Pythonda Tree Data Structure

Presented by: Alimardon Boqijonov

Dars Tartibi

Tree nima?
Treega oid rasmlar

Pythonda Tree

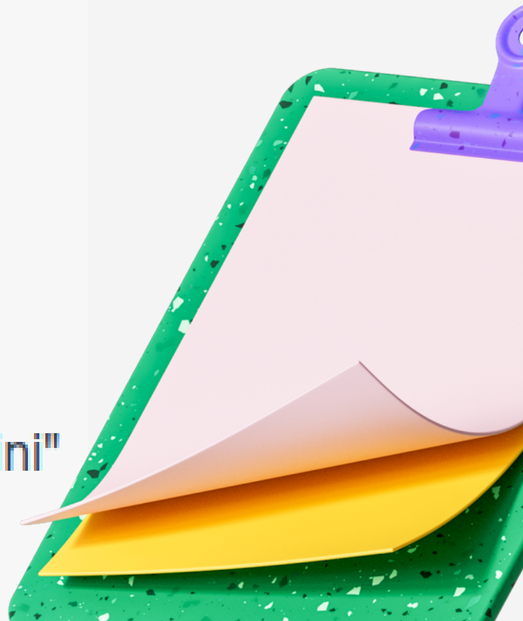
Treega oid masalalar
Uy ishi

Back to Agenda

Tree data structure

Tree (daraxt) dasturlashda ma'lumotlarni ierarxik tartibda ifodalovchi aksiyador strukturadir. Uning tuzilishi oddiy daraxtlar bilan o'xshash bo'lib, bir yoki bir nechta "ota-ona" elementdan (bo'limlardan) iborat bo'ladi. Har bir ota-ona elementga "farzand" elementlar bog'langan bo'lishi mumkin.

Daraxtning asosiy qismi "tosh" yoki "root" deb ataladi. Bu toshdan boshlanib, undan qaytadan bog'langan elementlar daraxtning ostida joylashadi. Daraxtdagi elementlar o'rtacha aloqalar bilan bir-biriga bog'langan va bir-biridan farq qiluvchi tarkibiy bog'lanishlardan iborat bo'ladi. Daraxtning uzunligi (balandligi) "daraxtning yuqori darajasini" bildiradi.



Tree data structure (2)

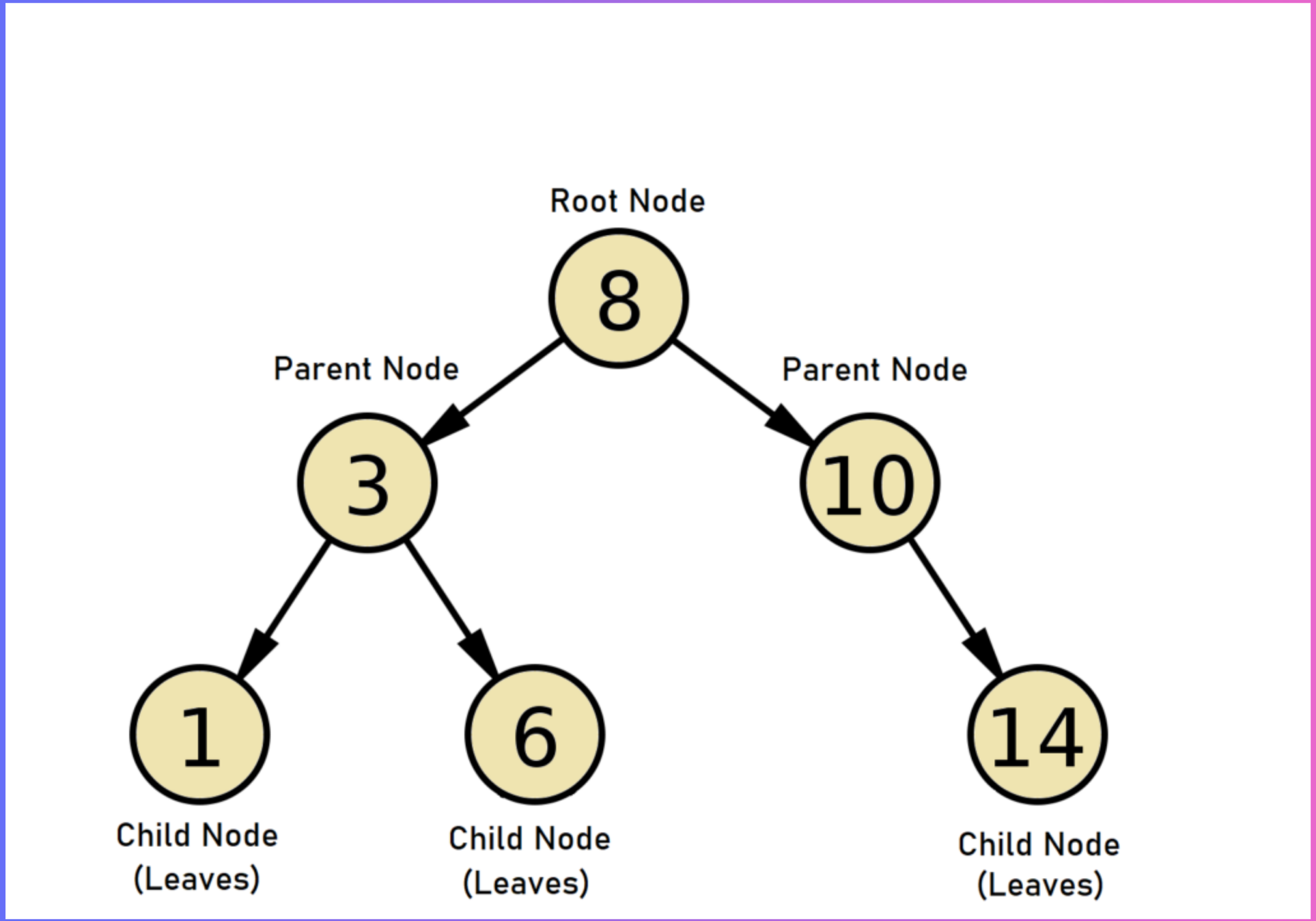
Daraxt strukturasining katta qismini "ota-ona" va "farzand" bog'lanishlar bildiradi. "Ota-ona" elementga "farzand" elementlarni qo'shish, o'chirish va o'zgartirish imkonini beradi. Daraxtda "ota-ona" va "farzand" bog'lanishlar orqali ma'lumotlarni tashuvchi va muhokamaga imkon beradi.

Daraxtning har bir elementi (tosh, farzand) o'z ichiga o'zgartiriladigan ma'lumotni saqlash uchun yoki daraxtning aloqasini ifodalash uchun yaratilgan obyektlar bo'lishi mumkin. Daraxtda har bir elementga oid ma'lumotlar va bog'lanishlar o'zgartirilishi mumkin.

Tree data structure (3)

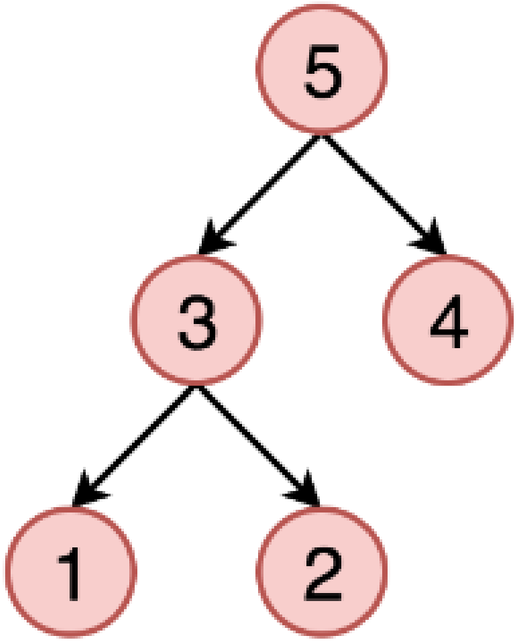
Daraxtlar bir qator muhim dasturlash maqsadlarida foydalaniladi, masalan, ma'lumotlarni tashuvchi va tahlil qiluvchi algoritmalarni boshqarishda, ma'lumot bazalarida, tizimlarda, nazariy jismoniy modellar va boshqalarda.

Daraxtlar ko'plab dasturlash tillarida mavjud bo'lib, har bir tillarning o'ziga xos sintaksisi va qo'llanmalariga ega. Masalan, Python, Java, C++, JavaScript tillarida daraxtlar yaratish va ular bilan ishlashning o'zgartirilishi mumkin.


Root Node
8
Parent Node
Parent Node
3
10
1
6
14
Child Node
(Leaves)
Child Node
(Leaves)
Child Node
(Leaves)

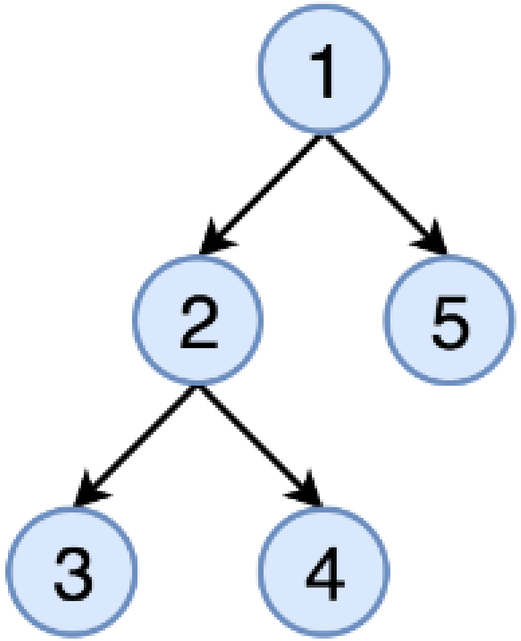
DFS Postorder

Bottom -> Top
Left -> Right



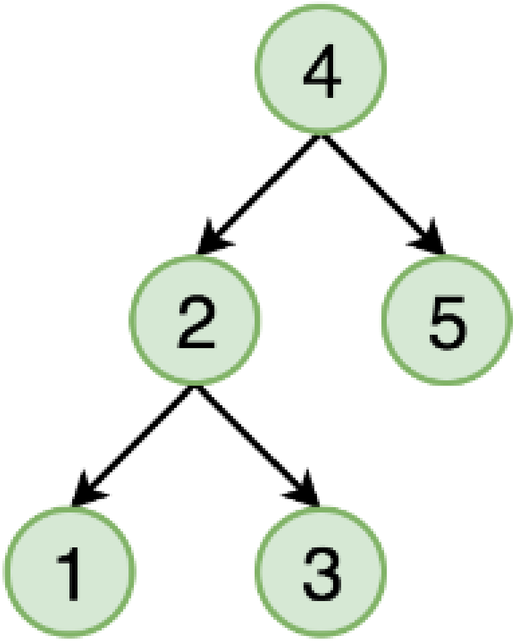
DFS Preorder

Top -> Bottom
Left -> Right



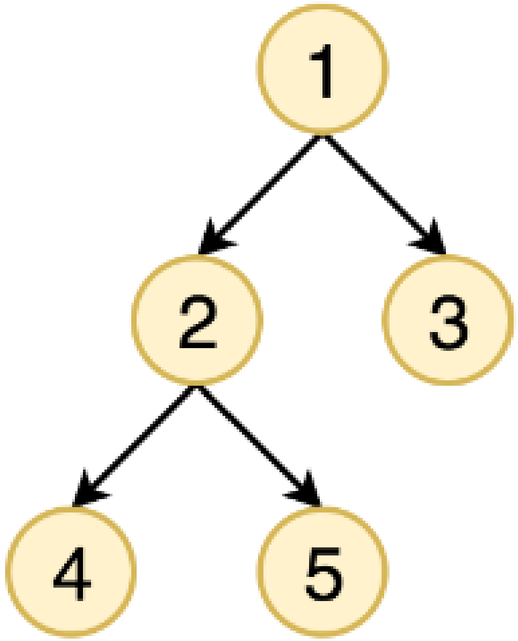
DFS Inorder

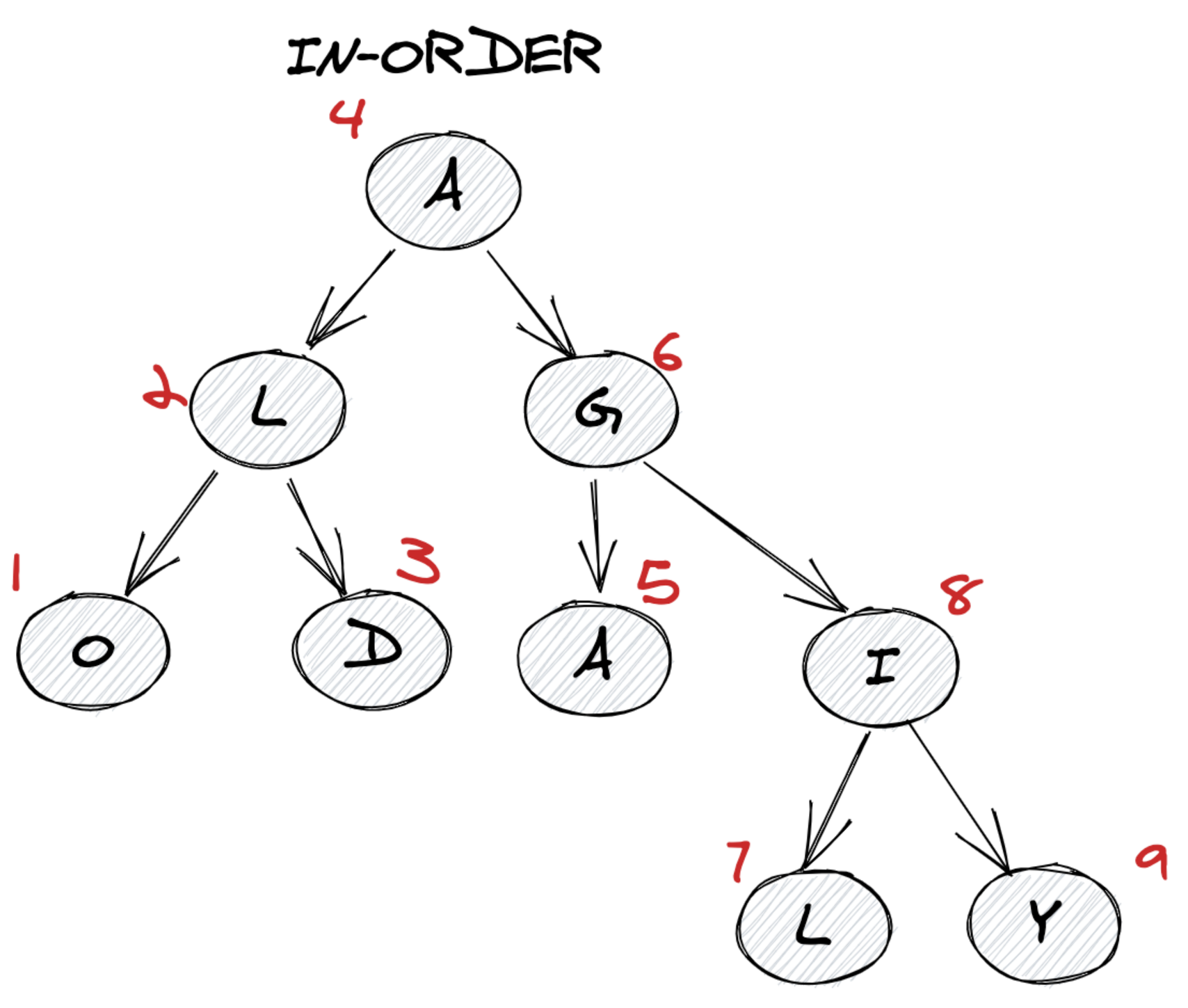
Left -> Node -> Right



BFS

Left -> Right
Top -> Bottom




IN-ORDER
4
A
2
L
6
G
1
O
3
D
5
A
8
I
7
L
9
Y

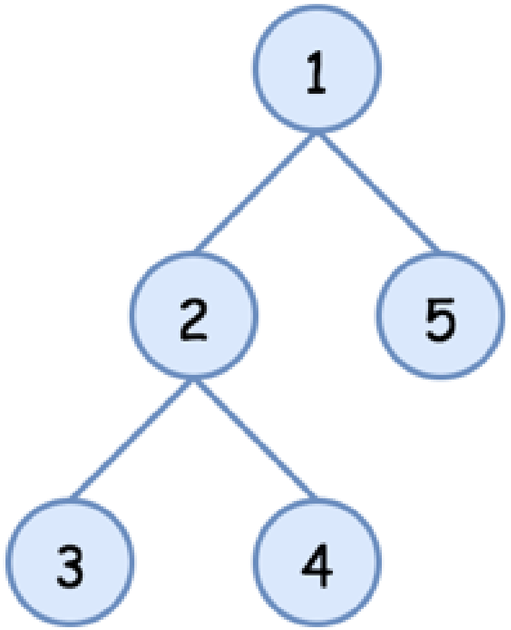
DFS Preorder
Node -> Left -> Right

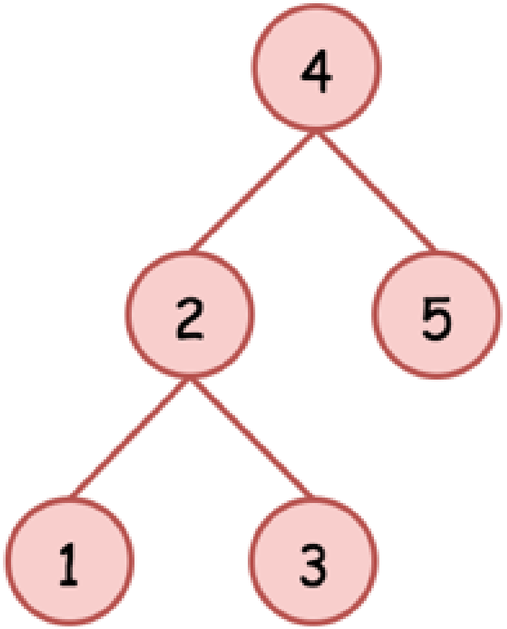
DFS Inorder
Left -> Node -> Right

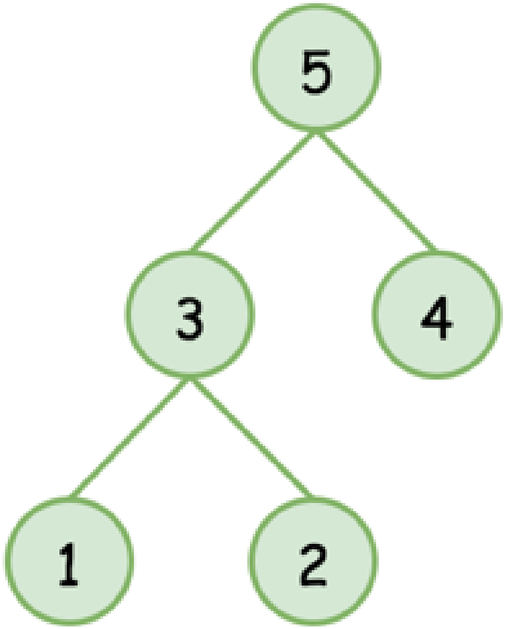
DFS Postorder
Left -> Right -> Node

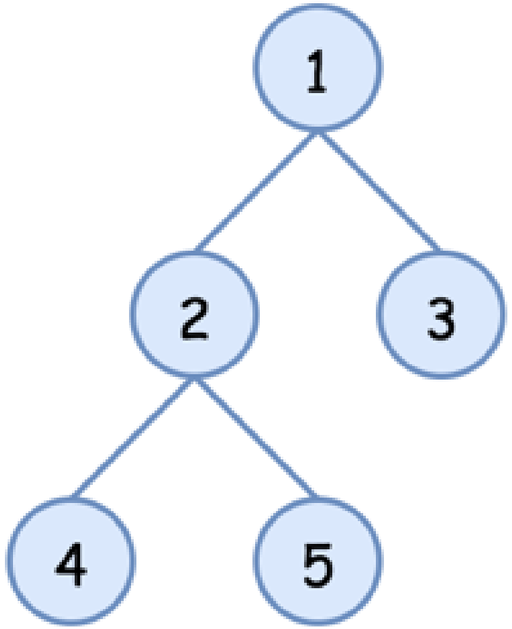
BFS
Node -> Left -> Right

Traversal = [1, 2, 3, 4, 5]









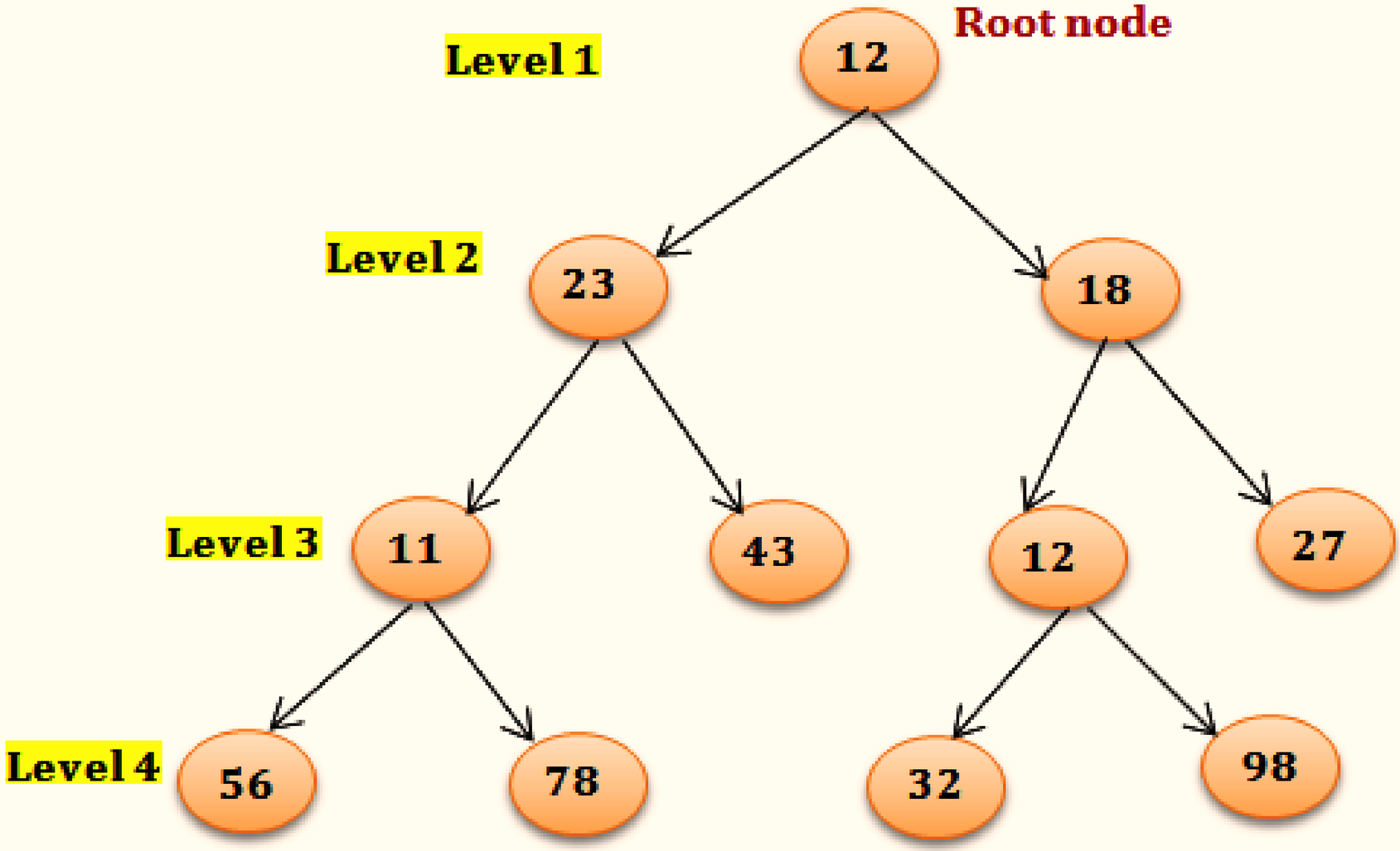
[root.val] +
preorder(root.left) +
preorder(root.right)
if root else []

inorder(root.left) +
[root.val] +
inorder(root.right)
if root else []

postorder(root.left) +
postorder(root.right) +
[root.val]
if root else []

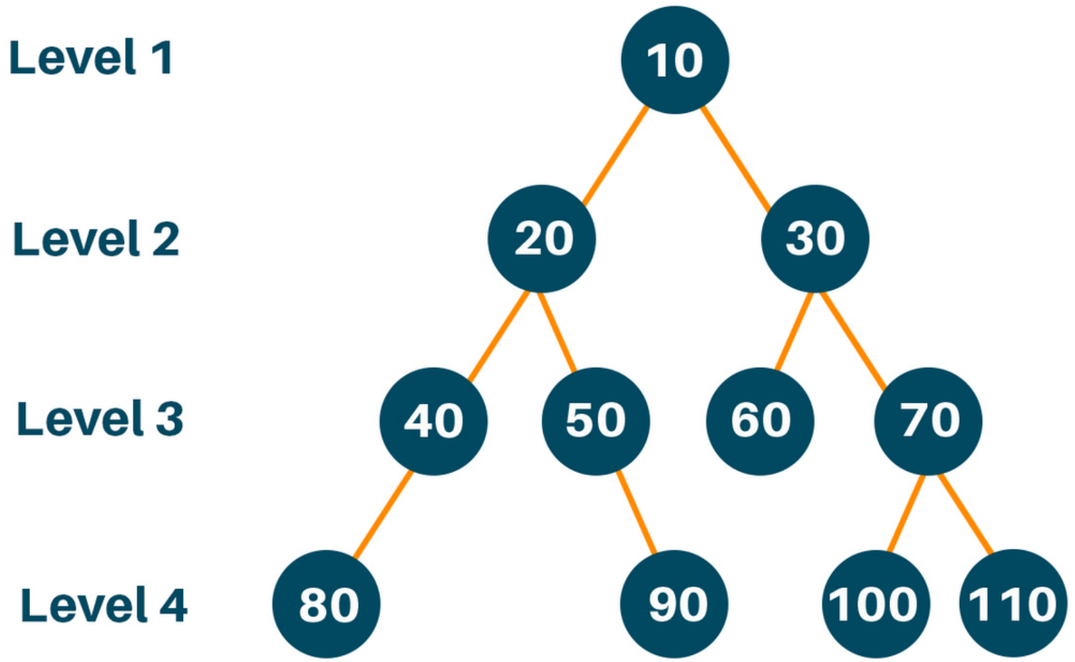
iterations
with the queue

Binary tree



python™

Level Order Traversal of a Binary Tree



InOrder(root) visits nodes in the following order:

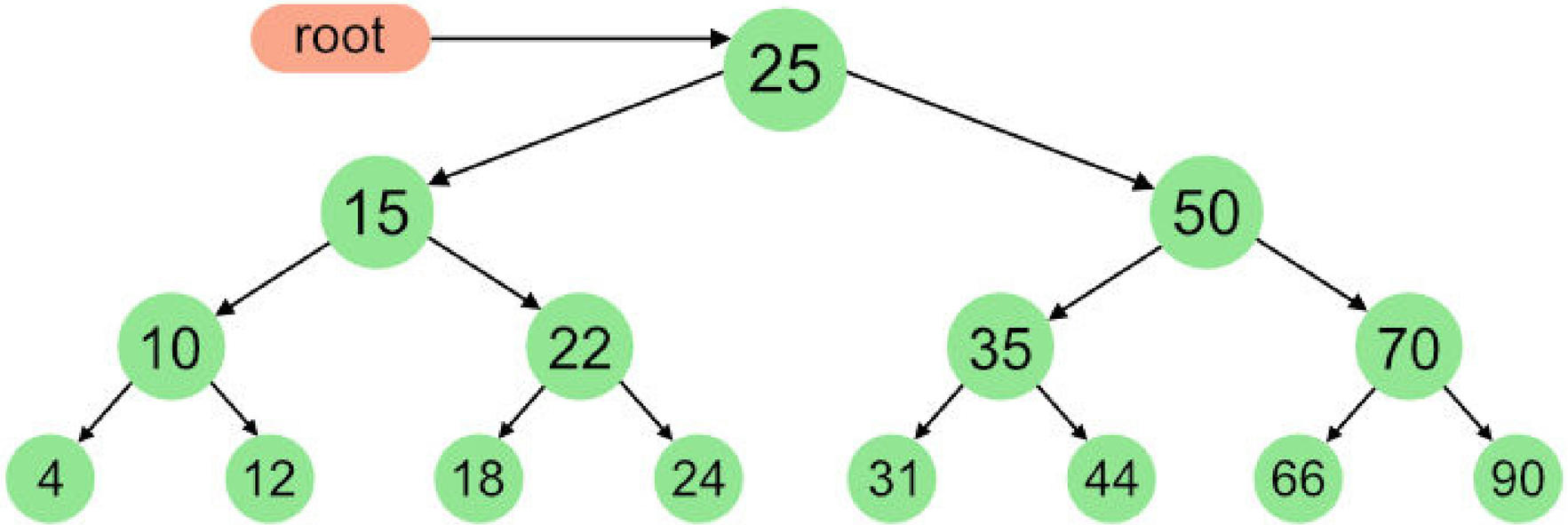
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:

25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:

4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



python Copy code

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

def insert(root, data):
    if root is None:
        return Node(data)
    else:
        if data < root.data:
            root.left = insert(root.left, data)
        else:
            root.right = insert(root.right, data)
    return root

def inorder_traversal(root):
    if root:
        inorder_traversal(root.left)
        print(root.data)
        inorder_traversal(root.right)
```

```
# Daraxtni yaratish
root = None
root = insert(root, 5)
root = insert(root, 3)
root = insert(root, 8)
root = insert(root, 1)
root = insert(root, 4)

# Daraxtni o'qish
inorder_traversal(root)
```

Ushbu kodingda `Node` nomli klass yaratiladi, unda har bir element (tosh) uchun `data`, `left` va `right` o'zgaruvchilar mavjud. `insert` funksiya yordamida yangi elementlar daraxtga qo'shiladi. `inorder_traversal` funksiyasi esa daraxtning ma'lumotlarini `inorder` tartibida (ortiqcha o'qish tartibida) chop etadi.

Kod natijasida quyidagi chiqish hosil bo'ladi:

```
1
3
4
5
8
```

Bu quyidagi daraxtni ifodalaydi:

```markdown
      5
    /   \
  3      8
 / \
1   4
```

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

def preorder_traversal(root):
    if root:
        print(root.data)
        preorder_traversal(root.left)
        preorder_traversal(root.right)

def inorder_traversal(root):
    if root:
        inorder_traversal(root.left)
        print(root.data)
        inorder_traversal(root.right)

def postorder_traversal(root):
    if root:
        postorder_traversal(root.left)
        postorder_traversal(root.right)
        print(root.data)
```

```python
# Daraxtni yaratish
root = Node(5)
root.left = Node(3)
root.right = Node(8)
root.left.left = Node(1)
root.left.right = Node(4)

# Daraxtni tarqatishlar
print("Preorder traversal:")
preorder_traversal(root)

print("Inorder traversal:")
inorder_traversal(root)

print("Postorder traversal:")
postorder_traversal(root)
```

```
Preorder traversal:
5
3
1
4
8
Inorder traversal:
1
3
4
5
8
Postorder traversal:
1
4
3
8
5
```

python

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Ikkilik daraxtni yaratish
def create_binary_tree():
    root = Node(1)
    root.left = Node(2)
    root.right = Node(3)
    root.left.left = Node(4)
    root.left.right = Node(5)
    return root
```

```python
# Ikkilik daraxtni inorder tartibida tarq
def inorder_traversal(node):
    if node is not None:
        inorder_traversal(node.left)
        print(node.data)
        inorder_traversal(node.right)

# Ikkilik daraxtni preorder tartibida tar
def preorder_traversal(node):
    if node is not None:
        print(node.data)
        preorder_traversal(node.left)
        preorder_traversal(node.right)
```

```python
# Ikkilik daraxtni postorder tartibida tarqatish
def postorder_traversal(node):
    if node is not None:
        postorder_traversal(node.left)
        postorder_traversal(node.right)
        print(node.data)

# Ikkilik daraxtni yaratish va tarqatish
binary_tree = create_binary_tree()

print("Inorder traversal:")
inorder_traversal(binary_tree)

print("Preorder traversal:")
preorder_traversal(binary_tree)

print("Postorder traversal:")
postorder_traversal(binary_tree)
```

```
Inorder traversal:
4
2
5
1
3
Preorder traversal:
1
2
4
5
3
Postorder traversal:
4
5
2
3
1
```

Ushbu kodingda `Node` nomli klass yaratiladi, unda har bir element (tosh) uchun `data`, `left` va `right` o'zgaruvchilar mavjud. `create_binary_tree` funksiyasi yordamida ikkilik daraxt yaratiladi. `inorder_traversal`, `preorder_traversal` va `postorder_traversal` funksiyalari yordamida daraxtni tarqatishlar amalga oshiriladi.

@INFIN1TY.UZ

E'TIBORINGIZ UCHUN

RAHMAT

Presented by: Alimardon Boqijonov