def tsp(graph, v, n, current\_pos, cost, count, ans):

    # Base case: If all nodes are visited and there's an edge back to the start

    if count == n and graph[current\_pos][0] > 0:

        # Update the answer with the minimum cost path found

        ans = min(ans, cost + graph[current\_pos][0])

        return ans

    # Traverse all nodes to explore possible paths

    for i in range(n):

        # If the node i is not visited and there’s an edge from current\_pos to i

        if not v[i] and graph[current\_pos][i] > 0:

            v[i] = True  # Mark node i as visited

            # Recursive call to explore the path including node i

            ans = tsp(graph, v, n, i, cost + graph[current\_pos][i], count + 1, ans)

            # Backtrack: Unmark node i as visited to explore other paths

            v[i] = False

    return ans  # Return the minimum cost found

def main():

    n = 4  # Number of cities (nodes)

    # Graph representing distances between each pair of cities

    graph = [

        [0, 10, 15, 20],

        [10, 0, 35, 25],

        [15, 35, 0, 30],

        [20, 25, 30, 0]

    ]

    v = [False] \* n  # Visited array to keep track of visited cities

    ans = float('inf')  # Initialize the minimum cost answer as infinity

    v[0] = True  # Start the journey from the first city (0th index)

    # Call tsp function to find the minimum traveling cost starting from city 0

    ans = tsp(graph, v, n, 0, 0, 1, ans)

    # Output the minimum cost of visiting all cities and returning to the start

    print(ans)

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

    main()