Task 1: BATCH

Hee Chul Kim, Jyrki Nummenmaa

Batch Scheduling

PROBLEM

There is a sequence of N jobs to be processed on one machine. The jobs are numbered from 1 to N, so that the sequence is 1, 2, …, N. The sequence of jobs must be partitioned into one or more batches, where each batch consists of consecutive jobs in the sequence. The processing starts at time 0. The batches are handled one by one starting from the first batch as follows. If a batch b contains jobs with smaller numbers than batch c, then batch b is handled before batch c. The jobs in a batch are processed successively on the machine. Immediately after all the jobs in a batch are processed, the machine outputs the results of all the jobs in that batch. The output time of a job j is the time when the batch containing j finishes.

A setup time S is needed to set up the machine for each batch. For each job i, we know its cost factor F_i and the time T_i required to process it. If a batch contains the jobs x, $x+1$, … , $x+k$, and starts at time t, then the output time of every job in that batch is $t + S + (T_x + T_{x+1} + \ldots + T_{x+k})$. Note that the machine outputs the results of all jobs in a batch at the same time. If the output time of job i is O_i, its cost is $O_i \times F_i$. For example, assume that there are 5 jobs, the setup time $S = 1$, $(T_1, T_2, T_3, T_4, T_5) = (1, 3, 4, 2, 1)$, and $(F_1, F_2, F_3, F_4, F_5) = (3, 2, 3, 3, 4)$. If the jobs are partitioned into three batches {1, 2}, {3}, {4, 5}, then the output times $(O_1, O_2, O_3, O_4, O_5) = (5, 5, 10, 14, 14)$ and the costs of the jobs are (15, 10, 30, 42, 56), respectively. The total cost for a partitioning is the sum of the costs of all jobs. The total cost for the example partitioning above is 153.

You are to write a program which, given the batch setup time and a sequence of jobs with their processing times and cost factors, computes the minimum possible total cost.

INPUT

Your program reads from standard input. The first line contains the number of jobs N, $1 \leq N \leq 10000$. The second line contains the batch setup time S which is an integer, $0 \leq S \leq 50$. The following N lines contain information about the jobs 1, 2, …, N in that order as follows. First on each of these lines is an integer T_i, $1 \leq T_i \leq 100$, the processing time of the job. Following that, there is an integer F_i, $1 \leq F_i \leq 100$, the cost factor of the job.

OUTPUT

Your program writes to standard output. The output contains one line, which contains one integer: the minimum possible total cost.

EXAMPLE INPUTS AND OUTPUTS

Example 1: input

```
2
50
100 100
100 100
```

output

```
45000
```

Example 2: input

```
5
1
1 3
3 2
4 3
2 3
1 4
```

output

```
153
```

Example 2 is the example in the text.

REMARK

For each test case, the total cost for any partitioning does not exceed $2^{31} - 1$.

SCORING

If your program outputs the correct answer for a test case within the time limit, then you get full points for the test case, and otherwise you get 0 points.

A. Solutions

This problem can be solved using dynamic programming. Let C_i be the minimum total cost of all partitionings of jobs $J_i, J_{i+1}, \dots, J_N$ into batches. Let $C_i(k)$ be the minimum total cost when the first batch is selected as $\{J_i, J_{i+1}, \dots, J_{k-1}\}$. That is, $C_i(k) = C_k + (S + T_i + T_{i+1} + \dots + T_{k-1}) * (F_i + F_{i+1} + \dots + F_N)$.

Then we have that

$C_i = \min \{ C_i(k) \mid k = i+1, \dots, N+1\}$ for $1 \le i \le N$,
and $C_{N+1} = 0$.

(a) $O(N^2)$ Time Algorithm

The time complexity of the above algorithm is $O(N^2)$.

(b) $O(N)$ Time Algorithm

Investigating the property of $C_i(k)$, P. Bucker[1] showed that this problem can be solved in $O(N)$ time as follows.

From $C_i(k) = C_k + (S + T_i + T_{i+1} + \dots + T_{k-1}) * (F_i + F_{i+1} + \dots + F_N)$, we have that for $i < k < l$,

$$C_i(k) \le C_i(l) \Leftrightarrow C_l - C_k + (T_k + T_{k+1} + \dots + T_{l-1}) * (F_i + F_{i+1} + \dots + F_N) \ge 0$$
$$\Leftrightarrow (C_k - C_l) / (T_k + T_{k+1} + \dots + T_{l-1}) \le (F_i + F_{i+1} + \dots + F_N)$$

Let $g(k,l) = (C_k - C_l) / (T_k + T_{k+1} + \dots + T_{l-1})$ and $f(i) = (F_i + F_{i+1} + \dots + O_N)$

Property 1: Assume that $g(k,l) \le f(i)$ for $1 \le i < k < l$. Then $C_i(k) \le C_i(l)$
Property 2: Assume $g(j,k) \le g(k,l)$ for some $1 \le j < k < l \le$ n. Then for each i with $1 \le i < j$, $C_i(j) \le C_i(k)$ or $C_i(l) \le C_i(k)$.

Property 2 implies that if $g(j,k) \le g(k,l)$ for $j < k < l$, C_k is not needed for computing F_i. Using this property, a linear time algorithm can be designed, which is given in the following.

Algorithm Batch

The algorithm calculates the values C_i for $i = N$ down to 1. It uses a queue-like list $Q = (i_r, i_{r-1}, \dots, i_2, i_1)$ with tail i_r and head i_1 satisfying the following properties:

$i_r < i_{r-1} < \dots < i_2 < i_1$ and
$g(i_r, i_{r-1}) > g(i_{r-1}, i_{r-2}) > \dots.. > g(i_2, i_1)$ -------- (1)

When C_i is calculated,

1. // Using $f(i)$, remove unnecessary element at head of Q.
   If $f(i) \ge g(i_2,i_1)$, delete i_1 from Q since for all $h \le i$, $f(h) \ge f(i) \ge g(i_2,i_1)$ and $C_h(i_2) \le C_h(i_1)$ by Property 1.
   Continue this procedure until for some $t \ge 1$, $g(i_r, i_{r-1}) > g(i_{r-1}, i_{r-2}) > \dots.. > g(i_{t+1}, i_t) > f(i)$.

   Then by Property 1, $C_i(i_{v+1}) > C_i(i_v)$ for $v = t, \dots, r$-1 or
   $r = t$ and Q contains only i_t.
   Therefore, $C_i(i_t)$ is equal to $\min\{C_i(k) \mid k = i+1, \dots, N+1\}$.

2. // When inserting i at the tail of Q, maintain Q for the condition (1) to be satisfied.
   If $g(i, i_r) \le g(i_r, i_{r-1})$, delete i_r from Q by Property 2.
   Continue this procedure until $g(i, i_v) > g(i_v, i_{v-1})$.
   Append i as a new tail of Q.

Analysis

Each i is inserted into Q and deleted from Q at most once. In each insertion and deletion, it takes a constant time. Therefore the time complexity is $O(N)$.

B. Test Data Information and Grading

Kee Moon Song

In total, 20 test cases are prepared and tested. Each test case is of 5 credits. Among them, 19 test cases are randomly generated so that negative integer by overflow does not occur during computing F_i. The remaining 1 test case is that setup time and all processing times and cost factors are 1.
The test case is mainly prepared to distinguish whether the competitors design an efficient algorithm or not. Among 20 test cases, algorithm by enumeration may solve for three ones within the given time limit, and an $O(N^2)$ time algorithm may solve for 14 test cases within the given time limit. If the competitors submit a correct $O(^{N})$ time algorithm, they will get 100 credits.

Timing Test for BATCH

No.	Optimal in C/C++	Optimal in Pascal	SubOpt. in C/C++	SubOpt. in Pascal
	O(N)	O(N)	$O(N^2)$	$O(N^2)$
1	< 0.01	< 0.01	< 0.01	< 0.01
2	< 0.01	< 0.01	< 0.01	< 0.01
3	< 0.01	< 0.01	< 0.01	< 0.01
4	< 0.01	< 0.01	< 0.01	< 0.01
5	< 0.01	< 0.01	< 0.01	< 0.01
6	< 0.01	< 0.01	< 0.01	< 0.01
7	< 0.01	< 0.01	< 0.01	< 0.01
8	< 0.01	< 0.01	< 0.01	< 0.01
9	< 0.01	< 0.01	< 0.01	< 0.01
10	< 0.01	< 0.01	< 0.01	< 0.01
11	< 0.01	< 0.01	< 0.01	< 0.01
12	< 0.01	< 0.01	< 0.01	0.02
13	< 0.01	< 0.01	< 0.01	0.04
14	< 0.01	< 0.01	0.03	0.06
15	< 0.01	< 0.01	0.23	0.87
16	< 0.01	< 0.01	0.3	1.19
17	< 0.01	< 0.01	0.33	1.3
18	< 0.01	< 0.01	0.39	1.37
19	< 0.01	< 0.01	0.41	1.52
20	< 0.01	< 0.01	0.45	1.81

Testing Data Description for BATCH

No.	Size(N)	Description	Solution
1	N = 5	Example 2	153
2	N = 10	Randomly generated data	170820
3	N = 15	Randomly generated data	322305
4	N = 20	Randomly generated data	596614
5	N = 30	Randomly generated data	1414590
6	N = 50	Randomly generated data	3900980
7	N = 100	Randomly generated data	12636575
8	N = 200	Randomly generated data	50649757
9	N = 300	Randomly generated data	124220878
10	N = 500	Each time & priority = 1	135794
11	N = 700	Randomly generated data	635041453
12	N = 1000	Setup time = 0	331524426
13	N = 1500	Randomly generated data	744367663
14	N = 2000	Randomly generated data	863732491
15	N = 7000	Randomly generated data	757615479
16	N = 8000	Randomly generated data	1003361707
17	N = 8500	Randomly generated data	915744544
18	N = 9000	Randomly generated data	1042629359
19	N = 9500	Randomly generated data	925702728
20	N = 10000	Randomly generated data	1025371921

C. Variations

There are other several variations of the batch problem [2].

(1) If there is no restriction on the scheduled sequence of jobs, that is, a batch consists of arbitrary set of jobs, then the problem is NP-hard.

(2) If the cost factor of all jobs are all 1 and there is no restriction on the scheduled sequence of jobs, then the problem can be solved in $O(N \log N)$ time.

(3) If all jobs have the same processing time and there is no restriction on the scheduled sequence of jobs, then the problem can be solved in $O(N \log N)$ time.

D. References

[1] P. Brucker, **Efficient algorithm for some path problems**, *Discrete Applied Mathematics* 62, pp. 77-85, 1995.
[2] S. Albers and P. Brucker, **The complexity of one-machine batching problems**, *Discrete Applied Mathematics* 47, pp. 87-107, 1993.

E. Source Code for BATCH

Kee Moon Song

```
/*
TASK : Batch
LANG : C

    Optimal solution - O(N) with Dynamic programming
*/


#include <stdio.h>

#define MAX 10000

int num, stime, answer;              // # of Jobs, Setup Time, answer
int data[MAX][2];                    // processing time &  priority
int queue[MAX + 1], table[MAX + 1]; // tables for DP

void getdata()
{
       int i;

       scanf ("%d %d", &num, &stime);

       for (i = 0; i < num; i++)
              scanf ("%d %d", &data[i][0], &data[i][1]);
}

void preprocessing()
{
       int i;

       for (i = 1; i < num; i++)
       {
              data[i][0] += data[i - 1][0];
              data[i][1] += data[i - 1][1];
       }
}

int func(int k)
{
       return data[num - 1][1] - (k ? data[k - 1][1] : 0);
}

int cost(int i, int j
{
       return func(i) * (stime + data[j - 1][0] - (i ? data[i - 1][0] : 0));
}

int delta(int i, int j)
{
       return (table[i] - table[j]) / (data[j - 1][0] - (i ? data[i - 1][0] :
0));
}

void solveproblem
{
       int head = 0, tail = 1;
       int i, j;

       queue[0] = num;
       table[num] = 0;

       for (j = num - 1; j >= 0; j
       {
              for (i = head; i < tail - 1; i++)
```

```
                    if (func(j) > delta(queue[i + 1], queue[i])) head++;
                    else break;

             table[j] = table[queue[head]] + cost(j, queue[head]);

             for (i = tail - 1; i > head; i--)
                    if (delta(j, queue[i]) <= delta(queue[i], queue[i - 1]))
                           tail--;
                    else break;

             queue[tail++] = j;
      }
      answer = table[0];
}

void outputs()
{
      printf ("%d\n", answer);
}

int main()
{
      getdata();
      preprocessing();
      solveproblem();
      outputs();

      return 0;
}
```