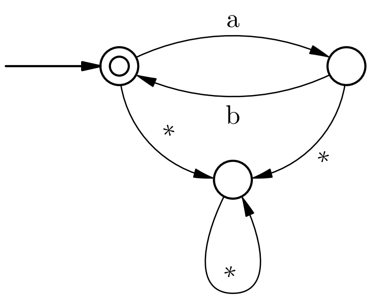
Problem D. Decomposable Single Word Languages

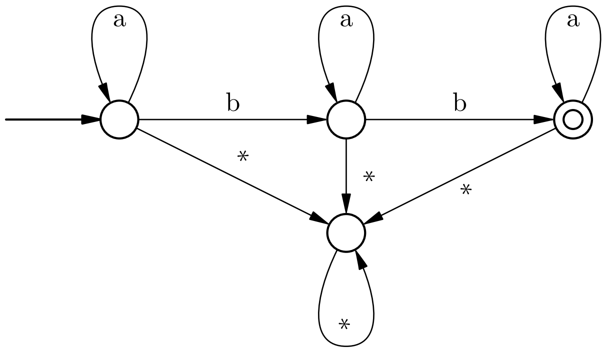
Input file:	`decomposable.in`
Output file:	`decomposable.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

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 $\Sigma = \{\text{a, b, \dots, z}\}$, 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 $L(A)$.

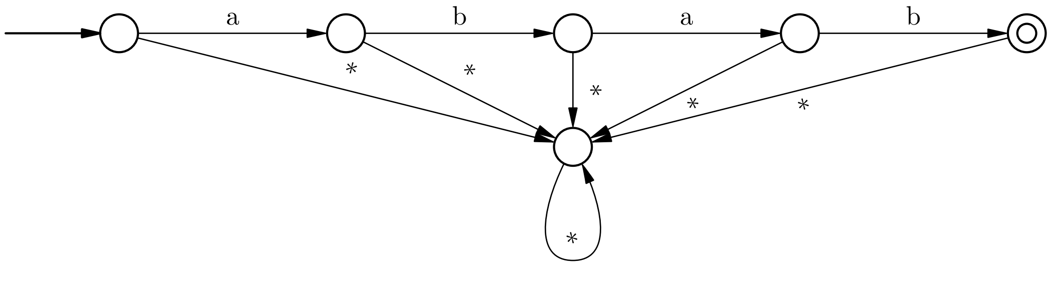
One can visualize DFA as a directed graph representing its states as vertices and its transitions as edges marked with characters. Terminal states are shown as double circled vertices, the initial state is marked by an arrow. The picture below on the left shows the automaton for a language $(ab)^*$ of 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.

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

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 probability that George would win in the first and the second variant of the game, respectively, if both players are trying to maximize their chances to win.

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 I. Isomorphism

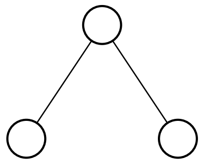
Input file:	`isomorphism.in`
Output file:	`isomorphism.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

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.

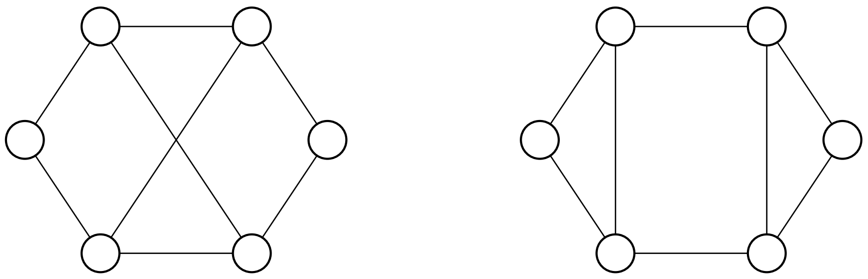
Two undirected graphs G and H are called isomorphic if they have the same number of vertices and there exists a bijection $\varphi : VG \to VH$ such that there is an edge uv in G if and only if there is an edge $\varphi(u)\varphi(v)$ in H. There are some characteristics of graphs that are invariant under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

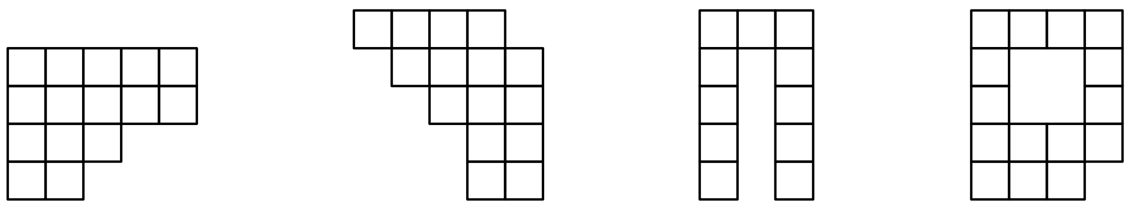
Examples

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

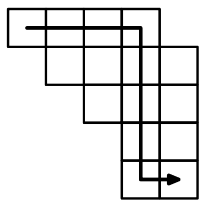
Input file:	`jinxiety.in`
Output file:	`jinxiety.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

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

Problem K. Kill The PSU

Input file:	`killthepsu.in`
Output file:	`killthepsu.out`
Time limit:	2 seconds
Memory limit:	256 mebibytes

You are working on the new software which will control user's PC health and help to fix usual software/hardware troubles.

Each device and its driver fall under one of three categories:

- Category I: Device just time from time goes to powersaving (sleep) mode once in a while. When it does, wake them!
- Category II: Device can suddenly block the system. Solution is to load driver if it is unloaded, or unload it, if it is loaded. Note that drivers are experimental, so at the boot time the OS does not load them.
- Category III: Important devices. If such a device seems to be reported in the log, add an extra power to it if you can, or recommend to buy the new power supply in case when lack of power for atleast one of those devices is critical.

  Your power supply (PSU) gives 100 watts for each important device at the moment of system boot. When important device is reported in the log, its lose 10 watts of power from PSU. Initially, your PSU allows to add extra 20 watts of the power. So while it is possible, those power losses are compensated (namely, it happen in first two powerloss cases). When PSU will give for at least one important device only 10 or less watts of power, recommend to buy the new power supply and exit the program regardless of next messages in the log (they should not be processed).

Given the list of devices in each category and the log of troubles, print out recommendations to the user.

Input

The first line of input will be T, the number of test cases ($0 \le T \le 100$). T cases follow.

The first line of each test case will be four integers A, B, C, D, separated by spaces. The first three numbers will be the number of devices in Category I, II and III, respectively. The fourth number is the number of driver/device troubles.

After this line follow A lines with the names of Category I devices, B lines with the names of Category II devices, C lines with Category III devices, and then D lines with log of troubles, where each trouble is described by the name of failed device ($0 \le A, B, C, D \le 100$, each device name will start and end with a lowercase English letter '`a`' — '`z`', and will consist of between 2 and 22 lowercase English letters and spaces).

Output

For each test case and for each trouble, output the recommendation in the new line in order of troubles appearing in the log file.

For Category I devices print "`wake` *devicename*". For Category II devices print "`load` *devicename*", if driver was not loaded (in the beginning or after unload) and "`unload` *devicename*" otherwise. For Category III devices print "`power fail on` *devicename*" if remaining power on device is greater than 10 watts (remember about 20 watts of reserve you had in beginning), or "`buy the new PSU`" otherwise. After printing recommendation about replacing PSU, no more events in this test case must be processed. Follow the sample format if something is unclear.

Examples

killthepsu.in	killthepsu.out
2	wake video card
1 1 2 5	load wireless network
video card	unload wireless network
wireless network	power fail on south bridge
south bridge	power fail on cpu
cpu	power fail on motherboard
video card	power fail on motherboard
wireless network	power fail on motherboard
wireless network	power fail on motherboard
south bridge	power fail on motherboard
cpu	power fail on motherboard
0 0 1 11	power fail on motherboard
motherboard	power fail on motherboard
motherboard	power fail on motherboard
motherboard	power fail on motherboard
motherboard	buy the new PSU
motherboard	
motherboard	
motherboard	
motherboard	
motherboard	
motherboard	
motherboard	
motherboard	

Problem L. Lucky tickets

Input file:	`lucky.in`
Output file:	`lucky.out`
Time limit:	1 second
Memory limit:	512 mebibytes

Little Kostya's dad often travels by tram. And he always buys the ticket. In order that the boy was fond of mathematics, dad tells him interesting facts from the world of numbers.

Today he told the task about lucky tickets, but instead of the accumulation operation of digits used multiplication. Ticket ID consists of $2N$ digits. In ID allowed to use only certain digits. The ticket is lucky, if the product of the first N digits equals to the product of the last N digits.

Find the total number of lucky tickets.

Input

The first line of input contains an integer N $(1 \le N \le 9)$. The second line contains digits that you can use in the ID of tickets. Digits written without spaces and all different.

Output

Output a single integer equals to the number of lucky tickets.

Examples

`lucky.in`	`lucky.out`

`1`
`7325`
	`4`

`9`
`1`
	`1`

`3`
`123456789`
	`7713`

`2`
`2015`
	`64`

Problem M. Mosaic

Input file:	`mosaic.in`
Output file:	`mosaic.out`
Time limit:	1 second
Memory limit:	512 mebibytes

Little Kostya likes not only playing mosaic but with pleasure comes up with a different games while folding it back into the box.

Mosaic has the shape of a rectangle $M \times N$ each cell of which contains one tile. On each tile written an integer from 1 to $M \cdot N$. Assembled mosaic has the following form:

1	2	3	$\dots$	$N-1$	N
$N+1$	$N+2$	$N+3$	$\dots$	$2 \cdot N - 1$	$2 \cdot N$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
$(M-1) \cdot N + 1$	$(M-1) \cdot N + 2$	$(M-1) \cdot N + 3$	$\dots$	$M \cdot N - 1$	$M \cdot N$

Kostya wants to play a game. He should remove as many tiles of the mosaic according to certain rules. In a single turn he can remove any tile that has 4 adjacent tiles (tiles are considered to be adjacent if they are in adjacent cells in the same row or in the same column).

Write a program that will determine the maximum number of tiles that Kostya can remove by the above rules. And find any correct maximum set of tiles.

Input

The only input line contains two integers M and N ($1 \le M, N \le 100$) the height and width of the mosaic.

Output

In the first line print the maximum number of tiles that Kostya can remove by the above rules. In the second line print the tile numbers in the order in which they can be removed. If there are several solutions, output any of them.

Examples

mosaic.in	mosaic.out

`3 3`	`1`
	`5`

`2 1`	`0`

`4 4`	`2`
	`6 11`

`4 5`	`3`
	`7 9 13`

Problem N. New Battle Tactics

Input file:	`newbattle.in`
Output file:	`newbattle.out`
Time limit:	1 second
Memory limit:	512 mebibytes

The time has come for the battle between the Dwarves and the Orcs. The Dwarven army, which consists of N warriors, is standing in a line. Every Dwarf has a strength attribute — an integer between 1 and N. All strengths are unique, that is, the strengths of different warriors are different. King Dáin has strength B.

King Dáin wants to develop a tactic for the upcoming battle. He wants several (possibly 0) of the leftmost and several (possibly 0) of the rightmost warriors in the line to hold their positions. Their job is to secure the flanks and protect the army from any flanking maneuvers.

The rest of the Dwarves will charge forward and attack the Orcs. Without doubt, King Dáin wants to be among the attacking Dwarves in order to boost their morale. However, he doesn't want to stand out among the attacking group, that is, he wants his strength to be equal to the median of the group's strengths. In other words, if the attacking group consists of $2k + 1$ Dwarves, then in the sorted list of their strengths King Dáin wants to be in the k-th position, so that the attacking group always consists of an odd number of Dwarves.

Your task is to calculate the number of different ways to choose the attacking group while also satisfying King Dáin's requirements. Two attacking groups are different if the sets of Dwarves they consist of are different.

Input

The first line contains 2 positive integers N and B ($1 \leq N \leq 10^5$, $1 \leq B \leq N$), seperated by a space — the number of warriors in the Dwarven army and strength of King Dáin.

The second line contains N space-separated positive integers $P_1, P_2, \ldots, P_N$ ($1 \leq P_i \leq N$, all P_i are different) — the strengths of the Dwarven army warriors from left to right.

Output

Output in a single line the number of different possible attacking groups.

Examples

newbattle.in	newbattle.out
3 2 2 3 1	2
5 3 1 2 3 4 5	3
6 1 4 3 5 2 1 6	1