**TRAVELLING SALESMAN PROBLEM**

from itertools import permutations

def calculate\_cost(graph, path):

cost = 0

for i in range(len(path)):

cost += graph[path[i] - 1][path[(i + 1) % len(path)] - 1]

return cost

def tsp\_all\_paths(graph, start\_city):

n = len(graph)

cities = list(range(1, n + 1))

cities.remove(start\_city)

all\_permutations = permutations(cities)

paths\_and\_costs = []

min\_cost = float('inf')

for perm in all\_permutations:

path = [start\_city] + list(perm) + [start\_city]

cost = calculate\_cost(graph, path)

paths\_and\_costs.append((path, cost))

if cost < min\_cost:

min\_cost = cost

# Display all paths with their costs

total\_paths = 0

print("All possible paths with their costs:")

path\_number = 1

for path, cost in paths\_and\_costs:

path\_str = " -> ".join(map(str, path))

if cost == min\_cost:

# Highlight shortest path with asterisks

print(f" \*\*\*Path {path\_number}: {path\_str} , Cost: {cost}\*\*\*")

else:

print(f"Path {path\_number}: {path\_str}, Cost: {cost}")

path\_number += 1

total\_paths += 1

print(f"\nTotal number of paths: {total\_paths}")

# Display shortest path(s) with stars

print("\nShortest path(s):")

path\_number = 1

for path, cost in paths\_and\_costs:

if cost == min\_cost:

shortest\_path\_str = " -> ".join(map(str, path))

print(f"Path {path\_number}: {shortest\_path\_str} , Cost: {cost}")

path\_number += 1

def get\_graph():

while True:

n = input("Enter the number of cities: ")

if n.isdigit() and int(n) > 1:

n = int(n)

break

else:

print("Invalid input. Please enter a positive integer.")

graph = []

print("Enter the distance matrix (each row separated by a newline):")

for i in range(n):

while True:

try:

row = list(map(int, input(f"Row {i + 1} (for city {i + 1}): ").split()))

if len(row) != n:

raise ValueError

if any(x < 0 for x in row):

raise ValueError

graph.append(row)

break

except ValueError:

print(f"Invalid input for row {i + 1}. Please enter {n} non-negative integers.")

return graph

if \_\_name\_\_ == "\_\_main\_\_":

graph = get\_graph()

while True:

try:

start\_city = int(input(f"Enter the starting city (1 to {len(graph)}): "))

if start\_city < 1 or start\_city > len(graph):

raise ValueError

break

except ValueError:

print(f"Invalid input. Please enter an integer between 1 and {len(graph)}.")

tsp\_all\_paths(graph, start\_city)