```python
V = 4
answer = []

# Function to find the minimum weight
# Hamiltonian Cycle
def tsp(graph, v, currPos, n, count, cost):

    if (count == n and graph[currPos][0]):
        answer.append(cost + graph[currPos][0])
        return

    # BACKTRACKING STEP
    # Loop to traverse the adjacency list
    # of currPos node and increasing the count
    # by 1 and cost by graph[currPos][i] value
    for i in range(n):
        if (v[i] == False and graph[currPos][i]):

            # Mark as visited
            v[i] = True
            tsp(graph, v, i, n, count + 1,
                cost + graph[currPos][i])

            # Mark ith node as unvisited
            v[i] = False

# n is the number of nodes i.e. V
if __name__ == '__main__':
    n = 4
    graph= [[ 0, 10, 15, 20 ],
            [ 10, 0, 35, 25 ],
            [ 15, 35, 0, 30 ],
            [ 20, 25, 30, 0 ]]

    # Boolean array to check if a node
    # has been visited or not
    v = [False for i in range(n)]

    # Mark 0th node as visited
    v[0] = True

    # Find the minimum weight Hamiltonian Cycle
    tsp(graph, v, 0, n, 1, 0)
```

# ans is the minimum weight Hamiltonian Cycle
print(min(answer))

```
    # Boolean array to check if a node
    # has been visited or not
    v = [False for i in range(n)]

    # Mark 0th node as visited
    v[0] = True

    # Find the minimum weight Hamiltonian Cycle
    tsp(graph, v, 0, n, 1, 0)

    # ans is the minimum weight Hamiltonian Cycle
    print(min(answer))
```

```
80
```