NAME: Meet Raut

DIV: S2-1

ROLL.NO: 2201084

Experiment 9:

- **AIM: To study and implement Traveling Salesman Problem.**

- **THEORY:**

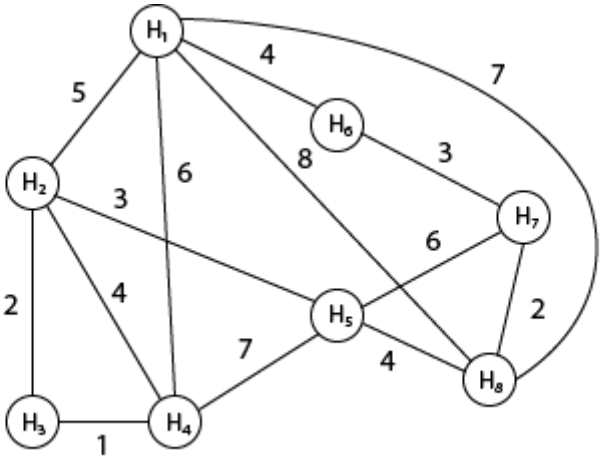
The traveling salesman problems abide by a salesman and a set of cities. The salesman has to visit every one of the cities starting from a certain one (e.g., the hometown) and to return to the same city. The challenge of the problem is that the traveling salesman needs to minimize the total length of the trip.

Suppose the cities are x_1 x_2..... x_n where cost c_{ij} denotes the cost of travelling from city x_i to x_j. The travelling salesperson problem is to find a route starting and ending at x_1 that will take in all cities with the minimum cost.

In Traveling Salesman Problem, we see a complete undirected graph and a distance between each village i.e.

Example: A newspaper agent daily drops the newspaper to the area assigned in such a manner that he has to cover all the houses in the respective area with minimum travel cost. Compute the minimum travel cost.

The area assigned to the agent where he has to drop the newspaper is shown in fig:



Solution: The cost- adjacency matrix of graph G is as follows:

$cost_{ij}$ =

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
H_1	0	5	0	6	0	4	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
H_6	4	0	0	0	0	0	3	0
H_7	0	0	0	0	6	3	0	2
H_8	7	0	0	0	4	0	2	0

The tour starts from area H_1 and then select the minimum cost area reachable from H_1.

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
(1) H_1	0	5	0	6	0	(4)	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
H_6	4	0	0	0	0	0	3	0
H_7	0	0	0	0	6	3	0	2
H_8	7	0	0	0	4	0	2	0

Mark area H_6 because it is the minimum cost area reachable from H_1 and then select minimum cost area reachable from H_6.

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
(1) H_1	0	5	0	6	0	(4)	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
(2) H_6	4	0	0	0	0	0	(3)	0
H_7	0	0	0	0	6	3	0	2
H_8	7	0	0	0	4	0	2	0

Mark area H_7 because it is the minimum cost area reachable from H_6 and then select minimum cost area reachable from H_7.

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
(**H_1**)	0	5	0	6	0	(4)	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
(**H_6**)	4	0	0	0	0	0	(3)	0
(**H_7**)	0	0	0	0	6	3	0	(2)
H_8	7	0	0	0	4	0	2	0

Mark area H_8 because it is the minimum cost area reachable from H_8.

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
(**H_1**)	0	5	0	6	0	(4)	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
(**H_6**)	4	0	0	0	0	0	(3)	0
(**H_7**)	0	0	0	0	6	3	0	(2)
(**H_8**)	7	0	0	0	(4)	0	2	0

Mark area H_5 because it is the minimum cost area reachable from H_5.

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
(H1)	0	5	0	6	0	(4)	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
(H5)	0	(3)	0	7	0	0	6	4
(H6)	4	0	0	0	0	0	(3)	0
(H7)	0	0	0	0	6	3	0	(2)
(H8)	7	0	0	0	(4)	0	2	0

Mark area H_2 because it is the minimum cost area reachable from H_2

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
(H1)	0	5	0	6	0	(4)	0	7
(H2)	5	0	(2)	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
(H5)	0	(3)	0	7	0	0	6	4
(H6)	4	0	0	0	0	0	(3)	0
(H7)	0	0	0	0	6	3	0	(2)
(H8)	7	0	0	0	(4)	0	2	0

Mark area H_3 because it is the minimum cost area reachable from H_3.

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
H_1	0	5	0	6	0	4	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
H_6	4	0	0	0	0	0	3	0
H_7	0	0	0	0	6	3	0	2
H_8	7	0	0	0	4	0	2	0

Mark area H_4 and then select the minimum cost area reachable from H_4 it is H_1.So, using the greedy strategy, we get the following.

$$H_1 \xrightarrow{4} H_6 \xrightarrow{3} H_7 \xrightarrow{2} H_8 \xrightarrow{4} H_5 \xrightarrow{3} H_2 \xrightarrow{2} H_3 \xrightarrow{1} H_4 \xrightarrow{6} H_1.$$

Thus, the minimum travel cost = 4 + 3 + 2 + 4 + 3 + 2 + 1 + 6 = 25

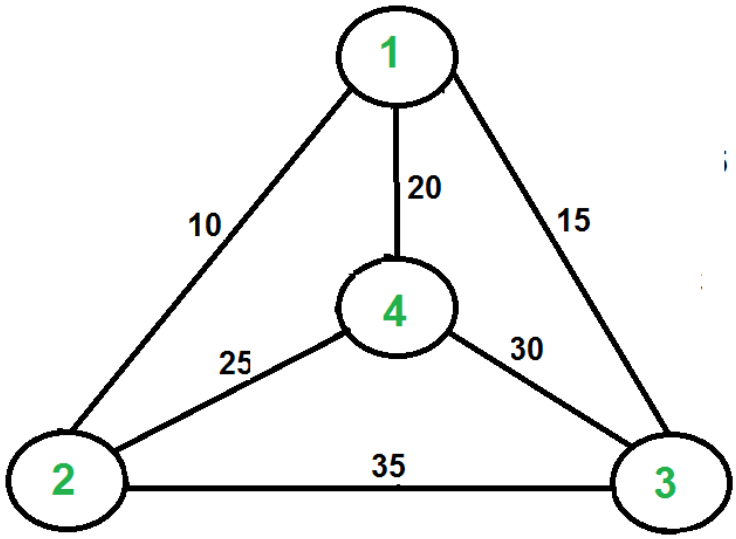
Matroids:

A matroid is an ordered pair M(S, I) satisfying the following conditions:

1. S is a finite set.
2. I is a nonempty family of subsets of S, called the independent subsets of S, such that if $B \in I$ and $A \in I$. We say that I is hereditary if it satisfies this property. Note that the empty set $\emptyset$ is necessarily a member of I.
3. If $A \in I$, $B \in I$ and $|A| < |B|$, then there is some element $x \in B - A$ such that $A \cup \{x\} \in I$. We say that M satisfies the exchange property.

We say that a matroid M (S, I) is weighted if there is an associated weight function w that assigns a strictly positive weight w (x) to each element $x \in S$. The weight function w extends to a subset of S by summation:

$$w(A) = \sum_{x \in A} w(x)$$



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: (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

Three popular Travelling Salesman Problem Algorithms

Here are some of the most popular solutions to the Travelling Salesman Problem:

1. The brute-force approach

The Brute Force approach, also known as the Naive Approach, calculates and compares all possible permutations of routes or paths to determine the shortest unique solution. To solve the TSP using the Brute-Force approach, you must calculate the total number of routes and then draw and list all the possible routes. Calculate the distance of each route and then choose the shortest one—this is the optimal solution.

This is only feasible for small problems, rarely useful beyond theoretical computer science tutorials.

2. The branch and bound method

The branch and bound algorithm starts by creating an initial route, typically from the starting point to the first node in a set of cities. Then, it systematically explores different permutations to extend the route one node at a time. Each time a new node is added, the algorithm calculates the current path's length and compares it to the optimal route found so far. If the current path is already longer than the optimal route, it "bounds" or prunes that branch of the exploration, as it would not lead to a more optimal solution.

This pruning is the key to making the algorithm efficient. By discarding unpromising paths, the search space is narrowed down, and the algorithm can focus on exploring only the most promising paths. The process continues until all possible routes are explored, and the shortest one is identified as the optimal solution to the traveling salesman problem. Branch and bound is an effective greedy approach for tackling NP-hard optimization problems like the travelling salesman problem.

3. The nearest neighbor method

To implement the Nearest Neighbor algorithm, we begin at a randomly selected starting point. From there, we find the closest unvisited node and add it to the sequencing. Then, we move to the next node and repeat the process of finding the nearest unvisited node until all nodes are included in the tour. Finally, we return to the starting city to complete the cycle.

While the Nearest Neighbor approach is relatively easy to understand and quick to execute, it rarely finds the optimal solution for the traveling salesperson problem. It can be significantly longer than the optimal route, especially for large and complex instances. Nonetheless, the Nearest Neighbor algorithm serves as a good starting point for tackling the travelling salesman problem and can be useful when a quick and reasonably good solution is needed.

This greedy algorithm can be used effectively as a way to generate an initial feasible solution quickly, to then feed into a more sophisticated local search algorithm, which then tweaks the solution until a given stopping condition.

- **C PROGRAM:**

```
#include<stdio.h>

int ary[10][10], completed[10],  n, cost=0;

void takeInput()

{

int i, j;

printf("Enter the number of villages: ");

scanf("%d",&n);



printf("\nEnter the Cost Matrix\n");



for (i=0;i < n;i++)

{

printf("\nEnter Elements of Row: %d\n",i+1);
```

```
for( j=0;j < n;j++)
scanf("%d",&ary[i][j]);
completed[i]=0;
}

printf("\n\nThe cost list is:");

for( i=0;i < n;i++)
{
printf("\n");
for(j=0;j < n;j++)
printf("\t%d",ary[i][j]);
}
}

void mincost(int city)
{
int i,ncity;
completed[city]=1;

printf("%d--->",city+1);
```

```
ncity=least(city);


if(ncity==999)
{
ncity=0;
printf("%d",ncity+1);
cost+=ary[city][ncity];
return;
}
mincost(ncity);
}


int least(int c)
{
int i,nc=999;
int min=999,kmin;


for(i=0;i < n;i++)
{
if((ary[c][i]!=0)&&(completed[i]==0))
if(ary[c][i]+ary[i][c] < min)
{
```

```
min=ary[i][0]+ary[c][i];
kmin=ary[c][i];
nc=i;
}
}

if(min!=999)
cost=cost + kmin;
return nc;
}

int main()
{
takeInput();
printf("\n\nThe Path is:\n");
mincost(0); //passing 0 because starting vertex
printf("\n\nMinimum cost is %d\n ",cost);
return 0;
}
```

OUTPUT:

```
--------------- TRAVELING SALESMAN PROBLEM (TSP) ---------------

Enter the number of villages: 4

Enter the Cost Matrix

Enter Elements of Row: 1
0 4 1 3

Enter Elements of Row: 2
4 0 2 1

Enter Elements of Row: 3
1 2 0 5

Enter Elements of Row: 4
3 1 5 0


The cost list is:
    0   4   1   3
    4   0   2   1
    1   2   0   5
    3   1   5   0

The Path is:
1--->3--->2--->4--->1

Minimum cost is 7
```

- **<u>CONCLUSION:</u> Hence, we have successfully implemented Traveling Salesman Problem (TSP); LO 1, LO 2.**