MUHAMMAD TARMIDZI BARIQ

51422161

3IA11

M7

```
# Python3 implementation of the above approach

# To create tree
tree = {}
path = []
maxHeight, maxHeightNode = -1, -1

# Function to store the path
# from given vertex to the target
# vertex in a vector path

[2] def getDiameterPath(vertex, targetVertex, parent, path):
```

implementasi pencarian path (jalur) pada sebuah struktur tree.

- Membuat sebuah tree (menggunakan dictionary tree)
- Menyimpan path dari satu node ke node target
- Menyiapkan variabel untuk mencari node dengan ketinggian (depth) maksimum

```
def getDiameterPath(vertex, targetVertex, parent, path):

    # If the target node is found,
    # push it into path vector
    if (vertex == targetVertex):
        path.append(vertex)
        return True

    for i in range(len(tree[vertex])):
        # To prevent visiting a
        # node already visited
        if (tree[vertex][i] == parent):
            continue

        # Recursive call to the neighbours
        # of current node inorder
        # to get the path
        if (getDiameterPath(tree[vertex][i], targetVertex, vertex, path)):
            path.append(vertex)
            return True
    return False
```

Mencari dan menyimpan jalur dari satu node ke node lain dalam tree, dengan cara DFS (depth-first search) rekursif, dan menghindari siklus dengan mencatat parent.

```
# Function to obtain and return the
# farthest node from a given vertex
def farthestNode(vertex, parent, height):
    global maxHeight, maxHeightNode
    # If the current height is maximum
    # so far, then save the current node
    if (height > maxHeight):
        maxHeight = height
        maxHeightNode = vertex

    # Iterate over all the neighbours
    # of current node
    if (vertex in tree):
        for i in range(len(tree[vertex])):

            # This is to prevent visiting
            # a already visited node
            if (tree[vertex][i] == parent):
                continue

            # Next call will be at 1 height
            # higher than our current height
            farthestNode(tree[vertex][i], vertex, height + 1)
```

Menggunakan DFS (Depth-First Search) untuk mencari node terjauh dari titik tertentu, menyimpan:

- maxHeight: jarak maksimum
- maxHeightNode: simpul yang berjarak maksimum

```
# Function to add edges
def addedge(a, b):
    if (a not in tree):
        tree[a] = []

    tree[a].append(b)

    if (b not in tree):
        tree[b] = []

    tree[b].append(a)
```

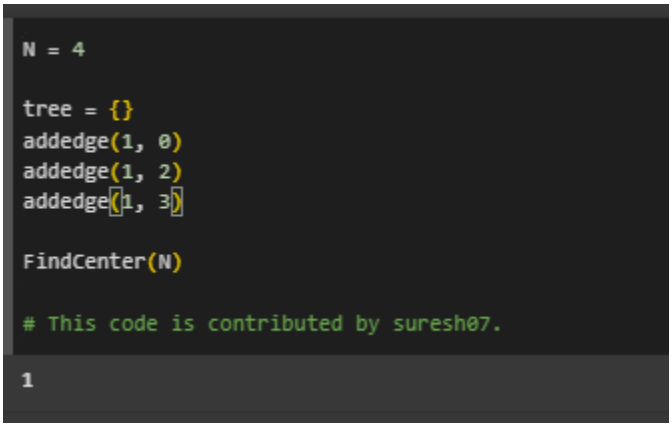
Fungsi addedge() membuat graph tidak berarah dengan:

- Menambahkan koneksi dua arah antara node a dan b
- Menjaga struktur data tetap dinamis dengan dictionary

```
def FindCenter(n):
    # Now we will find the 1st farthest
    # node from 0(any arbitrary node)

    # Perform DFS from 0 and update
    # the maxHeightNode to obtain
    # the farthest node from 0

    # Reset to -1
    maxHeight = -1

    # Reset to -1
    maxHeightNode = -1

    farthestNode(0, -1, 0)

    # Stores one end of the diameter
    leaf1 = maxHeightNode

    # Similarly the other end of
    # the diameter

    # Reset the maxHeight
    maxHeight = -1
    farthestNode(maxHeightNode, -1, 0)

    # Stores the second end
    # of the diameter
    leaf2 = maxHeightNode

    # Store the diameter into
    # the vector path
    path = []

    # Diameter is equal to the
    # path between the two farthest
    # nodes leaf1 and leaf2
    getDiameterPath(leaf1, leaf2, -1, path)

    pathSize = len(path)

    if (pathSize % 2 == 1):
        print(path[int(pathSize / 2)]*-1)
    else:
        print(path[int(pathSize / 2)], ", ", path[int((pathSize - 1) / 2)], sep = "", end = "")
```

Fungsi FindCenter(n):

- Menemukan diameter dari tree
- Mengambil node tengah dari diameter sebagai center
- Cocok untuk banyak aplikasi graf: manajemen jaringan, pohon keputusan, dsb.



```
N = 4

tree = {}
addedge(1, 0)
addedge(1, 2)
addedge(1, 3)

FindCenter(N)

# This code is contributed by suresh07.
```

```
1
```

Fungsi ini:

- Mencari diameter dari tree → dalam hal ini, diameter adalah antara node 0 dan 2 (atau 0 ke 3, dst.) → jalur seperti: 0 - 1 - 2
- Jalur terpanjang (path) = [2, 1, 0]
- Maka node tengah dari path tersebut adalah: 1

3-Tekrek-M7-Sortest-Path-DAG.ipynb

```
import sys
```

Import Library python sys

```
class Graph:
    # Constructor
    def __init__(self, edges, n):

        # A list of lists to represent an adjacency list
        self.adjList = [[] for _ in range(n)]

        # add edges to the directed graph
        for (source, dest, weight) in edges:
            self.adjList[source].append((dest, weight))


# Perform DFS on the graph and set the departure time of all vertices of the graph
```

Penjelasan:

- **edges** adalah daftar tuple (source, destination, weight) yang menyatakan sisi berarah dan berbobot.
- **n** adalah jumlah simpul dalam graf.
- **self.adjList** adalah list dari list (array 2D) untuk menyimpan representasi adjacency list.
- **self.adjList[source]** akan menyimpan tuple (dest, weight) untuk setiap edge dari source ke dest.

```python
def DFS(graph, v, discovered, departure, time):

    # mark the current node as discovered
    discovered[v] = True

    # set arrival time - not needed
    # time = time + 1

    # do for every edge (v, u)
    for (u, w) in graph.adjList[v]:
        # if `u` is not yet discovered
        if not discovered[u]:
            time = DFS(graph, u, discovered, departure, time)

    # ready to backtrack
    # set departure time of vertex `v`
    departure[time] = v

    time = time + 1
    return time
```

discovered[v] = True

Tandai simpul v sudah dikunjungi.

for (u, w) in graph.adjList[v]:

    if not discovered[u]:

        time = DFS(graph, u, discovered, departure, time)

Lakukan DFS ke setiap tetangga u dari v. Jika u belum dikunjungi, rekursi ke u.

departure[time] = v

time = time + 1

Ketika semua anak simpul sudah selesai dikunjungi (saat backtrack), masukkan simpul v ke departure dengan waktu sekarang. Lalu, waktu ditambah.

```python
# The function performs the topological sort on a given DAG and then finds
# the longest distance of all vertices from the given source by running one pass
# of the Bellman-Ford algorithm on edges of vertices in topological order
def findShortestDistance(graph, source, n):

    # `departure` stores the vertex number using departure time as an index
    departure = [-1] * n

    # to keep track of whether a vertex is discovered or not
    discovered = [False] * n
    time = 0

    # perform DFS on all undiscovered vertices
    for i in range(n):
        if not discovered[i]:
            time = DFS(graph, i, discovered, departure, time)

    cost = [sys.maxsize] * n
    cost[source] = 0

    # Process the vertices in topological order, i.e., in order
    # of their decreasing departure time in DFS
    for i in reversed(range(n)):

        # for each vertex in topological order, relax the cost of its adjacent vertices
        v = departure[i]

        # edge from `v` to `u` having weight `w`
        for (u, w) in graph.adjList[v]:
            # if the distance to destination `u` can be shortened by
            # taking edge (v, u), update cost to the new lower value
            if cost[v] != sys.maxsize and cost[v] + w < cost[u]:
                cost[u] = cost[v] + w

    # print shortest paths
    for i in range(n):
        print(f'dist({source}, {i}) = {cost[i]}')
```

departure = [-1] * n

discovered = [False] * n

time = 0

- departure[i]: Simpul dengan waktu keluar i saat DFS (untuk urutan topologis).
- discovered[i]: Penanda apakah simpul sudah dikunjungi dalam DFS.
- time: Counter waktu untuk urutan DFS.

```
for i in range(n):
    if not discovered[i]:
        time = DFS(graph, i, discovered, departure, time)
```

- DFS dijalankan dari setiap simpul yang belum dikunjungi.
- departure[] akan berisi simpul-simpul dalam urutan keluar (digunakan sebagai urutan topologis).

```
cost = [sys.maxsize] * n
cost[source] = 0
```

- cost[i] menyimpan jarak terpendek dari source ke simpul i.
- sys.maxsize berarti belum terjangkau (jarak tak hingga).

```
for i in reversed(range(n)):
    v = departure[i]
    for (u, w) in graph.adjList[v]:
        if cost[v] != sys.maxsize and cost[v] + w < cost[u]:
            cost[u] = cost[v] + w
```

Penjelasan:

- Urutan topologis = departure[] dibalik (karena DFS menyimpan urutan keluar).
- Untuk setiap simpul v, lakukan relaksasi ke semua tetangganya u:
- Jika cost[u] > cost[v] + w, perbarui cost[u].

```
for i in range(n):
    print(f'dist({source}, {i}) = {cost[i]}')
```

Menampilkan jarak terpendek dari simpul sumber ke semua simpul lain.

```python
if __name__ == '__main__':

    # List of graph edges as per the above diagram
    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)
    ]

    # total number of nodes in the graph (labelled from 0 to 7)
    n = 8

    # build a graph from the given edges
    graph = Graph(edges, n)

    # source vertex
    source = 7

    # find the shortest distance of all vertices from the given source
    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
```

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)

]

- Setiap tuple (u, v, w) menyatakan ada edge dari simpul u ke v dengan bobot w.
- Beberapa bobot bernilai negatif (misalnya (1, 2, -4), (7, 5, -4)), yang valid untuk DAG selama tidak membentuk siklus negatif (DAG tidak boleh ada siklus).

n = 8

Graf memiliki 8 simpul, diberi label dari 0 hingga 7.

graph = Graph(edges, n)

Membuat objek Graph dan membangun adjacency list dari daftar edges.

source = 7

Algoritma akan mencari jarak terpendek dari simpul 7 ke semua simpul lainnya.

findShortestDistance(graph, source, n)

- Ini adalah pemanggilan utama yang:
- Melakukan topological sort menggunakan DFS.
- Melakukan relaksasi edge satu arah dalam urutan topologis.
- Mencetak jarak terpendek dari source ke semua simpul.