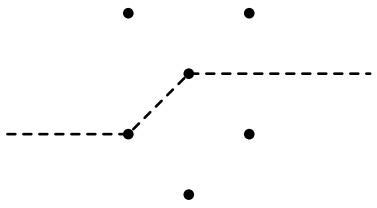
Problem A. Kingdom Division

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

Father King has died and left the kingdom to his three sons. They decided to divide the kingdom according to their profession. The older son is specializing in geology, he would like to get the northern lands rich in oil and gas. The middle son is specializing in tourism, he would like to get the southern lands with their nice beaches and the warm sea. And the younger son is military officer, he would like to take the border between the two parts of the older sons and defend them from each other.

There are n cities in the kingdom, the heirs decided that they would to separate the kingdom by a border that would have a form of an infinite polyline. Infinite parts of the polyline will be parallel to West-East direction, and the segments that form the polyline will have vertices in the cities. Each meridian (line parallel to North-South direction) will intersect the border in exactly one point.

The cities on the borderline will be given to the younger son, the cities to the North of the borderline will be given to the older son, the cities to the south of the borderline will be given to the middle son. The division must be organized in such a way that the difference between the minimal and the maximal number of the cities a son gets had been minimal possible. Help the sons to perform the division.



Input

The first line of the input file contains n — the number of the cities ($1 \le n \le 100$). The following n lines contain the coordinates of the cities — each line contains two integer numbers x_i and y_i ($-10^4 \le x_i, y_i \le 10^4$, Ox axis points eastwards, Oy axis points northwards).

Output

The first line of the output file must contain k — the number of cities on the borderline ($1 \le k \le n$). The second line must contain k integer numbers — the numbers of the cities along the borderline, from West to East. The cities are numbered from 1 to n as they are given in the input file. Adjacent segments of the borderline can be parallel, but no unlisted city must be on the borderline.

Example

division.in	division.out
6 0 1 1 0 2 1 1 2 2 3 0 3	2 1 4

Problem B. Financial Software

Input file:	`financial.in`
Output file:	`financial.out`
Time limit:	6 seconds
Memory limit:	256 megabytes

Jim is writing financial software. He specializes in various financial instruments. His latest project is history analysis of arbitrage with highly volatile stock.

The idea of the analysis is the following: given a quote of the stock for n consecutive time periods, select a subsequence of the quotes such that any two selected quotes that are adjacent in the selected subsequence are at least k points apart. For example, if the quotes are $1014, 1024, 1034, 1045, 1030, 998$ and $k = 15$, a valid subsequence is $1014, 1034, 998$.

The selected subsequence must be as long as possible. In the example above the selected subsequence is not optimal, selecting $1014, 1045, 1030, 998$ is better. Given a sequence of quotes, find the longest valid subsequence that can be selected.

Input

The first line of the input file contains n ($1 \le n \le 100\,000$) — the number of quotes, and k ($1 \le k \le 10^9$). The second line contains n integer numbers — the given sequence of quotes (all quotes are between 1 and 10^9, inclusive).

Output

The first line of the output file must contain l — the length of an optimal subsequence. The second line must contain l integer numbers — the elements of any such subsequence.

Example

financial.in	financial.out
6 15 1014 1024 1034 1045 1030 998	4 1014 1045 1030 998

Problem C. Late Again

Input file:	`lateagain.in`
Output file:	`lateagain.out`
Time limit:	12 seconds
Memory limit:	256 megabytes

Serge is late again for his informatics lesson. His teacher Mr Tinsmith says that it is because he doesn't choose the optimal path from his house to school, and always goes in some weird way. As a penalty for him being late again, the teacher asked Serge to draw a map of the city and find the shortest path from his house to the school. And get sure that his usual path is way far from optimal.

Serge has represented the map of the city as an undirected graph, with junctions as vertices and streets as edges. He put a time needed to go along each street next to the corresponding edge, and marked his usual path to school on the map. Of course, Serge sees that his path is not optimal. But he wants to prove the opposite to the teacher.

After some thought Serge decided that he would change times needed to go along streets in such way that his path became optimal. Of course, the teacher would suspect him cheating if the times would differ too much from the original, or become less than one minute. So he would like to change the times in such way, that maximal change was as small as possible, and no time became less than 1 minute.

Help Serge to defend his path to school

Input

The first line of the input file contains n and m — the number of junctions and the number of streets, respectively ($2 \le n \le 1000$, $1 \le m \le 20\,000$). The following m lines describe streets: each street is described with three integer numbers: the numbers of junctions it connects and the time in minutes needed to go along it (all times are integers from 1 to 10^4, inclusive). All streets are bidirectional and can be walked along in either direction.

The next line contains l — the number of streets Serge passes while going to school. The next line contains l lines — the list of these streets in the order Serge passes them. Serge doesn't visit the same junction (and therefore the same street) more than once. His house is at the junction 1, his school is at the junction n.

Output

The first line of the output file must contain v — the minimal value such that after changing passing time of each street by at most v Serge can make his path be one of the shortest possible paths to the school. The second line must contain m real numbers — the new passing times of the streets. The numbers must be accurate up to 10^{-5}.

Example

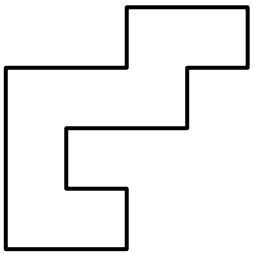
lateagain.in	lateagain.out
4 4 1 2 1 2 4 2 1 3 1 3 4 3 2 3 4	0.333333333 1.333333333 2.333333333 1 2.666666667

Problem D. Night Watch

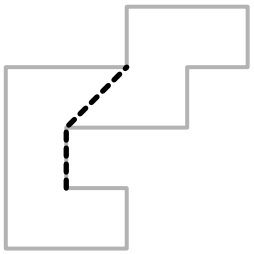
Input file:	`nightwatch.in`
Output file:	`nightwatch.out`
Time limit:	6 seconds
Memory limit:	256 megabytes

Vlad is a night watch in the museum. Every hour he has to make a walk around the museum to see that everything is intact. Since Vlad is quite old, he would like these inspection walks be as short as possible.

Consider the plan of the museum drawn at the sheet of paper so that the walls of the museum are parallel to the sides of the sheet. The museum has the form of a simple monotone rectangular polygon: all of its angles are right and any line $y = c$ intersects the museum with either a segment or an empty set. An example of museum plan is shown on the picture below.



Vlad can start his walk at any point, walk along any path, and must return to the starting point. Help him to find the shortest path around the museum such that any point inside the museum is visible from some point at the path. The point A is visible from the point B if they can be connected by a segment that is completely inside the museum. The optimal path in the museum above is shown on the picture below (Vlad must start at any point of the path and go along it there and back).



Help Vlad find the shortest inspection path.

Input

The first line of the input file contains n — the number of vertices of the polyline that the walls of the museum form on the plan ($4 \le n \le 100$). The following n lines contain coordinates of the vertices in the order of counterclockwise traversal. Coordinates do not exceed 10^4 by their absolute values.

Output

Output one real number — the length of the shortest inspection path. Your output must be accurate up to 10^{-5}.

Example

nightwatch.in	nightwatch.out
12 1 0 3 0 3 1 2 1 2 2 4 2 4 3 5 3 5 4 3 4 3 3 1 3	4.8284271247461901
4 0 0 1 0 1 1 0 1	0.0

In the second example the watch can stay at one place and still observe the whole museum.

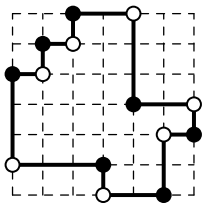
Problem E. Convex Permutominoes

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

A *polyomino* is the connected set of unit squares on a square grid. A polyomino is called *convex* if any row of the polyomino and any column of the polyomino is connected. For example, the polyomino on the picture (a) below is convex, but the polyomino on the picture (b) is not.

(a) (b)

A polyomino is called a *permutomino* if it has $2n$ vertices, all of its vertices have coordinates from 1 to n, and there are no two vertical edges of its boundary that have equal x coordinate, nor two horizontal edges that have equal y coordinate. The picture below shows an example of permutomino with 14 vertices. Note that this permutomino is convex.



Given n find the number of convex permutominoes with $2n$ vertices. Two permutominoes are considered equal if one can be transformed to another by translation. Rotations and reflections are not allowed.

Input

The input file contains one integer number n ($2 \le n \le 30$).

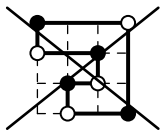
Output

Output one number — the number of convex permutominoes with $2n$ vertices.

Example

permutominoes.in	permutominoes.out
3	4
4	18

Note that there are non-convex permutominoes with 8 and more vertices, an example of such permutomino is shown at the picture on the right. You must output the number of convex permutominoes.



Problem F. Substring Search

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

Andrew has a little sister Ann. She likes writing messages to her boyfriend George. She wants nobody be able to read her messages, so she encrypts them using substitution cipher. Substitution cipher replaces each character in the message with some other character so that equal characters are replaced with equal characters, and different characters are replaced with different ones.

For example, in substitution cipher $e \to a$, $l \to b$, $o \to w$, $v \to c$ the word "*love*" is encrypted as "*bwca*".

Andrew has intercepted one of Ann's encrypted messages t and would like to check whether it contains some text p. In particular, he would like to find all such positions i that there is a substitution cipher such that $t[i..i + |p| - 1]$ is the encrypted version of p. Let us call such position a potential occurrence of p in t. Help Andrew to find all potential occurrences of p in t.

Input

The first line of the input file contains t. The second line of the input file contains p. Each string consists of characters with ASCII codes from 33 to 126. The length of p doesn't exceed the length of t. The length of t doesn't exceed 200 000. Both strings are not empty.

Output

The first line of the output file must contain k — the number of potential occurrences of p in t. The second line must contain k integer numbers — the positions of the potential occurrences. Positions are numbered starting from 1. Positions must be listed in the increasing order.

Example

substring.in	substring.out
abacabadabacaba aba	7 1 3 5 7 9 11 13
abacabadabacaba love	0

Problem G. Transportation Problem

Input file:	`transport.in`
Output file:	`transport.out`
Time limit:	9 seconds
Memory limit:	256 megabytes

Flat Oil company has recently won a tender for building an oil transportation system between a oil production points and b main oil consumers of Flatland. As a result a set E of m oil pipelines was built. All pipelines are designed to be unidirectional, so oil can only be transported in one (predefined) direction along the pipeline. In order to simplify the process of planning oil transportation (and since funding through the tender was quite impressive) each pipeline has daily capacity c equal to the daily production of oil in Flatland. The system is planned in such a way that it is possible to transport oil from any point to any other point (no matter, production or consumption).

However, when time came for planning transportation, it turned out that due to the extremely large size of the network, the problem is still quite difficult. So they needed external help. Help the company to create the oil transportation plan.

For each oil production point u its daily production p_u is known. For each oil consumption point u its daily consumption q_u is known. The sum of all p_u is equal to the sum of all q_u and is equal to the capacity of each pipeline c. Any production or consumption point can also be a transit point for oil transportation.

Oil transportation plan assigns each pipeline uv a *flow* $f(uv)$, so that the following conditions are satisfied:

- for each pipeline uv: $0 \le f(uv) \le c$;
- if u is a production point, $\sum\limits_{uv \in E} f(uv) - \sum\limits_{vu \in E} f(vu) = p_u$;
- if u is a consumption point, $\sum\limits_{vu \in E} f(vu) - \sum\limits_{uv \in E} f(uv) = q_u$.

Given the plan of the network, find the oil transportation plan for it.

Input

The first line of the input file contains three integer numbers: a, b and m ($1 \le a, b \le 50\,000$, $1 \le m \le 100\,000$). The second line contains a integer numbers: p_i ($1 \le p_i \le 10^4$). The third line contains b integer numbers: q_i ($1 \le q_i \le 10^4$). The sum of all p_i is equal to the sum of all q_i. The following m lines describe pipelines, each line contains two integer numbers: s and t, ranging from $-a$ to b except 0 — the number of the point the pipeline starts at and the number of the point the pipeline ends at. The production points are numbered from -1 to $-a$ in the decreasing order in order their production values are given in the second line. The consumption points are numbered from 1 to b according to the order their consumption values are given in the third line.

Output

If a transportation plan exists, output "`YES`" at the first line of the output file. The second line must contain m integer numbers: the flow in the pipelines in order they are described in the input file.

If the transportation plan doesn't exist, output "`NO`".

Example

transport.in	transport.out
2 2 6 5 3 4 4 -1 1 -2 1 -2 2 1 2 2 -1 2 -2	YES 5 0 3 1 0 0

Problem H. Truth is in the...

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

The well known logical paradox about self-referring is the statement "*This statement is false.*". Trying to decide whether the statement is true or false, you never get a satisfying result. Consider the sequence of statements each of which has one of the following forms: "*The next statement is true.*", "*The next statement is false.*", "*There are n true statements.*".

You have to determine if you can decide for each statement whether it is true or false, so that all true statements were indeed true, and all false statements were indeed false.

Input

The first line of the input file contains m — the number of statements ($1 \le m \le 100\,000$). Each of the following m lines contains one of the following:

- "`+`" — means the statement "*The next statement is true.*";
- "`-`" — means the statement "*The next statement is false.*";
- "`$` n" — means the statement "*There are n true statements.*" ($0 \le n \le 100\,000$). This statement is true if there are exactly n true statements in the sequence.

The next statement after the last statement is the first statement.

Output

If there is no way to decide for each statement whether it is true or false to get a consistent system, print "`inconsistent`" at the first line of the output file. In the other case print "`consistent`" at the first line of the output file. The second line must contain m characters, the i-th character must be "`t`" if the i-th statement is true, and "`f`" if it is false. If there are several consistent assignments, output any one.

Example

truth.in	truth.out
3 + + $ 3	consistent ttt
3 + - $ 3	consistent ttf
1 -	inconsistent

Problem I. Video on Demand

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

Josh owns a company that provides video on demand service in the internet. The user selects video and can watch it in his browser. However, many users have small bandwidth, so they cannot watch video in the real time. Andrew is one such user.

Andrew's bandwidth allows him to download d bytes per second. However, the bitrate of the video that he would like to watch is b bytes per second. The video player that Andrew will use to watch the video has the buffering option that allows to relax the problems with bandwidth lack. If the buffer size is set to s, the player acts in the following manner.

First it downloads s bytes of video material. After that it starts to play video, downloading new portions of video to buffer in background. When the buffer is exhausted, the playback stops until the buffer is full again. For sake of simplicity the time is considered continuous, and so is information in this problem.

If there is not enough material to fill the buffer, the player downloads all the material that is remaining.

Andrew would like to watch a program that lasts for t seconds. He is aware that there will be pauses in the playback, but he would like each pause, including the one before the playback starts (initial buffering), not to exceed p seconds. He would like to set such buffer size that the number of pauses was minimal possible. Help him.

Input

The input file contains four integer numbers: d, b, t and p ($1024 \le d \le 1073741824$, $d < b \le 1073741824$, $1 \le t \le 14400$, $1 \le p \le 3600$).

Output

Output one integer number — the size of the buffer that Andrew must set in order for the playback had the minimal number of pauses, and each pause was at most p seconds. If there are several such sizes, output the smallest possible.

Example

video.in	video.out
1024 2048 100 20	20480
1024 2048 90 20	18432
1024 1025 1 1	1

In the first example there will be five pauses, each 20 seconds long. After each pause there will be 20 seconds of playback.

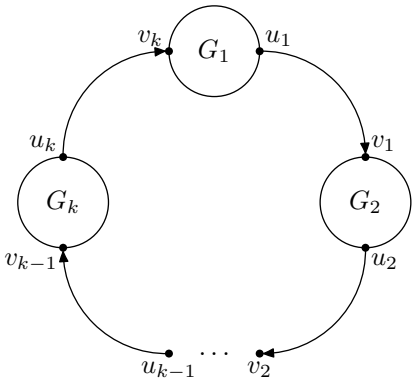
The third example demonstrates that the buffer can be smaller than the number of bytes needed to show one second. In our continuous model of information even 1 byte buffer is enough to make sure there is only one pause (before showing the video).

Problem J. Wheels within Wheels

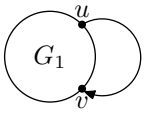
Input file:	`wheels.in`
Output file:	`wheels.out`
Time limit:	5 seconds
Memory limit:	256 megabytes

Some results in Computer Science, such as topological sorting, Dijkstra algorithm or maximal flow algorithm are well known and taught in universities all over the world. Other ideas are sometimes forgotten, and kept only on library shelves. When a person reading an article finds a reference to such results, even by a famous scientist, it is often difficult to keep a surprise in. This problem is dedicated to one such article by Donald E. Knuth, called "Wheels within Wheels".

You are given a directed strongly connected graph G. Knuth proved that any such graph can be represented as a cycle with strongly connected subgraphs $G_1, G_2, \ldots, G_k$, as vertices:



The cycle can have only one edge, as shown on the following figure.



Each of the graphs G_i that contains at least one edge can in turn be decomposed in same way, and so on until each of the components consists of one vertex and no edges.

Given a graph, you must find its wheels within wheels decomposition.

Input

The first line of the input file contains two integer numbers: n and m — the number of vertices and edges in the graph, respectively ($1 \le n \le 200$, $1 \le m \le 300$). The following m lines describe edges, each edge is described with its source vertex and its destination vertex. The graph can contain loops and opposite edges, but it doesn't contain parallel edges. The graph is strongly connected.

Output

The graph is described in the following way. If it contains one vertex and no edges, it is described as "`vertex` i" where i is the number of the vertex. In the other case it is described in the following way: "`wheel` k" starts the description, where k is the number of components along the wheel cycle, k descriptions of its parts follow. After each part description print "`edge` e" that must refer to the edge that goes from this part to the next part.

Each vertex and each edge of the original graph must be listed exactly once in its decomposition.

See examples for further clarification. Separate output by spaces and/or line feeds. Any number of spaces and/or line feeds is equivalent to one space.

Example

wheels.in	wheels.out

7 9
1 6
6 2
2 1
5 7
2 3
3 5
4 3
5 4
7 6
	wheel 3
	  wheel 3
	  vertex 1
	  edge 1
	  vertex 6
	  edge 2
	  vertex 2
	  edge 3
	  edge 5
	  wheel 3
	  vertex 3
	  edge 6
	  vertex 5
	  edge 8
	  vertex 4
	  edge 7
	  edge 4
	  vertex 7
	  edge 9

3 3
1 2
2 3
3 1
	wheel 3
	  vertex 1
	  edge 1
	  vertex 2
	  edge 2
	  vertex 3
	  edge 3

4 5
1 2
2 3
3 4
4 1
1 3
	wheel 1
	  wheel 4
	  vertex 1
	  edge 1
	  vertex 2
	  edge 2
	  vertex 3
	  edge 3
	  vertex 4
	  edge 4
	  edge 5

Output in examples is indented for ease of reading. You need not format your output with indentations.