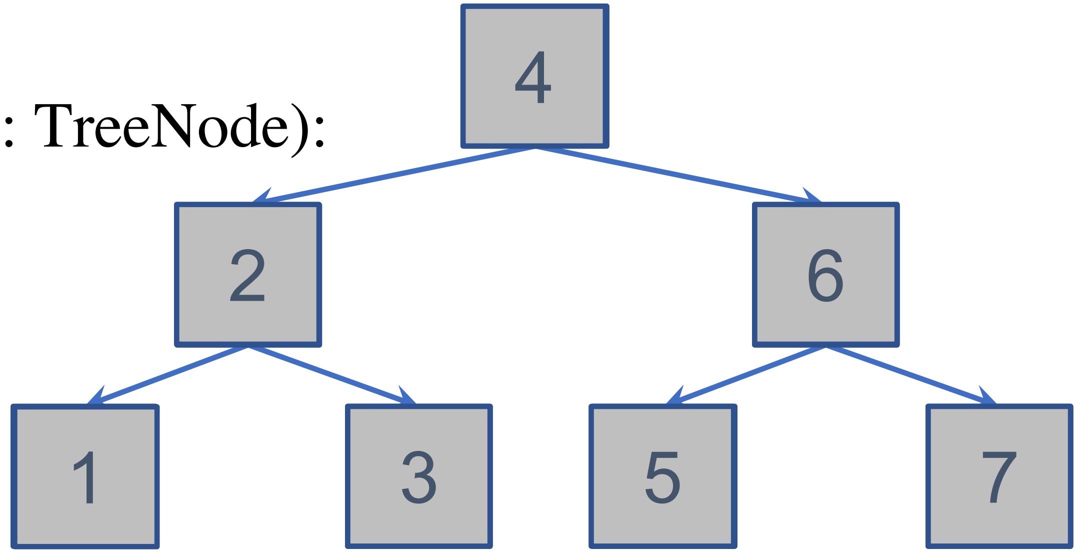
Depth First Traversals – Inorder

- Traverse a node's children from left to right and visit the node **in the middle**

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_inorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            if i == 1:
                self.visit(curNode)
            self.__DFT_inorderHelp(curNode.child[i])

    def DFT_inorder(self):
        self.__DFT_inorderHelp(self.root)
```



```
Out: 1 2 3 4 5 6 7
```

Depth First Traversals – Inorder

- **Application**: Covert a binary search tree to a sorted list (Flattening a BST)

Depth-First Traversal

- Depth-First Traversal
- Preorder
- Inorder
- **Postorder**

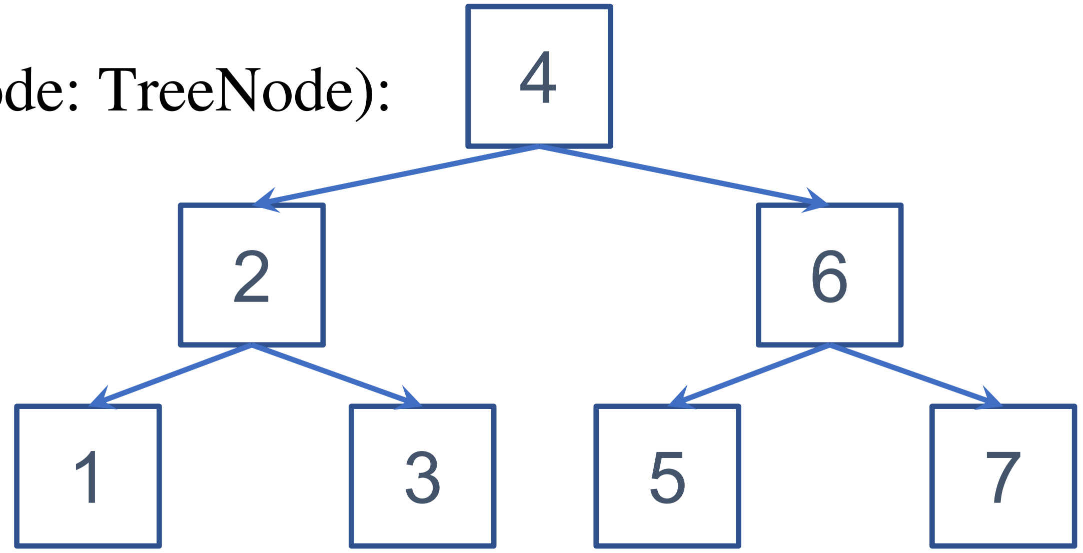
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out:

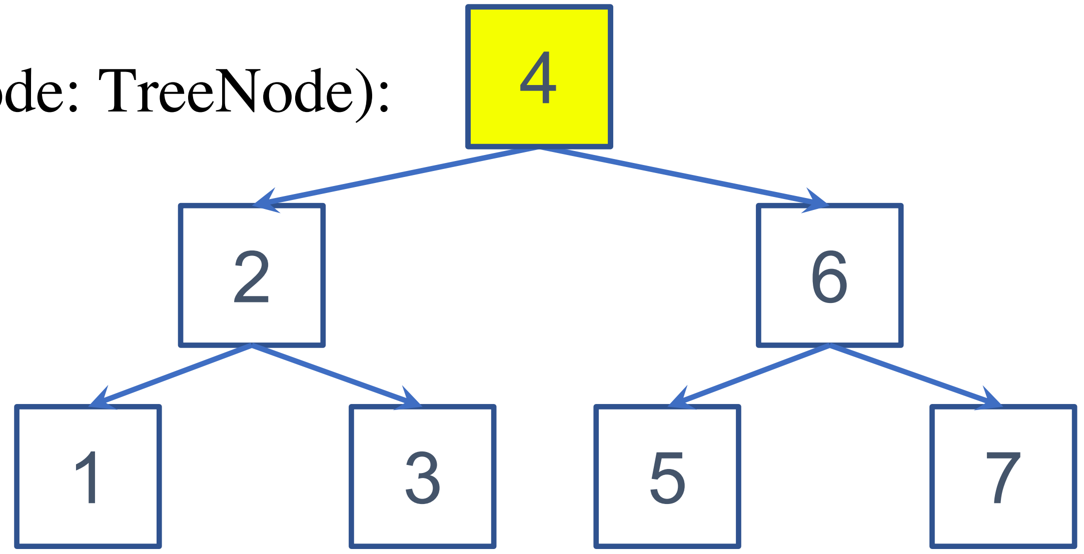
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out:

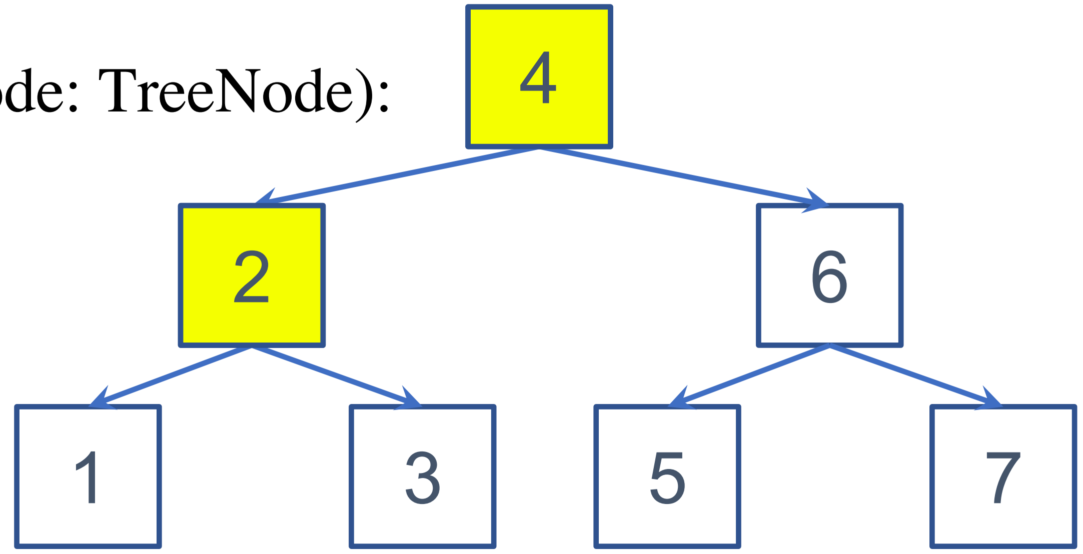
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out:

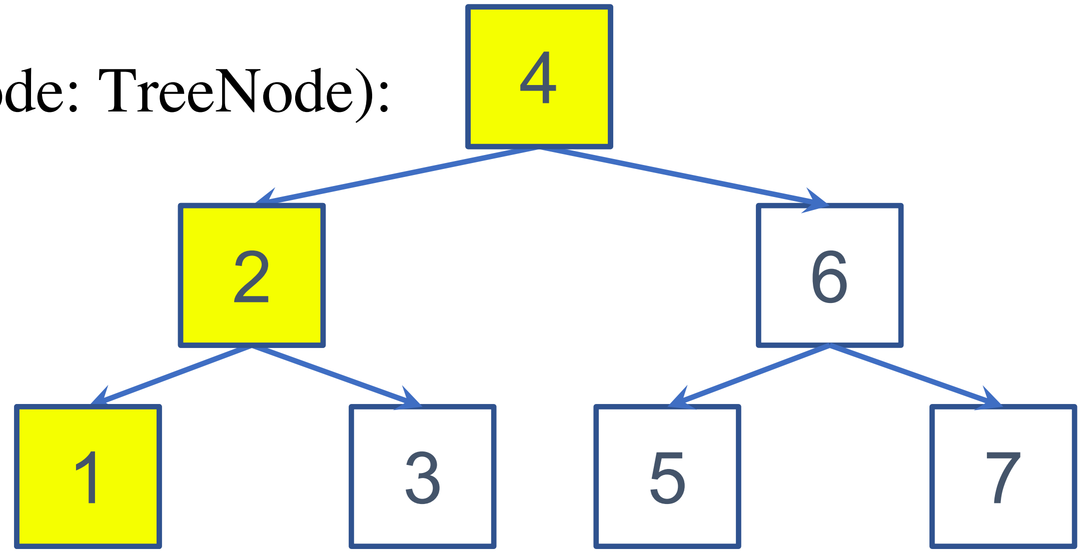
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out:

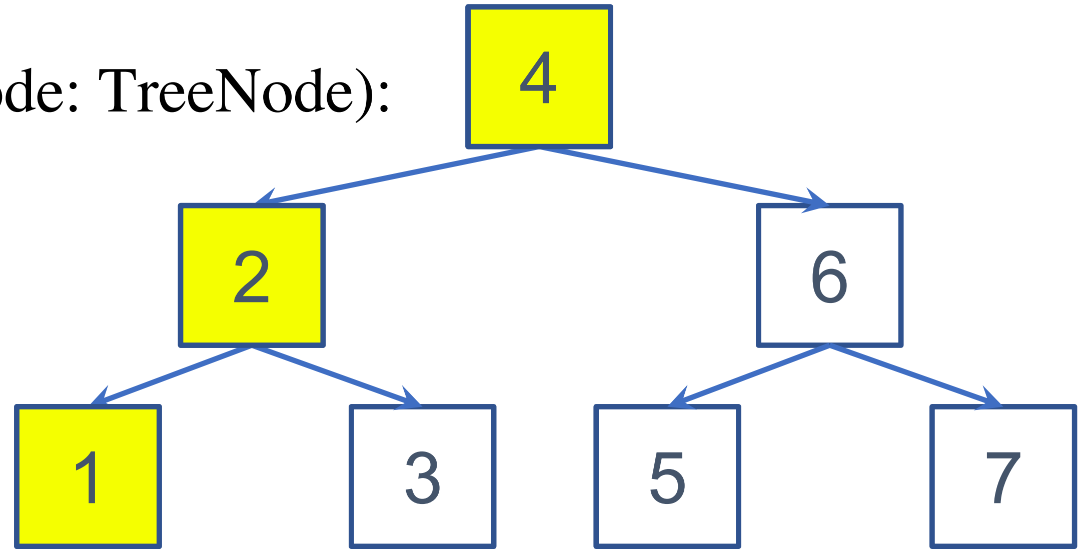
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out: 1

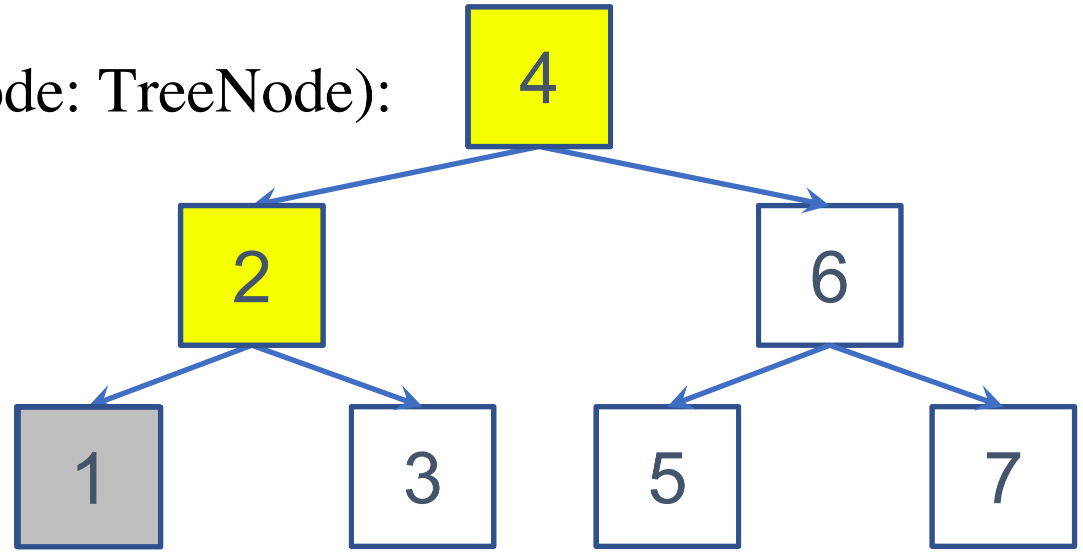
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



```
Out: 1
```

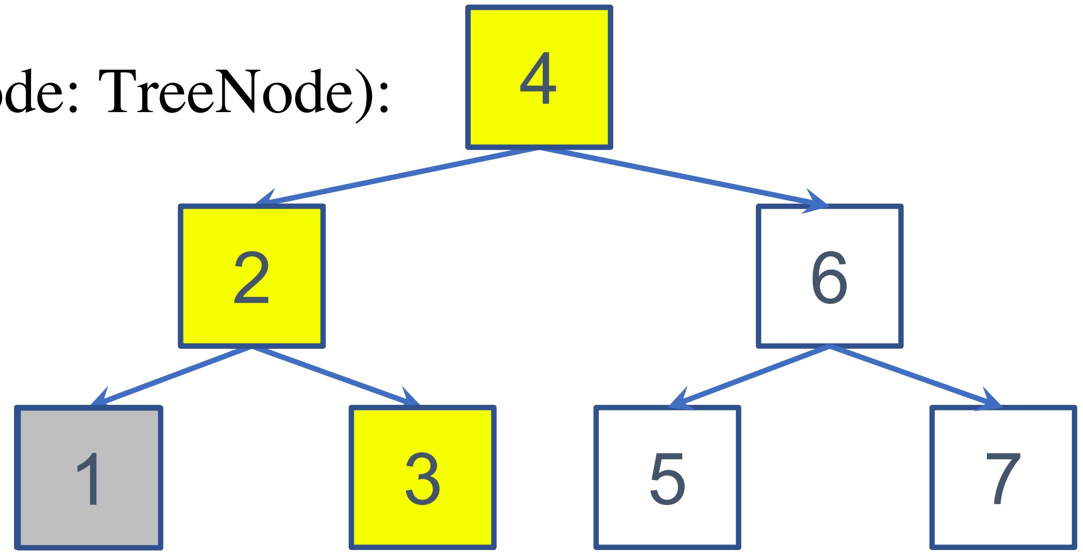
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out: 1

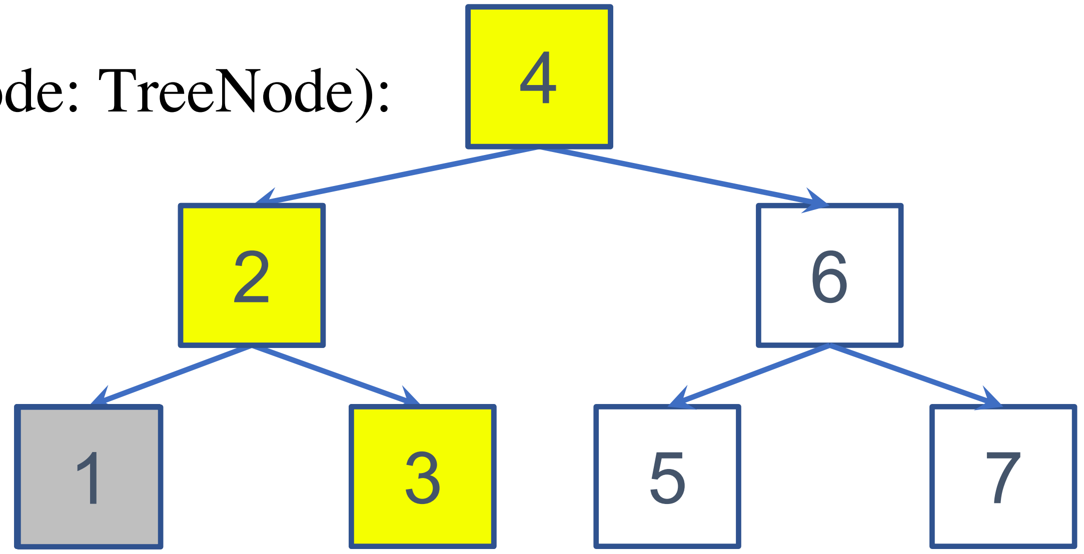
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out: 1 3

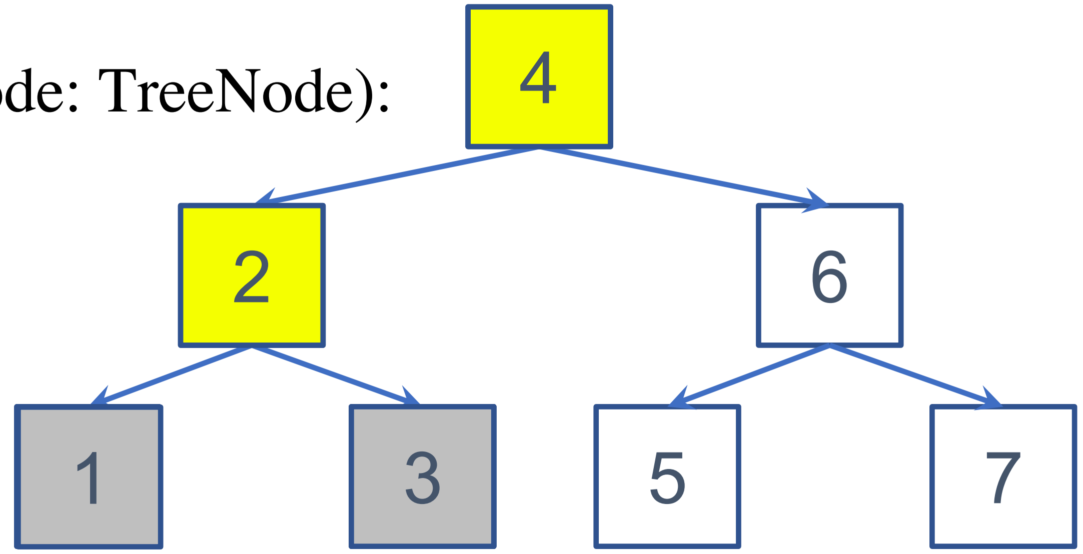
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
        def visit(self, node: TreeNode):
            print(node.val)

        def __DFT_postorderHelp(self, curNode: TreeNode):
            if curNode == None:
                return
            for i in range(len(curNode.child)):
                self.__DFT_postorderHelp(curNode.child[i])
            self.visit(curNode)

        def DFT_postorder(self):
            self.__DFT_postorderHelp(self.root)
```



```
Out: 1 3
```

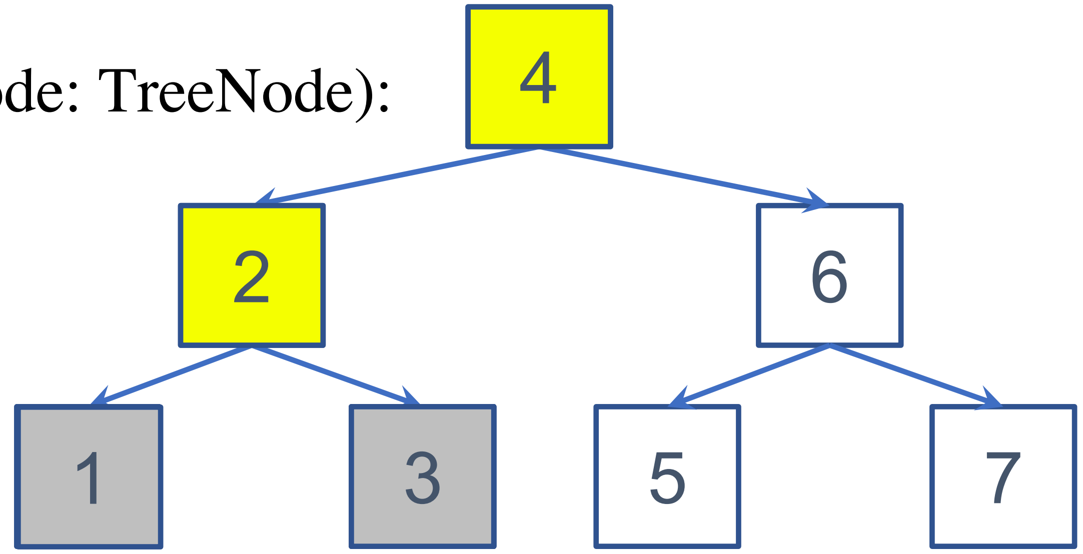
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



```
Out: 1 3 2
```

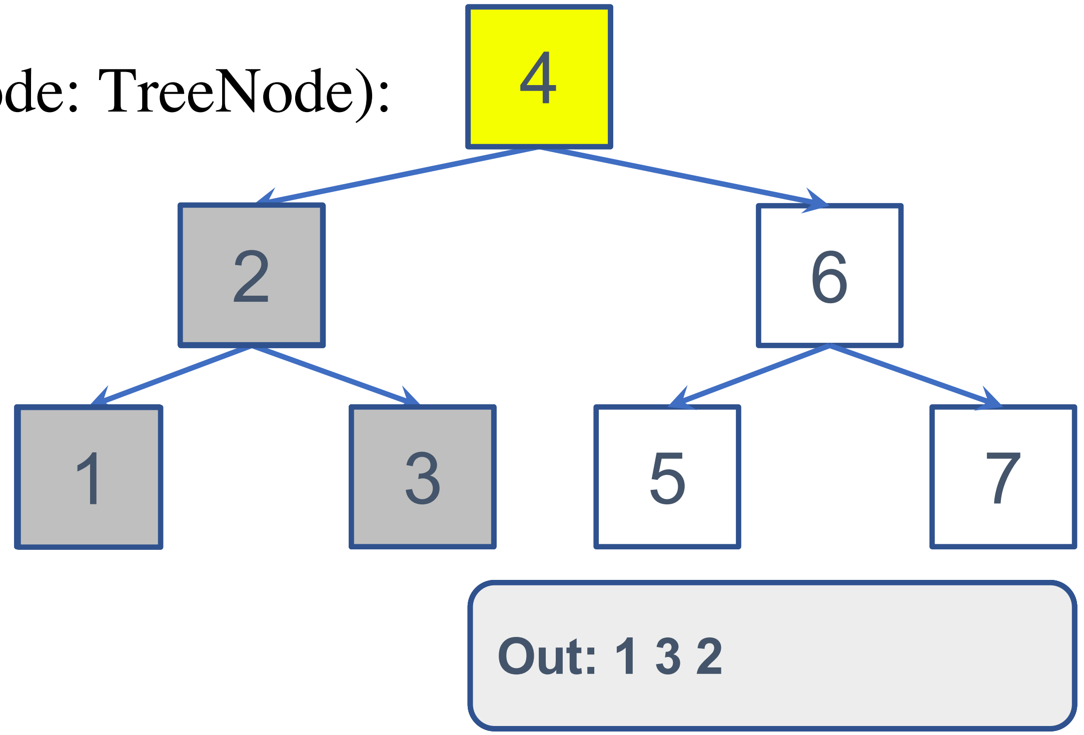
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



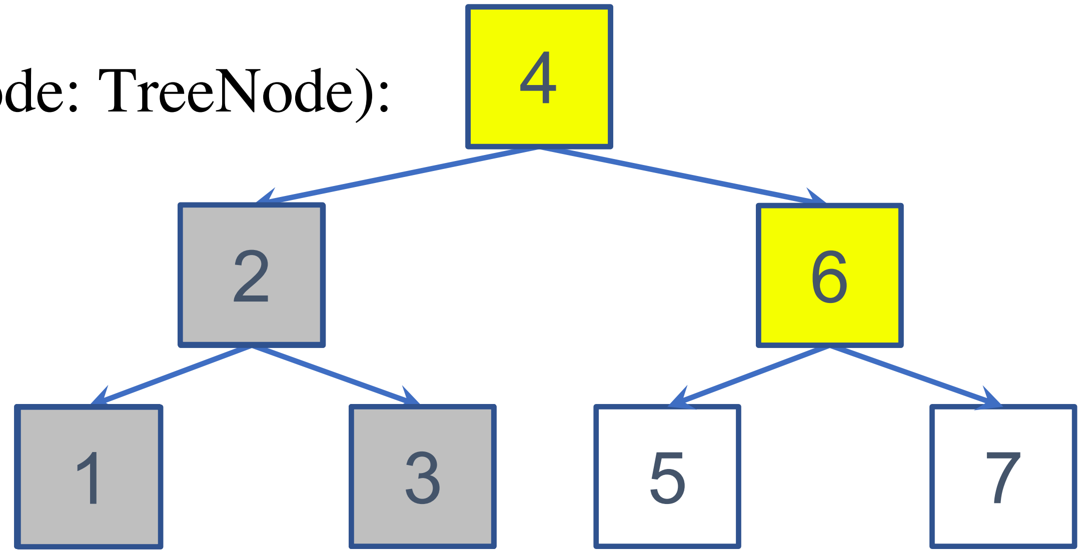
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out: 1 3 2

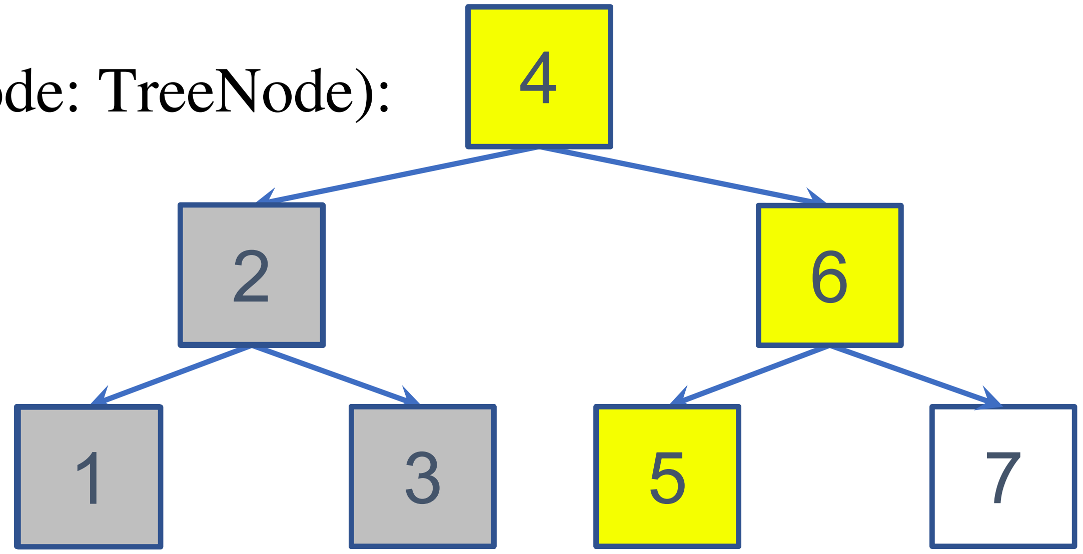
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out: 1 3 2

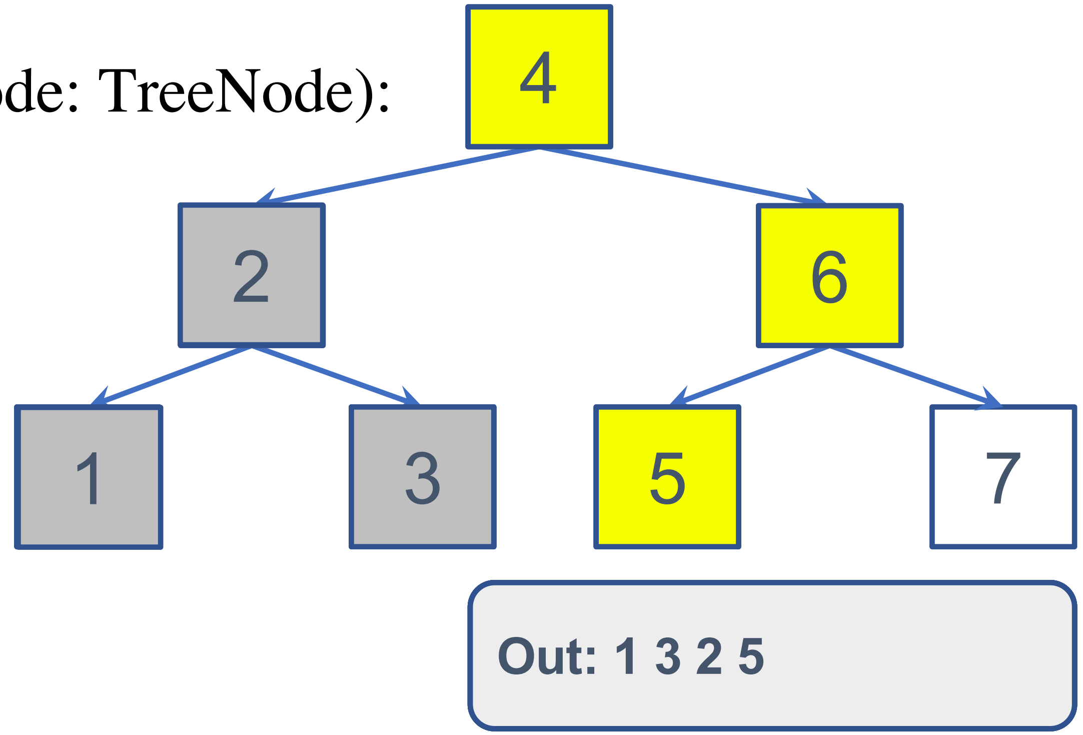
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



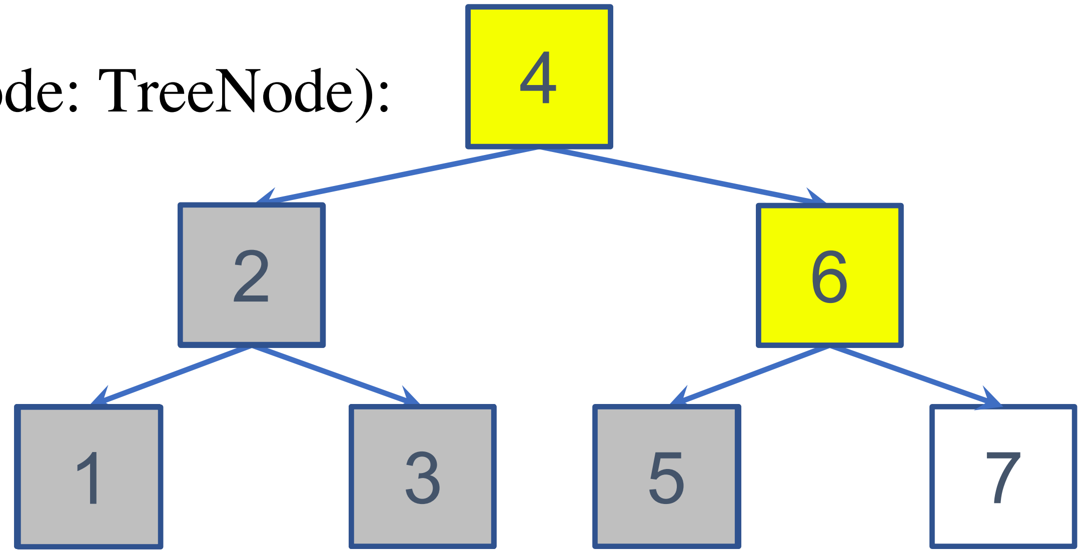
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



```
Out: 1 3 2 5
```

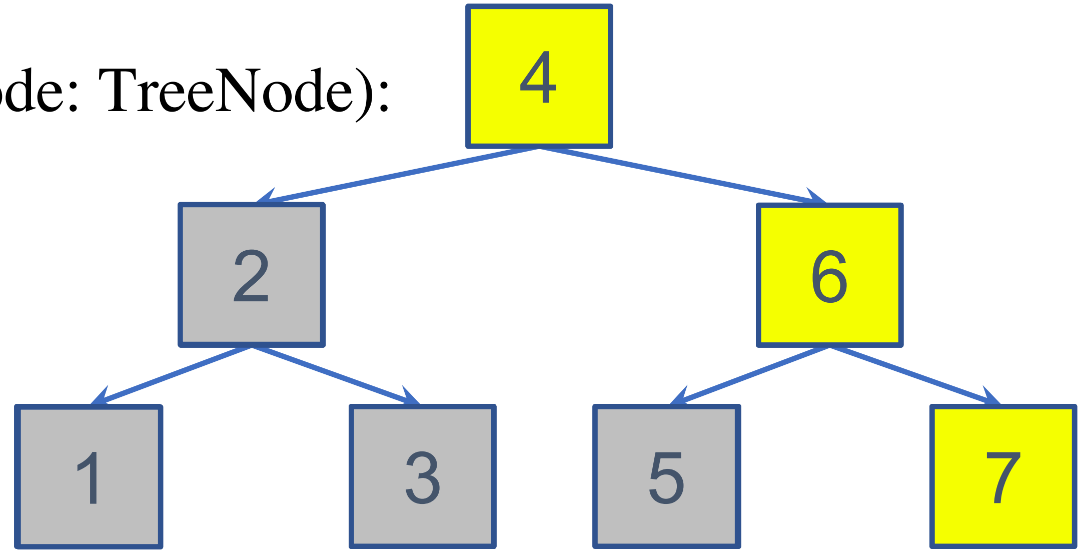
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out: 1 3 2 5

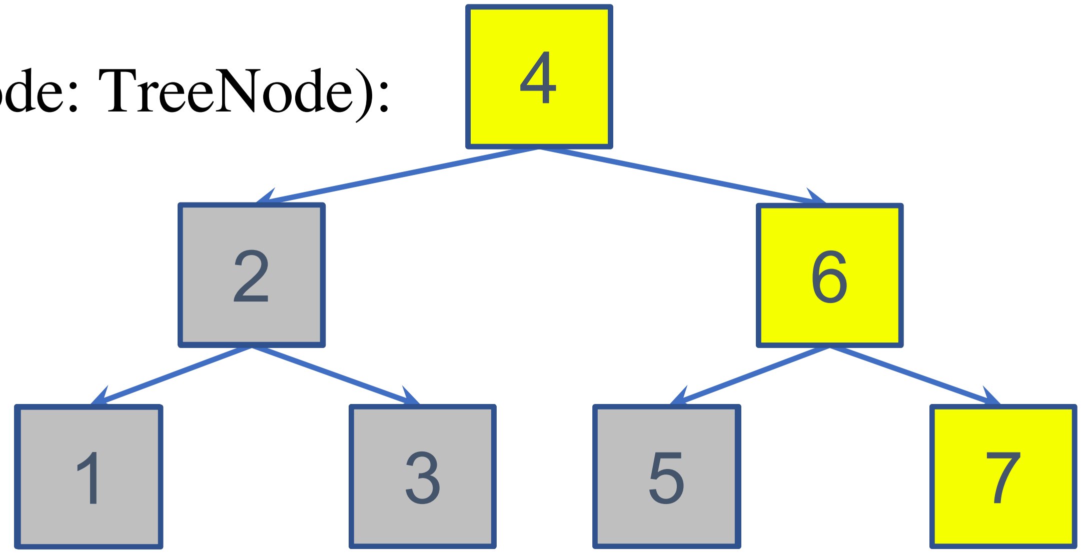
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



```
Out: 1 3 2 5 7
```

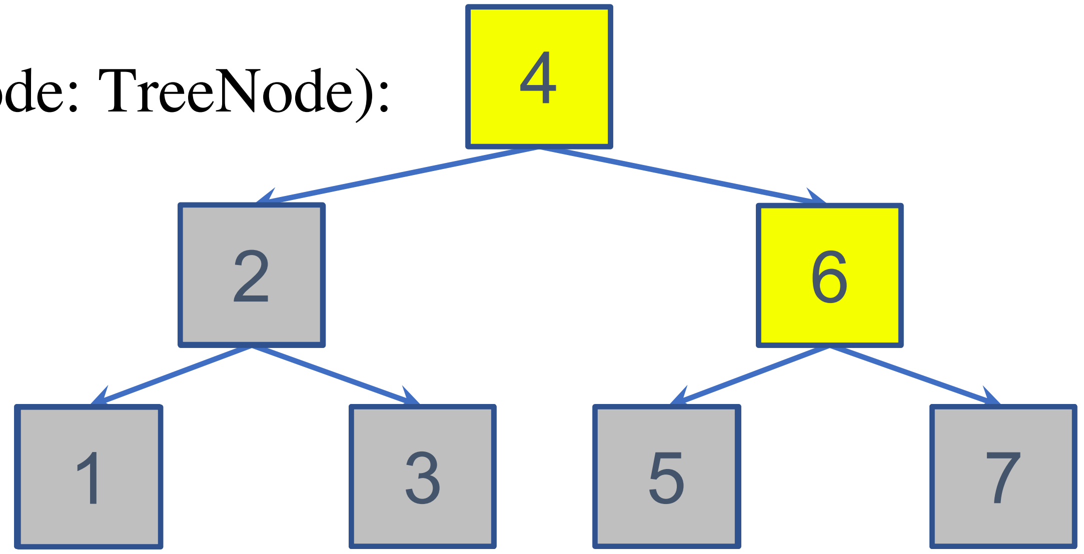
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out: 1 3 2 5 7

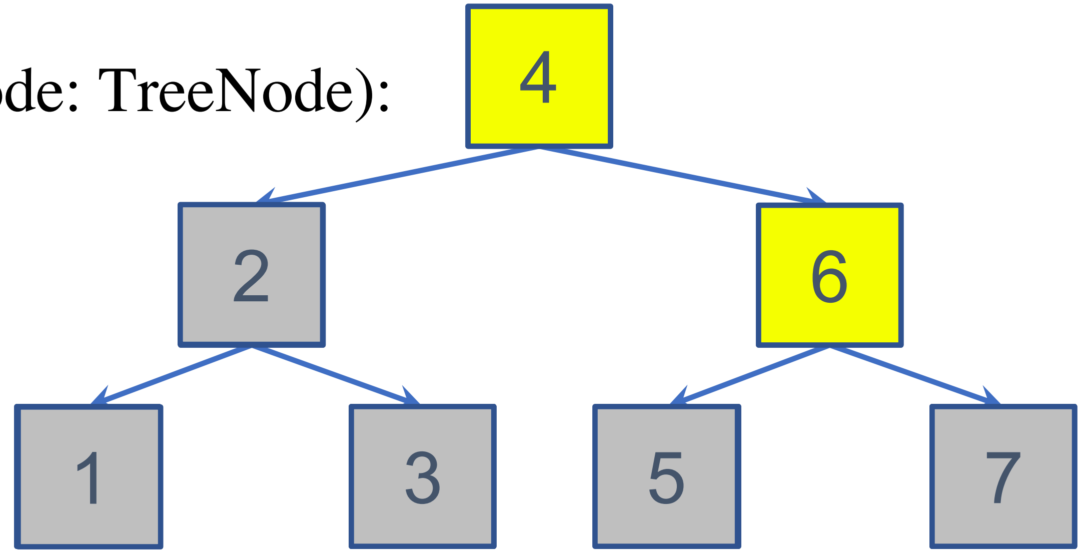
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out: 1 3 2 5 7 6

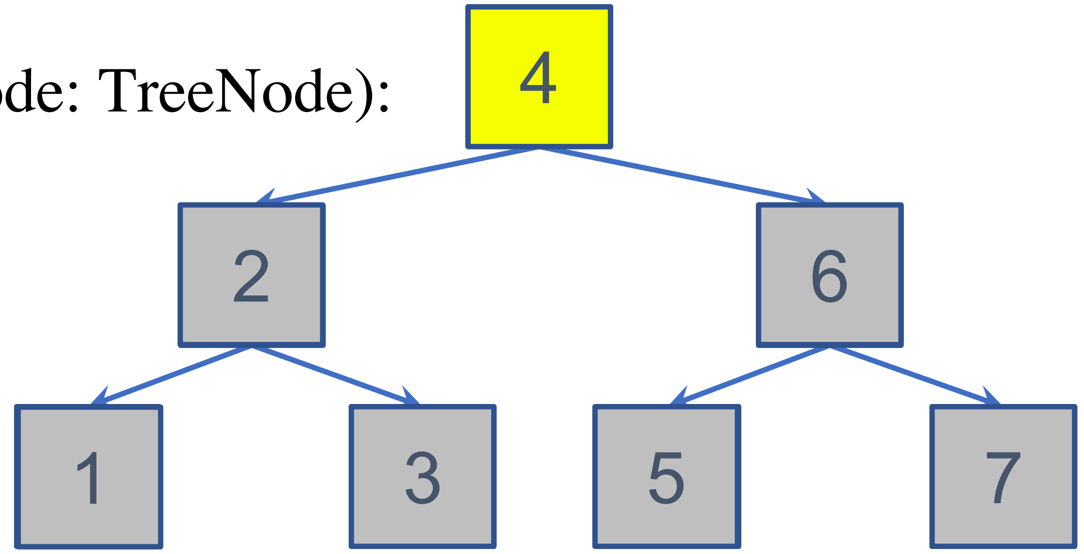
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



```
Out: 1 3 2 5 7 6
```

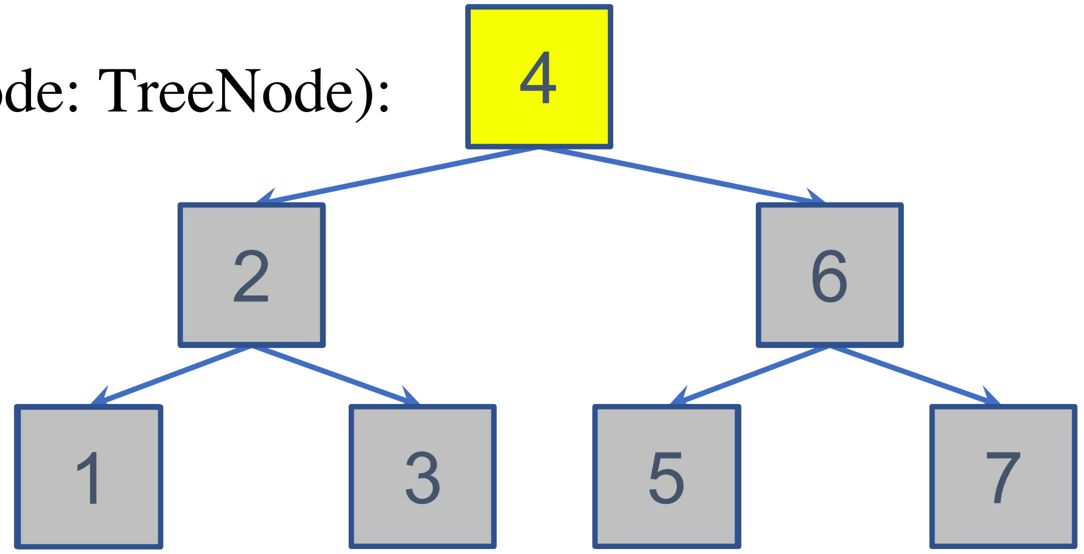
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out: 1 3 2 5 7 6 4

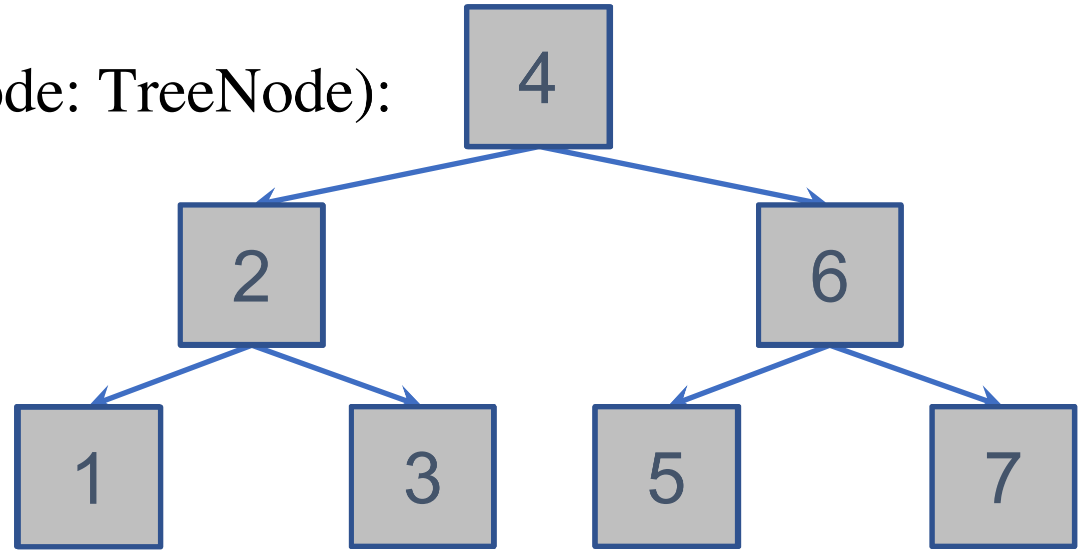
Depth First Traversals – Postorder

- Visit a node **after** traversing its children from left to right

```
class Tree():
    def visit(self, node: TreeNode):
        print(node.val)

    def __DFT_postorderHelp(self, curNode: TreeNode):
        if curNode == None:
            return
        for i in range(len(curNode.child)):
            self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode)

    def DFT_postorder(self):
        self.__DFT_postorderHelp(self.root)
```



Out: 1 3 2 5 7 6 4

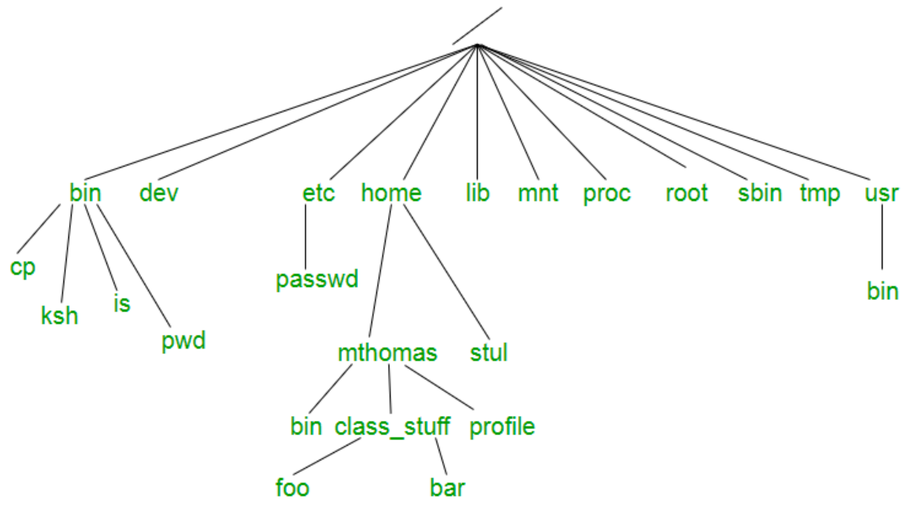
Depth First Traversals – Postorder

- **Application**: File size calculation

```
class Tree():
    def visit(self, node: TreeNode, size: float) -> None:
        node.val += size

    def __DFT_postorderHelp(curNode: TreeNode) -> float:
        if not curNode:
            return 0
        subSize = 0
        for i in range(len(curNode.child)):
            subSize += self.__DFT_postorderHelp(curNode.child[i])
        self.visit(curNode, subSize)
        return curNode.val

    def DFT_postorder(self) -> float:
        return self.__DFT_postorderHelp(self.root)
```



Summary

Summary

- Breadth-first traversal
  - Implementation using FIFO queue (deque in Python)

- Depth-first traversal
  - Implementation using recursion (or LIFO stack – also using deque in Python)
  - Three types for different purposes
    - Preorder
    - Inorder
    - Postorder

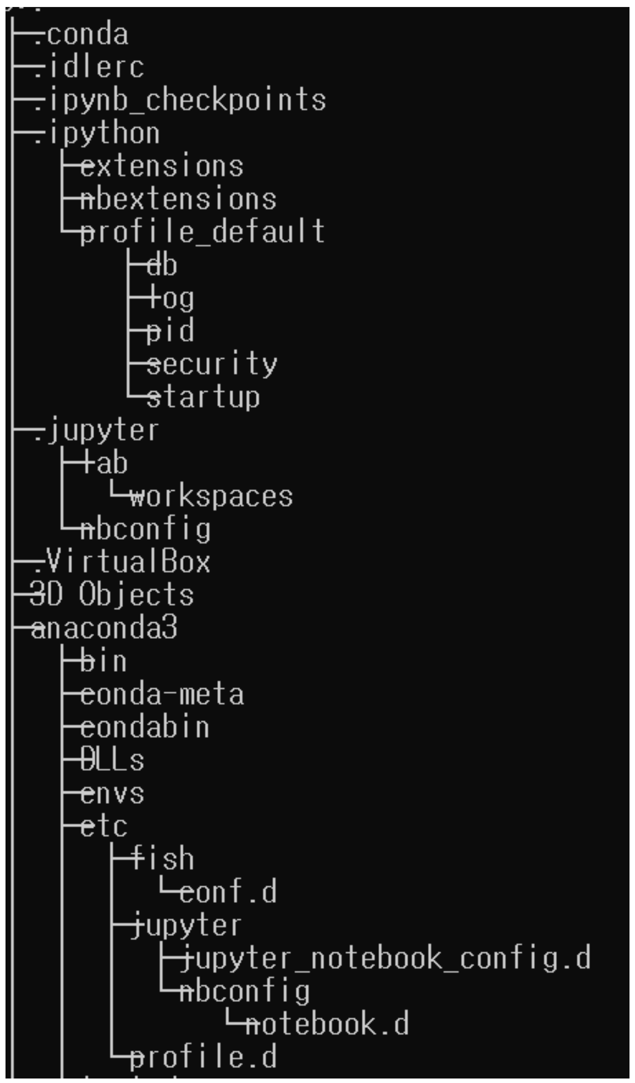
Practice 9

Trees

Practice Problem

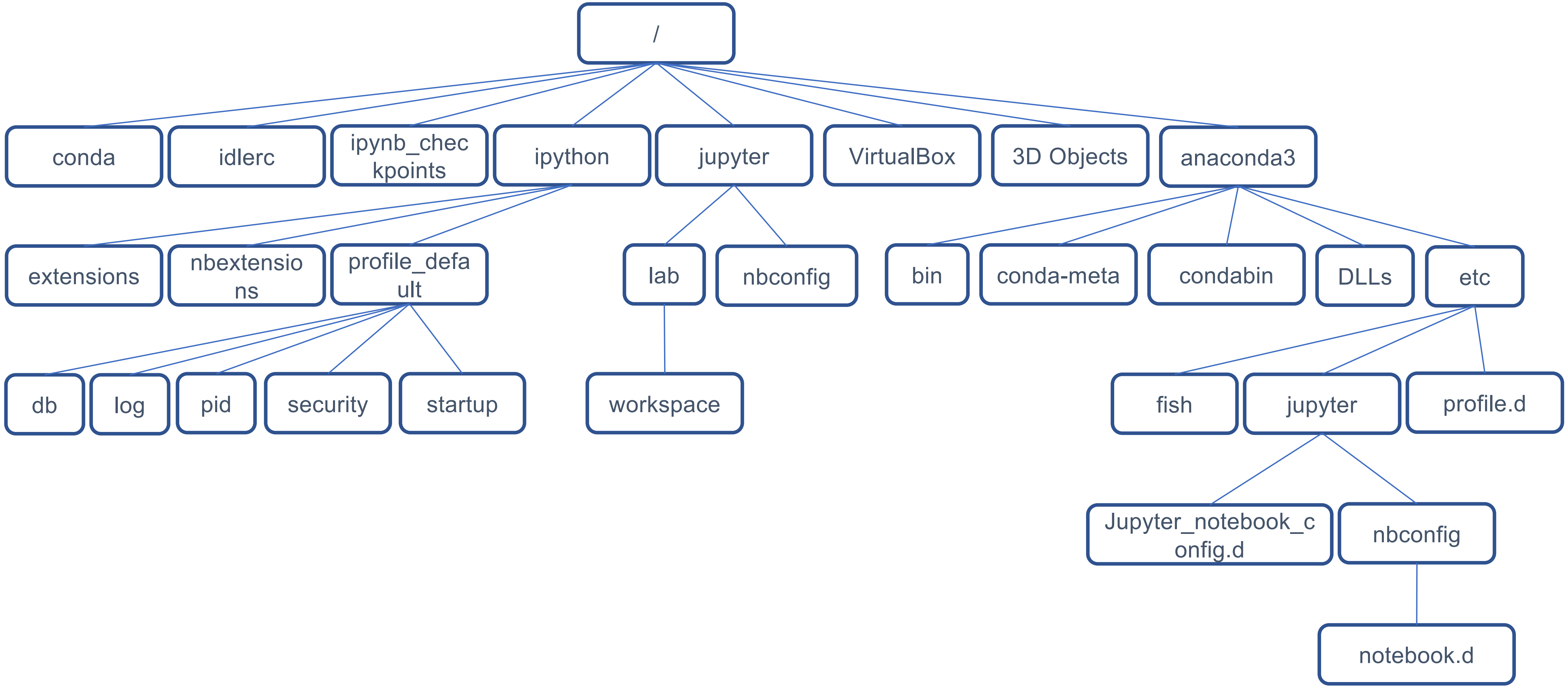
- Implement a preorder traversal for directory listing

- Each directory name is stored in a TreeNode below
  - class TreeNode():
  - def __init__(self, s: str, k: int):
  - self.name = s
  - self.ary = k
  - self.child = [None]*k



```
├─.conda
├─.idlerc
├─.ipynb_checkpoints
├─.ipython
│  ├─extensions
│  ├─nbextensions
│  └─profile_default
│      ├─db
│      ├─log
│      ├─pid
│      ├─security
│      └─startup
├─.jupyter
│  ├─lab
│  │  └─workspaces
│  └─nbconfig
├─.VirtualBox
├─3D Objects
├─anaconda3
│  ├─bin
│  ├─conda-meta
│  ├─condabin
│  ├─DLLs
│  ├─envs
│  ├─etc
│  │  ├─fish
│  │  │  └─conf.d
│  │  ├─jupyter
│  │  │  ├─jupyter_notebook_config.d
│  │  │  └─nbconfig
│  │  │      └─notebook.d
│  │  └─profile.d
```

Practice Problem – The Tree You Have



Practice Problem – The Ouput You Should See

```
 /
-- conda
-- idlerc
-- ipynb_checkpoints
-- ipython
---- extensions
---- nbextensions
---- profile_default
------ db
------ log
------ pid
------ security
------ startup
-- jupyter
---- lab
------ workspace
---- nbconfig
-- VirtualBox
-- 3D Objects
-- anaconda3
---- bin
---- conda-meta
---- condabin
---- DLLs
---- etc
------ fish
-------- conf.d
------ jupyter
-------- jupyter_notebook_config.d
-------- nbconfig
---------- notebook.d
------ profile.d
```

Q&A

Any questions?

Thanks!