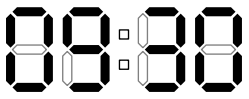
Problem A. Alarm Clock

Input file:	`alarm.in`
Output file:	`alarm.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Alice likes her digital alarm clock. She sets them up every evening. Last night Alice had a dream about her clock. Unfortunately, the only thing she is able to remember is the number of highlighted segments of the clock. Alice wonders what time was set on the clock in her dream.

Alice's clock have four digits: two for hours and two for minutes. For example, the clock below shows 9:30 (note the leading zero).



The clock uses following digit representation.

Input

The only line of the input file contains single integer n — the number of highlighted segments of the clock in Alice's dream ($0 \le n \le 30$).

Output

Output five characters in "`hh:mm`" format — the time shown on the clock in Alice's dream. The time must be correct: $0 \le$ `hh` < 24 and $0 \le$ `mm` < 60. If there are many possible correct times, output any of them. If there is none, output "`Impossible`".

Examples

alarm.in	alarm.out
23	09:30
28	Impossible
2	Impossible

Problem B. Buffcraft

Input file:	`buffcraft.in`
Output file:	`buffcraft.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Brenda enjoys a new role-playing game *Buffcraft*. Shields, swords, books and other carry-on items do not affects character stats in *Buffcraft*. The only way to increase the stats of your character is to buff her.

There are two types of buffs in *Buffcraft*. Direct buffs increase a base value of the stat, while percentage buffs increase stats by the fraction of the base value. To be precise, if unbuffed base value of your character stat is b, you have buffed her using n direct buffs of strength $d_1, d_2, ...d_n$ and m percentage buffs of strength $p_1, p_2, ..., p_m$, the resulting stat will be equal to $(b + d_1 + d_2 + \cdots + d_n)(100 + p_1 + p_2 + \cdots + p_m)/100$. Note that the resulting stat may be fractional.

Unfortunately, your character has only k buff slots and if you apply more than k buffs on her, only the last k buffs remains active. Thus, there is no reason to apply more than k buffs simultaneously. You cannot apply the same buff more than once.

Brenda is going to send his character to raid and wants to buff her health to maximal possible value. She has some direct and some percentage buffs at her disposal and needs your help to select the set of buffs that leads to maximal possible total health.

Input

The first line of the input file contains four integers b, k, c_d and c_p — the base health of the character, the number of buff slots, the number of available directs buffs, and the number of available percentage buffs.

The following line contains c_d integers d_i — strengths of direct buffs.

The last line of the input file contains c_p integer numbers p_i — strengths of percentage buffs.

All numbers in the input file are greater than or equal to zero, and less than or equal to fifty thousand.

Output

The first line of the output file must contain two integers n and m — the number of direct and percentage buffs to use ($0 \le n \le c_d$; $0 \le m \le c_p$; $0 \le n + m \le k$).

The following line must contain n different numbers — indices of direct buffs to apply (buffs are numbered from one).

The last line of the output must contain m different numbers — indices of percentage buffs to apply (also numbered from one).

The resulting total health after application of all $n + m$ buffs must be maximal possible.

Examples

buffcraft.in	buffcraft.out
70 3 2 2 40 30 50 40	2 1 2 1 1
1 2 3 4 6 6 5 8 10 7 9	2 0 1 2

Problem C. Combinator Expression

Input file:	`combinator.in`
Output file:	`combinator.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Combinatory logic may be thought as one of computational models allowing to express any computable function as a composition of functions from a small finite *basis*. In this problem we consider a restricted variant of BCKW basis, BCKI.

Combinator expression in BCKI basis is a string, corresponding to the following grammar:

$$\begin{aligned} \langle \text{Expression} \rangle &::= \langle \text{Expression} \rangle \langle \text{Term} \rangle \mid \langle \text{Term} \rangle \\ \langle \text{Term} \rangle &::= \text{`('} \langle \text{Expression} \rangle \text{`)'} \mid \text{`B'} \mid \text{`C'} \mid \text{`K'} \mid \text{`I'} \end{aligned}$$

As we can see from the grammar, the expression is a tree of applications where leafs are combinators B, C, K and I. The application is left-associative. For example BIC is equivalent to $(BI)C$, but not to $B(IC)$.

For the sake of the explanation we will use lowercase English letters ($a \dots z$) to represent sub-expressions. These lowercase letters will not appear in real data. For example, BIC can be represented by BxC (that is, $B\underset{x}{I}C$), x ($\underset{x}{BIC}$), xy ($\underset{x}{BI}\underset{y}{C}$), Bxy ($B\underset{x}{I}\underset{y}{C}$), but not by Bx.

We say that in expression pq we apply p to q. We can employ our intuition by saying that p is a function and q is its parameter. However, the evaluation process is quite different from traditional computation — instead of passing values over fixed expression tree, we evaluate by altering that tree so that the result is also some combinator expression.

To evaluate an expression, we need to select some sub-expression, corresponding to one of the patterns specified in the table below — that is, there should exist such x (and maybe y and z) that the pattern from the table becomes equal to the sub-expression. Then we need to replace the sub-expression with the reduction result from the table.

Pattern	Reduction result	Description
$Bxyz$	$x(yz)$	Composition function (Zusammensetzungsfunktion)
$Cxyz$	$(xz)y$	Exchange function (Vertauschungsfunktion)
Kxy	x	Constant function (Konstanzfunktion)
Ix	x	Identity function (Identitätsfunktion)

After the replacement took place we must repeat the process, until there remains no suitable sub-expressions. This final expression is *normal form* of the original one.

For example, in expression $CIC(CB)I$ we can make the following letter assignment

$$\underset{C}{C}\,\underset{x}{I}\,\underset{y}{C}\,\underset{z}{(CB)}\,I$$

and see that $CIC(CB)I \equiv (((CI)C)(CB))I \equiv (((Cx)y)z)I$ contains C combinator pattern and thus reduces to $((xz)y)I \equiv I(CB)CI$:

$$(\underset{C}{C}\,\underset{x}{I}\,\underset{y}{C}\,\underset{z}{(CB)})I \;\to\; (\underset{x}{I}\,\underset{z}{(CB)}\,\underset{y}{C})I$$

One more example: $B((CK)I)IC$ expression. Let us first reduce combinator B:

$$(\underset{B}{B}\,\underset{x}{((CK)I)})\,\underset{y}{I}\,\underset{z}{C} \;\to\; \underset{x}{((CK)I)}(\underset{y}{I}\,\underset{z}{C})$$

Now, let's reduce the last I:

$$((CK)I)(\underset{I}{I}\,\underset{x}{C}) \to ((CK)I)C$$

And now we finish evaluation with two more reductions:

$$((\underset{C}{C}\,\underset{x}{K})\,\underset{y}{I})\,\underset{z}{C} \to (\underset{K}{K}\,\underset{x}{C})\,\underset{y}{I} \to C$$

It is possible to show that the normal form remains the same irrespectable to the order of evaluation. For example, the following evaluation order:

$$C(K(II)(\underset{I}{I}\,\underset{x}{C})) \to C(K(\underset{I}{I}\,\underset{x}{I})(C)) \to C((\underset{K}{K}\,\underset{x}{I})\,\underset{y}{C}) \to CI$$

leads to the same result as

$$C(K(\underset{I}{I}\,\underset{x}{I})(IC)) \to C((\underset{K}{K}\,\underset{x}{I})\,\underset{y}{(IC)}) \to CI$$

However, as you see, the number of reductions is different: 3 in the first case and 2 in the second. This poses an interesting problem — to find an evaluation order with the minimal number of reductions for a given formula.

Your task is to write a program which finds the minimal number of reductions required for a given combinator expression to be evaluated to its normal form.

Input

The only line of the input file contains a combinator expression corresponding to the grammar above. The length of the expression does not exceed 30 000. The expression contains no whitespaces or symbols not specified in the grammar.

Output

Output a single integer — the minimal number of reductions required for the given formula to evaluate it to normal form.

Examples

combinator.in	combinator.out
C(K(II)(IC))	2
CIBI	3
BBBBBCCCCCKKKKKIIIII	15

Problem D. Digits

Input file:	digits.in
Output file:	digits.out
Time limit:	2 seconds
Memory limit:	256 megabytes

Little Petya likes integers. Recently he has learned about different properties of sums of number's digits. For example, if the sum of number's digits is divisible by 9, then the number itself is divisible by 9 as well.

Now little Petya is interested in numbers with equal sum of digits. He asks his older brother Dima to find n positive integers with equal sum of digits and minimal possible total sum. Dima has other important things to do, so he asked you to write a program that solves this problem for him.

Input

Input file contains a single integer n ($1 \le n \le 5000$).

Output

Output the minimal possible sum of n positive integers, that all have same sum of digits.

Examples

digits.in	digits.out
2	11
3	33

Problem E. Expression

Input file:	`expression.in`
Output file:	`expression.out`
Time limit:	10 seconds
Memory limit:	256 megabytes

In computing, regular expressions is a powerful tool for text search and string matching. In this problem a simplified version of regular expressions is used:

- An empty string "" is a regular expression, only the empty string matches it.
- A single lowercase letter "c" is a regular expression, a string consisting of a single letter c matches it.
- A dot "`.`" is a regular expression, a string consisting of any single letter matches it.
- Alternation: if α and β are regular expressions then "$(\alpha|\beta)$" is a regular expression, a string s matches it only if s matches α or s matches β.
- Concatenation: if α and β are regular expressions then "$(\alpha\beta)$" is a regular expression, a string s matches it only if $s = xy$, x matches α and y matches β.
- Kleene star: if α is regular expression then "$(\alpha*)$" is a regular expression, a string s matches it only if s is empty or $s = xy$, x is nonempty and matches α and y matches $(\alpha*)$. In other words, s consists of zero or more strings, each of them matches α.

Parentheses can be omitted, in this problem Kleene star has the highest priority, concatenation has medium priority and alternation has lowest priority. Thus "`abc*|de`" means "`(ab(c*))|(de)`".

For example, string "`abcabcab`" matches "`a(bc|a)*ab`", but string "`abcbab`" does not.

Your task is to find the shortest string that matches the given regular expression E and contains the given substring S.

Input

The first line of the input file contains the regular expression E. The second line of the input file contains the substring S ($1 \le |E|, |S| \le 10\,000$).

String S consists of lowercase English letters. Expression E consists of lowercase English letters and special characters: dots ('`.`'), parentheses ('`(`' and '`)`'), pipes ('`|`'), and asterisks ('`*`').

Output

Output the shortest possible string T that both matches E and contains S as substring. If there are no such strings, output "`NO`".

The string T should contain only lowercase English letters.

Examples

expression.in	expression.out
a.*b bab	abab
(ab)* bb	NO

Problem F. Fragmentation

Input file:	`fragmentation.in`
Output file:	`fragmentation.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Felix is working on a startup project in his garage. He has already found a great name for his project: SuperFastZilla. By now he is not sure what SuperFastZilla should do, but he is pretty sure it should do it fast, super fast.

Once he noticed that SuperFastZilla is working too slow, inspite of the fast algorithms used in it. Felix thinks that the problem may be caused by storage fragmentation.

The storage used by SuperFastZilla consists of n blocks of memory. SuperFastZilla performs some operations on this storage. Each block is used in one operation only, the i-th block is used in the a_i-th operation.

Felix wants to sort these blocks by the index of the operation they are used. To make it faster, Felix wants to split the storage into minimal number of segments of consecutive blocks, and then rearrange these segments to get the sorted array of blocks. After this rearrangement the order of block's indices of operations must be non-decreasing.

Help Felix to find the way to split the storage that minimizes the number of segments.

For example, if $a = [2, 3, 1, 1, 2, 2, 1]$, it can be split into three parts: $[2, 3]$, $[1, 1, 2, 2]$ and $[1]$. These parts can be rearranged to make the sorted array: $[1]$, $[1, 1, 2, 2]$, $[2, 3]$.

Input

The first line of input file contains an integer n ($1 \le n \le 10^5$). The next line contains n integers a_i ($1 \le a_i \le 10^5$).

Output

The first line of the output file must contain an integer number m — the minimal number of segments.

The next line must contains m integers, the lengths of the segments, from left to right.

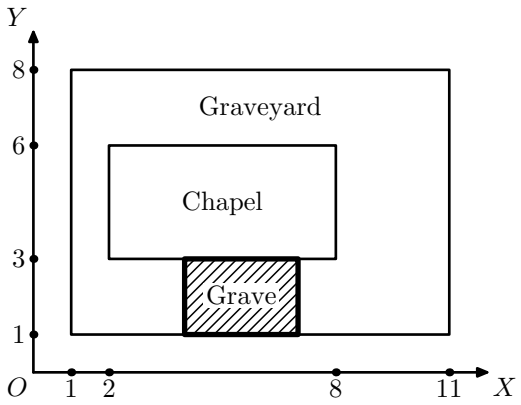
Example

fragmentation.in	fragmentation.out
7 2 3 1 1 2 2 1	3 2 4 1

Problem G. Grave

Input file:	`grave.in`
Output file:	`grave.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Gerard develops a Halloween computer game. The game is played on a rectangular graveyard with a rectangular chapel in it. During the game, the player places new rectangular graves on the graveyard. The grave should completely fit inside graveyard territory and should not overlap with the chapel. The grave may touch borders of the graveyard or the chapel.



Gerard asked you to write a program that determines whether it is possible to place a new grave of given size or there is no enough space for it.

Input

The first line of the input file contains two pairs of integers: x_1, y_1, x_2, y_2 ($-10^9 \le x_1 < x_2 \le 10^9$; $-10^9 \le y_1 < y_2 \le 10^9$) — coordinates of bottom left and top right corners of the graveyard. The second line also contains two pairs of integers x_3, y_3, x_4, y_4 ($x_1 < x_3 < x_4 < x_2$; $y_1 < y_3 < y_4 < y_2$) — coordinates of bottom left and top right corners of the chapel.

The third line contains two integers w, h — width and height of the new grave ($1 \le w, h \le 10^9$). Side with length w should be placed along OX axis, side with length h — along OY axis.

Output

The only line of the output file should contain single word: "`Yes`", if it is possible to place the new grave, or "`No`", if there is not enough space for it.

Examples

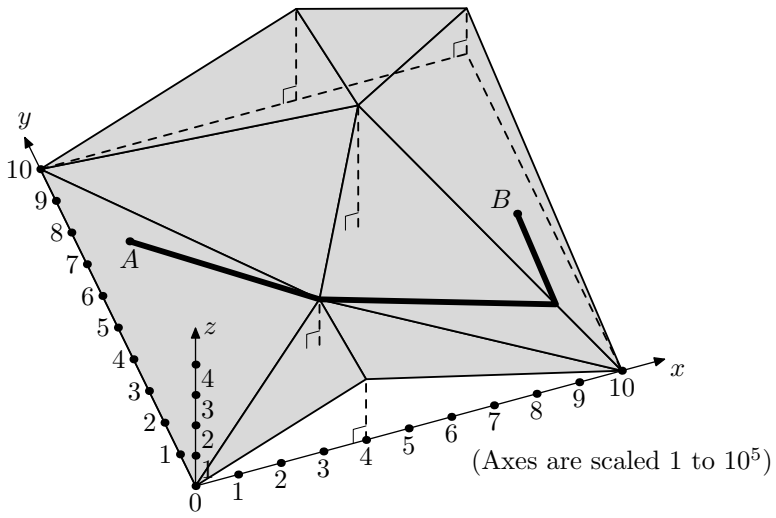
grave.in	grave.out
1 1 11 8 2 3 8 6 3 2	Yes
1 1 11 8 2 3 8 6 4 3	No

Problem H. Hiking in the Hills

Input file:	`hiking.in`
Output file:	`hiking.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Helen is hiking with her friends in a highland. Their plan is to hike from their camp A to a beautiful showplace B.

Unfortunately, Helen started feeling dizzy due to altitude sickness. Help her group find a route such that the topmost height on that route is as small as possible.



Input

The input file contains full information about the landscape of a square region $10^6 \times 10^6$ in the following format. The first line contains integer n — the number of triangles in the landscape ($2 \le n \le 2000$). Each of following n lines contains nine integers x_{i1}, y_{i1}, z_{i1}, x_{i2}, y_{i2}, z_{i2}, x_{i3}, y_{i3}, z_{i3} — coordinates of a triangle. All coordinates belong to the closed interval $[0, 10^6]$. The two last lines contain three integers each: x_A, y_A, z_A and x_B, y_B, z_B — coordinates of the camp A and the showplace B.

The given triangles are guaranteed to describe a consistent continuous landscape. Projections of triangles onto XY plane are non-degenerate and fill the square without overlapping. A vertex of one triangle never lays inside an edge of another triangle. Points A and B belong to the landscape surface and are different.

Output

Output a polyline route from A to B with the smallest possible topmost height. The first line should contain m, the number of vertices in this polyline. Each of following m lines should contain three integer coordinates of a polyline vertex: x_i, y_i, and z_i. Vertices must be listed along the polyline, from A to B (including these two endpoints).

All coordinates of polyline vertices should be integer. Each polyline edge must belong to some triangle from the input file (possibly, to its edge). The number of vertices in the polyline must not exceed $5n$.

Example

hiking.in

```
8
1000000 0 0 1000000 1000000 150000 600000 600000 400000
0 1000000 0 600000 600000 400000 600000 1000000 300000
0 1000000 0 400000 300000 150000 600000 600000 400000
400000 0 200000 1000000 0 0 400000 300000 150000
400000 300000 150000 1000000 0 0 600000 600000 400000
600000 600000 400000 1000000 1000000 150000 600000 1000000 300000
0 0 0 400000 0 200000 400000 300000 150000
0 1000000 0 0 0 0 400000 300000 150000
100000 700000 37500
900000 400000 137500
```

hiking.out

```
4
100000 700000 37500
400000 300000 150000
900000 150000 100000
900000 400000 137500
```

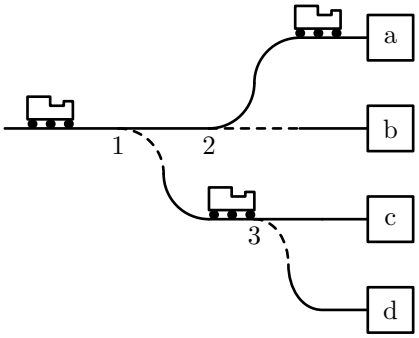
Problem I. Instruction

Input file:	`instruction.in`
Output file:	`instruction.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Ingrid is a head of a big railway station and, among other duties, is responsible for routing trains to the right platforms. The station has one entrance, and there are many switches that direct trains to other switches and platforms.

Each switch has one inbound track and two outbound tracks, platforms have one inbound track, and station entrance has one outbound track. Each outbound track is connected to one inbound track and vice versa. Every switch and platform is reachable from station entrance.

Platforms have a rail dead ends and you may assume that trains disappear from the platform immediately after arriving to it.



Each morning Ingrid looks at the timetable and writes switch toggling instruction: when and which switch to toggle. She would like to automate this process to save a lot of time.

Input

The first line of the input file contains a single integer n — the total number of switches and platforms on the station ($3 \le n \le 51$).

The i-th of the following n lines describes a switch or a platform with an index i. Description starts with a character '`p`' for a platform or '`s`' for a switch. Next number q_i indicates the number of the switch the inbound track is connected to or 0 if it is connected to station entrance ($0 \le q_i < i$). Description of the platform also contains a unique lowercase English letter — the platform identifier.

Trains spend exactly one minute to move between two connected switches or a switch and a platform. In the morning, each switch is toggled in a way that a train would pass to the one of the two outbound tracks connected to the switch/platform with the lower number.

Next line of the input file contains a single integer m ($1 \le m \le 1000$) — the number of trains in timetable.

Each of the following m lines contains integer a_i ($0 \le a_i \le 10\,000$; $a_i > a_{i-1}$) — the time in minutes when a train arrives to the station entrance, and the letter p_i — identifier of the destination platform for this train.

Output

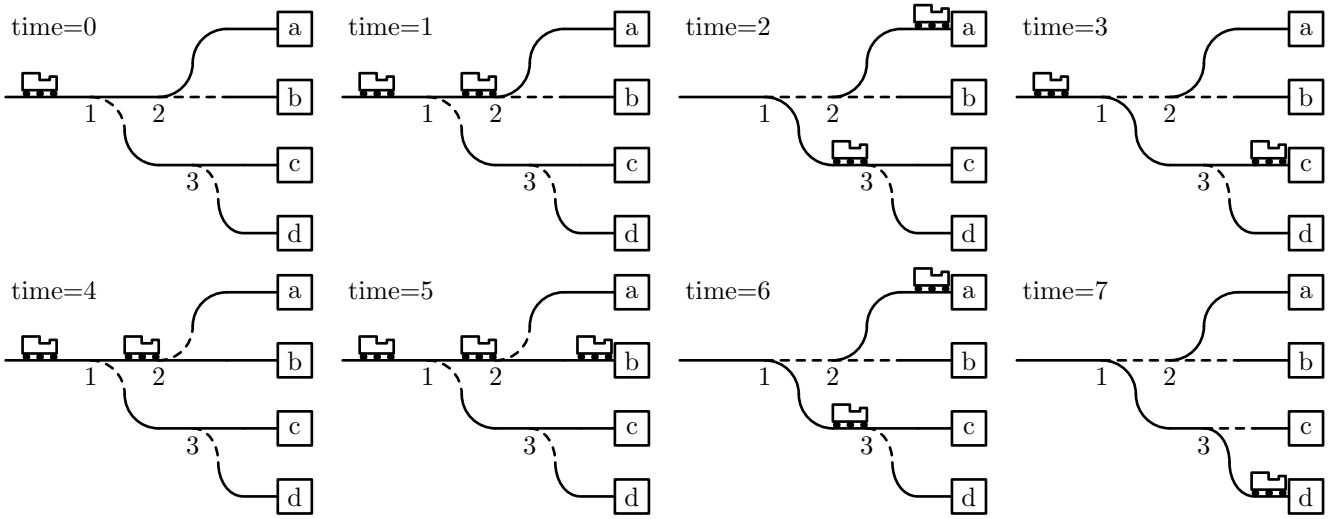
In the first line output integer c — the number of commands in the switch toggling instruction. For each command, output two integers s_i and t_i ($1 \le s_i \le n$; $0 \le t_i \le 10^9$) — the number of the switch and the time to toggle it. Assume that the switch is toggled between minutes $t_i - 1$ and t_i.

Output commands in order of non-decreasing time. The number of commands should not exceed 100 000.

Examples

instruction.in	instruction.out
7 s 0 s 1 s 1 p 2 a p 2 b p 3 c p 3 d 5 0 a 1 c 3 b 4 a 5 d	6 1 2 1 4 2 4 2 6 1 6 3 7
5 s 0 p 1 y s 1 p 3 z p 3 x 3 7 y 8 y 15 y	0
3 s 0 p 1 y p 1 z 3 7 y 8 y 10 y	5 1 1 1 2 1 2 1 3 1 200

Below is the time trace for the first example.

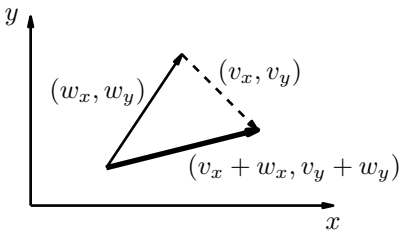


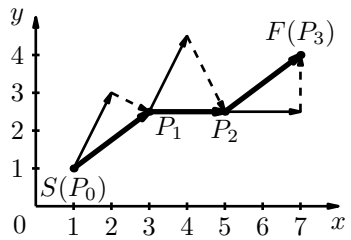
Problem J. Joy of Flight

Input file:	joy.in
Output file:	joy.out
Time limit:	2 seconds
Memory limit:	256 megabytes

Jacob likes to play with his radio-controlled aircraft. The weather today is pretty windy and Jacob has to plan flight carefully. He has a weather forecast — the speed and direction of the wind for every second of the planned flight.

The plane may have airspeed up to v_{max} units per second in any direction. The wind blows away plane in the following way: if airspeed speed of the plane is (v_x, v_y) and the wind speed is (w_x, w_y), the plane moves by $(v_x + w_x, v_w + w_y)$ each second.





Jacob has a fuel for exactly k seconds, and he wants to learn, whether the plane is able to fly from start to finish in this time. If it is possible he needs to know the flight plan: the position of the plane after every second of flight.

Input

The first line of the input file contains four integers S_x, S_y, F_x, F_y — coordinates of start and finish ($-10\,000 \le S_x, S_y, F_x, F_y \le 10\,000$).

The second line contains three integers n, k and v_{max} — the number of wind condition changes, duration of Jacob's flight in seconds and maximum aircraft speed ($1 \le n, k, v_{max} \le 10\,000$).

The following n lines contain the wind conditions description. The i-th of these lines contains integers t_i, w_{x_i} and w_{y_i} — starting at time t_i the wind will blow by vector (w_{x_i}, w_{y_i}) each second ($0 = t_1 < \cdots < t_i < t_{i+1} < \cdots < k$; $\sqrt{w_{x_i}^2 + w_{y_i}^2} \le v_{max}$).

Output

The first line must contain "Yes" if Jacob's plane is able to fly from start to finish in k seconds, and "No" otherwise.

If it can to do that, the following k lines must contain the flight plan. The i-th of these lines must contain two floating point numbers x and y — the coordinates of the position (P_i) of the plane after i-th second of the flight.

The plan is correct if for every $1 \le i \le k$ it is possible to fly in one second from P_{i-1} to some point Q_i, such that distance between Q_i and P_i doesn't exceed 10^{-5}, where $P_0 = S$. Moreover the distance between P_k and F should not exceed 10^{-5} as well.

Example

joy.in	joy.out
1 1 7 4 2 3 10 0 1 2 2 2 0	Yes 3 2.5 5 2.5 7 4

Problem K. Kebab House

Input file:	kebab.in
Output file:	kebab.out
Time limit:	2 seconds
Memory limit:	256 megabytes

The young man Vahtang Bumerang makes kebabs at the world-famous fast-food chain *Kebab House*. Each kebab contains many ingredients.

This morning Vahtang has received an order to make n kebabs. At first, he should put q_1 ingredients to the first kebab, then q_2 ingredients in the second one and so on. Vahtang spends one second to put one ingredient to a kebab, so it takes q_i seconds to make the i-th kebab. When he finishes the kebab he immediately proceeds to the next one.

Vahtang often dreams about his lovely boomerang while making kebabs. Each dream takes exactly one second and Vahtang forgets to put one ingredient to kebab during this second. Fortunately, he never dreams twice in any consecutive $(t + 1)$ seconds.

Due to dreams about boomerang, some kebabs may have lesser than the desired number of ingredients, but customers are still happy if the i-th kebab has at least x_i ingredients in it.

Vahtang wants to calculate the number of ways to have dream seconds during his work while keeping all customers happy. Can you help him? The real answer may be very huge, so you have to calculate it modulo $10^9 + 7$.

Input

The first line of the input file contains two integers n and t — the number of kebabs and minimal possible time between dream seconds ($1 \le n \le 1000$; $0 \le t \le 100$).

Each of the next n lines contains two integers q_i, x_i — the number of ingredients in the i-th kebab and the minimum number of ingredients to make the i-th customer happy ($1 \le q_i \le 250$; $0 \le x_i \le q_i$).

Output

The only line of the output file must contain one integer — the number of ways to distribute dream seconds modulo $10^9 + 7$.

Example

kebab.in	kebab.out
3 1 4 3 2 2 2 1	15