Problem A
EVEN vs. ODD

Two executives, Joe and Sarah, decided to have a friendly wager about the employees at their company. Joe thought that most of the employees were born in even years, while Sarah thought that most of the employees were born in odd years. They decided to ask you to write a program that would settle their bet. Your program must read the years that the employees were born, and output the number of them that are odd, and the number of them that are even. Then, it should display who won the bet (Joe or Sarah).

Input

You should read from standard input. The first line contains a single integer indicating how many employees there are. It is then followed by a series of years, one per line, representing the birth year of each employee.

Output

Make sure to format the output as shown below. You can assume that all of the birth years are between 1900 and 2013. You can also assume that there are no more than 50 employees. Finally, if there is a tie (that is, if the number of odd birth years is the same as the number of even birth years), then display that Sarah is the winner.

Sample Input	Sample Output
7 1962 1993 2001 1977 1998 1996 1995	Number of odds: 4 Number of evens: 3 The winner is Sarah.

Problem B
Happy Calc

You have decided that you can make *a lot* of money by selling technology products customized for various cultures. One such culture believes that palindromes are unlucky. So, your first product idea is a calculator that refuses to do any calculations involving palindromes. Your first step is to make a prototype of a simple version that just handles addition. If the calculator is ever asked to add or produce a palindromic number, it will refuse, declaring the computation unlucky. A number is a palindrome if its digits read forward and backward are the same. The numbers 11, 4, 323, 7997 are all palindromes whereas the numbers 98, 123, 2226 are not.

Input

Your program should read from standard input. The first line of input tells how many addition problems follow. Each problem is then given as a single line, with each line containing two non-negative integers. (Each integer and their sum is guaranteed to fit in a variable of type `int`, and will not start with a zero unless it is the number zero being represented.)

Output

For each problem, your program should add the two numbers and output the result on its own line without any leading zeros (unless zero is the answer of course). However, if one of the input numbers or the output number is a palindrome then your program should output "Unlucky! Unlucky!" instead.

Sample Input	Sample Output
3 1000 2000 123 321 11 12	3000 Unlucky! Unlucky! Unlucky! Unlucky!

Problem C
Two Step

We all have heard the saying "two steps forward and one step back". Presumably this means that you take two steps forward during time-unit 1 and then take one step back during time-unit 2. If you repeat this pattern, then after 5 time units, you will be 4 steps forward from your starting point ($+2 - 1 + 2 - 1 + 2$).

Write a program that reads a step pattern and a number of time units and calculates the number of steps you'd be away from your starting place if you followed the step pattern repeatedly for that many time units.

Input

You are to read from standard input. On the first line will be a positive integer indicating how many more lines of input you will be reading. Each subsequent line begins with two integers, the first is a non-negative time duration, the second is a positive value indicating how many numbers are in the step pattern you'll be following. There are then as many integers as this second number says there should be on the remainder of the line, indicating the step pattern. Positive numbers in the step pattern should be interpreted as steps forward while negative numbers indicate steps back. So, specifying the "two steps forward and one step back" pattern for five time units would look like `5 2 2 -1`.

Output

For each line that indicates a number of time units and step pattern, output a single line with a single integer indicating where you'd be relative to your starting position if you followed that pattern for that many time units.

Sample Input	Sample Output
2 2 2 +1 -2 5 2 +2 -1	-1 4

Problem D
Transitive Relations

A relation R on a set X is transitive if, for all x, y, z in X, whenever xRy and yRz then xRz. Examples of transitive relations include the equality relation on any set and the "less than" relation on any linearly ordered set. For example, less-than is transitive on the integers because if $x < y$ and $y < z$ then $x < z$ is true for all integers x, y, z. In this problem you are to write a program to check whether a relation is transitive or not.

Input

Your program should read relations from standard input. The first line of the input contains a number indicating the number of relations to be checked. There will then be that many relations. Each relation R is represented by a square 0/1 matrix M (guaranteed not to be all zeros). A 1 at column i and row j of M indicates that iRj is true. There will be no whitespace in the input except a single newline at the end of each line.

Output

For each relation, your program should print a single line with "yes" or "no" indicating whether the relation is transitive.

Sample Input	Sample Output
2 1010 0011 1100 0100 10011 01100 01100 10011 10011	no yes

Problem E
Lags

To play a game of pocket billiard (pool), players first shoot "lags" to decide who starts (breaks) the game.

A lag is using a cue stick to hit a billiard ball from one end of the table toward the far end to bounce it back toward the starting end. The person that can hit the ball to roll to a stop that is nearest to the starting end is the person to "break" (start) the billiard game, and the person who has the ball roll to a stop the second nearest plays second, etc.

Rules for this lag problem::

1. A lag is shot exactly from one end (the player's end) of a 10-foot table (the surface is 10-foot-long).
2. A lag shot is always length-wise straight across the table, parallel with the sides of the table.
3. The ball must first hit the far end of the table if there is any bouncing on the table. Multiple bounces on both ends are allowed.
4. If the ball doesn't even travel 10 feet to reach the far end of the table as its first bounce, then the lag result will include the missing 10 feet since it's required to bounce at least once. For example, if a ball travels only 1 foot, the lag result will be 19 feet (almost twice the table length).
5. The rolling speed of a billiard ball reduces 3 feet per second after each second until it becomes 0 (no negative value).
6. A ball can be hit with a spin so a "spin factor" is given. There are positive and negative spin factors which increases or decreases the speed of the ball: a positive spin increases the speed by the number of feet per second it indicates, while a negative one reduces it.
7. A spin factor also decreases in its magnitude (absolute value) by a foot per second after each second until it becomes 0.
8. Speed and distance calculations are based on whole seconds. If a ball has speed v at the beginning of a second, it moves v during that second.
9. In the second the ball produces zero or negative distance (calculated by the speed and spin factor), the ball is considered to have no movement during this second and have stopped at the end of the previous second.
10. Assume bouncing does not change a speed or spin factor.

Example: a ball has an initial speed of 7 feet per second, and has a positive spin factor of 4 (forward spin). In the 1st second it rolls 11 feet: 7 + 4. In the 2nd second it rolls 7 feet: (7-3) + (4-1) = 4 + 3. In the 3rd second it rolls 3 feet: (4-3) + (3-1) = 1 + 2. In the 4th second it rolls 1 foot: (1-3) + (2-1) = 0 + 1. In the 5th second it rolls 0 foot: 0 + (1-1) = 0 + 0. Hence, the ball rolls 22 feet in 4 seconds. The ball bounces twice: fist at the far end, second at the player's end; and stops 2 feet away from the player's end (lag result). If the spin factor were -2 in the previous example, the ball travels 9 feet in 3 seconds (lag result 11 feet) since: in the 1st second 5 feet: 7 + (-2), in the 2nd second 3 feet: 4 + (-1), in the 3rd second 1 foot: 1 + 0, and in the 4th second 0 foot: 0 + 0.

Input

Write a program that reads from standard input, which will have data in the following format: 1st line is the number of players (an integer ranging from 2 to 5). Each line (for a total of lines indicated from the previous line) that follows has: the single-word name (at most 20 characters) of a player, a single space character, the initial speed (an integer ranging from 0 to 100) of the ball hit by the player, another space character, the initial spin factor of the ball hit by the player (an integer ranging from -100 to 100), and lastly, a newline character.

Output

Output by the same order of the input with each line having a player's name and his/her lag result, which are the total distance the ball rolled, the number of seconds the ball rolled, and the lag distance. One player a line, starting from the 1st column, and ending with a newline character.

Sample Input	Sample Output
3 TOM 10 0 SANDY 8 10 DAVE 15 -10	TOM 22 4 2 SANDY 70 10 10 DAVE 9 3 11

Problem F
I Love Squares

You really like squares. Write a program that draws concentric squares using the asterisk character.

Input

Your program should read two integers from standard input. The first is positive and indicates the number of concentric squares to draw and the second is non-negative and indicates how many spaces should separate each square.

Output

Your innermost square is so small that it should be drawn as a single '*' character. Each subsequent square should be drawn so that each of its sides has the indicated number of spaces separating them from the sides of the next innermost square. (Note, your output probably won't look like a square when it prints to the screen because newlines are taller than characters are wide. What's important here is that the number of rows matches the number of columns.) To facilitate judging, your figure should be made entirely of the space and asterisk characters and there should be no extra spaces at the start or end of each line.

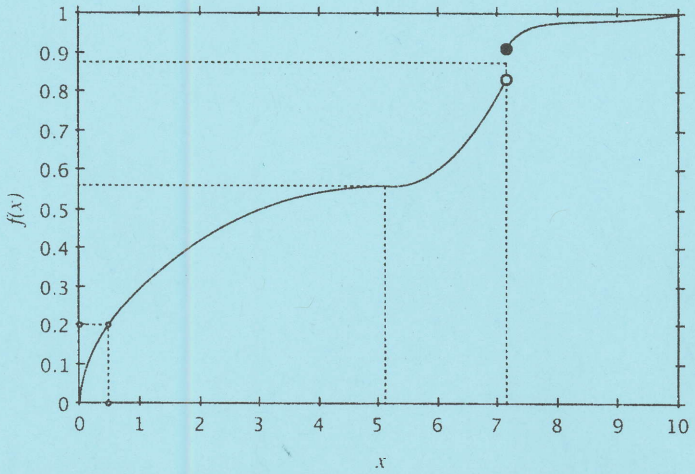
Sample Input	Sample Output
4 2	**********
	* *
	* *
	* ******** *
	* * * *
	* * * *
	* * ******** * *
	* * * * *
	* * * * *
	* * * * * * *
	* * * * *
	* * * * *
	* * ******** * *
	* * * *
	* * * *
	* ********** *
	* *
	* *
	**********

Problem G
Non-uniform

A cumulative distribution function (cdf) is a non-decreasing function from a set of numbers to the range [0,1]. (The notation $[a, b]$ is shorthand for the set $\{x \mid x \in \mathbb{R}, a \leq x \leq b\}$.) When f is a cdf describing the probability distribution of random variable X, the function $f(x)$ is defined as $\Pr[X \leq x]$.

A cdf f can be used to generate random values for the probability distribution described by f. Here's how you do it. First, you choose a uniformly distributed random real y in [0,1]. Next, you determine which is the smallest value of x from f's domain that makes $f(x) \geq y$. This process yields x which is randomly distributed according to the probability distribution described by f.

As an example, here is a hypothetical cdf graph from [0,10] to [0,1]. If you wanted a random value in [0,10] according to this distribution, you'd choose a uniform random value in [0,1], draw a horizontal line to find which x is the first to make $f(x) \geq y$, and that x would be your random value in [0,10].



Write a program that generates random values according to the exponential distribution with parameter λ and random values according to the geometric distribution with parameter p. The exponential cdf is $f(x) = 1 - e^{-\lambda x}$ which maps non-negative reals to [0,1]. The geometric cdf is $f(x) = 1 - (1 - p)^{x+1}$ which maps non-negative integers to [0,1]. The values of λ and p are constants in each cdf and will be supplied as part of the problem input.

Input

You are to read from standard input, first a positive integer n, and then a sequence of n triples of real numbers. Each triple will consist of random y in [0,1], positive real λ, and p in [0,1].

Output

For each triple you should output a single line consisting of the exponentially distributed value indicated by y and λ, and the geometrically distributed value indicated by y and p. Values within 0.001 of the correct answer will be considered correct.

Sample Input	Sample Output
2 0.5 1.0 0.25 0.0 2.0 0.1	0.6931 2 0.0000 0

Problem H
Battleship!

Two battleship commanders are meeting, looking at a map, and planning a rendezvous. The map shows routes between points in the sea, and the expected battle cost for traveling each route. To increase the chance that at least one of them reach the rendezvous point, the commanders decide to take completely different routes and wish to minimize their combined total battle costs. This is when they call you in, Ensign Boole.

Write a program that takes a description of the map and finds routes for the two battleships which never visit a common point (except their starting and rendezvous points) and whose combined sum total expected battle cost is minimal.

Input

You are to read from standard input one or more test cases. Each test case has its own map. Each map is described entirely as a sequence of integers separated by whitespace. The first integer n tells how many points are in the map ($3 \leq n \leq 1000$). The second integer m tells how many routes are in the map ($3 \leq m \leq 10000$). There are then m triples of integers a_i, b_i, c_i, which indicate the expected battle cost of traveling from Point a_i to Point b_i is c_i. For each triple, a_i and b_i are both in the range $1 \ldots n$, $a_i \neq b_i$ and $1 \leq c_i \leq 100$. The starting point is Point 1 and the rendezvous point is Point n. Route triples a_i, b_i, c_i are direct (meaning that one can travel from a_i to b_i without visiting any other point at cost c_i) and may not be symmetric (meaning one should not infer that there is also a route from b_i to a_i with the same cost). Subsequent test case descriptions begin immediately after the previous one.

Output

For each test case, your program should output a single integer indicating the minimal joint expected battle cost for the two battleships to get to their rendezvous without crossing paths.

Sample Input	Sample Output
3 3 1 2 10 1 3 10 2 3 10 6 11 1 2 23 1 3 12 1 4 99 2 5 17 2 6 73 3 5 3 3 6 21 4 6 8 5 2 33 5 4 5 6 5 20	30 86