ASSIGNMENT NO :- 05
ROLL NO :- 33252

```
#include<bits/stdc++.h>
using namespace std;
#define N 4

// FIND THE MINIMUM EDGE
int findMinEdge(int i, const vector<vector<int>>& costMatrix) {
    int min = INT_MAX;
    for (int j = 0; j < N; j++) {
        if (i != j && costMatrix[i][j] < min) {
            min = costMatrix[i][j];
        }
    }
    return min;
}

//CALCULATE THE LOWER BOUND
int calculateLowerBound(const vector<vector<int>>& costMatrix, const vector<bool>& visited, int
currentBound, int currentLevel) {
    int bound = currentBound;

    for (int i = 0; i < N; i++) {
        if (!visited[i]) {
            bound += findMinEdge(i, costMatrix);
        }
    }
    return bound;
}

// Recursive function to solve the TSP using branch and bound
void tspBranchAndBound(const vector<vector<int>>& costMatrix, vector<int>& currentPath,
vector<bool>& visited, int& finalCost, vector<int>& finalPath, int currentBound, int currentCost,
int currentLevel) {
    if (currentLevel == N) {
        if (costMatrix[currentPath[currentLevel - 1]][currentPath[0]] != 0) {
            int finalSolutionCost = currentCost + costMatrix[currentPath[currentLevel -
1]][currentPath[0]];

            if (finalSolutionCost < finalCost) {
                finalPath = currentPath;
                finalPath.push_back(currentPath[0]); // Complete the cycle
                finalCost = finalSolutionCost;
            }
        }
        return;
    }

    for (int i = 0; i < N; i++) {
        if (costMatrix[currentPath[currentLevel - 1]][i] != 0 && !visited[i]) {
            int temp = currentBound;
            currentCost += costMatrix[currentPath[currentLevel - 1]][i];

            if (currentLevel == 1) {
                currentBound -= (findMinEdge(currentPath[currentLevel - 1], costMatrix) +
findMinEdge(i, costMatrix)) / 2;
            } else {
                currentBound -= (findMinEdge(currentPath[currentLevel - 1], costMatrix) +
findMinEdge(i, costMatrix)) / 2;
            }

            if (currentBound + currentCost < finalCost) {
                currentPath[currentLevel] = i;
                visited[i] = true;

                tspBranchAndBound(costMatrix, currentPath, visited, finalCost, finalPath, currentBound,
currentCost, currentLevel + 1);
            }

            currentCost -= costMatrix[currentPath[currentLevel - 1]][i];
            currentBound = temp;

            fill(visited.begin(), visited.end(), false);
            for (int j = 0; j <= currentLevel - 1; j++) {
                visited[currentPath[j]] = true;
            }
        }
    }
}

// Main function to solve TSP using Branch and Bound
int solveTSP(const vector<vector<int>>& costMatrix) {
    vector<int> currentPath(N);
    vector<bool> visited(N, false);

    //INITIALIZE THE LOWER BOUND
    int currentBound = 0;
    for (int i = 0; i < N; i++) {
        currentBound += (findMinEdge(i, costMatrix) + findMinEdge(i, costMatrix));
    }
    currentBound = (currentBound & 1) ? currentBound / 2 + 1 : currentBound / 2;

    visited[0] = true;
    currentPath[0] = 0;

    vector<int> finalPath;
    int finalCost = INT_MAX;

    tspBranchAndBound(costMatrix, currentPath, visited, finalCost, finalPath, currentBound, 0, 1);

    cout << "Minimum cost: " << finalCost << endl;
    cout << "Path: ";
    for (int i : finalPath) {
        cout << i << " ";
    }
    cout << endl;

    return finalCost;
}

int main() {
    vector<vector<int>> costMatrix = {
        {0, 10, 15, 20},
        {10, 0, 35, 25},
        {15, 35, 0, 30},
        {20, 25, 30, 0}
    };

    solveTSP(costMatrix);
    return 0;
}
```

OUTPUT

```
mllab20@mllab20:~/Desktop/33252$ cd "/home/mllab20/Desktop/33252/0/" && g++
travelling_salesman.cpp -o travelling_salesman &&
"/home/mllab20/Desktop/33252/0/"travelling_salesman
Minimum cost: 80
Path: 0 1 3 2 0


mllab20@mllab20:~/Desktop/33252/0$ cd "/home/mllab20/Desktop/33252/0/" && g++
travelling_salesman.cpp -o travelling_salesman &&
"/home/mllab20/Desktop/33252/0/"travelling_salesman
Minimum cost: 60
Path: 0 1 2 0
```