Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.
Department of Master of Computer Application

Experiment	9
Aim	To implement branch and bound algorithm (To implement Travelling salesman problem)
Objective	To solve the Traveling Salesman Problem using the Branch and Bound algorithm and display the optimal path and minimum cost.
Name	Vivek Tiwari
UCID	2023510059
Class	FYMCA
Batch	C

Algorithm and Explanation of the technique used

```
func tspBranchAndBound(graph[][], path[], visited[], minCost, cost, pos)
    if pos == graph.size()
        minCost = min(minCost, cost + graph[path[pos - 1]][path[0]])
        return

    for each vertex i in graph.size()
        if not visited[i] and cost + graph[path[pos - 1]][i] < minCost
            visited[i] = true
            path[pos] = i
            tspBranchAndBound(graph, path, visited, minCost, cost +
graph[path[pos - 1]][i], pos + 1)
            visited[i] = false

func TravelingSalesman(graph[][], n)
    initialize path[] with size n
    initialize visited[] with size n
    path[0] = 0
    visited[0] = true
    minCost = INF
    tspBranchAndBound(graph, path, visited, minCost, 0, 1)

    if minCost == INF
        print "No feasible solution exists."
    else
        print "Minimum Cost:", minCost
```

Q.

$$\begin{array}{c|ccccc|c} & 1 & 2 & 3 & 4 & 5 & \\ \hline 1 & \infty & 20 & 30 & 10 & 11 & 10 \\ 2 & 15 & \infty & 16 & 4 & 2 & 2 \\ 3 & 3 & 5 & \infty & 2 & 4 & 2 \\ 4 & 19 & 6 & 18 & \infty & 3 & 3 \\ 5 & 16 & 4 & 7 & 16 & \infty & 4 \end{array} \quad 21$$

row reduced

$$\begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix} \xrightarrow{\text{col reduced}} \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

1 0 3 0 0 / 4

$\boxed{M1}$

21+4 = 25 = Reduction cost

$\boxed{\text{At 1 cost = 25}}$

Consider 1 to 2

row 1 = col 2 = ∞ & 2 to 1 = ∞ from M1

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

0 0 0 0 0

Rows & columns already reduced.

∴ cost = 0

Total cost = previous reduction cost + current cost + C(1,2)

= 25 + 0 + 10

= 35

∴ $\boxed{\text{1 to 2 = 35}}$

Consider 1 to 3. Row 1 = col 3 = (3 to 1) = ∞ in M1

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

11 0 0 0 0 / 11

Total = 25 + 11 + 17 = 53

$\boxed{\text{1 to 3 = 53}}$

Consider 1 to 4. Row 1 = col 4 = 4 to 1 = ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

0 0 0 0 0

∴ Row & col already reduced

∴ cost = 0

$\boxed{M2}$

Total = 25 + 0 + 0 = 25. $\boxed{\text{1 to 4 = 25}}$

1 to 5 ⇒

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix} \begin{matrix} 0 \\ 2 \\ 0 \\ 3 \\ 0 \end{matrix} \rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

5

0 0 0 0 0

Total cost = 25 + (2+3) + 0 + 1

= 31

$\boxed{\text{1 to 5 = 31}}$

(1) 25

(2) (3) (4) (5)

35 53 25 31

× × ×

1 to 4 is minimum ∴ Consider 1 to 4 as $\boxed{M2}$

Consider 1 to 4 to 2

4 row = 2 column = 2 to 1 = ∞ in M2

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

0 0 0 0 0

Already reduced

$\boxed{M3}$

Total = previous + current + C(4,2) in M2

= 25 + 0 + 3 = 28

$\boxed{\text{1 to 4 to 2 = 28}}$

Consider 1 to 4 to 3

4 row = 3 col = 3 to 1 = ∞ in M2

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{matrix} \rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix}$$

2

11 0 0 0 0 / 11

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$$

Total = 25 + 13 + 12 = 50. $\boxed{\text{1 to 4 to 3 = 50}}$

Consider 1 to 4 to 5

4 row = 5 column = 5 to 1 = ∞ in M2

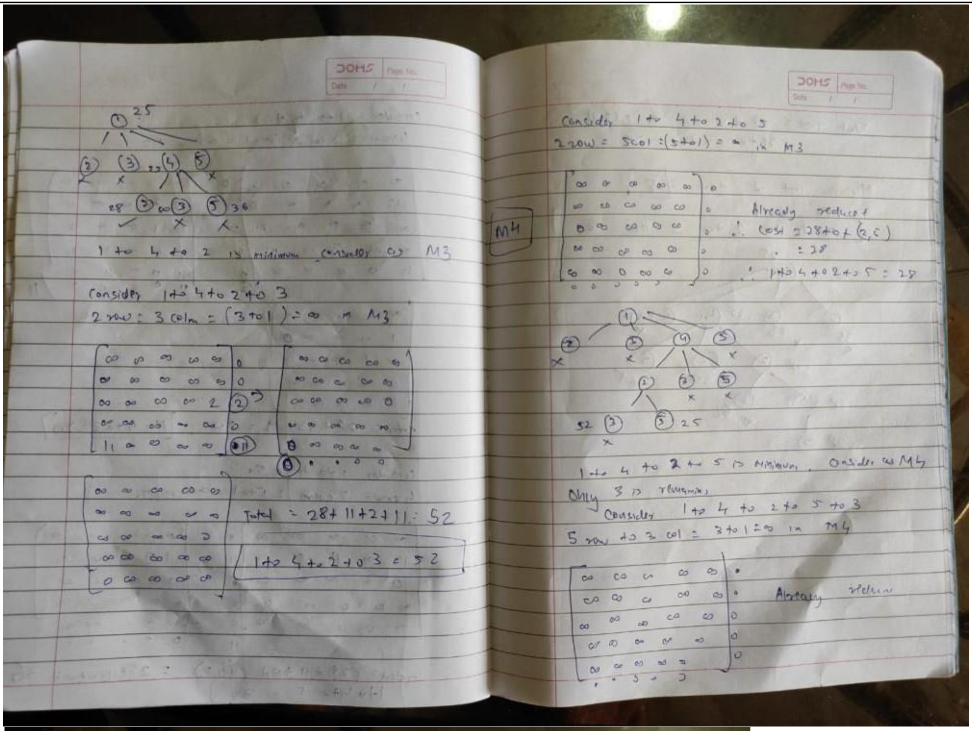
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix} \begin{matrix} 0 \\ 11 \\ 0 \\ 0 \\ 0 \end{matrix} \rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix}$$

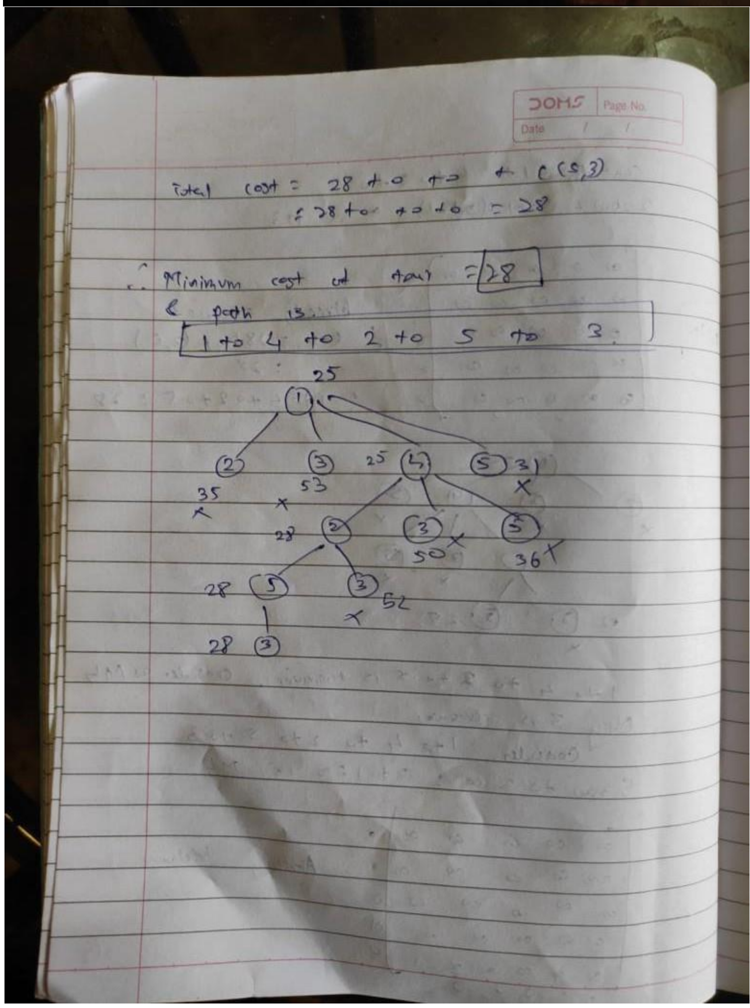
11

0 0 0 0 0

Total = 25 + 11 + 0 + C(4,5) = 25 + 11 + 0 + 0 = 36

$\boxed{\text{1 to 4 to 5 = 36}}$





Program(Code)

```
#include <iostream>
#include <vector>
```

```cpp
#include <algorithm>
#include <climits>
using namespace std;

const int INF = INT_MAX;

int calculateCost(const vector<vector<int>>& graph, const
vector<int>& path) {
    int cost = 0;
    for (int i = 0; i < path.size() - 1; ++i) {
        if (graph[path[i]][path[i + 1]] == INF)
            return INF;
        cost += graph[path[i]][path[i + 1]];
    }
    cost += graph[path.back()][path[0]];
    return cost;
}

void tspBranchAndBound(const vector<vector<int>>& graph,
vector<int>& path, vector<bool>& visited, vector<int>&
optimalPath, int& minCost, int cost, int pos) {
    if (pos == graph.size()) {
        int finalCost = cost + graph[path[pos - 1]][path[0]];
        if (finalCost < minCost) {
            minCost = finalCost;
            optimalPath = path;
            return;
        }
    }
    for (int i = 0; i < graph.size(); ++i) {
        if (!visited[i]) {
            visited[i] = true;
            path[pos] = i;
            if (cost + graph[path[pos - 1]][i] < minCost)
                tspBranchAndBound(graph, path, visited,
optimalPath, minCost, cost + graph[path[pos - 1]][i], pos + 1);
            visited[i] = false;
        }
    }
}

int main() {
    int n;
    cout << "Enter the number of cities: ";
    cin >> n;

    vector<vector<int>> graph(n, vector<int>(n));
```

```cpp
    cout << "Enter the cost matrix (Enter -1 for unreachable cities):\n";
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            cin >> graph[i][j];

    vector<int> path(n);
    vector<bool> visited(n, false);
    path[0] = 0;
    visited[0] = true;

    vector<int> optimalPath;
    int minCost = INF;
    tspBranchAndBound(graph, path, visited, optimalPath, minCost, 0, 1);

    if (minCost == INF)
        cout << "No feasible solution exists.";
    else {
        cout << "Optimal Path: ";
        for (int i = 0; i < n; ++i)
            cout << optimalPath[i]+1 << " ";
        cout << "1\n";
        cout << "Minimum Cost: " << minCost << endl;
    }

    return 0;
}
```

Output

```
Enter the number of cities: 5
Enter the cost matrix (Enter -1 for unreachable cities):
-1 20 30 10 11
15 -1 16 4 2
3 5 -1 2 4
19 6 18 -1 3
16 4 7 16 -1
Optimal Path: 1 4 2 5 3 1
Minimum Cost: 28
```

Justification of the complexity calculated

The time complexity of the Branch and Bound algorithm for the Traveling Salesman Problem is O(n!), where n is the number of vertices (cities) in the graph. This complexity arises from the algorithm's exploration of all possible permutations of the cities to find the optimal solution. At each level of recursion, the algorithm iterates over all remaining unvisited cities to extend the current partial tour, resulting in n - pos iterations. Although the algorithm prunes branches that cannot lead to an optimal solution through backtracking, in the worst case, it still explores a factorial number of permutations, leading to the O(n!) time complexity.

Conclusion	The Branch and Bound algorithm for the Traveling Salesman Problem offers significant advantages in solving optimization challenges. Its ability to systematically explore the solution space while intelligently pruning branches leads to efficient identification of the optimal solution. This algorithm finds applications across various domains, including logistics, transportation, and manufacturing. In logistics, it aids in route planning, ensuring the most cost-effective and time-efficient delivery schedules. In transportation, it helps in scheduling vehicle routes, minimizing fuel consumption and maximizing service coverage. Moreover, in manufacturing, it assists in optimizing production sequences, reducing operational costs and improving resource utilization. The Branch and Bound algorithm's versatility and effectiveness make it a valuable tool for tackling complex optimization problems, ultimately leading to improved efficiency and resource utilization in real-world scenarios.