Problem 1. Barsik

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

In one of the corners of a $N \times M$-sized rectangular field, in a cell with the coordinates (1, 1), sits a hungry cat named Barsik. Barsik's bowl is in the opposite corner of the field, in the cell with the coordinates (N, M). Barsik can traverse the field by moving between cells adjacent by side.

However, there is an obstacle, a vicious dog named Tuzik, who sits in a kennel with the coordinates (R, C). Tuzik is chained to the kennel and thus can only reach cells that are within S moves from the kennel (each move going to a cell adjacent by side). All such cells are filled with bones of dead barsiks, and our Barsik cannot make himself walk through them.

Barsik desperately needs to know if he can reach his bowl without stepping into Tuzik's area.

Input

The first line of the input file contains an integer T — the number of tests in the problem ($1 \le T \le 2\,000$).

The following T lines contain descriptions of tests, one per line.

Each tests consists of five space-separated integers N, M, R, C, and S ($1 \le R \le N \le 10^9$, $1 \le C \le M \le 10^9$, $1 \le S \le 10^9$).

It is guaranteed that the cell with Tuzik's kennel is placed in such a manner that he cannot reach neither the cell with the original position of Barsik nor the cell with Barsik's food.

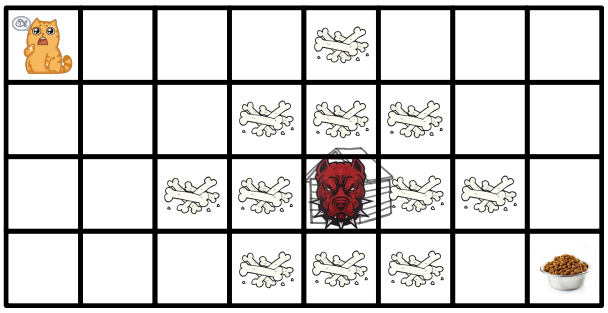
Output

For each test, print an answer in a separate line. Print `Barsik`, if Barsik can reach the food without stepping over bones. Otherwise print `Tuzik`.

Examples

input.txt	output.txt
2 8 4 5 3 2 8 4 3 3 1	Tuzik Barsik

Illustration for the first test from the sample:



Problem 2. The one who works is the one who eats... nothing

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

«Agreement is a product with full non-resistance of parties», nevertheless, Mechnikov was forced to haul the theatrical props.

By the time he had finished p percent of the task, the manager of the theater realized they did not have enough provision to feed the worker until he finished the work. Mechnikov ate q percent of the received provision, with q greater than p.

That's when the manager, by slightly altering the popular motto «the one who does not work is the one who eats» came up with «the one who works is the one who eats... nothing» (or almost nothing). For this reason, he decided to assign some free workforce to the toiling proletarian — a few coders who work for food. Each of the coders works a times faster than the electrician and eats b times slower.

The supply of free coders is virtually unlimited, but there is no reason to get too many of them. What if this becomes news and a great crowd of them arrives, craving free food? Help the manager to find the smallest necessary number of coders sufficient to make the available provision last until the work is finished.

Input

The first line of the input file contains a single number t ($1 \le t \le 3 \cdot 10^4$) — the number of tests. Each of the following t lines of the input file contains four integers $0 < p, q, a, b < 100$, denoting, correspondingly, the percentage of completed work, the percentage of consumed provision, and how many times faster does a coder work, and how many times slower he eats. It is guaranteed that $q > p$.

Output

As an answer to each of t tests, print a single integer — the minimum number of coders necessary to do the job. If it is impossible to complete the work, print -1.

Examples

input.txt	output.txt
3 1 2 99 1 30 60 2 2 30 80 1 2	1 10 -1

Problem 3. Bombs

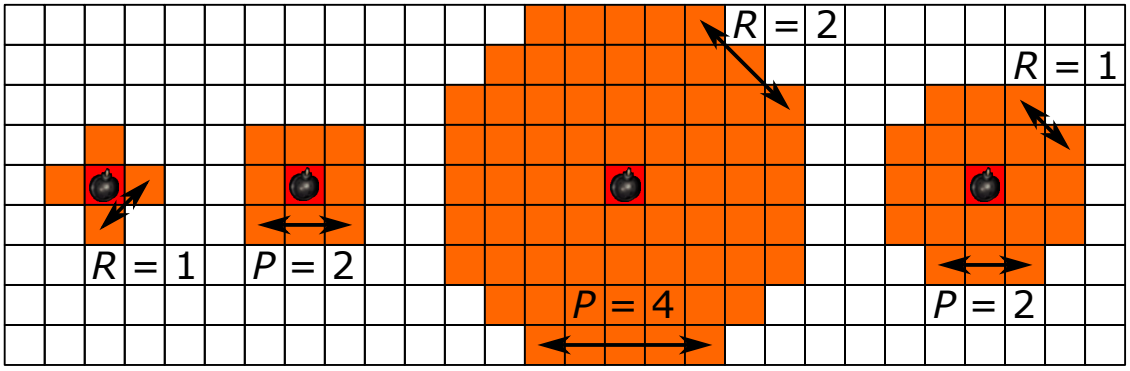
Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	2 seconds
Memory limit:	256 megabytes

Gennadiy is playing his favorite game.

The game level is a field of square cells. Each cell is either empty or filled with ground. An empty cell can contain the level entrance or exit. The player can walk on empty cells, moving to a cell adjacent by side with each move. The player cannot move outside the field.

Some levels cannot be passed by regular means, and in such cases, bombs are used. A bomb can be stuck to the ground, and it will explode, clearing the nearby cells from ground. For the purpose of this problem, assume that the player can place a bomb in the cell where he is currently located if it is adjacent to a ground cell. After the bomb explodes, all cells in the explosion zone become empty and thus traversable. The player, level entrance, and level exit can safely be in the explosion zone, they are not affected by the explosion.

The explosion zone is an octahedron with the center in the cell where the bomb was placed. The explosion zone has two parameters P and R, and its precise shape is defined in the following manner. To traverse its border cells along the perimeter, you must move up P times then move up and right R times, then move right P times, move right and down R times, then move down P times, etc. until you return to the starting cell. The shape of the explosion zone with some values of the parameters is shown in the figure:



Bombs are a limited and valuable resource and thus should be used sparingly. Help Gennadiy pass a level with a minimum number of bombs.

Input

The first line of the input file contains two integers N and M — the vertical and horizontal size of the field, respectively ($1 \le N, M \le 1\,500$). The second line contains two integers P and R — the parameters of the explosion zone ($0 \le P, R \le 1\,000$). It is guaranteed that the number P is even, and that $P + R > 0$.

Next, the level map is provided, as N lines with M symbols in each. Each symbol describes the contents of the corresponding cell:

- `@` — cell with ground.
- `.` — empty cell.
- `S` — empty cell with the level entrance.
- `E` — empty cell with the level exit.

It is guaranteed that the map only contains one entrance cell and one exit cell.

Output

In the first line of the output file, print an integer B — the minimum sufficient number of bombs ($B \geq 0$). In the remaining B lines, print the cells where bombs must be placed, in the order of their detonation. In each line, print two integers u and v — the indices of row and column, respectively ($1 \leq u \leq N$, $1 \leq v \leq M$).

Examples

input.txt	output.txt
9 10 0 1 @@......@@ @@S@@@@... @@@@@...@. @@@@@@@@@@ @@@@@@@@@@ ....@ @@@.@@@@.. @...@@@E@. @@@@@@@@@@	3 3 7 4 7 8 10
4 5 2 1 S@@@@ ..@@@ @@@@E @@@@@	1 2 2
7 5 4 0 @E@@@ @@.@@ @@@@@ @@@@@ @@... @@... @@..S	2 5 5 2 3

Example explanation

In the first example, the explosion is cross-shaped and affects the cells sharing a side with the bomb cell. Two bombs are needed to break the wall in the lines 4 and 5, and another bomb to reveal the exit. In the second example, the explosion is powerful enough to create a passage with just one bomb. In the third example, two bombs are required. Note that it is forbidden to place the first bomb in cell (6, 4), because this cell does not contact ground.

Problem 4. Fix the heap 8-bit

Input file:	`input.bin`
Output file:	`output.bin`
Time limit:	1 second
	3 seconds (for Java)
Memory limit:	**64** megabytes

Petya wrote a heap for dynamic memory allocation in his own programming language Tse. Here is how it works.

A correct heap is a continuous segment of memory consisting of N cells. Each cell contains an unsigned **8-bit** integer, which can have any value **from 0 to 255** inclusive. The heap is split into B sub-segments called blocks. Each of the memory cells belongs to strictly one block.

Each block consists of two or more cells. The first and the last cells of a block contain the effective size of the block, which is the number of cells in it, excluding the first and last cells. The rest of the block's cells can contain arbitrary data regardless of whether the block is free or occupied.

Tse is a very low-level language, and its users often mess everything up and corrupt the heap by inadvertently writing their data into wrong cell. Users are asking Petya to come up with a feature of recovering heap integrity after such incidents. The recovery code must analyze the contents of N cells of the memory segment where the heap is supposed to be located, and change the contents of several memory cells in such a manner that the segment becomes a correct heap. The number of modified cells must be minimal.

Input

The contents of N memory cells comprising the analyzed memory segment are provided in a binary file of size N bytes. Each byte of the file represents a single memory cell. $2 \le N \le 2^{24}$ holds true, meaning that the size of the file is not less than two bytes and not greater than 16 megabytes.

Output

Write the found set of K cell changes into a binary file of size $4K$ bytes.

Each change is described by four bytes. The first three bytes are treated as an unsigned 24-bit integer and define the address (index) of the modified cell. Naturally, this number must be smaller than N — the size of the input file. The last, fourth byte defines the new contents of the cell.

24-bit addresses must be written in little-endian byte order, with the first byte being the least significant one and the last byte being the most significant one. If there are several answers with the same number of changes, you can write any of them. The order of describing cell changes in the output file is irrelevant.

Examples

For convenience, the contents of the input and the output files are shown below in hex format. In the testing system, the files will be in binary! You can download samples in binary format in the «News» tab near the problem statements.

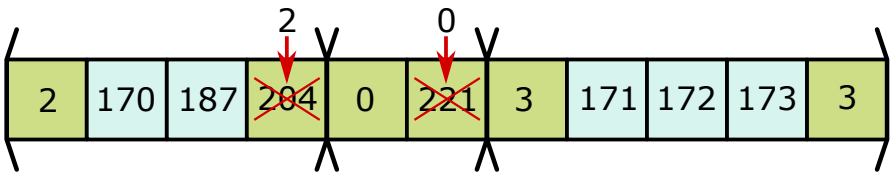
For fast file reading it is recommended to use one call of: `Files.readAllBytes` on Java and `fread` on C++. Don't forget that files should be opened in binary mode on C++.

input.bin	output.bin
02 AA BB 02 00 00 03 AB AC AD 03	
02 AA BB CC 00 DD 03 AB AC AD 03	03 00 00 02 05 00 00 00

Example explanation

In both examples $N = 11$. In the first example the heap is already correct, containing three blocks with effective sizes 2, 0, and 3, respectively. Since no changes are necessary, the output file must be empty. In the second example, the heap is incorrect, but it can be fixed with two changes, by turning it exactly into the heap from the first example. For this purpose, it is suggested that the cell with the address 3 be changed from `CC` to `02`, and the cell with the address 5 be changed from `DD` to `00`.

Illustration for the second example (integers displayed in decimal format):



Problem 5. Fix the heap 32-bit

Input file:	`standard input stream`
Output file:	`standard output stream`
Time limit:	3 seconds
Memory limit:	256 megabytes

Petya wrote a heap for dynamic memory allocation in his own programming language Tse. Here is how it works.

A correct heap is a continuous segment of memory consisting of N cells. Each cell contains an unsigned **32-bit** integer, which can have any value **from 0 to 4 294 967 295** inclusive. The heap is split into B sub-segments called blocks. Each of the memory cells belongs to strictly one block.

Each block consists of two or more cells. The first and the last cells of a block contain the effective size of the block, which is the number of cells in it, excluding the first and last cells. The rest of the block's cells can contain arbitrary data regardless of whether the block is free or occupied.

Tse is a very low-level language, and its users often mess everything up and corrupt the heap by inadvertently writing their data into wrong cell. Users are asking Petya to come up with a feature of recovering heap integrity after such incidents. The recovery code must analyze the contents of N cells of the memory segment where the heap is supposed to be located, and change the contents of several memory cells in such a manner that the segment becomes a correct heap. The number of modified cells must be minimal.

Interaction Protocol

The number of memory cells can be very large, and your program won't be able to read the whole segment. For this reason, this is an interactive task. Instead of file input-output, you will be working with a special program called interactor. Interaction with this program is performed by means of the standard input-output streams.

At the start, your program receives an integer N in the standard input stream, being the number of memory cells in the analyzed segment ($2 \le N \le 2^{32}$). Next, your program can make queries to discover the contents of various memory cells. To make a query, print the word `read` into the standard output stream, followed by a space-separated integer A, being the address/index of the cell ($0 \le A < N$). In response, a single integer V will be sent to the standard input stream, being the value stored in the A-th memory cell ($0 \le V < 2^{32}$).

Eventually, your program must print the answer to the problem instead of yet another query. In the first line of the answer, print the word `fix`, followed by a space-separated integer K, being the number of cells that must be changed ($K \ge 0$). In the remaining K lines, print the suggested changes, one per line. For each change, print two integers: A being address/index of the cell that must be changed and V being new value written into this cell ($0 \le A < N$, $0 \le V < 2^{32}$).

It is guaranteed that there exists an optimal answer resulting in a heap with no more than 25 000 blocks. Your program is allowed to make no more than 120 000 queries, otherwise the solution will get Wrong Answer.

Make sure you print the end-of-line symbol and clear the output stream buffer (the `flush` command of the language) after every printed query. Otherwise the solution can get Timeout. All numbers are written in decimal form in text format.

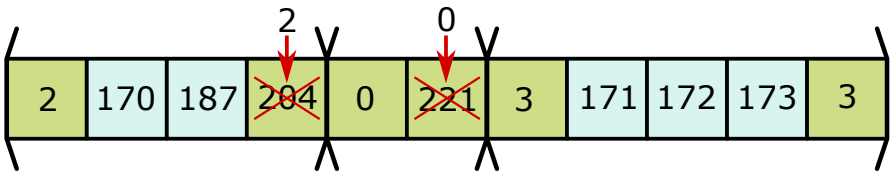
Examples

For reading convenience, commands in the example are separated by lines with the minus symbol. Your program must **not** print any minuses.

standard input stream	standard output stream
11 - 2 - 170 - 187 - 204 - 0 - 221 - 3 - 171 - 172 - 173 - 3 - - -	- read 0 - read 1 - read 2 - read 3 - read 4 - read 5 - read 6 - read 7 - read 8 - read 9 - read 10 - fix 2 3 2 5 0

Example explanation

The example fully repeats the second example from another problem «Fix the heap 8-bit»: both the contents of the memory cells and the suggested recovery fixes are the same. Illustration:



Problem 6. Fraction

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	3 seconds
Memory limit:	256 megabytes

During his first days in the mental ward, Berlaga pretended to be the viceroy of India. After giving it some thought, he decided it was risky — they could put him on an elephant and make him steer the beast through the streets. He opted for a change of story. He hasn't decided yet what will be his next persona. Meanwhile, he's enjoying his favorite pastime: playing Solitaire.

Every now and then, he begins to wonder: what portion of the games does he win? Solitaire itself provides the answer with only two decimal places, and Berlaga loves precision and sometimes calculates the proportion himself. The answer can be both a finite or a repeating fraction, depending on the proportion itself and on the base of the numeral system used. That's right, mad accountants can use any positional numeral system they want, and not only decimal. For instance, in the decimal system, $1/3$ is an infinite repeating fraction, and $4/10$ is a terminating fraction, however, in ternary numeral system, everything is vice versa: $1/3$ is a terminating fraction "0.1", and $4/10$ is an infinite recurring fraction "0.101210121012…".

Naturally, like all accountants, he does not like infinite fractions and prefers to calculate the proportion of his wins only when he is absolutely sure that it will be a terminating fraction. To do this, he needs to play a few more games.

Help Berlaga find the minimal number of games to be played additionally. The total number of the played games must not be greater than M.

Input

The first line of the input file contains three numbers: B — the base of the numeral system, M — maximum allowed number of games and N — number of queries ($2 \le B \le 5 \cdot 10^6$, $1 \le M \le 10^{18}$, $1 \le N \le 10^5$).

Next N lines describe the queries. Each line contains a single integer a_i — the number of games already played by Berlaga ($1 \le a_i \le M$).

Output

The output file must contain N lines, with an answer to the corresponding query in each line. Each answer must be an integer A — the number of games to be played additionally to assure that the proportion of wins is a terminal fraction ($A \ge 0$). If the total number of games in this case is greater than M, print -1 as the answer.

Examples

input.txt	output.txt
100 120 3 5 117 13	0 -1 3

Problem 7. Roadside optimization

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

Some of the towns in the province G are connected with roads; roads always go both ways. The budget to support the road network is scarce, therefore it was decided to leave the bare minimum of the roads. However, if it is currently possible to reach the town B from the town A, this possibility must remain even after the reduction.

Help to define the minimum number of roads that must remain.

Input

The first line of the input file contains a single integer T — the number of tests ($1 \le T \le 50\,000$). It is followed by the description of T tests.

The first line of the test number t contains a single integer N_t — the number of towns ($1 \le N_t \le 200$).

The following N_t lines contain N_t integers; each integer is either 0 or 1. If the line with the number i has 0 in the jth position, then the town j can not be reached from the town i (even by driving through other towns); if it is 1, then there is a passage. It is assumed that there is a passage from a town to the same town, so there will always be 1 in the ith line in the ith position ($1 \le i \le N_t$).

It is guaranteed that the sum of N_t^2 over all tests is not greater than 50 000.

Output

For each test, print a single integer on a separate line — the minimum number of roads to be kept.

Examples

input.txt	output.txt
1 1 1	0
2 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 0 1 1 0 1	3 4

Problem 8. Wooden pipeline

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	2 seconds
Memory limit:	256 megabytes

Once upon a time in a kingdom far, far away, there ruled a wise king. The king decided to make the life of his people better, and urged the lords to do innovation, modernization, and nanotechnology. The lords gave it a thought and came up with a pipeline. The nanotubes their serfs produced weren't «that» great, so they substituted them with wooden pipes.

The wooden pipeline connects all N towns of the kingdom into a single network, such that you can walk along the pipeline to reach any town from any other town. The pipeline network consists of $N-1$ pipelines. Each water pipeline leads directly from one town to another without any kind of forking. The capacity of each pipeline is known, which is the maximum volume of water that can flow through the line in a unit of time.

Years later, the reign of the wise king came to an end. His son switched the country to capitalist tracks. His grandson inherited a modern state with an empty budget and endless debt. All these years, the pipeline was on standby. But nothing happened to it, as it was built to last forever! Well, this is not exactly accurate. Something did happen — parts of it were privatized, just in case. At last, drought happened in the town of U, and the grandson recalled his grandfather's plans — they will pump water through the pipeline and supply the town of U.

All towns, except U, can be divided into two types: terminal and intermediate. A town is intermediate if it has a pipeline leading to a town which is more distant from U (distance is calculated along pipelines). All other towns are considered terminal. It is allowed to take water from all terminal towns in any volume and pump it along pipelines into the town of U. Water cannot be taken from intermediate towns.

Unfortunately, there are mean people who want to profit from the disaster and are taking a fee for pumping water through their sections of the pipeline. The king cannot do anything about it, because of private property, free market and so on. To counteract these vultures, loyal citizens declared that they will be honored to provide their segments of the pipelines to pump water to the town of U, and that they are even willing to pay money for that. As the result, for each pipeline we know the cost per unit of water flowing through it, and this cost can be negative.

The king is broke, and there is no money in the budget. Help him come up with a plan to supply U with as much water as possible, but in such a manner that no extra funding is required.

Input

The first line of the input file contains a single integer N — the number of towns ($2 \le N \le 200\,000$).

The remaining $N-1$ lines describe the pipelines, one per line. Each pipeline is described with four integers: a — the number of the town where the pipeline begins, b — the number of the town where the pipeline ends, M_{ab} — capacity of the pipeline (units of volume per unit of time), C_{ab} — cost of pumping a unit of volume through the pipeline ($1 \le a \ne b \le N$, $1 \le M_{ab} \le 10^6$, $|C_{ab}| \le 10^7$).

It is guaranteed that any town can be reached from any other town along the pipelines. The town U has the number 1.

Output

Print a single real number F — the maximum volume of water that can be pumped into the town of U in a unit of time. The absolute or relative error must not exceed 10^{-12}.

Examples

input.txt	output.txt
2 1 2 10 -15	10
6 1 3 5 -4 1 2 14 2 4 2 6 -1 5 2 3 5 6 2 6 1	15.666666666666666667

Example explanation

In the first example, there is one pipeline with negative cost. You can use it to maximum capacity and earn some money, too.

In the second example, the optimal answer is achieved in the following manner. Pump 5 units of water from 1 to 3, earning 20 coins of profit. Pump $10\frac{2}{3}$ units of water from 1 to 2, which will cost us $21\frac{1}{3}$ coins. Of these water units, pump 6 units from 2 to 4, receiving 6 coins of profit; pump the remaining $4\frac{2}{3}$ units of water from 2 to 6, which will cost us $4\frac{2}{3}$ coins.

Problem 9. Cape and gun

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

Gennadiy came up with a new challenge in his favorite game: pass levels without touching the ground.

In the game, a level is a field of cells. Each cell is either empty or filled with ground. In an empty cell, there can be a monster or the entrance or exit from the level. All cells outside the game field are filled with ground. Monsters cannot move.

Even though the level is presented as a field of cells, game time flows continuously and the player moves across the field continuously. Assume the player is a point. Thanks to the cape, the player is always moving down at a very slow constant speed. The cape allows the player to glide: he can choose the horizontal component of velocity as he wishes at any moment, in addition to the vertical component which cannot be changed. The player cannot pass through ground cells, and in the challenge, the player cannot even touch the ground.

The gun can only be used to shoot left or right. Bullets leave the gun and fly horizontally in the desired direction until the first contact with ground. Monsters perish in all cells on the bullet's path.

To pass a level, player must fly from the cell where the entrance to the level is located to the cell with the exit. It is allowed to start at any point inside the cell with the entrance, and to end at any point inside the cell with the exit. Gennadiy needs to know if he can do this, and if so, what is the maximum number of monsters he can kill in the process.

The player can have arbitrarily high horizontal velocity. The gun has endless ammo and can shoot as rapidly as desired. The player can fly through a cell with a monster, which kills the monster.

Input

The first line of the input file contains two integers N and M — the vertical and horizontal size of the field, respectively ($1 \le N, M \le 2\,000$).

The next N lines with M symbols in each describe the game field. Each symbol describes the contents of the corresponding cell:

- `@` — cell with ground.
- `.` — empty cell.
- `m` — empty cell with a monster.
- `S` — empty cell with level entrance.
- `E` — empty cell with level exit.

It is guaranteed that there is only one entrance and one exit cell on the level.

Output

If it is impossible to fly across the level, print -1, otherwise print the maximum number of monsters killed in the process.

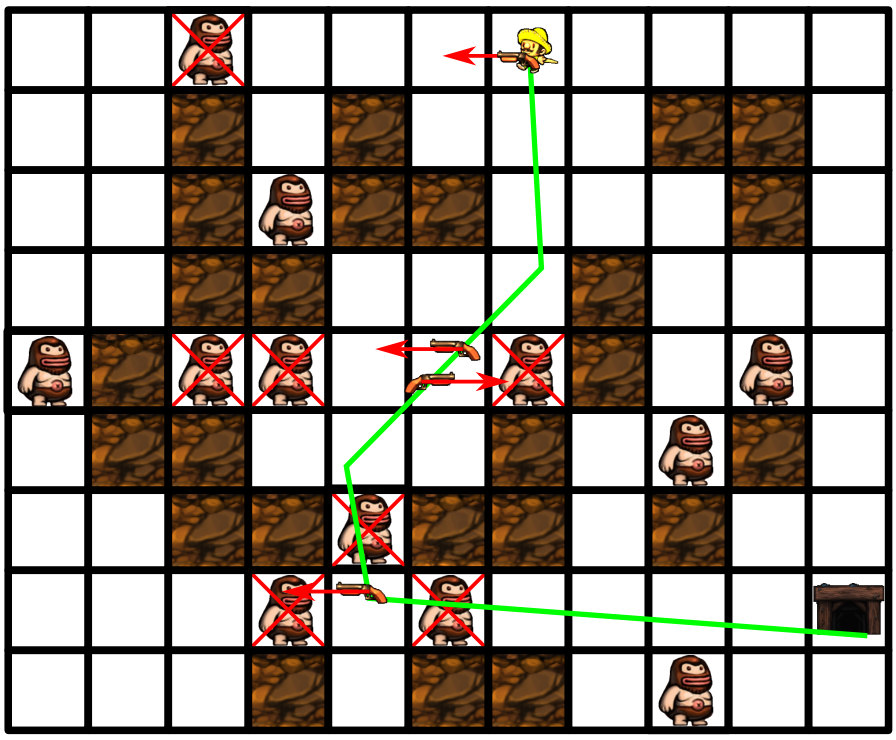
Examples

input.txt	output.txt
9 11 ..m...S.... ..@.@...@@. ..@m@@...@. ..@@...@... m@mm..m@.m. .@@...@.m@. ..@@m@@.@.. ...m.m....E ...@.@@.m..	7
3 1 S @ E	-1

Example explanation

The optimal plan of passing the level from the first example is provided below. Note that it is forbidden to fly diagonally between two ground cells.

In the second example, the path between the entrance and the exit is blocked, so it is impossible to fly through the level.



Problem 10. Ostap and chairs

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

There is an emotional moment in the computer game called «Ostap and chairs», when each Ostap, from a crowd of tiny Ostaps, runs to his own chair. The graphics of the event has been drawn already: there are two images — the first image is of Ostaps (with predefined coordinates x_i), and the second image is of the chairs (their coordinates y_i are also known).

Before you start a game, you cannot move Ostaps or chairs, but you can change the scale of the second picture using a linear transformation $y_i \to k * y_i + b$. After that, the first Ostap runs to the first chair, then the second Ostap runs to the second chair, etc., and the total elapsed time is summed up. The player's task is to make this time as short as possible, i.e. minimize the total summarized distance.

Your task is to find the minimum possible value:

$$\sum_{i=1}^{N} |x_i - (ky_i + b)|$$

Input

The first line of the input file contains a single integer N — the number of Ostaps and chairs ($2 \le N \le 300$). Each of the following two lines contains N integers: the second line of the file contains x_i — the coordinates of Ostaps, the third line contains y_i — the coordinates of chairs ($1 \le i \le N$, $|x_i|, |y_i| \le 10^3$). All x_i are different, all y_i are different.

Output

In your answer, print three real numbers: D — the minimum possible value of the total distance, K and B — coefficients, with which such distance is achievable.

The relative or absolute deviation of the distance D from the optimal must not exceed 10^{-9}. The total distance calculated using the coefficients K and B must match D with the same precision.

Examples

input.txt	output.txt
3 0 3 -5 4 1 -2	5.5 0.8333333333 -3.3333333333
2 -7 12 -7 12	0 1 0