Oddities
Problem ID: oddities

Some numbers are just, well, odd. For example, the number 3 is odd, because it is not a multiple of two. Numbers that are a multiple of two are not odd, they are even. More precisely, if a number n can be expressed as $n = 2 * k$ for some integer k, then n is even. For example, $6 = 2 * 3$ is even.

Some people get confused about whether numbers are odd or even. To see a common example, do an internet search for the query "is zero even or odd?" (Don't search for this now! You have a problem to solve!)

Write a program to help these confused people.

Input

Input begins with an integer $1 \leq n \leq 20$ on a line by itself, indicating the number of test cases that follow. Each of the following n lines contain a test case consisting of a single integer $-10 \leq x \leq 10$.

Output

For each x, print either 'x is odd' or 'x is even' depending on whether x is odd or even.

Sample Input 1	Sample Output 1
3 10 9 -5	10 is even 9 is odd -5 is odd

Ternarian Weights
Problem ID: ternarianweights

Back in the Hellenic era, there was a small island in the Mediterranean Sea known as Ternaria. It was close to Sparta, but because of its mountainous terrain the Spartans found it difficult to conquer and it remained an independent state until the great earthquake of 729BC when it sank beneath the sea. It had a remarkable civilization and some modern historians think it is the basis for the mythological city of Atlantis. Ternaria is still known for its fundamental contributions to science and mathematics, many of which were adopted by the Greeks and later by the Romans. For example, Ternarians were the first group to use the standard weight of pounds, which we still use today. Ternarian mathematics used base 3 for all its calculations. (Historians speculate that this was out of respect for King Ternary who lost two fingers on each hand while battling the Spartans.)

Ternarian trade scales were a standard for many centuries. They were known for their accuracy and ease of use. They were the first to construct a scale with weighing pans on each side and a fulcrum in the middle. The object to be weighed was placed on the left side of the scale and weights were placed on both sides, until balance was obtained. This sounds strange by modern standards, because typically on modern scales we would only place weights on the right side. However, the modern method requires additional weights. The Ternarian method only requires one weight for each power of three pounds, e.g. one weight of 1 pound, one weight of 3 pounds, one weight of 9 pounds, etc.

Say you are weighing a 2-pound Ternarian hen (known for their succulent white meat). You place the hen on the left side. Place the 3-pound weight on the right side. This is too heavy, so you place the 1-pound weight with the hen on the left side to achieve balance. Note that the sum of the weights on the right side minus the sum of the weights on the left side equals the weight of the object.

As another example, consider weighing a 21-pound Ternarian squash. Using the Ternarian system, you would place weights of 27 pounds and 3 pounds in the right pan and a weight of 9 pounds in the left pan (along with the object) again achieving balance.

Write a program that accepts as input the weight of an object in base 10 and outputs the weights to be placed in both pans.

Input

The first line contains an integer $1 \leq n \leq 100$, indicating how many test cases are to be solved. On each of the next n lines there is an integer $0 \leq x \leq 10^9$ giving the weight of the object placed on the left scale.

Output

For each test case the program should produce two lines of output. The first line should contain `left pan:` followed by a space, followed by the weights to be placed in the left pan in descending order. The second line should contain `right pan:` followed by a space, followed by the weights to be placed in the right pan in descending order. Print a blank line between each pair of test cases.

Sample Input 1

```
4
2
3
21
250
```

Sample Output 1

```
left pan: 1
right pan: 3

left pan:
right pan: 3

left pan: 9
right pan: 27 3

left pan: 3
right pan: 243 9 1
```

Goldbach's Conjecture
Problem ID: goldbach

The goal of this program is to find all unique ways to represent a given even number as the sum of two prime numbers. A prime number is an integer greater than 1 that is evenly divisible by only itself and 1. The first few prime numbers are 2, 3, 5, 7, 11,... The German mathematician Christian Goldbach (1690-1764) conjectured that every even number greater than 2 can be represented by the sum of two prime numbers. (This conjecture has never been proved nor has a counterexample ever been found. As such, you may assume it is true for the cases considered in this problem.) There may be several ways to represent a given even number by the sum of primes. For example, the even number 26 may be represented as 3 + 23, 7 + 19, or 13 + 13.

Input

Input starts with an integer n ($1 \leq n \leq 100$) indicating the number of cases. The following n lines each contain a test case of a single even number x ($4 \leq x \leq 32000$).

Output

For each test case x, give the number of unique ways that x can be represented as a sum of two primes. Then list the sums (one sum per line) in increasing order of the first addend. The first addend must always be less than or equal to the second to avoid duplicates. Print a blank line between each pair of test cases.

Sample Input 1

```
3
4
26
100
```

Sample Output 1

```
4 has 1 representation(s)
2+2

26 has 3 representation(s)
3+23
7+19
13+13

100 has 6 representation(s)
3+97
11+89
17+83
29+71
41+59
47+53
```

Shotcube
Problem ID: shotcube

Namco Bandai's video game 'Tales of Graces' introduces a puzzle minigame known as Shotcube. In this game, you have 9 cubes placed on a 7-by-7 square grid. The objective is to rearrange the cubes so that they all lie in a 3-by-3 square. In the video game, this 3-by-3 square must be in the exact center of the grid; for this problem, the 3-by-3 square may be anywhere on the grid.

The only way to rearrange the cubes is to "shoot" at them from the outside of the grid. If there is a cube adjacent to the edge of the grid, you may "shoot" it to push it in a straight line, in which it will keep moving until it hits another cube, at which point it stops. You may only shoot a cube if there is another cube in its path which stops it. If you shoot a cube in a direction such that one or more cubes are immediately behind it, all of those cubes will move in unison until the farthest cube hits another cube (after having travelled at least one grid square), at which point all of them stop.

Consider the grid below. The arrows mark the three legal shots; no other shots are legal, either because there is no cube adjacent to the edge of the grid, or because there is no cube present to stop the motion of the shot cubes. The three grids below that indicate the result of shooting the first row to the right, the first column down, and the fifth column down, respectively.

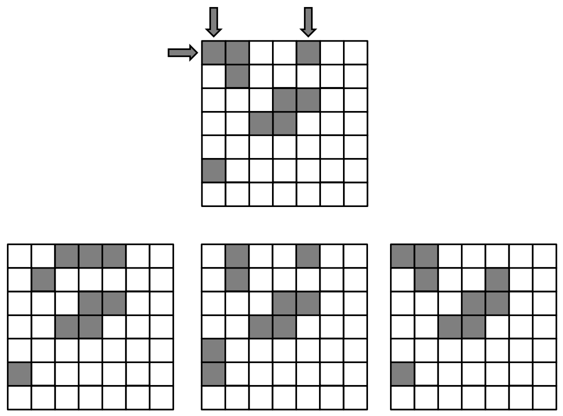

Figure 1: Illustration of legal shots and their outcomes.

Input

Input begins with a line with a single integer T, $1 \leq T \leq 10000$ denoting the number of test cases. Each test case consists of 7 lines, each with 7 characters, representing the 7-by-7 grid. Each character is either a "." (period) or an "X"; a "." indicates that that square is empty, while an "X" indicates that that square has a cube. Each test case is guaranteed to have exactly 9 cubes. Each pair of test cases is separated by a single blank line.

Output

For each test case, print out the minimum number of shots needed to arrange the cubes into a 3-by-3 square anywhere on the grid, following the rules described above. If it is impossible to do so, print out the number "-1" instead.

Sample Input 1

```
2
...X...
...X...
..X.X..
..XXX..
..X.X..
.......
.......

.......
....XXX
....XXX
......X
.......
.......
X....X.
```

Sample Output 1

```
-1
3
```

Erratic Ants
Problem ID: erraticants

The ants of a particular colony are in search of food. Unfortunately hidden dangers are all around the colony which makes foraging difficult. There are traps, obstacles, and predators lurking about. Fortunately, the colony has the perfect ant for the job. Max is neither a smart ant nor an efficient ant but he has got blind luck on his side. In all of his wanderings, he has always managed to stay on safe ground and he (eventually) always finds a source of food to report back to the colony.

The problem is that Max rarely takes anything resembling an optimal (shortest) route to find a food source. However, Max can reliably bring back the exact details of the (often winding and convoluted) path that he took to get to the food source. Your job is to help the colony by finding the optimal route located within Max's convoluted directions to allow the colony to forage more efficiently.

Input

The first integer in the input, $1 \leq n \leq 100$, denotes the number of paths to food that Max has reported back. This is followed by a blank line and then descriptions of each of the n paths. Each path description begins with an integer, s ($0 \leq s \leq 60$), which denotes the number of steps taken in the path. The next s lines contain directional steps expressed as upper-case characters N, E, S, and W corresponding to steps taken in the directions north, east, south, and west respectively. Each step moves Max one unit of distance. Max's paths always start at the colony and end at a food source. Between each pair of path descriptions is a blank line.

When searching for an optimal path, the only directional steps that may be taken are ones that have previously been taken by Max, or the same steps in reverse.

Output

For each given path, give the number of steps found to be in an optimal (shortest) path.

Sample Input 1	Sample Output 1

```
3

8
S
E
E
E
N
W
S
S

4
S
E
N
W

3
S
E
N
```
	```
	4
	0
	3
	```

Plotting Polynomials
Problem ID: plot

Graphical calculators have become popular among high school students. They allow functions to be plotted on screen with minimal efforts by the students. These calculators generally do not possess very fast processors. In this problem, you are asked to implement a method to speed up the plotting of a polynomial.

Given a polynomial $p(x) = a_n x^n + ... + a_1 x + a_0$ of degree n, we wish to plot this polynomial at the m integer points $x = 0, 1, \ldots, m-1$. A straightforward evaluation at these points requires mn multiplications and mn additions.

One way to speed up the computation is to make use of results computed previously. For example, if $p(x) = a_1 x + a_0$ and $p(i)$ has already been computed, then $p(i+1) = p(i) + a_1$. Thus, each successive value of $p(x)$ can be computed with one addition each.

In general, we can compute $p(i+1)$ from $p(i)$ with n additions, after the appropriate initialization has been done. If we initialize the constants C_0, C_1, ..., C_n appropriately, one can compute $p(i)$ using the following pseudocode:

```
p(0) = C_0; t_1 = C_1; ... t_n = C_n;
for i from 1 to m-1 do
     p(i)    = p(i-1)  + t_1;
     t_1     = t_1     + t_2;
     t_2     = t_2     + t_3;
               :
               :
     t_(n-1) = t_(n-1) + t_n;
end
```

For example, if $p(x) = a_1 x + a_0$, we can initialize $C_0 = a_0$ and $C_1 = a_1$.

Your task is to compute the constants $C_0, C_1, \ldots, C_n$ for the above pseudocode to give the correct values for $p(i)$ at $i = 0, \ldots, m-1$.

Input

The input consists of one case specified on a single line. The first integer is n, where $1 \leq n \leq 6$. This is followed by $n+1$ integer coefficients $a_n, \ldots, a_1, a_0$. You may assume that $|a_i| \leq 50$ for all i, and $a_n \neq 0$.

Output

Print the integers $C_0, C_1, \ldots, C_n$, separated by spaces.

Sample Input 1	Sample Output 1
`1 5 2`	`2 5`

Sample Input 2	Sample Output 2
`2 2 -4 5`	`5 -2 4`

Sample Input 3	Sample Output 3
`6 1 -20 4 30 0 -1 2`	`2 14 -302 -2136 -3144 -600 720`

Trapezoid Walkway
Problem ID: walkway

You are planning to place a small, prefabricated gazebo in your back yard, with a paved walkway connecting it to your back porch. You'll build the walkway out of paving stones purchased at your local home improvement store. The paving stones come in a variety of sizes, but they are all shaped like isosceles trapezoids. As illustrated on the left of Figure 1, an isosceles trapezoid is what you get if you take an isosceles triangle and cut off a corner with a line that is parallel to the base. We can describe such a paving stone with three parameters: the length of one of its parallel edges, a, the length of the other parallel edge, b, and the perpendicular distance, h, between these edges.

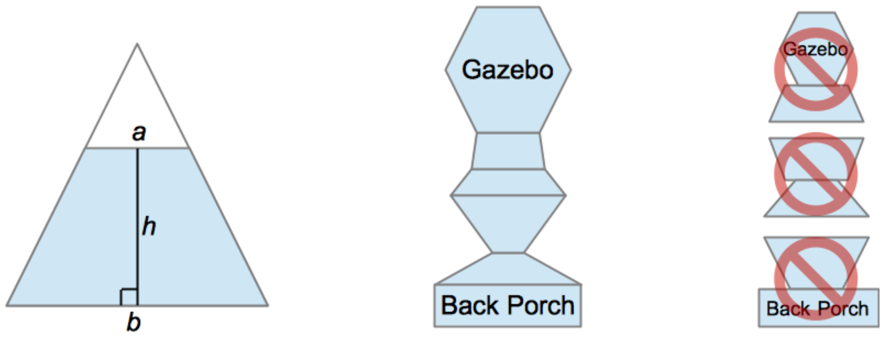

Figure 1: Isosceles trapezoid shape (left), walkway example (center) and join requirements (right)

You're going to build your walkway by joining the parallel edges of the paving stones, with one parallel edge meeting the back porch at the start and another meeting the gazebo at the end. You want your walkway to look nice, so you won't permit any of the situations illustrated on the right of Figure 1. Where two paving stones meet, their edges must be exactly the same length. Likewise, where a paving stone meets the back porch or the gazebo, its edge must be exactly the same length. Fortunately, your home improvement store has a wide selection of different trapezoid-shaped paving stones, so you are confident you can build a walkway that satisfies these requirements.

Paving stones are priced at two cents per square centimeter of surface area. You have a big back yard, so you don't care how long your walkway is. You just want the one that's the least expensive.

Input

Input will consist of multiple test cases. Each test case will start with a positive integer, n, the number of different types of paving stones available. The value, n, will be at least 1 and no greater than 1000. The next n lines each describe a type of stone by giving the lengths of its sides, a, b then h. These three values will be positive integers, measured in centimeters and not greater than 1000. No two types of stones are identical, but your home improvement store has a large stock of paving stones, so you can buy as many of each type as you need. The last line of each test case gives the width of the back porch, where the walkway will start, followed by the width of the edge of the gazebo where the walkway will end. A value of zero for n will mark the end of all test cases.

Output

For each test case, print the total cost, in dollars, for the least expensive walkway that meets your requirements. It will always be possible to build such a walkway.

Sample Input 1

```
6
120 350 60
120 150 95
240 300 60
240 350 220
150 300 100
300 350 120
120 240
2
100 140 50
100 140 80
140 100
2
150 250 100
150 250 60
150 150
0
```

Sample Output 1

```
1030.50
120.00
0.00
```

4 thought
Problem ID: 4thought

Write a program which, given an integer n as input, will produce a mathematical expression whose solution is n. The solution is restricted to using exactly four 4's and exactly three of the binary operations selected from the set $\{*, +, -, /\}$. The number 4 is the ONLY number you can use. You are not allowed to concatenate fours to generate other numbers, such as 44 or 444.

For example given $n = 0$, a solution is $4 * 4 - 4 * 4 = 0$. Given $n = 7$, a solution is $4 + 4 - 4 \,/\, 4 = 7$. Division is considered truncating integer division, so that 1/4 is 0 (instead of 0.25). Assume the usual precedence of operations so that $4 + 4 * 4 = 20$, not 32. Not all integer inputs have solutions using four 4's with the aforementioned restrictions (consider $n = 11$).

Hint: Using your forehead and some forethought should make an answer forthcoming. When in doubt use the fourth.

Input

Input begins with an integer $1 \le m \le 1000$, indicating the number of test cases that follow. Each of the next m lines contain exactly one integer value for n in the range $-1000000 \le n \le 1000000$.

Output

For each test case print one line of output containing either an equation using four 4's to reach the target number or the phrase `no solution`. Print the equation following the format of the sample output; use spaces to separate the numbers and symbols printed. If there is more than one such equation which evaluates to the target integer, print any one of them.

Sample Input 1

```
5
9
0
7
11
24
```

Sample Output 1

```
4 + 4 + 4 / 4 = 9
4 * 4 - 4 * 4 = 0
4 + 4 - 4 / 4 = 7
no solution
4 * 4 + 4 + 4 = 24
```

Biggest Slice
Problem ID: biggest

You are sharing a large, circular pizza with $n - 1$ of your friends. Your technique for slicing the pizza is shown in Figure 1; you rotate the pizza clockwise about its center by angle θ, and then you make a slice from the center of the pizza straight to the right. You repeat this process, rotating by the same angle θ and slicing to the right until you have done it a total of n times.

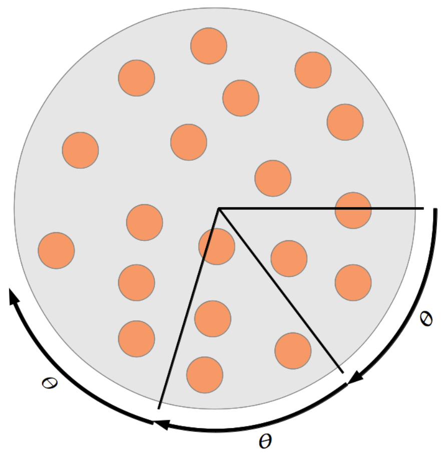

Figure 1: Rotate-and-slice pizza division technique.

Of course, this isn't really a good way to divide a pizza (unless θ is well-chosen). Some of the resulting slices may be larger than others, and you may not even end up with n different slices. You don't care so much about fairness. You just want to know how big the largest slice will be, so you can take it for yourself.

Input

Input begins with an integer $1 \leq m \leq 200$ indicating the number of test cases that follow. The following m lines each contain one test case. Each test case gives the pizza radius in centimeters, r, followed by the number of people sharing the pizza, n, followed by the rotation angle, θ. The quantities r, n and θ are all positive. The value r is an integer no greater than 100, and n is an integer no greater than 10^8. The angle θ is given as an integer number of degrees, followed by an integer number of minutes and an integer number of seconds. Degrees are between 0 and 359 (inclusive), while minutes and seconds are between 0 and 59 (inclusive).

Output

For each test case, print the area in square centimeters of the largest resulting slice of pizza. You do not need to worry about the precise formatting of the answer (e.g. number of places past the decimal), but the absolute error of your output must be smaller than 10^{-4}.

Sample Input 1	Sample Output 1

4
20 6 60 0 0
20 6 59 59 59
30 20 33 30 0
40 200 120 0 0
	209.439510
	209.444358
	263.108385
	1675.516082

Even Up Solitaire
Problem ID: evenup

The Even Up Solitaire can be played with a stack of cards each having a numerical value from 1 to 100. The cards are laid out from left to right in a row. At every step, the player is allowed to remove two adjacent cards if the sum of their values is even. The gap is then “closed” by shifting the cards to the right of the gap. The order of the remaining cards is not changed. The game stops when all cards are removed or when no more cards can be removed. The player wins when all cards are removed. If this is not possible, the player should try to minimize the number of cards remaining.

You are given the initial list of cards, in left-to-right order. Determine the minimum number of cards that remain if the player moves optimally.

Input

The input consists of one case. The first line contains an integer n ($1 \leq n \leq 100000$) giving the number of cards to follow. The second line contains n integers indicating the card values from left to right. Each card value is in the range 1 to 100.

Output

Print the minimum number of cards that remain if the player moves optimally.

Sample Input 1	Sample Output 1
10 1 2 3 4 5 6 7 8 9 10	10

Sample Input 2	Sample Output 2
10 1 3 3 4 2 4 1 3 7 1	2

Island Buses
Problem ID: island

The Island Cargo and Passenger Company is planning to provide bus service for a number of South Pacific island nations. Each nation consists of a collection of islands, some of which are connected by bridges. To save costs, the company wants to use the fewest number of buses possible. Every island must have access to a bus, but the company will only use one bus for any group of two or more islands that is connected by bridges.

Your job is to write a program which, given a map of the island nation, computes the number of islands, bridges, and the smallest number of buses needed for that country.

Input

Input contains a sequence of maps, each given as a rectangular array of characters (at most 80 by 80). There is one blank line between each pair of maps. Input ends at end of file.

All maps contain only the characters dot (.), X, #, and B. Dots represent ocean water. X's and #'s represent island land. B's represents bridges, and each X represents the land that is the endpoint of one or more bridges.

Each map contains 1 or more rectangular islands. Different islands are not adjacent vertically or horizontally. Diagonal adjacency is not enough to drive a bus across.

Each map contains 0 or more horizontal or vertical bridges. Each bridge has at least one B, and connects only the two islands at its two endpoints. Bridges do not cross each other or extend over the # land of an island.

Output

For each map you should print the map number, followed by lines giving the number of islands, bridges, and buses needed. Print a blank line between each pair of map outputs. Follow the format of the sample output.

Sample Input 1

```
....................
....................
.....###............
.....##XBBBBX.......
.....###............
....................
.............###....
....####............
....####............
....###XBBBBX.......
.......B....#.......
.......B....#.......
.......B....X.##....
.......B....B.##....
...####X####X#......
...###########......
...###########......
...###########......
...###########......
....................

....######X###......##X##.......
..........B...........B.........
.........#X###########X###......

.......
.#.....
..#....
....##.
....##.
.......
.##....
...#...
.....#.

....##...
....##...
.#XBBBBX#
.##....##
```

Sample Output 1

```
Map 1
islands: 7
bridges: 4
buses needed: 4

Map 2
islands: 3
bridges: 2
buses needed: 1

Map 3
islands: 6
bridges: 0
buses needed: 6

Map 4
islands: 3
bridges: 1
buses needed: 2
```