Problem A. Aquarium

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

The aquarium at which you work is hoping to expand its meagre selection of aquatic life, but lacks the funds to do so. You have been tasked to help promote the aquarium by taking photos of the two exhibits. Taking the first photo went swimmingly, because the catfish were very cooperative. For the piranhas, you have an arrangement of piranhas in mind that will look great on the photo. However, the only way to get the piranhas to move is by recklessly sticking your finger into the water to lure the piranhas. Your goal is to move the piranhas to the desired positions as quickly as possible without losing your finger in the process.

The piranha exhibit can be divided into positions $1, \ldots, n$ from left to right. The exhibit contains k piranhas and every position is occupied by at most one piranha. You can stick your finger into any unoccupied position. This will lure the nearest piranha to the left of your finger and the nearest piranha to the right of your finger. These piranhas will swim towards your finger, moving forward one position per second. All other piranhas simply stay in place. A piranha will bite your finger if it reaches the same position, so you must pull your finger away before this happens. Pulling your finger away and sticking it into a different position does not take any time.

For example, suppose there are piranhas at positions 2, 7 and 9. If you stick your finger into the water at position 4, the piranhas will be at positions 3, 6 and 9 after one second. You now have to pull your finger away to prevent the piranha at position 3 from biting your finger one second later. If you now stick your finger into the water at position 1, only the piranha at position 3 will move and will end up at position 2 after one second.

Input

- One line containing two integers n ($1 \leq n \leq 1000$), the number of positions, and k ($1 \leq k \leq n$), the number of piranhas.
- One line containing k integers $1 \leq p_1 < \ldots < p_k \leq n$, the current positions of the piranhas.
- One line containing k integers $1 \leq d_1 < \ldots < d_k \leq n$, the desired positions of the piranhas.

Output

Output the minimum number of seconds needed to get all of the piranhas at the desired positions. If it is impossible to do so, output "impossible".

Example

Input	Output
9 3 3 7 9 3 5 9	4
8 3 1 5 8 2 4 7	impossible
20 6 1 4 7 10 13 20 2 5 8 11 14 17	17

Problem B. Briefcases Full of Money

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

It is your birthday party and the six UCF programming team coaches arrive, each holding a briefcase containing money (gift) for you. The coaches were planning to give you the six briefcases but Dr. "O" points out that the team needs money to travel to World Contest Finals! So, you get to choose one briefcase, i.e., you are not getting all the briefcases (sorry). Each briefcase contains a stack of bills; each briefcase containing one of the 6 denominations \$1, \$5, \$10, \$20, \$50, \$100, i.e., first briefcase contains only \$1 bills, second contains only \$5 bills, third only \$10 bills, fourth only \$20 bills, fifth only \$50 bills, and sixth only \$100 bills. You, of course, want to pick the one with the highest total amount.

You can randomly pick one briefcase but, as a programmer, you trust your coding skills more than chance and decide to write a program to help you pick the briefcase with the highest amount.

Input

There is only one input line; it contains six integers (each integer between 1 and 1000, inclusive). These integers represent, respectively, the number of \$1, \$5, \$10, \$20, \$50, \$100 bills

Output

Output which briefcase to choose by printing the denomination in that briefcase (1, 5, 10, 20, 50, 100). If two or more briefcases have the highest total, pick (print) the one with fewest number of bills among those briefcases since that one is lighter!

Example

Input	Output
84 111 2 3 2 3	5
200 3 20 5 4 1	50
1000 2 2 2 2 2	1

Explanation of the second Sample Input/Output: Three briefcases (\$1, \$10, \$50) have the highest total (\$200) so the output is the briefcase with the fewest number of bills.

Problem C. Corrupt Judge

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

You are organising a programming competition in which the rank of a team is first determined by how many problems they have solved. In case of a tie, the team with the lowest time penalty is ranked above the other. However, contrary to the UKIEPC, the time penalty is equal to t if the *latest* accepted submission was submitted in the tth minute, or 0 if no problem was solved.

For example, if team A solved their first problem in the 5th minute, their second problem in the 10th minute and their third problem in the 60th minute, then their time penalty is 60. If team B also solved three problems, in the 30th, 40th and 50th minute, their time penalty is 50 and they would rank above team A.

The contest has finished and you would like to enter the final standings. However, due to a corrupted file you have lost part of the scoreboard. In particular, the column indicating how many problems each team has solved is gone. You do still have the time penalties of all the teams and know that they are in the right order. You also remember how many problems the contest had. You wonder whether, given this information, it is possible to uniquely reconstruct the number of problems that each team has solved.

Input

- One line containing two integers: n ($1 \leq n \leq 10^4$), the number of teams participating, and p ($1 \leq p \leq 10^4$), the number of contest problems.
- n lines with on line i the time score t_i in minutes ($0 \leq t_i \leq 10^6$) of the team that is ranked in the ith place.

A positive time score of t indicates that a team has submitted their last accepted submission in the tth minute. A time score of 0 indicates that a team hasn't solved any problem.

The input always originates from a valid scoreboard.

Output

If it is possible to uniquely reconstruct the scores of all the teams, output n lines containing the number of problems that the ith team has solved on the ith line. Otherwise, output "`ambiguous`".

Example

Input	Output
9 3 140 75 101 120 30 70 200 0 0	3 2 2 2 1 1 1 0 0
6 3 100 40 40 50 0 0	ambiguous

Problem D. Divvying Up

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

A solid competitive programming team can rack up a lot of prize money. Knowing how strong your team is, you are certain to win a lot of contests, so you had better sit down now and check that everybody will receive the same fair distribution of winnings.

You will participate in multiple contests, and at the end of each one receive a set amount of prize money. You can distribute any amount to each of the three members of your team each time, but by the end everyone must have the same amount of total winnings.

Can you distribute the winnings such that everyone gets an equal amount by the end?

Input

- One line containing the number of contests, n ($1 \leq n \leq 10^4$).
- One line containing the prize purse for each contest, $w_1 \ldots w_n$ ($1 \leq w \leq 10^5$).

Output

Output `yes` if the winnings can be distributed equally between three contestants, otherwise `no`.

Example

Input	Output
2 10 3	no
3 9 8 7	yes

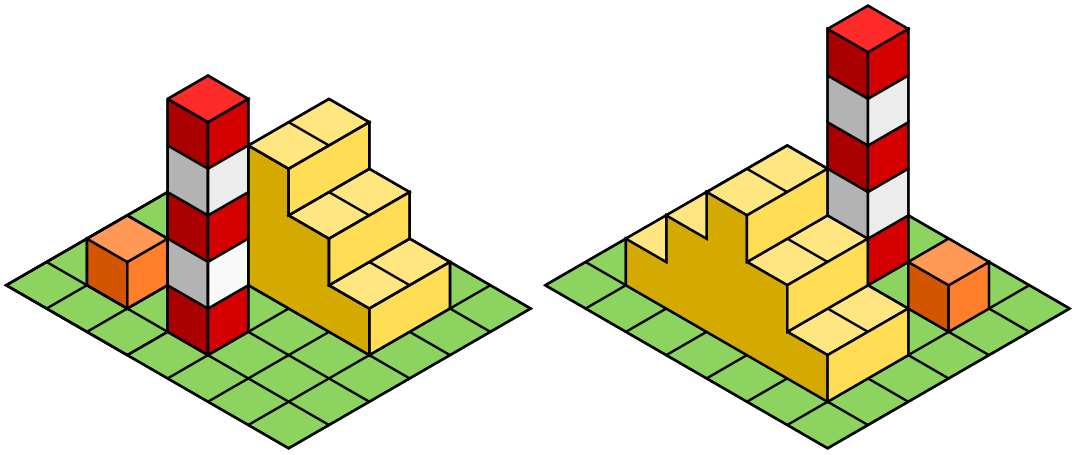
Problem E. Elevator Pitch

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

You are in charge of ensuring all building designs meet accessibility requirements. As law dictates, every part of your building should be reachable for wheelchair users, which means elevators will have to be installed. You are given the blueprints of the company's current project and have to determine the minimum number of elevators required.

The floor plan is laid out on a square grid and the blueprints tell you the number of floors above any given square. You can place an elevator at any square, which stops at all floors of that square. A wheelchair user can move up and down between floors using the elevators and can freely move to any of the four adjacent squares on the same floor. Buildings do not connect diagonally.

The image below shows the second sample input. Designs can consist of multiple buildings; this one contains three buildings. The design requires two elevators: one for the pyramid-shaped building and one for the tall tower. The small building of height one does not require an elevator, since it only has a ground floor.



A visualisation of the second example input.

Input

- One line containing integers h and w ($1 \leq h, w \leq 500$), the height and width of the grid.
- h lines of w integers each, where $x_{i,j}$ ($0 \leq x_{i,j} \leq 10^9$), the jth integer on the ith line, denotes the number of floors at position (i, j) of the grid.

Output

Output the minimum number of elevators you need to build to be able to reach every part of the building(s) in the grid.

Example

Input	Output
3 3 1 2 3 0 0 4 7 6 5	1
6 7 0 0 0 0 0 0 0 0 1 2 3 2 1 0 0 1 2 3 2 1 0 0 0 0 0 0 0 0 0 1 0 5 0 0 0 0 0 0 0 0 0 0	2
4 4 1 1 2 1 2 2 1 2 1 2 2 1 2 1 2 2	4

Problem F. Family Fares

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

After a long time apart, your family will gather next year for a reunion in an idyllic village in the centre of the country. Since everybody lives apart, most will need to travel by train.

You are in charge of finding the best deal on tickets. Everyone must take an optimal route, that is to say they may only travel a route if no other route is shorter.

Two types of ticket are available: *individual* or *group*. All tickets come with a start and destination between which to travel. Individual tickets are unlimited and the price is equal to the shortest distance in kilometres between stations.

Group tickets are more complicated. First, you may only buy at most one and it must be for a set list of people. There is no limit to the number of people named, but all must be present. The ticket is priced according to the number of named persons.

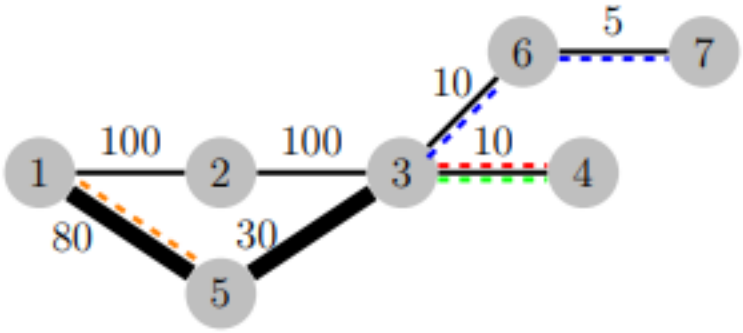


Figure F.1: Example 2. Group or individual tickets are shown by thick or thin lines, respectively.

Input

- One line with four integers: n ($2 \leq n \leq 1000$), the number of stations, m ($n - 1 \leq m \leq 10^5$), the number of connections between stations, p ($1 \leq p \leq 100$), the number of family members, and g ($1 \leq g \leq 10^6$), the cost per person of a group ticket.
- One line with p integers v_i ($1 \leq v \leq n$), meaning that family member i starts at station v_i.
- m further lines, each with three integers a, b, and c ($1 \leq a, b \leq n$, $a \neq b$, and $1 \leq c \leq 10^6$), indicating that there is a bidirectional connection between stations a and b with a length of c kilometres.

Each pair of distinct stations has at most one direct connection and every station can be reached from any other station. Station number 1 serves the idyllic village.

Output

Output the total amount you must spend so that every family member can travel from their starting station to the idyllic village.

Example

Input	Output
6 5 3 10 4 5 6 1 2 10 2 3 10 3 4 10 4 5 2 4 6 3	35
7 7 4 10 5 4 4 7 1 2 100 2 3 100 3 4 10 1 5 80 3 5 30 3 6 10 6 7 5	145
4 5 2 10 2 4 1 2 20 2 4 5 1 3 20 3 4 5 1 4 30	25

Problem G. Generators

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

The volcanic island of Fleeland has never had a proper electric net, but finally the administration of the island have agreed to build the island's power plants and network.

On the island's coast are its n cities. The administration has surveyed the cities and proposed m of them as possible locations for a power plant, with the ith proposal stating that the company can build a plant in city c_i for cost a_i.

These power plants are very modern and a single plant could power the whole island, but the volcano makes building power lines across the island a dangerous affair. For $1 \leq i < n$, the company can build power lines between cities i and $i+1$ for a cost of b_i, and between cities n and 1 for a cost of b_n. A city will receive power if it contains a power plant or is connected to a city with a power plant via power lines.

What is the cheapest way to power all the cities on the island?

Input

- One line containing two integers n $(3 \leq n \leq 10^5)$ and m $(1 \leq m \leq n)$, the number of cities and the number of possible locations for a power plant.
- Then follow m lines, the ith of which contains c_i $(1 \leq c_i \leq n)$ and a_i $(1 \leq a_i \leq 10^9)$, the ith possible location for a power plant, and the cost to build it.
- Then follows a line containing n integers b_i $(1 \leq b_i \leq 10^9)$, the costs of building the power lines.

The values of $c_{1,\ldots,n}$ are unique and given in strictly increasing order.

Output

Output the minimal cost of powering all cities on the island.

Example

Input	Output
3 2 1 100 2 200 150 300 150	400
3 2 1 100 2 200 300 300 150	450

Problem H. A Game Called Mind

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

This problem deals with a simplified version of a game called "Mind". In this game, a group tries to cooperatively solve a problem. Assume:

- There are 2-6 players (called A, B, C, D, E, F).
- Each player has 1-9 cards in sorted order.
- Each card value is between 10 and 99 (inclusive).
- There are no duplicate values, i.e., a card (number) is in at most one hand.

The objective of the game is for all the players to put all the cards down in sorted order (in the real game, players do not see their own cards, hence the name for the game – Mind). Given the cards in each hand, you are to provide the order for the players to place their cards down.

Input

The first input line contains an integer, $p(2 \leq p \leq 6)$, indicating the number of players. Each of the following p input lines provides the cards for one player. Each line starts with an integer, $c(1 \leq c \leq 9)$, indicating the number of cards for that player; the count is followed by the cards in sorted order (each card between 10 and 99, inclusive).

Output

Print the order for the players to place down the cards so that the cards are in order.

Example

Input	Output
2 3 10 40 50 2 20 30	ABBAA
3 4 40 51 60 70 3 12 32 42 5 20 53 80 90 95	BCBABACAACCC

Problem I. Unique Values

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

Arup has to make many practice questions for his Computer Science 1 students. Since many of the questions deal with arrays, he has to generate arrays for his students. Since he doesn't want to give them difficult practice problems, he always guarantees that the arrays (given to the students) have unique values. Namely, no value will appear twice in any of his arrays.

Unfortunately, Arup is too lazy to generate arrays! About 20 years ago when he started teaching Computer Science 1, he made one really long array to reuse but this long array may have duplicate values. When he makes problems, he simply grabs a contiguous subsequence of this long array to be the array to be used for a problem but he needs to make sure the contiguous subsequence does not contain duplicates. If the long array has terms a[0], a[1], ..., a[n-1], a contiguous subsequence of the long array is any sequence of j - i + 1 terms a[i], a[i+1], ..., a[j] where $0 \leq i \leq j \leq n$ - 1.

Given an array of n integers, determine how many contiguous subsequences of the array do not contain any repeated values. Note that two subsequences with the same contents are considered different (i.e., both counted in the total) if they start at different locations of the original long array

Input

The first input line contains a single positive integer, $n(1 \leq n \leq 10^5)$, representing the size of the input array. The following line contains n space separated integers, representing the values of the input array, in the order they appear in the array. Each of the array values will be an integer between 1 and 10^9 , inclusive.

Output

On a line by itself, output the total number of contiguous subsequences of the input array that do not contain any repeated values.

Example

Input	Output
5 1 1 2 1 5	9
8 2 12 3 12 3 2 6 9	22

Problem J. Gone Fishing

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

It is getting dark and the mosquitoes are attacking Fisherman Floyd. Floyd decides to throw his circular net one last time and wants to make the most out of the last throw.

Given the size (radius) of Floyd's net and positions (x,y) of a set of fish, what is the maximum fish Floyd can catch with one throw? That is, find the maximum points you can have in the net (circle). If a point (fish) is within $10^{-}6$ of the circle boundary, consider the point in the circle.

Input

The first input line provides the radius of the circle. The second input line contains an integer, $n(1 \leq n \leq 100)$, indicating the number of fish. Each of the next n input lines provides the location (x,y) for one fish. Assume all these points are distinct. Also assume that all the x and y values in the input (as well as the circle radius) are integers between 1 and 1000, inclusive.

Output

Print the maximum fish Floyd can catch with one throw.

Example

Input	Output
20 4 6 7 5 5 190 100 4 4	3
3 2 1 1 95 4	1

Problem K. Kleptocrat

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

Your company has a policy that every employee should be refunded an amount of money proportional to the shortest distance between their home and office. This causes the loophole that many employees intentionally move very far away to claim the maximum possible reimbursement.

One employee has taken this policy way too far and is in danger of bankrupting you. You must find a way to stop this before cancelling the policy next year. However, the rules are strict: as long as the employee keeps track of the distances they have travelled, you are forced to reimburse them.

Suddenly you have a flash of inspiration: nowhere does it say that you have to use the *Euclidean* distances! You start working on more subtle distance functions and now you have a first prototype: XOR distance. The length of a path is defined as the XOR of the lengths of the edges on the path (as opposed to the sum). The distance between two locations is defined as the length of the shortest path between them.

You will need to test this principle on the transport network with the locations of each of your employees in turn.

Input

- One line containing three integers n ($2 \leq n \leq 10^4$), m ($n-1 \leq m \leq 10^5$), and q ($1 \leq q \leq 10^5$), the number of nodes, edges, and questions respectively.
- m lines describing an edge. Each line consists of three integers x, y, w ($1 \leq x, y \leq n$, $x \neq y$ and $0 \leq w \leq 10^{18}$), indicating that there is an undirected edge of length w between nodes x and y.
- q lines describing a question. Each line consists of two integers a, b ($1 \leq a, b \leq n$) asking for the shortest distance between nodes a and b.

Between every pair of distinct nodes, there is at most one edge, and every node can be reached from any other node.

Output

For every question, output one line containing the shortest distance between nodes a and b.

Example

Input	Output
3 3 3 1 2 2 1 3 2 2 3 3 1 2 1 3 2 3	1 1 0
7 10 5 1 2 45 2 3 11 2 4 46 3 4 28 3 5 59 3 6 12 3 7 3 4 5 11 5 6 23 6 7 20 1 4 2 6 3 5 1 7 5 5	1 5 0 5 0

Problem L. Lost Map

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

An amateur Viking historian needs your help finding the silver left by Egill Skallagrímsson, of Egil's saga. She has found two old treasure maps that are supposed to lead to it. A treasure map is a list of instructions of the form "*direction k*", where *direction* can be "n", "s", "e", or "w". The maps are sadly old, so some of the instructions are missing and we represent them with a simple "?" instead.

The first map is larger while the second map is a smaller fragment. She wants to know how she can overlay her maps such that they coincide.

Two maps coincide if the corresponding instructions are either identical or at least one of them is lost to time. All instructions must have a corresponding instruction on the other map when overlaying the maps.

Input

- The first line of the input contains two integers, $1 \leq m < n \leq 4 \cdot 10^5$.
- The next n lines describe the first map with each containing either "?", or "(n|s|e|w)" followed by the number of steps s ($1 \leq s \leq 7$).
- The next m lines describe the second map with each containing either "?", or "(n|s|e|w)" followed by the number of steps s ($1 \leq s \leq 7$).

Output

Output the number of indices such that if the second map was overlaid at this index on the first map then they would coincide.

Example

Input	Output
4 3 n 4 e 1 ? s 5 ? e 1 ?	2
4 3 n 4 e 1 w 3 s 5 ? e 1 ?	1

Problem M. Moderate Pace

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

An ultra-marathon is a race that takes place over an uncomfortably long distance and time, typically lasting for five hours or more. You are part of a group of three ultra-marathon runners looking to place in this year's Great South-to-North run from Plymouth to Aberdeen.

You have a set number of days until the next race to train. You will all train together, as training alone can be dangerous. As everyone has their own schedule in mind for how many kilometres to run per day, this will not be easy, you will have to compromise.

The fairest option is to look at each day individually, examine the three options for how far to run, and to take the median one. That is to say, the option taken for each day should be one that is not be greater or lesser than both of the other possibilities at the same time.

Input

- A line with the integer n ($1 \leq n \leq 1000$), the number of days of training.
- A line with n integers $k_{1,\ldots,n}$ ($0 \leq k \leq 10^6$), your ideal daily distances.
- A line with n integers $a_{1,\ldots,n}$ ($0 \leq a \leq 10^6$), your first colleague's ideal daily distances.
- A line with n integers $b_{1,\ldots,n}$ ($0 \leq b \leq 10^6$), your second colleague's ideal daily distances.

Output

Output a plan for the n days as n integers, where the distance for every day corresponds to the median of choices for that day.

Example

Input	Output
4 1 2 3 4 4 3 2 1 2 2 2 2	2 2 2 2
6 3 1 4 1 5 9 2 7 1 8 2 8 1 6 1 8 0 3	2 6 1 8 2 8