Problem A. Funny Games

Input file: `funny.in`
Output file: `funny.out`

Nils and Mikael are intergalaxial fighters. Now they are competing for the planet Tellus. The size of this small and unimportant planet is $1 < X < 10000$ gobs. The problem is that their pockets only have room for one gob, so they have to reduce the size of the planet. They have available $1 \leq K \leq 6$ FACTOR-weapons characterized by numbers $F_1, F_2, \ldots, F_k$, all less than 0.9. As is commonly known, a FACTOR-weapon will blow off part of the planet, thus reducing the planet to a fraction of its size, given by the characteristic. Thus, with e.g. $F_1 = 0.5$ an application of the first weapon will half the size of the planet. The fighter who reduces the size to less than, or equal to, 1 gob can take the planet home with him. They take turns attacking the planet with any weapon. If Nils starts, who will win the planet? Assume that both Nils and Mikael are omniscient and always make a winning move if there is one.

Technical note: To ease the problem of rounding errors, there will be no edge cases where an infinitesimal perturbation of the input values would cause a different answer.

Input

The first line of input is $N \leq 100$, the number of test cases. Each of the next N lines consists of X, K and then the K numbers $F_1, F_2, \ldots, F_k$, having no more than 6 decimals.

Output

For each test case, produce one line of output with the name of the winner (either Nils or Mikael).

Sample input and output

funny.in	funny.out
4 6 2 0.25 0.5 10 2 0.25 0.5 29.29 4 0.3 0.7 0.43 0.54 29.30 4 0.3 0.7 0.43 0.54	Mikael Nils Nils Mikael

Problem B. Nullary Computer

Input file: `registers`
Output file: `registers`

Brian Huck has invented a new power-saving computer. With the current CMOS-based processors, a certain amount of power is lost each time a bit is changed from 0 to 1 or back. To avoid this problem, Brian's new Nullary Core stores only zeros. All numbers are stored in nullary form, as shown in Table 1.

Table 1: Numbers in nullary form

Decimal	Nullary
0	
1	0
2	00
3	000
4	0000
5	00000
...	...

His initial 64-nit model has 26 registers, each of which may store up to 64 nits, and any attempt to store more than 64 nits will result in a run time error. There is also a flag register, which contains either a zero, or nothing (initially it contains nothing). The instruction set is given in Table 2.

Table 2: NC instruction set

Instruction	Explanation
`A`	Add a zero to the value in register `A` (similarly for all uppercase letters).
`a`	First, empty the `flag` register. Then, if possible, remove a zero from register `A`, and place it in the `flag` register.
`(`	If the `flag` register is empty, jump past the matching `)`. Otherwise, empty the `flag` register.
`)`	Jump to the matching `(`.

Apart from instructions, no other characters than whitespace and/or newlines are allowed in a nullary program.

Brian has provided some programs to illustrate the elegance and simplicity of his computer (see Table 3).

Table 3: Sample NC programs

`b(b)a(Ba)`	Move register `A` to register `B` (by first emptying register `B`, then repeatedly pulling a single zero from register `A` and placing it into `B`).
`XXXa(GIa)i(g(FYg)y(Gy)f(Zb(z)z(i(YBi)y(Iy))f)Zb(zb)z(xz)i)x`	Set the `flag` register if the number of zeros in register `A` is prime.

Your task will be to write a sorting program for Brian's Nullary Core-based Prototype Computer. The NCPC has limited memory, so your program must be no longer than 5432 instructions. Also, the running time of your program must be no more than $5 \cdot 10^6$ steps for any possible input, where a step is considered to be the execution of one instruction.

Important note: In the testing system, there is a special language for submitting solutions to this problem: "Nullary Core Instructions (*.null)". To solve this problem, you should submit source files containing

Nullary Code instructions. Any attempt to solve this problem using another programming language, or (if you are curious enough!) any other problem using Nullary Core instructions will result in a Runtime Error or in a Wrong Answer verdict.

Input

The numbers to be sorted will be given in the first 24 registers `A`–`X`; the remaining two registers (`Y` and `Z`) will be empty.

Output

The sorted numbers should be in registers `A` through `X`, in increasing order. Register `Y` and `Z` should be empty.

Sample input and output

registers	registers
A 0	A
B 000000000	B
C 000000	C
D 0000	D
E 00000000	E
F 0000000	F
G 0000	G
H 000000	H
I 000000000	I
J 000	J
K	K
L	L
M	M
N	N 0
O	O 0
P	P 000
Q	Q 0000
R	R 0000
S	S 000000
T	T 000000
U	U 0000000
V	V 00000000
W	W 000000000
X 0	X 000000000
Y	Y
Z	Z

Problem C. Worst Weather Ever

Input file: `weather.in`
Output file: `weather.out`

– "Man, this year has the worst weather ever!", David said as he sat crouched in the small cave where we had sought shelter from yet another sudden rainstorm.

– "Nuh-uh!", Diana immediately replied in her traditional know-it-all manner.

– "Is too!", David countered cunningly.

Terrific. Not only were we stuck in this cave, now we would have to listen to those two nagging for at least an hour. It was time to cut this discussion short.

– "Big nuh-uh. In fact, 93 years ago it had already rained five times as much by this time of year."

– "Duh", David capitulated, "so it's the worst weather in 93 years then."

– "Nuh-uh, this is actually the worst weather in 23 years.", Diana again broke in.

– "Yeah, well, whatever", David sighed, "Who cares anyway?".

Well, dear contestants, you care, don't you?

Your task is to, given information about the amount of rain during different years in the history of the universe, and a series of statements in the form "Year X had the most rain since year Y", determine whether these are true, might be true, or are false. We say that such a statement is true if:

- The amount of rain during these two years and all years between them is known.
- It rained at most as much during year X as it did during year Y.
- For every year Z satisfying $Y < Z < X$, the amount of rain during year Z was less than the amount of rain during year X.

We say that such a statement might be true if there is an assignment of amounts of rain to years for which there is no information, such that the statement becomes true. We say that the statement is false otherwise.

Input

Input specifications The input will consist of several test cases, each consisting of two parts. The first part begins with an integer $1 \leq n \leq 50000$, indicating the number of different years for which there is information. Next follow n lines. The ith of these contains two integers $-10^9 \leq y_i \leq 10^9$ and $1 \leq r_i \leq 10^9$ indicating that there was r_i millilitres of rain during year y_i (note that the amount of rain during a year can be any nonnegative integer, the limitation on r_i is just a limitation on the input). You may assume that $y_i < y_{i+1}$ for $1 \leq i < n$.

The second part of a test case starts with an integer $1 \leq m \leq 10000$, indicating the number of queries to process. The following m lines each contain two integers $-10^9 \leq Y < X \leq 10^9$ indicating two years.

There is a blank line between test cases. The input is terminated by a case where $n = 0$ and $m = 0$. This case should not be processed.

Technical note: Due to the size of the input, the use of `cin`/`cout` in C++ might be too slow in this problem. Use `scanf`/`printf` instead. In Java, make sure that both input and output is buffered.

Output

There should be m lines of output for each test case, corresponding to the m queries. Queries should be answered with "true" if the statement is true, "maybe" if the statement might be true, and "false" if the statement is false.

Separate the output of two different test cases by a blank line.

Sample input and output

weather.in	weather.out
4 2002 4920 2003 5901 2004 2832 2005 3890 2 2002 2005 2003 2005 3 1985 5782 1995 3048 2005 4890 2 1985 2005 2005 2015 0 0	false true maybe maybe

Problem D. Kingdom

Input file: `kingdom.in`
Output file: `kingdom.out`

King Kong is the feared but fair ruler of Transylvania. The kingdom consists of two cities and $N < 150$ towns, with nonintersecting roads between some of them. The roads are bidirectional, and it takes the same amount of time to travel them in both directions. Kong has $G < 353535$ soldiers.

Due to increased smuggling of goat cheese between the two cities, Kong has to place his soldiers on some of the roads in such a way that it is impossible to go from one city to the other without passing a soldier. The soldiers must not be placed inside a city or a town, but may be placed on a road, as close as Kong wishes, to any town. Any number of soldiers may be placed on the same road. However, should any of the two cities be attacked by a foreign army, the king must be able to move all his soldiers fast to the attacked city. Help him place the soldiers in such a way that this mobilizing time is minimized.

The cities have ZIP-codes 95050 and 104729, whereas the towns have ZIP-codes from 0 to $N - 1$. There will be at most one road between any given pair of towns or cities.

Input

The input contains several test cases. The first line of each test case is N, G and E, where N and G are as defined above and $E < 5000$ is the number of roads. Then follow E lines, each of which contains three integers: A and B, the ZIP codes of the endpoints, and ϕ, the time required to travel the road, $\phi < 1000$. The last line of the input is a line containing a single 0.

Output

For each test case in the input, print the best mobilizing time possible, with one decimal. If the given number of soldiers is not enough to stop the goat cheese, print "Impossible" instead.

Sample input and output

kingdom.in	kingdom.out
4 2 6 95050 0 1 0 1 2 1 104729 1 95050 2 1 2 3 3 3 104729 1 4 1 6 95050 0 1 0 1 2 1 104729 1 95050 2 1 2 3 3 3 104729 1 4 2 7 95050 0 1 0 1 2 1 104729 1 95050 2 1 2 3 3 3 104729 1 2 1 5 0	2.5 Impossible 3.0

Problem E. Shoot-out

Input file: `shootout.in`
Output file: `shootout.out`

This is back in the Wild West where everybody is fighting everybody. In particular, there are n cowboys, each with a revolver. These are rather civilized cowboys, so they have decided to take turns firing their guns until only one is left standing. Each of them has a given probability of hitting his target, and they all know each other's probability. Furthermore, they are geniuses and always know which person to aim at in order to maximize their winning chance, so they are indeed peculiar cowboys. If there are several equally good targets, one of those will be chosen at random. Note that a cowboy's code of ethics forces him to do his best at killing one of his opponents, even if intentionally missing would have increased his odds (yes, this can happen!)

Input

On the first line of the input is a single positive integer t, telling the number of test cases to follow. Each case consists of one line with an integer $2 \leq n \leq 13$ giving the number of cowboys, followed by n positive integers giving hit percentages for the cowboys in the order of their turns.

Output

For each test case, output one line with the percent probabilities for each of them surviving, in the same order as the input. The numbers should be separated by a space and be correctly rounded to two decimal places.

Sample input and output

shootout.in	shootout.out
5 2 1 100 3 100 99 98 3 50 99 100 3 50 99 99 3 50 99 98	1.00 99.00 2.00 0.00 98.00 25.38 74.37 0.25 25.38 49.50 25.12 25.63 24.63 49.74

Problem F. Tour Guide

Input file: `tourguide.in`
Output file: `tourguide.out`

You are working as a guide on a tour bus for retired people, and today you have taken your regular Nordic seniors to The Gate of Heavenly Peace. You let them have a lunch break where they could do whatever they like. Now you have to get them back to the bus, but they are all walking in random directions. You try to intersect them, and send them straight back to the bus. Minimize the time before the last person is in the bus. You will always be able to run faster than any of the tour guests, and they walk with constant speed, no matter what you tell them. The seniors walk in straight lines, and the only way of changing their direction is to give them promises of camphor candy. A senior will neither stop at nor enter the bus before given such a promise.

Input

A number of test cases consisting of:

- a line with an integer $1 \leq n \leq 8$, the number of people on the tour;
- a line with a floating point number $1 < v \leq 100$, your maximum speed (you start in the bus at the origin);
- n lines, each containing four floating point numbers x_i, y_i, v_i, a_i, the starting coordinates ($-10^6 \leq x_i, y_i \leq 10^6$), speed ($1 \leq v_i < 100$) and direction ($0 \leq a_i < 2\pi$) of each of the tour guests.

The input is terminated by a case with $n = 0$, which should not be processed. All floating point numbers in the input will be written in standard decimal notation, and have no more than 10 digits.

Output

For each test case, print a line with the time it takes before everybody is back in the bus (the origin). Round the answer to the nearest integer. The answer will never be larger than 10^6.

Sample input and output

tourguide.in	tourguide.out
1 50.0 125.0 175.0 25.0 1.96 3 100.0 40.0 25.0 20.0 5.95 -185.0 195.0 6.0 2.35 30.0 -80.0 23.0 2.76 0	20 51

Problem G. Jezzball

Input file: `jezzball.in`
Output file: `jezzball.out`

"JezzBall is a computer game in which red-and-white "atoms" bounce about a rectangular field of play. The player advances to later levels (with correspondingly higher numbers of atoms and lives) by containing the atoms in progressively smaller spaces, until at least 75% of the area is blocked off." (wikipedia.org)

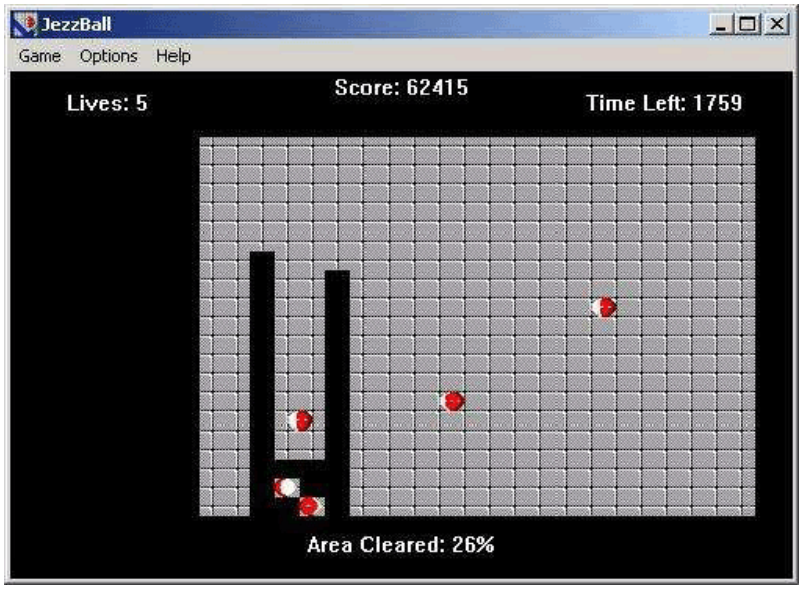

Figure 1: Jezzball screenshot

Figure 1 shows a screenshot from the original game, where the player has already covered some space (the black part). In this problem we will consider a slightly different, non-discrete, version of the game. That is, while the length unit is still pixels, you should treat them as non-discrete in the sense that all objects can be at non-integer coordinates and all movements are continuous.

The size of the playing field will be 1024×768 pixels. The atoms that bounce around will be infinitely thin (and not round balls like in the screenshot). The atoms will move at a constant speed and only change direction when hitting the edge of the playing field (x-coordinate 0 and 1024 or y-coordinate 0 and 768), where they bounce without loss of energy. The atoms do not hit each other.

The player can divide the playing field in two by shooting a horizontal or vertical ray from (in this problem) a fixed point on the playing field. The ray will then extend in both directions simultaneously (up and down for vertical rays, or left and right for horizontal rays) at a uniform speed (in this problem always 200 pixels per second). The rays will also be infinitely thin. If no atom touches any part of the ray while it's still being extended, the field has sucessfully been divided. Otherwise the player loses a life.

If an atom touches the endpoint of an extending edge, this will not be counted as a hit. Also, if an atom hits the ray at the same instant it has finished extending, this will also not count as a hit. Write a program that determines the minimum time the player must wait before he can start extending a ray so that an atom will not hit it before the ray has been completed.

Input

Each test case starts with a line containing a single integer n, the number of atoms ($1 \leq n \leq 10$). Then follows a line containing two integers, x and y, the position where the two ray ends will start extending from ($0 < x < 1024$, $0 < y < 768$). Then n lines follow, each containing four integers, x, y, v_x and v_y describing the initial position and speed of an atom ($0 < x < 1024$, $0 < y < 768$, $1 \leq |v_x| \leq 200$, $1 \leq |v_y| \leq 200$). The speed of the atom in the x direction is given by v_x , and the speed in the y direction is given by v_y. All positions in each input will be distinct. The input is terminated by a case where

$n = 0$, which should not be processed. There will be at most 25 test cases.

Output

For each test case, output the minimum time (with exactly 5 decimal digits) until the player can extend either a horizontal or vertical ray without an atom colliding with it while it is being drawn. The input will be constructed so that the first time this occurs will be during an open interval at least 10^{-5} seconds long. If no such interval is found during the first 10000 seconds, output "Never" (without quotes).

Sample input and output

jezzball.in	jezzball.out
3 700 420 360 290 170 44 900 150 -53 20 890 100 130 -100 4 10 10 1 1 192 144 513 385 192 144 1023 767 -192 -144 511 383 -192 -144 0	2.80094 Never

Problem H. Traveling Salesman

Input file: `traveling.in`
Output file: `traveling.out`

Long before the days of international trade treaties, a salesman would need to pay taxes at every border crossed. So your task is to find the minimum number of borders that need to be crossed when traveling between two countries. We model the surface of Earth as a set of polygons in three dimensions forming a closed convex 3D shape, where each polygon corresponds to one country. You are not allowed to cross at points where more than two countries meet.

Input

Each test case consists of a line containing c, the number of countries ($4 \leq c \leq 6000$), followed by c lines containing the integers $nx_1y_1z_1 \ldots x_ny_nz_n$, describing (in order) the n corners of a closed polygon ($3 \leq n \leq 20$). Then follows a line with one integer m ($0 < m \leq 50$), and then m lines with queries c_ac_b, where c_a and c_b are country numbers (starting with 1). No point will be on the line between two connected points, and $-10^6 \leq x, y, z \leq 10^6$ for all points. No two non-adjacent edges of a country share a common point. The input is terminated by a case where $c = 0$, which should not be processed.

Output

For each query, output the number of borders you must cross to go from c_a to c_b.

Sample input and output

traveling.in	traveling.out
6 4 0 0 0 0 0 1 0 1 1 0 1 0 4 1 0 0 1 0 1 1 1 1 1 1 0 4 0 0 0 1 0 0 1 0 1 0 0 1 4 0 1 0 1 1 0 1 1 1 0 1 1 4 0 0 0 0 1 0 1 1 0 1 0 0 4 0 0 1 0 1 1 1 1 1 1 0 1 2 1 2 1 3 0	2 1

Problem I. Whac-a-Mole

Input file: `whacamole.in`
Output file: `whacamole.out`

While visiting a traveling fun fair you suddenly have an urge to break the high score in the Whac-a-Mole game. The goal of the Whac-a-Mole game is to... well... whack moles. With a hammer. To make the job easier you have first consulted the fortune teller and now you know the exact appearance patterns of the moles.

The moles appear out of holes occupying the n^2 integer points (x, y) satisfying $0 \leq x, y < n$ in a two-dimensional coordinate system. At each time step, some moles will appear and then disappear again before the next time step. After the moles appear but before they disappear, you are able to move your hammer in a straight line to any position (x_2, y_2) that is at dis- tance at most d from your current position (x_1, y_1). For simplicity, we assume that you can only move your hammer to a point having integer coordinates. A mole is whacked if the center of the hole it appears out of is located on the line between (x_1, y_1) and (x_2, y_2) (including the two endpoints). Every mole whacked earns you a point. When the game starts, before the first time step, you are able to place your hammer anywhere you see fit.

Input

The input consists of several test cases. Each test case starts with a line containing three integers n, d and m, where n and d are as described above, and m is the total number of moles that will appear ($1 \leq n \leq 20$, $1 \leq d \leq 5$, and $1 \leq m \leq 1000$). Then follow m lines, each containing three integers x, y and t giving the position and time of the appearance of a mole ($0 \leq x, y < n$ and $1 \leq t \leq 10$). No two moles will appear at the same place at the same time.

The input is ended with a test case where $n = d = m = 0$. This case should not be processed.

Output

For each test case output a single line containing a single integer, the maximum possible score achievable.

Sample input and output

whacamole.in	whacamole.out
4 2 6 0 0 1 3 1 3 0 1 2 0 2 2 1 0 2 2 0 2 5 4 3 0 0 1 1 2 1 2 4 1 0 0 0	4 2

Problem J. Copying DNA

Input file: `dna.in`
Output file: `dna.out`

Evolution is a seemingly random process which works in a way which resembles certain approaches we use to get approximate solutions to hard combinatorial problems. You are now to do something completely different.

Given a DNA string S from the alphabet {`A`,`C`,`G`,`T`}, find the minimal number of copy operations needed to create another string T. You may reverse the strings you copy, and copy both from S and the pieces of your partial T. You may put these pieces together at any time. You may only copy contiguous parts of your partial T, and all copied strings must be used in your final T. Example: From S = "`ACTG`" create T = "`GTACTATTATA`"

1. Get `GT.........` by copying and reversing "`TG`" from S.
2. Get `GTAC.......` by copying "`AC`" from S.
3. Get `GTAC...TA..` by copying "`TA`" from the partial T.
4. Get `GTAC...TAAT` by copying and reversing "`TA`" from the partial T.
5. Get `GTACAATTAAT` by copying "AAT" from the partial T.

Input

The first line of input gives a single integer, $1 \leq t \leq 100$, the number of test cases. Then follow, for each test case, a line with the string S of length $1 \leq m \leq 18$, and a line with the string T of length $1 \leq n \leq 18$.

Output

Output for each test case the number of copy operations needed to create T from S, or impossible if it cannot be done.

Sample input and output

dna.in	dna.out
5 ACGT GTAC A C ACGT TGCA ACGT TCGATCGA A AAAAAAAAAAAAAAAAA	2 impossible 1 4 6

Problem K. Circle of Debt

Input file: `debt.in`
Output file: `debt.out`

The three friends Alice, Bob, and Cynthia always seem to get in situations where there are debts to be cleared among themselves. Of course, this is the "price" of hanging out a lot: it only takes a few resturant visits, movies, and drink rounds to get an unsettled balance. So when they meet as usual every Friday afternoon they begin their evening by clearing last week's debts. To satisfy their mathematically inclined minds they prefer clearing their debts using as little money transaction as possible, i.e. by exchanging as few bank notes and coins as necessary. To their surprise, this can sometimes by harder than it sounds. Suppose that Alice owes Bob 10 crowns and this is the three friends' only uncleared debt, and Alice has a 50 crown note but nothing smaller, Bob has three 10 crown coins and ten 1 crown coins, and Cynthia has three 20 crown notes. The best way to clear the debt is for Alice to give her 50 crown note to Cynthia, Cynthia to give two 20 crown notes to Alice and one to Bob, and Bob to give one 10 crown coin to Cynthia, involving a total of only five notes/coins changing owners. Compare this to the straight-forward solution of Alice giving her 50 crown note to Bob and getting Bobs three 10 crown notes and all his 1 crown coins for a total of fourteen notes/coins being exchanged!

Input

On the first line of input is a single positive integer, $1 \leq t \leq 50$, specifying the number of test cases to follow. Each test case begins with three integers ab, bc, $ca \leq 1000$ on a line of itself. ab is the amount Alice owes Bob (negative if it is Bob who owes Alice money), bc the amount Bob owes Cynthia (negative if it is Cynthia who is in debt to Bob), and ca the amount Cynthia owes Alice (negative if it is Alice who owes Cynthia).

Next follow three lines each with six non-negative integers a_{100}, a_{50}, a_{20}, a_{10}, a_5, a_1, b_{100}, …, b_1, and c_{100}, …, c_1, respectively, where a_{100} is the number of 100 crown notes Alice got, a_{50} is the number of her 50 crown notes, and so on. Likewise, b_{100}, …, b_1 is the amount of notes/coins of different value Bob got, and c_{100}, …, c_1 describes Cynthia's money. Each of them has at most 30 coins (i.e. $a_{10} + a_5 + a_1$, $b_{10} + b_5 + b_1$, and $c_{10} + c_5 + c_1$ are all less than or equal to 30) and the total amount of all their money together (Alice's plus Bob's plus Cynthia's) is always less than 1000 crowns.

Output

For each test case there should be one line of output containing the minimum number of bank notes and coins needed to settle the balance. If it is not possible at all, output the string "impossible".

Sample input and output

debt.in	debt.out
3 10 0 0 0 1 0 0 0 0 0 0 0 3 0 10 0 0 3 0 0 0 -10 -10 -10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -10 10 10 3 0 0 0 2 0 0 2 0 0 0 1 0 0 1 1 0 3	5 0 impossible

Problem L. Full Tank?

Input file: `tank.in`
Output file: `tank.out`

After going through the receipts from your car trip through Europe this summer, you realised that the gas prices varied between the cities you visited. Maybe you could have saved some money if you were a bit more clever about where you filled your fuel?

To help other tourists (and save money yourself next time), you want to write a program for finding the cheapest way to travel between cities, filling your tank on the way. We assume that all cars use one unit of fuel per unit of distance, and start with an empty gas tank.

Input

The first line of input gives $1 \leq n \leq 1000$ and $0 \leq m \leq 10000$, the number of cities and roads. Then follows a line with n integers $1 \leq p_i \leq 100$, where p_i is the fuel price in the ith city. Then follow m lines with three integers $0 \leq u, v < n$ and $1 \leq d \leq 100$, telling that there is a road between u and v with length d. Then comes a line with the number $1 \leq q \leq 100$, giving the number of queries, and q lines with three integers $1 \leq c \leq 100$, s and e, where c is the fuel capacity of the vehicle, s is the starting city, and e is the goal.

Output

For each query, output the price of the cheapest trip from s to e using a car with the given capacity, or "impossible" if there is no way of getting from s to e with the given car.

Sample input and output

tank.in	tank.out
5 5 10 10 20 12 13 0 1 9 0 2 8 1 2 1 1 3 11 2 3 7 2 10 0 3 20 1 4	170 impossible