Nama: Dewa Bagus Putu Arya Dhananjaya

NPM: 10122362

Kelas: 3KA21

1. DFS-Leaf-Sum

```
# Membuat graph sebagai adjacency list (setiap node menunjuk ke list node tetangganya)
graph = {
    'A' : ['B','C'],  # Node A terhubung ke B dan C
    'B' : ['D', 'E'], # Node B terhubung ke D dan E
    'C' : [],         # Node C tidak punya tetangga (leaf)
    'D' : [],         # Node D tidak punya tetangga (leaf)
    'E' : ['F'],      # Node E terhubung ke F
    'F' : []          # Node F tidak punya tetangga (leaf)
}

# Membuat dictionary yang berisi nilai untuk masing-masing node
graph_val = {
    "A": 10,  # Nilai node A = 10
    'B': 12,  # Nilai node B = 12
    'C': 3,   # Nilai node C = 3
    'D': 4,   # Nilai node D = 4
    'E': -6,  # Nilai node E = -6
    'F': -5   # Nilai node F = -5
}
```

```
# Membuat set kosong untuk menyimpan node yang sudah dikunjungi
visited = set()  # Untuk menghindari pengulangan kunjungan node saat DFS
```

```
# Membuat set kosong untuk menyimpan semua leaf nodes
leaf = set()     # Leaf adalah node tanpa tetangga
```

```
# Mendefinisikan fungsi DFS untuk traversal graph
def dfs(visited, graph, node):
    if node not in visited:            # Jika node belum pernah dikunjungi
        visited.add(node)              # Tandai node sebagai sudah dikunjungi

        if len(graph[node]) == 0:       # Jika node tidak punya tetangga (daftar kosong)
            leaf.add(node)              # Masukkan node ke dalam set leaf

        for neighbour in graph[node]:   # Untuk setiap tetangga dari node saat ini
            dfs(visited, graph, neighbour)  # Rekursif: lanjutkan DFS ke tetangga tersebut
```

```
# Mendefinisikan fungsi untuk menghitung jumlah nilai leaf nodes
def leaf_sum(leaf):
    jmlh = 0                           # Inisialisasi jumlah dengan 0
    for leaf_node in leaf:             # Iterasi setiap node dalam set leaf
        leaf_val = graph_val[leaf_node] # Ambil nilai dari node leaf
        jmlh = jmlh + leaf_val          # Tambahkan nilai ke jumlah total
    return jmlh                        # Kembalikan jumlah total setelah loop selesai
```

```
dfs(visited, graph, 'A')    # Memulai DFS dari node 'A' untuk mencari semua leaf
```

```
leaf_sum(leaf)              # Menghitung dan menampilkan jumlah nilai dari semua leaf node
```

```
2
```

2. Graph-Height

```
# Membuat class Node untuk merepresentasikan sebuah node dalam binary tree
class Node:
    # Constructor untuk membuat node baru
    def __init__(self, data):
        self.data = data    # Menyimpan nilai (data) node
        self.left = None    # Pointer ke child sebelah kiri
        self.right = None   # Pointer ke child sebelah kanan
```

```
# Fungsi untuk menghitung 'maxDepth' atau tinggi maksimal dari tree
def maxDepth(node):
    # Jika node kosong (tree kosong)
    if node is None:
        return -1          # Mengembalikan -1 (karena tidak ada node)

    else:
        # Rekursif: hitung kedalaman subtree kiri
        lDepth = maxDepth(node.left)

        # Rekursif: hitung kedalaman subtree kanan
        rDepth = maxDepth(node.right)

        # Gunakan kedalaman yang lebih besar antara kiri dan kanan, lalu tambah 1 (untuk node saat ini)
        if (lDepth > rDepth):
            return lDepth + 1
        else:
            return rDepth + 1
```

```
# Membuat root node dengan data 1
root = Node(1)
# Menambahkan child kiri pada root dengan data 2
root.left = Node(2)
# Menambahkan child kanan pada root dengan data 3
root.right = Node(3)
# Menambahkan child kiri pada node 2 dengan data 4
root.left.left = Node(4)
# Menambahkan child kanan pada node 2 dengan data 5
root.left.right = Node(5)
```

```
# Cetak tinggi tree yang dihitung dari root
print("Height of tree is %d" %(maxDepth(root)))
```

```
Height of tree is 2
```