Bing

Voici une traduction possible de l'algorithme de Dijkstra en Python :

# Définir le graphe sous forme d'une liste de listes

graphe = [
[0, 10, 0, 0, 0, 8], # Ville 0
[10, 0, 0, 2, 0, 0], # Ville 1
[0, 0, 0, 16, 12, 0], # Ville 2
[0, 2, 16, 0, 6, 0], # Ville 3
[0, 0, 12, 6, 0, 9], # Ville 4
[8, 0, 0, 0, 9, 0] # Ville 5
]

# Définir le nombre de villes

n = len(graphe)

# Définir la ville de départ et la ville d'arrivée

depart = 0
arrivee = n - 1

# Initialiser les ensembles S et Q

S = set()
Q = set([depart])

# Initialiser les tableaux D et P

D = [float("inf")] \* n
D[depart] = 0
P = [None] \* n

# Tant que Q n'est pas vide

while Q:

# Sélectionner dans Q la ville u qui a la plus petite distance dans D

u = min(Q, key=lambda x: D[x])

# Ajouter u à S et le retirer de Q

S.add(u)

Q.remove(u)

# Si u est la ville d'arrivée, arrêter l'algorithme

if u == arrivee:

break

# Sinon, pour chaque ville v voisine de u qui n'est pas dans S

for v in range(n):

if graphe[u][v] > 0 and v not in S:

# Calculer la distance entre u et v par la route qui les relie

dist\_uv = graphe[u][v]

# Ajouter cette distance à la distance entre la ville de départ et u dans D

dist\_du\_v = D[u] + dist\_uv

# Si cette somme est inférieure à la distance entre la ville de départ et v dans D

if dist\_du\_v < D[v]:

# Mettre à jour D avec cette somme

D[v] = dist\_du\_v

# Mettre à jour P avec u comme prédécesseur de v

P[v] = u

# Ajouter v à Q si ce n'est pas déjà fait

Q.add(v)

# Construire le plus court chemin en partant de la ville d'arrivée et en remontant les prédécesseurs dans P jusqu'à la ville de départ

chemin = []
v = arrivee
while v is not None:
chemin.append(v)
v = P[v]
chemin.reverse()

# Afficher le plus court chemin et la distance correspondante

print("Le plus court chemin entre la ville", depart, "et la ville", arrivee, "est :")
print(chemin)
print("La distance totale du trajet est :")
print(D[arrivee])