Preliminaries

for the 2014 Benelux Algorithm Programming Contest

The Problem Set

A	Choosing Ice Cream
B	Failing Components
C	Floor Painting
D	Lift Problems
E	Pawns
F	Runway Planning
G	Spy Network
H	Talent Selection
I	Train Station Tunnel
J	Word Search

Almost blank page

A Choosing Ice Cream

You are standing in the supermarket in front of the freezers. You have a very tough task ahead of you: you have to choose what type of ice cream you want for after dinner that evening. After a while, you give up: they are all awesome! Instead, you take your (fair) k-sided die out of your pocket and you decide to let fate decide.

Of course, the number of ice cream choices, n, may not be precisely k, in which case you could not just throw the die once, rolling i, and take the ith ice cream choice. You therefore have to use some algorithm that involves zero or more die throws that results in an ice cream choice with every choice being exactly equally likely. Being a good computer scientist, you know about the accept-reject method, which would let you make such a fair choice.

Example: For $n = 4$ and $k = 20$ one throw is enough.

At that point, you remember that you have a very important competition to attend that same afternoon. You absolutely cannot afford to be late for that competition. Because of this, you decide you cannot use the accept-reject method, as there may be no bound on the number of die throws needed to ensure a fair result, so you may end up standing there for a long time and miss the competition! Instead, you resolve to find an algorithm that is fair and uses as few dice choices as possible in the worst case.

Given n and k, can you determine the minimum number i such that there is a fair algorithm that uses at most i die throws per execution?

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with two space-separated integers n and k ($1 \leq n, k \leq 10^9$): the number of ice cream choices and the number of sides of your die, respectively.

Output

Per test case:

- one line with a single integer: the smallest number of throws after which you are guaranteed to be able to make a fair choice. If there is no such number, print "`unbounded`" instead.

Sample in- and output

Input	Output
3 4 2 2 4 3 2	2 1 unbounded

Almost blank page

B Failing Components

As a jury member of the Best Architectural Planning Contest, you are tasked with scoring the reliability of a system. All systems entered in the contest consist of a number of components which depend on each other. The reliability of such a system depends on the damage done by a failing component. Ideally a failing component should have no consequences, but since most components depend on each other, some other components will usually fail as well.

Most components are somewhat resilient to short failures of the components they depend on. For example, a database could be unavailable for a minute before the caches expire and new data must be retrieved from the database. In this case, the caches can survive for a minute after a database failure, before failing themselves. If a component depends on multiple other components which fail, it will fail as soon as it can no longer survive the failure of at least one of the components it depends on. Furthermore no component depends on itself directly, however indirect self-dependency through other components is possible.

You want to know how many components will fail when a certain component fails, and how much time passes before all components that will eventually fail, actually fail. This is difficult to calculate by hand, so you decided to write a program to help you. Given the description of the system, and the initial component that fails, the program should report how many components will fail in total, and how much time passes before all those components have actually failed.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with three space-separated integers n, d and c ($1 \leq n \leq 10\,000$ and $1 \leq d \leq 100\,000$ and $1 \leq c \leq n$): the total number of components in the system, the number of dependencies between components, and the initial component that fails, respectively.
- d lines with three space-separated integers a, b and s ($1 \leq a, b \leq n$ and $a \neq b$ and $0 \leq s \leq 1\,000$), indicating that component a depends on component b, and can survive for s seconds when component b fails.

In each test case, all dependencies (a, b) are unique.

Output

Per test case:

- one line with two space-separated integers: the total number of components that will fail, and the number of seconds before all components that will fail, have actually failed.

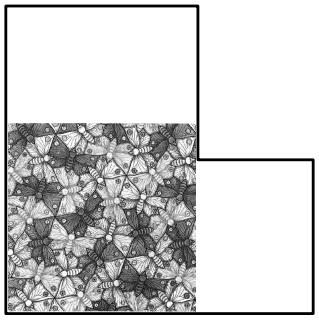
Sample in- and output

Input	Output
2 3 2 2 2 1 5 3 2 5 3 3 1 2 1 2 3 1 8 3 2 4	2 5 3 6

C Floor Painting

The museum of Bizarre Argentinean Pocket Calculators (BAPC) has found a great painter to make a nice floor painting for in the museum. All walls in the museum are straight, and any two adjacent walls meet at a right angle. All non-adjacent walls are pairwise disjoint. Furthermore, the room in which the painting is supposed to come has no pillars. Hence, it is a rectilinear simple polygon.

The design for the painting is square. The museum wants the painting to be as large as possible. Furthermore, it should be placed such that the edges of the square are parallel to the walls. Find the maximum possible width for the painting.



Floor plan of the first sample input with a 5×5 painting.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with a single integer n ($4 \leq n \leq 1\,000$): the number of corners in the room.
- n lines, each with two space-separated integers x and y ($0 \leq x, y \leq 100\,000$): the coordinates of each corner of the room.

The corners are given in clockwise order.

Output

Per test case:

- one line with a single integer: the maximum width of a square painting that fits on the floor of the museum.

Sample in- and output

Input	Output
3 6 0 8 5 8 5 4 8 4 8 0 0 0 4 0 4 4 4 4 0 0 0 14 4 8 12 8 12 6 14 6 14 8 16 8 16 3 8 3 8 0 0 0 0 6 3 6 3 7 4 7	5 4 6

D Lift Problems

On the ground floor (floor zero) of a large university building a number of students are waiting for a lift. Normally, the lift stops at every floor where one or more students need to get out, but that is annoying for the students who want to get out on a higher floor. Alternatively, the lift could skip some floors, but that is annoying for the students who wanted to get out on one of those floors.

Specifically, a student will be annoyed on every floor where the lift stops, if the lift has not yet reached the floor on which he or she wants to get out. If the lift skips the floor on which a student wants to get out, he or she will be annoyed on that floor and every higher floor, up to (and excluding) the floor where the lift makes its next stop and the student can finally get out to start walking back down the stairs to his or her destination.

For example, if a student wants to get out on the fifth floor, while the lift stops at the second, seventh and tenth floor, the student will be annoyed on floors two, five and six. In total, this student will thus be annoyed on three floors.

Upon entering the lift, every student presses the button corresponding to the floor he or she wants to go to, even if it was already pressed by someone else. The CPU controlling the lift thus gets to know exactly how many students want to get out on every floor.

You are charged with programming the CPU to decide on which floors to stop. The goal is to minimize the total amount of lift anger: that is, the number of floors on which every student is annoyed, added together for all students.

You may ignore all the people who may (want to) enter the lift at any higher floor. The lift has to operate in such a way that every student waiting at the ground floor can reach the floor she or he wants to go to by either getting out at that floor or by walking down the stairs.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with a single integer n ($1 \leq n \leq 1\,500$): the number of floors of the building, excluding the ground floor.
- one line with n space-separated integers s_i ($0 \leq s_i \leq 1\,500$): for each floor i, the number of students s_i that want to get out.

Output

Per test case:

- one line with a single integer: the smallest possible total amount of lift anger.

Sample in- and output

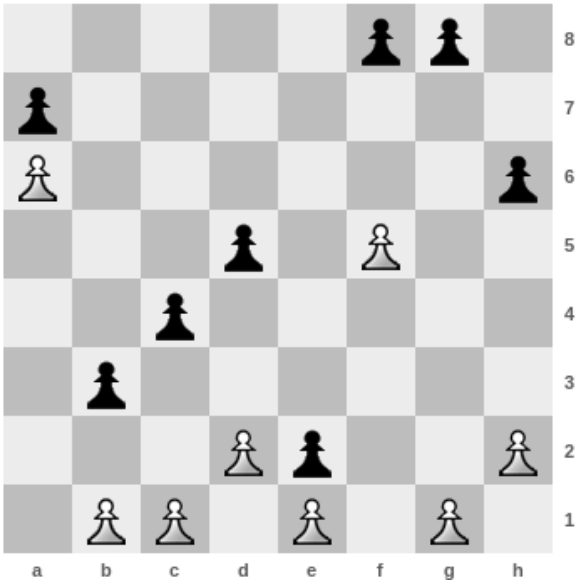
Input	Output
3 5 0 3 0 0 7 5 0 0 3 0 7 10 3 1 4 1 5 9 2 6 5 3	7 6 67

E Pawns

Carl and Nathan are completely bored with the game of chess; it's too easy! They have come up with their own game, which is surely a greater test of one's intelligence.

This game is played on a board with n by m squares. At the start, white and black pawns are placed quasi-randomly over the board, with the following constraint: in every column there is one white pawn and one black pawn, with the white pawn on some square below the black one.

Each player in turn makes a move with one of his pawns. A pawn is only allowed to move one square forward, provided that this square is empty. "Forward" means in the direction of the opponent, so white pawns move up and black pawns move down. In addition, a pawn on the first rank – that is, a white pawn on the bottom row, or a black pawn on the top row – may also move two squares forward, provided that both squares are empty. Unlike normal chess, the pawns are never taken from the board and never change column.



For example, in the position above, White (the player using the white pieces) has eight moves: one with each of the pawns on b1, d2, f5 and h2, and two with both the pawn on c1 and the pawn on g1. The pawns on a6 and e1 cannot move.

Eventually and inevitably, the pawns will meet up in every column, leaving neither player able to move. The game is then finished, and the winner is the player who made the last move.

As usual, White gets the first move. With optimal play, who would win for a given starting position?

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with two space-separated integers n and m ($3 \leq n \leq 20$ and $1 \leq m \leq 20$): the number of rows and columns of the board, respectively.
- n lines with m characters, describing the position on the board at the start of the game:
  - '`W`' is a white pawn.
  - '`B`' is a black pawn.
  - '`.`' is an empty square.

  Each column contains exactly one '`W`' and one '`B`', with the '`W`' being below the '`B`'.

In every test case, the starting position will be such that White has at least one move.

Output

Per test case:

- one line with the string "`White wins`" if White can win with optimal play, or "`Black wins`" if Black has a winning strategy.

Sample in- and output

Input	Output
5 8 8 .....BB. B....... W......B ...B.W.. ..B..... .B...... ...WB..W .WW.W.W. 6 4 .... B..B .... .B.W W.B. .WW. 5 3 ... BBB ... WW. ..W 4 6 .BBB.B B...B. ...W.. WWW.WW 7 7 .B.B..B ....B.B.. B....B. ....WW.W WWW..W.	Black wins Black wins White wins White wins Black wins

Almost blank page

F Runway Planning

Most airports have multiple runways. To identify runways, they are given a number indicating the direction of the runway. Such a runway number is obtained by dividing the heading of the runway in degrees by ten, rounding the result, and optionally prefixing it with a '0' if the result has only a single digit. For example, a runway with a heading of $82°$ is indicated by the number 08.

If you are paying attention, you might think "a runway can be used in both directions, and therefore has two headings, but it is only assigned one runway number." You are correct: normally a runway is identified by two numbers, based on the direction in which the runway is used. To simplify matters, we only concern ourselves with the smallest of these two numbers, except if it is zero; we then use 18 instead. The runway numbers thus range from 01 to 18.

Now, you might think, "what if two runways have the same heading?" In that case, the characters 'L' and 'R' are appended to the number of the left and right runway, respectively. But what if three runways have the same heading? Then, the character 'C' is appended to the center runway. "But", I can hear you ask, "what if four runways have the same heading?" If you really want to know, look up how Dallas/Fort Worth International Airport solved this problem after the contest. At any rate, we do not concern ourselves with multiple runways having the same heading in this problem, so you can forget all you read in this paragraph.

The runway in use is mostly determined by the current direction of the wind. It is preferred to take off and land with headwind. If it is not possible to have the wind coming from straight ahead, its direction should be as close to that as possible. For example, if an airport has the runways 05 and 15, and the wind has a heading of $70°$, taking off and landing using runway 05 is preferred, since the heading of that runway is closest to the heading of the wind.

Now, consider an airport already having one or more runways, and planning the construction of a new runway. Obviously, this runway should have a unique runway number: not only would we otherwise have a problem outside the boundaries of our restricted runway numbering outlined above, but, most importantly, this increases the probability of being able to take off or land with headwind.

The engineers at the airport under consideration have already determined the heading of the new runway, but still need you to determine the runway number. Note that the engineers are not very considerate with their input to your problem. They give you one heading of the runway, but it can be either the lowest or the highest heading of the runway. Be sure to give the lowest of the two runway numbers, as discussed in the second paragraph of this problem statement, even if you are given the highest of the two headings from the engineers.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with a single integer h ($1 \leq h \leq 360$): the heading of the new runway in degrees.

Output

Per test case:

- one line with the runway number of the newly constructed runway.

Sample in- and output

Input	Output
4 82 115 316 4	08 12 14 18

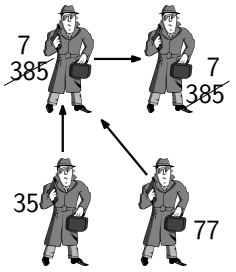
G Spy Network

The NSA has found a way to factor large integers! They are using it to send encrypted messages over their spy networks. Their spy network can be modelled as a directed graph $G = (V, E)$, where vertices are agents of the NSA and an edge from u to v models that u can inconspicuously tell its information to v ('v is a contact of u'). Note that v might not be able to send information back to u!

The NSA is constantly looking out for signs of various known terrorist organisations. Every few months its agents report to their colleagues what organisations they have heard rumours about during their surveillance. The protocol for this is as follows. On the specified date, all agents will send the information they have gathered to their contacts. If an agent receives information $i_{received}$ from any source, it looks at the information i_{old} it has found so far, and generates from this a new report i_{new}, in which it stores data on terrorist organisations that were present in both $i_{received}$ and i_{old} (this is because the NSA gets too many false alerts, so they believe a message if ALL sources agree on something). The agent then stores i_{new} as the new 'data it currently knows about', and then sends this information to all its own contacts, who then act in the same manner, and so on. After a while, agents only receive messages that do not change their stored data. At the end of the day, agents stop sending messages, which is always enough for the aforementioned state to be reached. Note that the graph may not be connected and that it may contain cycles for redundancy.

The NSA uses the following encryption for this protocol. For every large terrorist organisation, they pick a large prime number and they distribute this mapping to all their agents. If a member generates his report, he multiplies the large prime numbers that correspond to the organisations he has heard about, and then the resulting number will be his 'report' that he sends to his contacts. When an agent receives $i_{received}$, he takes the greatest common divisor of $i_{received}$ and i_{old}. The resulting number is precisely the number you would get if you multiply the large prime numbers corresponding to the terrorist organisations that both members of the NSA that generated the reports have heard rumours about, and so exactly forms i_{new}. If an agent is important enough, he gets access to the factoring algorithm and is able to read the messages sent by his contacts. Most agents just propagate the messages though, and using the greatest common divisor algorithm, they can do so without having to factor the number.

The NSA has recently learned that an encrypted report that was generated after such a day has been leaked! Although the message was encrypted and so no information was lost, the NSA still wants to know which of its members has leaked the message and has asked for your help. Of course, you must write a program to do the work for them - you are not allowed to know the data yourself. Your program will get the structure of their spy network, along with the initial reports for every member and the leaked report. You are to print out the number of members that had this leaked report as a final report, so that the NSA knows how far they can narrow down their investigation.



Spy network of first sample input.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with three space-separated integers n, m and l ($1 \leq n \leq 10\,000$ and $0 \leq m \leq 10\,000$ and $1 \leq l < 2^{63}$): the number of agents, contact edges and the leaked report.
- m lines, each with two space-separated integers a and b ($1 \leq a, b \leq n$ and $a \neq b$), indicating that agent b is a contact of agent a.
- n lines, each with a single integer i ($1 \leq i < 2^{63}$): the initial information of each agent.

Output

Per test case:

- one line with a single integer: the number of agents with l as its stored message at the end of the message sending day.

Sample in- and output

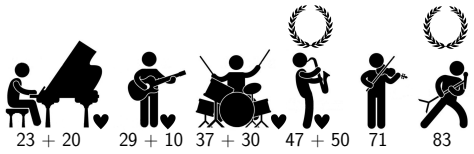
Input	Output
3 4 3 7 1 3 2 3 3 4 35 77 385 385 4 4 159 1 2 2 3 3 1 4 1 159 159 159 2014 5 5 9 2 1 2 3 3 2 3 4 5 4 27 54 90 315 135	2 0 2

H Talent Selection

You find yourself in the jury of a talent show. In the first round, people from all over the country come around to show you their skills. Only a certain number of people can be allowed to advance to the next round.

Now that all candidates have been seen, it is time to select the people who will advance. Every jury member has some favourites they want to put forward. After a lot of discussion, it is clear that no consensus will be reached this way. They decide to let every jury member award points to a certain number of candidates. The points are all added up and the candidates with the most points advance.

You have your own list of favourites, of whom you would like to let as many as possible advance. Fortunately, from the discussion, it is quite obvious how all the other jury members are going to divide their points. Points are assigned using stickers. Each sticker is worth a certain number of points and you can only assign one sticker to a candidate. Furthermore all stickers have to be assigned. How many of your favourites can you get through if you distribute your stickers optimally?



A sticker distribution for the first sample input that advances one of your favorites.

Should there be some candidates with the same number of points of whom only some can advance, you can use your powers of charm and persuasion to get as many of your favourites through as possible.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with three space-separated integers n, s and f ($1 \leq s, f < n \leq 100\,000$): the number of candidates, the number of people selected to advance and the number of your favourites, respectively.
- one line with f space-separated integers t_i ($0 \leq t_i \leq 10^6$): the number of points your favourites get from the other jury members.
- one line with $n - f$ space-separated integers t_j ($0 \leq t_j \leq 10^6$): the number of points the other candidates get from the other jury members.
- one line with a single integer k ($1 \leq k \leq n$): the number of stickers which you can (and must) use to assign points.
- one line with k space-separated integers p_i ($1 \leq p_i \leq 100\,000$): The number of points each sticker is worth, in increasing order.

Output

Per test case:

- one line with a single integer: the maximum number of your favourites that you can let advance to the next round.

Sample in- and output

Input	Output
3 6 2 4 23 29 37 47 71 83 4 10 20 30 50 5 3 3 40 60 80 85 85 4 10 20 30 50 6 3 3 14 15 92 65 35 89 4 10 20 30 50	1 2 2

I Train Station Tunnel

In Eindhoven station, there is only one way to get on or off the tracks: the train tunnel. During rush hour it's very busy with people who want to go from one side of the station to the other. However, both exits are roughly equally important, so about half the people walk one way and the other half walk the other way, and as people cannot walk through each other, this leads to conflicts. Furthermore, people walk at different speeds, which means some fast people have to wait for people in front of them.

We want to get some data on how efficient the tunnel is at handling large numbers of people. In order to do this, we regard a tunnel of a certain length and width as a two-dimensional grid, where each person takes up exactly one grid point. We will ignore that there are train tracks and assume that the entrances are the entire left and right sides of the tunnel. The top-left corner is located at $(1, 1)$ and the bottom-right corner at $(length, width)$, which are both inside the tunnel. We have a number of people who walk through the tunnel walking at different speeds, starting at different positions (representing their different arrival times).

Time passes in ticks. Every tick, each person walks some distance. No person can walk through another person or through the wall. If a person walks into the back of another person, then that has no influence on the speed of the person in front of her: the person behind the other person then goes as far as possible while staying behind the person in front of her. If a person walks into another person going in the opposite direction, the moving person ends her move one grid point in front of the person she walked into.

As the university is on the right side of the tunnel, we obviously have that the people moving from left to right are more in a hurry. Therefore, at every tick, the people that move from left to right will move before the people that move from right to left. People moving in the same direction move simultaneously.

If a person walked into another person and was able to walk only half the distance (rounded up) that person would like to walk in a tick or less, then that person becomes annoyed and will try to take a step to either side before the next tick starts. This movement between ticks happens as follows. First, from top to bottom, every annoyed person moving to the right will try to take a step left (that person's left, so up). Then, from bottom to top, every annoyed person moving left will try to step left (that person's left, so down). Then from bottom to top every annoyed person moving right who was unable to step left ('still annoyed') will try to step right. Finally, from top to bottom every still annoyed person moving left will try to step right. An annoyed person is not annoyed anymore at the start of a new tick.

We want to know at what time every person has left the tunnel (that is, ends up behind the tunnel exit that person is moving towards), to get some data on how efficient the tunnel is at handling large numbers of people with the above characteristics. It is always possible for all people to reach the end of the tunnel from their starting position.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with three space-separated integers l, w and p ($1 \leq l, w \leq 3\,000$ and $1 \leq p \leq 1\,000$): the length and width of the tunnel and the number of people, respectively.
- p lines, each with three space-separated integers x, y and s ($0 < x \leq l$ and $0 < y \leq w$ and $0 < s \leq 1\,000$): the starting position (x, y) and speed s of each person. This is followed

by a space and a single character that is either '`L`' or '`R`', indicating that the person walks to the left or the right side of the tunnel, respectively.

A person that exits the tunnel is considered to be removed from the grid. A person exits the tunnel at $(x \leq 0)$ or $(x > l)$.

Output

Per test case:

- one line with a single integer: the (smallest) number of ticks after which all people will have exited the tunnel.

Sample in- and output

Input	Output
2 11 10 3 1 1 2 R 10 1 2 L 11 1 3 L 8 4 3 4 2 3 R 1 3 3 R 8 2 5 L	8 4

J Word Search

A word search puzzle is a puzzle that involves a rectangular grid of letters and a list of words. The objective is to find and mark all those words, which are hidden inside the grid. The words may be placed horizontally, vertically, or diagonally, in either direction. When you have found a word, you mark all the letters in the grid that are involved. A letter may be part of multiple words. At the end, all the unmarked letters, from top to bottom and from left to right, form a message; this is the solution.

A certain magazine has a bunch of word search puzzles in it. They would like you to check, for each puzzle, that all words are actually in the grid. You should also be on the lookout for words that can be found in two (or more) different places – even if it does not influence the final solution. If all is well, just give the solution.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with three integers n, h and w ($1 \leq n \leq 256$ and $1 \leq h, w \leq 32$): the number of words and the height and width of the grid, respectively.
- h lines with w uppercase letters: the grid.
- n lines, each with a single string s ($1 \leq \mathit{length}(s) \leq 32$), consisting of uppercase letters only: the words to be found in the grid.

Output

Per test case:

- one line with a single string of uppercase letters: the solution. If there is a word that is not present in the grid, print "`no solution`" instead. If all words are present, but there is a word for which there are two (or more) different sets of letters that could be marked, print "`ambiguous`" instead. If all words are present and can be found uniquely in the grid, yet there are no unmarked letters remaining, print "`empty solution`" instead.

Sample in- and output

Input	Output
4 10 7 8 ELIPMOCN TACODEOL IMELBORP MGOALRRM BIPLEIEA UCATZUCE SBHEMSTT BAPC TUE TEAM PROBLEM CODE COMPILE SUBMIT CORRECT BALLOON PRIZE 2 4 3 BCB AOA PDP CEC BAPC CODE 3 4 3 BCB AOA PDP CEC BAPC CODE TEAM 2 2 10 DELEVELEDB ATESTSETPC DELEVELED TESTSET	ALGORITHMS ambiguous no solution BAPC