****

**NAME: -** Darji Akshatkumar Hiteshbhai

**RollNo: -** 23MCD001

**Branch: -** M.tech-CSE**(Data Science)**

**Subject: -** Complexity Theory & Algorithms

**Practical-7**

**Aim:** Implement Assembly Line Scheduling problem using dynamic programming concepts.

**Code for Assembly Line Scheduling –**

#include <bits/stdc++.h>

using namespace std;

void printStations(vector<vector<int>>& l, int n, int l\_star) {

    int i = l\_star;

    cout << "Line " << i << ", Station " << n << endl;

    for (int j = n; j >= 2; j--) {

        i = l[i - 1][j - 1];

        cout << "Line " << i << ", Station " << j - 1 << endl;

    }

}

void assemblyLineScheduling(int n) {

    vector<vector<int>> a(2, vector<int>(n));

    vector<vector<int>> t(2, vector<int>(n));

    vector<int> e(2);

    vector<int> x(2);

    cout << "Enter processing times for Line 1: ";

    for (int j = 0; j < n; j++) {

        cin >> a[0][j];

    }

    cout << "Enter processing times for Line 2: ";

    for (int j = 0; j < n; j++) {

        cin >> a[1][j];

    }

    cout << "Enter transfer times from Line 1 to Line 2: ";

    for (int j = 0; j < n - 1; j++) {

        cin >> t[0][j];

    }

    cout << "Enter transfer times from Line 2 to Line 1: ";

    for (int j = 0; j < n - 1; j++) {

        cin >> t[1][j];

    }

    cout << "Enter entry times for Line 1 and Line 2: ";

    cin >> e[0] >> e[1];

    cout << "Enter exit times for Line 1 and Line 2: ";

    cin >> x[0] >> x[1];

    vector<vector<int>> f(2, vector<int>(n));

    vector<vector<int>> l(2, vector<int>(n));

    f[0][0] = e[0] + a[0][0];

    f[1][0] = e[1] + a[1][0];

    for (int j = 1; j < n; j++) {

        if (f[0][j - 1] + a[0][j] <= f[1][j - 1] + t[1][j - 1] + a[0][j]) {

            f[0][j] = f[0][j - 1] + a[0][j];

            l[0][j] = 1;

        } else {

            f[0][j] = f[1][j - 1] + t[1][j - 1] + a[0][j];

            l[0][j] = 2;

        }

        if (f[1][j - 1] + a[1][j] <= f[0][j - 1] + t[0][j - 1] + a[1][j]) {

            f[1][j] = f[1][j - 1] + a[1][j];

            l[1][j] = 2;

        } else {

            f[1][j] = f[0][j - 1] + t[0][j - 1] + a[1][j];

            l[1][j] = 1;

        }

    }

    int f\_star, l\_star;

    // f\* and l\*

    if (f[0][n - 1] + x[0] <= f[1][n - 1] + x[1]) {

        f\_star = f[0][n - 1] + x[0];

        l\_star = 1;

    } else {

        f\_star = f[1][n - 1] + x[1];

        l\_star = 2;

    }

    cout << "Optimal Cost: " << f\_star << endl;

    cout << "Optimal Path: Line " << l\_star << " -> Station " << n << endl;

    cout << "DP Table for Line 1:" << endl;

    for (int j = 0; j < n; j++) {

        cout << "Station " << j + 1 << ": " << f[0][j] << "[" << l[0][j] << "]" << " ";

    }

    cout << endl;

    cout << "DP Table for Line 2:" << endl;

    for (int j = 0; j < n; j++) {

        cout << "Station " << j + 1 << ": " << f[1][j] << "[" << l[1][j] << "]" << " ";

    }

    cout << endl;

    printStations(l, n, l\_star);

}

int main() {

    int n;

    cout << "Enter the number of stations: ";

    cin >> n;

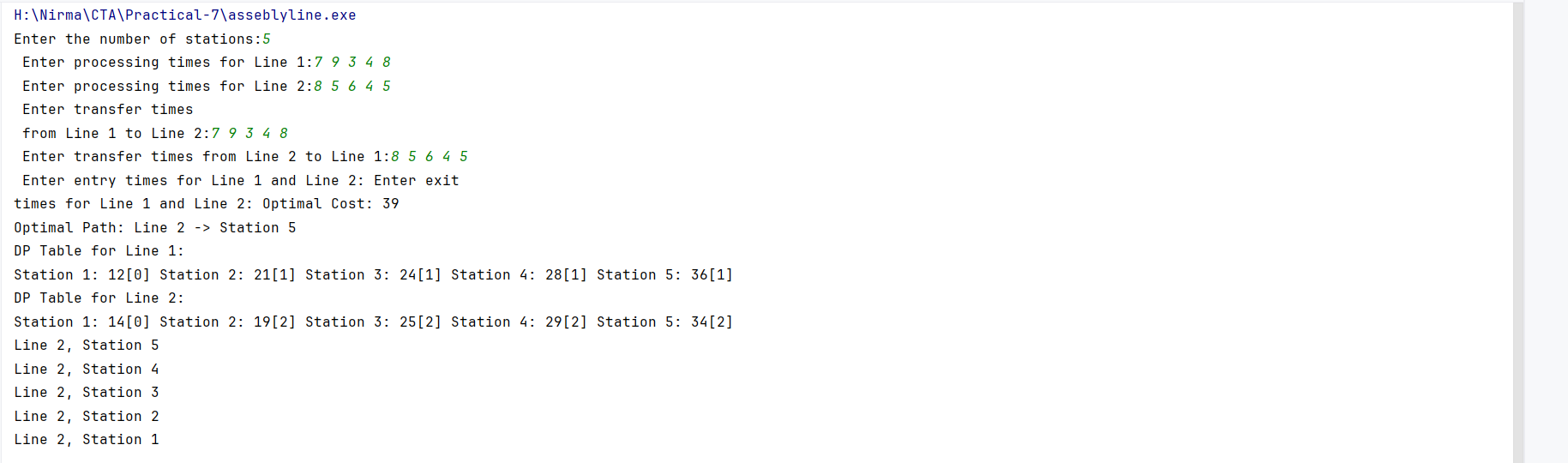
    assemblyLineScheduling(n);

    return 0;

}

**Output –**

**Test Case – 1**

****