Nama: Dewa Bagus Putu Arya Dhananjaya

NPM: 10122362

Kelas: 3KA21

FindCenter

```
Tree = {}  # Struktur data dictionary untuk merepresentasikan pohon.
           # Setiap key adalah node, dan value-nya adalah list anak-anak dari node tersebut.

path = []  # List kosong untuk menyimpan jalur (path) dari root ke suatu node.
           # Biasanya digunakan dalam pencarian jalur terpanjang atau pencarian node tertentu.

maxHeight, maxHeightNode = -1, -1  # Inisialisasi variabel untuk mencatat tinggi maksimum pohon (maxHeight)
                                   # dan node pada tingkat tertinggi (maxHeightNode).
                                   # Keduanya diatur ke -1 sebagai nilai awal (belum ditemukan).
```

```
def getDiameterPath(vertex, targetVertex, parent, path):
    # Basis: Jika simpul saat ini adalah target yang dicari
    if(vertex == targetVertex):
        path.append(vertex)  # Tambahkan simpul target ke path
        return True          # Kembalikan True karena path ditemukan

    # NOTE: Baris berikut ini seharusnya tidak menjorok ke dalam blok if
    # Loop ini tidak akan pernah dijalankan karena berada setelah return
    for i in range(len(tree[vertex])):
        # Jika simpul yang dikunjungi adalah parent-nya, lewati (untuk mencegah backtracking)
        if(tree[vertex][i] == parent):
            continue

        # Rekursif: coba eksplorasi anak simpul
        if(getDiameterPath(tree[vertex][i], targetVertex, vertex, path)):
            path.append(vertex)  # Jika ditemukan, tambahkan simpul saat ini ke path
            return True          # Dan kembalikan True

    return False  # Jika tidak ditemukan path ke targetVertex, kembalikan False
```

```
# Fungsi untuk mencari simpul (node) yang paling jauh dari simpul awal dalam sebuah pohon
def farthestNode(vertex, parent, height):
    # Menggunakan variabel global untuk menyimpan tinggi maksimum dan simpul dengan tinggi maksimum
    global maxHeight, maxHeightNode

    # Jika tinggi saat ini lebih besar dari tinggi maksimum yang tercatat, perbarui nilai maksimum dan simpulnya
    if height > maxHeight:
        maxHeight = height
        maxHeightNode = vertex

    # Periksa apakah simpul saat ini memiliki anak dalam struktur pohon (Tree)
    if vertex in Tree:
        # Iterasi semua anak dari simpul saat ini
        for i in range(len(Tree[vertex])):

            # Jika simpul anak sama dengan simpul induk, lanjut ke iterasi berikutnya (hindari traversal ke belakang)
            if Tree[vertex][i] == parent:
                continue

            # Rekursif untuk mengeksplorasi simpul anak dan memperbarui tinggi
            farthestNode(Tree[vertex][i], vertex, height + 1)
```

```
def addedge(a, b):  # Fungsi untuk menambahkan edge antara node a dan b
  if(a not in Tree):  # Jika node a belum ada dalam struktur Tree
    Tree[a] = []      # Buat entri baru untuk node a dengan list kosong sebagai tetangganya

    Tree[a].append(b)  # Tambahkan node b sebagai tetangga dari node a

  if(b not in Tree):  # Jika node b belum ada dalam struktur Tree
    Tree[b] = []      # Buat entri baru untuk node b dengan list kosong sebagai tetangganya

  Tree[b].append(a)   # Tambahkan node a sebagai tetangga dari node b
```

```
def FindCenter(n):
    # Inisialisasi nilai maksimum tinggi dan node dengan tinggi maksimum.
    maxHeight = -1
    maxHeightNode = -1
```

```
    # Langkah pertama: temukan node paling jauh dari node 0 menggunakan DFS (dalam fungsi farthestNode).
    farthestNode(0, -1, 0)
    leaf1 = maxHeightNode  # Node paling jauh dari node 0 (salah satu ujung diameter pohon).

    # Langkah kedua: dari leaf1, temukan node paling jauh darinya, yaitu leaf2 (ujung lain dari diameter).
    maxHeight = -1
    farthestNode(maxHeightNode, -1, 0)
    leaf2 = maxHeightNode

    # Temukan jalur (path) dari leaf1 ke leaf2 — ini adalah diameter pohon.
    path = []
    getDiameterPath(leaf1, leaf2, -1, path)

    # Hitung panjang jalur diameter.
    pathSize = len(path)

    # Jika diameter memiliki panjang ganjil, maka titik pusat adalah node tunggal di tengah-tengah path.
    if(pathSize % 2 == 1):
        print(path[int(pathSize / 2)] * -1)  # Dikalikan -1, kemungkinan sebagai penanda pusat tunggal.
    else:
        # Jika diameter genap, maka pusat terdiri dari dua node yang bersebelahan di tengah path.
        print(path[int(pathSize / 2)], ", ", path[int((pathSize - 1) / 2)], sep = "", end = "")
```

```
# N = 4 berarti jumlah simpul (nodes) dalam pohon adalah 4
N = 4

# Inisialisasi struktur data tree sebagai dictionary kosong
tree = {}

# Menambahkan edge (sisi) antara simpul 1 dan 0
addedge(1, 0)

# Menambahkan edge antara simpul 1 dan 2
addedge(1, 2)

# Menambahkan edge antara simpul 1 dan 3
addedge(1, 3)

# Memanggil fungsi FindCenter untuk mencari pusat dari pohon dengan N simpul
FindCenter(N)
```

```
1
```

Shortest-Path-DAG

```
import sys  # Mengimpor modul sys untuk mengakses fungsi dan variabel tingkat sistem, seperti argumen baris perintah, jalur modul, atau keluar
```

```
class Graph:
  # Konstruktor kelas Graph
  def __init__(self, edges, n):
    # Inisialisasi adjacency list sebagai list kosong untuk setiap simpul (node)
    self.adjList = [[] for _ in range(n)]

    # Mengisi adjacency list dengan edge (sisi) yang diberikan
    # Setiap edge berisi: (source, dest, weight)
    for (source, dest, weight) in edges:
      # Menambahkan simpul tujuan dan bobot ke list simpul asal
      self.adjList[source].append((dest, weight))
```

```
def DFS(graph, v, discovered, departure, time):
    # Tandai bahwa simpul v telah ditemukan (dikunjungi)
    discovered[v] = True

    # Iterasi semua tetangga u dari simpul v bersama bobot w (jika ada)
    for (u, w) in graph.adjList[v]:
        # Jika simpul u belum ditemukan, lakukan DFS secara rekursif
        if not discovered[u]:
            time = DFS(graph, u, discovered, departure, time)

    # Setelah semua tetangga dari simpul v selesai dijelajahi,
    # simpan waktu keberangkatan (departure time) untuk simpul v
    departure[time] = v

    # Tambah waktu (counter) setelah mencatat departure time
    time = time + 1

    # Kembalikan nilai waktu terkini ke pemanggil fungsi
    return time
```

```
def findShortestDistance(graph, source, n):
    # Inisialisasi array departure untuk menyimpan waktu selesai dari DFS
    departure = [-1] * n
    # Menandai apakah simpul sudah dikunjungi dalam DFS
    discovered = [False] * n
    # Waktu awal DFS
    time = 0

    # Melakukan DFS untuk setiap simpul yang belum dikunjungi
    for i in range(n):
        if not discovered[i]:
            time = DFS(graph, source, discovered, departure, time)

    # Inisialisasi biaya jarak terpendek dari source ke semua simpul dengan nilai tak hingga
    cost = [sys.maxsize] * n
    # Jarak dari simpul sumber ke dirinya sendiri adalah 0
    cost[source] = 0

    # Mengakses simpul berdasarkan urutan topologis terbalik (departure time)
    for i in reversed(range(n)):
        v = departure[i]

        # Memperbarui biaya jarak ke tetangga-tetangga simpul `v`
        for (u, w) in graph.adjList[v]:
            if cost[v] != sys.maxsize and cost[v] + w < cost[u]:
                cost[u] = cost[v] + w

    # Mencetak jarak minimum dari source ke semua simpul
    for i in range(n):
        print(f'dist({source}, {i}) = {cost[i]}')
```

```
if __name__ == "__main__":

    # Mendefinisikan daftar edges yang berisi tuple yang terdiri dari (node_1, node_2, weight)
    edges = [
        (0, 6, 2), (1, 2, -4), (1, 4, 1), (1, 6, 8), (3, 0, 3), (3, 4, 5),
        (5, 1, 2), (7, 0, 6), (7, 1, -1), (7, 3, 4), (7, 5, -4)
    ]

    # Jumlah node pada graph, dalam hal ini ada 8 node (0 sampai 7)
    n = 8

    # Membuat objek graph berdasarkan edges dan jumlah node yang sudah didefinisikan sebelumnya
    graph = Graph(edges, n)

    # Menetapkan node sumber (source) sebagai node ke-7
    source = 7

    # Memanggil fungsi untuk menghitung jarak terpendek dari node sumber ke semua node lainnya
    findShortestDistance(graph, source, n)
```

```
dist(7, 0) = 6
dist(7, 1) = -2
dist(7, 2) = -6
dist(7, 3) = 4
dist(7, 4) = -1
dist(7, 5) = -4
dist(7, 6) = 6
dist(7, 7) = 0
```