Subject:

Date

① الف) غ

ب) غ ۳ حداکثر فاصله برگ تا ریشه

پ) ص درخت دودویی درختی که هرگره حداکثر ۲ فرزند دارد.

ت) غ

ث) ص تفاوت سطح برگ‌ها حداکثر یک برابر یک

ج) ص

چ) ع

ح) ص

خ) ص

د) غ

② Function Height (node):

If node is NULL:

Return 0

else:

Left Height = Height (node.Left)

Right Height = Height (node.Right)

Return (max (Left Height, Right Height) + 1

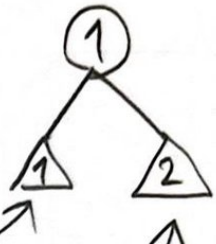
مرتبه زمانی $O(n)$ زیرا هر گره را یک بار صدا می‌کنیم.

Preorder: 1 3 6 5 7 4 2

1: Left 2: Right

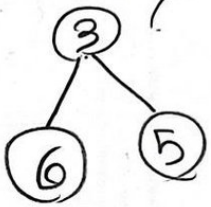
Post order: 6 5 3 4 2 7 1

1: Left 2: Right



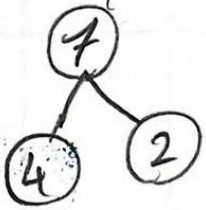
Pre 1: 3 6 5

Post 1: 6 5 3

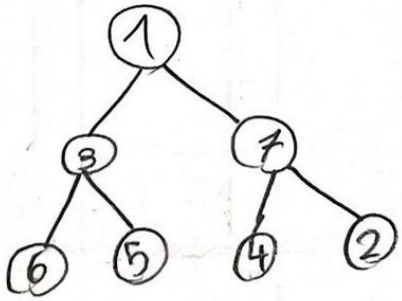


Pre 2: 7 4 2

Post 2: 4 2 7



درخت به طور کلی



(ب

Pre : 3 4 5 2 6 1 7

1: Left 2: Right

Post: 5 2 4 1 7 6 3

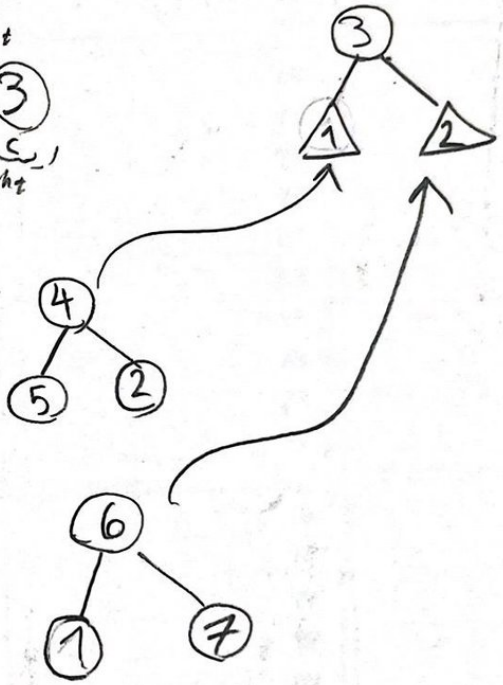
1: Left 2: Right

Pre1: 4 5 2

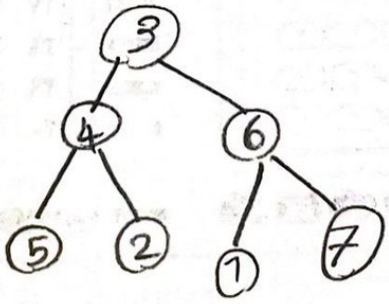
Post1: 5 2 4

Pre2: 6 1 7

Post2: 1 7 6



درخت به طور کل



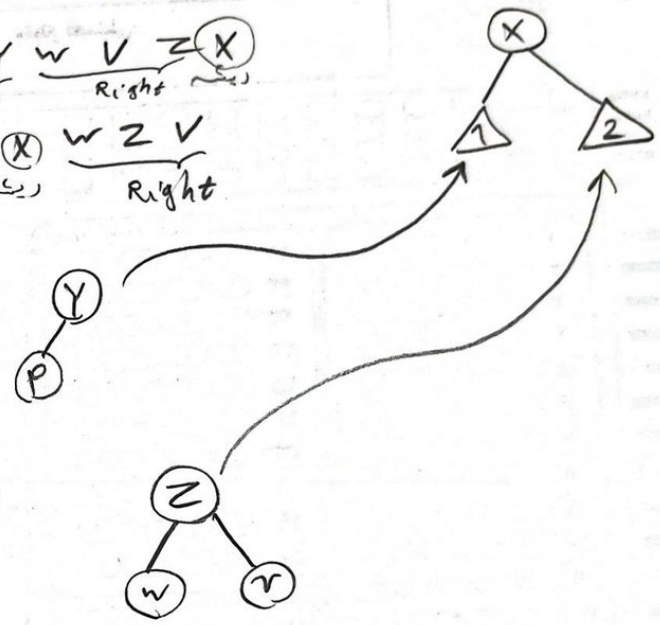
ب)

Post order: P Y w V Z X
Left Right ریشه

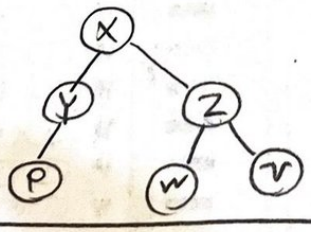
in order: P Y X w Z V
Left ریشه Right

Post 1: P Y
in 1: P Y

Post 2: w v Z
in 2: w Z v



درخت به طور کل



pre: (H) A E J I F B G C D

Left Right

in order: E A I J B F (H) G D C

Left Right

in 1: E (A) I J B F

Left Right

pre 1: (A) E J I F B

Left Right

in 1,2: I (J) B F

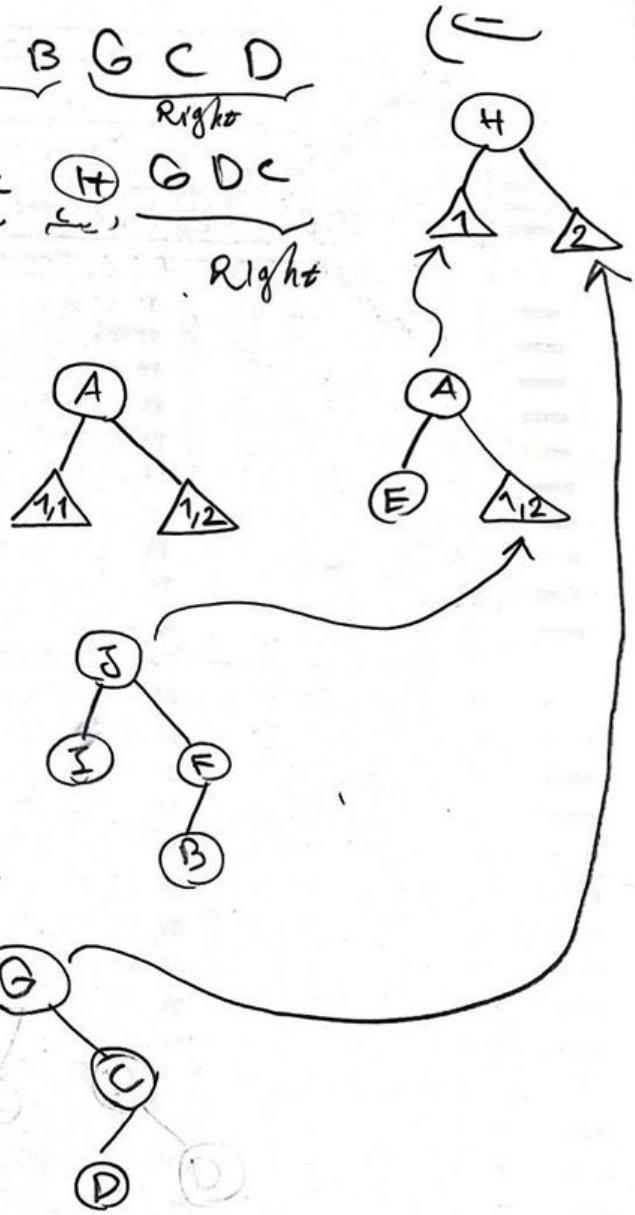
Left Right

pre 1,2: (J) I F B

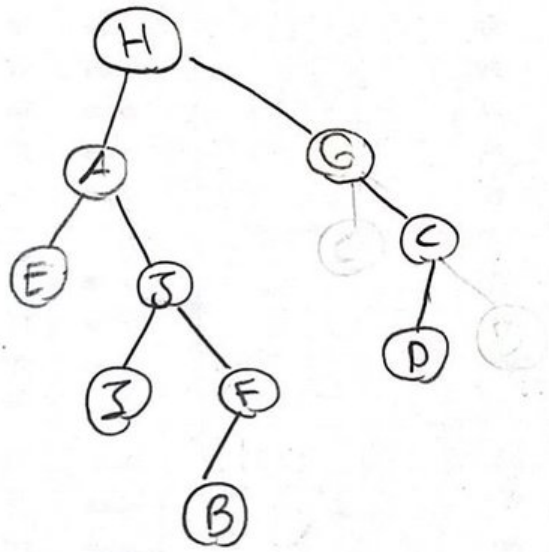
Left Right

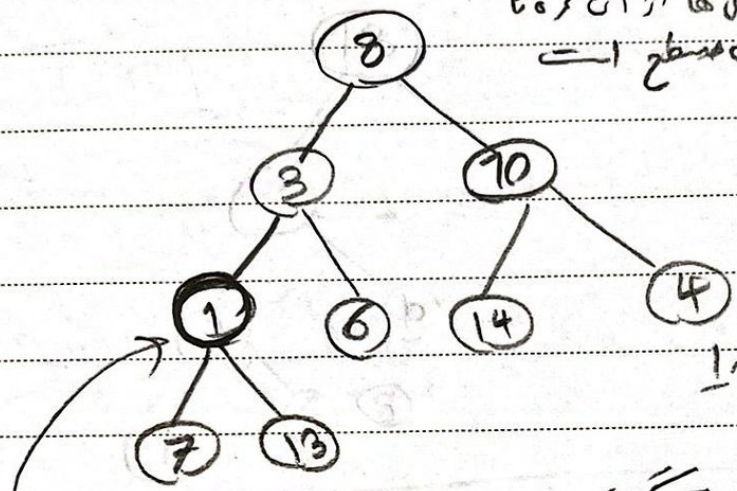
in 2: G D C

pre 2: (G) C D



درخت بطور کامل





عمق هر گره (4) تعداد یال‌ها از ریشه تا آن گره است

بیشترین تعداد یال‌ها از آن گره تا برگ پایین‌ترین سطح است

الف) عمق و ارتفاع و سطح درخت را محاسبه کنید؟

سطح گره مربر عمق گره به اضافه 1 است.

ب) تعداد برگ‌ها را مشخص کنید؟ 5 برگ

4 و 14 و 13 و 7 و 6

پ) گره‌ای که کوچکترین عمق را دارد مشخص کنید؟

	Level
depth(8) = 0 →	1
depth(3) = 1 →	2
// (10) = 1	2
// (1) = 2	3
// (6) = 2	3
// (14) = 2	3
// (4) = 2	3
// (7)(13) = 3	4

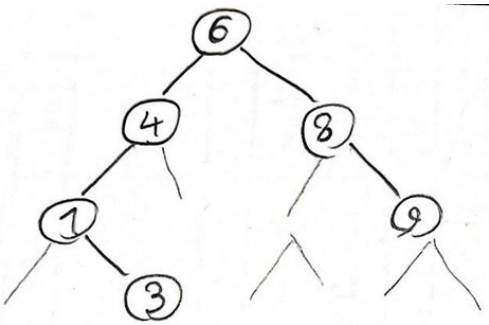
height(کل درخت) = 8 = 3

h(3) = 2

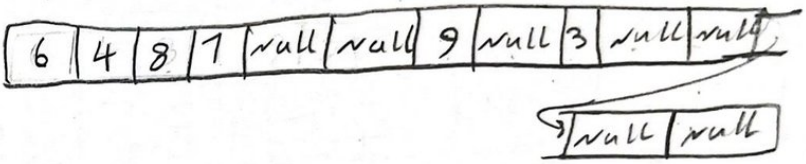
h(1) = 1

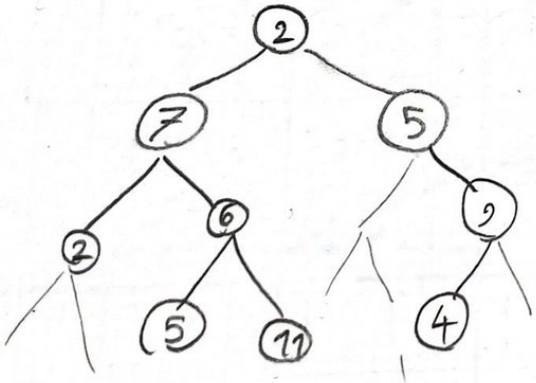
h(10) = 1

h(4, 14, 7, 13, 6) = 0

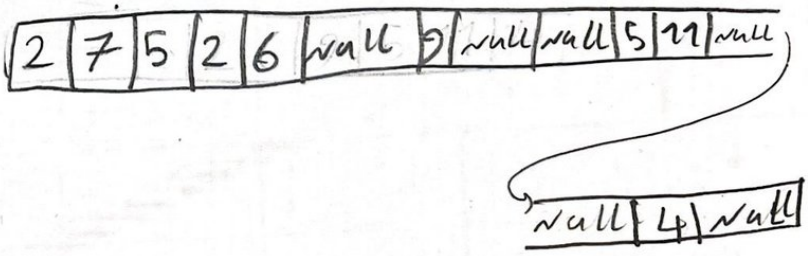


Post order: 3 1 4 9 8 6 —5





Post order: 2 5 11 6 7 4 9 5 2



```python
Codeium: Refactor | Explain
class BinaryTree:
    Codeium: Refactor | Explain | Generate Docstring | ×
    def __init__(self):
        self.tree = []  # درخت دودویی به صورت آرایه‌ای ذخیره می‌شود

    Codeium: Refactor | Explain | ×
    def insert(self, data):
        """افزودن یک مقدار به اولین خانه خالی"""
        self.tree.append(data)

    Codeium: Refactor | Explain | ×
    def find_node(self, data):
        """پیدا کردن اولین گره با مقدار داده‌شده"""
        for i, value in enumerate(self.tree):
            if value == data:
                return i  # اندیس گره برگردانده می‌شود
        return None

    Codeium: Refactor | Explain | ×
    def delete(self, node):
        """حذف یک گره مشخص"""
        if node < 0 or node >= len(self.tree):  # چک کردن صحت اندیس
            return False

        # جایگزینی گره با آخرین گره
        self.tree[node] = self.tree[-1]
        self.tree.pop()  # حذف آخرین گره
        return True

    Codeium: Refactor | Explain | ×
    def get_height(self):
        """محاسبه ارتفاع درخت"""
        import math
        return math.ceil(math.log2(len(self.tree) + 1)) - 1

    Codeium: Refactor | Explain | ×
    def get_size(self):
        """برگرداندن تعداد گره‌های درخت"""
        return len(self.tree)

    Codeium: Refactor | Explain | ×
    def LMC(self):
        """برگ چپ‌ترین گره را پیدا می‌کند"""
        if not self.tree:
```

```python
class BinaryTree:
    Codeium: Refactor | Explain | ×
    def LMC(self):
        """برگ چپ‌ترین گره را پیدا می‌کند"""
        if not self.tree:
            return None
        index = 0
        while 2 * index + 1 < len(self.tree):  # رفتن به فرزند چپ
            index = 2 * index + 1
        return self.tree[index]

    Codeium: Refactor | Explain | ×
    def parent(self, node):
        """پیدا کردن والد گره داده‌شده"""
        if node <= 0 or node >= len(self.tree):
            return None
        return self.tree[(node - 1) // 2]

    Codeium: Refactor | Explain | ×
    def print_preorder(self, index=0):
        """چاپ مقادیر درخت به صورت پیش‌وندی"""
        if index >= len(self.tree):
            return
        print(self.tree[index], end=" ")
        self.print_preorder(2 * index + 1)  # فرزند چپ
        self.print_preorder(2 * index + 2)  # فرزند راست

    Codeium: Refactor | Explain | ×
    def print_inorder(self, index=0):
        """چاپ مقادیر درخت به صورت میان‌وندی"""
        if index >= len(self.tree):
            return
        self.print_inorder(2 * index + 1)  # فرزند چپ
        print(self.tree[index], end=" ")
        self.print_inorder(2 * index + 2)  # فرزند راست

# نمونه استفاده
bt = BinaryTree()
bt.insert(8)
bt.insert(3)
bt.insert(10)
bt.insert(1)
bt.insert(6)
bt.insert(14)
```

```python
print("Pre-order Traversal:")
bt.print_preorder()
print("\nIn-order Traversal:")
bt.print_inorder()
print("\nHeight of Tree:", bt.get_height())
print("Size of Tree:", bt.get_size())
print("Leftmost Leaf:", bt.LMC())
print("Parent of Node 6:", bt.parent(bt.find_node(6)))

Codeium: Refactor | Explain | ×
def print_postorder(index=0):
    """چاپ مقادیر درخت به صورت پس‌وندی"""
    if index >= len(bt.tree):  # اگر از اندیس آرایه خارج شد
        return
    print_postorder(2 * index + 1)  # پیمایش فرزند چپ
    print_postorder(2 * index + 2)  # پیمایش فرزند راست
    print(bt.tree[index], end=" ")  # چاپ مقدار گره


    class TreeNode:

        Codeium: Refactor | Explain | Generate Docstring | ×
        def __init__(self, data):
            self.data = data
            self.left = None
            self.right = None

Codeium: Refactor | Explain | ×
def create_tree(inorder, postorder):
    """ساخت درخت از آرایه‌های inorder و postorder"""
    if not inorder or not postorder:  # اگر آرایه خالی است
        return None

    # مقدار ریشه از postorder
    root_value = postorder.pop()
    root = TreeNode(root_value)

    # یافتن موقعیت ریشه در inorder
    root_index = inorder.index(root_value)

    # بازسازی زیردرخت راست و چپ (باید از آخر به اول استفاده شود postorder)
    root.right = create_tree(inorder[root_index + 1:], postorder)
    root.left = create_tree(inorder[:root_index], postorder)
```

```
def print_postorder(index=0):
    return
    print_postorder(2 * index + 1)  # پیمایش فرزند چپ
    print_postorder(2 * index + 2)  # پیمایش فرزند راست
    print(bt.tree[index], end=" ")  # چاپ مقدار گره


    class TreeNode:

Codeium: Refactor | Explain | Generate Docstring | ×
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

Codeium: Refactor | Explain | ×
def create_tree(inorder, postorder):
    """ساخت درخت از آرایه‌های inorder و postorder"""
    if not inorder or not postorder:  # اگر آرایه خالی است
        return None

    # مقدار ریشه از postorder
    root_value = postorder.pop()
    root = TreeNode(root_value)

    # یافتن موقعیت ریشه در inorder
    root_index = inorder.index(root_value)

    # بازسازی زیردرخت راست و چپ (باید از آخر به اول استفاده شود postorder)
    root.right = create_tree(inorder[root_index + 1:], postorder)
    root.left = create_tree(inorder[:root_index], postorder)

    return root

    def print_preorder_tree(node):


    if not node:
        return
    print(node.data, end=" ")
    print_preorder_tree(node.left)
    print_preorder_tree(node.right)
```