Problem A. 1's For All

Source file name:	Forall.c, Forall.cpp, Forall.java, Forall.py
Input:	`Standard`
Output:	`Standard`

The *complexity* of an integer is the minimum number of 1's needed to represent it using only addition, multiplication and parentheses. For example, the complexity of 2 is 2 (writing 2 as $1 + 1$) and the complexity of 12 is 7 (writing 12 as $(1 + 1 + 1) \times (1 + 1 + 1 + 1)$). We'll modify this definition slightly to allow the concatenation operation as well. This operation (which we'll represent using ©) takes two integers and "glues" them together, so 12 © 34 becomes the four digit number 1234. Using this operation, the complexity of 12 is now 3 (writing it either as $(1$ © $1)+1$ or 1 © $(1+1)$). Note that the concatenation operation ignores any initial zeroes in the second operand: 1 © 01 does not result in 101 but results in 11.

We'll give you 1 guess what the object of this problem is.

Input

Each test case consists of a single line containing an integer n, where $0 < n \leq 100\,000$.

Output

Output the complexity of the number, using the revised definition above.

Example

Input	Output
2	2
12	3

Problem B. Abridged Reading

Source file name:	Abridged.c, Abridged.cpp, Abridged.java, Abridged.py
Input:	`Standard`
Output:	`Standard`

Miss Othmar is a grade school teacher who uses a very interesting textbook in her science class. All of the chapters in the book have material that depends on at most one previous chapter's material, while several chapters are labeled "Culminating Concept" chapters that have no chapters depending on them —- they basically represent the culmination of the stream of material in all the previous chapters that must be read before them. Chapters that are not Culminating Concept chapters are referred to as "prerequisite chapters."

Because of various delays caused by the pandemic, Miss Othmar is far behind where she wants to be in the class. It's too late to try to cover all of the Culminating Concept chapters in the book (and their required prerequisite chapters) so she has decided to cover just two more Culminating Concept chapters. To give the kids a break she has decided to pick the two Culminating Concept chapters which require the least amount of reading —- this would include not only those chapters but their prerequisite chapters as well.

Input

Input starts with a line containing two integers n m where n ($2 \leq n \leq 1\,000$) indicates the number of chapters (numbered 1 to n) and m ($0 \leq m < n$) indicates the number of chapter dependencies. After this are n positive values indicating the number of pages in each chapter. These values will be on one or more lines and the number of pages in any chapter is $\leq 1\,000$. After this are m lines each containing two integers a b ($1 \leq a < b \leq n$) indicating that chapter a must be read before chapter b. No chapter appears as the second integer in these lines more than once. There will be at least two Culminating Concept chapters.

Output

Output the minimum number of pages that need to be read in order to complete two Culminating Concept chapters.

Example

Input	Output
7 6 10 9 6 4 2 10 12 1 2 1 3 2 4 2 5 3 6 3 7	25
4 2 10 7 4 6 1 4 2 3	27

Problem C. Ball of Whacks

Source file name:	Ball.c, Ball.cpp, Ball.java, Ball.py
Input:	`Standard`
Output:	`Standard`

Danny has a new toy called Ball of Whacks. It's a rhombic triacontahedron (which we all know is a thirty sided polyhedron) made of small pyramid-shaped pieces which look like the following:

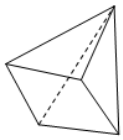


Figure 1: Game piece.

When all 30 pieces are all put together, they form the object shown in Figure 2a. Figures 2b–2f show one way to build a rhombic triacontahedron: at the bottom is a petal-shaped section of five pyramids (2b — note that only the base of each pyramid piece is shown); on top of that is a ring of five more pieces (2c); on top of that is a larger ring of ten pieces (2d), following by a five-piece ring (2e) and another petal-shaped section (2f). Figure 3 contains two other views showing how the pieces are connected to one another.

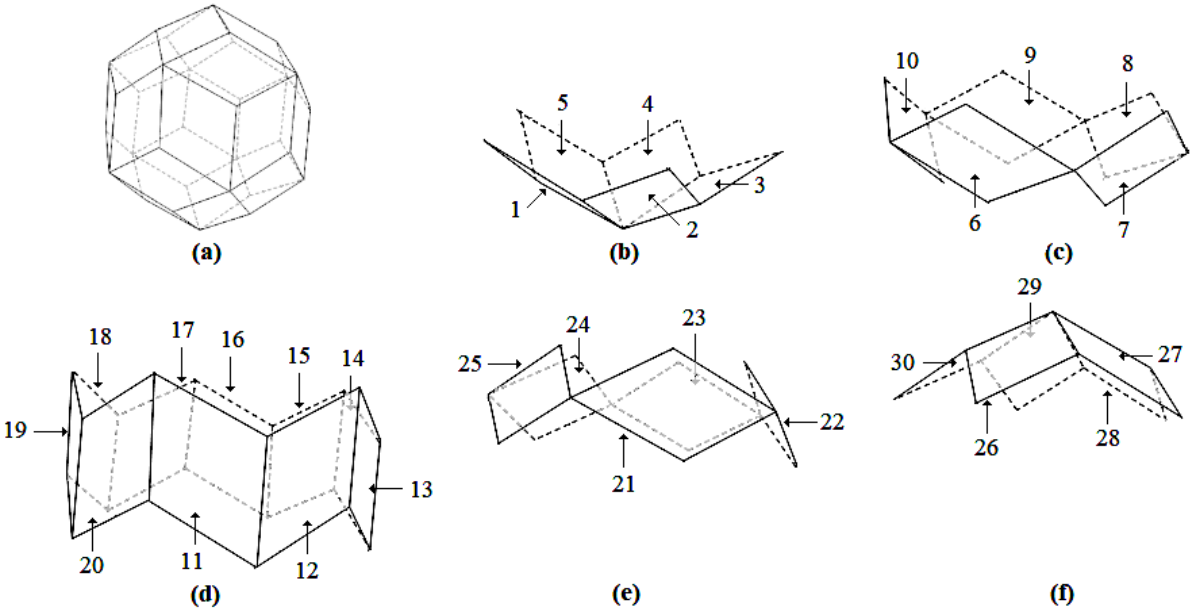


Figure 2: Construction of a Ball of Whacks.

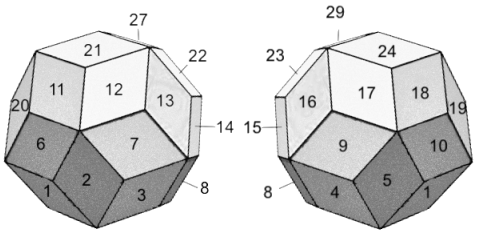


Figure 3: Numbering scheme of an assembled Ball of Whacks

Danny has several friends who also own Balls of Whacks, and when they get together... well, balls get whacked. The result are a number of sections of varying size and shape lying on the floor. When it's time to go home, Danny needs to know which sections fit together to make one complete Ball of Whacks. He would like you to help him.

Input

The input file consists of descriptions of three Ball of Whacks sections, each description on a single line. Each description starts with an integer m indicating the number of pyramid-shaped pieces in the section, followed by a set of numbers between 1 and 30 indicating the relative alignment of the pyramid pieces in the section, using the numbered locations in diagrams Figure 2b-2f. Each section description starts with a piece in position 1, though that piece — along with all other pieces in the section — may need to be moved in order for all three sections to fit together. The pieces in any section will be connected; i.e., you can get from any one piece in a section to any other by crossing over pieces which share an internal face.

Output

Output either `Yes` if the three sections can be fit together to get one Ball of Whacks, or `No` if they cannot be fit together.

Example

Input
25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 3 1 2 3 2 1 2
Output
Yes

Input
24 1 4 5 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 25 26 27 30 5 1 2 3 4 5 1 1
Output
No

Problem D. Tomb Hater

Source file name:	Tomb.c, Tomb.cpp, Tomb.java, Tomb.py
Input:	`Standard`
Output:	`Standard`

You've discovered the long-lost tomb of the great pharaoh T'om Bha Ter. After eluding a devious sequence of snake traps, boulder traps, ant traps, sand traps and speed traps, you've about had it with traps. Only one challenge remains between you and a hoard of priceless, museum-belonging-in artifacts. You enter a room from the North whose floor is a rectangular grid, each tile of which is inscribed with a glyph. A small plaque contains a list of glyph words (constructed from individual glyphs in a specified order) and the following brief set of instructions:

1. From any given tile you may only move one tile to the South, West or East.
2. You may not step on a tile more than once.
3. The path you take must spell out a sequence of complete glyph words from the given list (repetition of glyph words is allowed).
4. If multiple paths exist, you must find the path that steps on the least number of tiles.

Disobey any of these rules and you'll find yourself hurtling into a bottomless abyss. Since you brought your laptop, you immediately get to coding.

Input

Input starts with a line containing three positive integers m n k ($m, n, k \leq 50$), where m and n are the dimensions of the rectangular grid and k is the number of glyph words. The next m lines contain n upper-case letters, separated by single spaces, representing the rectangular grid. The next k lines each contain a glyph word, also consisting entirely of capital letters. Each glyph word will contain at most 50 letters.

Output

If there is a route that starts at any Northernmost tile, ends at any Southernmost tile, and obeys all of the above rules, output the length of the shortest such route. Otherwise output `impossible`.

Example

Input	Output
3 4 1 X X P X X X A T X X X H PATH	4
5 5 2 X X P X X X X A T X X X H H X X M O X X X E X X X HOME PATH	8
5 5 2 X X P X X X X A T X X X H X X X M O X X X E X X X HOME PATH	impossible

Problem E. Gambling Game

Source file name:	Game.c, Game.cpp, Game.java, Game.py
Input:	`Standard`
Output:	`Standard`

The Ionian Commission on Procuring Cash has come up with a new gambling game to raise funds for the government. The game is played as follows: Each week, the government will televise a set of m balls (numbered $1 \dots m$) being selected one at a time without replacement. Anyone who wants to play will have to buy a game card. Each card contains n squares (where $n \leq m/2$) and in each square are two numbers between 1 and m. No number appears more than once on a card. An example card is shown in Figure 4.

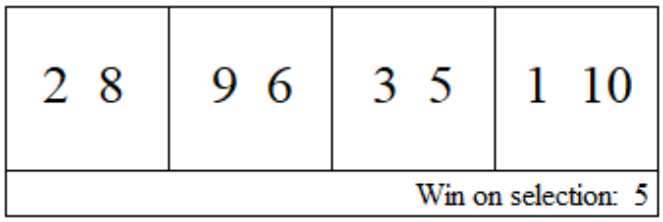


Figure 4: Example game card with $m = 10$, $n = 4$ and $p = 5$.

After each ball is selected, players cover any square which contains that number (there will be at most one such square on any card). Each game card also has a number p printed on it, and a contestant wins if he or she covers all n squares after exactly p ball selections (i.e., prior to the p^{th} selection, they only had $n - 1$ squares covered). Before issuing cards to its citizens, the government would like to get an idea of the likelihood of winning for various values of m, n and p so they can set up the payoffs appropriately. They have procured you to write a program to solve this problem.

Input

Input consists of a single line containing 3 integers m, n and p, as described above, where $2 \leq m \leq 33$, $0 \leq n \leq m/2$ and $0 \leq p \leq m$.

Output

Output the probability of winning on the p^{th} selection as a fraction `x/y` in simplest form.

Example

Input	Output
10 4 5	8/45
10 4 3	0/1

Problem F. Growing Some Oobleck

Source file name:	Oobleck.c, Oobleck.cpp, Oobleck.java, Oobleck.py
Input:	`Standard`
Output:	`Standard`

Oobleck is a fascinating magical substance — leave it in sunlight and it grows in an ever-expanding circle. If two circles ever manage to touch each other, both circles magically merge into one combined circle with area equal to the total of the combined areas of both circles and center at the midpoint of the centers of the two intersecting circles. The expansion rate of this new circle is the maximum rate of both colliding circles. An example of this is shown in Figures 5 and 6.

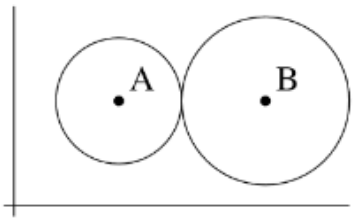


Figure 5: Circle A has center $(10, 10)$, radius 6 and area 36π; circle B has center $(24, 10)$, radius 8 and area 64π.

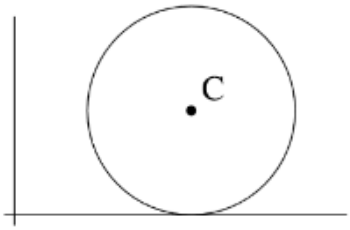


Figure 6: After they merge, they are replaced by circle C with center $(17, 10)$, radius 10 and area 100π.

After a new circle is created it may intersect other existing circles, causing a chain reaction of merges. This combination of an intersection followed by one or more merges is technically known as an *ooblection*. Ooblections occur instantaneously at the moment of the initial intersection even though they may contain a cascade of merges. If three or more circles are involved in a merge the resulting circle has its center at the average of the centers of all of the circles involved and the resulting area is the sum of all the areas of the circles involved. The expansion rate of the final circle after an ooblection is the maximum rate of any circle involved.

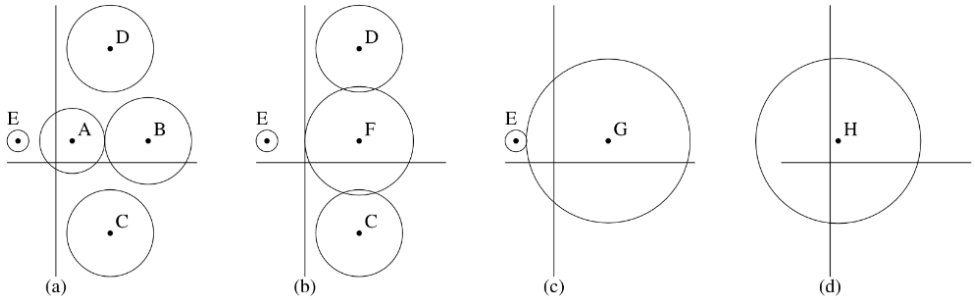


Figure 7: A five-circle ooblection.

An example of a five-circle ooblection is shown in Figure 7 (described in Example Input 2). At time $t = 1$ the circles have expanded to the point shown in Figure 7a. At that time circles A and B merge to form circle F (Figure 7b). Circle F intersects circles C and D, merging to form circle G (Figure 7c). Finally circle G intersects circle E resulting in the final circle H (Figure 7d). All of this takes place at time $t = 1$. Circle H has an expansion rate equal to the maximum expansion rate of circles A through E (using the values in Example Input 2 this maximum growth speed is 2).

Eventually as any set of circles grow, they will eventually ooblect into a single circle. Your task is to find the position and area of that circle the moment it is created.

Input

The input sequence begins with a positive integer n ($n \leq 100$), the number of circles. There then follows n lines, one for each circle. Each line consists of 4 numbers: x y r and s: x and y ($-10^9 \leq x, y \leq 10^9$) specifies the location of the radius of the circle; r ($1 \leq r \leq 10^6$) is the initial radius of the circle; s ($1 \leq s \leq 10^6$) is the expansion rate of the circle. No two circles listed in the input are touching. At most one ooblection begins at any given moment in time, and each ooblection starts with the touching of exactly two circles.

Output

Output two lines. The first line will contain two numbers x y, the location of the center of the final circle at the moment it is created. The second line will contain the radius of that circle. Your answers should be accurate to a relative or absolute error of 10^{-6}.

Example

Input	Output
2 1 1 1 1 5 1 1 1	3.00000000 1.00000000 2.82842712
5 3 4 5 1 17 4 7 1 10 -13 6 2 10 21 7 1 -7 4 1 1	1.50000000 4.00000000 15.23154621

Problem G. Noonerized Spumbers

Source file name:	Spumbers.c, Spumbers.cpp, Spumbers.java, Spumbers.py
Input:	`Standard`
Output:	`Standard`

Everyone has heard of spoonerisms, named after William Archibald Spooner, an Oxford professor who had a habit of swapping prefixes of words, often with comical results. "May I show you to your seat?" became "May I sew you to your sheet?" and "a crushing blow" became "a blushing crow."

Just imagine him as a student of arithmetic, occasionally swapping the prefixes of the numbers he was calculating with and then wondering why his equations never made any sense. For instance, when he writes:

$$92 + 2\,803 = 669\,495$$

what he really intended to write was:

$$6\,692 + 2\,803 = 9\,495$$

(He swapped prefixes "9" and "669" in the first and third numbers.) And when he writes:

$$6\,891 * 723 = 4\,979\,753$$

what he really intended to write was:

$$7\,291 * 683 = 4\,979\,753$$

(He swapped the prefix "72" with the prefix "68" in the first and second numbers.)

Grading homework from young Mr. Spooner is quite a challenge. Fleas pined a way to help!

Input

The input consists of a single line containing an expression of the form "$x + y = z$" or "$x * y = z$", where x, y, and z are positive integers less than 2^{31}. There will be single spaces surrounding the "+" and "*" operators and the "=" sign. The expression will not be a mathematically correct equation.

Output

Output a mathematically correct equation consisting of the input line modified by swapping proper prefixes of two of the three numbers x, y, z. (A proper prefix of a string s is a prefix that is neither empty nor equal to s.) Separate the numbers, operators, and the "=" sign with single spaces. All integers in the correct equation will be non-negative and less than 2^{31}. There is guaranteed to be only one possible correct equation that can be formed by swapping proper prefixes.

Example

Input	Output
`92 + 2803 = 669495`	`6692 + 2803 = 9495`
`6891 * 723 = 4979753`	`7291 * 683 = 4979753`

Problem H. Numble

Source file name:	Numble.c, Numble.cpp, Numble.java, Numble.py
Input:	`Standard`
Output:	`Standard`

Numble is a crossword game that is similar to Scrabble, but with numbers. The player has a hand of tiles with digits on them (1–9) and needs to make sequences of numbers using a subset of the tiles in hand to place on the board. The rules of a valid number sequence are:

- A valid sequence has to have at least two tiles in a straight line without spaces in a row or column (never diagonally) and must be connected to one or more tiles already on the board
- The numbers in a valid sequence must be arranged in non-increasing or non-decreasing order. Sequences "1467", "7641" and "766641" are valid, but "7461" is not.
- The total of all values in a valid sequence must be divisible by 3. This is calculated after any modifications given from bonus spaces (see below)

Like a Scrabble board, the Numble board contains four special types of spaces:

- *double* and *triple number spaces* double or triple the numerical value of a tile. This doubled or tripled value is counted in the total of the sequence to see if the total is divisible by 3.
- *double* and *triple sequence spaces* double or triple the entire value of the sequence (after individual tiles are doubled or tripled, if any). Note that a triple sequence space will always make any sequence of values a multiple of three.

It's possible for a sequence to hit multiple special spaces which compounds the bonus. For example, if a sequence covers both a double sequence space and a triple sequence space, the total value of the sequence is multiplied by 6.

A player's move consists of laying down a set of tiles from their hand onto empty spaces on the board across a single row or column. The row or column of tiles created cannot have an empty space in it, but can skip over tiles already on the board. Once the tiles are placed, all rows and columns of tiles that intersect with the newly placed tiles must form a valid sequence. A player scores for all sequences that are formed. Note that only sequences that place a tile over a bonus space scores the bonus — any already-placed tile that is covering a bonus does not score the bonus a second time.

Figure 8 shows a example board and a series of moves, where "d" and "t" represent double and triple number score spaces and "D" and "T" represent double and triple sequence score spaces.

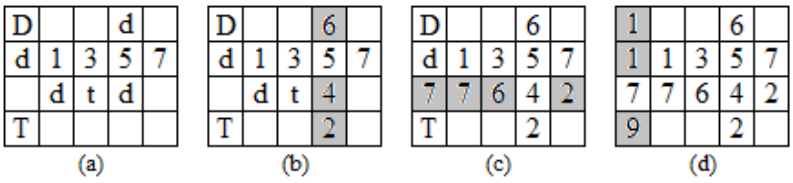


Figure 8: (a) Initial board. (b) After playing $6, 4, 2$. (c) After playing $7, 7, 6, 2$. (d) After playing $1, 1, 9$.

The initial board is shown in 8a. The first move places a $6, 4$ and 2 as shown in 8b scoring $6 \times 2 + 5 + 4 \times 2 + 2 = 27$ points. The next move places $7, 7, 6$ and 2 as shown in 8c, forming multiple valid sequences — one horizontal and three vertical. The horizontal sequence scores $7 + 7 \cdot 2 + 6 \cdot 3 + 4 + 2 = 45$

and the three vertical sequences score $1 + 7 \cdot 2 = 15$, $3 + 6 \cdot 3 = 21$ and $7 + 2 = 9$ (notice that all four sequences formed total to multiples of 3). Overall this play is worth $45 + 15 + 21 + 9 = 90$ points. The final move places $1, 1$ and 9 across both a double and triple sequence space. The score for the vertical sequence is $(1+1\cdot2+7+9)\cdot2\cdot3 = 114$ and the score for the horizontal sequence is $1\cdot2+1+3+5+7 = 18$ for a total score of 132 points (this corresponds to Example Input 1).

Given a board and a hand of tiles, what is the maximum possible score that can be made?

Input

The input begins with two positive integers r c ($r, c \leq 20$), the number of rows and columns on the board. After this are r rows of c characters, separated by spaces. Each character is either:

- a digit, representing a tile already on the board
- '`-`', representing an empty space
- '`d`' or '`t`', representing an empty double or triple number space
- '`D`' or '`T`', representing an empty double or triple sequence space

There will always be at least one digit on the board, and the rows and columns of digits of the board will be connected, and follow the non-decreasing/non-increasing sequence rules. (Sequences on the board may not follow the divisible by 3 rule because they may be covering bonus tiles).

There will then follow a positive integer t ($t \leq 10$), the number of tiles in hand, and then t digits between 1 and 9 specifying the tile values.

Output

Output the maximum score that can be made in a single turn using the board and tiles provided.

Example

Input	Output
4 5 D - - 6 - d 1 3 5 7 7 7 6 4 2 T - - 2 - 5 1 1 7 7 9	132
4 5 D - - 6 - d 1 3 5 7 7 7 6 4 2 T - - 2 - 5 1 1 6 7 9	150

Problem I. Pinned Files

Source file name:	Files.c, Files.cpp, Files.java, Files.py
Input:	`Standard`
Output:	`Standard`

You recently discovered a new feature in Visual Studio — pinned files! You have no idea what pinned files are good for — uou could hear a pin drop when you asked your friends about it — but that doesn't stop you from pinning and unpinning files all day and night.

Visual Studio maintains a list of the files you currently have open. The list starts with the pinned files and is followed by the unpinned files. Changing the order of the list is done not by click-and-drag operations but by toggling whether a file is pinned or not. If a pinned file is unpinned it is removed from the list and then added before the first unpinned file. If an unpinned file is pinned it is removed and then added after the last pinned file.

Consider the following example with four files labeled 1 through 4. The starting configuration is $2, 1, 4, 3$ and files 1 and 2 are pinned. If the desired ending configuration is $4, 2, 3, 1$ with only file 4 pinned, it can be accomplished with the following five toggle operations:

2*	1*	4	3	Initial State (* represents a pinned file)
2*	1*	3*	4	Pin file 3
2*	1*	3*	4*	Pin file 4
2*	3*	4*	1	Unpin file 1
2*	4*	3	1	Unpin file 3
4*	2	3	1	Unpin file 2

These five operations are the minimum number to get from the starting configuration to the ending configuration.

Given the initial list of files, determine the minimum number of moves to reach a different ordering of the list. The files are conveniently numbered 1 to n.

Input

Input consists of four lines. The first line contains two non-negative integers, p, the number of pinned files, and u, the number of unpinned files. The total number of files $n = p + u$ satisfies $1 \leq n \leq 100$. The next line consists of n numbers representing the list. The first p numbers will be pinned files and the next u numbers will be unpinned files. The next two lines will be the ending configuration in the same format as the first two lines. The number of files will be the same for the starting and ending configurations. The numbers 1 through n will appear once in each file list.

Output

Output the minimum number of moves to get from the starting configuration to the ending configuration.

Example

Input	Output
1 1 1 2 1 1 2 1	2
1 1 1 2 1 1 1 2	0
2 2 2 1 4 3 1 3 4 2 3 1	5

Problem J. Recycling

Source file name:	Recycling.c, Recycling.cpp, Recycling.java, Recycling.py
Input:	`Standard`
Output:	`Standard`

Heck S. Quincy (or Hex as his friends call him) is in charge of recycling efforts in the Quad Cities within the greater Tri-state area as well as the Twin Cities in the Lone Star State. One of the programs he oversees is the placement of large recycling bins at various locations in the cities. Transporting these bins is very expensive so he tries to keep them at any given location for as long as possible, emptying them once a week. He's willing to keep a bin at a given location as long as it is full at the end of each week.

Hex has very reliable estimates of the amount of recyclable materials (in cubic meters) that he can expect each week at each location. Given this information he would like to know when to install the recycling bin of an appropriate size to maximize the amount of material recycled. He will keep the bin at that location up to (but not including) the week when they don't expect it to be filled to capacity.

For example, suppose Hex has the following estimates for the next seven weeks: 2 5 7 3 5 10 2. Hex has several options for placing bins, including:

- A capacity-2 bin from week 1 until week 7, recycling 14m^3
- A capacity-5 bin from week 2 until week 3, recycling 10m^3
- A capacity-3 bin from week 2 until week 6, recycling 15m^3 (this is the maximum possible)

Hex would not place a capacity-5 bin from week 2 until week 6, since it would not be filled in week 4.

Input

Input starts with a line containing a single positive integer n ($n \leq 100\,000$) indicating the number of weeks which Hex has estimates for. Weeks are numbered 1 to n. Following this are n non-negative integers listing, in order, the amount of recycling expected for each of the n weeks. These values may be over multiple lines.

Output

Output three integers s e r where s and e are the start and end weeks for when to place the bin to maximize the total recycling and r is that maximum amount. If there are two or more time periods that lead to the same maximum amount, output the one with the smallest s value.

Example

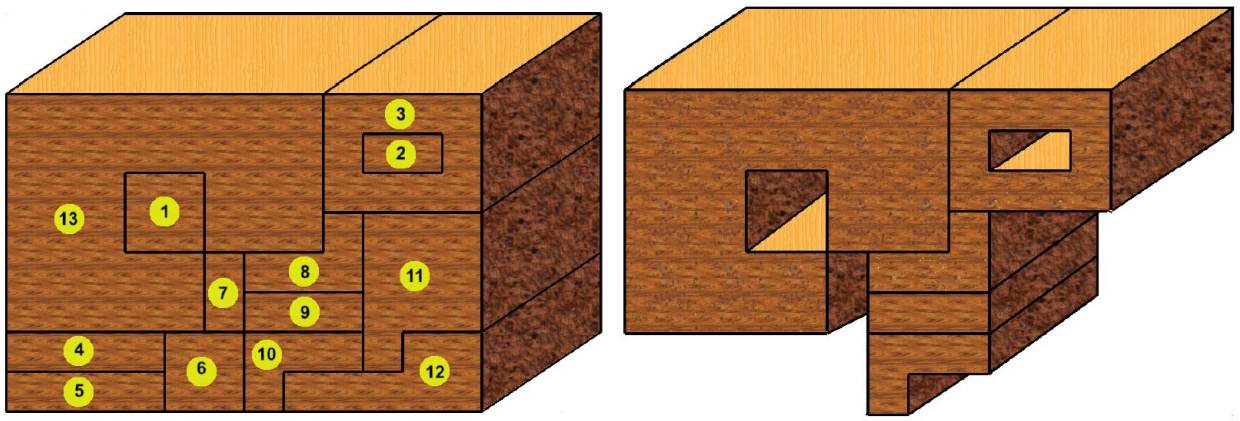
Input	Output
7 2 5 7 3 5 10 2	2 6 15
8 2 5 7 3 5 10 2 4	1 8 16

Problem K. Stable Table

Source file name:	Table.c, Table.cpp, Table.java, Table.py
Input:	`Standard`
Output:	`Standard`

Avant-garde carpenter Mort S. Tenon specializes in assembling furniture out of odd-shaped pieces of wood. Using strong glue and cleverly-hidden weights and floor anchors, Tenon can make tables that appear to defy gravity, as shown in Figure 9(b). He begins by creating a rectangular block composed of pieces of wood whose vertical cross-sections are rectilinear polygons, that is, polygons whose sides are either horizontal or vertical. Some of these pieces may have holes containing other pieces. Then he removes as many pieces as possible, leaving a structure consisting of the smallest possible number of stable pieces that include the entire top surface of the rectangular block. For aesthetic reasons (something having to do with patterns of wood grain), Mort never has more than two pieces forming the table's top surface.

Figure 9(a) shows an initial rectangular block with cuts. By removing the pieces numbered 1 and 2, 4 through 7, and 11 and 12, he produces the stable table shown in Figure 9(b). A piece is considered stable if either it is directly in contact with the floor or at least one of its horizontal edges lies above and is in contact with a horizontal edge of a stable piece. The region of contact between two adjacent pieces must have positive area in order for the contact to be stable; contact only along a corner is not sufficient.



(a) Original rectangular block (b) A five-piece stable table

Figure 9: Illustration of Example Input 1

Given a description of the initial rectangular block of pieces, help Mort determine a stable table with the minimum number of pieces.

Input

The first line of input contains two positive integers h and w ($1 \leq h, w \leq 100$), the height and width of the initial block of pieces. Each of the remaining h lines contains w positive integers describing which piece is associated with each 1×1 unit of the block. Pieces all have distinct numbers from 1 up through the number of pieces in the block. There can be at most 9 902 pieces since the first row of input can have at most two piece numbers. Pieces may have holes containing other pieces; each piece is connected using shared edges among the 1×1 units comprising the piece.

Output

Output a single integer equal to the minimum number of pieces in a stable table.

Example

Input	Output
8 12 13 13 13 13 13 13 13 13 3 3 3 3 13 13 13 13 13 13 13 13 3 2 2 3 13 13 13 1 1 13 13 13 3 3 3 3 13 13 13 1 1 13 13 13 8 11 11 11 13 13 13 13 13 7 8 8 8 11 11 11 13 13 13 13 13 7 9 9 9 11 11 11 4 4 4 4 6 6 10 10 10 11 12 12 5 5 5 5 6 6 10 12 12 12 12 12	5
3 4 8 8 8 8 5 6 7 8 1 2 3 4	2

Problem L. Statues

Source file name:	Statues.c, Statues.cpp, Statues.java, Statues.py
Input:	`Standard`
Output:	`Standard`

Central City is best known for its famous Statue Park. The park is laid out as a grid, with each grid square housing either a statue or some park water feature (see Figure 10a). The statues come in all shapes and sizes and therein lies the problem, as some of the statues are so large that they obscure the view of others from the main paths surrounding the park — roads N, E, S and W. The Park Statue Keeper is Milo D. Venus and he wants to move some of the statues so that visitors on two of these paths can see all of the statues. He realizes that one way to do this is to pick a corner of the park and then start placing the statues along the diagonals of the grid in order of the diagonals' distance from the selected corner, skipping over any water feature squares. The smallest statue would be placed in the first diagonal (assuming there is no water feature there). The next smallest statues will be placed in the second diagonal (this could be 0, 1 or 2 statues depending on the number of water feature squares in the diagonal). The next smallest statues are placed in the third diagonal, and so on. With this scheme though there are a lot of choices to make. For example, if you wanted the best viewing to be along roads N and W, you would use the diagonals in Figure 10b placing the smallest statue in diagonal a, the next smallest in diagonal b, the next three smallest in diagonal c, the next two smallest in diagonal d, and so on. If you wanted the best viewing to be along roads W and S you would use the diagonals in Figure 10c placing the smallest statue in diagonal a, the next two smallest in diagonal b, the next three smallest in diagonal c, the next smallest in diagonal d, and so on. Within any diagonal which can fit more than two statues the placement of those statues can be in any order, leading to even more choices.

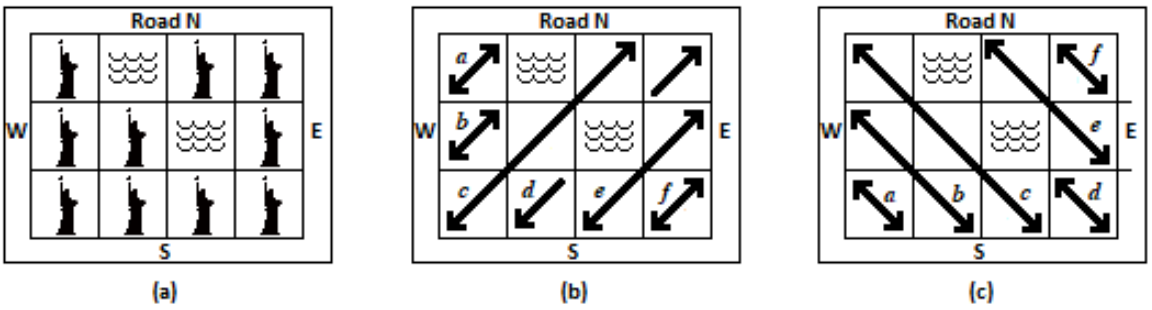


Figure 10: Statue Park. Figures (b) and (c) show two possible placements of the diagonals

Given all this flexibility it seems natural to pick the scheme that minimizes the number of statues that need to be moved, but which scheme is best is not immediately clear. Now Milo is a very competent administrator and very charming (he's downright disarming) but he's not all that great in solving problems like this. Can you carve out some time to help him?

Input

Input begins with two positive integers n m $(n, m \leq 50)$ specifying the number of rows and columns in the park grid. After this are n rows each containing m integers. The j^{th} value in the i^{th} row indicates what is located in row i, column j in the park grid. If this value is positive it indicates a statue of that height; if it is -1 it indicates a water feature. No two statues will have the same height.

Output

Output the minimum number of statues that need to be moved among all the various ways they can be placed with the scheme described above.

Example

Input	Output
3 4 2 -1 9 6 10 8 -1 4 1 3 5 7	5
3 4 30 -1 24 18 28 22 -1 16 26 20 14 12	0