Problem A. Asymmetric Art

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

I'd tear
like a wolf
at bureaucracy.
For mandates
my respect's but the slightest.
To the devil himself
I'd chuck
without mercy
every red-taped paper,
But this ...
I pull out
of my wide trouser-pockets
duplicate
of a priceless cargo.
You now:
read this
and envy,
I'm a citizen
of the Soviet Socialist Union!

Inspired by Black Square and barcodes, a young aspiring artist wants to create his first masterpiece. The masterpiece will consist of n vertical stripes, each colored either black or white. However, not every such picture looks great, so the artist wants no three black stripes to be *quasi-symmetric*: black stripes a, b and c ($a < b < c$) are quasi-symmetric when $2(b - a) = c - b$. This should never happen.

What is the maximum number of black stripes in a masterpiece with no three quasi-symmetric black stripes?

Note that we don't allow any three black stripes to be quasi-symmetric. For example, for $n = 7$ coloring stripes 1, 2, 3 and 7 black is not allowed because the triple 1, 3, 7 is quasi-symmetric.

Input

The only line of the input file contains just one integer n, $1 \leq n \leq 70$.

Output

On the first line of output, print the maximum number m of black stripes in the masterpiece. One the second line, print m numbers of stripes to be colored black. All numbers in this line must be different integers beween 1 and n, inclusive, written *in increasing order*. In case multiple solutions are possible that maximize the number of black stripes, output any.

Examples

art.in	art.out

`10`	`7`
	`1 2 3 6 8 9 10`

Problem B. Lots of Combinations

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

The infuriated princess hung herself on her own plait, cos he said how many grains were in the bag and how many drops in the sea and how many stars in the sky. Here's to cybernetics!

A *binomial coefficient* $\binom{n}{k}$ is one of the basic concepts of combinatorics. It can be defined as the number of ways to choose k items out of n, or, alternatively, by the formula

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

where ! stands for factorial:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$$

while $0! = 1$.

For given n and k, you need to find the last 10 digits of $\binom{n}{k}$.

Input

The only line of the input file contains two integers n and k, $1 \le n \le 10^{18}$, $0 \le k \le n$.

Output

In case the sought binomial coefficient contains 10 digits or less, print it to the output file. In case it contains more than 10 digits, print three dots, then its last 10 digits, without spaces.

Examples

combi.in	combi.out
`6 2`	`15`
`100 10`	`...0309456440`

Problem C. Curiosity

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

Comrades! I know you're wondering, is there life on Mars? From down here stars look tiny, but if you look through a telescope, you can see two, three, four, or best of all, even five stars. If there is or isn't life on Mars, science hasn't discovered. We're not in the know.

A martian rover is checking the surface of Mars for radioactivity. For simplicity, let's assume it's assigned to check a square region separated into 6×6 unit squares, with each unit square being either radioactive or not. The rover does the following: first, it sends the radioactivity of the unit square where it has landed, with "`1`" denoting a radioactive square and "`0`" denoting a non-radioactive square. Then, it moves randomly into one of the adjacent unit squares (there are four adjacent unit squares for most unit squares, three for unit squares on the boundary of the region, and two for the corner unit squares), picking each with the same probability, and sends the direction of its movement ("`L`" for left, "`R`" for right, "`U`" for up, and "`D`" for down). Then it sends the radioactivity of the unit square he's arrived to (either "`0`" or "`1`"). Then he makes another random move, and so on.

All in all, the transmission from the rover that has made n steps is a string of length $2n + 1$ and looks like `0L1U1R0D0L1L1U0`.

Unfortunately, the transmission comes to the Earth damaged — some of its characters are unrecognizable and thus replaced by question marks ("`?`"), and we also don't know where in the region the rover has landed (but the rover itself knows, so he knows not to step outside the 6×6 field).

Your task is to recover the radioactivity map of the region given this damaged transmission.

The testcases for this problem were generated randomly, and are available for you to download at `http://goo.gl/cLEH2`. That archive contains 10 testcases, `1.lab`, `2.lab`, ..., `10.lab`. For each testcase, each unit square is created radioactive with probability $\frac{1}{2}$, different unit squares are generated independently. Then, the rover lands into a random unit square, again chosen with equal probabilities, and then performs the above procedure for 100 000 steps, yielding a transmission of 200 001 characters. Then, each character is replaced by "`?`" with probability $\frac{1}{2}$ (also independently).

There are 10 testcases, testcase 1 corresponding to the example output below.

Input

The only line of the input file contains one integer — the test case number, between 1 and 10.

Output

Output 6 lines with 6 characters each, "`1`" denoting the radioactive unit squares, and "`0`" denoting non-radioactive ones. If there are several possible solutions for which the transmission in the input is possible, output any.

Examples

curiosity.in	curiosity.out
1	010111 000001 010011 101110 001110 011110

Problem D. Domination

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

Pears and apples blossomed on their branches.
River mist was spreading high and wide.
On the steep and lofty bank at morning
Kate came walking by the riverside.

Recall that a set S of vertices of an undirected graph G is a *dominating set* if each vertex of the graph that is not in S is connected to a vertex from S with an edge.

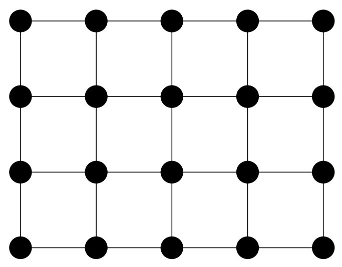
Let's consider a slightly more complicated definition: a mapping M from the set V of vertices of the graph G to the set $0, 1, 2$ is called a *1-2-dominating mapping* if each vertex of the graph that is mapped to zero is connected to a vertex that is mapped to 2 with an edge. The vertices that are mapped to 2 correspond to vertices from dominating sets, but now we have a new possibility — mapping some vertices to 1 covers them, but *not* their neighbors. We define the size of the 1-2-dominating mapping to be the sum of its values: $|M| = \sum_{v \in V} M(v)$.

One might imagine that we're trying to organize the defense of a country, and want to place small garrisons that can defend one region, and large garrisons that can defend one region and still have forces left to send to neighboring regions to defend them.

What is the minimum possible size of a 1-2-dominating mapping for the given graph? You might've guessed that this is a tough question in general, since even finding the smallest dominating set is hard. However, in this problem we're concerned with a very special type of graph: a grid.

A grid is a graph with nm vertices labeled from $(1, 1)$ to (n, m), where vertices (x, y) and $(x + 1, y)$ are adjacent, for all suitable values of x and y, and so are vertices (x, y) and $(x, y + 1)$.

Here's a grid with $n = 4$, $m = 5$:



Input

The only line of the input file contains two integers n and m, $1 \le n \le 10$, $1 \le m \le 1000$.

Output

On the first line of the output file print the smallest size of a 1-2-dominating mapping for the given grid. On the next n lines output m characters each ("0", "1" or "2"): the values assigned to each vertex by the mapping that achieves the smallest size. There should be no spaces on those lines.

In case there are several possible mappings that achieve the smallest size, output any.

Examples

domination.in	domination.out
4 5	11 00201 20000 00022 02000

Problem E. Easy Learning

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

He climbed over the sofa and hugged me around the chest.
“You listen to me, do you hear?” he said threateningly. “Nothing in the world is identical. Everything fits the Gaussian distribution. One water is different from another... This old fool didn't reckon that there is a dispersion of properties...”
“Hey, friend,” I called to him. “The New Year is almost here; don't get carried away!”

Machine learning is usually concerned with reconstructing hidden dependencies in empirical data. This problem is concerned with a specific kind of machine learning, called *Regression Decision Trees*.

A *regression* machine learning task is set up as follows: we have n objects and m features, with each object having a floating-point value for each feature. For example, we could have 2 objects and 3 features, with the first object described as $(1.0, 0.5, 0.3)$, and the second object described as $(0.0, 0.0, 0.4)$, meaning that the value of the first feature for the first object is 1.0, the value of the second feature for the first object is 0.5, and so on. Each object also has an associated *objective function* — another floating-point value.

Our goal is to reconstruct the objective function using only the features. For example, in the above example let's say the first object's objective function is 0.6, and the second object's objective function is -1.0. Then we could use the following reconstruction: if the value of the second feature is less than 0.2, then the value of the objective function is -1.0, otherwise the value of the objective function is 0.6. It's not hard to see that this rule will reconstuct the objective function correctly given the features (in fact, just the second feature) for the above two objects. We'll call such rule a *model*.

Of course, in real life there will be much more objects than features, and thus the task of building a good model is not that simple. More specifically, we'll have 1000 objects and 10 features in this problem.

We'll also consider only a specific class of models, called *decision trees*. The rule mentioned above is a very simple decision tree, with just one “decision” in it. More generally, a decision tree is a function from the feature values to the objective function value that looks like follows:

- if feature 1 is less than 0.3:
  - if feature 2 is less than 0.5:
    - return 1.5
  - else:
    - return -1.0
- else:
  - if feature 1 is less than 0.7:
    - return 2.0
  - else:
    - return 0.0

Each node in the decision tree is either a *split node*, characterized by the feature number and a floating-point boundary value for that feature, or a *leaf node*, characterized by the predicted objective function value. A split node always has exactly two children, the first for objects that have the feature less than the boundary value, and the second for objects that have the feature greater than or equal to the boundary value. A leaf node has no children.

The *depth* of a leaf node in a decision tree is the number of split nodes leading to it. The depth of the entire tree is the highest depth of its leaf nodes. For example, the depth of the above tree is 2. In case all leaf nodes in a decision tree have the same depth d, this is a *full depth-d decision tree* — the above tree, for example, is a full depth-2 decision tree.

In order to express complex formulas with decision trees, they need to be very deep, which is not very convenient. An alternative way which we'll use in this problem is to express the objective function as the sum of several decision trees. For example, if we sum the two decision trees from above (the first one which had just one split node, and the second one which was full depth-2), then the predicted objective function value for the first object will be $-1.0 + 0.0 = -1.0$, and for the second object we get $0.5 + 1.5 = 2.0$.

Given feature values and the objective function values, we need to construct the sum of decision trees that approximates that objective function.

The testcases for this problem are generated randomly. We have 1000 objects and 10 features, with each feature value chosen independently and uniformly between 0.0 and 1.0. Then, we construct 100 random decision trees and sum them to get the objective function. Each random decision tree is full depth-2, with each split node's feature chosen independently and uniformly between 1 and 10, each split node's value chosen independently and uniformly between 0.0 and 1.0, and each leaf node's value chosen independently and uniformly between -1.0 and 1.0. You can use this decsription to create many testcases to work with on your computer.

You don't need to reconstruct those 100 decision trees exactly. You need to output a sum of at most 10000 decision trees with depth at most 4 that is close to the given objective function. More precisely, if the given objective function is f_i and your decision trees sum to g_i, then you just need that

$$\sqrt{\frac{\sum_i (f_i - g_i)^2}{1000}} < 2.0$$

However, just finding such sum would be a useless task, because we know the value of the objective function anyway. In order for the constructed function to be useful, it must have *predictive power* — work well on other objects that are related to the known ones. For the purpose of this problem, we will actually construct *two* sets of 1000 objects each, using the same 100 random decision trees to construct the objective function from features for both of them. Your program will then be given the first set of objects, and will output the sum of decision trees. The result will be tested against the second set of objects, and the above inequality on the difference between the true objective function and the predicted objective function must hold there. Since the objective function is computed in the same way for both sets of objects, you should be able to construct such sum of decision trees that works both for the objects you're given and for the ones only known to the judging program.

Input

The input file contains exactly 1000 lines with exactly 11 floating-point numbers each. Each line describes one object. The first number in each line is the objective function value, and the next 10 numbers are the values of features. Those objects are generated according to the algorithm described in the problem statement.

Since this would be too many numbers to print in a problem statement, you can find an example input at `http://goo.gl/aq0EP`.

Output

On the first line of the output file, print the number k of trees in your sum ($0 \leq k \leq 10000$). On the next k lines describe the trees, one per line. The tree is described as follows: if the tree is just a leaf node, then you print 0, followed by the return value for that leaf node. If the tree has a split node as the first node, then you print the number of the feature you split on (between 1 and 10, inclusive), the boundary value for that feature, then print the left subtree recursively (the one that corresponds to objects where the given feature is less than the given boundary value), and then print the right subtree recursively. Note that all trees must have depth of at most 4.

You can find an example output that yields the average error of 0.27 (according to the above definition) for the objects given in the example input, and the average error of 1.74 for the *second* set of objects with the objective function generated in the same way as in the example input, at `http://goo.gl/PJmDL`. The second set of objects that your solution is tested on for this example is available at `http://goo.gl/zVyqJ`. Notice that the average error on the second set can be quite different from the error on the set you learn on.

Problem F. Hash

Input file:	`hash.in`
Output file:	`hash.out`
Time limit:	25 seconds
Memory limit:	4 megabytes

I write to you — no more confession
is needed, nothing's left to tell.
I know it's now in your discretion
with scorn to make my world a hell.

Cryptographic hash functions are an awesome concept that's useful for many different aspects of today's electronic communications, and digital signatures that make sure the letter does indeed come from the alleged sender is one of its important uses.

However, in order for a hash function to be useful in such context, it needs to be good enough. The definition of "good enough" varies, but a reasonable estimation is that an adversary should not be able to produce two different messages that hash to the same value.

Probably the simplest reasonable hash function is the *polynomial hash function*. Assuming the message is a sequence a_i $(1 \le i \le n)$ of zeroes and ones, its hash is defined as:

$$hash(a) = \left(\sum_{1 \le i \le n} (a_i + 1) \cdot b^{n-i} \right) \mod m$$

where b and m are the hash function's parameters.

Given b and m, find two different messages a and b such that $hash(a) = hash(b)$.

Note that the memory limit in this problem is just 4 megabytes. We hope that the judging system will handle such small limit correctly, and we do have passing solutions in C++ and Java. Please don't use too much memory!

Input

The only line of the input file contains two integers b and m, $2 \le b \le m - 2$, $4 \le m \le 10^{14}$.

Output

Output two lines containing one sequence of zeroes and ones each. Each line should contain at least 1 and at most 1000 characters. Any two different sequences that hash to the same value will be accepted.

Examples

hash.in	hash.out
3 10	10000 00001

Problem G. RLE Size

Input file:	`rle-size.in`
Output file:	`rle-size.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

They say with disdain that we are insane,
That we all perish just in vain,
But it's much better than from vodka or rum —
The others will come, Quitting lives, so calm,
Afraid of no risk or harm —
They'll make it to the top if you succumb!

Run-length encoding (RLE) is one of the basic compression methods which encodes sequences of equal bits/bytes/words as a single bit/byte/word plus count. Thus the compressed size of the data is proportional to the number of consecutive sequences of equal bits/bytes/words in it.

This problem is concerned with strings consisting of just two characters, '`+`' and '`-`'. The *RLE size* of such string is the number of blocks of consecutive equal characters. For example, the RLE size of string '`++--+---`' is 4, because it has the following 4 blocks of consecutive equal characters: '`++`', '`--`', '`+`', '`---`'.

Now suppose you know some of the characters in such string but not the others. What is the minimum and maximum possible RLE size of that string?

For example, consider string '`+??+`', where '`?`' stands for unknown character. The RLE size of this string can be 1 (when all '`?`' characters are actually '`+`' characters, '`++++`') or 3 (when one or two of them are '`-`' characters, like '`++-+`' or '`+--+`').

Input

The first line of the input file contains n, $1 \le n \le 100$ — the number of characters in the string.

The second line of the input file contains a string with n characters, each character being one of '`+`', '`-`' or '`?`'.

Output

Output the minimum possible RLE size of the string in the input, followed by a space, followed by the maximum possible RLE size of the string in the input.

Examples

rle-size.in	rle-size.out

`4`
`+??+`
	`1 3`

Problem H. Good Students and Bad Students

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

But there was such accumulated bitterness and contempt in the young man's heart, that, in spite of all the fastidiousness of youth, he minded his rags least of all in the street. It was a different matter when he met with acquaintances or with former fellow students, whom, indeed, he disliked meeting at any time.

$2kn$ students have entered the university this year, and we need to split them into n groups, each of size $2k$. Surprisingly, we've decided to take *their* opinion into account.

Each student has declared whether he wants to be in the stronger half of his group, or in the weaker half. More formally, each student is assigned a distinct integer *level*, and some students want to be in the first half of their group when sorted by level descending (mostly to feel good about themselves), and all other students want to be in the second half of their group in such ordering (to actually learn as much as possible from others).

You need to separate the students into groups in such a way that as many preferences are satisfied as possible.

Input

The first line of the input file contains two positive integers k and n ($1 \le 2kn \le 100\,000$). The second line contains $2kn$ distinct positive integers not exceeding 10^9, denoting the levels of the students. The third line contains $2kn$ integers, each either 1 if the corresponding student wants to be in the stronger half, or 0 if he wants to be in the weaker half.

Output

On the first line, output one integer — the number of preferences that we can satisfy. Then output n lines with $2k$ space-separated integers each, placing the levels of the students that go into one group on one line. In case there are several solutions that maximize the number of satisfied preferences, output any.

Examples

students.in	students.out

`3 2`
`15 14 12 11 8 7 6 5 4 3 2 1`
`1 1 0 1 0 1 0 0 0 1 1 0`
	`8`
	`15 14 12 8 6 5`
	`11 7 4 3 2 1`

Problem I. Tennis Scores

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

The window looked out on to a narrow street. A militiaman was walking up and down outside the little house opposite, built in the style of a Gothic tower, which housed the embassy of a minor power. Behind the iron gates some people could be seen playing tennis. The white ball flew backward and forward accompanied by short exclamations. "Out!" said Ostap. "And the standard of play is not good."

Let's recall the scoring rules in tennis. I was trying to have an accurate representation of the tennis rules below, but please refer to the formal definitions from this problem statement instead of your previous tennis knowledge to avoid misunderstandings.

Two players play a sequence of *sets*, with each set being a sequence of *games*, with each game being a sequence of *points*. Each point is awarded to exactly one of the two players, and can't be drawn.

In each game, players play points until one of them reaches at least G points, while at the same time having at least two more points than his opponent. For example, when $G = 4$, the game will be awarded to the first player at the score 4:1 and 4:2, but will continue if the score is 4:3 until the difference between players is at least two.

In each set, players play games with $G = 4$ (we'll call those *normal games*) until one of them wins 6 games, and is declared the winner of the set, with one exception: when they have five games each, they play two more games, and if one of them has won 7 games at this point he wins the set (with the score 7:5 or 5:7), but if the score is still even at 6:6, they play one more game with $G = 7$ (a *tie-break game*), and whoever wins that game wins the set.

Finally, in the match, players play sets until one of them has won two sets, and whoever does that wins the match. The *final score* of the match describes how many games each player has won in each set, with the match winner's scores listed first. For example, 6:4 5:7 7:6 describes a three-set match.

Now, let's dive into more detail about points. In each point, one player *serves* and the other player *receives*. In a normal game, the same player always serves. In a tie-break game, both players serve, with the number of serves for each player using 1-2-2-2-2-…rule: the first player to serve serves just one point, then the other player serves two points, then the first player serves two more points, and so on. The first player to serve in each game (which is the player who serves the entire game for normal games) alternates: in the very first game, the first player serves, in the second game, the second player serves, and so on. Please note that this alternation is not reset when a new set starts, and continues throughout the whole match. Please also note that a tie-break game also factors into this alternation: the player to serve in the last normal game before the tie-break will serve second on the tie-break but will then serve in the first game of the next set, if any.

There are two types of serves: from right to left (from *deuce court*) and from left to right (from *advantage court*). Those alternate in each game, with the first serve in each game being the from deuce court.

That's it for the rules of tennis, now let's create a mathematical model for it. For one match, we will postulate that only four probabilities guide the entire game: the probability of the first player to win a point when serving from deuce court, the probability of the first player to win a point when serving from advantage court, the probability of the second player to win a point when serving from deuce court, the probability of the second player to win a point when serving from advantage court. It's not hard to see that fixing those probabilities and treating different points as independent gives us a way to model the entire match probabilistically. Please note that the "first player" in this paragraph refers to the player to serve in the very first game of the match.

What is the probability that the match ends with the given final score (note that the winning player is always listed first in the final score, so you need to incorporate both the probability of the first player winning with this score and the probability of the second player winning with this score)?

Input

The first line contains four integers between 1 and 99, inclusive, denoting the four probabilities defining our model in percent (the probability of the first player to win a point when serving from deuce court, the probability of the first player to win a point when serving from advantage court, the probability of the second player to win a point when serving from deuce court, the probability of the second player to win a point when serving from advantage court). The second line contains the requested score of the match, written as a space-separated sequence of colon-separated pairs of integers denoting the number of games won by the overall winning player and by the overall losing player in each set.

It is guaranteed that this score is a valid tennis match final score.

Output

Output one floating-point number denoting the probability of that final score for the match. Your answer will be accepted if it's within 10^{-9} of the correct answer.

Examples

tennis.in	tennis.out

`40 50 50 50`
`6:0 6:1`
	`0.002899707691413504`

Problem J. Three Squares

Input file:	`three-squares.in`
Output file:	`three-squares.out`
Time limit:	6 seconds
Memory limit:	256 megabytes

Past the woods and mountains steep,
Past the rolling waters deep,
You will find a hamlet pleasant
Where once dwelt an aged peasant.
Of his sons — and he had three,
Th'eldest sharp was as could be;
Second was nor dull nor bright,
But the third — a fool all right.

The aforementioned three sons have grown up and are now trying to settle in life. They've built houses on their father's land, and now need plots of land to cultivate. The father has decided to give each a plot in the form of a square with side equal to 10. Each son's house needs to be exactly in the center of the corresponding plot.

Given the locations of the houses, you need to determine if it's possible to place three square plots with side equal to 10, each centered at the corresponding house, so that the plots don't intersect or touch (to be more precise, we'll require the distance between plots to be at least 10^{-5} — see output format for details). In case that's possible, you also need to find at least one such placement.

Note that the plots may be rotated arbitrarily, their sides don't need to be parallel to coordinate axes.

Input

The input file will contain exactly 3 lines, each containing two integer coordinates not exceeding 100 by absolute value — the locations of the houses. The x axis goes from left to right, and the y axis goes from bottom to top, as usual.

Output

In the first line, output "PEACE" (without quotes) if such placement is possible, and "WAR" (without quotes) otherwise. In case the placement is possible, output three floating-point numbers on the second line: how much (in radians) each square is rotated counter-clockwise relative to the parallel to coordinate axes position. Note that all rotations are covered by values between 0 and $\frac{\pi}{2}$, but we allow arbitrary values for your convenience.

Please output the numbers as precisely as possible. When checking whether your solution is correct, we'll verify that the distance between plots is at least 10^{-5}. It is guaranteed that when any sought placement is possible, there will be a placement with distance between plots at least 10^{-4}, and when no such placement is possible, there will be no possible placement for plots with side equal to $10 - 2 * 10^{-4}$ as well.

Examples

three-squares.in	three-squares.out

0 0
2 10
50 50
	PEACE
	1.1780972450961724 1.1780972450961724
	0.7853981633974483

Note

Here's the picture for example output: