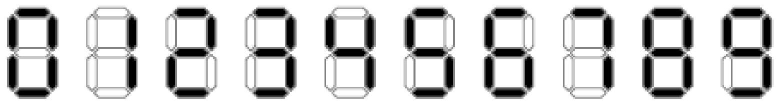
Problem A. Auxiliary Project

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

Anna has just finished her course project. She has a lot of seven-segment LED displays as leftovers and a small power source. Each display consumes power proportionally to the number of lit segments, e.g. '9' consumes twice more power than '7'.



Anna wonders what is the maximum possible sum of digits she is able to achieve, if her power source is able to light n segments, and she wants to light exactly n segments.

Input

The single line of the input contains one integer n — the number of segments that should be lit ($2 \leq n \leq 10^6$).

Output

Output a single integer — the maximum possible sum of digits that can be displayed simultaneously.

Example

Input	Output
4	4
7	11
6	14

Explanation

In the first example, a single '4' should be displayed ('7' has greater value, but has only three segments). In the second example '4' and '7' should be displayed, in the third one — two '7's.

Problem B. Bouldering

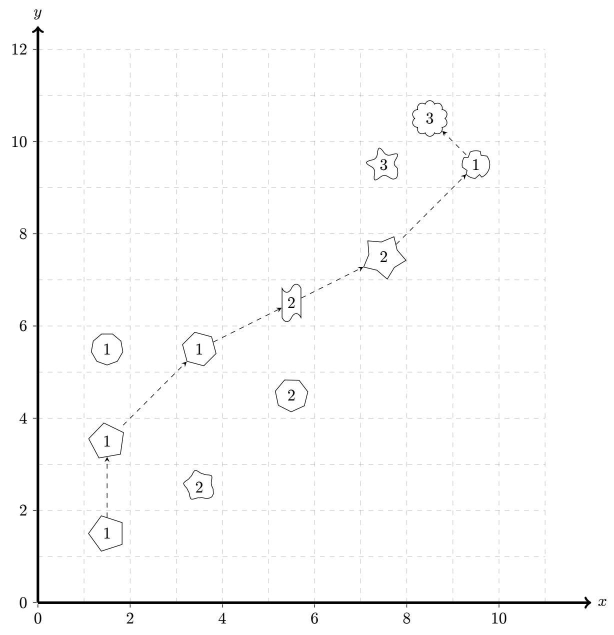
Source file name:	bouldering.c, bouldering.cpp, bouldering.java, bouldering.py
Input:	`Standard`
Output:	`Standard`

After a few particularly long afternoons of procrastinating in his box, playing video games all night long, Carl decided it was finally time to start his New Year's Resolution – going to the bouldering gym.

There, he took to one of the easier walls and tried to make his way up. Unfortunately, he could never quite reach the top, as he would always run out of stamina and fall down.

While climbing, he noticed the holds all have different shapes with some of them being much harder to hold than others, so gripping them uses up different amounts of stamina. Frustrated, he asks one of the regulars at the bouldering hall how to scale the wall – you. Show him the shortest way up that he can take without running out of stamina.

The bouldering wall is a rectangular grid of cells of size 1×1 where holds can be installed. For this problem we do not consider the varying sizes of the holds, so you can assume them to be the shape of a singular point exactly in the middle of the cell. Carl can only move from one hold to another if their distance (the Euclidean distance between the centres of the cells) does not exceed his arms' reach.



Example test case 1

Input

The input consists of:

- A line with four integers h, w, r and s ($2 \le h \le 25,\ 1 \le w \le 25,\ 1 \le r \le 3,\ 1 \le s \le 10^9$) where h and w are the height and width of the bouldering wall, r is the reach of Carl's arms and s is a numerical representation of Carl's stamina.
- h lines, each with w characters, describing the holds on the bouldering wall. Each character is either a digit c ($1 \le c \le 9$), which means that a hold with difficulty c is installed at this position, or "`.`", which means there is no hold installed.

The first line corresponds to the top of the bouldering wall and the last line to the bottom.

A sequence of holds is a valid route for Carl if the following conditions are satisfied:

- The route starts at the bottommost hold and ends at the topmost hold. There will always be a unique bottommost and a unique topmost hold, and these are guaranteed to be distinct.
- The sum of difficulty levels of the used holds is at most s.
- The Euclidean distance between any two consecutive holds along the route is at most r.

Output

Output the total length of a shortest route Carl can climb to reach the topmost hold without running out of stamina. Your answer should have an absolute error of at most 10^{-6}. If it is not possible for Carl to reach the top, output `impossible`.

Example

Input	Output

`12 11 3 11`
`...........`
`........3..`
`.......3.1.`
`...........`
`.......2...`
`.....2.....`
`.1.1.......`
`.....2.....`
`.1.........`
`...2.......`
`.1.........`
`...........`
	`13.543203766865055`

`8 16 3 15`
`......1.........`
`....1..1.1......`
`..2........1....`
`...2......1.....`
`.....4.1..2..1..`
`................`
`.......1........`
`................`
	`6.414213562373095`

`10 10 2 10`
`...2......`
`..........`
`...5.2....`
`..........`
`.....3....`
`....5.....`
`..2....2..`
`..1.......`
`....2.....`
`..1.......`
	`impossible`

Problem C. Circle Meets Square

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

We all know that you can't put a round peg in a square hole. Asking you to do so in this contest would be cruel and unusual punishment, which is banned by the Eighth Amendment to the United States Constitution. But, perhaps a more reasonable problem that the Framers of the Constitution never thought about is determining if a given circle and square have an intersection of positive area (overlap), or touch (share a point in common), or don't touch at all.

The Framers of the US Constitution and the UCF Programming Team coaches would like to know, given a circle and a square, do the two overlap in some positive area, touch, or don't touch at all. Help them out by writing a program to solve the problem!

Given the description of a square and circle in the Cartesian plane, determine if the intersection between the two has positive area (overlap), is a single point (touches) or doesn't exist.

Input

The first line of input contains three integers: x $(-1000 \leq x \leq 1000)$, y $(-1000 \leq x \leq 1000)$, and r $(0 < r \leq 1000)$, representing the x and y coordinates and radius, respectively of the circle.

The second line of input contains three integers: t_x $(-1000 \leq t_x < 1000)$, t_y $(-1000 \leq t_y < 1000)$, and s $(0 < s \leq 1000)$, where (t_x, t_y) represents the coordinates of the bottom left corner of the square and s represents the side length of the square. The square's top right corner is $(t_x + s, t_y + s)$, so that its sides are parallel to the x and y axes.

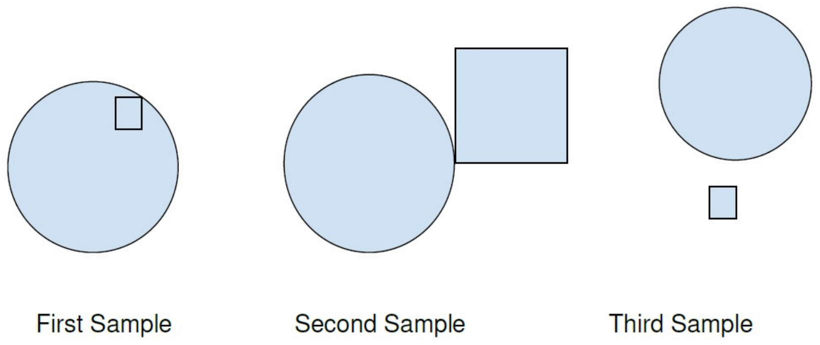
Output

If the circle and square **don't touch**, output 0 (zero). If they **touch** at a single point, output 1 (one). If they **overlap** in positive area, output 2 (two).

Example

Input	Output
0 0 5 2 3 1	2
0 0 5 5 0 6	1
0 5 4 -1 -1 1	0

Pictures of the Sample Input:



Problem D. Singin' in the Rain

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

During the time of the 2016 UCF Local Programming Contest, Arup's younger daughter Anya (3 years old at the time), made Arup incessantly listen to Taylor Swift's song "Wildest Dreams". A full year later, we are proud to report that Anya's listening habits have matured greatly. Rather than wanting to hear the same song over and over again, Anya has embraced an embryonic notion of diversity in music. Now, she wants to hear various different tracks, in sequence, **all from the same CD**.

This year, it turns out that Anya's favorite CD is "Singing in the Rain", which her older sister Simran obtained when she performed in the "Singing in the Rain" production at a local theater. Whenever Anya is in the car with Arup, she'll listen to a track and then call out the number of the next track that she wants to listen to. The problem is that Arup's car has a rather primitive CD player:

- If track number k has completed, then track $k + 1$ will play. If track k is the last track on the CD, the CD will wrap around and track 1 will play.
- Arup can also change tracks by pressing a forward button. If track number k has completed, if Arup presses the forward button, then track $k + 2$ will play. If k is the last track and Arup presses forward when k finishes, the CD will wrap around and track 2 will play next. If k is next to the last track and Arup presses forward when k finishes, track 1 will play next.
- Arup can also change tracks by pressing a backward button. If track number k has completed, if Arup presses the backward button, track number k will play again. If k is the first track and Arup presses the backward button twice right after track 1 completes, the CD will wrap around and track t will play next, where t is the number of tracks on the CD.

This means Arup is pressing either the forward or backward button a great deal. Help him minimize the number of times he presses the buttons.

Given the number of tracks on Anya's favorite CD and the sequence of tracks Anya wants to be played, determine the minimum number of button presses Arup must make to get the appropriate sequence of songs played. For the purposes of this problem, assume that at the very beginning the CD player is cued up to play the first song in the sequence (the first song Anya wants to be played), so Arup does not have to press any buttons for the first song in the sequence to be played, i.e., the first time any buttons might have to be pressed is in between the first and second songs in the sequence (after the first song in the sequence plays).

Input

The first input line contains a positive integer, n, indicating the number of test cases to process. Each test case starts with two space separated integers on a single line: t ($1 \le t \le 10^9$), the number of tracks on Anya's favorite CD and s ($1 \le s \le 1000$), the number of songs Anya would like to listen to from the CD. The second line of each test case contains s space separated integers, representing the sequence of tracks from the CD that Anya would like to listen to. Each of these will be in between 1 and t, inclusive.

Output

For each test case, output a single integer on a line by itself indicating the minimum number of button presses Arup can use to play the desired sequence of songs from the CD.

Example

Input	Output
3	73
68 6	3000000000
67 57 66 67 48 15	1
1000000000 7	
1 500000002 3 500000004 5 500000006 7	
3 3	
3 1 1	

Explanation

In Case #1, we first push the backwards button 11 times after 67 completes to cue up 57. Then we press the forward button 8 times to get 66 cued up after 57 finishes. After 66 finishes, we press no buttons and let 67 play, followed by pressing the backward button 20 times to arrive at track 48. Finally, we can either press the backward button 34 times to get to 15 or the forward button 34 times to get to 15. Thus, the minimal number of button presses is $11 + 8 + 0 + 20 + 34 = 73$.

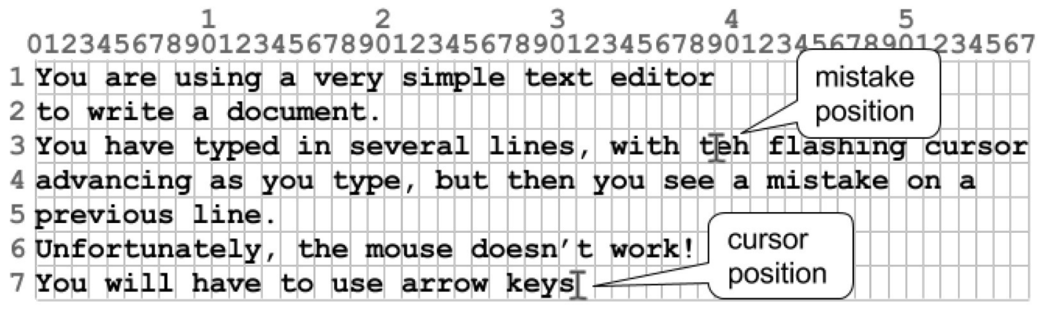
In Case #2, we must either press the forward or backward button $500,000,000$ times between each pair of consecutive songs on the list.

In Case #3, we must press the backward button once in between track 1 playing the first time and the second time.

Problem E. Editor Navigation

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

You are using a very simple text editor to create a document. You have typed in several lines, with the flashing cursor advancing as you type, but then you see a mistake on a previous line. Unfortunately, the mouse doesn't work! You will have to press arrow keys to move the cursor back to the position where you can fix the mistake. Of course, you want to get to this position as quickly as possible.



```
          1         2         3         4         5
  01234567890123456789012345678901234567890123456789012345678
1 You are using a very simple text editor
2 to write a document.
3 You have typed in several lines, with teh flashing cursor
4 advancing as you type, but then you see a mistake on a
5 previous line.
6 Unfortunately, the mouse doesn't work!
7 You will have to use arrow keys
```

This simple editor uses a monospace font, so each character is exactly the same width. The cursor can be at the beginning of the line (before the first character), end of the line (after the last character), or at a horizontal position between two characters on a given line. The following keys can be pressed to move the cursor (each keypress is independent of any preceding keypress):

← (Left arrow key)	Moves the cursor one character to the left on the same line, unless the cursor is at the beginning of the line, in which case it moves to the end of the previous line. If the cursor is at the beginning of the line and there is no previous line, the cursor does not move.
→ (Right arrow key)	Moves the cursor one character to the right on the same line, unless the cursor is at the end of the line, in which case it moves to the beginning of the next line. If the cursor is at the end of the line and there is no next line, the cursor does not move.
↑ (Up arrow key)	Moves the cursor up to the previous line; keeps the same horizontal position, unless the previous line is shorter, in which case the cursor goes to the end of the previous line. If there is no previous line, the cursor does not move.
↓ (Down arrow key)	Moves the cursor down to the next line; keeps the same horizontal position, unless the next line is shorter, in which case the cursor goes to the end of the next line. If there is no next line, the cursor does not move.

Given the line lengths of a text file that was loaded in this simple editor, along with the current cursor position and a different, desired cursor position (e.g., to fix a mistake), you are to determine the minimum number of keypresses, using arrow keys only, required to move the cursor to the desired position.

Input

The first input line contains a positive integer, n, indicating the number of editor navigation scenarios to process. Each scenario will occupy exactly 4 input lines. The first line of each scenario contains an integer f ($1 \leq f \leq 120$), indicating the number of lines of text in the file that is loaded in the editor. The next input line contains f integers, s_1 to s_f, where each value s_i ($0 \leq s_i \leq 80$) indicates the number of characters on line i of the file; the values will be separated by exactly one space. A value $s_i = 0$ means that there are no characters on line i. The newline character (common character indicating end of a line) does not count as a character on the line.

The third input line of each scenario will contain two integers (separated by a space) providing the current cursor position in the file: r_c ($1 \leq r_c \leq f$) and c_c ($0 \leq c_c \leq 80$), where r_c represents the line of the file, counting from 1 (as with i), and c_c represents the horizontal position of the cursor on that line, 0 for the beginning (before the first character). It is guaranteed that the cursor position is valid, so if, for instance, $r_c = 17$ and $s_{17} = 56$, then $0 \leq c_c \leq 56$; the maximum value of c_c is the end of the line. The fourth input line of each scenario is similar to the third and indicates the desired cursor position to begin fixing the mistake, i.e., this line consists of two integers separated by a space, r_m ($1 \leq r_m \leq f$) and c_m ($0 \leq c_m \leq 80$). The constraints on the values for r_c and c_c also apply to r_m and c_m.

Output

For each scenario in the input, output a single integer on a line by itself indicating the minimum number of keypresses needed to move the cursor from its current position (r_c, c_c) to the desired position (r_m, c_m) for fixing the mistake.

Example

Input	Output

2
7
39 20 57 54 14 38 31
7 31
3 39
3
15 30 20
1 12
3 3
	21
	8

Explanation

For Case #1, one possible sequence for the minimum number of keypresses to move the cursor from its current position to the desired position is: Up, Up, Right, Up, Left, Up, Left 15 times.

Problem F. World Cup Fever

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

The 2018 World Cup was held recently in Russia. Some great soccer countries (e.g., Italy, Netherlands, Chile, USA) did not qualify for this World Cup. These countries have found out that they needed more effective passing.

Given the player positions for two teams, determine the minimum number of passes needed to get the ball from one player to another player. For the purposes of this problem, players do not change position, i.e., they do not move.

Player P_1 can pass the ball directly to P_2 if they are on the same team and no other player is in between the two players.

Let's assume:

- P_1 and P_2 are on the same team
- P_1, P_2, P_3 form a line with P_3 between P_1 and P_2
- There are no other players on the line formed by P_1, P_2, P_3

Then,

- If P_3 is on the other team, P_1 cannot pass the ball directly to P_2.
- If P_3 is on the same team, P_1 can pass the ball to P_3 to pass it to P_2.

Input

The first input line contains an integer, n ($2 \leq n \leq 11$), indicating the number of players on each team. The second input line contains $2n$ integers, providing the (x, y) coordinates for the n players on Team 1; the first integer on this input line is the x coordinate for Player 1, the second integer is the y coordinate for Player 1, the third integer is the x coordinate for Player 2, etc. The third input line provides (in a similar fashion) the (x, y) coordinates for the n players on Team 2. Assume that all coordinates are integers between 1 and 999 (inclusive) and that all players are on distinct locations, i.e., no two players occupy the same spot (point).

Assume Player 1 on Team 1 has the ball and wants to pass the ball to Player n on Team 1. Assume that any player can pass the ball any distance.

Output

The output consists of a single integer, indicating the minimum number of passes needed to get the ball from Player 1 on Team 1 to Player n on Team 1. If it is not possible to get the ball from Player 1 to Player n, print -1.

Example

Input	Output

3
10 15 13 17 10 19
10 17 16 17 13 19
	2

5
1 1 3 1 5 1 7 1 9 1
2 1 4 1 6 1 8 1 10 1
	-1

3
1 1 5 5 2 2
10 10 50 50 20 20
	1

Problem G. Videogame Probability

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

You have just joined the illustrious Ewokin guild (group of players) in your favorite video game. This guild is known for attempting the hardest raids the instant that they are released. For these new raids you need the best gear (items) in order to have a viable chance of success. With your skills as a programmer, you have managed to determine the current drop rates for different item types in the game (drop rate refers to the probability of catching an item when the item is dropped). Now you need to know how likely it is that you will gather your gear (items) and bring your guild one step closer to success.

Given the number of different item types in a game, how many of each item type you need, the probability of obtaining (catching) each item type when it drops, and the maximum number of possible attempts you have for catching the items, determine the probability that you will obtain the desired number of each item type.

Input

The first input line contains a positive integer, n, indicating the number of test cases to process. Each test case starts with an integer, g ($1 \leq g \leq 50$), indicating the number of different item types in the game. The following g input lines provide the information about the different item types. Each such line contains two values: an integer, c ($0 \leq c \leq 50$), representing how many of this item type is needed, and a floating point number, p ($0.0 \leq p \leq 1.0$), representing the probability of catching this item type when it drops. The last input line for a test case contains an integer, a ($0 \leq a \leq 10^4$), representing the maximum number of attempts you have for catching the items. Note that this last input represents the total number of attempts and not the attempts for each item type. An attempt can, of course, be used (applied) to obtain any item type.

Output

For each test case, print a floating point number on a line by itself, representing the probability that you will obtain the desired number of each item type. Output the results to 3 decimal places, rounded to the nearest thousandth (e.g., 0.0113 should round to 0.011, 0.0115 should round to 0.012, and 0.0117 should round to 0.012).

Example

Input	Output

4
2
3 0.5
3 0.3
20
4
2 0.75
1 0.01
2 1.0
3 0.8
25
2
1 0.5
1 0.3
2
2
50 0.4
50 0.3
250
	0.816
	0.153
	0.150
	0.036

Problem H. Maximum Non-Overlapping Increasing Subsequences

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

Given a list of integers, we can find an increasing subsequence from that list, i.e., select specific elements from the list where the selected numbers are in increasing order. There is a classic problem that asks to find the longest increasing subsequence, i.e., increasing subsequence with the maximum number of elements. In our problem, we are going to do a modified version of that problem. Instead of finding one increasing subsequence, we will find multiple increasing subsequences with the following two constraints:

1. The length of each subsequence should be at least of a given size k.
2. If a subsequence starts at index i and ends at index j, no other subsequence can start or end in the range i to j (inclusive), i.e., no other subsequence can have any elements in between index i and j (inclusive), i.e., the subsequences cannot overlap.

The objective is to find subsequences (satisfying the above two conditions) resulting in the maximum number of elements being selected, i.e., the total number of elements in all these subsequences combined is the maximum.

For example, consider the list 2 1 9 3 4 4 5 6.

If $k = 2$, we can get three non-overlapping increasing subsequences with maximum of 7 elements: [2, 9], [3, 4], [4, 5, 6].

If $k = 3$, we can get two non-overlapping increasing subsequences with maximum of 6 elements: [2, 3, 4], [4, 5, 6].

Given a list of n integers, determine the maximum number of integers you can include in the non-overlapping increasing subsequences for all k where $1 \leq k \leq n$.

Input

The first input line contains an integer, t ($1 \leq t \leq 50$), indicating the number of test cases to process. Each test case starts with an integer, n ($1 \leq n \leq 100$), indicating the number of integers in the list. The second line of each test case contains the list of n integers (each in the range of -10^6 to 10^6, inclusive).

Output

For each test case, output n integers (on a single line) indicating, respectively, the maximum number of integers in the non-overlapping subsequences for all k where $1 \leq k \leq n$.

Example

Input	Output

`3`
`8`
`2 1 9 3 4 4 5 6`
`2`
`1 1`
`3`
`1 2 3`
	`8 7 6 5 5 0 0 0`
	`2 0`
	`3 3 3`

Problem I. Rotating Cards

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

A magician has a stack of n cards labeled 1 through n, in random order. Her trick involves discarding all of the cards in numerical order (first the card labeled 1, then the card labeled 2, etc.). Unfortunately, she can only discard the card on the top of her stack and the only way she can change the card on the top of her stack is by moving the bottom card on the stack to the top, or moving the top card on the stack to the bottom. The cost of moving any card from the top to the bottom or vice versa is simply the value of the label on the card. There is no cost to discard the top card of the stack. Help the magician calculate the minimum cost for completing her trick.

Given the number of cards in the magician's stack and the order of those cards in the stack, determine the minimum cost for her to discard all of the cards.

Input

The first input line contains a positive integer, t, indicating the number of test cases to process. Each test case is on a separate input line by itself and starts with an integer, c ($1 \leq c \leq 10^5$), indicating the number of cards in the stack, followed by c labels for the cards in the stack (starting from the top going to the bottom). Each of these labels will be in between 1 and c, inclusive, and each label will be unique.

Output

For each test case, output a single integer on a line by itself indicating the minimum cost for the magician to complete her magic trick.

Example

Input	Output

2
5 3 5 1 4 2
3 1 2 3
	15
	0

Problem J. Electric Bill

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

To encourage customers to conserve energy (and protect the environment), the electric companies typically have a lower rate for the first 1000 kilowatt-hour (KWH) of electric usage and a higher rate for the additional usage (KWH is a derived unit of energy equal to 3.6 mega-joules).

Given the rate (per KWH) for the first 1000 KWH usage, the rate (per KWH) for the additional usage, and a customer's energy consumption, you are to determine the charges (bill) for the customer.

Input

The first input line contains two integers (each between 2 and 20, inclusive), indicating the rate/KWH for the first 1000 KWH and the rate/KWH for the additional usage, respectively. The next input line contains a positive integer, n, indicating the number of customers to process. Each of the following n input lines contains an integer (between 1 and 50000, inclusive), indicating a customer's energy consumption.

Output

For each customer, print the energy consumption, followed by a space, followed by the charges.

Example

Input	Output

`6 10`
`4`
`1000`
`1001`
`700`
`4800`
	`1000 6000`
	`1001 6010`
	`700 4200`
	`4800 44000`

Problem K. K-Item Shopping Spree

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

It's back to school time and you've won a shopping spree! It's got some interesting rules. There are n items to choose from and you must select a sequence of k items. Any of the n items can be selected any number of times. A shopping spree is identified by the indices of the selected items (and not by the item values) and two shopping sprees are considered to be different if any of the corresponding item indices are different. For example, if there are 5 items (indexed 1 to 5) and you must select a sequence of 3 of them, any of the following would be different valid shopping sprees: (1, 2, 1), (2, 1, 1), (2, 1, 3), (5, 5, 2), (3, 5, 4). Note, again, that the shopping sprees are identified by item indices and not item values, e.g., the first two sprees listed are different because the first item chosen in the two sprees are different. Thus, the same combination of items can result in different shopping sprees if chosen in different orders.

For this problem, you are to find the number of K-Item Shopping Sprees that have a specified total value.

Given the number of items available in a shopping spree, their values, the value of k, and a price target, X, calculate the number of possible shopping sprees of precisely k items that has a total value of exactly X. Since the number of possible sprees can be extremely large, calculate it modulo 997.

Input

The first input line contains a positive integer, c, indicating the number of test cases to process. Each test case starts with an integer, n ($1 \leq n \leq 10^4$), representing the number of items available in the shopping spree. Each of the following n input lines provides a floating point number, p_i ($0.01 \leq p_i \leq 500.00$) with two digits past the decimal point, representing the value in dollars of each item available in the shopping spree.

The next input line of each test case will contain two space separated integers: k ($1 \leq k \leq 10^4$), representing the number of items to be selected in sequence for the shopping spree, and q ($1 \leq q \leq 10^4$), representing the number of queries for the test case. Each of the following q lines will contain one query. Each query will be a single floating point number, t ($0.01 \leq t \leq 500.00$) with two digits past the decimal point, representing the value in dollars of a target value for the shopping spree.

Output

For each query in a test case, print (on a line by itself) the number of possible shopping sprees (modulo 997) that are equal in value to the query value. Leave a blank line after the output for each test case.

Example

Input	Output
3	0
5	2
0.07	4
0.08	4
0.12	
0.08	3
0.15	0
2 4	1
0.12	
0.19	27
0.15	
0.23	
2	
0.05	
0.08	
3 3	
0.18	
0.23	
0.24	
3	
1.00	
1.00	
1.00	
3 1	
3.00	

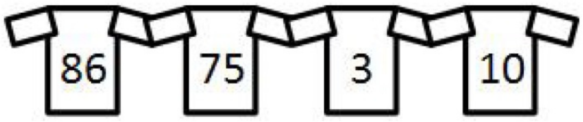
Problem L. Team Shirts/Jerseys

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

The Legendary Mathematician Travis and his friends are in a recreational (rec) league for competitive programming. Travis has n friends and his friends want to show a sense of unity for this competitive programming league, so they splurged on team shirts/jerseys each of which has an integer between 1 and 99 (inclusive) on the back. Each of Travis's friends have already selected their own (not necessarily distinct) jersey number so we have a list of n integers. Travis now needs to choose his jersey number to complete a list of $n + 1$ integers. Travis will also choose an integer between 1 and 99 (inclusive) and he does not have to pick a number different from what his friends have picked (i.e., he can choose to duplicate a jersey number).

Even though Travis can choose any jersey number, he wants to show why he is considered to be a legendary mathematician. Travis has a favorite positive integer and he wants to select his jersey number such that he would be able to pick a set of numbers from the list of $n + 1$ integers, concatenate this set of numbers together, and reach his favorite integer without extra leading or trailing digits (since Travis wants his favorite integer and not some other integer, he is trying to do so without any extra trailing or leading digits).

Travis has the fortunate option to choose his jersey number last. Now he just needs to determine if he can select some number that guarantees he can form his favorite integer. For example, suppose Travis's friends have selected the jersey numbers 3, 10, 9, and 86. Then, by selecting the number 75, Travis could form his favorite positive integer 8675310. Note that Travis didn't need to use the jersey number 9.



Note that if Travis chooses to use a jersey number, he must use the number completely, i.e., he cannot use just some of the digits in the number. For example, if he decides to use 75 in the above example, he must use "75"; he cannot just use "7" or use just "5". Note also that if he wants to use a particular jersey number multiple times, he must have multiple friends with that jersey number. For example, if he wants to use 75 multiple times, he must have multiple friends with the number 75 (he can, of course, pick 75 for himself as well to increase the number of occurrences of 75 in the list of $n + 1$ numbers).

We now have to break the secret that Travis is very temperamental! He tends to gain and lose friends very easily (so much for that whole unity thing); he also changes his favorite integer more often than the lights change at a busy intersection. For these reasons, Travis needs your help to write a program to solve this problem for a general set of friends and favorite numbers.

Given a set of positive integers representing friends' jersey numbers, and a favorite integer, determine if Travis can choose a jersey number (for himself) that can allow for the creation of his favorite integer via concatenation of zero or more friends' numbers and potentially his number. Additionally, since this is a rec league, the list of friends may contain duplicate jersey numbers, and Travis can choose an already chosen number for his shirt.

Input

The first line of input contains exactly one positive integer, t ($t < 10^9$), representing Travis's favorite integer. The second line of input contains exactly one positive integer, n ($n \leq 25$), representing how many friends Travis has today. The next input line contains n space separated integers which denote the jersey number chosen by each of Travis's friends. The jersey numbers will be between 1 and 99 (inclusive) and will have no leading zeroes, e.g., the input would have jersey number 7 but not 07.

Output

If Travis is capable of reaching his favorite integer with (or without) his set of friends, print 1 (one); otherwise output 0 (zero).

Example

Input	Output

707
2
7 24
	1

70707
2
7 7
	0

1122
3
21 1 23
	0

715 1 75	0

Problem M. Multiples

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

Sumon's 9-year-old son Samin likes programming. He becomes happy when his program compiles, runs, and gives correct output (at least for the small test cases). He does not bother much about making time-efficient solutions. It was good easy life for Samin, until the 2016 UCF Local Programming Contest.

His baby brother Saraf was born the very next day after the 2016 UCF Practice Local Contest, with a desperate interest to participate in the Real Local Contest. Unfortunately, for various reasons, we could not fulfill Saraf's wish to participate in the very special "`2 0 16`" ($= 2\ \ 0\ \ 2^{2*2} = 2\ \ 0\ \ (2*2)^2$) UCF Local Contest at the age of "6" $(2+2+2)$ days. Anyway, being born in the UCF contest week, Saraf is clearly destined to grow sophisticated taste for programming, and make his older brother's programming life harder. Now, Saraf wants his brother to run hard and medium range test cases at least for 20 iterations to test his code efficiency. If Samin's code does not run fast enough, Saraf gets angry, sits on the laptop, and keeps hitting it until the result is displayed.

Samin now needs to do a project for his school. To save Samin's laptop from Saraf's rage while running that program, you are asked to help Samin by writing the best possible time efficient solution to this problem.

Given two integers a and b, you are to find how many integers $1 \leq b_i \leq b$ are multiple of any integer $2 \leq a_i \leq a$. For example, if $a = 3$ and $b = 30$, we are interested in how many integers in the range of 1-30 are a multiple of 2 or 3; there are 20 such integers: 2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 26, 27, 28, 30.

Input

The first input line contains a positive integer, n $(1 \leq n \leq 100)$, indicating the number of queries to process. Each query will be on a separate input line and will contain two integers separated by a space: $2 \leq a \leq 130,\ \ 1 \leq b \leq 10^{15}$.

Output

For each query, output a single integer on a line by itself indicating the number of integers in the b range that are a multiple of any number in the a range.

Example

Input	Output

2
3 30
11 123
	20
	97