BeeCrowd | 2769 | Assembly Line

Nama : Timothy Hosia Budianto

NRP : 50525211098

Permasalahan dan pendekatan :

beecrowd | 2769

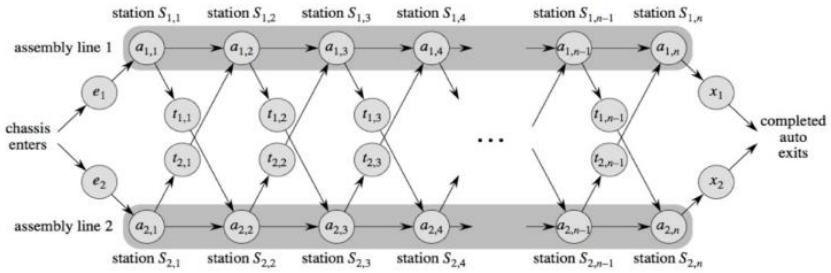
Assembly Line

By André Fillipi, INATEL Brazil

Timelimit: 1

With the spread of Industry 4.0's concepts and the development of Internet of Things, now it's simple to keep up with all the parts of a product's prodution in a assembly line. Having those informations, it is possible to optimize the production and reduce the time spent to have everything done.

An industry has the following production scheme:



Knowing the time spent in each station, and the time required to swich between the assembly lines, calculate the least time spent to produce a single item.

Soal ini mencari jalur mana, dalam sebuah jalur produksi yang tersusun dari mesin mesin, yang memerlukan waktu paling sedikit dengan ilustrasi gambar soal seperti diatas. Pada soal akan di berikan berapa jumlah mesin dari 1 line produksi (dari 2), waktu untuk masuk ke sebuah line produksi, waktu yang diperlukan pada tiap mesin, waktu yang diperlukan untuk pindah line produksi, dan terakhir waktu saat keluar dari produksi. Dalam memilih jalannya, kitab isa teteap dijalan yang sama atau pindah ke jalur produksi lain untuk mendapatkan waktu akhir yang tercepat.

Dengan Constraint tiap test case bisa antara 1 dan 1000 dengan value antara 0 dan 10^5.

Input Sample	Output Sample
3 1 1 1 2 3 3 2 1 1 2 2 1 1 1	7

Pada input :

line pertama (N) menunjukan jumlah mesin dalam 1 line produksi,
line kedua (e1,e2) menunjukan waktu yang di perlukan untuk masuk ke line 1 dan line 2,
line ke 3 (a11,a12) berisi waktu yang diperlukan tiap mesin pada line produksi 1 (sebanyak N)
line ke 4 (a21,a22) berisi waktu yang diperlukan tiap mesin pada line produksi 2 (sebanyak N)
line ke 5 (t11,t12) berisi waktu yang diperlukan untuk pindah ke line produksi 2 (sebanyak N-1)
line ke 6 (t21,t22) berisi waktu yang diperlukan untuk pindah ke line produksi 1 (sebanyak N-1)
yang terakhir (x1) berisi waktu untuk exit line 1, dan (x2) untuk exit line 2

Pada Output :
Berisi waktu tercepat yang bisa ditempuh dari awal hingga akhir line produksi

Abstraksi :

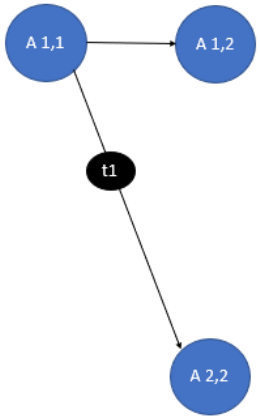
Pada soal ini jika dilakukan dengan cara brute force akan membutuhkan waktu yang luar biasa lama dan space memori yang sangat besar karena harus mencoba output setiap line dan disimpan dalam sebuah array, kemudian meng compare semuanya. Yang menyebabkan kompleksitas waktunya bisa hingga 2^N.

Oleh karena itu akan menggunakan beberapa metoode pendekatan baru yaitu **dynamic programming**, **bottom up**, dan **flying table.**

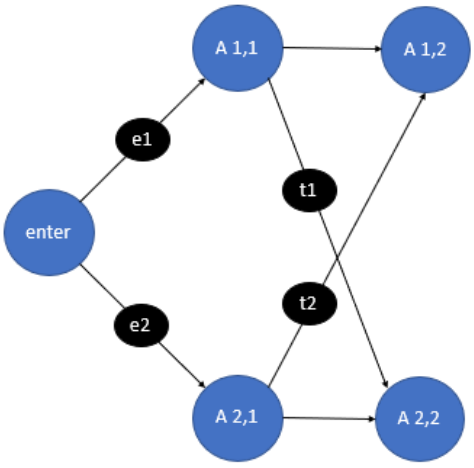
Dynamic Programming digunakan untuk mengoptimasi sebuah persoalan. Pertama kita haru mencari penyelesaian minimum / maximum kemudian di return ke value optimal **(bottom up)**. Langkah dalam dynamic programming adalah sebagai berikur :

1. Cari karakter dari struktur untuk mendapat solusi yang optimal (biasa matematis)
2. Secar rekursif cari value dari solusi optimal
3. Komputasi value dari solusi optimal secara bottom up
4. Kemudian konstruksikan solusi optimal menjadi computed information.

Di soal ini, dalam menentukan jalan mana yang ditempuh akan ada dilemma sebagai berikut ;



Pada gambar diatas Ketika ingin pindah ke mesin ke 2, kita dihadapkan pilihan bisa langsung direct ke a1,2 tanpa cost apa-apa atau bisa pindah ke line produksi lain yang memakan waktu t1. Oleh karena itu kita perlu mengcompare dengan 2 langkah sebelumnya untuk melihat waktu yang paling cepat sebagai berikut.



Kita akan melihat nilai minimum mana yang bisa diambil dengan menghitung nilai e dan bobot tiap mesin. Misal kita akan ke line 2 kita akan mencari minimum dari e1+A11+A12, e1+A11+t1+A22, e2+A21+A22, atau e2+A21+t2+A12. Untuk mencari rumus **kita akan menggunakan metode bottom up untuk mencari nilai minimal sebuah langkah**

Dari pengetahuan tersebut kita akan mencari solusi rekursif untuk problem ini. Misal f* = waktu tercepat untuk melalui seluruh mesin sampai exit - fi[j] = waktu tercepat untuk melewati suatu mesin i,j tertentu. Maka kita dapat rumuskan f* sebagai berikut

$$f^* = \min(f1[n] + x1, f2[n] + x2)$$

f* adalah minimum dari f1[n] ditambah exit 1 dan f2[n] ditambah exit 2. Sementara untuk

mencari fi[j] dapat kita rumuskan sebagai berikut.

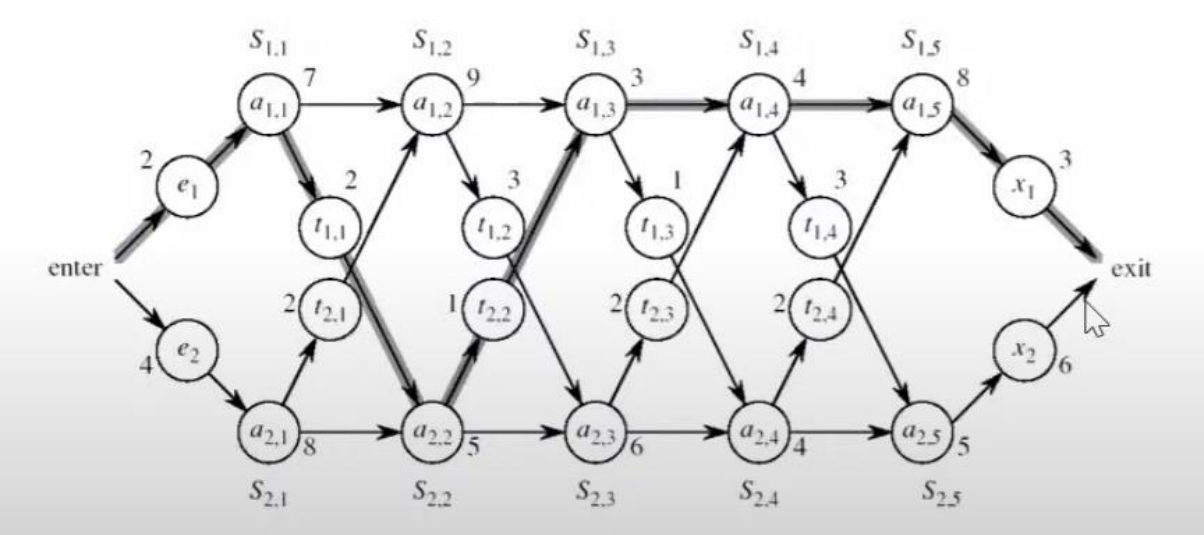
$$f1[j] = \min(f1[j-1] + a1,j,\ f2[j-1] + t2,j-1 + a1,j)$$

Untuk f1[j] = minimum dari f1[j-1] ditambah a1,j dengan f2[j-1] ditambah dengan waktu transfer t2,j-1 dan a1,j. Untuk f2[j] berlaku kebalikan dari f1[j]. Rumusan transisi fi[j] dapat dirangkum sebagai berikut

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1 \\ \min(f_1[j-1] + a_{1,j},\ f_2[j-1] + t_{2,j-1} + a_{1,j}) & \text{if } j \geq 2 \end{cases}$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1 \\ \min(f_2[j-1] + a_{2,j},\ f_1[j-1] + t_{1,j-1} + a_{2,j}) & \text{if } j \geq 2 \end{cases}$$

Misal pada ilustrasi soal sebagai berikut



Dengan rumus diatas dan ilustrasi testcase diatas akan menghasilkan table sebagai berikut

	1	2	3	4	5
f1[j]	2 + 7 = 9	9 + 9 = 18	16 + 1 + 3 = 20	20 + 4 = 24	24 + 8 = 32
f2[j]	4 + 8 = 12	9 + 2 + 5 = 16	16 + 6 = 22	20 + 1 + 4 = 25	25 + 5 = 30
	Sehingga f* = min(32 + 3, 30 + 6) = 35				

Setelah itu kita akan membuat pseudocode algoritma dalam approach soal ini sebagai berikut

1. f1[1] ← e1 + a1,1

2. f2[1] ← e2 + a2,1
3. **for** j ← 2 to n
4. **do if** f1[j-1] + a1,j ≤ f2[j-1] + t2,j-1 + a1,j
5. **then** f1[j] ← f1[j-1] + a1,j
6. **else** f1[j] ← f2[j-1] + t2,j-1 + a1,j
7. **if** f2[j-1] + a2,j ≤ f1[j-1] + t1,j-1 + a2,j
8. **then** f1[j] ← f1[j-1] + a1,j
9. **else** f2[j] ← f1[j-1] + t1,j-1 + a2,j

Line 4-6 menghitung value f1[j]

Line 7-9 menghitung value f2[j]

Implementasi Code

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;
int dp[2][2];
int n,e1,e2,x1,x2;
int a[2][1024],t[2][1024];

int GetInt(){
    int r=0;
    char c;
    while(1) {
        c = getchar_unlocked();
        if (c==' ' || c == '\n') continue;
        else break;
    }
    r = c-'0';
    while(1) {
        c = getchar_unlocked();
        if(c >= '0' && c <= '9') r=10*r+c-'0';
        else break;
    }
    return r;
}

int main()
{
    while(scanf("%d", &n) != EOF) {
        dp[0][0] = GetInt();
        dp[1][0] = GetInt();
        for(int j=0; j<2; j++)
            for (int i=0; i<n; i++) a[j][i] = GetInt();
        for (int j=0; j<2; j++)
            for (int i=0; i<n-1; i++) t[j][i] = GetInt();
        x1 = GetInt();
        x2 = GetInt();
        dp[0][0] += a[0][0];
        dp[1][0] += a[1][0];
        for (int i=1; i<n; i++) {
            dp[0][i%2] = min(dp[0][(i-1)%2],dp[1][(i-1)%2]+t[1][i-1])+a[0][i];
            dp[1][i%2] = min(dp[1][(i-1)%2],dp[0][(i-1)%2]+t[0][i-1])+a[1][i];
        }
        printf("%d\n", min(dp[0][(n-1)%2]+x1,dp[1][(n-1)%2]+x2));
    }
}
```

Line 1-6 adalah library dan variable yang akan digunakan

Line 8-23 aalah fungsi untuk mendapatkan interger dengan bypass handler

Pada main pertama scan jumlah mesin dalam variable n hingga EOF

Line 28-29 entry 1 dan 2 dimasukan ke variable matriks 2 dimensi yaitu dp

Selanjutnya processing time tiap mesin di simpan pada array 2d a

Lalu time untuk pindah line produksi akan ditaruh di varray 2d t

Line 34-35 waktu exit 1 dan exit 2 disimpan di variable x

Line 36-37 waktu entry 1 dan 2 akan langsung dimasukkan pada variable dp

Line 38-41 menggunakan for loop sampai N, akan mencari waktu terkecil (min) dari tiap Langkah yang bisa diambil.

```
dp[0][0] += a[0][0];
dp[1][0] += a[1][0];
for (int i=1; i<n; i++) {
    dp[0][i%2] = min(dp[0][(i-1)%2],dp[1][(i-1)%2]+t[1][i-1])+a[0][i];
    dp[1][i%2] = min(dp[1][(i-1)%2],dp[0][(i-1)%2]+t[0][i-1])+a[1][i];
}
```

Variable dp disini adalah implementasi **flying table** dimana dp akan menyimpan hanya 2 langkah sebelumnya untuk digunakan di iterasi berikutnya

Line 42 yang terakhir adalah meng output langkah minimun yang diambil setelah menambahkan waktu exit.

SUBMISSION # 30632524

PROBLEM:	2769 - Assembly Line
ANSWER:	**Accepted**
LANGUAGE:	C++17 (g++ 7.3.0, -std=c++17 -O2 -lm) [+0s]
RUNTIME:	0.000s
FILE SIZE:	968 Bytes
MEMORY:	-
SUBMISSION:	11/1/22, 8:11:18 AM