Problem A. Ambitious Plan

Input file:	`ambitious.in`
Output file:	`ambitious.out`
Time limit:	4 seconds
Memory limit:	512 megabytes

Steven Sleepberg is going to produce the forty seventh sequel to his film "Battlefield Jupiter" and is now planning the main battle scene. The script of the scene says that n empire drones would attack m republic forts. The attack proceeds as follows: a drone shoots the laser cannon towards one of the forts. The defense in turn has t power towers, and one pair of towers creates a power shield. The power shield is a segment connecting the two towers. If the shield intersects the segment which connects the shooting drone with its target a spectacular explosion takes place.

Steven wants the scene to be as spectacular as possible. So he wants to film all possible pairs of shoot and shield that cause explosion. Help him to estimate the length of the scene by counting the number of possible explosions.

Let us consider drones, forts and power towers as points on a plane. The line of defense coincides with the line $y = 0$, so y-coordinates of all forts and towers are less than 0, and y-coordinates of all drones are greater than 0. You are given coordinates of n drones, m forts and t towers. Find the number of sets $\{D, F, T_1, T_2\}$ such that D is drone, F is fort, T_1 and T_2 are towers, and segments DF and T_1T_2 intersect. No two points coincide, no three points are on the same line.

Input

The input file contains multiple test cases.

Each test case starts with n — the number of drones ($1 \le n \le 1500$), n lines follow, each line contains dx_i, dy_i — coordinates of a drone. Integer m — the number of forts ($1 \le m \le 1500$) — follows, with m more lines, each line contains fx_i, fy_i — coordinates of a fort. Finally there goes t — the number of towers ($2 \le t \le 1500$) followed by t lines, each line contains tx_i, ty_i — coordinates of a tower.

All drones coordinates satisfy $-10^9 \le dx_i \le 10^9$, $0 < dy_i \le 10^9$. All forts and towers coordinates satisfy $-10^9 \le fx_i, tx_i \le 10^9$, $-10^9 \le fy_i, ty_i < 0$.

Input is followed by a line containing a single zero.

The total number of drones in the input file is at most 1500. The total number of forts in the input file is at most 1500. The total number of towers in the input file is at most 1500.

Output

For each test case print one integer: the number of possible pairs of shoot-shield that cause an explosion.

Example

ambitious.in	ambitious.out

3
1 12
10 30
30 10
1
10 -10
4
2 -11
9 -1
11 -1
15 -14
0
	7

Problem B. Borderless Words

Input file:	`borderless.in`
Output file:	`borderless.out`
Time limit:	4 seconds
Memory limit:	512 megabytes

A word w is called *bordered* if there exists a word u, other then w and empty word ε, such that u is both suffix and prefix if w. For example, a word «abbababb» is bordered because «abb» is both its prefix and its suffix. A word that is not bordered is called *borderless*. For example, a word «aabab» is borderless.

Consider all borderless words of length n composed of letters «a» and «b». Let us denote the number of such words as C_n. Order them lexicographically — by the first letter, then by the second one, etc, and number from 1 to C_n. Given k find the k-th word in this order.

Input

The input file contains multiple test cases.

Each test case contains two integers n and k on a line ($1 \le n \le 64$, $1 \le k \le C_n$).

Input is followed by a line with $n = k = 0$. There are at most 1000 test cases in one input file.

Output

For each test case output one line — the k-th lexicographically borderless word of length n.

Examples

`borderless.in`	`borderless.out`
`5 1`	`aaaab`
`5 2`	`aaabb`
`5 3`	`aabab`
`5 4`	`aabbb`
`5 5`	`ababb`
`0 0`	

Problem C. Catalan Combinatorial Objects

Input file:	`catalan.in`
Output file:	`catalan.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

Andrew likes Catalan numbers. Also Andrew likes to joke.

He is an experienced problem setter and prepares lots of contests for training camps. Each contest he prepares a problem that has one integer as input, one integer as output, and answers for 0, 1, 2, 3, 4, and 5 are, respectively, 1, 1, 2, 5, 14 and 42. However, answers for greater inputs don't coincide with corresponding Catalan numbers.

Andrew has already prepared so many contests, that he is short of good problems with such property. So he decided to automate the process of creating such problems. As a good pool of possible problems he considers problems of counting combinatorial objects of a specific nature. Andrew has chosen k — the desired answer for the input 6, and wants to find a description of a combinatorial object that has 1, 1, 2, 5, 14, 42, k objects with weight 0, 1, 2, 3, 4, 5, 6, respectively.

Andrew uses the following ways to construct combinatorial objects.

The base set B consists of one unit object u of weight 1. Each constructed object x has some weight $w(x)$. If the object is constructed of one or more other objects, its weight is equal to the sum of their weights.

Let X be the set of some combinatorial objects. Consider the following ways to construct new sets.

The set $L(X)$ contains all possible lists of finite length, each element of which belongs to X and has positive weight. For example, $L(B)$ contains $[]$, $[u]$, $[u, u]$, $[u, u, u]$, etc. Similarly, $L(L(B))$ contains $[]$, $[[u]]$, $[[u], [u]]$, $[[u, u], [u]]$, $[[u], [u, u]]$, etc. Note, that the last two lists are different: the order of elements in the list matters. Also note that $[[]]$ is not a valid list in $L(L(B))$ because only objects of positive weight are allowed in lists, and $[]$ has zero weight.

The set $S(X)$ contains all possible multisets of finite size, each element of which belongs to X and has positive weight. For example, $S(B)$ contains $\{\}$, $\{u\}$, $\{u, u\}$, $\{u, u, u\}$, etc. Another example: $S(L(B))$ contains such sets as $\{[u]\}$, $\{[u], [u]\}$. Note that multiset may contain several equal objects. Another example: $\{[u], [u, u]\}$, note that in a multiset the order doesn't matter, so this is the same as $\{[u, u], [u]\}$.

The set $C(X)$ contains all possible cycles of finite length, each element of which belongs to X and has positive weight. Two cycles are considered equal if one can be converted to another by a cyclic shift. For example $C(L(B))$ contains $([u], [u, u], [u, u, u])$. Note that this this object is the same as $([u, u], [u, u, u], [u])$, but not the same as $([u, u, u], [u, u], [u])$.

Again, the weight of a list, a set, or a cycle is the sum of weights of its elements. So, for example, the weight of $([u], [u, u], [u, u, u])$ is 6.

The final way to construct the new class of objects is pair. If X and Y are sets of objects $P(X, Y)$ is the set of ordered pairs of objects where the first component is from X and the second one is from Y. For example, $P(S(B), L(B))$ contains $\langle\{u, u\}, [u, u, u]\rangle$ or $\langle\{\}, [u]\rangle$. Note that unlike lists, sets, or cycles, pairs can have zero-weight components.

Given k find the description of a set of combinatorial object that contains 1 element of weight 0, 1 element of weight 1, 2 elements of weight 2, 5 elements of weight 3, 14 elements of weight 4, 42 elements of weight 5 and k elements of weight 6.

Input

The input file contains multiple test cases.

Each test case contains a single integer k on a line by itself ($120 \le k \le 140$).

Input is followed by a line with $n = 0$.

Output

For each test case print the description of a set of combinatorial objects in the format described in problem statement on a line by itself. Your description must have length of at most 2000. Do not print spaces.

Examples

catalan.in
125 0
catalan.out
L(P(L(P(L(L(B)),B)),P(B,L(P(P(B,B),P(B,B))))))

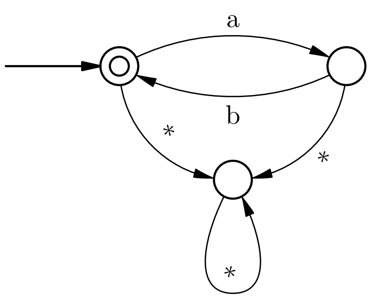
Problem D. Decomposable Single Word Languages

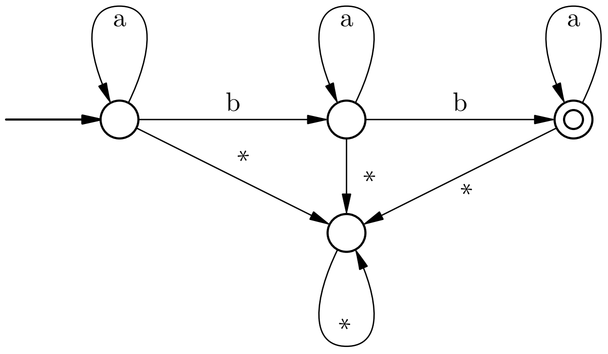
Input file:	`decomposable.in`
Output file:	`decomposable.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

Deterministic finite automaton (DFA) is an ordered set $\langle \Sigma, U, S, T, \varphi \rangle$ where Σ is the finite set called *input alphabet*, in this problem $\Sigma = \{\text{a, b, \ldots, z}\}$, U is the finite set of *states*, $S \in U$ is the *initial state*, $T \subset U$ is the set of *terminal states* and $\varphi : U \times \Sigma \to U$ is the *transition function*.

The input of the automaton is the string α over Σ. Initially the automaton is in state s. Each step the automaton reads the first character c of the input string and changes its state to $\varphi(u, c)$ where u is the current state. Then the first character of the input string is removed and the step repeats. If after its input string is empty the automaton is in the terminal state, it accepts the initial string α, in the other case it rejects it. The set of all words accepted by an automaton A is denoted as $L(A)$.

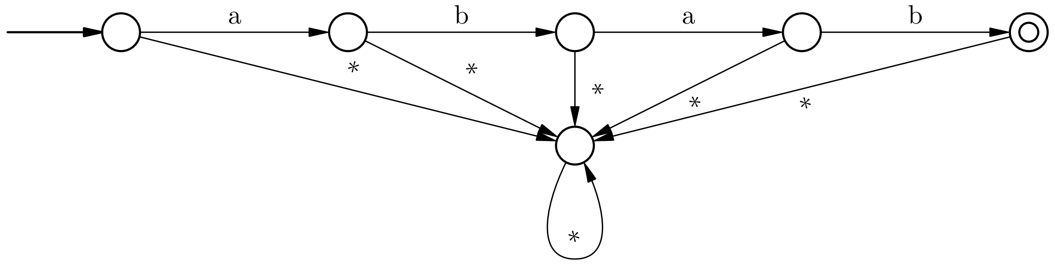
One can visualize DFA as a directed graph representing its states as vertices and its transitions as edges marked with characters. Terminal states are shown as double circled vertices, the initial state is marked by an arrow. The picture below on the left shows the automaton for a language $(ab)^*$ of words that consist of zero or more repeated words "ab". The picture below on the right shows the automaton for a language $a^*ba^*ba^*$ that consist of "a" and "b" and contain two "b"-s. For sake of clarity edges marked by "$*$" represent all transitions not explicitly drawn.





A set X of words is called a *regular language* if it is equal to $L(A)$ for some DFA A. The *index* of a regular language X denoted as $ind(X)$ is the minimal number of states in a DFA A such that $L(A) = X$. For example, the two automatons shown on the picture above are indeed the minimal DFA-s for the described languages, so $ind((ab)^*) = 3$ and $ind(a^*ba^*ba^*) = 4$.

It is well known that if X_1 and X_2 are two regular languages its intersection $X_1 \cap X_2$ is also a regular language. For example, the intersection of the two languages described above is the language $Y = (ab)^* \cap (a^*ba^*ba^*) = \{abab\}$ that contains a single word "abab". Clearly a single word language is regular, the automaton for Y is shown on the picture below.



It is easy to see that if W is a single word language $W = \{w\}$, and length of w is n, the index of W is equal to $n + 2$.

A regular language X is called decomposable if X can be represented as an intersection of two regular languages $X = X_1 \cap X_2$ and $ind(X) > ind(X_1)$ and $ind(X) > ind(X_2)$. For example, the single word language $Y = \{\text{abab}\}$ is decomposable.

Given a word w of length n find whether the single word language $W = \{w\}$ is decomposable and if it is, find two automatons A_1 and A_2 such that number of states in both A_1 and A_2 is less than $n + 2$ and $W = L(A_1) \cap L(A_2)$.

Input

The input file contains multiple test cases.

Each test case consists of a word w on a line on itself, w consists of lowercase letters of the English alphabet, length of w is between 1 and 50, inclusive.

There are at most 100 tests in one input file.

Output

For each test case first print «YES» if the corresponding single-word language is decomposable, or «NO» if it is not. If the language is decomposable, the description of two DFA-s must follow. Each DFA description must start with k — the number of states, $1 \le k \le n + 1$, where n is the length of the input word. Let states be numbered from 1 to k, the initial state is the state number 1. Then print t — the number of terminal states, $1 \le t \le k$, followed by t integers from 1 to k — terminal states. The following k lines must contain 26 integers each: for a state u print $\varphi(u, \text{a})$, $\varphi(u, \text{b})$, …, $\varphi(u, \text{z})$.

Examples

decomposable.in
abab a

decomposable.out
YES 3 1 1 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 1 3 2 1 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 NO

Problem E. Elegant Square

Input file:	`elegant.in`
Output file:	`elegant.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

Many people know about magic squares — squares that contain distinct numbers and have equals sums of rows and columns. Recently Eve has heard about magic squares, and now she has invented her own version: *elegant squares*.

Eve calls a square of $n \times n$ integers elegant if the following conditions are satisfied:

- All entries of the square are distinct positive integers.
- All integers are square free. That means that no integer is divisible by t^2 for any $t > 1$.
- The product of numbers in any row and any column is the same.

For example, the picture below shows an elegant 3×3 square.

$$\begin{array}{ccc} 1 & 21 & 10 \\ 6 & 5 & 7 \\ 35 & 2 & 3 \end{array}$$

All of its entries are distinct positive square free integers, and product of any row and any column is 210.

Help Eve, find an $n \times n$ elegant square. All numbers in the square must not exceed 10^{18}. It is guaranteed that for the given constraints there exists such square.

Input

The input file a single integer n ($3 \le n \le 30$).

Output

Output $n \times n$ integers: the found elegant square. All printed integers must not exceed 10^{18}.

Examples

elegant.in	elegant.out

3	1 21 10
	6 5 7
	35 2 3

Problem F. Four Colors

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 megabytes

This is an interactive problem.

Fred and Fiona are tired of playing tic-tac-toe at boring lessons, so they have invented the new game. The board for the game is a connected undirected graph without cycles, also known as a tree. Players alternate their turns, Fred moves first. The players use pencils of four colors for the game: red, green, blue and yellow, we will denote colors by integers from 1 to 4.

Initially all vertices of the tree are uncolored. Each move the current player chooses some uncolored vertex and colors it with one of the four possible colors in such way that no two vertices connected by an edge have the same color.

If the current player has no moves because all vertices are colored already, or there is no vertex that can be correctly colored, the game ends and the winner is determined in the following way. If all vertices are colored, Fred wins, if there is an uncolored vertex, Fiona wins.

Friends have played many games, and Fiona has noticed that Fred always wins. After a thought she understood that there is a winning strategy for Fred that he has found. Can you do the same?

Interaction Protocol

Your program will play the game with judges' program via standard input and output.

First your program must read the description of the tree. If is specified by n — the number of vertices ($2 \le n \le 1000$), followed by $n - 1$ pairs of vertices u_i, v_i — the edges of the tree (vertices are numbered from 1 to n).

Then the interaction proceeds as follows. Your program makes the first move by printing the yet uncolored vertex number u and the color it wishes to color it c to standard output. Then it must read the opponent's move in the same format, make its move in return, and so on.

When the game is over because your program has won (either by coloring the last vertex, or by forcing the judges' program to color the last vertex), instead of opponent's move your program will read "-1 -1" from standard input. After reading it you program must terminate. Though you can deduce that you have won earlier, or understand that your move is the last move, you should exit only after reading "-1 -1" from standard input.

Examples

standard input	standard output
8	
1 2	
1 3	
2 4	
2 5	
3 6	
3 7	
3 8	
	1 1
6 2	
	7 2
8 3	
	3 4
4 2	
	2 3
-1 -1	

Note

Input is formatted to show which output makes response to the corresponding input. There will be no empty lines in real interaction.

In the given example the judges' program makes the last move (either "5 1", "5 2" or "5 4") but doesn't report it to your program, because after that the game is over and your program has won.

Problem G. Greater Number Wins

Input file:	`greater.in`
Output file:	`greater.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

George and Gordon are playing a game called "Greater number wins".

The game proceeds as follows. Each player has a row of d cells. Initially all cells are empty. There are two variants of the game, both are considered in this problem.

In the first variant the players make move in turn, George moves first. Each move the player rolls a special dice which generates a random digit from 0 to $b-1$, each with equal probability. After that the player puts the digit to one of the free cells of his row.

After each player has made d moves the game is over and the winner is the player who has greater number in base b written in his row. If both players have the same number the game is draw.

The second variant of the game is almost the same, but first George makes his d moves, and then Gordon makes his d moves.

Given d and b find the maximal winning probability that George can achieve by choosing correct moves in the first and the second variant of the game, respectively, regardless of Gordon's moves.

Input

The input file contains multiple test cases.

Each test case contains two integers d and b on a line ($1 \le d \le 10$, $2 \le b \le 10$, $(b+1)^d \le 3000$).

Input is followed by a line with $d = b = 0$.

Output

For each test case output two numbers on a line: the probability that George would win in the first and the second variant of the game, respectively. Your answer must be accurate up to 10^{-6}.

Examples

greater.in	greater.out

`1 2`
`2 2`
`0 0`
	`0.25 0.25`
	`0.3125 0.3125`

In sample test for both variants each player must play using the following strategy. If he gets 1, put it to the leftmost empty cell of his row, which corresponds to the most significant digit, if he gets 0, put it to the rightmost empty cell of his row. If there is only one cell George wins if he gets 1 and Gordon gets 0, the probability of such outcome is 1/4. If there are two cells, five outcomes of 16 possible are good for George: 11 vs 10, 01, or 00; 10 vs 00; and 01 vs 00.

Problem H. Higher Math Lesson

Input file:	`higher.in`
Output file:	`higher.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

Henry is sleeping at a boring higher math lesson in National Search University of This and That. When he suddenly wakes up the teacher asks him to solve the following problem. Given an $n \times n$ matrix A of integers Henry must find invertible integer matrices L and R such that the following conditions are satisfied:

- $B = LAR$ is a diagonal matrix;
- there is such j ($0 \le j \le n$) that $b_{i,i} = 0$ if and only if $i > j$;
- for all i from 2 to j the number $b_{i,i}$ is divisible by $b_{i-1,i-1}$.

An integer matrix C is called invertible if there exist an integer matrix C^{-1} such that $CC^{-1} = I$ where I is a unit matrix.

Henry has been sleeping for most of lessons, so he doesn't know how to do it. So he asks you to help.

Input

The input file contains multiple test cases.

Each test case starts with an integer n — size of a matrix, followed by n lines of n integers each — the given matrix ($2 \le n \le 5$, elements of matrices are from -10 to 10).

Input is followed by a line with $n = 0$. Each input file contains at most 100 test cases.

Output

For each test case print four integer matrices: L, L^{-1}, R and R^{-1}. It is guaranteed that such matrices always exist. Separate matrices by a blank line. If there are several solutions, print any one.

Examples

higher.in	higher.out

`3`
`1 2 3`
`6 5 4`
`7 8 9`
`0`
	`1 0 0`
	`1 1 -1`
	`-13 -6 7`
	
	`1 0 0`
	`6 7 1`
	`7 6 1`
	
	`1 -2 1`
	`0 1 -2`
	`0 0 1`
	
	`1 2 3`
	`0 1 2`
	`0 0 1`

In the given example

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & -1 \\ -13 & -6 & 7 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 1 & -2 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Problem I. Isomorphism

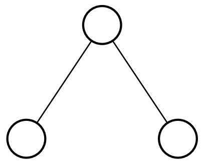
Input file:	`isomorphism.in`
Output file:	`isomorphism.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

Graph isomorphism is an important problem in Computer Science. It is not known whether the polynomial algorithm exists for this problem, neither it is known to be NP-complete.

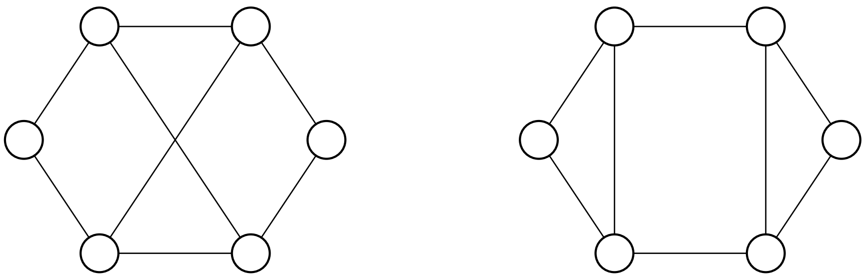
Two undirected graphs G and H are called isomorphic if they have the same number of vertices and there exists a bijection $\varphi : VG \to VH$ such that there is an edge uv in G if and only if there is an edge $\varphi(u)\varphi(v)$ in H. There are some characteristics of graphs that are invariant under isomorphism. One of such parameters is *degree profile of the graph.*

The degree $\deg(u)$ of a vertex u is the number of other vertices connected to u by edges. Consider a connected undirected graph G with n vertices. For each vertex u find sets $V_{u,0}, V_{u,1}, \ldots, V_{u,n-1}$ of vertices at distance $0, 1, \ldots, n-1$ from u (some of these sets may be empty). For each such set find the multiset $D_{u,i}$ of degrees of vertices from $V_{u,i}$. The list of these multisets $D_u = [D_{u,0}, D_{u,1}, \ldots, D_{u,n-1}]$ is the *degree profile* of vertex u. The multiset of degree profiles of all vertices of the graph is its degree profile.

For example, the graph displayed below has degree profile $\big\{[\{1\},\{2\},\{1\}],[\{2\},\{1,1\},\varnothing],[\{1\},\{2\},\{1\}]\big\}$.



It is clear that degree profile is invariant under isomorphism. However, there can be graphs that have the same degree profile but are not isomorphic. The example of two such graphs is shown on the picture below. Degree 2 vertices of both graphs have degree profiles $[\{2\},\{3,3\},\{3,3\},\{2\},\varnothing,\varnothing]$, and degree 3 vertices have degree profiles $[\{3\},\{2,3,3\},\{2,3\},\varnothing,\varnothing,\varnothing]$, but graphs are clearly not isomorphic.



Note that when different degree profiles prove that graphs are not isomorphic, same degree profiles do not give easy way to find isomorphism even if it exists, because correspondence between vertices of the same degree profile can be difficult to establish. There is however class of graphs for which degree profile allows to easily check for isomorphism. These are graphs where all vertices have different degree profiles. Let us call such graphs *degree distinguishable.*

However, even degree distinguishable graphs can have the same degree profile but be non-isomorphic. Given n find two non-isomorphic connected degree distinguishable graphs with n vertices that have the same degree profile.

Input

The input file contains one integer n ($3 \le n \le 100$).

Output

If there are no two non-isomorphic degree distinguishable graphs with n vertices that have the same degree profile, print "`NO`" at the first line of the output file. In the other case print "`YES`" followed by two graph description.

Each description must start with m — number of edges, followed by m pairs of integers: pairs of vertices connected by edges. There must be at most one edge between a pair of vertices, no edge must connect a vertex to itself.

Vertices of each graph must be numbered from 1 to n in such way that vertices i of both graphs have the same degree profile. No two vertices of the same graph must have the same degree profile.

Note

The second example gives two non-isomorphic graphs with the same degree profile but not degree distinguishable. They are provided to illustrate output format, but such output for $n = 6$ will not be accepted.

Examples

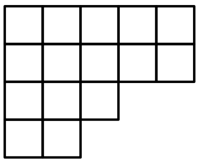
`isomorphism.in`	`isomorphism.out`
`3`	`NO`

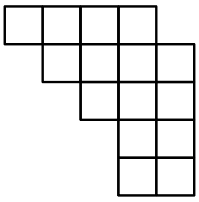
`6`	`YES`
	`8`
	`1 2`
	`1 6`
	`2 3`
	`2 5`
	`3 4`
	`3 6`
	`4 5`
	`5 6`
	`8`
	`1 2`
	`1 6`
	`2 3`
	`2 6`
	`3 4`
	`3 5`
	`4 5`
	`5 6`
	Note that this is incorrect output

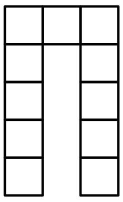
Problem J. Jinxiety of a Polyomino

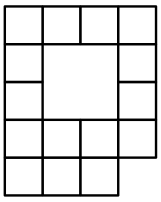
Input file:	`jinxiety.in`
Output file:	`jinxiety.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

A polyomino is a connected set of unit squares on a square grid. The picture below shows 4 examples of polyominoes.


(a)

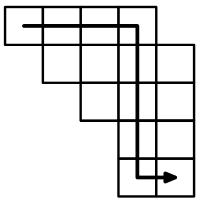

(b)


(c)


(d)

Polyomino is called *convex* if its intersection with any vertical or horizontal line is a segment. The picture above shows two convex polyominoes (a) and (b) and two non-convex ones (c) and (d).

Two squares are called adjacent if they share a common side. It is easy to see that for any two squares of a convex polyomino it is possible to get from any square to any other one moving from a square to adjacent one and using only two directions. The picture below shows an example of such path for polyomino (b).



For a convex polyomino P let us define its *jinxiety* $J(P)$ as a minimal k such that it is possible to get from any square to any other square by a path that uses two directions and makes at most k turns. For example, the polyomino (a) has jinxiety of 1 and polyomino (b) has jinxiety of 2.

Given a convex polyomino you have to find its jinxiety.

Input

The input file contains multiple test cases.

Each test case contains two integers h and w — the number of rows and columns in polyomino description, respectively ($1 \le h, w \le 2000$).

The following h lines contain w characters each and describe the polyomino. Each character is either "`.`" for an empty square, or "`#`" for a polyomino square. It is guaranteed that the described figure is a convex polyomino.

Input is followed by a line with $h = w = 0$. The total number of characters in all polyomino descriptions of the input file is at most $4 \cdot 10^6$. There are at most 40 000 tests.

Output

For each test case print one integer: the jinxiety of the polyomino in the input.

Examples

jinxiety.in	jinxiety.out
4 5 ##### ##### ###.. ##... 5 5 ####. .#### ..### ...## ...## 0 0	1 2