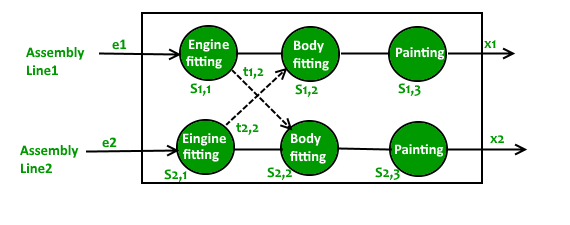
**Dynamic Programming | Set 34 (Assembly Line Scheduling)**

A car factory has two assembly lines, each with n stations. A station is denoted by Si,j where i is either 1 or 2 and indicates the assembly line the station is on, and j indicates the number of the station. The time taken per station is denoted by ai,j. Each station is dedicated to some sort of work like engine fitting, body fitting, painting and so on. So, a car chassis must pass through each of the n stations in order before exiting the factory. The parallel stations of the two assembly lines perform the same task. After it passes through station Si,j, it will continue to station Si,j+1 unless it decides to transfer to the other line. Continuing on the same line incurs no extra cost, but transferring from line i at station j – 1 to station j on the other line takes time ti,j. Each assembly line takes an entry time ei and exit time xi which may be different for the two lines. Give an algorithm for computing the minimum time it will take to build a car chassis.

The below figure presents the problem in a clear picture:



The following information can be extracted from the problem statement to make it simpler:

* Two assembly lines, 1 and 2, each with stations from 1 to n.
* A car chassis must pass through all stations from 1 to n in order(in any of the two assembly lines). i.e. it cannot jump from station i to station j if they are not at one move distance.
* The car chassis can move one station forward in the same line, or one station diagonally in the other line. It incurs an extra cost ti, j to move to station j from line i. No cost is incurred for movement in same line.
* The time taken in station j on line i is ai, j.
* Si, j represents a station j on line i.

**Breaking the problem into smaller sub-problems:**

We can easily find the ith factorial if (i-1)th factorial is known. Can we apply the similar funda here?

If the minimum time taken by the chassis to leave station Si, j-1 is known, the minimum time taken to leave station Si, j can be calculated quickly by combining ai, j and ti, j.

**T1(j)** indicates the minimum time taken by the car chassis to leave station j on assembly line 1.

**T2(j)** indicates the minimum time taken by the car chassis to leave station j on assembly line 2.

***Base cases:***

The entry time ei comes into picture only when the car chassis enters the car factory.

Time taken to leave first station in line 1 is given by:

T1(1) = Entry time in Line 1 + Time spent in station S1,1

T1(1) = e1 + a1,1

Similarly, time taken to leave first station in line 2 is given by:

T2(1) = e2 + a2,1

***Recursive Relations:***

If we look at the problem statement, it quickly boils down to the below observations:  
The car chassis at station S1,j can come either from station S1, j-1 or station S2, j-1.

Case #1: Its previous station is S1, j-1

The minimum time to leave station S1,j is given by:

T1(j) = Minimum time taken to leave station S1, j-1 + Time spent in station S1, j

.  
T1(j) = T1(j-1) + a1, j

Case #2: Its previous station is S2, j-1

The minimum time to leave station S1, j is given by:

T1(j) = Minimum time taken to leave station S2, j-1 + Extra cost incurred to change the assembly line + Time spent in station S1,

j  
T1(j) = T2(j-1) + t2, j + a1, j

The minimum time T1(j) is given by the minimum of the two obtained in cases #1 and #2.  
T1(j) = min((T1(j-1) + a1, j), (T2(j-1) + t2, j + a1, j))

Similarly the minimum time to reach station S2, j is given by:

T2(j) = min((T2(j-1) + a2, j), (T1(j-1) + t1, j + a2, j))

The total minimum time taken by the car chassis to come out of the factory is given by:  
Tmin = min(Time taken to leave station Si,n + Time taken to exit the car factory)  
Tmin = min(T1(n) + x1, T2(n) + x2)

**Why dynamic programming?**

The above recursion exhibits overlapping sub-problems. There are two ways to reach station S1, j:

1. From station S1, j-1
2. From station S2, j-1

So, to find the minimum time to leave station S1, j the minimum time to leave the previous two stations must be calculated(as explained in above recursion).

Similarly, there are two ways to reach station S2, j:

1. From station S2, j-1
2. From station S1, j-1

Please note that the minimum times to leave stations S1, j-1 and S2, j-1 have already been calculated.

So, we need two tables to store the partial results calculated for each station in an assembly line. The table will be filled in bottom-up fashion.

**Note:**  
In this post, the word “leave” has been used in place of “reach” to avoid the confusion. Since the car chassis must spend a fixed time in each station, the word leave suits better.

**Implementation:**

C:

// A C program to find minimum possible time by the car chassis to complete

#include <stdio.h>

#define NUM\_LINE 2

#define NUM\_STATION 4

// Utility function to find minimum of two numbers

**int** min(**int** a, **int** b) { **return** a < b ? a : b; }

**int** carAssembly(**int** a[][NUM\_STATION], **int** t[][NUM\_STATION], **int** \*e, **int** \*x)

{

**int** T1[NUM\_STATION], T2[NUM\_STATION], i;

    T1[0] = e[0] + a[0][0]; // time taken to leave first station in line 1

    T2[0] = e[1] + a[1][0]; // time taken to leave first station in line 2

    // Fill tables T1[] and T2[] using the above given recursive relations

**for** (i = 1; i < NUM\_STATION; ++i)

    {

        T1[i] = min(T1[i-1] + a[0][i], T2[i-1] + t[1][i] + a[0][i]);

        T2[i] = min(T2[i-1] + a[1][i], T1[i-1] + t[0][i] + a[1][i]);

    }

    // Consider exit times and retutn minimum

**return** min(T1[NUM\_STATION-1] + x[0], T2[NUM\_STATION-1] + x[1]);

}

**int** main()

{

**int** a[][NUM\_STATION] = {{4, 5, 3, 2},

                {2, 10, 1, 4}};

**int** t[][NUM\_STATION] = {{0, 7, 4, 5},

                {0, 9, 2, 8}};

**int** e[] = {10, 12}, x[] = {18, 7};

**printf**("%d", carAssembly(a, t, e, x));

**return** 0;

}

Java

// A java program to find minimum possible

// time by the car chassis to complete

import java.io.\*;

class GFG

{

    static int NUM\_LINE = 2;

    static int NUM\_STATION = 4;

    // Utility function to find minimum of two numbers

    static int min(int a, int b)

    {

        return a < b ? a : b;

    }

    static int carAssembly(int a[][], int t[][], int e[], int x[])

    {

        int T1[]= new int [NUM\_STATION];

        int T2[] =new int[NUM\_STATION] ;

        int i;

        // time taken to leave first station in line 1

        T1[0] = e[0] + a[0][0];

        // time taken to leave first station in line 2

        T2[0] = e[1] + a[1][0];

        // Fill tables T1[] and T2[] using

        // the above given recursive relations

        for (i = 1; i < NUM\_STATION; ++i)

        {

            T1[i] = min(T1[i - 1] + a[0][i],

                    T2[i - 1] + t[1][i] + a[0][i]);

            T2[i] = min(T2[i - 1] + a[1][i],

                    T1[i - 1] + t[0][i] + a[1][i]);

        }

        // Consider exit times and retutn minimum

        return min(T1[NUM\_STATION-1] + x[0],

                    T2[NUM\_STATION-1] + x[1]);

    }

    // Driver code

    public static void main (String[] args)

    {

        int a[][] = {{4, 5, 3, 2},

                    {2, 10, 1, 4}};

        int t[][] = {{0, 7, 4, 5},

                    {0, 9, 2, 8}};

        int e[] = {10, 12}, x[] = {18, 7};

        System.out.println(carAssembly(a, t, e, x));

    }

}

// This code is contributed by vt\_m

Python3

# Python program to find minimum possible

# time by the car chassis to complete

**def** carAssembly (a, t, e, x):

    NUM\_STATION **=** len(a[0])

    T1 **=** [0 **for** i **in** range(NUM\_STATION)]

    T2 **=** [0 **for** i **in** range(NUM\_STATION)]

    T1[0] **=** e[0] **+** a[0][0] # time taken to leave

                           # first station in line 1

    T2[0] **=** e[1] **+** a[1][0] # time taken to leave

                           # first station in line 2

    # Fill tables T1[] and T2[] using

    # above given recursive relations

**for** i **in** range(1, NUM\_STATION):

        T1[i] **=** min(T1[i**-**1] **+** a[0][i],

                    T2[i**-**1] **+** t[1][i] **+** a[0][i])

        T2[i] **=** min(T2[i**-**1] **+** a[1][i],

                    T1[i**-**1] **+** t[0][i] **+** a[1][i] )

    # consider exit times and return minimum

**return** min(T1[NUM\_STATION **-** 1] **+** x[0],

               T2[NUM\_STATION **-** 1] **+** x[1])

a **=** [[4, 5, 3, 2],

     [2, 10, 1, 4]]

t **=** [[0, 7, 4, 5],

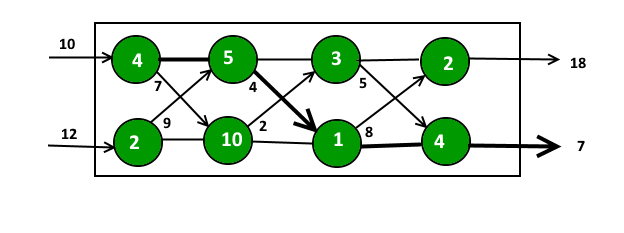
     [0, 9, 2, 8]]

e **=** [10, 12]

x **=** [18, 7]

print(carAssembly(a, t, e, x))

# This code is contributed by Soumen Ghosh



The bold line shows the path covered by the car chassis for given input values.

**Exercise:**  
Extend the above algorithm to print the path covered by the car chassis in the factory.

**References:**  
[Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest](http://www.flipkart.com/introduction-algorithms-3rd/p/itmczynzhyhxv2gs?pid=9788120340077&affid=sandeepgfg)

This article is compiled by [**Aashish Barnwal**](https://www.facebook.com/barnwal.aashish). Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above