VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

BACHUPALLY (Via), KUKATPALLY – 500090

TAMASOMA JYOTIRGAMAYA

NAME: G. VAISHNAVI

ROLL NO.: 21071A6717

DEPARTMENT OF CSE-DS, CYS AND AI&DS

FLAT ASSIGNMENT

What is TSP?

Travelling Salesman Problem (TSP):

Given a set of cities and the distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. Note the difference between Hamiltonian Cycle and TSP. The Hamiltonian cycle problem is to find if there exists a tour that visits every city exactly once. Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact, many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.

Euler1:

For example, consider the graph shown in the figure on the right side. A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80. The problem is a famous NP-hard problem. There is no polynomial-time know solution for this problem. The following are different solutions for the traveling salesman problem.

Naive Solution:

1) Consider city 1 as the starting and ending point.

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

3) Calculate the cost of every permutation and keep track of the minimum cost permutation.

4) Return the permutation with minimum cost.

Time Complexity: Θ(n!)

Dynamic Programming:

Let the given set of vertices be {1, 2, 3, 4,… .n}. Let us consider 1 as starting and ending point of output. For every other vertex I (other than 1), we find the minimum cost path with 1 as the starting point, I as the ending point, and all vertices appearing exactly once. Let the cost of this path cost (i), and the cost of the corresponding Cycle would cost (i) + dist(i, 1) where dist(i, 1) is the distance from I to 1. Finally, we return the minimum of all [cost(i) + dist(i, 1)] values. This looks simple so far.

Now the question is how to get cost(i)? To calculate the cost(i) using Dynamic Programming, we need to have some recursive relation in terms of sub-problems.

Let us define a term C(S, i) be the cost of the minimum cost path visiting each vertex in set S exactly once, starting at 1 and ending at i. We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset, then we calculate C(S, i) for all subsets S of size 3 and so on. Note that 1 must be present in every subset.

If size of S is 2, then S must be {1, i},

C(S, i) = dist(1, i)

Else if size of S is greater than 2.

C(S, i) = min { C(S-{i}, j) + dis(j, i)} where j belongs to S, j != i and j != 1.

Below is the dynamic programming solution for the problem using top down recursive+memoized approach:-

For maintaining the subsets we can use the bitmasks to represent the remaining nodes in our subset. Since bits are faster to operate and there are only few nodes in graph, bitmasks is better to use.

For example: –

10100 represents node 2 and node 4 are left in set to be processed

010010 represents node 1 and 4 are left in subset.

NOTE:- ignore the 0th bit since our graph is 1-based

#code for travelling sales person

```
n = 4
dist = [[0, 0, 0, 0, 0], [0, 0, 10, 15, 20], [
    0, 10, 0, 25, 25], [0, 15, 25, 0, 30], [0, 20, 25, 30, 0]]
memo = [[-1]*(1 << (n+1)) for _ in range(n+1)]


def fun(i, mask):
    if mask == ((1 << i) | 3):
        return dist[1][i]
    if memo[i][mask] != -1:
        return memo[i][mask]
    res = 10**9  # result of this sub-problem
    for j in range(1, n+1):
        if (mask & (1 << j)) != 0 and j != i and j != 1:
            res = min(res, fun(j, mask & (~(1 << i))) + dist[j][i])
    memo[i][mask] = res  # storing the minimum value
    return res
# Driver program to test above logic
ans = 10**9
for i in range(1, n+1):
    # try to go from node 1 visiting all nodes in between to i
    # then return from i taking the shortest route to 1
    ans = min(ans, fun(i, (1 << (n+1))-1) + dist[i][1])
print("The cost of most efficient tour = " + str(ans))
```

```
}
int main()
{
        int graph[][V] = {{ 0, 10, 15, 20 },
                                { 10, 0, 35, 25 },
                                { 15, 35, 0, 30 },
                                { 20, 25, 30, 0 }};
        int s = 0;
        cout << travllingSalesmanProblem(graph, s) << endl;
        return 0;
}
```

#ALGORITHM Travelling Sales Person

```
C ({1}, 1) = 0
for s = 2 to n do
  for all subsets S Є {1, 2, 3, … , n} of size s and containing 1
    C (S, 1) = ∞
  for all j Є S and j ≠ 1
    C (S, j) = min {C (S – {j}, i) + d(i, j) for i Є S and i ≠ j}
Return minj C ({1, 2, 3, … , n}, j) + d(j, i)
```

ApplicationsoftheTravellingSalesPersonAlgorithm:

1. Drilling of printed Circuit Boards
2. Overhauling gas turbine engines
3. Computer wiring
4. The order-picking problem in warehouses
5. Vehicle routing