**Practical No : 10**

**Name :** Samrudhi Pawar

**Roll No :** 44

**Title :**Develop a program for Travelling Salesman Problem using Branch and Bound.

**Program :**

// C++ program to solve Traveling Salesman Problem

// using Branch and Bound.

#include <bits/stdc++.h>

using namespace std;

const int N = 4;

// final\_path[] stores the final solution ie, the

// path of the salesman.

int final\_path[N+1];

// visited[] keeps track of the already visited nodes

// in a particular path

bool visited[N];

// Stores the final minimum weight of shortest tour.

int final\_res = INT\_MAX;

// Function to copy temporary solution to

// the final solution

void copyToFinal(int curr\_path[])

{

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

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

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

}

// Function to find the minimum edge cost

// having an end at the vertex i

int firstMin(int adj[N][N], int i)

{

int min = INT\_MAX;

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

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

min = adj[i][k];

return min;

}

// function to find the second minimum edge cost

// having an end at the vertex i

int secondMin(int adj[N][N], int i)

{

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

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

{

if (i == j)

continue;

if (adj[i][j] <= first)

{

second = first;

first = adj[i][j];

}

else if (adj[i][j] <= second &&

adj[i][j] != first)

second = adj[i][j];

}

return second;

}

// function that takes as arguments:

// curr\_bound -> lower bound of the root node

// curr\_weight-> stores the weight of the path so far

// level-> current level while moving in the search

// space tree

// curr\_path[] -> where the solution is being stored which

// would later be copied to final\_path[]

void TSPRec(int adj[N][N], int curr\_bound, int curr\_weight,

int level, int curr\_path[])

{

// base case is when we have reached level N which

// means we have covered all the nodes once

if (level==N)

{

// check if there is an edge from last vertex in

// path back to the first vertex

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

{

// curr\_res has the total weight of the

// solution we got

int curr\_res = curr\_weight +

adj[curr\_path[level-1]][curr\_path[0]];

// Update final result and final path if

// current result is better.

if (curr\_res < final\_res)

{

copyToFinal(curr\_path);

final\_res = curr\_res;

}

}

return;

}

// for any other level iterate for all vertices to

// build the search space tree recursively

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

{

// Consider next vertex if it is not same (diagonal

// entry in adjacency matrix and not visited

// already)

if (adj[curr\_path[level-1]][i] != 0 &&

visited[i] == false)

{

int temp = curr\_bound;

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

// different computation of curr\_bound for

// level 2 from the other levels

if (level==1)

curr\_bound -= ((firstMin(adj, curr\_path[level-1]) +

firstMin(adj, i))/2);

else

curr\_bound -= ((secondMin(adj, curr\_path[level-1]) +

firstMin(adj, i))/2);

// curr\_bound + curr\_weight is the actual lower bound

// for the node that we have arrived on

// If current lower bound < final\_res, we need to explore

// the node further

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

{

curr\_path[level] = i;

visited[i] = true;

// call TSPRec for the next level

TSPRec(adj, curr\_bound, curr\_weight, level+1,

curr\_path);

}

// Else we have to prune the node by resetting

// all changes to curr\_weight and curr\_bound

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

curr\_bound = temp;

// Also reset the visited array

memset(visited, false, sizeof(visited));

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

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

}

}

}

// This function sets up final\_path[]

void TSP(int adj[N][N])

{

int curr\_path[N+1];

// Calculate initial lower bound for the root node

// using the formula 1/2 \* (sum of first min +

// second min) for all edges.

// Also initialize the curr\_path and visited array

int curr\_bound = 0;

memset(curr\_path, -1, sizeof(curr\_path));

memset(visited, 0, sizeof(curr\_path));

// Compute initial bound

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

curr\_bound += (firstMin(adj, i) +

secondMin(adj, i));

// Rounding off the lower bound to an integer

curr\_bound = (curr\_bound&1)? curr\_bound/2 + 1 :

curr\_bound/2;

// We start at vertex 1 so the first vertex

// in curr\_path[] is 0

visited[0] = true;

curr\_path[0] = 0;

// Call to TSPRec for curr\_weight equal to

// 0 and level 1

TSPRec(adj, curr\_bound, 0, 1, curr\_path);

}

// Driver code

int main()

{

//Adjacency matrix for the given graph

int adj[N][N] = { {0, 10, 15, 20},

{10, 0, 35, 25},

{15, 35, 0, 30},

{20, 25, 30, 0}

};

TSP(adj);

printf("Minimum cost : %d\n", final\_res);

printf("Path Taken : ");

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

printf("%d ", final\_path[i]);

return 0;

}

Output:

Minimum cost : 80

Path Taken : 0 1 3 2 0