**Experiment- 10**

**Aim-** Implement Traveling Salesman problem based on Branch and Bound technique.

**Theory-**

The Traveling Salesman Problem (TSP) requires finding the shortest possible route that visits every city exactly once and returns to the starting city. Using the Branch and Bound technique, we systematically explore all possible city sequences, but prune branches that exceed the current best solution. This approach reduces the number of potential paths that need to be explored, making it more efficient than brute-force solutions for solving TSP.

**Software Used –** Visual Studio Code

**Code-**

#include <iostream>

#include <vector>

#include <algorithm>

#include <cfloat>

using namespace std;

class TSPSolver {

int N;

vector<vector<int> > cost;

double final\_res;

vector<int> final\_path;

vector<bool> visited;

public:

TSPSolver(vector<vector<int> > cost\_matrix) {

cost = cost\_matrix;

N = cost.size();

final\_res = DBL\_MAX;

final\_path.resize(N + 1);

visited.resize(N, false);

}

void copyToFinal(vector<int>& curr\_path) {

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

final\_path[i] = curr\_path[i];

}

final\_path[N] = curr\_path[0];

}

int firstMin(int i) {

int min = INT\_MAX;

for (int k = 0; k < N; k++) {

if (cost[i][k] < min && i != k)

min = cost[i][k];

}

return min;

}

int secondMin(int i) {

int first = INT\_MAX, second = INT\_MAX;

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

if (i == j)

continue;

if (cost[i][j] <= first) {

second = first;

first = cost[i][j];

} else if (cost[i][j] <= second && cost[i][j] != first) {

second = cost[i][j];

}

}

return second;

}

void TSPRec(double curr\_bound, double curr\_weight, int level, vector<int>& curr\_path) {

if (level == N) {

if (cost[curr\_path[level - 1]][curr\_path[0]] != 0) {

double curr\_res = curr\_weight + cost[curr\_path[level - 1]][curr\_path[0]];

if (curr\_res < final\_res) {

copyToFinal(curr\_path);

final\_res = curr\_res;

}

}

return;

}

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

if (cost[curr\_path[level - 1]][i] != 0 && !visited[i]) {

double temp = curr\_bound;

curr\_weight += cost[curr\_path[level - 1]][i];

if (level == 1)

curr\_bound -= ((firstMin(curr\_path[level - 1]) + firstMin(i)) / 2.0);

else

curr\_bound -= ((secondMin(curr\_path[level - 1]) + firstMin(i)) / 2.0);

if (curr\_bound + curr\_weight < final\_res) {

curr\_path[level] = i;

visited[i] = true;

TSPRec(curr\_bound, curr\_weight, level + 1, curr\_path);

}

curr\_weight -= cost[curr\_path[level - 1]][i];

curr\_bound = temp;

fill(visited.begin(), visited.end(), false);

for (int j = 0; j <= level - 1; j++)

visited[curr\_path[j]] = true;

}

}

}

void solve() {

vector<int> curr\_path(N + 1, -1);

double curr\_bound = 0.0;

fill(visited.begin(), visited.end(), false);

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

curr\_bound += (firstMin(i) + secondMin(i));

curr\_bound = curr\_bound / 2.0;

visited[0] = true;

curr\_path[0] = 0;

TSPRec(curr\_bound, 0.0, 1, curr\_path);

cout << "Minimum cost: " << final\_res << endl;

cout << "Path taken: ";

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

cout << final\_path[i];

if (i != N)

cout << " -> ";

}

cout << endl;

}

};

int main() {

vector<vector<int> > cost\_matrix = {

{ 0, 10, 15, 20 },

{ 10, 0, 35, 25 },

{ 15, 35, 0, 30 },

{ 20, 25, 30, 0 }

};

TSPSolver tsp(cost\_matrix);

tsp.solve();

return 0;

}

**Output-**

**A black background with white text

Description automatically generated**