**LAB : 1**

**OBJECTIVE :**

To Implement Travelling Salesman problem

**Problem Statement :**

A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. What is the shortest possible route that he visits each city exactly once and returns to the origin city.

**Implementation :**

1. To print shortest possible route : We can use brute-force approach to evaluate every possible tour and select the best one. For **n** number of vertices in a graph, there are **(*n* - 1)!** number of possibilities.

**Code :**

#include<bits/stdc++.h>

using namespace std;

void INPUT()

{

#ifndef ONLINE\_JUDGE

freopen("C:/Users/arvin/Desktop/Current/input.txt", "r", stdin);

freopen("C:/Users/arvin/Desktop/Current/output.txt", "w", stdout);

#endif

}

int n ;

// Adj Matrix which defines the graphvector<vector<int>> dist;

list<int> sequence;

//If all cities have been visited

int VISITED\_ALL;

int ans\_overall = INT\_MAX;

int tsp(int mask, int pos, list<int> &temp, int cur\_cost) {

if (mask == VISITED\_ALL) {

if (ans\_overall > cur\_cost + dist[pos][0]) {

sequence = temp;

sequence.push\_back(0);

}

//all\_path[cur\_cost + dist[pos][0]] = temp;

return dist[pos][0];

}

int ans = INT\_MAX;

//visiting an unvisited city

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

if ((mask & (1 << city)) == 0) {

temp.push\_back(city);

int newAns = dist[pos][city] + tsp( mask | (1 << city), city, temp , cur\_cost + dist[pos][city]);

ans = min(ans, newAns);

temp.pop\_back();

}

}

return ans;

}

int main() {

INPUT();

cin >> n;

VISITED\_ALL = (1 << n) - 1;

dist = vector< vector<int> > (n + 1, vector<int> (n + 1, INT\_MAX));

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

{

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

{

cin >> dist[i][j];

}

}

list<int> temp;

temp.push\_back(0);

int tsp\_cost = tsp(1, 0, temp, 0);

cout << "Min weight hamiltonian path costs : " << tsp\_cost << endl;

cout << "\n\nPath: ";

for (auto i : sequence) {

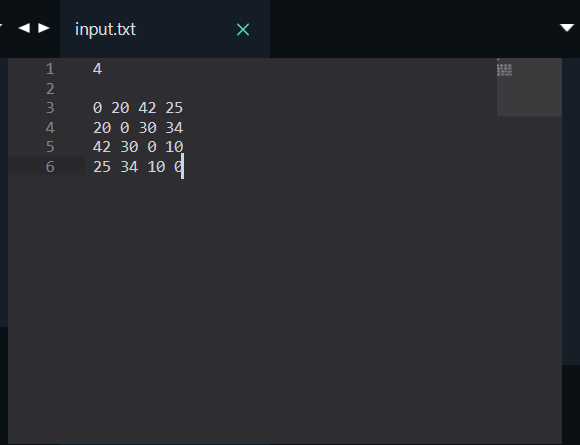
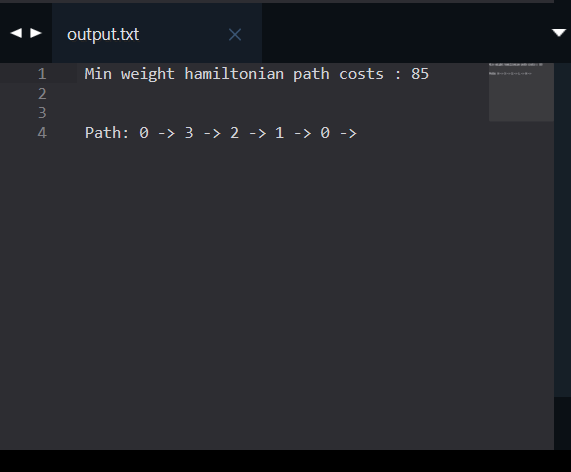
cout << i << " -> ";

}

return 0;

}

**Output :**

1. Optimised method to find minimum travelling cost using Dynamic Programming : Instead of brute-force using dynamic programming approach, the solution can be obtained in lesser time, though there is no polynomial time algorithm.

**Algorithm: Traveling-Salesman-Problem**

C ({1}, 1) = 0

for s = 2 to n do

for all subsets S Є {1, 2, 3, … , n} of size s and containing 1

C (S, 1) = ∞

for all j Є S and j ≠ 1

C (S, j) = min {C (S – {j}, i) + d(i, j) for i Є S and i ≠ j}

Return minj C ({1, 2, 3, …, n}, j) + d(j, i)

**Analysis :**

There are at the most 2*n*.*n* sub-problems and each one takes linear time to solve. Therefore, the total running time is *O*(2*n*.*n*2).

**Code :**

#include <iostream>

#include<vector>

#include<climits>

#include<cstring>

using namespace std;

void INPUT()

{

#ifndef ONLINE\_JUDGE

freopen("C:/Users/arvin/Desktop/Current/input.txt", "r", stdin);

freopen("C:/Users/arvin/Desktop/Current/output.txt", "w", stdout);

#endif

}

int n = 4;

vector<vector<int>> dp;

// Adj Matrix which defines the graph

vector<vector<int>> dist;

//If all cities have been visited

int VISITED\_ALL = (1 << n) - 1;

int tsp(int mask, int pos) {

if (mask == VISITED\_ALL) {

return dist[pos][0];

}

//Lookup

if (dp[mask][pos] != -1) {

return dp[mask][pos];

}

int ans = INT\_MAX;

//visiting an unvisited city

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

if ((mask & (1 << city)) == 0) {

int newAns = dist[pos][city] + tsp( mask | (1 << city), city);

ans = min(ans, newAns);

}

}

return dp[mask][pos] = ans;

}

int main() {

INPUT();

cin >> n;

VISITED\_ALL = (1 << n) - 1;

dist = vector< vector<int> > (n + 1, vector<int> (n + 1, INT\_MAX));

dp = vector< vector<int> > (1 << n, vector<int> (n, -1));

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

{

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

{

cin >> dist[i][j];

}

}

cout << "Min weight hamiltonian path costs " << tsp(1, 0) << endl;

return 0;

}

**Output :**

