TSP by Dynamic Programming
Algorithm using masks to trace visited and unvisited cities

1) Input: Number of cities=n, Distance matrix 'dist' of input graph

2) Initially mask=1; So initial function call of g(int mask, int position) is g(1,0); It is a function to compute g(i,S)

   [mask is n bit binary code in which city indexing starts from 0 to n-1 from R.H.S.

   i^{th} bit =1 indicates i^{th} city is visited and if i^{th} =0 indicates i^{th} city is unvisited.

   e.g If number of cities=n=4 then we have mask values from 0000 to 1111.

   Mask value 0001 indicates 0^{th} city (city1) is visited.

   Mask value 0011 indicates 0^{th} city (city1) and 1^{st} city (city2) are visited.

   Mask value 0101 indicates 0^{th} city (city1) and 2^{nd} city (city3) are visited.

   Mask value 1111 indicates all 4 cities are visited and so on.]

3) int **all_visited = (1<<n)-1;**

   /*Left shift number 1 by n positions and then subtract 1 from it.

   E.g. If n=4 , and mask=1111 ➔ All 4 cities are visited.

   i.e. mask 1111=Decimal value (2^4 -1) = Binary (1<<4)-1*/

4) Initialize memoization table dp_table

   int dp_table[2^n][n]; // to store cost value of each state associated with specific mask & position

   for(int i=0;i<(1<<n);i++) // considering all masks from 0 to (1<<n)-1

   for(int j=0; j<n; j++) // considering n cities

   dp_table[i][j]= -1; // initialization

5) //Function to compute g(i,S)

   int g(int mask, int position)

   {

   //Base case |S|={}

   if(mask==all_visited)

```
        {
           return=dist[position][0]; //path from the last city to starting city(0th city)
        }
        //Lookup memoization table
           if (dp_table[mask][position]!= -1)
                return dp_table[mask][position];

         //Goto unvisited cities & estimate the minimum cost
       int mincost=999999;  //initialization to max limit
       for(int curr_city = 0; curr_city<n; curr_city++)
       {
            // check for unvisited city
            if((mask &(1<<curr_city))==0) //if city is unvisited
           {
             int cost = dist[position][curr_city] + g(mask | (1<<curr_city), curr_city);
                        /*bitwise OR function updates mask value and
                          function g() recursively estimates cost of further path.*/
             int mincost= min( mincost,cost);
           }
       }
         dp_table[mask][position]=mincost;
   }
```

6) Trace the path by storing the values of curr_city that gives minimum cost value in function g().
7) Output: optimal cost and path of a tour