Problem A. Approximate Bounding Box

Input file:	approximate.in
Output file:	approximate.out
Time limit:	3 seconds
Memory limit:	256 megabytes

Bounding box problem is stated as follows: given n points (x_i, y_i) on a plane, find the smallest possible rectangle with edges parallel to coordinate axes that contains all the given points. The problem can be easily solved by finding minimal and maximal coordinates of points: $x_L = \min\{x_i\}$, $x_R = \max\{x_i\}$, $y_B = \min\{y_i\}$, $y_T = \max\{y_i\}$, the corresponding rectangle has corners at (x_L, y_B), (x_L, y_T), (x_R, y_B), and (x_R, y_T).

However the problem becomes more difficult if the points are given approximately. In this problem each point can be at any location inside a circle with center at (x_i, y_i) and radius r.

Given approximate locations of points find the maximal possible area of their bounding box.

Input

The input file contains multiple test cases.

The first line of each test case contains two integers: n and r ($2 \le n \le 100\,000$, $1 \le r \le 10^9$).

Each of the following n lines contain two integers x_i and y_i: the coordinates of approximate locations of the points ($-10^9 \le x_i, y_i \le 10^9$).

The sum of n for all test cases in the input file doesn't exceed 100 000. There are at most 10 test cases.

The last test case is followed by two zeroes that should not be processed.

Output

For each test case output one floating point number: the maximal possible area of the bounding box for the points if each of them is located somewhere inside a circle of radius r centered in its approximate location.

Your answer must be within 10^{-9} relative or absolute error.

Examples

approximate.in	approximate.out
4 1 0 0 1 1 1 0 0 1 0 0	9.0

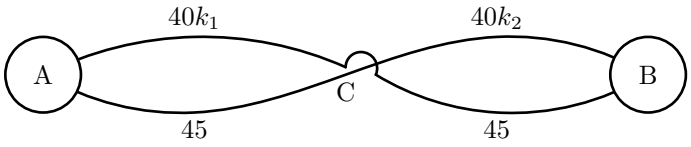
In the given example the bounding box has area of 9 if, for example, points are located at $(0, -1)$, $(1, 2)$, $(2, 0)$ and $(-1, 1)$, correspondingly.

Problem B. Braess's Paradox

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

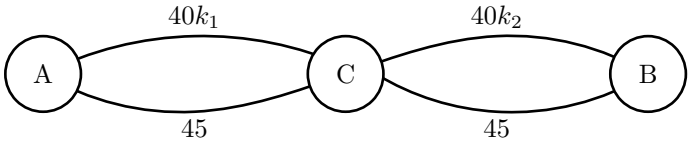
Little known Braess's paradox illustrates the fact that adding more roads and interchanges to the road system can actually make travel time worse for *all* drivers that use corresponding routes. An example shown on the picture below is adapted from Wikipedia. Consider two cities A and B and people driving from city A where they live to city B where they work. There are two roads between the cities that intersect in the middle at a point C, but have no interchange and it is impossible to get from one road another at this point.

The trip by the road one from A to C takes $40k_1$ minutes where k_1 is the fraction of the drivers that use this road. So, for example, if half of the drivers use road one, the trip would take 20 minutes. The trip on the road one from C to B takes 45 minutes regardless of its load. The parts of the road two have the same properties, but other way round. A trip from A to C by the road two takes 45 minutes, and the trip from C to B takes $40k_2$ minutes ($k_1 + k_2 = 1$).



Now it is optimal that half of the drivers use road one and half of the drivers use road two, since in any other situation it would take longer to drive along one of the roads and it would be beneficial for some of the drivers who use a longer route to switch to a shorter route until they become equally long. So all drivers get to their destination in 65 minutes.

If the interchange is built at point C which allows to switch between the roads there, the situation would change in the following way. Now it is always better to drive along the road one from A to C and drive along the road two from C to B, regardless of the road load. Therefore all drivers would choose this route, and now for *all* drivers the trip takes 80 minutes instead of 65.



In this problem we consider a generalized version of the above paradox. Consider two cities A and B connected by two roads that intersect at $n - 1$ points that divide each of the roads into n parts. For the i-th part of the road one the time it takes to travel along it is $a_i k_{i,1} + b_i$ minutes where $k_{i,1}$ is the fraction of the drivers that choose to drive this part by road one ($0 \le k_{i,1} \le 1$). Similarly, the time to drive along the i-th part of the second road is $c_i k_{i,2} + d_i$ ($k_{i,2}$ is the fraction of the drivers that use this part of the road two, $k_{i,1} + k_{i,2} = 1$).

Traffic authorities are planning to build interchanges at some of the road intersections that would allow to switch between the roads at these points. Then the drivers would selfishly choose optimal routes: the drivers one after another consider their current route and switch to a better route if it exists. The process continues until for no drivers it is preferential to change their route. It can be proved that this situation will always be reached in constraints of this problem. Since all drivers are equal, they will all have the

same trip duration from A to B, this time is called the *equilibrium time*. We assume that there are so many drivers that $k_{i,j}$ can be any floating point value from 0 to 1.

You have to check the current situation, the situation if all possible interchanges are built and also find two ways to build zero or more interchanges. The first way must minimize the equilibrium time for the drives, the second one must maximize it.

Input

The first line of the input file contains n ($2 \le n \le 5000$). The following n lines contain four integers each: a_i, b_i, c_i and d_i ($0 \le a_i, b_i, c_i, d_i \le 1000$).

Output

Output four floating point numbers, one on a line:

- equilibrium time if no interchanges are built;
- equilibrium time if interchanges are built at all intersections;
- minimal possible equilibrium time;
- maximal possible equilibrium time.

Your answer must have absolute or relative error at most 10^{-9}.

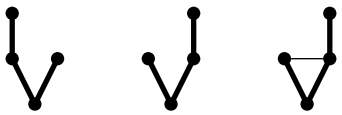
Examples

braess.in	braess.out
2 40 0 0 45 0 45 40 0	65.0 80.0 65.0 80.0

Problem C. Catalonian Forest

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

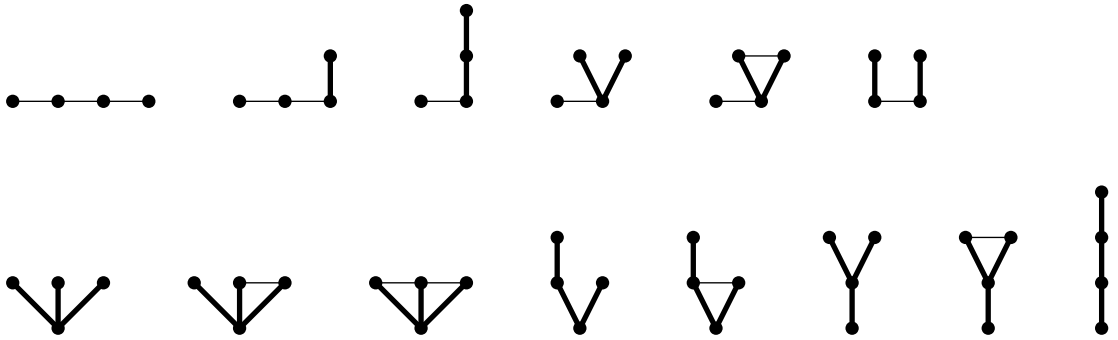
Consider rooted unlabelled trees. For each vertex of the tree let its children be divided to one or more unnamed groups, each child belongs to exactly one group. When drawing such tree, children from the same group are positioned continuously and stroke by a horizontal line. Let us call such tree a *Catalonian tree*. The picture below shows three Catalonian trees with 4 vertices each. The trees grow bottom up. The first two are actually different drawings of the same tree but the last one is different from them, because two root's children are in the same group.



A *Catalonian forest* is the set of one or more Catalonian trees.

Given n find the number of different Catalonian forests with n vertices in all the trees together.

The picture below shows all 14 Catalonian forests with 4 vertices. Roots of the trees of the same forest are connected by a line for clarity. The trees grow bottom up.



Input

The input file contains multiple test cases. Each test case consists of an integer n on a line by itself ($1 \le n \le 60$). The last line of the input file contains 0, it must not be processed.

Output

For each test case output one number — the number of Catalonian forests with n vertices.

Examples

catalonian.in	catalonian.out
1 2 3 4 5 0	1 2 5 14 42

Problem D. Detect Shuffling Method

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

You are working in Laboratory of Artificial Intelligence. Now you are teaching your AI to find difference between two ways of data shuffling. You have decided to use solutions of student teams from Petrozavodsk Training Camps as training data.

Consider a program in Java or C++ that solves some problem from Petrozavodsk Training Camp. Let the whole program be placed into a string s with all characters with ASCII codes less or equal to 32 (space) replaced by spaces. The length of s is between 5 to 20 kilobytes, s is appended with spaces until its length is divisible by 4.

There are two ways to shuffle a string s of length L that you have to distinguish between.

The first way is the following: pick random permutation π of numbers from 1 to L and shuffle characters in s according to π. Let us call such shuffling method "random".

The second way is the following: divide s to blocks of four consecutive characters. Pick random permutation of numbers from 1 to $L/4$ and shuffle blocks according to it. Then for each block pick random permutation of numbers from 1 to 4 and shuffle characters of this block with it. Random permutations for each block are selected independently. After that $\lfloor L/10 \rfloor$ times choose two integers from 1 to L at random and swap characters at corresponding positions. Let us call such shuffling method "block".

You are given a file with n lines, each one is either random or block shuffle of the same string s, exactly half of these lines are produced with random shuffle, the other half are produced with block shuffle. For each line identify whether it was produced with random or block shuffle. You must detect shuffling method correctly for at least 80% of lines in each test case.

Input

The first line of each input file contains n — the number of lines to follow ($10 \le n \le 100$, n is even).

Each of the following lines has the same length L from 5000 to 20000 and contains characters with ASCII codes from 32 to 126. Each of them was produced from the same string s with either random shuffle or block shuffle. Exactly $n/2$ of these lines were produced with random shuffle, another $n/2$ lines were produced with block shuffle. The string s is source code of some submitted solution from Petrozavodsk Training Camp written either in C++ or in Java with all characters with codes less then 32 replaced with spaces.

Sample input corresponds to "Hello World" program in C++, it violates the restriction that all lines have length between 5000 and 20000, any correctly formatted answer to this input would be accepted (note: judges' program correctly solves this input as well, however).

Output

For each line print "`random`" if this line represents randomly shuffled s, or "`block`" if the line represents s shuffled with block shuffle.

Sample output in the problem statement is the correct answer to the sample input.

Examples

detect.in

```
10
 c<( {  ldulm ca};   p   itno, loerWdd"!oi t">fsntrielH ) % ", si#inn)("
 ! dtn"or   #iis np "Woo ,%  )lfn) {clr m>t(ud;  tl<idsec la,    H} (i "ne"i
l s,tdio( n)p l     {     };ud ,tirnlroWc <slH"etn # o ema %inic " >f"(i"!)d
i e nirr ai )pi(; d cnt ,l o s"n  {"W,l>!"  m Htt  <o  }("uil#s %)nd c fodle
H   ";niorWloidtldue"!)d      { n ()le" >  plm ,t,rn aio s"ini#cs <c(%ft   }
lWcu#  eniaf(eo dn   !>) "dtisd oni %n tr l"  lt,i)mc H(; <} p" s,"r  {  ilo
idtolrWa> n  <csm o " s,ledur it itn%"f(i#cnlH el ,o    )"d!  "{) (n ;}  pi
lttdnri an" l#  "ms    o p>{i   i ct r %o)osi ,nfHni!d  "u;l (l cd" W}e(<),e
iencsc< !d ""ileW{ olo rldu#  };  ) n  (s ,"tin "(f%   ptHrntoidl),  ami  >
nnod)plo{c #"  n%(elcu ", t d t,s") ( iims nWl ;iiadt!  r  rf }i   H> e "o<l
```

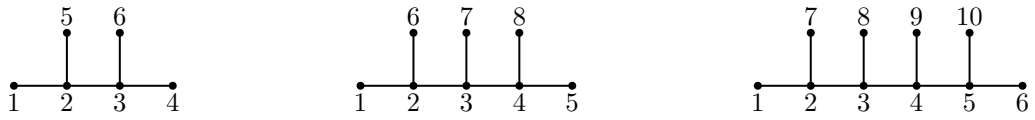
detect.out

```
block
random
block
random
block
random
block
random
block
random
```

Problem E. Embedding Caterpillars

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

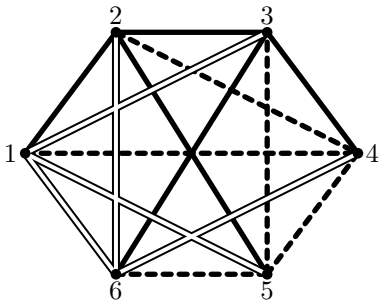
Caterpillar of order n is a graph with $2n$ vertices arranged in the following way: a path of length n has another vertex attached to every internal vertex. The picture below shows caterpillars of order 3, 4 and 5.



Embedding of graph G to graph H is mapping $\varphi : VG \to VH$ of vertices of G to vertices of H such that φ is one to one and if there is an edge uv in G then there is an edge $\varphi(u)\varphi(v)$ in H.

Simultaneous embedding of several graphs $G_1, G_2, \ldots, G_k$ to H is a collection of embeddings $\varphi_1, \varphi_2, \ldots, \varphi_k$ such that for each edge uv of H there is at most one i such that there is an edge xy in G_i such that $uv = \varphi_i(x)\varphi_i(y)$.

The picture below shows simultaneous embedding of three caterpillars of order 3 to a complete graph K_6. You have to generalize this construction and find simultaneous embedding of three caterpillars of order n to K_{2n}.



Input

There are multiple tests cases in the input file. Each test case consists of a single integer n on a line by itself ($3 \le n \le 100$). The last test case is followed by $n = 0$, it must not be processed. There are at most 10 test cases in each input file.

Output

For each test case output three lines. Let vertices of each caterpillar be numbered from 1 to $2n$ in the following way: vertices from 1 to $n + 1$ are arranged along the path, vertex $n + 2$ is connected to vertex 2, vertex $n + 3$ is connected to vertex 3, etc. For each caterpillar output $\varphi_i(u)$ for all u from 1 to $2n$.

Examples

embedding.in	embedding.out
3 0	1 2 3 4 5 6 1 4 5 3 2 6 2 6 1 3 4 5

Problem F. Funny Card Game

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

Andrew and his k friends are playing a funny card game. They have a deck of n cards, each one contains a single integer a_i. Andrew is a dealer. His friends are sitting in a circle around him and he deals cards to them.

Andrew chooses one of his friends and starts dealing cards to him, one after another. After each card a player can either say "stop", or say "more". If the player says "more" he is dealt another card. After he says "stop", he gets no more cards and his score is equal to the maximal number of times some value occurs among his cards. For example, if the player is dealt cards with values 2, 3, 4, 3, 2, 1, 2 and 5, his score is 3, because 2 occurs 3 times among his cards and no other value has more occurrences.

Then the next player who has no cards yet is chosen and Andrew deals cards to him in the same way. The game continues until all but one friends have their cards. The last player gets all the remaining cards.

Andrew's friends has seen the order in which the cards are arranged in the deck. Now they want to choose such strategy that the sum of their scores was maximal possible. Also they want each player to be dealt at least one card.

Help them to develop their strategy: for each player from 1 to $k-1$ find the card that he must say "stop" after. The last player would receive the rest of the cards.

Input

Input file contains multiple test cases. The first line of each test case contains two integers: n and k ($2 \le k \le n \le 10^5$, $k \le 100$). The second line contains n integers: numbers on the cards in order they are arranged in the deck. Numbers on the cards do not exceed 10^9 by their absolute values.

The last test case is followed by a line containing two zeroes that should not be processed.

Output

For each test case output two lines. The first line must contain the maximal possible sum of points that the players can achieve. The second line must contain instructions for the players: $k-1$ integers b_i. The number b_i must be the number of the card in the original deck after which the i-th player must say "stop". Each player, including the last one, must receive at least one card. The cards are numbered from 1 to n in order they are arranged in the deck.

Examples

funny.in	funny.out
10 2 2 3 2 3 2 1 2 1 2 1 0 0	6 5

In the given example the first player would get cards 2, 3, 2, 3 and 2, his score is 3. Similarly, the second player gets cards 1, 2, 1, 2 and 1, his score is also 3. The sum of players' scores is 6 and is maximal possible.

Problem G. Gold Mines

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

There are n gold mines in Flatland, the i-th gold mine is located at a point with coordinates (x_i, y_i). There are three companies *Zero*, *One* and *Two* working on the mines, for each mine the value of its owner $c_i \in \{0, 1, 2\}$ is known.

Recently the parliament of Flatland has adopted the new law which allows the company to work on a gold mine only if the mine is located inside its territory. The territory of each company must be a non-degenerate rectangle with sides parallel to coordinate axes. Territories of different companies must not have a common part of non-zero area (although they may have common borders).

However, it turned out that all three companies belong to the same holding *Three*, so the managers of the companies have decided to choose territory for each company to maximize the total number of mines that the companies would be able to work at. If the mine is on the border of two or three territories, any of their owners can work on this mine. Help the managers of the companies to set up the territories.

Input

The input file contains multiple test cases. The first line of each test case contains n ($1 \le n \le 50\,000$). The following n lines contain three integers each: x_i, y_i and c_i ($-10^9 \le x_i, y_i \le 10^9$, $c_i \in \{0, 1, 2\}$).

The last test case is followed by $n = 0$, it must not be processed. The sum of n for all test cases in one input file doesn't exceed $50\,000$.

Output

Output four lines for each test case. The first of these lines must contain k — the maximal possible number of mines that can be worked on after territories of the companies are set up. The following three lines must describe territories of the companies. Each territory must be described by four integers: x_1, y_1, x_2 and y_2 ($-2 \cdot 10^9 \le x_1 < x_2 \le 2 \cdot 10^9$, $-2 \cdot 10^9 \le y_1 < y_2 \le 2 \cdot 10^9$) — the coordinates of two opposite corners of the rectangle. A mine inside the first rectangle can be worked if $c_i = 0$, a mine inside the second rectangle can be worked if $c_i = 1$, a mine inside the third rectangle can be worked if $c_i = 2$.

Examples

gold.in	gold.out
12 0 0 0 1 0 0 2 0 1 3 0 2 0 1 1 1 1 1 2 1 2 3 1 1 0 2 1 1 2 0 2 2 2 3 2 2 0	10 -1 -1 2 0 -1 0 2 3 2 -1 4 3

Problem H. Huffman Codes

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

Huffman codes is a famous family of optimal prefix codes. Let us briefly remind information about them.

Consider alphabet Σ of n characters. Binary prefix code for this alphabet is a mapping $c : \Sigma \to \{0,1\}^*$ which assigns a word of 0-s and 1-s to each character of Σ such that for any $u \neq v$ the code word $c(u)$ is not prefix of the code word $c(v)$. Let the length of the code word $c(a)$ be denoted as $|c(a)|$. Given some text t with $p(a)$ occurrences of character a the prefix code is *optimal* for t if the value $\sum_{a \in \Sigma} |c(a)|p(a)$ is minimal possible.

Huffman algorithm produces optimal prefix code for the given text. It proceeds as follows. If $n = 2$ then clearly the optimal prefix code is 0 for one character and 1 for another. Let $n > 2$. Consider two characters x and y with smallest values of $p(x)$ and $p(y)$, respectively. Unite these characters to a new character z with $p(z) = p(x) + p(y)$. Build optimal prefix code for the resulting $n-1$-character alphabet. Now append 0 to code word for z to get code word for x and append 1 to code word for z to get code word for y.

For example, consider text "abacaba". We have $p(a) = 4$, $p(b) = 2$ and $p(c) = 1$. Unite b and c to one character d to get $p(a) = 4$ and $p(d) = 3$. Now we have $c(a) = 0$ and $c(d) = 1$. Getting back to b and c we have $c(b) = 10$ and $c(c) = 11$. Clearly, Huffman code is not unique, because we are free to swap x and y at any step of the algorithm above to get another Huffman code. In the given example another possible Huffman code is $c(a) = 0$, $c(b) = 11$, $c(c) = 10$.

You are given integers u_i and v_i for i from 1 to n. Find out whether there exists some text t and some Huffman code for it such that the code word for the i-th character contains exactly u_i zeroes and v_i ones. If there exists such text and code you have to print the corresponding Huffman code.

Input

The input file contains multiple test cases.

The first line of each test case contains n ($2 \le n \le 100$). The following n lines contain two integers each: u_i and v_i ($0 \le u_i, v_i \le 100$, $u_i + v_i > 0$).

The last test case is followed by a line containing zero, it must not be processed.

Output

For each test case first output a line containing "`Yes`" if there exists corresponding text t or "`No`" if there exists none. If the answer is positive, the following n lines must contain code words, one on a line. Code words must consist of 0-s and 1-s, the i-th of the printed code words must contain u_i zeroes and v_i ones, the printed code must be Huffman code for some text over n-character alphabet.

If there are several solutions, print any one.

Examples

huffman.in	huffman.out
3 1 0 0 2 1 1 7 1 1 2 1 3 0 1 1 1 2 1 3 0 4 2 1 1 1 1 0	Yes 0 11 10 Yes 10 001 000 01 110 1110 1111 No

Problem I. Intelligent Tourist

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

Johnny is a tourist. But he is also a computer science student in the University of Flatland. Johnny has to pass n exams soon, so he need to prepare for them, but he has to go to m hiking expeditions.

Fortunately the expeditions are arranged in such way that no exam takes place during any expedition. However, preparing for exams requires internet access and great concentration, so it is impossible to study during the expedition. Johnny cannot skip an expedition.

Johnny has numbered all the ongoing days starting from 1. For each exam Johnny knows its day d_i and the number of days p_i the he needs to study to prepare for the exam. For each expedition Johnny knows its starting day s_j and its final day t_j. It is impossible to prepare for exams during any day between s_j and t_j, inclusive. No exam date is during any expedition. No two expeditions have a common day. Johnny's memory is very good, so he can prepare for any exam during any day, but he only prepares to one exam during one particular day. If Johnny doesn't prepare to an exam he doesn't try to pass it. Johnny doesn't prepare for exams during days of exams that he tries to pass. All exams are on different days.

Help Johnny to find the maximal number of exams that he can prepare for.

Input

The input file contains multiple test cases.

The first line of each test case contains n ($1 \le n \le 10^5$). The following n lines contain two integers each: d_i and p_i ($0 \le p_i \le 10^9$, $1 \le d_i \le 10^{18}$). The following line contains m ($0 \le m \le 10^5$). The following m lines contain two integers each: s_i and t_i ($1 \le s_i \le t_i \le 10^{18}$).

The last test case is followed by a line containing zero, it must not be processed.

The sum of values of n in all test cases in one input file doesn't exceed 10^5. The sum of values of m in all test cases in one input file doesn't exceed 10^5.

Output

For each test case output two lines. The first line must contain k — the maximal possible number of exams that Johnny can prepare for. The second line must contain k integers: the numbers of these exams. The exams are numbered from 1 to n in order they are described in the input file.

If there are several optimal solutions, print any of them.

Examples

intelligent.in	intelligent.out
3 4 2 10 3 13 4 1 5 8 0	2 1 3

In the given example Johnny can, for example, prepare for exam 1 during days 1 and 3, and prepare for

exam 3 during days 2, 9, 10 and 11. Days 4 and 13 are used for exams, days from 5 to 8 are used for the expedition.

Problem J. Judges Problem

Input file:	judges.in
Output file:	judges.out
Time limit:	3 seconds
Memory limit:	256 megabytes

Judges of Flatland University Contest have prepared an excellent problem set. But the morning before the contest they found out that computer virus has destroyed input files to one of the problems. Now they urgently need to restore the files. They decided to ask for your help. Given answer files for "Prefix Cover" problem you have to restore the input files.

The problem is stated as follows. Consider array $A = \langle a_1, a_2, \ldots, a_n \rangle$ of integers. Array $B = \langle b_1, b_2, \ldots, b_m \rangle$ is said to *cover* array A if it is possible for each i from 1 to n to find subarray of A equal to B containing a_i. For example, $A = \langle 1, 2, 1, 2, 1, 1, 2, 1 \rangle$ is covered by $B = \langle 1, 2, 1 \rangle$. Minimal m such that A can be covered by an array of length m is called *mincover* of A and denoted as $mc(A)$. In the example above $mc(\langle 1, 2, 1, 2, 1, 1, 2, 1 \rangle) = 3$.

Given A its prefix cover array $P = \langle p_1, p_2, \ldots, p_n \rangle$ is an array of mincovers of A prefixes, that is $p_i = mc(\langle a_1, a_2, \ldots, a_i \rangle)$. Prefix cover array for the array in the above example is $P = \langle 1, 2, 3, 2, 3, 6, 7, 3 \rangle$.

The judges wanted to ask teams to create prefix cover array for the given array. Now you have to solve the inverse problem: given an array P find some array A for which P is the prefix cover array.

Input

The input file contains multiple test cases.

The first line of each test case contains n ($1 \le n \le 5 \cdot 10^5$). The second line contains n integers p_i ($1 \le p_i \le i$).

The last test case is followed by a line containing zero, it must not be processed.

The sum of values of n in all test cases in one input file doesn't exceed $5 \cdot 10^5$.

Output

For each test case output "Yes" if there exists such array A that P is its prefix cover array. In such case the following line must contain n integers ranging from 1 to n — elements of A. If there is no such array, output "No".

Examples

judges.in	judges.out
8 1 2 3 2 3 6 7 3 5 1 1 1 1 1 3 1 2 2 7 1 2 3 4 5 6 7 0	Yes 1 2 1 2 1 1 2 1 Yes 1 1 1 1 1 No Yes 1 2 3 4 5 6 7