**بسم تعالی**

کیان پورآذر ۴۰۱۳۱۴۰۳

سوال ۱:

برای حل این مسئله ما از الگوریتم Floyed-Warshall استفاده میکنیم به این صورت که:

ورودی: ماتریس مجاورت گراف graph با n راس

خروجی: ماتریس کوتاه‌ترین مسیرها dist بین همه جفت رئوس

def floyd\_warshall(graph):

n = len(graph)

dist = [[float('inf')] \* n for \_ in range(n)]

😀ماتریس راه حل را مانند ماتریس گراف ورودی مقداردهی اولیه میکنیم:

for i in range(n):

for j in range(n):

if i == j:

dist[i][j] = 0

elif graph[i][j] != 0:

dist[i][j] = graph[i][j]

😀همه رئوس را یکی یکی به مجموعه رئوس میانی اضافه میکنیم:

for k in range(n):

for i in range(n):

for j in range(n):

if dist[i][j] > dist[i][k] + dist[k][j]:

dist[i][j] = dist[i][k] + dist[k][j]

return dist

# Example:

graph = [

[0, 3, float('inf'), 5],

[2, 0, float('inf'), 4],

[float('inf'), 1, 0, float('inf')],

[float('inf'), float('inf'), 2, 0]

]

shortest\_paths = floyd\_warshall(graph)

for row in shortest\_paths:

print(row)

این الگوریتم بدلیل وجود ۳ عدد loop for تودرتو در O(n3) اجرا میشود.

سوال۲:

برای حل این مسئله ما از الگوریتم Kruskal استفاده میکنیم به این صورت که:

ورودی: لیست یال‌ها و تعداد رئوس گراف

خروجی: مجموع وزن یال‌ها در کوچکترین درخت پوشا

class UnionFind:

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

self.parent = list(range(n))

self.rank = [1] \* n

def find(self, u):

if self.parent[u] != u:

self.parent[u] = self.find(self.parent[u])

return self.parent[u]

def union(self, u, v):

root\_u = self.find(u)

root\_v = self.find(v)

if root\_u != root\_v:

if self.rank[root\_u] > self.rank[root\_v]:

self.parent[root\_v] = root\_u

elif self.rank[root\_u] < self.rank[root\_v]:

self.parent[root\_u] = root\_v

else:

self.parent[root\_v] = root\_u

self.rank[root\_u] += 1

def kruskal(n, edges):

uf = UnionFind(n)

mst\_weight = 0

edges.sort(key=lambda x: x[2])

for u, v, weight in edges:

if uf.find(u) != uf.find(v):

uf.union(u, v)

mst\_weight += weight

return mst\_weight

# Example:

edges = [

(0, 1, 10),

(0, 2, 6),

(0, 3, 5),

(1, 3, 15),

(2, 3, 4)

]

n = 4

minimum\_weight = kruskal(n, edges)

print(minimum\_weight)

این الگوریتم در O(|E| log |V|) اجرا میشود.

سوال ۳:

کلاس Graph: این کلاس برای ساخت گراف و افزودن یال‌ها استفاده می‌شود.

متد kruskal\_mst: این متد الگوریتم کروسکال را پیاده‌سازی می‌کند تا درخت پوشای کمینه را پیدا میکند.

الگوریتم کروسکال را برای یافتن درخت پوشای کمینه (Minimum Spanning Tree) اعمال میکنیم.

class Graph:

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

self.V = vertices

self.graph = []

def add\_edge(self, u, v, w):

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

def find(self, parent, i):

if parent[i] == i:

return i

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

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

root\_x = self.find(parent, x)

root\_y = self.find(parent, y)

if rank[root\_x] < rank[root\_y]:

parent[root\_x] = root\_y

elif rank[root\_x] > rank[root\_y]:

parent[root\_y] = root\_x

else:

parent[root\_y] = root\_x

rank[root\_x] += 1

def kruskal\_mst(self):

result = []

i = 0

e = 0

self.graph = sorted(self.graph, key=lambda item: item[2])

parent = []

rank = []

for node in range(self.V):

parent.append(node)

rank.append(0)

while e < self.V - 1:

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

i = i + 1

x = self.find(parent, u)

y = self.find(parent, v)

if x != y:

e = e + 1

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

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

return result

این الگوریتم در O(E log E + E log V) اجرا میشود که V تعداد رئوس و E هم تعداد یال ها است.