**Aaditya Sharma 199302117**1.Traveling Salesman Problem Branch and Bound.

import math

maxsize = float('inf')

def fin(curr\_path):

    final\_path[:N + 1] = curr\_path[:]

    final\_path[N] = curr\_path[0]

def fMin(adj, i):

    min = maxsize

    for k in range(N):

        if adj[i][k] < min and i != k:

            min = adj[i][k]

    return min

def sMin(adj, i):

    first, second = maxsize, maxsize

    for j in range(N):

        if i == j:

            continue

        if adj[i][j] <= first:

            second = first

            first = adj[i][j]

        elif(adj[i][j] <= second and

            adj[i][j] != first):

            second = adj[i][j]

    return second

def TS(adj, curr\_bound, curr\_weight,

            level, curr\_path, visited):

    global final\_res

    if level == N:

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

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

                                        [curr\_path[0]]

            if curr\_res < final\_res:

                fin(curr\_path)

                final\_res = curr\_res

        return

    for i in range(N):

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

                            visited[i] == False):

            temp = curr\_bound

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

            if level == 1:

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

                                fMin(adj, i)) / 2)

            else:

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

                                fMin(adj, i)) / 2)

            if curr\_bound + curr\_weight < final\_res:

                curr\_path[level] = i

                visited[i] = True

                TS(adj, curr\_bound, curr\_weight,

                    level + 1, curr\_path, visited)

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

            curr\_bound = temp

            visited = [False] \* len(visited)

            for j in range(level):

                if curr\_path[j] != -1:

                    visited[curr\_path[j]] = True

def TP(adj):

    curr\_bound = 0

    curr\_path = [-1] \* (N + 1)

    visited = [False] \* N

    for i in range(N):

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

                    sMin(adj, i))

    curr\_bound = math.ceil(curr\_bound / 2)

    visited[0] = True

    curr\_path[0] = 0

    TS(adj, curr\_bound, 0, 1, curr\_path, visited)

adj = [[0, 5, 15, 10],

    [30, 0, 5, 5],

    [10, 30, 0, 40],

    [15, 20, 30, 0]]

N = 4

final\_path = [None] \* (N + 1)

visited = [False] \* N

final\_res = maxsize

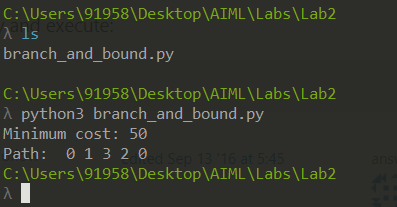
TP(adj)

print("Minimum cost:", final\_res)

print("Path: ", end = ' ')

for i in range(N + 1):

    print(final\_path[i], end = ' ')

OUTPUT  


2. Traveling Salesman Problem nearest neighbour.

#include<bits/stdc++.h>

using namespace std;

int cost = INT\_MAX;

int Matrix[5][5] = {

        {0, 15, 10, 5, 15},

        {15, 0, 30, 20, 15},

        {15, 30, 0, 30, 15},

        {5, 20, 10, 0, 50},

        {5, 20, 10, 10, 0}

};

void swap(int \*x, int \*y) {

    int flag;

    flag = \*x;

    \*x = \*y;

    \*y = flag;

}

void arrayFn(int \*arr, int flag){

    int sum = 0;

    for(int i = 0; i <= flag; i++) {

        sum += Matrix[arr[i % 4]][arr[(i + 1) % 4]];

    }

    if (cost > sum) {

        cost = sum;

    }

}

void TP(int \*arr, int src, int end) {

    if (src == end) {

        arrayFn(arr, end);

    }

    else {

        for (int j = src; j <= end; j++) {

            swap((arr + src), (arr + j));

            TP(arr, src + 1, end);

            swap((arr + src), (arr + j));

        }

    }

}

int main() {

   int arr[] = {0, 1, 2, 3, 4};

   TP(arr, 0, 4);

   cout<<"Minimum Cost: "<< cost;

   return 0;

}

OUTPUT

