Kode Latihan :

import heapq

def dijkstra(graph, start, end):

# Inisialisasi jarak ke setiap node dengan nilai infinity

distances = {node: float('inf') for node in graph}

# Jarak ke node awal diatur ke 0

distances[start] = 0

# Setiap node yang sudah dievaluasi disimpan di set visited

visited = set()

# Queue untuk menyimpan node yang akan dievaluasi berikutnya

heap = [(0, start)]

while heap:

# Ambil node dengan jarak terdekat dari heap

(distance, current\_node) = heapq.heappop(heap)

# Jika node tersebut belum dievaluasi sebelumnya

if current\_node not in visited:

# Tandai node tersebut sebagai dievaluasi

visited.add(current\_node)

# Loop melalui semua node yang terhubung dengan node tersebut

for neighbor, weight in graph[current\_node].items():

# Hitung jarak baru ke node tersebut

new\_distance = distance + weight

# Jika jarak baru lebih pendek dari jarak sebelumnya ke node tersebut

if new\_distance < distances[neighbor]:

# Update jarak ke node tersebut

distances[neighbor] = new\_distance

# Tambahkan node tersebut ke heap

heapq.heappush(heap, (new\_distance, neighbor))

# Kembalikan jarak terpendek ke node tujuan

return distances[end]

# Definisikan grafik

graph = {

'A': {'B': 30, 'C': 50},

'B': {'C': 10, 'D': 10},

'C': {'D': 20, 'E': 30},

'D': {'E': 50},

'E': {}

}

# Hitung jarak terpendek dari A ke E

shortest\_distance = dijkstra(graph, 'A', 'E')

# Cetak jarak terpendek

print(f"Jarak terpendek dari A ke E: {shortest\_distance}")

Percobaan:

def find\_shortest\_path(graph, start, end):

# Basis: jika start dan end sama, maka jaraknya 0

if start == end:

return 0

# Jika tidak, cari jarak terpendek dari titik awal ke titik tujuan

else:

# Inisialisasi jarak terpendek ke infinity

shortest\_path = float('inf')

# Loop melalui setiap node yang terhubung dengan start

for neighbor, path\_length in graph[start].items():

# Jika jarak tersebut lebih pendek dari jarak terpendek yang sudah ditemukan

if path\_length < shortest\_path:

# Cari rute terpendek dari node tersebut ke titik tujuan

recursive\_path = find\_shortest\_path(graph, neighbor, end)

# Jika rute tersebut ditemukan dan jarak rute tersebut lebih pendek dari jarak terpendek yang sudah ditemukan

if recursive\_path is not None and recursive\_path + path\_length < shortest\_path:

# Update jarak terpendek

shortest\_path = recursive\_path + path\_length

# Jika jarak terpendek tidak berubah dari nilai awal, artinya tidak ada jalur yang ditemukan

if shortest\_path == float('inf'):

return None

# Kembalikan jarak terpendek yang ditemukan

return shortest\_path

graph = {

'A': {'B': 1, 'C': 4},

'B': {'D': 2},

'C': {'D': 3},

'D': {}

}

start = 'A'

end = 'D'

shortest\_path = find\_shortest\_path(graph, start, end)

print(f"Jarak terpendek dari {start} ke {end} adalah {shortest\_path}")