Experiment Number: 9

Aim:- Implementation of Travelling Salesman Problem using Branch & Bound.

Problem Statement:- Determining the path of minimum cost for a given directed graph of Travelling Salesperson Problem by using Branch and Bound.

Theory:-

In Travelling Salesperson problem, a directed graph G = (V, E) with n vertices and edge costs c_{ij} is given. Edge cost $c_{ij} > 0$ for all i and j and $c_{ij} = \infty$ if $\langle i,j \rangle \notin E$.

A tour of G is a directed simple cycle that includes every vertex in V. The cost of the tour is sum of the cost of the edges on the tour. The travelling salesperson problem is to find a tour of minimum cost. A tour is a simple path that starts and ends at vertex 1.

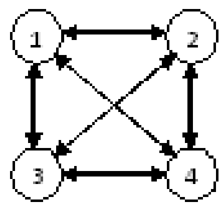


Fig.1. Directed Graph

$$\begin{pmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{pmatrix}$$

Fig.2. Edge length matrix c

Optimal Solution for TSP using Branch and Bound

A branch-and-bound algorithm consists of a systematic enumeration of all candidate solutions, where large subsets of fruitless candidate s are discarded ,by using upper and lower estimated bounds of the quantity being optimized.

The Branch and Bound strategy divides a problem to be solved into a number of sub-problems. It is a system for solving a sequence of sub problems each of which may have multiple possible solutions and where the solution chosen for one sub-problem may affect the possible solutions of later sub-problems.

Suppose it is required to minimize an objective function. Suppose that we have a method for getting a lower bound on the cost of any solution among those in the set of solutions represented by some subset. If the best solution found so far costs less than the lower bound for this subset, we need not explore this subset at all.

Let S be some subset of solutions.

L(S)=a lower bound on the cost of any solution belonging to S Let C=cost of the best solution found so far

If C ≤ L(S),there is no need to explore S because it does not contain any better solution.

If C > L(S),then we need to explore S because it may contain a better solution.

Algorithm:

function CheckBounds(st,des,cost[n][n]) [3] //Cal the bounds

Global variable: cost[N][N] - the cost assignment.

```
    pencost[0] = t
    for i ← 0, n − 1 do
        for j ← 0, n − 1 do
            reduced[i][j] = cost[i][j]
        end for
    end for
    for j ← 0, n − 1 do
        reduced[st][j] = ∞
    end for
    for i ← 0, n − 1 do
        reduced[i][des] = ∞
    end for
    reduced[des][st] = ∞
    RowReduction(reduced)
    ColumnReduction(reduced)
    pencost[des] = pencost[st] + row + col + cost[st][des]
    return pencost[des]
end function
```

function RowMin(cost[n][n],i) .//Cal. min in the row

```
    min = cost[i][0]
    for j ← 0, n − 1 do
        if cost[i][j] < min then
            min = cost[i][j]
        end if
    end for
    return min
end function
```

function ColMin(cost[n][n],i) .// Cal. min in the col

```
    min = cost[0][j]
    for i ← 0, n − 1 do
        if cost[i][j] < min then
            min = cost[i][j]
        end if
    end for
    return min
end function
```

function Rowreduction(cost[n][n]) .// makes row reduction

```
    row = 0
    for i ← 0, n − 1 do
        rmin = rowmin(cost, i)
        if rmin 6= ∞ then
            row = row + rmin
        end if
```

```
        for j ← 0, n − 1 do
            if cost[i][j] 6= ∞ then
                cost[i][j] = cost[i][j] − rmin
            end if
        end for
    end for
end function
```

function Columnreduction(cost[n][n]) //makes column reduction

```
    col = 0
    for j ← 0, n − 1 do
        cmin = columnmin(cost, j)
        if cmin 6= ∞ then
            col = col + cmin
        end if
        for i ← 0, n − 1 do
            if cost[i][j] 6= ∞ then
                cost[i][j] = cost[i][j] − cmin
            end if
        end for
```

```
    end for
end function
function Main                                                    // main function
    for i ← 0, n − 1 do
        select[i] = 0
    end for
    rowreduction(cost)
    columnreduction(cost)
    t = row + col
    while allvisited(select) 6= 1 do
        for i ← 1, n − 1 do
            if select[i] = 0 then
                edgecost[i] = checkbounds(k, i, cost)
            end if
        end for
        min = ∞
        for i ← 1, n − 1 do
            if select[i] = 0 then
                if edgecost[i] < min then
                    min = edgecost[i]
                    k = i
                end if
            end if
        end for
        select[k] = 1
        for p ← 1, n − 1 do
            cost[j][p] = ∞
        end for
        for p ← 1, n − 1 do
            cost[p][k] = ∞
        end for
        cost[k][j] = ∞
        rowreduction(cost)
        columnreduction(cost)
    end while
end function
```

NAME: HARSHBHAI SOLANKI
ROLL.N0:62
SE-4-D

EXPERIMENT N0- 9

AIM: Implementation of Travelling Salesman Problem using Branch & Bound.

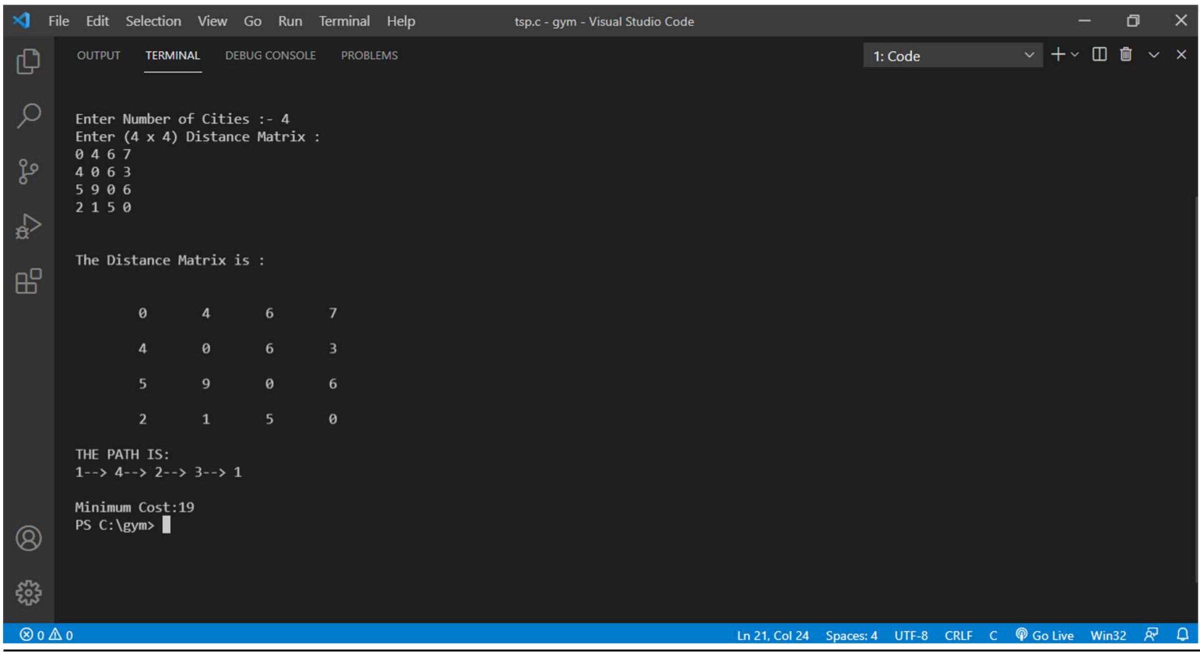
CODE:

```
#include<stdio.h>
#include<conio.h>
int DistanceMatrix[10][10],VisitedCities[10],n,cost=0;
void getData()
{
	int i,j;
	printf("\n\nEnter Number of Cities :- ");
	scanf("%d",&n);
	printf("Enter (%d x %d) Distance Matrix : \n",n,n);
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
		{
			scanf("%d",&DistanceMatrix[i][j]);
		}
		VisitedCities[i]=0;
	}
	printf("\n\nThe Distance Matrix is : \n");
	for(i=0;i<n;i++)
	{
		printf("\n\n");
		for(j=0;j<n;j++)
```

```
                {
                        printf("\t%d",DistanceMatrix[i][j]);
                }
        }
}
void mincost(int city){
   int i,ncity;
   VisitedCities[city] = 1;
   printf("%d--> ",city+1);
   ncity=least(city);
   if(ncity==999){
      ncity=0;
      printf("%d",ncity+1);
      cost += DistanceMatrix[city][ncity];
      return;
   }
   mincost(ncity);
}
int least(int c){
   int i,nc=999;
   int min=999,kmin;
   for(i=0;i<=n;i++){
      if((DistanceMatrix[c][i]!=0) && (VisitedCities[i]==0))
         if(DistanceMatrix[c][i]<min){
            min = DistanceMatrix[i][0] + DistanceMatrix[c][i];
            kmin=DistanceMatrix[c][i];
            nc=i;
```

```
        }
    }
    if(min != 999)
    {
        cost += kmin;
    }
    return nc;
}
void DisplayPath(){
    printf("\n\nMinimum Cost:");
    printf("%d",cost);
}
int main(){
    getData();
    printf("\n\nTHE PATH IS: \n");
    mincost(0);
    DisplayPath();
}
```

OUTPUT:



CONCLUSION:

By performing the Travelling Salesman problems we can that:

•The worst case complexity of Branch and Bound remains same as that of the Brute Force clearly because in worst case, we may never get a chance to prune a node.

•But in practice it performs very well depending on the different instance of the TSP.

•The complexity also depends on the choice of the bounding function as they are the ones deciding how many nodes to be pruned.

•The time complexity of the program is O(n^2) as explained above for the row and column reduction functions.