**NAME:** Aditya Jethani

**ROLL NO:** 21BCP166

**DIV-**3 **G-**5

**DAA Lab Assignment 9**

**AIM:** Use Branch and Bound to solve Travelling Salesman Problem (TSP)

**THEORY:** Branch and Bound is a widely used algorithmic technique that is often used to solve combinatorial optimization problems, such as the Traveling Salesman Problem (TSP). The basic idea behind the Branch and Bound approach is to systematically explore the solution space by dividing it into smaller subproblems, and then pruning those subproblems that are guaranteed to lead to suboptimal solutions. In the context of the TSP, the Branch and Bound algorithm works by dividing the problem into smaller subproblems, where each subproblem represents a partial tour of the cities. The algorithm explores each subproblem by considering all possible extensions to the tour, and prunes subproblems that cannot lead to better solutions than the current best solution. The Branch and Bound algorithm for the TSP is guaranteed to find the optimal solution, but its running time can be exponential in the number of cities, and thus is not practical for large instances of the problem.

**CODE:**

#include <bits/stdc++.h>

using namespace std;

const int SIZE = 5;

int final\_path[SIZE + 1];

bool visited[SIZE];

int final\_res = INT\_MAX;

void copyToFinalPath(int current\_path[])

{

    for (int i = 0; i < SIZE; i++)

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

    final\_path[SIZE] = current\_path[0];

}

int findFirstMinimum(int graph[SIZE][SIZE], int vertex)

{

    int minCost = INT\_MAX;

    for (int j = 0; j < SIZE; j++)

        if (graph[vertex][j] < minCost && vertex != j)

            minCost = graph[vertex][j];

    return minCost;

}

//

int findSecondMinimum(int graph[SIZE][SIZE], int vertex)

{

    int firstMin = INT\_MAX, secondMin = INT\_MAX;

    for (int j = 0; j < SIZE; j++)

    {

        if (vertex == j)

            continue;

        if (graph[vertex][j] <= firstMin)

        {

            secondMin = firstMin;

            firstMin = graph[vertex][j];

        }

        else if (graph[vertex][j] <= secondMin &&

                 graph[vertex][j] != firstMin)

            secondMin = graph[vertex][j];

    }

    return secondMin;

}

void TSPRecursive(int graph[SIZE][SIZE], int curr\_bound, int curr\_weight, int level, int curr\_path[])

{

    if (level == SIZE)

    {

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

        {

            int curr\_res = curr\_weight + graph[curr\_path[level - 1]][curr\_path[0]];

            if (curr\_res < final\_res)

            {

                copyToFinalPath(curr\_path);

                final\_res = curr\_res;

            }

        }

        return;

    }

    for (int i = 0; i < SIZE; i++)

    {

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

        {

            int temp = curr\_bound;

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

            if (level == 1)

                curr\_bound -= ((findFirstMinimum(graph, curr\_path[level - 1]) + findFirstMinimum(graph, i)) / 2);

            else

                curr\_bound -= ((findSecondMinimum(graph, curr\_path[level - 1]) + findFirstMinimum(graph, i)) / 2);

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

            {

                curr\_path[level] = i;

                visited[i] = true;

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

            }

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

            curr\_bound = temp;

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

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

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

        }

    }

}

void TSP(int graph[SIZE][SIZE])

{

    int curr\_path[SIZE + 1];

    int curr\_bound = 0;

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

    memset(visited, 0, sizeof(visited));

    for (int i = 0; i < SIZE; i++)

        curr\_bound += (findFirstMinimum(graph, i) + findSecondMinimum(graph, i));

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

    visited[0] = true;

    curr\_path[0] = 0;

    TSPRecursive(graph, curr\_bound, 0, 1, curr\_path);

}

int main()

{

    int graph[SIZE][SIZE] = {{0, 10, 15, 20, 25},

                             {10, 0, 35, 25, 30},

                             {15, 35, 0, 30, 40},

                             {20, 25, 30, 0, 30},

                             {25, 30, 40, 30, 0}};

    TSP(graph);

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

    cout << "Path Taken : ";

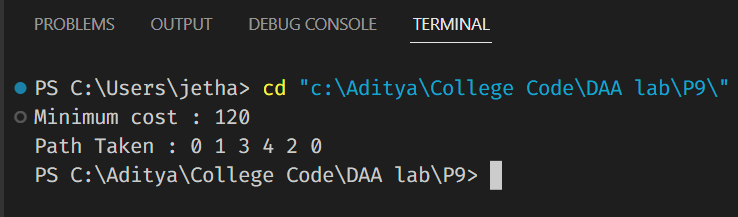
    for (int i = 0; i <= SIZE; i++)

        cout << final\_path[i] << " ";

    return 0;

}

**OUTPUT:**

****