

LAB 2 – TRAVELLING SALESMAN PROBLEM

AI LAB

SHUBHAM SHARMA

RA1911003010649

ALGORITHM:

Step 1: Consider city 1 as the starting and ending point.

Step 2: Generate all $(n-1)!$ Permutations of cities.

Step 3: Calculate cost of every permutation and keep track of minimum cost permutation.

Step 4: Return the permutation with minimum cost.

CODE:

```
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("Minimum weight: ",travellingSalesmanProblem(graph, s))

```

OUTPUT:

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like 'TSP.py', '10 con puzzle.py', and 'README.md'. The code editor displays the following Python code:

```
1 # Python3 program to implement traveling salesman
2 # problem using naive approach.
3 from sys import maxsize
4 from itertools import permutations
5 V = 4
6
7 # Implementation of traveling Salesman Problem
8 def travellingSalesmanProblem(graph, s):
9
10     # store all vertex apart from source vertex
11     vertex = []
12     for i in range(V):
13         if i != s:
14             vertex.append(i)
15
16     # store minimum weight Hamiltonian Cycle
17     min_path = maxsize
18     next_permutation=permutations(vertex)
19     for i in next_permutation:
20
21         # store current Path weight(cost)
22         current_pathweight = 0
23
24         # compute current path weight
25         k = s
26         for j in i:
27             current_pathweight += graph[k][j]
```

The code is a Python3 program to implement the Traveling Salesman Problem using a naive approach. It defines a function `travellingSalesmanProblem` that takes a graph and a source vertex `s` as input. The function generates all permutations of vertices except the source vertex and calculates the total weight of each path. The minimum path weight is stored in `min_path`.

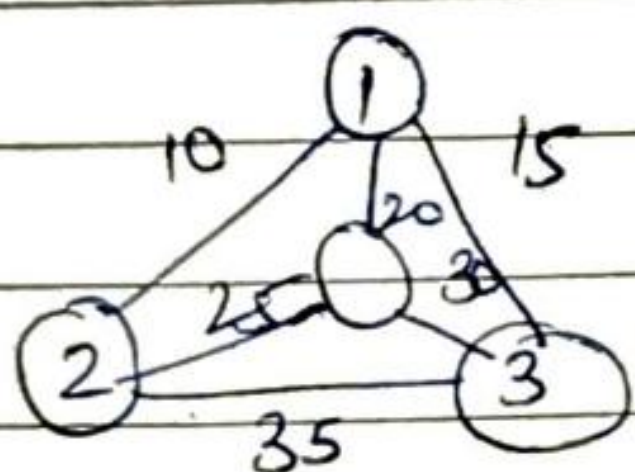
RESULT: Hence, the implementation of Travelling Salesman Person was successfully done.

AI Lab-2

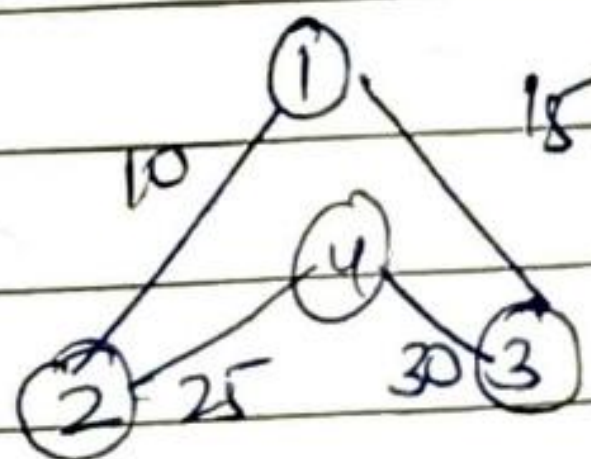
Aim - To develop agent programs for real world problem is Travelling Salesman Problem.

Problem formulation →

for a given complete graph with n vertices & weight w defined on the edges, the objective is to construct a tour, i.e. a circuit that passes through each vertex only once and return back to the starting with minimum total weight.



Initial state

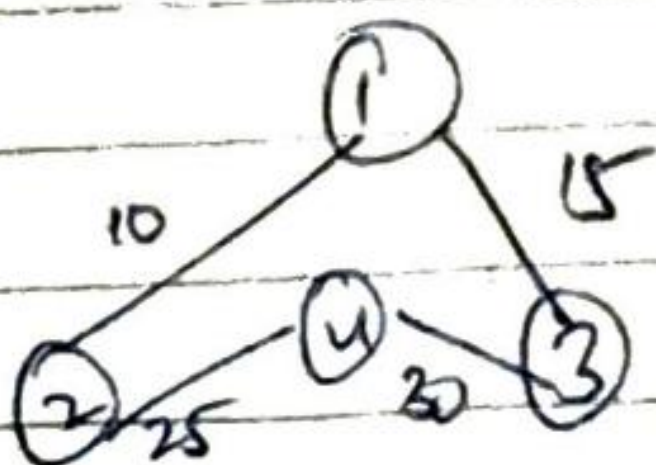


final state

Problem Solving :-

We start at vertex 1 and find the minimum cost path with 1 as starting point & all vertices appearing exactly once.

Now first we find a path for the same condition and try various permutations to find the min path out of various path. For this example, it is 1-2-4-3-1



Min. weight = 80