**Question:** Implement a dynamic programming solution for the Travelling Salesman Problem.

**Algorithm:**

Start

Step 1: Initialise an integer variable n with the number of cities, an integer variable VISITED\_ALL with 2n -1.

Step 2: Initialise a 2D adjacency matrix dist of the city and another 2D matrix dp

Algorithm for method main():

Start

Step 1: Initialise all the elements of dp with -1

Step 2: Call subroutine tsp with 1 and 0 as parameters and display the result

Stop

Algorithm for subroutine tsp(int, int):

Start

Step 1: Accept two integer variables mask and pos as parameters.

Step 2: if mask = VISITED\_ALL

1. return dist[pos][0]

Step 3: if dp[mask][pos] != -1

1. return dp[mask][pos]

Step 4: Initialise an integer variable ans with ∞

Step 5: Run a loop for city = 0 to n – 1

1. if mask AND 2city = 0
   1. newAns <-- dist[pos][city] + tsp(mask | (2city), city)
   2. ans <-- min(ans, newAns)

Step 6: dp[mask][pos] <-- ans

Step 7: return dp[mask][pos]

Stop

Stop

**Code:**

#include<iostream>

using namespace std;

#define INT\_MAX 999999

int n=4;

int dist[10][10] = {

{0,20,42,25},

{20,0,30,34},

{42,30,0,10},

{25,34,10,0}

};

int VISITED\_ALL = (1<<n) -1;

int dp[16][4];

int tsp(int mask,int pos){

if(mask==VISITED\_ALL){

return dist[pos][0];

}

if(dp[mask][pos]!=-1){

return dp[mask][pos];

}

int ans = INT\_MAX;

for(int city=0;city<n;city++){

if((mask&(1<<city))==0){

int newAns = dist[pos][city] + tsp( mask|(1<<city), city);

ans = min(ans, newAns);

}

}

return dp[mask][pos] = ans;

}

int main(){

for(int i=0;i<(1<<n);i++){

for(int j=0;j<n;j++){

dp[i][j] = -1;

}

}

cout<<"Travelling Saleman Distance is "<<tsp(1,0)<<'\n';

return 0;

}

**Output:**

