Problem A. Beatlemania

Input file:	`beatlemania.in`
Output file:	`beatlemania.out`
Time limit:	1 second
Memory limit:	64 Mebibytes

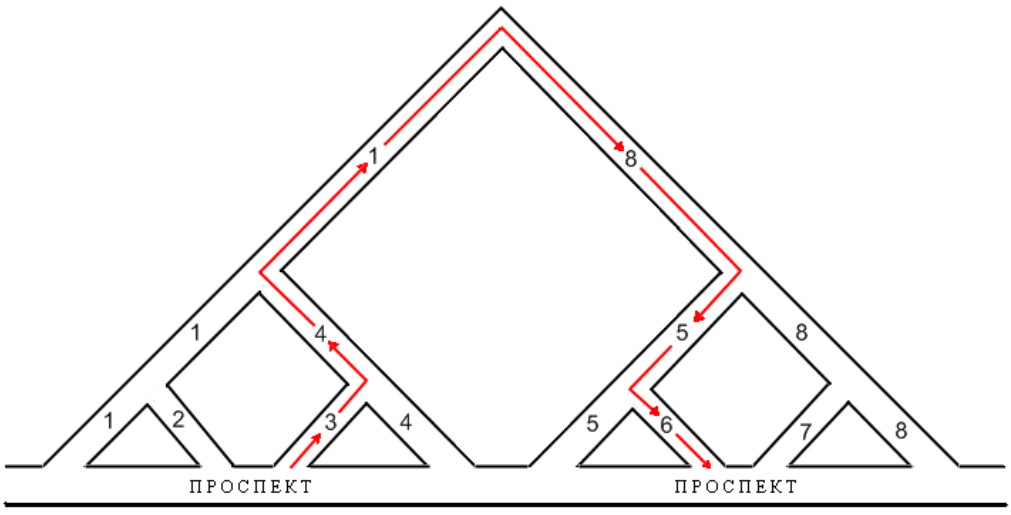
In 1960s the popularity of The Beatles was so high that it should be considered a mania: for months songs of the band headed hit parades, their concerts drew unimaginable (for that time) crowds, their albums had repeatedly hit sale records. However, the reverse of the coin was that the musicians could not travel across the city undisturbed...

On streets, in hotels and airports, the Fab Four was constantly chased by crowds of fans, especially by ladies admirers. In order to escape from a building surrounded by fans, the musicians would have to ask the police for assistance. But in an unfamiliar city, among a maze of streets, a telephone booth was the only (yet temporary) shelter from the fans — the best way to shake off pursuit is to call the manager of the band!

The Fab Four sheltered in a telephone booth located at one of the busy streets in the city, near a foot of a hill. There is no use in escaping along the road — too many people. A way to escape is to run into side-streets flowing from the street and going up the hill, which is divided into H levels. In total, there are 2^H side-streets, they are all enumerated in order, starting from 1.

In each of the levels, a half of side-streets "join perpendicularly" the other half according to the following rules. Consider a list A_i, which contains, in increasing order, the numbers of side-streets from the i-th level. All neighboring side-streets will be combined in pairs, starting from the 1st: $A_{i,1}, A_{i,2}$), ($A_{i,3}, A_{i,4}$) etc. In pairs with odd indexes, the second side-street "joins perpendicularly" the first one, and in pairs with even indexes, vice versa. At the last level two side-streets are joined without any further extension.

The figure depicts the layout of side-streets for a hill with three levels:



The telephone booth is near the beginning of the side-street with number S. In order to puzzle the crowd and not to let the musicians get lost in the mesh of side-streets, the manager of the band has proposed the following plan:

1. Ascent until you reach one of the levels.
2. Descent to the foot of the hill along the reflection of the climb trajectory.

Repeat these ascents and descents until the musicians reach the foot of the hill at the beginning of side-street T, where a car will wait for them. Determine the minimal number of hill climbs necessary to shake off the pursuit.

Input

Input file contains three space-separated integers: the number of levels H ($1 \leqslant H \leqslant 25$) and numbers of the initial and terminal side-streets S, T ($1 \leqslant S, T \leqslant 2^H$).

Output

The required number of hill climbs.

Example

beatlemania.in	beatlemania.out
3 3 6	1
2 1 3	2

Problem B. Second Chance

Input file:	`chance.in`
Output file:	`chance.out`
Time limit:	1 second
Memory limit:	64 Mebibytes

In the 60s, first multitasking operating systems, such as Multics, OS/360, UNIX etc. were introduced. Apart from multitasking, there was a new trend in memory management: swapping.

If the primary storage is full, in order to acquire space for a new page we should replace one of the previously stored pages. Replacement rules define which page will be replaced. Ideally, this should be the page which would be accessed the latest. In reality, however, operating system cannot predict future page access sequence, so heuristics are used to pick the page for replacement. One of such heuristics is the Second-chance page replacement algorithm described below.

From now on, the primary storage is considered to be organized as a connected list of no more than N elements. Each element is a memory page with corresponding flags.

- The reference flag for a page is set whenever a page is accessed for reading or writing.
- The modification flag for a page is set whenever a page is accessed for writing.

Page replacement rules are the following.

1. If the new page is already present in the list, nothing has to be changed.
2. If the list contains less than N elements and the new page is not present in the list, it is added to the tail of the list.
3. If the list contains exactly N elements and the new page is not present in the list, the list is traversed from the head until a page is found with reference flag not set. If the modification flag is set for that page, it is unloaded to auxiliary storage and removed from the list. Otherwise, the page is just removed from the list. If during the traversal, a page is found with reference flag set, the reference flag is cleared, and the page is moved to the tail of the list. After all these operations, the new page is added to the tail of the list.

For a given sequence of operations, simulate the second-chance algorithm to determine the order in which pages are loaded to primary storage and removed from there.

Input

The first line contains integers N and M ($1 \leqslant N \leqslant 1000$, $1 \leqslant M \leqslant 10^4$) separated by a space: the maximal number of elements in the list and the number of operations, respectively.

The next M lines describe operations. Each of these lines starts either with '`R`' for reading operation or '`W`' for writing operation, followed by a space and the integer number A_i ($1 \leqslant A_i \leqslant 10^6$) of the page which is being referenced.

Output

Output M blocks separated by blank lines, one block for each operation.

If during an operation, a page was unloaded to auxiliary storage, write "`UNLOAD`" and the page number on the first line of a block.

If a page was removed from the list, write "`DELETE`" and the page number on the first line of a block.

If a page was loaded to primary storage, write "`LOAD`" and the page number.

If during an operation, the list remained the same, write "`NONE`" in the corresponding block.

Example

chance.in	chance.out
3 6 R 1 W 2 R 3 W 7 W 3 W 9	LOAD 1 LOAD 2 LOAD 3 DELETE 1 LOAD 7 NONE UNLOAD 2 LOAD 9

Problem C. Conveyor

Input file:	`conveyor.in`
Output file:	`conveyor.out`
Time limit:	8 seconds
Memory limit:	64 Mebibytes

The remarkable economic development of Japan's economy in 1960s is known as the Japanese post-war economic miracle. One of the main factors contributed to such an abrupt growth rate was a shift in priorities to knowledge-intensive industries and mastering of new technologies. Japanese manufacturers, which are mainly export-oriented, were one of the first to use automated product quality control at all stages of manufacture.

A conveyor consists of N blocks of M cells in each. A cell may be empty or filled with one item of product. Initially, all the cells are empty and the conveyor is idle. There are two operations on the conveyor.

- **A load operation**: X copies of some sequence of cells are continuously loaded at the beginning of the conveyor, and the content of the conveyor moves forward. The filled cells passing the outer border of the last block are considered containing end products, which are placed in a warehouse.
- **A rollback of the conveyor**: the content of the conveyor moves backward by Y cells. This operation empties the content of the filled cells that pass the outer border of the first block, and the product contained in these cells is considered as discarded.

After any operation of load or rollback, the conveyor stops, and the products in the cells that did not pass the whole sequence of blocks remain unchanged until the next operation begins. The most important property of the conveyor is the stage-by-stage production control performed with the aid of special devices located at the joint of each pair of adjacent blocks. When the conveyor starts moving (irrespective of the direction), then any filled cell which first pass through the device is emptied, and the content of this cell is inspected. All empty cells, as well as filled cells passing through the device, remain unchanged after that.

Your goal is to calculate the number of end products, discarded products, as well as the number of products send for inspection, and number of those left on the conveyor after completion of all operations.

Input

The first row contains two space-separated integers N ($1 \leqslant N \leqslant 5000$), M ($1 \leqslant M \leqslant 60$) and K ($1 \leqslant K \leqslant 5000$); these are the number of blocks, number of cells in a block, and number of operations.

The next K lines specify the operations.

If The first symbol on a line is '`>`', it is followed by an integer X_i ($1 \leqslant X_i \leqslant 10^6$), a space and a row S_i of length not exceeding M which consists only of symbols '`0`' and '`1`'. Such a line specifies a loading operation of X_i copies of the sequence of cells S_i ('`0`' means that a cell is empty, '`1`' means that a cell is filled).

If The first symbol on a line is '`<`', it is followed by an integer Y_i ($1 \leqslant Y_i \leqslant 10^6$). Such a line specifies the rollback of the conveyor by Y_i cells.

Output

Four space-separated integers: number of end products, discarded products, number of products sent for inspection, and number of those left on the conveyor.

Example

conveyor.in	conveyor.out
4 5 5 > 2 111 > 5 001 < 3 > 4 00100 > 1 00	1 1 10 3

Замечание

11111 00000 00000 00000 — after 1-st operation (1 for inspection);

00100 10010 01001 11000 — after 2-nd operation (3 for inspection);

00000 10000 01010 00000 — after 3-rd operation (1 to discard, 3 for inspection);

00100 00100 00100 00000 — after 4-th operation (1 ready, 3 for inspection);

00001 00001 00001 00000 — after 5-th operation (no changes).

Problem D. The Dots Game

Input file:	`dots.in`
Output file:	`dots.out`
Time limit:	2 seconds
Memory limit:	64 Mebibytes

No one knows when the Dots Game was adopted in schools and universities of the USSR. Probably it happened in the 60s as well. This pen-and-paper variation of game "Go" quickly gained popularity among pioneers and Komsomol members. It also possessed a significant advantage over other games: the players made no noise and were looking just like thoughtful students making remarks in their copybooks.

The Dots Game goes on a rectangular grid of $N \times M$ nodes. A node is an intersection of grid lines.

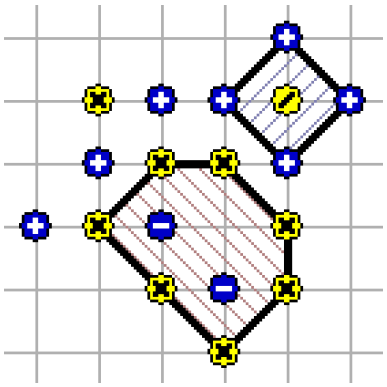
Two players move in turns. During a move, a player must place a dot of his colour in one of the free nodes.

A **free node** is a node without a dot in it which does not belong to any surrounded area.

A **surrounded area** of a player is an area surrounded by a closed polyline containing at least one dot of another player. Each segment of the polyline should be drawn between two horizontally, vertically or diagonally adjacent active dots of the player which the area belongs to. A surrounded area is always created in such a way that the number of active dots of its owner inside and on the border is minimal possible.

An **active dot** is a dot which does not belong to any surrounded area of the opposite player.

A **passive dot** is a dot which belongs to a surrounded area of the opposite player.



You should write a program that will, given the moves of the game, output the position after all these moves. If a move is not permitted (the node is not free), you should ignore it altogether without changing who moves next. After each move, your program should construct all possible new surrounded areas for the current player. Inside these areas, all dots of the opposite player become passive, and all dots of the current player become active whether or not they were surrounded previously.

Input

The first line contains the dimensions N and M of the grid ($1 \leqslant N, M \leqslant 50$) and the number of moves K ($0 \leqslant K \leqslant 2500$).

Next K lines contain integers R_i and C_i ($1 \leqslant R_i \leqslant N$, $1 \leqslant C_i \leqslant M$) which are the row and column where the current player tries to put his dot.

Output

Output the position after all the given moves as a field of size $(N + 2) \times (M + 2)$ where each symbol corresponds to one node. Symbol '`#`' corresponds to the border of the field. Symbol '`+`' corresponds to an active dot of the first player. Symbol '`-`' corresponds to a passive dot of the first player. Symbol '`*`'

corresponds to an active dot of the second player. Symbol '/' corresponds to a passive dot of the second player. Finally, a space corresponds to a node without a dot.

Examples

dots.in	dots.out
6 6 20 2 3 2 2 2 4 3 3 3 5 2 5 2 6 3 4 1 5 4 2 4 1 4 5 3 2 6 4 5 4 5 5 4 3 5 3 4 4 1 5	######## # + # # *++/+# # +**+ # #+*- * # # *-* # # * # ########
4 5 11 3 3 4 5 2 2 1 2 1 3 3 2 2 4 2 1 3 1 2 3 4 2	####### # *+ # #*-*+ # #+*+ # # + *# #######
5 5 17 3 3 3 2 4 2 4 3 4 4 3 4 3 5 2 3 5 3 5 4 2 4 1 4 1 3 1 2 2 2 5 2 3 1	####### # *+* # # +/+ # #+/+/+# # +/+ # # *+* # #######

Problem E. HAL 9000

Input file:	hal9000.in
Output file:	hal9000.out
Time limit:	1 second
Memory limit:	64 Mebibytes

"2001: A Space Odyssey", a 1968 epic science fiction film produced and directed by Stanley Kubrick, is rightfully considered as one of the best films in cinematography. This film is famous not only for its revolutionary special effects which allowed a super-realistic picture of the outer space, but also by its forecasts regarding the nearest future of the human race.

According to the story, the ship's computer HAL 9000 is an intellectual supercomputer capable of self-education. The computer fully controls the mission, talks to people using natural language, expresses feelings, and can surely beat any human in chess. To give the crew any chance for success at all, the rules of the game were changed significantly.

The game will be carried out on a board of $N \times M$ squares. Each player is given only one figure: a rook, whose coordinates are chosen randomly before the beginning of a game. The players alternate moves, moving his/her figure horizontally or vertically; a player is not allowed to put his figure on the horizontal or vertical line at which the opponent's figure is placed, as well as to cross these two lines. If a player cannot move then he/she loses.

Find the game outcome assuming that a man moves first and both play their best (i. e., they try to win in a minimum number of moves if they can, or to lose in a maximum number of moves if the loss is inevitable).

Input

Six space-separated integers: board dimensions N, M ($1 \leqslant N, M \leqslant 100$), coordinates of a human player rook X_1, Y_1 and computer player rook X_2, Y_2 ($1 \leqslant X_1, X_2 \leqslant N$, $1 \leqslant Y_1, Y_2 \leqslant M$, $X_1 \neq X_2$, $Y_1 \neq Y_2$).

Output

Print "MAN" in the first line if the game is won by the human player; print "HAL" otherwise. Second line should contain the number of moves made by both players at the completion of the game.

Examples

hal9000.in	hal9000.out
2 2 1 1 2 2	HAL 0
3 9 1 8 3 9	MAN 1

Problem F. Hippiemobile

Input file:	hippiemobile.in
Output file:	hippiemobile.out
Time limit:	1 second
Memory limit:	64 Mebibytes

Volkswagen's T1 was the name of one of the first civil vans. Not did it only revolutionize the field of automotive industry, but it also became a legendary "hippiemobile", a symbol of late 1960s... Travels were very popular among hippies, and such a van proved the best solution for this purpose, because it was capacious, inexpensive and could serve as a house for people during stops. Hippies used to color those vans in bright colors, cover them with slogans and even changed the original Volkswagen's insignia with the peace sign. In short, it was impossible to miss such a "piece of art" on a road!

In one of the countries, a pilgrimage place for hippies, there are N settlements connected by M roads, along which one can travel only in one direction. There is an old legend among the hippies that some roads are mysterious: if one drives through such a road, one can never return back to its start settlement. Find the number of such roads in the country.

Input

The first row contains two space-separated integers N ($1 \leqslant N \leqslant 10^5$) and M ($1 \leqslant M \leqslant 10^6$). These are the numbers of settlements and roads, respectively.

Then follow M rows containing integers U_i and V_i ($1 \leqslant U_i, V_i \leqslant N$, $U_i \neq V_i$) separated by blanks. These are the numbers of settlements that are connected by the road with number i. Any move along a road is allowed only in the direction from U_i to V_i.

Multiple roads between any pair of settlements are allowed.

Output

Number of mysterious roads in the country.

Example

hippiemobile.in	hippiemobile.out
5 7 1 2 1 3 1 5 3 2 3 4 4 1 5 4	2

Problem G. Sea of Holes

Input file:	`holes.in`
Output file:	`holes.out`
Time limit:	1 second
Memory limit:	64 Mebibytes

"Once upon a time or maybe twice, there was an unearthly paradise called Pepperland. 80,000 leagues beneath the sea it lay, or lie, I'm not too sure," — thus begins "Yellow Submarine", a 1968 animated feature film based on the music of The Beatles.

The peace of Pepperland is threatened by an attack of the Blue Meanies, who hate all music and beauty. In the last minute before his own capture, Pepperland's elderly Lord Mayor sends Old Fred, a sailor, off in the Yellow Submarine to get help. Old Fred travels to Liverpool, where he persuades The Beatles to return to Pepperland with him and overthrow the reign of the Blue Meanies with their music. The five journey back to Pepperland in the yellow submarine. On their way to Pepperland, they pass through several seas: Sea of Time, Sea of Science, Sea of Monsters and Sea of Holes.

Sea of Holes is in fact a rectangular grid of size $N \times M$ containing holes in its cells. Only K of these holes lead to Pepperland, other ones do not lead to anything, so if one arrives on them, nothing happens. In order not to get lost in the Sea of Holes, a visitor should move in the following way:

1. Move as long as possible in the current horizontal direction (changing the column).
2. Move one cell in the current vertical direction (changing the row).
3. Change the current horizontal direction to the opposite.
4. Go to step 1.

If on step 2 there is no cell in the current vertical direction, a visitor should instead stand still, change both the current horizontal direction and the current vertical direction to the opposite ones and then go to step 1.

For each cell of the Sea of Holes, find out the time in minutes a visitor will need to get to Pepperland starting from that cell. Moving between two adjacent cells takes exactly one minute. Getting to Pepperland from a cell which contains a hole leading there is instantaneous. A visitor always picks moving right as the initial horizontal direction and moving down as the initial vertical direction.

Input

The first line contains three integers separated by spaces: N and M, the dimensions of the field ($1 \leqslant N, M \leqslant 300$), and K, the number of holes which lead to Pepperland ($1 \leqslant K \leqslant N \times M$). Each of the next K lines contains two integers R_i and C_i separated by a space ($1 \leqslant R_i \leqslant N$, $1 \leqslant C_i \leqslant M$) which are the row and column numbers of the cells which contain holes leading to Pepperland. The rows are numbered from top to bottom, the columns are numbered from left to right.

Output

Output should consist of N lines. Each line should contain M integers separated by spaces. The j-th number on the i-th line, A_{ij}, should be the time in minutes a visitor will need to get to Pepperland starting from cell (i, j).

Example

holes.in	holes.out
4 5 3 1 3 2 1 3 5	2 1 0 6 5 0 4 3 2 1 4 3 2 1 0 13 12 11 10 9
1 5 1 1 3	2 1 0 3 2

Problem H. Monopolies

Input file:	monopolies.in
Output file:	monopolies.out
Time limit:	2 seconds
Memory limit:	64 Mebibytes

In USA and Western Europe, the 1960s went to the music of control of monopolies. This problem proved to be particularly pronounced in civil aviation — in a number of regions air corporations gained full control over air space, tickets were overpriced, no adequate level of service was secured.

The Antimonopoly Committee issued a decree that an air company for which your work must delegate a series of rights to perform air flight to other air companies. The company management's concern is that this decree would not change the company's status in the region, and set the task to count up the number of pairs of cities between which the air space is considered by the company as insignificant.

An air company operates flights between N cities whose coordinates are known. The air space between any pair of cities can be considered as a line segment, with ends corresponding to the cities that it connects. An air space is called strategically insignificant if, between any pair of cities not lying in this air space, there is at least one route (possibly, through other cities that has no common points with the first air space).

Input

First row of input file contains one integer N ($1 \leqslant N \leqslant 10^5$), the number of cities.

The next N rows contain coordinates of the cities — two space-separated integers X_i, Y_i, not exceeding 10^9 in absolute value. ($(X_i - X_j)^2 + (Y_i - Y_j)^2 > 0$ for any pair $i \neq j$).

Output

Number of pairs of cities with strategically insignificant air space between them.

Example

monopolies.in	monopolies.out
4 -1 0 0 0 0 1 1 2	5

Замечание

The pairs of cities (1, 2), (1, 3), (1, 4), (2, 4) and (3, 4) are strategically insignificant.

Problem I. Strawberry Fields Forever

Input file:	`strawberry.in`
Output file:	`strawberry.out`
Time limit:	1 second
Memory limit:	64 Mebibytes

I always wondered what's the story behind the name of the popular song of The Beatles, "Strawberry Fields Forever". There's a story about some orphanage in Liverpool with the same name where John Lennon used to go... But we all know that the official biography tends to prettify the facts.

So what it *could* mean? Well, maybe it's about picking strawberries with younger brother. The fields seem infinite, the process eternal, and the younger brother doesn't help at all: he just eats all strawberries he could reach... Wait, John didn't have a brother! Still, this fantasy was enough to lead to an interesting problem.

The strawberry field consists of an infinite number of similar rectangular glades, each of size $N \times M$ strawberry bushes, which touch each other and form a rectangular grid. Each bush contains a certain number of strawberries A_{ij}. The following picture shows a part of the field with glades of size 2×2.

A_{11}	A_{12}	A_{11}	A_{12}	A_{11}	A_{12}
A_{21}	A_{22}	A_{21}	**A_{22}**	A_{21}	A_{22}
A_{11}	A_{12}	**A_{11}**	**A_{12}**	**A_{11}**	A_{12}
A_{21}	**A_{22}**	**A_{21}**	**$\underline{A_{22}}$**	**A_{21}**	**A_{22}**
A_{11}	A_{12}	**$\underline{A_{11}}$**	**$\underline{A_{12}}$**	**$\underline{A_{11}}$**	A_{12}
A_{21}	A_{22}	A_{21}	**$\underline{A_{22}}$**	A_{21}	A_{22}

The elder brother first chooses a bush as a starting point and then picks all strawberries from all bushes no further than K units from it. The distance between two bushes is the sum of absolute differences of their coordinates. One possible area to be gathered by the elder brother for $K = 2$ is marked in bold on the picture above.

The younger brother also chooses a bush as a starting point and then eats all strawberries from all bushes no further than L units from it. The starting bush is chosen in such a way that it is as far as possible from the elder brother's starting point, yet the area to be gathered by the elder brother fully contains the area eaten by the younger brother. One possible area to be eaten by the younger brother for $L = 1$ is underlined on the picture above.

Write a program that will find the maximal possible number of berries the elder brother can pick. The younger brother always eats all berries within his reach before the elder brother gets a chance to collect them.

Input

The first line contains integers N and M separated by a space ($1 \leqslant N, M \leqslant 100$) which are the dimensions of a glade. Next N lines contain M space-separated integers each; j-th number on i-th of these lines is A_{ij}, the number of strawberries on the respective bush ($0 \leqslant A_{ij} \leqslant 100$).

The last line of input contains integers K and L separated by a space ($1 \leqslant K \leqslant 10^6$, $1 \leqslant L < K$) which are the radius of elder brother's reach and the radius of younger brother's reach, respectively.

Output

Output the maximal number of strawberries elder brother can collect.

Example

strawberry.in	strawberry.out
2 2 2 1 2 4 2 1	21

Problem J. Recording Studio

Input file:	`studio.in`
Output file:	`studio.out`
Time limit:	2 seconds
Memory limit:	64 Mebibytes

Nowadays many professional musicians prefer to record their music at home using the appropriate software, and use studios only for mixing. Back in the 60s, things were different: the recording equipment was bulky and expensive, so even renting it was not at all cheap. The musicians had to rehearse their compositions beforehand and work on schedule. It is hard to believe that to record their first album which sold more than 10 mln. copies, the legendary Led Zeppelin spent only 36 hours in the studio...

The high skill of musicians is not enough to achieve such an efficiency. Everything going on in the studio should be well coordinated. Even cleaning of the rooms should be done in time. There are N rooms in the studio. For each room, we know the time T_i required to fully clean it per day, as well as the schedule of visitors using the room. There is one cleaning master in the studio. You should make a schedule for him so that all rooms are fully cleaned and the total time spent by visitors in the rooms which are being cleaned is minimal possible.

There is also a list of restrictions which contains the time intervals for each room when cleaning is either forbidden or should last no longer than a certain amount of time in minutes. Cleaning master could split the cleaning process for each room into several intervals during the day, but the start and finish times of each cleaning interval in minutes should be integers. At every moment of time, cleaning master can be cleaning no more than one room. You can neglect the time needed to move between rooms.

Input

The first line contains the number of rooms N ($1 \leqslant N \leqslant 30$). The second line contains the times when cleaning master comes to the studio and leaves it in format `HH:MM`. The two times are separated by a space. His workday lasts between 1 and 500 minutes.

Then follow N blocks describing the rooms. Each block is preceded by an empty line. The first line of a block contains T_i ($0 \leqslant T_i \leqslant 500$) which is the time required per day to clean room i.

The second line of a block contains K_i ($0 \leqslant K_i \leqslant 100$) which is the number of records in the schedule of visits to the room.

Then follow K_i lines each of which contains the start and finish times of a visit to the room in format `HH:MM`. The two times are separated by a space. The finish time is greater than the start time by at least one minute.

Next line contains L_i ($0 \leqslant L_i \leqslant 10$) which is the number of records in the list of restrictions.

Next follow L_i lines each of which contains the start and finish times of a time interval in format `HH:MM` and W_{ij} ($0 \leqslant W_{ij} \leqslant 500$), the number of minutes allowed for cleaning during that interval. On each of these lines, tokens are separated by spaces. Each interval is at least one minute in length. No two intervals in one block intersect. All numbers in the input are integers.

The start of the day is designated by `00:00`, the end of the day is `24:00`.

Output

If it is impossible to clean all rooms according to the rules, output "`NO`". Otherwise, on the first line, output the minimal total time across all visitors when they were in rooms which were being cleaned. After that, output N blocks with the schedule of cleaning each room in the same order as in the input. Each block should be preceded by an empty line. The first line of a block should contain P_i, the number of time intervals for cleaning room i. Next P_i lines should contain the start and finish times of each of these intervals in format `HH:MM` separated by a space. These lines should be sorted by the start time. If the

finish time of an interval coincides with the start time of another interval, they should be merged into one interval. If there are several optimal solutions, output any of them.

Examples

studio.in	studio.out
2 16:50 19:00 80 3 08:00 18:50 17:00 17:30 17:50 18:09 1 17:00 18:00 10 7 2 12:00 17:35 17:45 24:00 1 17:45 24:00 0	79 4 16:50 17:00 17:30 17:35 17:45 17:50 18:00 19:00 1 17:35 17:42
1 00:00 02:01 120 2 00:00 24:00 01:00 02:00 0	179 2 00:00 01:59 02:00 02:01
1 08:00 12:00 300 0 0	NO

Problem K. Tournament

Input file:	`tournament.in`
Output file:	`tournament.out`
Time limit:	1 second
Memory limit:	64 Mebibytes

In 1960s, when soccer broadcastings have just started to conquer their own audience, the broadcasting companies already dictated their terms to soccer leagues...

In most of the tournaments, the geographic location of teams is not taken into account in making up the calendar. This results in a series of problems in large countries, where a transfer between cities may take significant time — after long flights players are not in their best conditions, soccer fans have to spend much of their time and money to see games live, while TV companies decline to broadcast games live because of time differences.

As an experiment, a decision was made to test a scheme with account for the geographical factor on a small cup system. Assume that N teams participate in the tournament. Each team is assigned a unique value L_i, which is the rating of the team determined from previous results.

First, all the teams are split in pairs. For each such partition, one can define the uniformity coefficient W, which is the sum of absolute values of differences of ratings of the teams for all pairs. A partition is called optimal if its uniformity coefficient is maximal possible.

To take the geographical factor into account, it is proposed to supplement the list of rules with the rule that each pair in a partition must be composed of teams from one region. In the initial list of teams, the teams from the first region are given even numbers, and the teams from the second region are given odd numbers. Such a partition is called geographically optimal.

Since there may be many geographically optimal partitions, the number of a partition was introduced. If we write a partition as an array of N numbers, in which each position contains the rating of the corresponding opponent in the pair, and then sort the partitions lexicographically (first sort by increasing order of numbers in the 1st position, then in the 2nd position, etc.), then the number of a partition is, by definition, its rating in the sorted list of geographically optimal partitions.

For a given number K, it is required to find the K-th geographically optimal partition.

Input

The first line of input contains an integer number of teams N ($4 \leqslant N \leqslant 48$), which is always divisible by 4. The second line contains N space-separated positive integers L_i, each not exceeding 10^9; these are the ratings of the teams. The third line contains integer K ($1 \leqslant K \leqslant 10^{18}$); this is the number of the sought-for partition.

Output

If there is no geogrpaphically optimal partition (i. e., in any optimal partition there exists a pair containing odd and even team), print "`NO`".

If the number of a sought-for partition exceeds the total number of optimal partitions, print "`OVER`"; otherwise print "`YES`" in the first line and then print the partition itself in the second line, separating numbers by spaces.

Examples

tournament.in	tournament.out
8 21 72 16 83 51 32 64 45 2	YES 5 8 7 6 1 4 3 2
8 22 71 82 14 55 38 69 40 3	NO
4 11 22 33 44 2	OVER

Problem L. Woodstock

Input file:	`woodstock.in`
Output file:	`woodstock.out`
Time limit:	1 second
Memory limit:	64 Mebibytes

Woodstock Music and Art Fair is one of the most famous music festival ever. It took place at a dairy farm near the hamlet of White Lake in the town of Bethel, New York, from August 15 to August 18, 1969. By the time, it was considered to be the largest group of people ever assembled in one place to "have three days of fun and music and have nothing but fun and music", as it was said by Max Yasgur, the owner of the farm where the festival was held.

The organizers faced many problems in preparation of the festival. For example, to avoid crowding and uncontrolled motion of people's masses, the tickets were sold to so-called perimeters formed around the scene. During a day, one was permitted to move only inside his or her perimeter. Perimeters consist of sectors and are numbered by the distance to the scene, which is considered the perimeter number zero. Perimeter number K consists of $4K+1$ sectors around perimeter number $K-1$, as shown on the picture.

3	2	1	0	1	2	3
3	2	1	1	1	2	3
3	2	2	2	2	2	3
3	3	3	3	3	3	3

The sound volume and quality vary from sector to sector and are expressed by an integer A_{ij}. Write a program that will find the perimeter where the sum of A_{ij} is maximal possible.

Input

The first line of input contains N ($1 \leqslant N \leqslant 100$), the maximal size of a perimeter.

Next $N+1$ lines contain $2N+1$ integers each; i-th of these lines contains the values A_{ij} for the corresponding sectors ($|A_{ij}| \leqslant 10^5$). The scene is located at the first of these lines in position $N+1$.

Output

Output two numbers: the number of perimeter with maximal sum of A_{ij} and the sum itself. If there are multiple perimeters with the same maximal sum of A_{ij} in them, output the one with the lowest number.

Examples

woodstock.in	woodstock.out
2 1 2 3 2 -2 2 3 5 4 -1 3 5 5 3 0	1 16
3 -1 1 1 10 1 1 -1 -2 1 1 12 1 1 -2 -3 1 1 15 1 1 -3 -7 10 20 30 20 10 -6	3 65