TRAVELLING SALESMAN PROBLEM

CODE :

```python
from sys import maxsize
from itertools import permutations
V = 4

# implementation of traveling Salesman Problem
def travellingSalesmanProblem(graph, s):

  # store all vertex apart from source vertex
  vertex = []
  for i in range(V):
    if i != s:
      vertex.append(i)

  # store minimum weight Hamiltonian Cycle
  min_path = maxsize
  next_permutation=permutations(vertex)
  for i in next_permutation:

    # store current Path weight(cost)
    current_pathweight = 0

    # compute current path weight
    k = s
    for j in i:
      current_pathweight += graph[k][j]
      k = j
    current_pathweight += graph[k][s]

    # update minimum
    min_path = min(min_path, current_pathweight)

  return min_path


# Driver Code
if __name__ == "__main__":

  # matrix representation of graph
  graph = [[0, 10, 15, 20], [10, 0, 35, 25],
      [15, 35, 0, 30], [20, 25, 30, 0]]
  s = 0
```

```
print(travellingSalesmanProblem(graph, s))
```

OUTPUT :

```
80
```

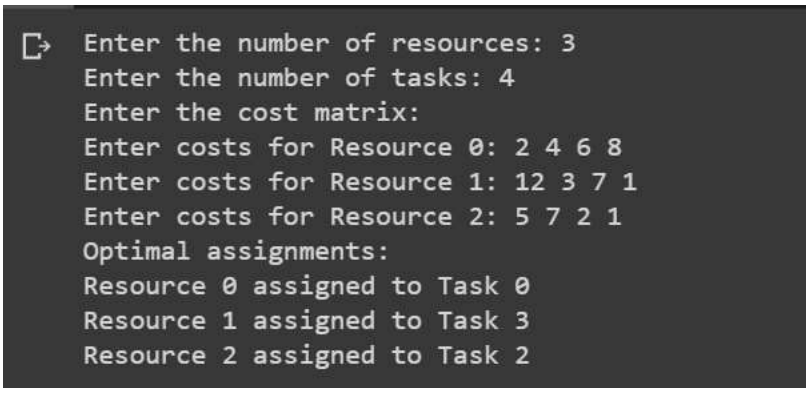
ASSIGNMENT PROBLEM

CODE :

```
from scipy.optimize import linear_sum_assignment

def get_integer_input(prompt):
    while True:
        try:
            value = int(input(prompt))
            return value
        except ValueError:
            print("Invalid input. Please enter an integer.")

def get_cost_matrix(num_resources, num_tasks):
    cost_matrix = []
    print("Enter the cost matrix:")
    for i in range(num_resources):
        while True:
            row = list(map(int, input(f"Enter costs for Resource {i}:
").split()))
            if len(row) == num_tasks:
                cost_matrix.append(row)
                break
            else:
                print(f"Invalid number of tasks. Expected {num_tasks}
tasks.")

    return cost_matrix

# Prompt the user for the number of resources and tasks
num_resources = get_integer_input("Enter the number of resources: ")
num_tasks = get_integer_input("Enter the number of tasks: ")

# Prompt the user for the cost matrix
cost_matrix = get_cost_matrix(num_resources, num_tasks)
```

```python
# Solve the assignment problem
row_indices, col_indices = linear_sum_assignment(cost_matrix)

# Print the optimal assignments
print("Optimal assignments:")
for row, col in zip(row_indices, col_indices):
    print(f"Resource {row} assigned to Task {col}")
```

OUTPUT :



```
Enter the number of resources: 3
Enter the number of tasks: 4
Enter the cost matrix:
Enter costs for Resource 0: 2 4 6 8
Enter costs for Resource 1: 12 3 7 1
Enter costs for Resource 2: 5 7 2 1
Optimal assignments:
Resource 0 assigned to Task 0
Resource 1 assigned to Task 3
Resource 2 assigned to Task 2
```