**Name:** Rohan Arun Nalawade

**Roll No:** 231012

**PRN:** 22310407

**SY IT A**

**Assignment 8A**

**Code implementation:**

#include <iostream>

#include <climits>

using namespace std;

const int INF = INT\_MAX;

const int MAX = 20;

class TSP {

private:

    int n;

    int costMatrix[MAX][MAX];

    bool visited[MAX];

    int path[MAX], bestPath[MAX];

    int minCost;

    // Reduce the matrix and return cost of reduction

    int reduceMatrix(int matrix[MAX][MAX]) {

        int reductionCost = 0;

        // Row reduction

        for (int i = 0; i < n; i++) {

            int rowMin = INF;

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

                if (matrix[i][j] < rowMin)

                    rowMin = matrix[i][j];

            }

            if (rowMin != INF && rowMin > 0) {

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

                    if (matrix[i][j] != INF)

                        matrix[i][j] -= rowMin;

                }

                reductionCost += rowMin;

            }

        }

        // Column reduction

        for (int j = 0; j < n; j++) {

            int colMin = INF;

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

                if (matrix[i][j] < colMin)

                    colMin = matrix[i][j];

            }

            if (colMin != INF && colMin > 0) {

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

                    if (matrix[i][j] != INF)

                        matrix[i][j] -= colMin;

                }

                reductionCost += colMin;

            }

        }

        return reductionCost;

    }

    int calculateBound(int matrix[MAX][MAX], int currentCost) {

        int temp[MAX][MAX];

        for (int i = 0; i < n; i++)

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

                temp[i][j] = matrix[i][j];

        return currentCost + reduceMatrix(temp);

    }

    void solve(int matrix[MAX][MAX], int currentCost, int level) {

        if (level == n) {

            int totalCost = currentCost + costMatrix[path[level - 1]][path[0]];

            if (totalCost < minCost) {

                minCost = totalCost;

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

                    bestPath[i] = path[i];

            }

            return;

        }

        for (int i = 0; i < n; i++) {

            if (!visited[i] && matrix[path[level - 1]][i] != INF) {

                int temp[MAX][MAX];

                // Copy matrix

                for (int x = 0; x < n; x++)

                    for (int y = 0; y < n; y++)

                        temp[x][y] = matrix[x][y];

                // Set row and column to INF to eliminate future choices

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

                    temp[path[level - 1]][j] = INF;

                    temp[j][i] = INF;

                }

                temp[i][path[0]] = INF;

                int costToNext = costMatrix[path[level - 1]][i];

                int bound = calculateBound(temp, currentCost + costToNext);

                if (bound < minCost) {

                    visited[i] = true;

                    path[level] = i;

                    solve(temp, currentCost + costToNext, level + 1);

                    visited[i] = false;

                }

            }

        }

    }

public:

    TSP(int cities, int inputMatrix[MAX][MAX]) {

        n = cities;

        minCost = INF;

        for (int i = 0; i < n; i++) {

            visited[i] = false;

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

                costMatrix[i][j] = inputMatrix[i][j];

            }

        }

    }

    void run() {

        path[0] = 0;

        visited[0] = true;

        int initialMatrix[MAX][MAX];

        for (int i = 0; i < n; i++)

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

                initialMatrix[i][j] = costMatrix[i][j];

        int initialCost = reduceMatrix(initialMatrix);

        solve(initialMatrix, initialCost, 1);

    }

    void printResult() {

        cout << "\nBest Path: ";

        for (int i = 0; i < n; i++) {

            cout << bestPath[i] << " -> ";

        }

        cout << bestPath[0] << endl; // Closing the tour

        cout << "\nStep-by-step Costs:\n";

        int totalCost = 0;

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

            int from = bestPath[i];

            int to = bestPath[i + 1];

            cout << from << " -> " << to << " = " << costMatrix[from][to] << endl;

            totalCost += costMatrix[from][to];

        }

        // Return to start

        int last = bestPath[n - 1];

        int back = bestPath[0];

        cout << last << " -> " << back << " = " << costMatrix[last][back] << endl;

        totalCost += costMatrix[last][back];

        cout << "\nTotal Minimum Cost (from manual traversal): " << totalCost << endl;

    }

};

int main() {

    int n;

    int input[MAX][MAX];

    cout << "Enter number of cities: ";

    cin >> n;

    cout << "Enter the cost matrix (use -1 to represent no path):\n";

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            int val;

            cin >> val;

            input[i][j] = (val == -1 ? INF : val);

        }

    }

    TSP tsp(n, input);

    tsp.run();

    tsp.printResult();

    return 0;

}

**Output:**