Fireworks in RightAngleles

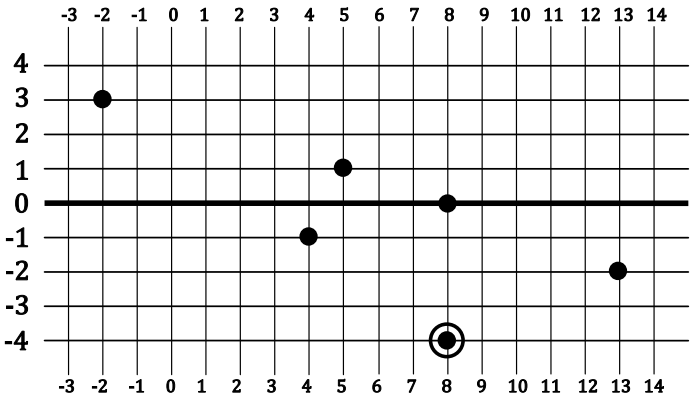
In the city of RightAngleles streets are built as an infinite square grid – any two of them are either parallel or perpendicular to each other and the distance between two nearest parallel streets is the same (let's denote this distance as one *unit*). All streets that are oriented in West-East direction are called *horizontal streets* and are numbered by consecutive integers in South-North direction. All streets that are oriented in South-North direction are called *vertical streets* and are numbered by consecutive integers in West-East direction.

Every citizen lives in a house with entrance located at one particular horizontal and vertical street intersection. There may be several citizens living in the same house.

Mayor of RightAngleles would like to brush up his popularity by organizing a fireworks display at an intersection of the main horizontal street (labelled with number 0) and some vertical street. It is known where the citizens interested in coming and enjoying the fireworks live. Fireworks will be seen along both streets at which intersection the display will take place; furthermore, due to safety reasons during the display observers must be at least S units away from the intersection from which the fireworks are launched. Thus, if the fireworks will be launched from the intersection with vertical street V, then every interested citizen must come to an intersection on the main horizontal street (with number 0) or the vertical street V no closer than S units from the intersection of the main horizontal street and vertical street V. For example, if S=2, then observing may be done from any intersection on the main horizontal street, except the ones that intersect with streets V-1, V, and V+1, and from all intersections on the vertical street V, except the ones that intersect with horizontal streets -1, 0, and 1.

The overall positive influence of fireworks is strongly related to the total distance which citizens need to move to observe the display. Therefore, the intersection for the fireworks must be chosen in a way to minimize this total distance.

For example, if S=2 and there are seven citizens whose locations are shown on the map (there are two of them in (-4;8)), then the best place for the fireworks is the main street intersection with 8th vertical street; in this case the total distance which citizens need to move is 9 units.



Write a program which calculates the minimum possible total distance (in units) which citizens have to move to observe the fireworks.

Input data

Input data is given in the text file **fire.in**. The first line contains two positive integers separated by spaces: the number of citizens N ($N \leq 10^5$) and the safety distance S ($S \leq 10^6$) in units. The next N lines contain the descriptions of the locations of citizens. Each of these lines contains two integers H_i and V_i separated by a space; H_i and V_i ($-10^9 \leq H_i, V_i \leq 10^9$) denote the numbers of the horizontal and vertical street, respectively, of the intersection where the entrance to the i-th citizen's house is located.

Output data

The first and only line of the text file **fire.out** must contain exactly one integer – the minimum total distance (in units) which citizens must move to observe the fireworks.

Example (corresponds to the figure given in the task description)

Input data (file **fire.in**)	Output data (file **fire.out**)
7 2 3 -2 0 8 -4 8 -1 4 -2 13 -4 8 1 5	9

Grading

Test cases where $0 \leq V_i \leq 5000$ are worth 20 points.
Test cases where $N \leq 5000$ are worth 40 points.

Melody

Linas likes to play some musical instrument, and nobody knows what it is called. The instrument has S holes and Linas is able to play *N* different notes (numbered from 1 to *N*) by covering each hole in one of 10 different ways (numbered from 0 to 9). Every note can be played by covering all holes in exactly one way, described by a sequence of digits corresponding to coverings of respective holes. If the holes are covered incorrectly (i.e., not corresponding to any note), the instrument starts to produce very unpleasant sounds, so Linas plays a wrong note rather than covers holes incorrectly.

Linas plays in a band where he has to play complicated tunes very quickly. Linas has written a tune (i.e., a sequence of numbers, corresponding to notes) and he wants to play it together with the band. Unfortunately, Linas doesn't play perfectly. He can only play two successive notes if by playing the second he has to cover no more than *G* holes differently than when playing the first one. Hence he decided to sometimes play a wrong note in the tune. Each incorrect note Linas plays is called mistake.

Task

For a given tune find a modified tune that Linas can play making the least possible number of mistakes.

Input data

First line of text file **melody.in** contains three integers: number of possible notes $N(1 \le N \le 100)$, number of holes *S* and fingers' speed $G(0 \le G < S \le 100)$. Next *N* lines contain the list of possible notes. There are *S* digits without spaces in each of them. The *j*-th digit in the *i*+1-th line corresponds to covering of the *j*-th hole required to play the *i*-th note (hole can be covered in various ways, labelled by digits from 0 to 9). No two notes are played in the same way.

N+2-th line contains the length of the tune $L(1 \le L \le 10^5)$. The last line contains the tune: *L* integers separated by spaces, corresponding to the notes played successively in the tune.

Output data

The first line of the text file **melody.out** must contain one non-negative integer – the minimum number of mistakes. The second line must contain a valid tune which obtains the minimum number of mistakes: *L* integers separated by spaces, corresponding to the notes that Linas should play. If there are multiple such tunes, output any of them.

Example

Input data (file `melody.in`)	Output data (file `melody.out`)	Comment

`5 4 2`
`1111`
`2101`
`2000`
`0100`
`0000`
`7`
`1 5 4 5 3 2 1`
	`1`
	`1 2 4 5 3 2 1`
		Linas can't play note 5 directly after note 1.

Grading

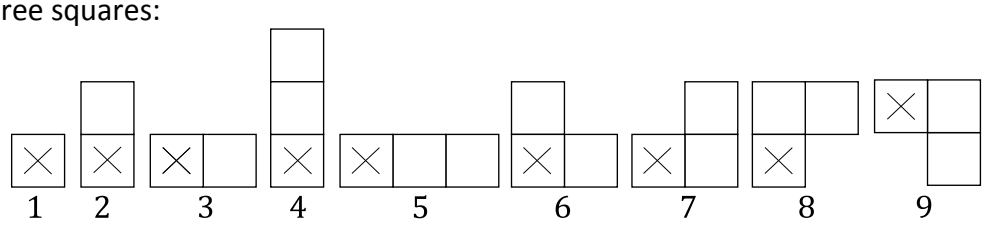
Test cases where L≤100 are worth 40 points.

Test cases where L≤5000 are worth 65 points.

Tiny (open input task)

Elder people still remember the famous computer game "TETRIS" created by Alexey Pajitnov, where pieces consisting of four squares (tetrominoes) fall from the sky and the goal of the game is to rotate and land every piece in a rectangular container creating as many lines of blocks without gaps as possible. When such lines are created, they disappear giving more space for the following pieces.

Let's investigate a simpler version of the game, called "Tiny TETRIS" (or just "Tiny" for short). There are only nine different Tiny pieces (or t-pieces) consisting of one to three squares:



The number denotes the type of a t-piece and will be used further to reference the particular t-piece.

The goal of the game is the same – falling t-pieces must be put in a rectangular container which is 9 units wide and 9 units high. Contrary to TETRIS, t-pieces cannot be rotated. Moreover, they cannot be moved to the left or right after they start falling. Thus, for each t-piece the player must only choose the container's column number (integer from 1 to 9) where the leftmost square of the piece (marked as ×) will fall.

Each game consists of a finite sequence of N t-pieces from which as many as possible must be dropped in the container without exceeding its upper level or making an illegal move. The score of the game is equal to the number of successfully dropped t-pieces.

At the beginning the game counter is set to 0.

The algorithm of the game is the following:

1) Player chooses the column for the leftmost square of the current t-piece;

2) If the column is set correctly (for example, column 8 can never be correct for the t-piece 5), t-piece falls down until it meets an obstacle; otherwise the game is over.

3) If the t-piece fully fits inside the container (i.e., all squares are inside the 9×9 rectangle) the value of the counter is increased by one. Otherwise, the game is over.

4) Then it is checked whether there are any completed horizontal lines (horizontal lines filled completely with blocks of t-pieces without any gaps). If there are any then these lines disappear and the lines above them are shifted down without changing their configuration.

5) If there are any t-pieces left, proceed to 1). Otherwise the game is over.

Score of a particular game is the value of the counter at the moment when the game ends.

Let's analyze one particular game.

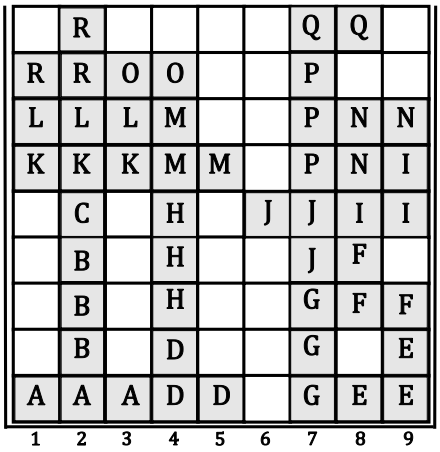
Sequence of the given 20 t-pieces is the following: 5,4,1,6,7,6,4,4,7,9,5,5,6,8,3,4,3,7,4,2. Let's assume that the first 17 t-pieces have already been

						Q	Q	
		O	O			P		
L	L	L	M			P	N	N
K	K	K	M	M		P	N	I
	C		H		J	J	I	I
	B		H			J	F	
	B		H			G	F	F
	B		D			G		E
A	A	A	D	D		G	E	E
1	2	3	4	5	6	7	8	9

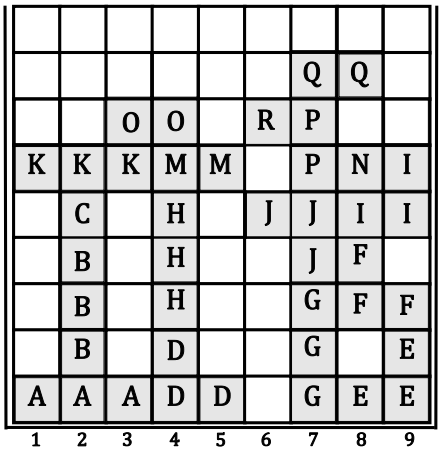
successfully dropped in the container in the columns 1,2,2,4,8,8,7,4,8,6,1,1,4,8,3,7,7, respectively. Until this moment no lines have been completed, the current value of the counter is 17 and it is time to drop t-piece 7 (letters in the figure are assigned consecutively to t-pieces):

There are only two valid columns where t-piece 7 can be dropped:

a) column 1:



b) column 5 (in this case one line will be completed and, therefore, disappear):



Task

You are given five files each containing a description of a particular game: **tiny.i1, tiny.i2, tiny.i3, tiny.i4,** and **tiny.i5**. Each file contains the sequence of t-pieces and has the following format: the first line contains a single integer N. The next N lines describe the t-piece sequence; each line contains an integer between 1 and 9 – the number of the particular t-piece. T-pieces are given in the order how they must be dropped in the container; the i-th t-piece is given in the i+1-st line of the file.

For each of the given input file you must submit a corresponding output file (**tiny.o1, tiny.o2, tiny.o3, tiny.o4,** and **tiny.o5**) with at most N rows – the numbers of the columns where pieces are dropped. The i-th row of the output file must contain the number of the column where the i-th t-piece from the input data must be dropped.

It is guaranteed that for each input file there exists a sequence of columns which allows all t-pieces to be successfully dropped in the container (and gets the final score for the game equal to N).

Grading and feedback

Each of the five test cases is worth 20 points. The amount of points you will receive for a particular output file (test case) is calculated using the following formula:

`20 × your_score/maximum_score_among_all_contestants`,

rounded to the nearest number with 2 digits after the decimal.

During the competition you will receive feedback for each submitted output file: your score and the amount of points you would receive for this output assuming that someone gets the perfect score. After the competition the output files will be re-evaluated with the actual maximum score among all contestants and you may receive more points for the file.