Problem A. Eight Queens

Source file name:	queens.c, queens.cpp, queens.java
Input:	`standard`
Output:	`standard`

In the game of chess, the queen is a powerful piece. It can attack by moving any number of spaces in its current row, in its column or diagonally.
In the eight queens puzzle, eight queens must be placed on a standard 8 x 8 chess board so that no queen can attack another. The center figure below shows an invalid solution; two queens can attack each other diagonally. The figure on the right shows a valid solution. Given a description of a chess board, your job is to determine whether or not it represents a valid solution to the eight queens puzzle.

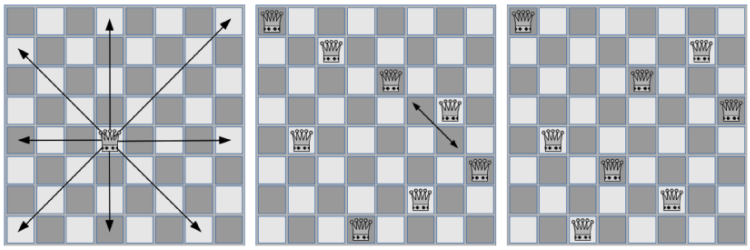


Figure A.1: Queen movement (left), invalid solution (center), valid solution (right).

Input

Input will contain a description of a single chess board, given as eight lines of eight characters each.
Input lines will consist of only the characters '.' and '*'. The '.' character represents an empty space on the board, and the '*' character represents a queen.

Output

Print a single line of output. Print the word "valid" if the given chess board is a valid solution to the eight queens problem. Otherwise, print "invalid".

Example

Input	Output
....*... ......*. .*...... ....* .....*.. ...*....	invalid
....*... ....* .*...... ...*.... .....*.. ..*.....	valid

Problem B. Color Walk

Source file name:	colorwalk.c, colorwalk.cpp, colorwalk.java
Input:	`standard`
Output:	`standard`

Alice and Bob are playing a game on a directed graph, fully visible to both of them. The graph has n nodes labelled 1 through n and directed arcs between nodes. Each arc is colored either red or black. In addition to the graph, there is a queue of capacity k, fully visible to both of them. At the start of the game, Alice places a game piece on node 1, and the queue is empty. Bob then inserts colors (either red or black) into the back of the queue until the queue is full. Now the main portion of the game begins: Alice removes the color at the front of the queue and moves the game piece along an outgoing arc of the same color from the current node to its destination node (which could be the same node in the case of a self-loop). The queue now has space for one more color, which Bob must now provide. Bob wins if Alice is ever unable to move the game piece, i.e., because there are no outgoing arcs of the appropriate color from the current node. Alice wins if she can keep playing forever (she really likes this game).

Given the complete description of the graph, determine the minimum capacity k for the queue such that no matter how well Bob plays, Alice can always win.

Input

Input begins with a line with a single integer t, $1 \leq t \leq 20$, denoting the number of test cases. Each test begins with a line with a single integer n, $1 \leq n \leq 12$, denoting the number of nodes in the graph. Next follow n lines each with n space-separated integers, where the jth number in the ith line is 1 if there is a red arc from node i to node j, and 0 otherwise. Next follow n lines each with n space-separated integers, where the jth number in the ith line is 1 if there is a black arc from node i to node j, and 0 otherwise. Remember that node 1 is always the start node.

Output

For each test case, print a single integer k on a single line equal to the minimum capacity for the queue such that no matter how well Bob plays, Alice can always win. If for all k, no matter how well Alice plays, Bob can always win, print 0.

Example

Input	Output
3 3 1 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 1 1 2 0 1 1 0 0 0 0 1	2 1 0

Problem C. Flexible Spaces

Source file name:	flexible.c, flexible.cpp, flexible.java
Input:	`standard`
Output:	`standard`

Golomb Industries is designing their new office building following modern principles that allow for flexible, reconfigurable meeting spaces. Their plans include a very wide rectangular room, with a series of optional parallel partitions.

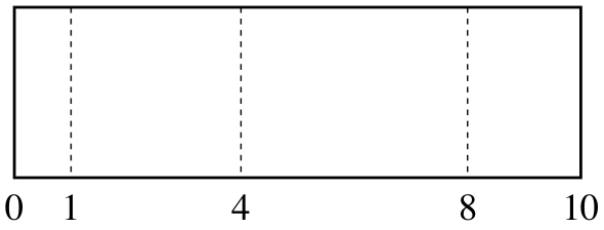


Figure C.1: An example of a configurable space

Figure C.1 illustrates such a room having a width of 10 units and three optional partitions located 1 unit, 4 units, and 8 units away from the left wall of the room. The width of the room always measures the distance between the left and right walls, and partitions always run parallel to these walls. We do not concern ourselves with the depth of the room.

For this example, if no partitions are used, a meeting can be held in the original space having width 10. If the company needs a space that is precisely 4 units wide, they can use the portion of the room between the left wall and a partition placed at location 4 (or they could use the portion between the partitions at locations 4 and 8). To provide a space having width 7 they can use the portion of the room between the partitions placed at locations 1 and 8 (omitting the partition at location 4).

Given a particular room design, your job is to determine all feasible widths for a meeting.

Input

The first line of the input contains two integers: the overall width W of the room, with $2 \leq W \leq 100$, and the number P of intermediate partitions, such that $1 \leq P < W$. Following that is a line with P integers, each designating the location L of an optional partition, such that $0 < L < W$. Each partition is at a distinct location and the partitions' locations will be listed in increasing order.

Output

Your program should output a list, from smallest to largest, of each distinct width that can be achieved for a meeting space.

Example

Input	Output
10 3 1 4 8	1 2 3 4 6 7 8 9 10

Problem D. Flip Five

Source file name:	flip.c, flip.cpp, flip.java
Input:	`standard`
Output:	`standard`

This is a logic puzzle in which you have a square grid of 3x3 cells. Each cell is initially either white or black. When you click on a square it flips, or toggles, the color of that square and the colors of its four immediate north, south, east and west neighbors that exist (they don't exist if they would be outside the grid). The problem is to find the minimum number of cell clicks to transform a grid of all white cells into the input grid (which is always possible). You cannot rotate the grid.

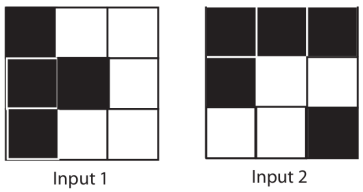


Figure D.1: The two sample problems

Input

The first value in the input file is an integer P ($0 < P \leq 50$) on a line by itself giving the number of problems to solve.
For each of the P problems, 3 lines of 3 characters describe the input grid. The characters in the grid descriptions are '*' (for black) and '.' (for white).

Output

For each problem output a single integer giving the minimum number of clicks necessary to transform a grid of all white cells into the pattern given in the input.

Example

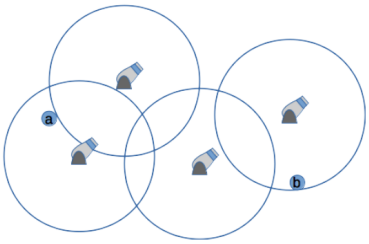
Input	Output
2 *.. **. *.. *** *.. ..*	1 3

Problem E. Human Cannonball Run

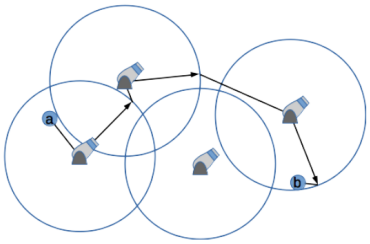
Source file name:	human.c, human.cpp, human.java
Input:	`standard`
Output:	`standard`

You are a world-famous circus performer, a human cannonball. This means that you climb into a big, fake cannon and launch yourself great distances to delight young and old alike. Today, you're not alone. You are at the international human cannonball conference and exposition, where hundreds of similar circus performers have gathered together to share their experiences and practice their craft. While you normally have just one cannon to work with, at the conference there are usually lots of cannons to examine and try out.

The availability of several cannons creates some interesting opportunities for navigating the conference. If you want to travel quickly from point a to point b, you could just run straight from a to b, or, you could run to a nearby cannon and launch yourself somewhere else. From there, you can continue to run toward your destination or you can continue to use cannons in an effort to get to your destination more quickly. With cannons positioned like Figure E.1, you could follow a path like the one in Figure E.2 to get from a to b. The arrows show places where you launched yourself out of a cannon, and the lines show where you ran to the next cannon or to your destination.



(a) Figure E.1: Illustration of the sample input.



(b) Figure E.2: A suboptimal solution.

You run at a rate of 5 meters per second. All cannons launch you a distance of 50 meters, in any direction you'd like. Climbing into a cannon, launching yourself and landing takes a total of 2 seconds. Cannons are not obstacles; if a cannon is in your way, you can jump over or run around it without it slowing you down. Given your current location, a desired destination and the positions of available cannons, you want to plan how to get to the destination as quickly as possible.

Input

The input describes a single navigation problem. The first line gives a pair of real numbers, the X and Y coordinates where you're currently located. The next line give the real-valued X and Y coordinates of the location you'd like to reach. This is followed by a line with an integer, n, the number of cannons available. The remaining n input lines each contain a pair of real values giving the X and Y coordinates for a cannon. All coordinates are measured in meters, and the value of n will be between 0 and 100, inclusive.

Output

Print a single line of output, the total number of seconds required to reach your destination as quickly as possible. Your answer must be accurate to within 0.001 seconds.

Example

Input	Output
25.0 100.0 190.0 57.5 4 125.0 67.5 75.0 125.0 45.0 72.5 185.0 102.5	19.984901

Problem F. Mixed Fractions

Source file name:	mixed.c, mixed.cpp, mixed.java
Input:	`standard`
Output:	`standard`

You are part of a team developing software to help students learn basic mathematics. You are to write one part of that software, which is to display possibly improper fractions as mixed fractions. A proper fraction is one where the numerator is less than the denominator; a mixed fraction is a whole number followed by a proper fraction. For example the improper fraction 27/12 is equivalent to the mixed fraction 2 3/12. You should not reduce the fraction (i.e. don't change 3/12 to 1/4).

Input

Input has one test case per line. Each test case contains two integers in the range $[1, 2^{31} - 1]$. The first number is the numerator and the second is the denominator. A line containing 0 0 will follow the last test case.

Output

For each test case, display the resulting mixed fraction as a whole number followed by a proper fraction, using whitespace to separate the output tokens

Example

Input	Output
27 12 2460000 98400 3 4000 0 0	2 3 / 12 25 0 / 98400 0 3 / 4000

Problem G. Pesky Mosquitoes

Source file name:	pesky.c, pesky.cpp, pesky.java
Input:	`standard`
Output:	`standard`

Mosquitoes are relentless this time of year! They have absolutely ruined your attempt at a picnic and it is time to take your revenge. Unfortunately, you are not well equipped to ward off these pests. All you have got at your disposal is an empty bowl that previously held potato salad. As you glance down at the picnic table, you see a number of mosquitoes waiting idly for you to let your guard down. This is your chance to fight back.

Your task is to determine the maximum number of mosquitoes that can be trapped by quickly bringing down the inverted bowl onto the table. You will be provided with the diameter of the bowl and the exact location of each mosquito on the table. In this exercise you can assume that the mosquitoes are incredibly small and can simply be modeled as a point. A mosquito that lies exactly under the edge of the bowl is considered trapped.

Input

The first number in the input will be an integer $1 \leq n \leq 100$ that denotes the number of mosquito-trapping scenarios that follow. A blank line comes at the beginning of each scenario. Then follows a line containing an integer $1 \leq m \leq 32$ (the number of mosquitoes) and a real number $0 < d \leq 200$ (the diameter of the bowl). Each of the following m lines will specify the location of a mosquito in the form of real coordinates $-100 \leq x \leq 100$ and $-100 \leq y \leq 100$.

Output

For each scenario, you are to print the maximum number of mosquitoes that can be caught under the bowl in that scenario. You may assume that the answer would not change if the diameter of the bowl is increased by at most 10^{-5} .

Example

Input	Output
2 4 1.5 1.0 3.75 3.0 1.0 1.0 2.25 1.5 3.0 8 3.0 -1.0 3.0 -1.0 2.0 -2.0 1.0 0.0 1.0 1.0 0.0 1.0 -1.0 2.0 -2.0 3.0 -1.0	3 4

Problem H. Narrow Art Gallery

Source file name:	narrow.c, narrow.cpp, narrow.java
Input:	`standard`
Output:	`standard`

A long art gallery has $2N$ rooms. The gallery is laid out as N rows of 2 rooms side-by-side. Doors connect all adjacent rooms (north-south and east-west, but not diagonally). The curator has been told that she must close off k of the rooms because of staffing cuts. Visitors must be able to enter using at least one of the two rooms at one end of the gallery, proceed through the gallery, and exit from at least one of the two rooms at the other end. Therefore, the curator must not close off any two rooms that would block passage through the gallery. That is, the curator may not block off two rooms in the same row or two rooms in adjacent rows that touch diagonally. Furthermore, she has determined how much value each room has to the general public, and now she wants to close off those k rooms that leave the most value available to the public, without blocking passage through the gallery.

7	8
4	9
3	7
5	9
7	2
10	3
0	10
3	2
6	3
7	9

Figure H.1: The art gallery shows an optimal solution to the third sample input problem. The gray rooms show those that should be closed.

Input

Input will consist of multiple problem instances (galleries). Each problem instance will begin with a line containing two integers N and k, where $3 \leq N \leq 200$ gives the number of rows, and $0 \leq k \leq N$ gives the number of rooms that must be closed off. This is followed by N rows of two integers, giving the values of the two rooms in that row. Each room's value v satisfies $0 \leq v \leq 100$. A line containing 00 will follow the last gallery.

Output

For each gallery, output the amount of value that the general public may optimally receive, one line per gallery.

Example

Input	Output
6 4 3 1 2 1 1 2 1 3 3 3 0 0 0 0	17
4 3 3 4 1 1 1 1 5 6 0 0	17
10 5 7 8 4 9 3 7 5 9 7 2 10 3 0 10 3 2 6 3 7 9 0 0	102

Problem I. Tractor

Source file name:	tractor.c, tractor.cpp, tractor.java
Input:	`standard`
Output:	`standard`

Bessie the Cow has stolen Farmer John's tractor and is running wild on the coordinate plane! She, however, is a terrible driver, and can only move according to the following rules:

1. Each of her movements is in the same direction as either the positive x-axis or the positive y-axis.
2. Her nth movement takes her 2^{n-1} units forward in her chosen direction. (On her first movement, $n = 1$, so she moves 1 unit.)

Farmer John's farm is on the coordinate plane, in the shape of a rectangle with corners at $(0, 0)$, $(A, 0)$, $(0, B)$ and (A, B). If Bessie starts at $(0, 0)$, how many points inside the farm, including the boundary, could she reach?

Input

The input begins with an integer N ($1 \leq N \leq 100$) on a line by itself, indicating the number of test cases that follow. Each of the following N lines contains two space separated integers A and B ($1 \leq A, B \leq 10^8$), describing the upper-right corner of Farmer John's farm.

Output

Output N lines, with the N-th line containing the number of points that Bessie could possibly reach in the N-th test case.
In the first test case of the sample, Bessie can reach the following six points: $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 2)$,$(2, 1)$ and $(3, 0)$.

Example

Input	Output
2 2 3 7 7	6 15

Problem J. Units

Source file name:	units.c, units.cpp, units.java
Input:	`standard`
Output:	`standard`

The use of units is ubiquitous in science. Physics uses units to distinguish distance (e.g., meters, kilometers), weight (e.g., kilograms, grams), and many other quantities. Computer scientists have specialized units to describe storage capacity (e.g., kibibytes, mebibytes, etc.). You are to write a program to display the conversion factors for a set of units.

Specifying the relationship between various units can be done in many different, but equivalent, ways. For example, the units for metric distance can be specified as the group of relationships between pairs for units: 1km=1000m, 1m=100cm, and 1cm=10mm. An alternative set of pairs consists of: 1km=100000cm, 1km=1000000mm, and 1m=1000mm. In either presentation, the same relationship can be inferred: 1km=1000m=100000cm=1000000mm.

For this problem, the units are to be sorted according to their descending size. For example, among the length units cm, km, m, mm, km is considered the biggest unit since 1km corresponds to a length greater than 1cm, 1m, and 1mm. The remaining units can be sorted similarly. For this set, the sorted order would be: km, m, cm, mm.

This problem is limited to unit-systems whose conversion factors are integer multiples. Thus, factors such as 1 inch = 2.54 cm will not be considered. Further, the set of units and the provide conversions will always permit a larger unit to be expressed as an integer multiple of the next smaller unit.

Input

The input will consist of several problems. Each problem begins with a line giving the number of units, N, where N is an integer in the interval [2, 10]. The following line will contain N unique case-sensitive units, each of which consists of at most 5 characters. Following the set of units will be $N - 1$ unique lines, each specifying a relationship between two different units, with the format containing the following four space-separated pieces: name of the unit; an "="; a positive integer multiplier larger than 1; and the name of a second unit that is smaller than the one to the left of the equal sign. Each of these lines establishes how many units are equivalent to the larger unit on the left. Each unit will appear in the set of $N - 1$ lines and will given in such a way to ensure the entire system is defined.. The set of multiples will yield conversion factors that will not exceed $2^{31} - 1$.

A line containing just a zero will mark the end of all problem descriptions.

Output

For each set of units, produce one line of output that contains the equivalent conversions. The conversions should be sorted left to right, with the largest unit appearing on the left. The conversion factors should be defined with respect to the leftmost unit (i.e., the largest unit) and should be separated by "=".

Example

Input	Output
4 km m mm cm km = 1000 m m = 100 cm cm = 10 mm 4 m mm cm km km = 100000 cm km = 1000000 mm m = 1000 mm 6 MiB Mib KiB Kib B b B = 8 b MiB = 1024 KiB KiB = 1024 B Mib = 1048576 b Mib = 1024 Kib 6 Kib B MiB Mib KiB b B = 8 b MiB = 1048576 B MiB = 1024 KiB MiB = 8192 Kib MiB = 8 Mib 0	1km = 1000m = 100000cm = 1000000mm 1km = 1000m = 100000cm = 1000000mm 1MiB = 8Mib = 1024KiB = 8192Kib = 1048576B = 8388608b 1MiB = 8Mib = 1024KiB = 8192Kib = 1048576B = 8388608b

Problem K. Yikes – Bikes!

Source file name:	yikes.c, yikes.cpp, yikes.java
Input:	`standard`
Output:	`standard`

In recent years the riders in the Tour de France bicycle race have been having problems with dogs running in the road, causing expensive damage to bicycles and serious injuries to riders. In this problem you will take the point of view of a dog named Max. Just as the peloton approaches, Max notices a juicy rabbit in the field on the opposite side of the road. He decides to make a dash across the road. Your task is to determine if Max will safely cross the road if he begins his dash at a given time.

The road is ten meters wide and straight in this part of the country. Max's dash will be in a straight line at ninety degrees to the road's center line. He will run at a constant speed of M $(0 < M < 20)$ meters per second. His body can be represented by a circle of diameter one meter projected onto the road surface. See the diagram below.

All the cyclists are in single file racing down the center line of the road, all at the same speed, B $(0 < B < 40)$ meters per second. Each bicycle can be represented by a line segment of length two meters. Its projection onto the road's surface is collinear with the center line of the road. The bicycles are spaced such that their projections form a dashed line with a distance of two meters between the end of one line segment and the beginning of the next. If, at any time during Max's dash, his circle touches or is intersected by one or more line segments representing the bicycles, a collision will occur.

Assume that at $t = 0$, Max is in position to begin his dash (His circle is completely on the sidewalk, with the edge of the road collinear with one edge of Max's circle). Assume also that, at $t = 0$, the front endpoint of the line segment used to represent the lead bicycle is exactly D meters from the line that Max will take to cross the highway. There are 10 cyclists in the peloton.

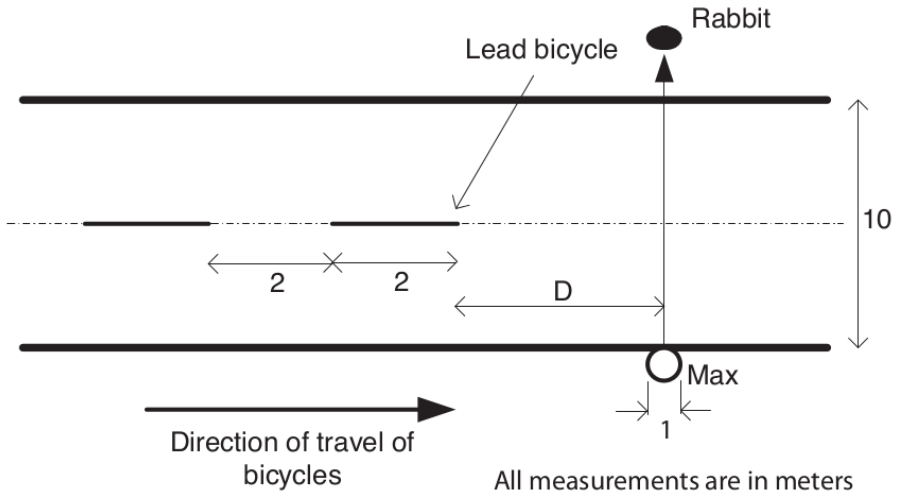


Figure K.1: Initial Setup for Max's Dash

Input

Input begins with a positive integer N ($1 \leq N \leq 40$) giving the number of problem sets to follow. Then, follows four lines for each problem set, each line containing one value as follows:

- a real number M ($0 < M < 20$) giving Max's speed in meters per second, then
- a real number B ($0 < B < 40$) giving the speed of the cyclists in meters per second, then
- a real number D ($0 \leq D \leq 50$) giving the distance in meters between the front of the lead bicycle at $t = 0$ and the line that Max will use to dash across the highway
- a real number T ($0 \leq T \leq 20$) giving the start time in seconds of Max's dash

Output

For each problem set output a single line giving one of the phrases, "Max beats the first bicycle", "Max crosses safely after bicycle k", ($1 \leq k \leq 10$), or "Collision with bicycle k" ($1 \leq k \leq 10$). If multiple collisions could occur, report only the first.

Example

Input	Output
3 5.0 5.0 5.0 0.0 5.0 5.0 6.3 0.0 5.0 5.0 0.0 5.9	Collision with bicycle 1 Max beats the first bicycle Max crosses safely after bicycle 9