



XL Maratón Nacional de Programación Colombia - ICPC / ACIS / REDIS / CCPL 2026

Problems

(This set contains 13 problems; problem pages are numbered from 1 to 27)

A: Balanced System Reactor	1
B: Bankey	3
C: Into the Onion	4
D: Bingwhenever	7
E: Custom Keypad	9
F: Heist	11
G: Math United FC	14
H: HCubes Costs	16
I: Fair Workload Distribution	18
J: Courage is Contagious	20
K: Sample Median Preservation	22
L: Don't Ask Why 2D	24
M: Byte Flu	26

General Information. Unless otherwise stated, the conditions stated below hold for all the problems. However, since some problems may have specific requirements, it is important to read the problem statements carefully.

Program name. Each source file (your solution!) must be called

`<codename>.c`, `<codename>.cpp`, `<codename>.java`, or `<codename>.py`

as instructed below each problem title.

Input.

1. The input must be read from the standard input.
2. In most problems, the input can contain several test cases. Each test case is described using a number of lines specific to the problem.
3. In most cases, when a line of input contains several values, they are separated by single blanks. No other spaces appear in the input and there are no empty lines.
4. Every line, including the last one, has the usual end-of-line mark.
5. If no end condition is given, then the end of input is indicated by the end of the input stream. There is no extra data after the test cases in the input.

Output.

1. The output must be written to the standard output.
2. The result of each test case must appear in the output using a number of lines, which depends on the problem.
3. When a line of results contains several values, they must be separated by single spaces. No other spaces should appear in the output. There should be no empty lines.
4. Every line, including the last one, must have the usual end-of-line mark.
5. After the output of all test cases, no extra data must be written to the output.
6. To output real numbers, if no particular instructions are given, round them to the closest rational with the required number of digits after the decimal point. Ties are resolved rounding to the nearest upper value.

A: Balanced System Reactor

Source file name: `balsys.c`, `balsys.cpp`, `balsys.java`, or `balsys.py`

Author: Rafael García

In an experimental industrial facility, engineers work with chains of modular energy reactors. Each reactor module is assigned a positive integer power factor that determines its individual contribution to the system.

The *total linear output* of the system is the sum of all individual contributions from the modular energy reactors. The *compound amplification* produced by the system can be calculated as the product of all individual power factors from the modular energy reactors.

Due to strict safety regulations, the system is considered *perfectly balanced* only if the following condition is satisfied:

The total linear output of the system is exactly equal to the compound amplification produced by chaining all reactor modules together.

Formally, if the power factors of the n reactor modules are $x_1, x_2, \dots, x_n$, the system is perfectly balanced if and only if

$$x_1 + \dots + x_n = x_1 \cdot \dots \cdot x_n.$$

At the very beginning of the project the facility was very small. There were only 3 reactors with powerfactors 1, 2, 3. Clearly, the system was balanced, since $1 + 2 + 3 = 1 \cdot 2 \cdot 3$.

Now the project is mature enough to think in adding some more reactors in order to produce a greater total linear output. Anyway, engineers know that they want a balanced system, and they find that using several reactors with power factor equal to 1, they could achieve a balanced system. For instance, they could have 8 reactors with power factors 1, 1, 1, 1, 1, 1, 2, 8, the total linear output and the compound amplification are both 16.

However, safety analysts have also determined that many reactors with power factor 1 can lead to resonance instabilities, with eventual dangerous consequences. For this case, they prefer to have a power factor configuration like 1, 1, 1, 1, 1, 2, 2, 3. It is balanced and has only 5 reactors with power factor 1. Therefore, they require the following additional restriction:

For $n > 3$, exactly $n - 3$ reactor modules must share power factor 1.

Enforcing this restriction may mean that there is no way to balance the system, for some values of the system size n (the number of reactor modules). Anyway, given n , your task is to decide if configuring the system in a balanced way is possible and, if this is the case, tell how to do it.

Input

There are several cases to solve. Each case is defined with just a line with a single integer n , the number of reactor modules in the system ($4 \leq n \leq 10^7$). The end of the input is signaled with a line containing 0.

The input must be read from standard input.

Output

For each case, print a line with the values of the power factors different from 1, in ascending order, if the system may be configured in a balanced way. Otherwise, print IMPOSSIBLE.

The output must be written to standard output.

Sample Input	Sample Output
8	2 2 3
13	2 3 3
6	IMPOSSIBLE
0	

B: Bankey

Source file name: `bankey.c`, `bankey.cpp`, `bankey.java`, or `bankey.py`

Author: Rodrigo Cardoso

Pedro is a generally distrustful guy. He is always looking for unusual ways to hide his bank account passwords. Recently, he had the following idea to hide his ICPC Bank password: he saves a text file containing a sequence of N decimal digits, where $0 < N < 10^5$, and chooses an integer M such that $1 \leq M \leq N$. The numbers in the sequence are arranged in consecutive integer positions. The bank password is the largest possible sum of M digits in the sequence, in consecutive positions of the same parity. You can always think of it as adding M numbers, so that positions before the first digit or after the last digit of the sequence are assumed to hold 0 values.

For example, suppose that $N = 7$, $M = 3$, and the sequence is 7210396. The digits in odd positions are 7, 1, 3, 6, while those in even positions are 2, 0, 9. The sum ending at position 5 is $7 + 1 + 3 = 11$ and the sum ending at position 6 is $2 + 0 + 9 = 11$. The sum ending at position 7 is $1 + 3 + 6 = 10$. Padding with zeros is needed when fewer than M positions of the appropriate parity precede the ending position. For example, the sum ending at position 3 is $0 + 7 + 1 = 8$, where the initial 0 corresponds to a position before the beginning of the sequence. The key encoded by this sequence is 11.

Can you help Pedro develop a program that, given a sequence as described above, determines the numerical key encoded in it?

Input

The input consists of several test cases. Each test case begins with a line containing two integers N and M , where $1 \leq M \leq N < 10^5$. The next line contains a sequence of N decimal digits. The input end is signaled with two blank-separated zeros.

The input must be read from standard input.

Output

For each test case, print one answer line with an integer number corresponding to the bank key corresponding to the input data.

The output must be written to standard output.

Sample Input	Sample Output
7 4	17
7210396	11
7 3	
7210396	
0 0	

C: Into the Onion

Source file name: onion.c, onion.cpp, onion.java, or onion.py

Author: Rafael García

A group of biologists is studying the structure of an onion. They cut the onion through its center and obtain a two-dimensional cross-section. In the resulting image, they identify N cells. For each cell, they record its position (x_i, y_i) and a measured value w_i (that's not worth explaining here).

The biologists observe that the cells form several layers, from the outside towards the center of the onion. The outermost layer L_1 , consists of all the cells lying on the boundary of the smallest convex polygon that contains all N cells, including not only its vertices, but also any cell that lies exactly on one of its edges. After removing those cells of L_1 , layer L_2 is determined the same way from remaining cells: it consists of all the remaining cells lying on the boundary of the smallest convex polygon that contains them. This process is repeated, layer after layer, until no cell remains outside the last convex polygon obtained, that is, until every remaining cell lies on its boundary. That last layer is called the *heart* of the onion.

The layers are numbered from the outside towards the center, so L_1 is the outermost layer and L_K is the heart of the onion (see Figure 1). Let s_i denote the number of cells in layer L_i .

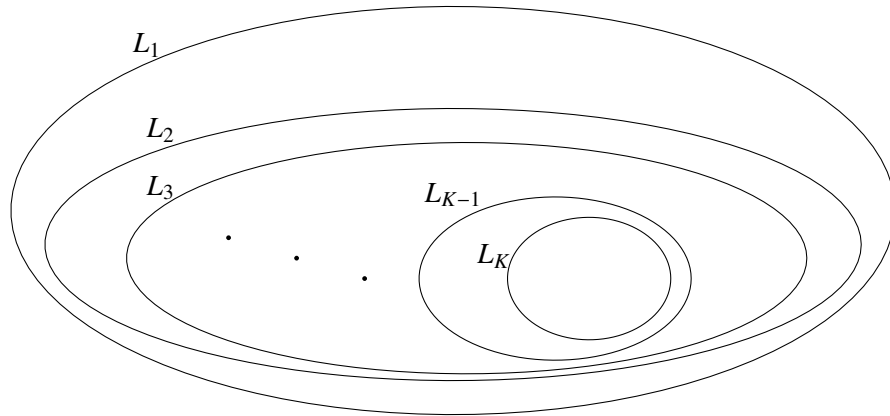


Figure 1: Example of a two-dimensional cross-section obtained by cutting an onion.

The density of a layer affects how easily a chemical marker can move through the onion. For a cell u in layer L_i , the biologists define the *transmission field* of u as a disk with center in u and radius $\frac{3}{2} \cdot d(L_i, L_{i+1})$ where $d(L_i, L_{i+1})$ is the minimum distance between the cells in the layer L_i and the cells in the layer L_{i+1} .

If v is a cell in the next layer L_{i+1} , the marker can pass from u to v exactly when v lies inside the transmission field of u , that is, when the distance between u and v is less than or equal to the radius of the transmission field of u . Note that the marker can only advance from a layer to the immediately next layer (see Figure 2).

The marker can be introduced into any cell of the outermost layer. From there, it repeatedly moves towards the center, always going from one layer to the next according to the rule above. Whenever the marker enters a cell, the biologists collect its measured value w_i (including the cell where the marker was initially introduced). The value of an experiment is the sum of the values collected along the entire sequence of cells visited by the marker. If there is more than one possible sequence, the biologists are interested only in the one with the maximum value.

For example, suppose 7 cells are identified (see Figure 3):

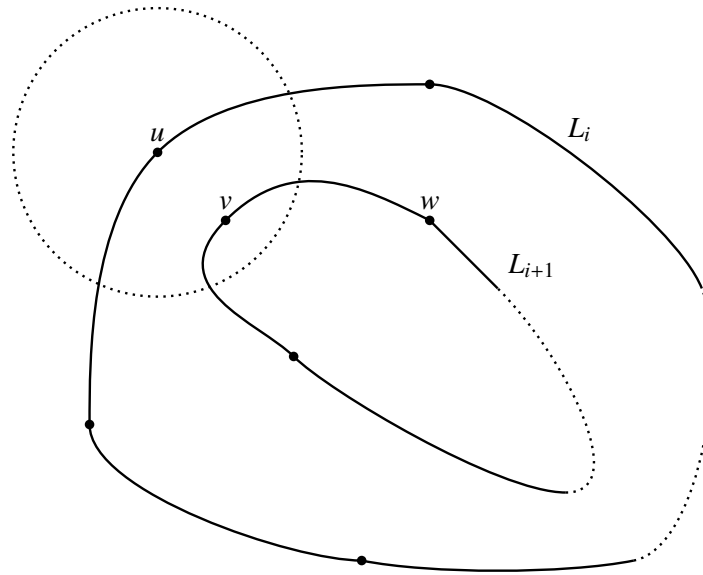


Figure 2: The cell v lies inside the transmission field of u , but the cell w lies out the transmission field of u . The transmission field of u is the circle with center in u and radius $\frac{3}{2} \cdot d(L_i, L_{i+1})$.

Cell number	x	y	w
1	2	2	1
2	-2	2	1
3	-2	-2	1
4	3	-2	1
5	2	0	5
6	0	1	3
7	-1	-1	8

The minimum distance between L_1 and L_2 is reached between cell 3 and the cell 7.

If the marker is introduced in c_1 , it can pass to c_5 . If the marker is introduced in c_3 , it can pass to c_7 . The best choice is to introduce the marker at c_3 : the value of the experiment is $w_3 + w_7 = 1 + 8 = 9$.

The team needs to determine the maximum value that can be obtained by introducing the marker at some cell of the outermost layer and successfully carrying it to one of the cells in the heart of the onion.

If no valid propagation reaches heart, the situation must be notified.

Input

There are several test cases.

Each test case starts with a line containing an integer N , the number of cells. Each of the next N lines contains three integers x_i , y_i and w_i describing a cell: (x_i, y_i) is the position in the xy -plane and w_i the measured value of the cell. No two cells share the same position. $1 \leq N \leq 3000$, $-10^8 \leq x_i, y_i \leq 10^8$ and $1 \leq w_i \leq 10^6$.

The input end is signaled with a line with $N = 0$ (that should not be processed).

The input must be read from standard input.

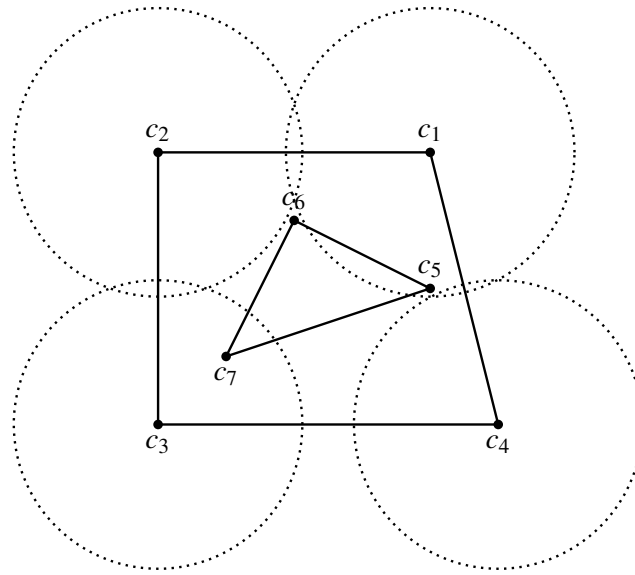


Figure 3: In the example the marker can be introduced into c_1 , c_2 , c_3 and c_4 . The marker can pass from c_1 to c_5 and from c_3 to c_7 .

Output

A single line per case: the maximum value that can be obtained by a valid propagation from the outermost layer to the heart of the onion. If no valid propagation reaches the heart, print -1.

The output must be written to standard output.

Sample Input	Sample Output
14 0 4 3 -4 3 1 -5 -1 1 -1 -3 3 3 -3 3 4 1 3 0 2 10 1 1 1 -3 2 2 2 -2 3 -2 0 3 0 1 100 0 0 30 -1 0 3 0	113

D: Bingwhenever

Source file name: `bingwhenever.c`, `bingwhenever.cpp`, `bingwhenever.java`, or `bingwhenever.py`

Author: Rafael García

A group of N friends wants to play bingo together. Unfortunately, they cannot agree on a date, a time, or even a place to meet next Saturday. They eventually come up with a brilliant solution: they will play Bingwhenever.

Random Enterprises Developing Intelligent Systems (REDIS) has developed Bingwhenever, a rather simple generalization of common bingo. Each friend receives a Bingwhenever card, i.e., a card with K different integer numbers on it, each one between 1 and M . Then REDIS produces a random sequence of B different numbers, each one also between 1 and M , just as the numbered balls drawn in a common bingo.

The sequence is sent to everyone by email, and each player uses a small program to follow the game and mark the numbers on his/her card as the balls are drawn. The game is won at step s of the game by those players that have marked all the numbers of his/her card when the s -th ball of the sequence is drawn, provided that nobody has done it before (at an earlier step).

In order to avoid discussions, at the end of the day another email from REDIS will announce the winner.

Your task is to help REDIS determining the winner or winners of the game.

For example, let's say Alice, Bob, and Charles are friends who are going to play Bingwhenever, and each card has 4 numbers. Alice's card is 3, 7, 10, 15; Bob's card is 3, 4, 7, 10, and Charles's card is 2, 7, 11, 15. Let's also say that the list of 8 balls is drawn in the following order: 4, 5, 7, 3, 2, 10, 11, 15. In that case, Bob is the first to complete his card and win the game (Alice marked 3 numbers and Charles only 2). But if the list of balls is 5, 7, 3, 2, 10, 11, 15, 4, then Alice and Charles complete their cards when the ball 15 is drawn, and Bob is left waiting for the ball 4.

Given the players' names, their Bingwhenever cards, and the order in which the balls are drawn, determine the names of all winners.

Input

The input consists of several test cases. Each test case starts with a line containing three integers N ($1 \leq N \leq 10^3$), K ($1 \leq K \leq 10^3$), and B ($K \leq B \leq 10^6$), representing the number of players, the number of numbers on each Bingwhenever card, and the number of balls that will be drawn (i.e., the length of the random sequence).

The next N lines describe the players. Each line contains a player's name followed by the K different integer numbers on their Bingwhenever card, each one of these between 1 and M , with $M = 10^9$. Player names consist only of lowercase English letters and contain no spaces. All player names are distinct.

The next line contains B different integers, representing the sequence of the numbered balls in the order in which they are drawn. All ball numbers are integers between 1 and M . Every number appearing in a Bingwhenever card also appears in the sequence of B drawn balls.

The last line of the input will be 0 0 0 and should not be processed.

The input must be read from standard input.

Output

For each case print a line with the names of all winners in lexicographical ascending order, separated by a single space.

The output must be written to standard output.

Sample Input	Sample Output
<pre>3 4 8 alice 3 7 10 15 charles 2 7 11 15 bob 3 4 7 10 5 7 3 2 10 11 15 4 3 4 8 alice 3 7 10 15 charles 2 7 11 15 bob 3 4 7 10 4 5 7 3 2 10 11 15 0 0 0</pre>	<pre>alice charles bob</pre>

E: Custom Keypad

Source file name: `customkeypad.c`, `customkeypad.cpp`, `customkeypad.java`, or `customkeypad.py`

Author: Edwin Niño

Early mobile phone keypads featured numeric keys that performed double duty for text entry. In classic multi-tap text entry systems (and predecessor layouts to predictive systems such as T9), the 26 letters of the Roman alphabet were distributed across a reduced number of physical buttons. Typically, 8 buttons (labeled 2 through 9) were used, with 3 or 4 letters assigned to each key in alphabetical order (for example, key 2 mapped to abc, key 3 to def, and so on).

To write a specific letter on such a keypad, a user must press the corresponding button repeatedly: pressing the button once yields its first letter, pressing it twice yields its second letter, pressing it three times yields its third letter, and so on. For instance, if key 2 contains abc, typing 'a' takes 1 press, typing 'b' takes 2 presses, and typing 'c' takes 3 presses.

A specialized hardware manufacturer designs custom mobile devices for different linguistic markets and domain-specific applications. Because typing efficiency depends heavily on the letter frequencies of the intended language, the engineering team evaluates each keypad layout against a comprehensive text corpus that serves as a representative sample of that language. Given such a representative sample text and the number of available letter buttons B ($1 \leq B \leq 26$), your task is to partition the 26 letters of the Roman alphabet ('a' through 'z') across the B buttons so that the total number of button presses required to type all letters in the sample text is minimized.

The keypad design must satisfy the following rules:

- Every one of the 26 Roman letters ('a' through 'z') must be assigned to exactly one button.
- Each of the B buttons (numbered 1 through B) must be assigned at least one letter.
- The partition must preserve alphabetical order across the buttons: all letters assigned to button i must be alphabetically smaller than (precede) all letters assigned to subsequent buttons $j > i$ (equivalently, every letter on a button must be alphabetically greater than all letters on preceding buttons).
- Within each button, letters must form an alphabetically consecutive range and are arranged in alphabetical order. For example, if $B = 3$, a valid partition could assign a-k to button 1, l-n to button 2, and o-z to button 3.
- The cost of typing a letter is its 1-based position within its assigned button.
- While the sample text may contain space characters, spaces are entered via a separate dedicated key (independent of the B letter buttons) and are **not** counted in the total keystroke cost.

Given the number of buttons B and the representative text sample, determine an optimal partition of the 26 letters across the B buttons that minimizes the total keystrokes.

Input

The input consists of multiple test cases.

Each test case consists of two lines:

- The first line contains a single integer B ($1 \leq B \leq 26$) --- the number of letter buttons available on the keypad.

F: Heist

Source file name: `heist.c`, `heist.cpp`, `heist.java`, or `heist.py`

Author: Julian Badillo

Danny O'Sea is pulling the biggest, most awesome heist of her life. *"This is the last one"* she promised you and your crew, *"after this one I will retire"*.

You and your team of expert hackers / infiltrators, of course, know better, but what can you do? The pay is so good.

*"Here's the plan: the **Ultra Luxurious McGuffin** is currently displayed in the **Very Secure Gallery**, in the **Extremely Securest Room**, which, as expected, is protected by a set of lasers that will trigger an alarm if interrupted.*

*"Now, I have acquired, at a great cost, a list of the lasers' positions in the room. So, what I need you to do is figure out what is the **safest location** in the room. We're pulling an *Ethan Hunt*".¹ She winks at you.*

You look at the map of the **Extremely Securest Room**. It is a rectangle of $W \times L$, dangerously criss-crossed with a list of Z lasers. Each laser is defined by its emitter and its receiver, two points $E = (ex, ey)$ and $R = (rx, ry)$, both of which lie on the walls of the room.

As you may know, from any heist movie, or well, your actual experience, you need to be as far as possible from any laser.

"Don't forget the walls" - says Danny - *"stay away from the walls."* Ok, as far as possible from any laser and any wall.

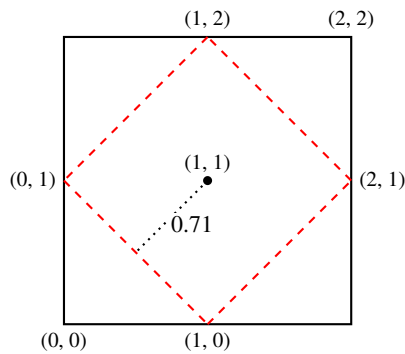
Your mission is to find the safest point to land in the Extremely Securest Room, that is, the point that is farthest from any laser path, and any wall.

"And the distance to the closest laser, or wall" - says Danny - *"we need to know the distance!"* Ok, the safest point and the distance to the closest laser or wall.

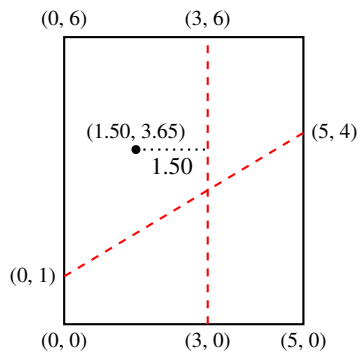
If more than one point is farthest from any laser path or wall, pick the one that lies in the largest area. If more than one such point lies in the largest area, or there are more than one largest areas, pick the one closest to the coordinate $(0, 0)$, why? Because that's the emergency exit.

Here are some examples:

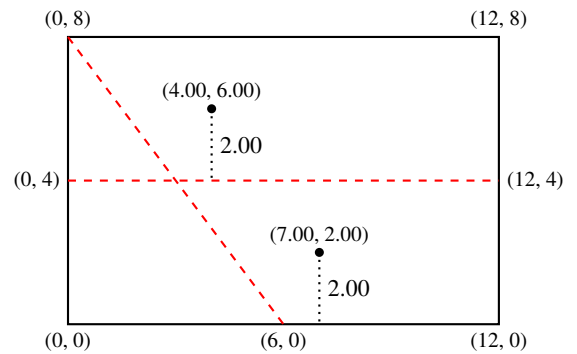
¹This is heist lingo for "hanging from the ceiling, snatching the actual prize, and pulling back"



Example 1



Example 2



Example 3

- On Example 1, it's evident the point in the center is farthest from all lasers and walls.
- On Example 2, there are multiple safest points in the larger region, the one depicted is the closest to (0, 0).
- On Example 3, there are multiple safest points in both the top and below areas, the one at the top is on the largest area, and closest to (0, 0).

Input

The input consists of several test cases. Each test case starts with three integers W, L, Z , ($1 \leq W, L \leq 1000$), ($1 \leq Z \leq 100$) separated by single spaces: W and L are the dimensions of the **Extremely Securest Room**. Z is the number of lasers installed.

Each of the following Z lines contains four integers ex, ey, rx, ry , ($0 \leq ex, rx \leq W$), ($0 \leq ey, ry \leq L$) separated by single spaces, which are the positions of the **laser emitter** and its corresponding **receiver**, which are each on a different wall of the room.

The last line of input will be `0 0 0` and should not be processed.

The input must be read from standard input.

Output

For each test case, print a single line with three decimal numbers sx, sy, d , the position of the **safest location** to land in the room, and the distance to the closest laser or wall, with two decimals, and an error of at most 10^{-2} .

The output must be written to standard output.

Sample Input	Sample Output
2 2 4	1.00 1.00 0.71
1 0 2 1	1.50 3.65 1.50
2 1 1 2	4.00 6.00 2.00
1 2 0 1	
0 1 1 0	
5 6 2	
3 0 3 6	
0 1 5 4	
12 8 2	
0 4 12 4	
0 8 6 0	
0 0 0	

G: Math United FC

Source file name: `mathunited.c`, `mathunited.cpp`, `mathunited.java`, or `mathunited.py`

Author: Juan Corena

Math United FC is the football club with the most powerful data analytics department in the entire world. For that reason, it is the most wanted team for starting and experienced players worldwide.

You are the head of the player trading division which is in charge of hiring players. This division starts the player transfer season with a budget $B \geq 0$ that should never become negative due to financial fair play regulations. Using this budget, the team can hire two kinds of players:

1. **Commercial players:** players that in their current state bring revenue to the club. Hiring them increases the budget of the team. A commercial player has a strictly positive value.
2. **Football prospects:** players that have amazing talent, but currently will not generate any revenue for the club if hired. In fact, hiring them reduces or does not affect the overall budget of the club. Therefore, a prospect has a zero or negative value.

Your work in the trading division consists of looking at a long list of players of the two kinds to decide whether to hire or not a player with the available budget. Your hiring strategy should maximize the number of football prospects that are hired. The market is too competitive, so that, if you wait, then the player will be lost to another team. With these requirements in mind, you look at the whole list and tell each player if he/she is hired or not, in the same order he/she appears in the list.

For instance, suppose a budget of 9 and a list of 6 players, say A, B, C, D, E, F , whose values are $-4, -7, 5, -6, -1, -4$, respectively. Since you must decide on-the-fly if you hire each player in the list, and you want to maximize the hired prospects, in this case you should hire the players A, C, E, F , i.e., 3 prospects (A, E, F) and a commercial one (C). Note that, as required, the budget is never negative as long as you take these hiring decisions in the same order of the original list.

Input

The input consists of several test cases. Each test case starts with a line with two number $0 \leq B \leq 10^9$ and $1 \leq P \leq 200000$ separated by a single space. These values represent respectively the initial budget and the number of players you need to process in order. The next line contains P numbers separated by a single space representing how hiring the i -th player $-10^9 \leq P_i \leq 10^9$ affects the budget B .

The last line of input will be `0 0` and should not be processed.

The input must be read from standard input.

Output

For each test case, print a single line with one number x , stating the number of prospects that can be hired with the initial budget.

The output must be written to standard output.

Sample Input	Sample Output
9 6	3
-4 -7 5 -6 -1 -4	4
10 6	3
-4 -3 5 -2 -1 -6	
0 5	
-5 10 -2 -2 -2	
0 0	

H: HCubes Costs

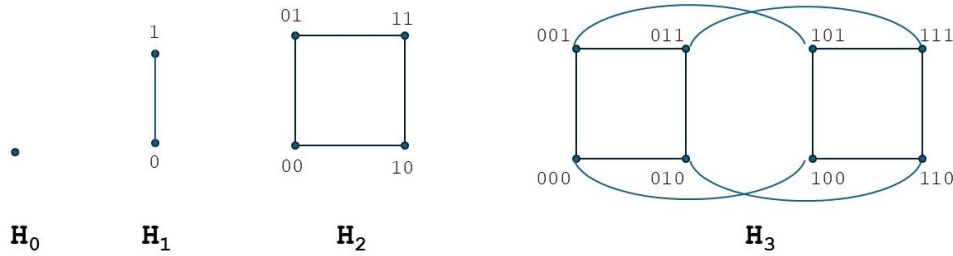
Source file name: `hcubes.c`, `hcubes.cpp`, `hcubes.java`, or `hcubes.py`

Author: Rodrigo Cardoso

An n -hypercube, $n \geq 0$, is a data structure with numerous applications in databases and communications. It is defined recursively as follows²:

- The 0-hypercube H_0 is the undirected graph with one vertex denoted by the empty string $\langle \rangle$ and 0 edges.
- For $n \geq 0$, the $(n + 1)$ -hypercube H_{n+1} is the graph consisting of two copies of H_n , with vertex sets $V_{n+1} = 0V_n \cup 1V_n$ and edges in each of the copies H_n , plus one edge $(0v, 1v)$ for each $v \in H_n$.

It is rather easy to see that V_n is exactly the set of all the binary strings with length n , for $n \geq 0$. The figure illustrates the representation of H_n , for $n \leq 3$:



InterConnectivity Per Cubes (ICPC) is a start-up that uses n -hypercubes in network designs. Nowadays they are solving core software to determine minimum costs for data communication between pairs of nodes of an n -hypercube network. More precisely, the cost of a communication between nodes $a, b \in H_n$ is the cost of a minimum cost path between the nodes. The cost of a path is the sum of the costs of its edges. For $u, v \in V_n$, so that there is an edge $(u, v) \in H_n$, let $c_n(u, v)$ the cost of this edge. ICPC models defines recursively the costs of the edges so that:

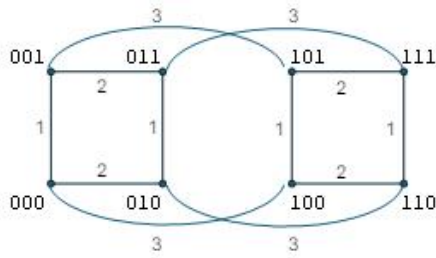
- $c_1(0, 1) = 1$
- $c_n(u, v) = c_n(v, u)$ for $u, v \in V_n, n > 0$
- $c_{n+1}(xu, xv) = c_n(u, v)$, if $x \in \{0, 1\}, u, v \in V_n, n > 0$
- $c_{n+1}(xu, yu) = n + 1$, if $x, y \in \{0, 1\}, u \in V_n, x \neq y, n > 0$

For instance, H_3 has the following arc costs:

Since every node has a unique n -bit string that denotes it, each node u can be denoted with the integer value that represents, for instance, in decimal notation. For example, in H_3 the nodes 110 and 010 can be denoted with the integers 6 and 2, respectively.

Your task is to help ICPC with the problem of determining the cost of a minimum cost path of a communication between two nodes of their hypercubes. For instance, in the case of the above described H_3 , the cost of a communication between nodes 6 and 2 is 3, between 1 and 7 is 5, etc.

²Binary strings are used to denote nodes of hypercubes. Some notation about binary strings: for $x \in \{0, 1\}$ and α a binary string, $x\alpha$ is the binary string adding the bit x before α . If A is a set of binary strings and $x \in \{0, 1\}$, then xA denotes the set of binary strings of the form $x\alpha$, where $\alpha \in A$.



Input

The input consists of several test cases. A case begins with a line with three integer numbers n , a and b , $0 < n \leq 20$, $0 \leq a, b < 2^n$, where a and b are decimal denotations of vertexes of the corresponding n -hypercube.

The input end is signaled with a line (that should not be processed)

0 0 0

The input must be read from standard input.

Output

For each test case, print one answer line with an integer number corresponding to the cost of a minimum cost path from a to b in the n -hypercube.

The output must be written to standard output.

Sample Input	Sample Output
3 6 2	3
3 1 6	6
2 3 3	0
1 0 1	1
0 0 0	

I: Fair Workload Distribution

Source file name: fairworkload.c, fairworkload.cpp, fairworkload.java, or fairworkload.py

Author: Edwin Niño

The logistics department of **ByteCorp** employs N workers, numbered consecutively from 1 to N . To handle incoming delivery requests across D consecutive days, management implements a daily **round-robin** assignment scheme designed to distribute the workload as fairly as possible.

On day i ($1 \leq i \leq D$), a total of M_i tasks need to be completed. To prevent favoritism, the system randomly selects a starting worker k_i ($1 \leq k_i \leq N$). The M_i tasks are then allocated one by one as follows:

- The first task is assigned to worker k_i .
- In general, after worker x receives a task, the next task is assigned to worker $(x \bmod N) + 1$ (that is, worker $x + 1$ if $x < N$, or worker 1 if $x = N$).

This process continues sequentially until all M_i tasks of day i have been assigned. Note that this scheme guarantees that within any single day, the difference in the number of tasks assigned to any two workers is at most 1.

At the end of the season, management wants to evaluate the accumulated workload. Given the log of the D days, you are asked to answer Q performance queries of the form:

“How many total tasks did worker w complete across days 1 through d (inclusive)?”

For example, consider $N = 5$ workers across $D = 3$ days:

- On day 1, $M_1 = 7$ tasks are assigned starting with worker $k_1 = 2$. The tasks are allocated in sequence to workers 2, 3, 4, 5, 1, 2, and 3. In total, workers 2 and 3 complete 2 tasks each, while workers 1, 4, and 5 complete 1 task each.
- On day 2, $M_2 = 3$ tasks are assigned starting with worker $k_2 = 4$, which are allocated to workers 4, 5, and 1. Workers 2 and 3 receive 0 tasks on this day.
- On day 3, $M_3 = 10$ tasks are assigned starting with worker $k_3 = 1$. Because $10 = 2 \times 5$, two full rounds are completed and every worker receives exactly 2 tasks.

If a query asks for the total tasks completed by worker 2 up to day 1 ($d = 1, w = 2$), the answer is 2. For worker 2 across days 1 through 2 ($d = 2, w = 2$), the answer is $2 + 0 = 2$. For worker 4 across days 1 through 2 ($d = 2, w = 4$), the answer is $1 + 1 = 2$. Finally, for worker 5 across days 1 through 3 ($d = 3, w = 5$), the total accumulated tasks is $1 + 1 + 2 = 4$.

Input

The input consists of several test cases.

The first line of each test case contains three integers N , D , and Q ($1 \leq N \leq 10^9$, $1 \leq D \leq 100\,000$, $1 \leq Q \leq 100\,000$) --- the number of workers, the number of days, and the number of queries, respectively.

The next D lines describe the daily assignments. The i -th of these lines ($1 \leq i \leq D$) contains two integers k_i and M_i ($1 \leq k_i \leq N$, $0 \leq M_i \leq 10^9$) --- the identifier of the starting worker and the number of tasks to be distributed on day i .

The next Q lines describe the queries. Each line contains two integers d and w ($1 \leq d \leq D$, $1 \leq w \leq N$) --- representing the day threshold d and the worker ID w .

The input is terminated by a line containing three zeros (0 0 0). This line should not be processed.

It is guaranteed that the sum of D and the sum of Q across all test cases in the input will not exceed 500 000.

The input must be read from standard input.

Output

For each query, print a single line with an integer: the total number of tasks assigned to worker w from day 1 up to day d , inclusive.

The output must be written to standard output.

Sample Input	Sample Output
5 3 4	2
2 7	2
4 3	2
1 10	4
1 2	
2 2	
2 4	
3 5	
0 0 0	

J: Courage is Contagious

Source file name: `courage.c`, `courage.cpp`, `courage.java`, or `courage.py`

Author: David Yepes

On the plains of *Tiě Shā* --- the Land of Iron Sand --- the wind does not blow: it howls. The Northern Barbarians have crossed the Frozen River for the tenth time in a generation, and this time they are not coming to plunder. They are coming to stay. General **Wei Zhan** gathers his soldiers at the edge of the abyss: behind them, the Han Empire; ahead, a tide of steel and fury. There is no wall. There are no reinforcements. There are only men --- and the question each one asks in silence as the horizon darkens: *How many of my brothers are going to fight?*

General Wei Zhan knows his men well. Each soldier i carries a bravery threshold k_i : at the start of each round he looks at *how many soldiers are fighting in total*, and he will be fighting in the next round if and only if that total is at least k_i . It does not matter *who* is fighting, only *how many*. At dawn, some number of warriors draw their swords; from that moment the count updates round by round --- soldiers whose threshold is met join the fight, soldiers whose threshold is not met stand down --- until the number of fighters no longer changes.

Because every soldier reacts only to the *total* number of fighters, the outcome depends solely on that number at dawn, never on the particular soldiers who started. If c soldiers are fighting in a round, then in the next round the soldiers fighting are exactly those with $k_i \leq c$; the process repeats until the count is stable.

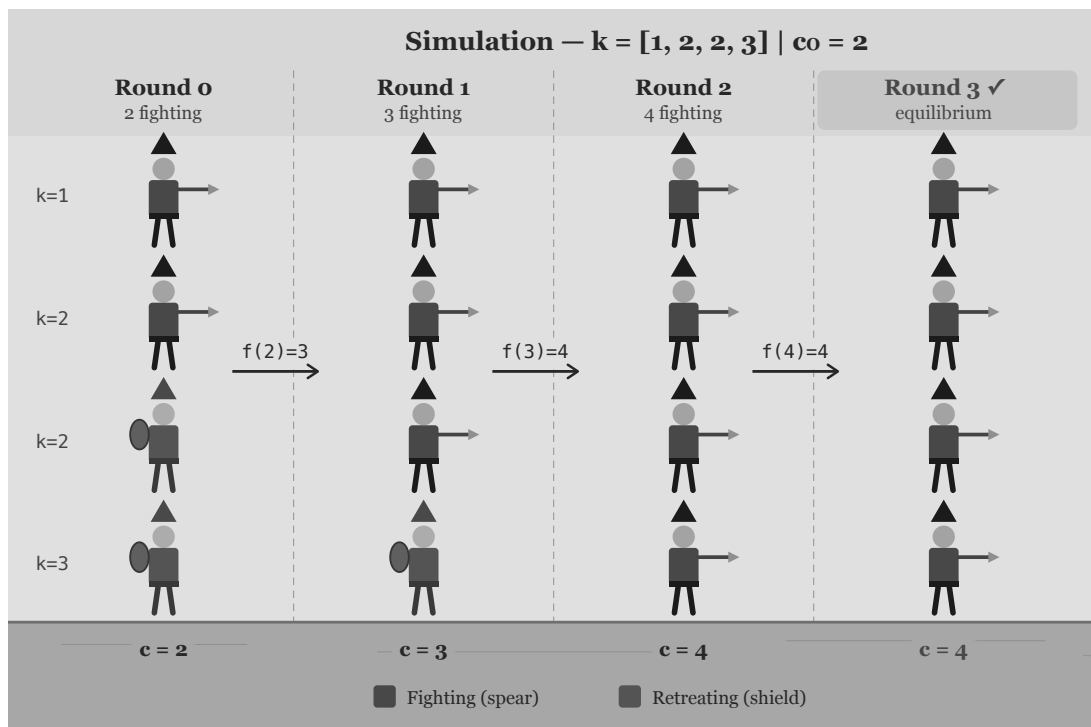


Figure 4: Simulation for $N = 4$ soldiers with thresholds $k = [1, 2, 2, 3]$ and initial count $c_0 = 2$. The number of fighters grows $2 \rightarrow 3 \rightarrow 4$ and then stabilizes: all 4 soldiers are fighting.

For example, consider $N = 4$ soldiers with thresholds $k = [1, 2, 2, 3]$ and an initial count of $c_0 = 2$ warriors at dawn. With 2 fighting, every soldier with threshold ≤ 2 fights next --- that is three soldiers (thresholds 1, 2, 2), so the count rises to 3. With 3 fighting, soldiers with threshold ≤ 3 fight --- all four --- so the count rises to 4.

With 4 fighting, again all soldiers with threshold ≤ 4 fight: still 4. The count is stable. The answer is **4**.

Before dawn, Wei Zhan must know the outcome of every possible initial formation. Given only the number c_0 of soldiers fighting at daybreak, how many will be fighting when the plains of Tiě Shā reach their equilibrium?

Input

The input consists of multiple test cases. Each test case begins with a line containing two integers N and Q ($1 \leq N \leq 10^5$, $1 \leq Q \leq 10^5$), the number of soldiers and the number of queries, respectively. The next line contains N integers $k_1, k_2, \dots, k_N$ ($0 \leq k_i \leq N$), the bravery threshold of each soldier. The following line contains Q integers c_0 ($0 \leq c_0 \leq N$), each representing an initial count of soldiers fighting at dawn. The sum of N over all test cases and the sum of Q over all test cases each do not exceed 10^6 . The input ends with a line containing 0.

The input must be read from standard input.

Output

For each query c_0 , print a single integer: the number of soldiers fighting when the battlefield reaches equilibrium. Print one result per line.

The output must be written to standard output.

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

K: Sample Median Preservation

Source file name: `samplemedian.c`, `samplemedian.cpp`, `samplemedian.java`, or `samplemedian.py`

Author: Edwin Niño

The National Metrology and Statistics Institute is conducting a quality control study on industrial sensor readings. They have collected a dataset consisting of N integer measurements $A_1, A_2, \dots, A_N$, where N is guaranteed to be an **odd** integer.

The true representative benchmark of this dataset is defined as its **median**: when all N measurements are sorted in non-decreasing order, the median is the unique value located exactly at index $(N + 1)/2$.

Due to computational constraints, the analysis team cannot always process the entire dataset. Instead, they choose a random sample of K measurements uniformly at random **without replacement**, where K is also an **odd** integer ($1 \leq K \leq N$). That is, each of the $\binom{N}{K}$ possible subsets of K distinct elements is equally likely to be selected. The sample median is similarly defined as the value at position $(K + 1)/2$ once the K chosen elements are sorted.

The team wants to evaluate the reliability of this sampling technique. What is the probability that the sample median equals the true median of the complete dataset?

Input

The input consists of several test cases.

Each test case begins with a line containing two odd integers N and K ($1 \leq K \leq N \leq 10^6$), the size of the dataset and the size of the sample, respectively.

The next line contains N integers $A_1, A_2, \dots, A_N$ ($-10^9 \leq A_i \leq 10^9$), the measurements in the dataset. Note that values may appear more than once.

The input terminates with a line containing:

0 0

which should not be processed.

It is guaranteed that the sum of N over all test cases in the input does not exceed 10^6 .

The input must be read from standard input.

Output

For each test case, print a single line with an integer representing the probability that the sample median equals the median of the complete dataset.

It can be proven that this probability can be expressed as an irreducible fraction P/Q , where $\gcd(Q, 10^9 + 7) = 1$. Output the integer value:

$$P \cdot Q^{-1} \pmod{10^9 + 7}$$

where Q^{-1} denotes the modular multiplicative inverse of Q modulo $10^9 + 7$, that is, the unique integer $0 \leq Q^{-1} < 10^9 + 7$ satisfying $Q \cdot Q^{-1} \equiv 1 \pmod{10^9 + 7}$.

The output must be written to standard output.

Sample Input	Sample Output
5 3	8000000006
1 2 3 4 5	1
5 3	1
1 2 2 2 3	2000000002
1 1	
42	
5 1	
10 20 20 20 30	
0 0	

Explanation of the sample cases:

- In the first test case ($N = 5, K = 3$, dataset $[1, 2, 3, 4, 5]$), the true dataset median is 3. Out of the 10 possible samples of size 3, exactly 4 samples ($\{1, 3, 4\}$, $\{1, 3, 5\}$, $\{2, 3, 4\}$, and $\{2, 3, 5\}$) have a median of 3. Thus, the probability is $4/10 = 2/5$. Since $5^{-1} \equiv 400\,000\,003 \pmod{10^9 + 7}$, the required output is $2 \cdot 400\,000\,003 \equiv 800\,000\,006 \pmod{10^9 + 7}$.
- In the second test case ($N = 5, K = 3$, dataset $[1, 2, 2, 2, 3]$), the true dataset median is 2. Every sample of size 3 chosen from this dataset contains the median 2 at its center position, so the sample median is always 2. The probability is $1 \equiv 1 \pmod{10^9 + 7}$.

L: Don't Ask Why 2D

Source file name: `dont.c`, `dont.cpp`, `dont.java`, or `dont.py`

Author: David Yepes

In the year 2089, the **Intergalactic Center for Protein Crafting (ICPC)** stands as humanity's most prestigious molecular engineering facility, nestled in the volcanic highlands of Iceland. Its crown jewel, *SynthCore-9*, can simulate any protein folding configuration in three-dimensional space with femtosecond precision. On a quiet Thursday morning, Dr. Aldric Novak --- ICPC's lead simulation engineer and self-proclaimed "*best coder in the northern hemisphere*" --- deployed what he described in the commit message as "*a small, totally harmless maintenance update*" to *SynthCore-9*. Within minutes, the simulator locked itself into **2D Planar Mode**: a forgotten legacy setting that forces every protein chain to fold like a path traced on an infinite grid, never crossing itself, never lifting off the plane. Before he could file an incident report, Dr. Novak noticed something unsettling: the proteins synthesized in this broken mode displayed *extraordinary* stability scores --- far beyond anything *SynthCore-9* had ever achieved in 3D. He immediately fired off a message to his colleague Dr. Sofia Reyes at **POLI** (*Protein Optimization Lab Initiative*): "*I need your brain. Right now. Don't ask why everything is 2D.*"

When Dr. Reyes arrived, she found *SynthCore-9*'s output logs scattered across three monitors. She spent the first hour quietly reading, occasionally scribbling on a whiteboard. By the second coffee, she had cracked the pattern. In 2D Planar Mode, each protein is a chain of N amino acids, each classified as either **H** (hydrophobic) or **P** (polar), with exactly B amino acids labeled H. The chain is laid out on an integer grid as a *self-avoiding walk*: starting from the origin, each amino acid occupies an adjacent cell (up, down, left, or right), never revisiting a cell. The stability of the protein is measured by its **hydrophobic contacts** --- pairs of H amino acids that are not consecutive in the chain yet find themselves occupying adjacent cells in the grid. "*More contacts, more stable. Simple,*" she said. To illustrate, she sketched the smallest interesting case on the whiteboard: a chain of $N = 4$ amino acids with a budget of $B = 2$ hydrophobic amino acids.

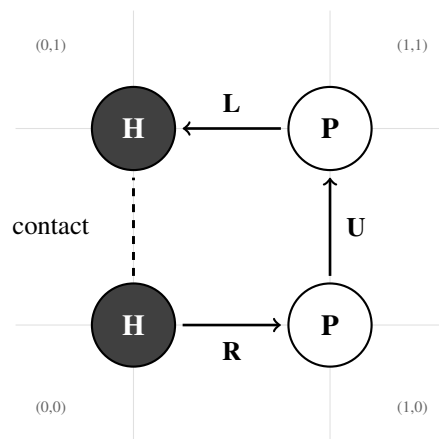


Figure 1: Sequence HPPH folded with directions RUL: 1 hydrophobic contact (dashed line). H nodes are drawn dark, P nodes light, so the figure remains legible in grayscale.

The sequence HPPH folded as RUL places the two H amino acids at opposite ends of the walk --- yet adjacent in the grid. Despite being separated by two P amino acids in the chain, they form one hydrophobic contact, the maximum achievable for these parameters. "*Someone has to find the optimal sequence and fold for every pending order,*" she said, already reaching for the keyboard.

Describing a protein. A protein is described by two strings:

- An **amino acid sequence**: a string of length N over the alphabet $\{H, P\}$ (hydrophobic / polar) with exactly B occurrences of H. The i -th character is the class of the i -th amino acid in the chain.
- A **folding**: a string of $N - 1$ *directions* that lays the chain on the grid. The first amino acid starts at the origin, and each subsequent amino acid is placed one cell away from the previous one, in the given direction. The four directions and their effect on the current cell (x, y) are:

$$R : (x+1, y) \quad L : (x-1, y) \quad U : (x, y+1) \quad D : (x, y-1)$$

The walk must be **self-avoiding**: no two amino acids may share a cell. Two amino acids are *adjacent in the grid* if their cells differ by exactly one step (one of the four directions above). A **hydrophobic contact** is a pair of H amino acids that are adjacent in the grid but *not* consecutive in the chain.

Dr. Novak's supervisor lands in Reykjavik in less than three hours. The pending synthesis queue holds several urgent orders, each arriving as a pair of integers N and B . For each order, determine the **maximum number of hydrophobic contacts** achievable, and provide an amino acid sequence and a folding that produce it --- all on a single line. The queue ends when $N = 0$ and $B = 0$. Dr. Reyes is waiting.

Input

The input consists of multiple synthesis orders. Each order is given on a single line containing two integers N and B , where

$$1 \leq N \leq 15, \quad 0 \leq B \leq N.$$

There are at most $T \leq 60$ orders before the terminating line. The input terminates when a line containing $0 \ 0$ is encountered.

The input must be read from standard input.

Output

For each order, print a single line with three space-separated tokens:

1. the maximum number of hydrophobic contacts;
2. an amino acid sequence of length N (a string of H/P with exactly B occurrences of H);
3. a folding: a string of $N - 1$ directions from $\{R, L, U, D\}$ (as defined above) that lays the sequence on the grid as a self-avoiding walk achieving that number of contacts.

If $N = 1$, print only the contact count and the sequence (there is no folding). Many optimal solutions may exist; **any** sequence and folding that achieves the maximum number of contacts is accepted.

The output must be written to standard output.

Sample Input	Sample Output
4 2	1 HPPH RUL
6 4	2 HHPPHH RRULL
3 2	0 HHP RR
1 1	0 H
0 0	

M: Byte Flu

Source file name: `flu.c`, `flu.cpp`, `flu.java`, or `flu.py`

Author: Rafael García

A computer virus known as Byte Flu has discovered a vulnerability in the identification system used by a company's network. The network consists of N devices, numbered from 1 to N , each with a unique string identifier. Byte Flu can move directly between two different devices if and only if the identifier of one can be transformed into the identifier of the other using at most K edit operations. Each edit operation consists of inserting one letter at any position, deleting one letter, or replacing one letter with a different one. Intermediate strings produced during a transformation do not need to be identifiers of network devices.

Some devices are *critical* because they provide services that are essential to the network. Consequently, critical devices cannot be disconnected.

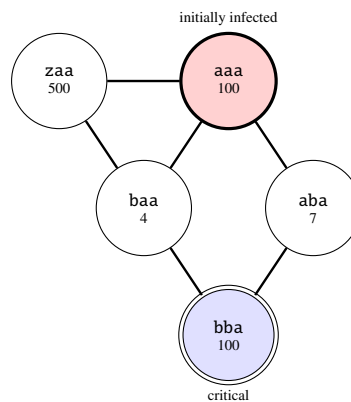
Initially, some non-critical devices are infected with Byte Flu. Whenever the virus reaches a device, it can move directly from that device to any other device whose identifier satisfies the condition above. This process continues, so the virus can eventually reach a device through a sequence of direct movements involving other devices. If a device is disconnected, the virus can neither enter it nor use it to reach another device. Initially infected devices cannot be disconnected.

Each device i has an associated disconnection cost c_i . To protect the network, the managers may disconnect any collection of devices that are neither critical nor initially infected. The cost of doing so is the sum of the disconnection costs of all disconnected devices. Their goal is to ensure that Byte Flu cannot reach any critical device from any initially infected device while minimizing the total disconnection cost. In some situations, protecting all critical devices may be impossible.

For example, consider a network with five devices whose identifiers and disconnection costs are

(aaa, 100), (baa, 4), (aba, 7), (bba, 100), (zaa, 500).

For aaa initially infected, bba critical, and $K = 1$, the network is depicted next:



Byte Flu can reach bba through the sequences

aaa $\rightarrow$ baa $\rightarrow$ bba

aaa $\rightarrow$ aba $\rightarrow$ bba.

If only baa is disconnected, the virus can still use the second sequence; if only aba is disconnected, it can still use the first. Disconnecting both devices prevents Byte Flu from reaching bba, at a total cost of $4 + 7 = 11$. No valid solution has a lower cost.

Given the device identifiers and disconnection costs, the initially infected devices, the critical devices, and the reachability threshold K , can you write a program that determines the minimum total cost required to prevent Byte Flu from reaching any critical device? If this is impossible, your solution must also report it.

Input

The input consists of several test cases. Each test case begins with a line containing four integers N ($2 \leq N \leq 1000$), K ($0 \leq K \leq 20$), I ($1 \leq I$), and C ($1 \leq C$ and $I + C \leq N$) representing the number N of devices, the reachability threshold K , the number I of initially infected devices, and the number C of critical devices.

The next N lines describe the devices. The i -th of these lines contains a string s_i ($1 \leq |s_i| \leq 20$) of lowercase English letters and an integer c_i ($1 \leq c_i \leq 10^6$), representing the identifier and the disconnection cost of device i , respectively. All device identifiers in a test case are distinct.

The next line contains I pairwise distinct integers $p_1, p_2, \dots, p_I$ ($1 \leq p_j \leq N$) identifying the initially infected devices. The following line contains C pairwise distinct integers $q_1, q_2, \dots, q_C$ ($1 \leq q_j \leq N$), identifying the critical devices. No device is both initially infected and critical. Although a disconnection cost is provided for every device, initially infected and critical devices cannot be disconnected.

The end of the input is indicated by a line with four blank-separated zeros and should not be processed.

The input must be read from standard input.

Output

For each test case, output one line containing a single integer: the minimum total cost of disconnecting devices so that Byte Flu cannot reach any critical device from any initially infected device. If no device needs to be disconnected, output 0. If it is impossible to prevent the virus from reaching all critical devices, output -1 .

The output must be written to standard output.

Sample Input	Sample Output
5 1 1 1 aaa 100 baa 4 aba 7 zaa 500 bba 100 1 5 4 2 1 1 aaaa 100 aabb 6 bbbb 100 zzzz 10 1 3 0 0 0 0	11 6