Problem A. Abstract Dances

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

Abstract dances are getting more and more popular. Peter's dance teacher, Mr Kot O'Lahn invented a series of abstract dances that he is going to present at an upcoming dance festival.

Abstract dance is represented by one *movement*. A movement is composed of a series of *figures*, presented one after another and separated by *turns*. A figure consists of a *step* followed by some (possibly empty) set of movements, separated by *flips*, followed by a *jump*.

Now Peter wants to count how many different abstract dances exist. He noticed that some dances, being not exactly different, are still very similar. Peter calls two figures *equivalent*, if their corresponding sets of movements are equal as multisets (order of movements doesn't matter). Peter calls two movements *equivalent* if one can obtained from another by a circular shift of its figures. Peter calls two dances *different*, if their corresponding movements are not equivalent.

Formally, let figure F_1 consist of movements $m_1, m_2, \ldots, m_k$ and figure F_2 consist of movements $n_1, n_2, \ldots, n_k$. Figures F_1 and F_2 are considered equivalent, if there is a bijection $\varphi : \{1 \ldots k\} \to \{1 \ldots k\}$ such that m_i is equivalent to $n_{\varphi(i)}$ for all i from 1 to k. Let movement M_1 consist of figures $f_1, f_2, \ldots, f_l$ and movement M_2 consists of figures $g_1, g_2, \ldots, g_l$. Movements M_1 and M_2 are considered equivalent, if there is j such that f_i is equivalent to $g_{(i+j) \bmod n+1}$ for all i from 1 to l.

For example, let us denote turn as "`,`", step as "`(`", jump as "`)`" and flip as "`!`". Then figures "`((()))`" and "`(()!())`" are different, but figures "`(()!(()))`" and "`((())!())`" are equivalent (because they only differ by the order of movements between a step and a jump). Movements "`(),(()),((()))`" and "`(()),(),((()))`" are different, but movements "`(),(()),((()))`" and "`(()),((())),()`" are equivalent because they only differ up to a cyclic shift.

Recently little Peter got interested, how many different abstract dances with n steps are there. Help him to find that out. For example, there are 5 abstract dances with 3 steps: "`((()))`", "`(()!())`", "`((),())`", "`(()),()`" (equivalent to "`(),(())`"), "`(),(),()`".

Input

Input contains several test cases. Each test case consists of one integer n ($1 \le n \le 60$) on a line by itself. The last test case is followed by $n = 0$ which must not be processed.

Output

For each test case print its number followed by the number of different abstract dances of n steps. Adhere to the format of sample output.

Examples

abstract.in	abstract.out
1 2 3 4 5 0	Case 1: 1 Case 2: 2 Case 3: 5 Case 4: 14 Case 5: 42

Problem B. Betting Fast

Input file:	`betting.in`
Output file:	`betting.out`
Time limit:	1 second
Memory limit:	256 megabytes

Betting in a casino could just be the way to spend free time. However, some people try to consider it as a way to earn money. Usually they finally lose all they have, but Josh thinks that he will win. Actually, he is planning to win fast.

Josh comes to casino with s dollars. His goal is to have n dollars. He plays some card game, in which the player has probability $p\%$ of winning and doubling his bet ($1 \le p \le 50$). If Josh has x dollars, he can choose any sum y from 1 dollar to $\min(x, n - x)$ and bet it. With probability $p\%$ he would win and have $x + y$ dollars, with probability $100 - p\%$ he would lose and have $x - y$ dollars.

If Josh has 0 dollars, he loses and leaves casino with no money, sad and lonely. If he has n dollars, he is satisfied and goes home with n dollars, fun and girls. However, Josh doesn't want to play for too long. Therefore he is only planning to play at most t games. If after t games he still doesn't have n dollars, he leaves casino with some money, but still sad and lonely. Josh plays using optimal strategy to maximize the chance that he would win in t games or less.

Help Josh to find probability that he would win.

Input

Input contains several test cases.

Each test case consists of four integers on a line: n, s, p and t ($2 \le n \le 10^5$, $1 \le s < n$, $1 \le p \le 50$, $1 \le t \le 100$).

The last test case is followed by four zeroes which must not be processed.

Input file contains at most 1500 tests.

Output

For each test case print one floating point number — the probability that Josh would win and leave casino with n dollars. Your output must be accurate up to 10^{-9}.

Examples

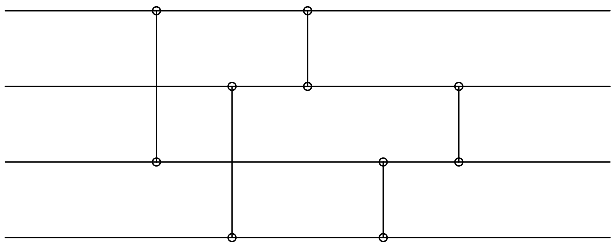
betting.in	betting.out
10 5 50 1	0.5
10 3 50 1	0.0
10 3 50 10	0.2998046875
10 8 50 10	0.7998046875
10 9 10 10	0.2723910049
0 0 0 0	

Problem C. Comparator Networks

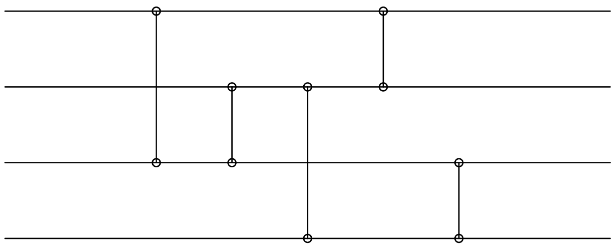
Input file:	`comparator.in`
Output file:	`comparator.out`
Time limit:	1 second
Memory limit:	256 megabytes

Comparator network is an abstract mathematical model of a network constructed of wires and comparator modules that is used to sort a sequence of numbers. Each comparator connects two wires and sorts the values by sending the smaller value to the upper wire, and the larger to the lower one. The main difference between sorting networks and comparison sorting algorithms is that the sequence of comparisons is set in advance, regardless of the outcome of previous comparisons.

Comparator network is called *sorting* if regardless of input values its output values are sorted. The picture below shows sorting network for 4 wires.



In order to test whether comparator network is sorting, so called 0-1 principle is often used. It claims that comparator network is sorting if and only if it sorts any input consisting of 0-s and 1-s. For example, the sorting network on the picture below is not sorting because it doesn't sort input $[1, 0, 1, 0]$.



Actually, it turns out that input $[1, 0, 1, 0]$ is the only 0-1 vector that is not sorted by this network. In this problem we are interested in such networks that sort any 0-1 vector except the given one.

You are given several 0-1 vectors. For each vector construct comparator network that sorts any 0-1 vector except the given one, or find out that it is impossible.

Input

Input contains several test cases.

Each test case starts with integer n ($2 \le n \le 10$) — number of wires, followed by n integers $a_1, a_2, \ldots, a_n$ ($a_i \in \{0, 1\}$).

The last test case is followed by $n = 0$ which must not be processed.

Output

For each test case print "-1" if no comparator network sorts any 0-1 vector except $[a_1, a_2, \ldots, a_n]$. In the other case output the description of such network. The description must consists of m — the number of comparators ($0 \leq m \leq 1000$), followed by comparator descriptions. Each comparator must be described by numbers of wires it connects.

It is guaranteed that if the answer exists, there exists an answer which contains at most 100 comparators (and you can use up to 1000).

Examples

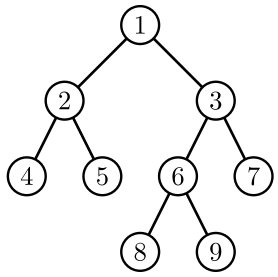
comparator.in	comparator.out

4
1 0 1 0
4
1 1 1 1
0
	5
	1 3
	2 3
	2 4
	1 2
	3 4
	-1

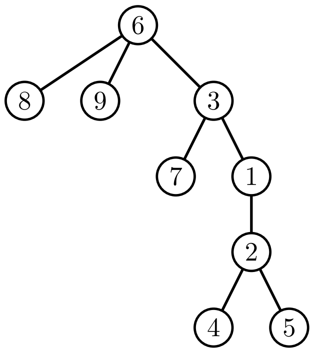
Problem D. Dynamic LCA

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

Least common ansestor problem for trees is stated as follows. Given a rooted tree T and two vertices u and v, $lca(u, v)$ is the vertex with maximal depth that is an ancestor of both u and v. For example, $lca(8, 7)$ in a tree on a picture below is 3.



We can change the root of the tree with the operation $chroot(u)$ by marking u as a new root of the tree and directing edges of the tree away from the root. Least common ancestors of vertices change accordingly. For example, if we perform $chroot(6)$ on a picture above, the tree changes to the one below and $lca(8, 7)$ becomes 6.



You are given a tree T. Initially vertex 1 is the root of the tree. Write a program that supports the following two operations: $lca(u, v)$ and $chroot(u)$.

Input

Input contains several test cases.

The first line of each test case contains positive integer n — number of vertices in a tree ($1 \le n \le 100\,000$). The following $n - 1$ lines contain two integers each and describe edges of a tree. The line with one integer m — number of operations — follows ($1 \le m \le 200\,000$). The following m lines contain operations. Each operation is specified as either "`?` u v" for $lca(u, v)$ or "`!` u" for $chroot(u)$.

The last test case is followed by $n = 0$ which must not be processed.

The sum of n for all test cases in the input file doesn't exceed $100\,000$. The sum of m for all test cases in the input file doesn't exceed $200\,000$.

Output

For each "`?` u v" operation print the value of $lca(u, v)$ on a line by itself.

Examples

dynamic.in	dynamic.out
9 1 2 1 3 2 4 2 5 3 6 3 7 6 8 6 9 10 ? 4 5 ? 5 6 ? 8 7 ! 6 ? 8 7 ? 4 5 ? 4 7 ? 5 9 ! 2 ? 4 3 0	2 1 3 6 2 3 6 2

Problem E. Equal Hash Codes

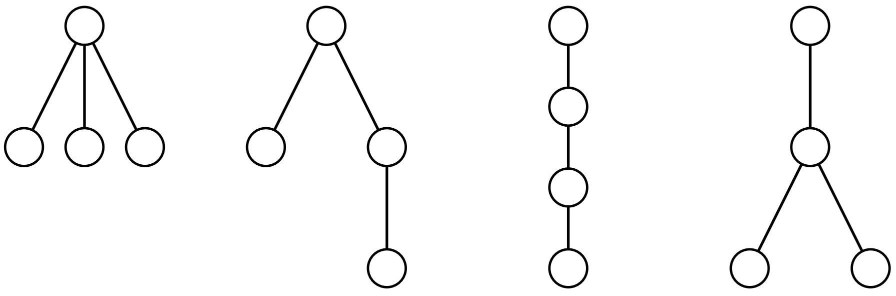
Input file:	`equal.in`
Output file:	`equal.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Many of you have probably heard that there are some test cases for which polynomial hash function for strings with modulo 2^k has many collisions. However, even more surprising for some of you might be the fact that polynomial hash function for *trees* when checking for isomorphism can have many collisions for *any* modulo.

It can be explaind in the following way. Let us consider polynomial hash function for trees. A tree T with root u and child subtrees $T_1, T_2, \ldots, T_k$ has hash function $h(T) = (1+h(T_1)t+h(T_2)t^2+\ldots+h(T_k)t^k) \bmod R$ for some t and R, where hash values for subtrees are ordered so that $h(T_1) \le h(T_2) \le \ldots \le h(T_k)$. Let us ignore R for a while, and set t as a variable. In this case hash function of a tree is actually a polynimoial in t. Let us call this polynomial a *hash polynomial* of a tree.

When comparing polynomials, for the purpose of ordering children of u, compare them by degree first, and then by coefficients, starting from highest. So, for example, $t^3 > t^2 + 2t + 1$ and $t^2 + 2t > t^2 + t + 1$.

It turns out, that many trees can have the same hash polynomial. For example, all four trees on a picture below have the same hash polynomial $t^3 + t^2 + t + 1$.



It is clear that if two trees have the same hash polynomial, they will have the same hash value for most selections of t and R (the only reason their hash values could differ is because of overflow modulo R which changes sort order of subtrees).

You are given n and a polynomial $p(t)$. Find all pairwise non-isomorphic rooted trees with n vertices that have hash polynomial $p(t)$.

Input

Input contains several test cases.

The first line of each test case contains n — the number of vertices in a tree ($1 \le n \le 18$). The following line contains d — the degree of the polynomial p ($0 \le d \le 18$). The following line contains $d+1$ numbers: $a_d, a_{d-1}, \ldots, a_1, a_0$, hash polynomial p is $a_d t^d + a_{d-1} t^{d-1} + \ldots + a_1 t + a_0$ ($1 \le a_i \le 18$).

The last test case is followed by $n = 0$ which must not be processed. The sum of 2^n for all n in the input file doesn't exceed 300 000.

Output

For each test case print the number of rooted non-isomorphic trees with n vertices that have hash polynomial p. After that print trees themselves. Trees must be printed as parenthesis sequences using the following algorithm. Denote as $s(T)$ a string representation of T. If T is a single vertex, $s(T)$ =“`()`”.

In the other case let $T_1, \dots T_k$ be T's children subtrees, then $s(T) =$"`(`$s(T_1) \dots s(T_k)$`)`" where T_i must be ordered in non-decreasing order of their $s(T_i)$ representations. Tree descriptions must be ordered lexicographically.

The total number of trees to output in all test cases of one input file will not exceed 10^5.

Examples

equal.in	equal.out

4
3
1 1 1 1
4
2
1 2 1
0
	4
	(((())))
	((()()))
	((())())
	(()()())
	0

Problem F. Funny Game

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

Vera and Harry are playing a funny game with the cake their mom had cooked yesterday. The cake has the form of a rectangle with width w and height h divided to unit cells. Vera and Harry make moves in turn, Vera moves first. Before each move there are several pieces of cake with sizes $w_1 \times h_1, \ldots, w_k \times h_k$ on the table. The player to move can choose any piece and cut it.

Vera cuts a piece along one or more vertical lines to get several pieces of equal size. Unit cells cannot be divided. For example, Vera can cut a piece of size 6×4 either to 2 pieces of size 3×4, or to 3 pieces of size 2×4 or to 6 pieces of size 1×4.

Harry cats a piece along one or more horizontal lines ot get several pieces of equal size. Similarly, unit cells cannot be divided. For example, Harry can cut a piece of size 6×4 either to 2 pieces of size 6×2 or to 4 pieces pieces of size 6×1.

Players are not allowed to rotate or flip cake pieces.

A player who cannot make turn because all of pieces have size 1 along the corresponding size, loses.

Now Vera and Harry wonder, who would win depending on the size of the cake if both players act optimally. Help them find it out.

Input

Input contains several test cases.

Each test case consists of two integers: w and h ($1 \le w, h \le 10^7$).

Input file is terminated with a line containing two zeroes which must not be processed. The total number of test cases in the input file doesn't exceed 10^5.

Output

For each test case print w and h followed by a dash followed by the name of the player who wins. Adhere to the format of sample output.

Examples

funny.in	funny.out

4 4
6 4
4 6
6 3
0 0
	4 4 - Harry
	6 4 - Harry
	4 6 - Harry
	6 3 - Vera

Problem G. Grand Tour

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

The head of Flatland parliament, Mr Notmyshkin has decided to take a grand tour and visit n leading universities of Flatland to talk to students about innovation and modernization. Visiting university takes exactly one day, so universities will be visited in some order on days $1, 2, \ldots, n$. But there are some limitations on the tour. Some universities are more important than others. More important universities must be visited earlier. Totally there are m constraints of a form "university x must be visited before university y".

Since it is difficult to gather students for a lecture during vacations, it is preferrable for Mr Notmyshkin to visit each university before summer vacations start there. For each university the last day of the semester d_i is known. If the last day of the semester in the university is d_i and Mr Notmyshkin visits it at day a_i, *discomfort* for this university is $\max(0, a_i - d_i)$. The *badness* of the tour is maximal discomfort of a university caused by it.

Officials want to plan the tour in such way that badness of the tour was as small as possible. Help them to make such tour plan.

Input

Input contains several test cases.

The first line of each test case contains n — the number of universities ($1 \le n \le 100\,000$). The second line contains n integers: $d_1, d_2, \ldots, d_n$ — the last days of semester in each university ($1 \le d_i \le 10^9$). The following line contains m — the number of constraints ($0 \le m \le 100\,000$). The following m lines contain two integers each, pair x, y means that university x must be visited before university y. Universities are numbered from 1 to n. The set of constraints is consistent.

The last test case is followed by $n = 0$ which must not be processed. Both the sum of n and the sum of m in each input file doesn't exceed 10^5.

Output

For each test case print two lines. The first line must contain the minimal possible badness of the tour. The second line must contain n integers: order in which universities must be visited in order to achieve such badness. If there are several optimal solutions, output any one.

Examples

grand.in	grand.out

6
4 5 6 3 5 3
7
1 2
2 3
4 3
5 2
5 4
6 5
6 1
0
	0
	6 5 4 1 2 3

Problem H. Hats and Wizards

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

One day n wizards gathered around the table and decided to play the following game with Merlin.

Wizards sit down around the round table. They choose m positive integer numbers $k_1, k_2, \ldots, k_m$, such that $k_1 + k_2 + \ldots + k_m = n$. After that they are temporarily blindfolded and Merlin does the following. He chooses some permutation $p_1, \ldots, p_m$ of numbers from 1 to m. After that he takes k_{p_1} hats of color 1, k_{p_2} hats of color 2, etc, k_{p_m} hats of color m and puts them on the wizards. After that blindfolds are removed and the wizards can see the colors of each other's hats. However, nobody can see his own hat's color.

Now wizards are not allowed to talk or use gestures to communicate. Each wizard must write down the color of his own hat to a piece of paper and pass the answer to Merlin. Wizards cannot see each other's answers. Wizards' score is equal to the number of wizards that answered the color of their hats correctly.

Wizards decided that they will all use the same deterministic strategy to play the game. Each wizards sees some sequence of hat colors, starting from the one his left, following clockwise around the table, and finishing by the one on his right. For each sequence the players decided beforehands which color should the corresponding wizard write down to his piece of paper. So formally, *strategy* is the function $s : C \to \{1, \ldots, m\}$ where C is the set of all possible sequences of colors, wizards can see. Now they want to choose such function that however Merlin acts, their score was maximal possible in the worst case. Help them to choose such function s.

For example, if there are three wizards and $k_1 = 1$, $k_2 = 2$, there are either 2 hats of color 1 and 1 hat of color 2, or 1 hat of color 1 and 2 hats of color 2. Therefore there are the following sequences that a wizard can see: $C = \{(1,1), (1,2), (2,1), (2,2)\}$. One optimal strategy is the following: if the wizard sees two hats of the same color, answer another color. In the other case answer the color of the wizard to his left. So $s(1,1) = 2$, $s(1,2) = 1$, $s(2,1) = 2$, $s(2,2) = 1$. In this case the wizard which has the hat of the unique color always answers correctly, and among two other wizards one answers correctly and another one incorrectly, so wizards score 2 points. There is no strategy that allows them to always score 3 points.

Input

The first line of the input file contains two integers: n and m ($1 \le n \le 10$, $2 \le m \le 4$). The second line of the input file contains m integers: $k_1, k_2, \ldots, k_m$ ($1 \le k_i \le n$, $k_1 + \ldots + k_m = n$).

Output

The first line of the input file must contain p — the maximal possible score the wizards can achieve in the worst case.

The following lines must contain their optimal strategy. For each possible sequence of colors a wizard can see output this sequence followed by a space followed by the color the wizard must write down if he sees such sequence. Do not separate colors in the sequence by spaces. It is guaranteed that there are at most 10 000 possible sequences.

If there are several optimal strategies, you can output any one.

Examples

hats.in	hats.out
3 2	2
1 2	11 2
	12 1
	21 2
	22 1
4 2	4
2 2	112 2
	121 2
	122 1
	211 2
	212 1
	221 1

Problem I. Incomparable Suffixes

Input file:	`incomparable.in`
Output file:	`incomparable.out`
Time limit:	5 seconds
Memory limit:	256 megabytes

Consider a word s. A word y is called *suffix* of s if $s = xy$ for some word x. A word y is called *prefix* of s if $s = yz$ for some word z.

Two words a and b are called *distinguishable* with respect to s if there is a word z such that az is a suffix of s and bz is not, or vice versa, bz if a suffix of s and az is not. For example, "`ab`" and "`b`" are distinguishable with respect to $s =$"`abbab`" because $z =$"`ab`" distinguishes them: "`abab`" is not a suffix of "`abbab`", while "`bab`" is.

Two suffixes x and y of a word s are called *incomparable* if for any non-empty prefix x_1 of x and for any non-empty prefix y_1 of y words x_1 and y_1 are distinguishable with respect to s.

For example, suffixes "`ab`" and "`bab`" of "`abbab`" are incomparable, because all pairs ("`a`", "`b`"), ("`a`", "`ba`"), ("`a`", "`bab`"), ("`ab`", "`b`"), ("`ab`", "`ba`"), ("`ab`", "`bab`"), are distinguishable, but suffixes "`bab`" and "`abbab`" are not (because no word distinguishes "`ba`" from "`abba`"), neither are "`bab`" and "`bbab`", because their prefixes of length one are both "`b`" and therefore clearly not distinguishable.

You are given a word s. Find a set S of its suffixes such that all suffixes in S are pairwise incompatible, and the number of suffixes in S is maximal possible. Among all such sets find the one that has maximal sum of lengths of suffixes it contains.

Input

Input file contains several test cases. Each test case consists of a word s on a line by itself. The word s has length from 1 to 10^5 and consists of lowercase English letters. The total length of all words in the input file doesn't exceed 10^5. The number of test cases in the input file doesn't exceed 1024.

Output

For each test case print n — the maximal number of pairwise incomparable suffixes of s, followed by l — the maximal possible sum of lengths of n pairwise incomparable suffixes of s. The following n lines must contain the suffixes themselves, one on a line, in any order. If there are several optimal solutions, print any one.

Examples

incomparable.in	incomparable.out
abbab aaaaa abacaba	2 6 bbab ab 1 5 aaaaa 3 7 a ba caba

Problem J. Journey Planner

Input file:	`journey.in`
Output file:	`journey.out`
Time limit:	4 seconds
Memory limit:	256 megabytes

Retrozavodsk travel agency "You Go Further" is planning to open a new office in Flatland. It is planning to sell vairous sightseeing journeys to its clients. The traveller would start in a city where the company office is located and follow some roads visiting different cities on its way. The problem is that after recent traffic rules reform in Flatland all of its roads are one-directional. More of that, it is impossible to start from some city a, follow at least one road, and return to city a.

Now "You Go Further" managers are facing a problem: they would like to choose such city for their office that as many cities as possible could be reached from it along the roads. However, they are in hurry, since they are affraid of new reforms Flatland govennent could enforce. So they are not asking for exactly the best city. They would accept any city for their office, such that there is no city that has more than 2 times more cities reachable from it. Help them to find such city.

Input

Input contains several test cases.

The first line of each test case contains n and m — the number of cities and roads, respectively ($1 \le n \le 100\,000$, $0 \le m \le 200\,000$). The following m lines contain two integers each, pair x, y means that there is a road from city x to city y. It is impossible to start from some city a, follow at least one road and return to city a.

The last test case is followed by $n = 0$ which must not be processed. The sum of n in the input file doesn't exceed 100 000. The sum of m in the input file doesn't exceed 200 000.

Output

For each test case print two integers: v and d — the city where the office can be located and the number of cities reachable from v.

Examples

journey.in	journey.out

7 8
1 3
1 4
2 4
2 5
5 6
3 7
4 7
6 7
0
	1 4

Note

In the sample input the optimal solution is city 2, it is possible to reach 5 cities from it. So it is allowed to output any city from which at least 3 cities can be reached, any city from the set $\{1, 2, 5\}$ would be accepted.