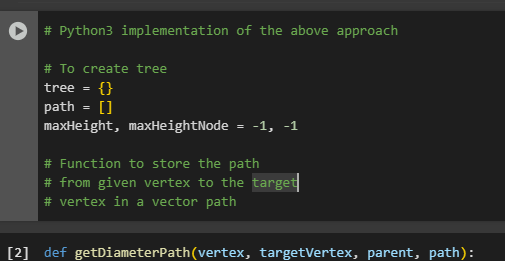
MUHAMMAD TARMIDZI BARIQ

51422161

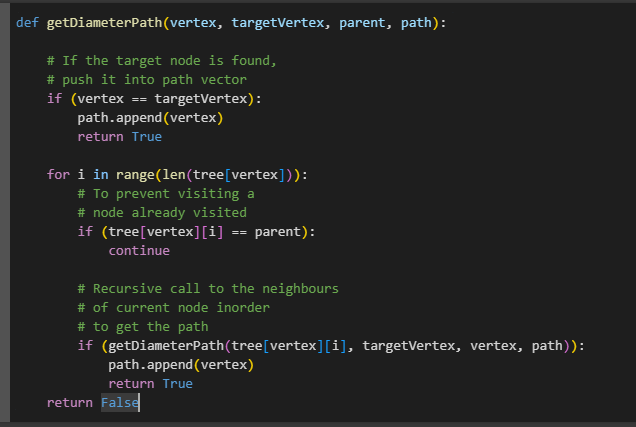
3IA11

M7



**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

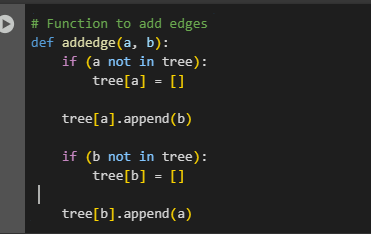


Mencari dan menyimpan jalur dari satu node ke node lain dalam tree, dengan cara DFS (depth-first search) rekursif, dan menghindari siklus dengan mencatat parent.



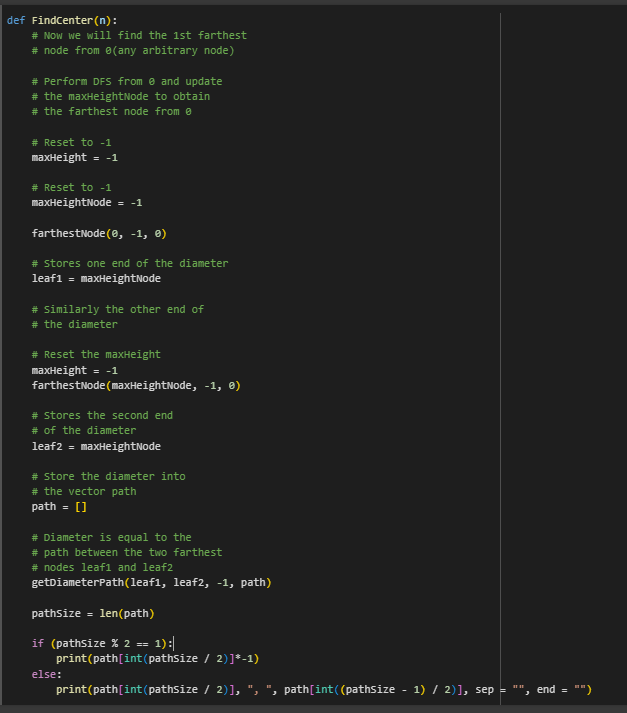
Menggunakan DFS (Depth-First Search) untuk mencari node terjauh dari titik tertentu, menyimpan:

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



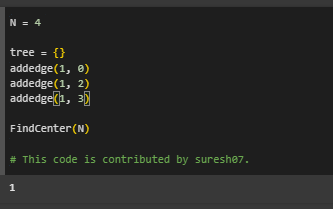
Fungsi addedge() membuat graph tidak berarah dengan:

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



Fungsi FindCenter(n):

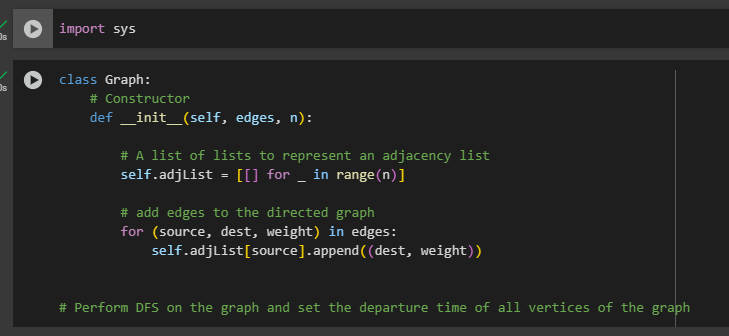
* Menemukan diameter dari tree
* Mengambil node tengah dari diameter sebagai center
* Cocok untuk banyak aplikasi graf: manajemen jaringan, pohon keputusan, dsb.



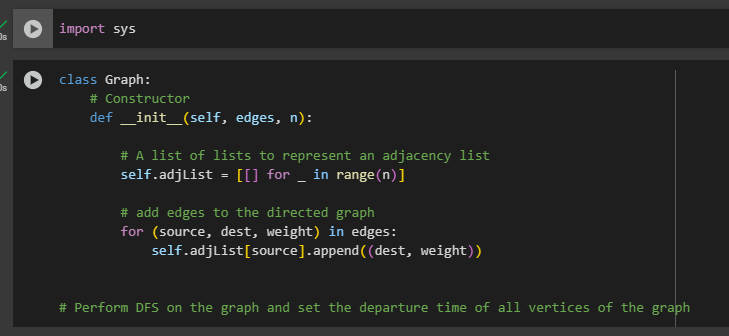
Fungsi ini:

* 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**

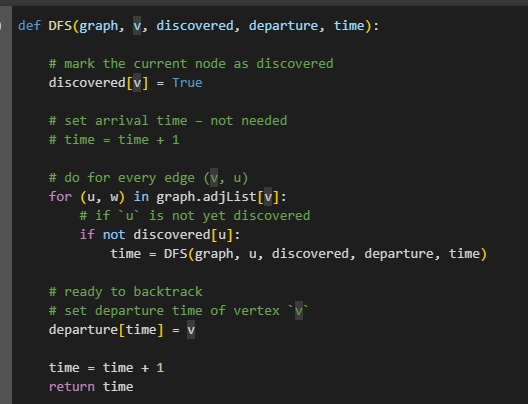
****

Import Library python sys



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.



discovered[v] = True

Tandai simpul v sudah dikunjungi.

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

if not discovered[u]:

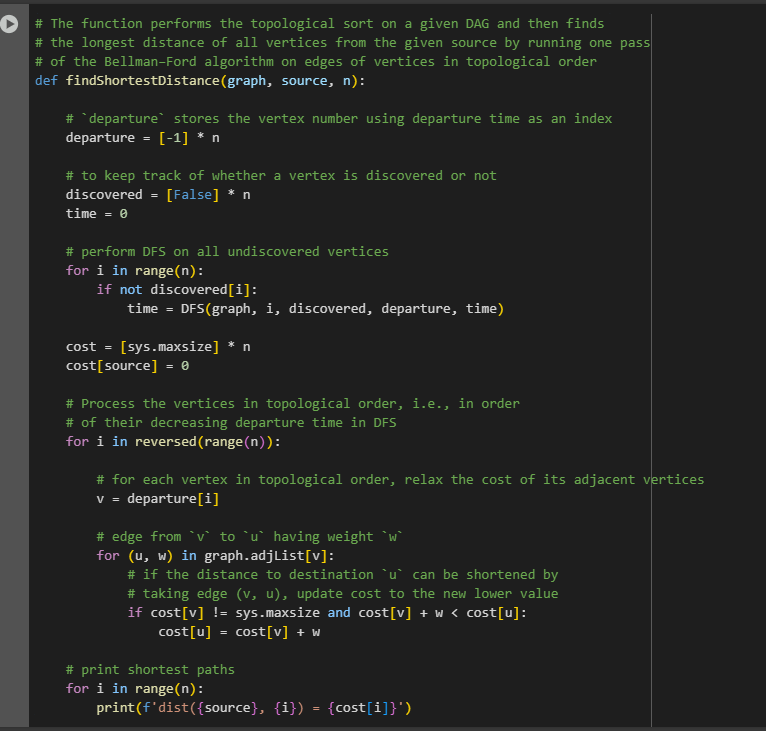
time = DFS(graph, u, discovered, departure, time)

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.



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

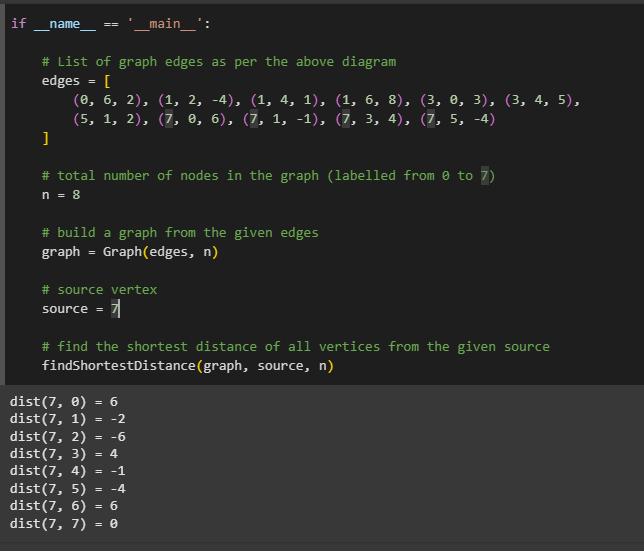
Penjelasan:

* 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.



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.