Prims +krushkals

Prims

import sys

class Graph():

def \_\_init\_\_(self, vertices):

self.V = vertices

self.graph = [[0 for \_ in range(vertices)] for \_ in range(vertices)]

def printMST(self, parent, heuristics, total\_cost):

print("\nEdge \tWeight \tHeuristic Value")

for i in range(1, self.V):

print(f"{parent[i]} - {i} \t {self.graph[i][parent[i]]} \t {heuristics[i]}")

print(f"\nTotal MST Cost: {total\_cost}")

def minKey(self, key, mstSet):

min\_val = sys.maxsize

min\_index = -1

for v in range(self.V):

if key[v] < min\_val and not mstSet[v]:

min\_val = key[v]

min\_index = v

return min\_index

def primMST(self, heuristics, source=0):

key = [sys.maxsize] \* self.V

parent = [None] \* self.V

key[source] = 0

mstSet = [False] \* self.V

parent[source] = -1

for \_ in range(self.V):

u = self.minKey(key, mstSet)

mstSet[u] = True

for v in range(self.V):

if self.graph[u][v] > 0 and not mstSet[v] and self.graph[u][v] < key[v]:

key[v] = self.graph[u][v]

parent[v] = u

total\_cost = sum(self.graph[i][parent[i]] for i in range(1, self.V))

self.printMST(parent, heuristics, total\_cost)

if \_\_name\_\_ == '\_\_main\_\_':

print(" Heuristic is used for helping to choose the next edge. If the heuristic value is less then the edge is selected.")

vertices = int(input("\nEnter the number of vertices: "))

g = Graph(vertices)

# Creates a graph object with vertices number of vertices.

print("\nEnter the graph as an adjacency matrix (use 0 for no edge FOR 4 VERTICES(EG:0 10 2 0):")

for i in range(vertices):

row = list(map(int, input(f"Row {i}: ").split()))

for j in range(vertices):

g.graph[i][j] = row[j]

# User inputs each row of the adjacency matrix.A 0 represents **no edge between those two vertices**.Any positive value indicates the **edge weight**.

heuristics = []

print("\nEnter heuristic values for each vertex:")

for i in range(vertices):

h\_value = int(input(f"Heuristic value for vertex {i}: "))

heuristics.append(h\_value)

source\_vertex = int(input("\nEnter the source vertex (0 to V-1): "))

g.primMST(heuristics, source=source\_vertex)

#  Starts building the MST from the given source vertex.

#Uses:

#key[] — to pick the next minimum weight edge.

#mstSet[] — to track the vertices included in the MST.

#parent[] — to store the constructed MST.

# Estimated cost to reach the goal from the current node

# Enter the number of vertices: 5

# Enter the graph matrix:

# 2 3 5 6 1

# 2 1 4 8 9

# 0 2 4 0 3

# 5 6 1 8 10

# 2 0 5 0 6

# Enter the heuristic values for each vertex:

# Heuristic value for vertex 0: 2

# Heuristic value for vertex 1: 1

# Heuristic value for vertex 2: 0

# Heuristic value for vertex 3: 3

# Heuristic value for vertex 4: 4

# Enter the source vertex: 0

# Edge Weight Heuristic Value

# 0 - 1 2 1

# 1 - 2 2 0

# 0 - 3 5 3

# 0 - 4 2 4

# Total MST Cost: 11

Krushkals

class Graph:

def \_\_init\_\_(self, vertices):

self.V = vertices

self.graph = []

self.heuristics = {}

# initialize Number of vertices.Empty list for edges

def addEdge(self, u, v, w, h=0):

# Add edge to the graph and heuristic values to the dictionary

self.graph.append([u, v, w])

self.heuristics[(u, v)] = h

self.heuristics[(v, u)] = h

def find(self, parent, i):

# Find function with path compression

if parent[i] != i:

parent[i] = self.find(parent, parent[i])

return parent[i]

def union(self, parent, rank, x, y):

# Union function to connect two components

if rank[x] < rank[y]:

parent[x] = y

elif rank[x] > rank[y]:

parent[y] = x

else:

parent[y] = x

rank[x] += 1

#Merges two trees based on rank

def KruskalMST(self):

result = [] # To store the resulting MST

i = 0 # Initial edge index

e = 0 # Initial count of edges in the MST

# Sort edges based on the weight and heuristic

self.graph.sort(key=lambda item: item[2] + self.heuristics.get((item[0], item[1]), 0))

# Initialize disjoint sets

parent = list(range(self.V))

rank = [0] \* self.V

while e < self.V - 1 and i < len(self.graph):

u, v, w = self.graph[i]

i += 1

x = self.find(parent, u)

y = self.find(parent, v)

if x != y: # If u and v are not in the same set, add edge to MST

e += 1

result.append([u, v, w])

self.union(parent, rank, x, y)

# Print the result

minimumCost = sum([weight for \_, \_, weight in result])

print("\nEdges in the constructed MST:")

for u, v, weight in result:

print(f"{u} -- {v} == {weight}")

print("Minimum Spanning Tree Cost:", minimumCost)

if \_\_name\_\_ == '\_\_main\_\_':

print("Name: minimum spanning tree-krushkals ")

print("Kruskal's Algorithm is a greedy algorithm used to find the Minimum Spanning Tree (MST) .we sort all edges here pick smallest edge and check if it doesn'St form cycle ")

print("Heuristic is used for helping to choose the next edge. Edge with low weight+ low heuristic value is chosen")

# Input the number of vertices

vertices = int(input("Enter the number of vertices: "))

g = Graph(vertices)

# Input the number of edges

edges = int(input("Enter the number of edges: "))

print("Enter the edges in the format: u v weight heuristic")

# Input edges and heuristics

for \_ in range(edges):

u, v, w, h = map(int, input().split()) # Read edge: u, v, weight, heuristic

g.addEdge(u, v, w, h)

# Run Kruskal's Algorithm to find the Minimum Spanning Tree (MST)

g.KruskalMST()

#sample input

#Enter the number of vertices: 5

#Enter the number of edges: 4

#Enter the edges in the format: u v weight heuristic

#0 1 10 2

# 2 4 6 1

# 0 3 5 0

#1 3 15 1