Problem A: Armed bandit

You are in a casino, and you are anxious to play on the new *one-armed bandit* – a slot machine they just installed.

The slot machine has n wheels in a row. The i-th wheel from the left contains the integers from 1 to k_i, inclusive. When you pull the handle, the wheels will spin for a while. When they stop, each wheel will show you one of its integers. The integers are randomly chosen. All random choices are uniform and mutually independent.

Problem specification

You are given the number of wheels n and the size of every wheel k_i.

Easy subproblem A1: Output any sequence of numbers you could see on the given slot machine.

Hard subproblem A2: Suppose we ignore the spaces between the wheels and read the sequence of numbers as one long string of digits. Find the sequence of numbers that produces the *lexicographically smallest* such string.

Lexicographic order

When comparing two different strings S and T, find the smallest index i at which they differ. The one with a smaller character at that index is lexicographically smaller than the other. If there is no such character (i.e., one of the strings is a prefix of the other), the shorter string is the lexicographically smaller one.

For example:

- `22` is lexicographically smaller than `220` because `22` is a prefix of `220`
- `123` is lexicographically smaller than `14` because `2` is less than `4`.

Input specification

The first line of the input file contains an integer $t \leq 100$ specifying the number of test cases. Each test case is preceded by a blank line.

Each test case begins with a line consisting of a single integers n. The following line contains n space-separated integers $k_1, \ldots, k_n$. In both subproblems we have $1 \leq n \leq 1\,000$ and $\forall i : 1 \leq k_i \leq 100$.

Output specification

For each test case, output a single line with n integers separated by single spaces: the numbers as they appear on the individual wheels of the machine.

What to submit

Do not submit any programs. Your task is to produce and submit the correct **output files** `a1.out` and `a2.out` for the provided input files `a1.in` and `a2.in`.

Example for subproblem A1

input

```
2

3
6 6 6

2
10 2
```

output for A1

```
2 4 6
7 2
```

There are many other correct outputs. Any correct output will be accepted.

Example for subproblem A2

input

```
2

3
6 6 6

2
10 2
```

output for A2

```
1 1 1
10 1
```

The lexicographically smallest string consisting of 3 integers from 1 to 6 (inclusive) is `111`.

The lexicographically smallest string consisting of an integer from 1 to 10 (inclusive) and an integer from 1 to 2 (inclusive) is `101`. Note that this string is smaller than `11` because the second character in `101` is smaller than the second character in `11`.

Problem B: Brain fold

Shandyna loves nerd sniping. (Relevant xkcd. Don't worry, she only does it in safe enviroments.)

Last time she got five of them at once. When I saw them, they were still sitting around, trying to imagine a folded sheet of paper.

Problem specification

You have a rectangular piece of paper. You fold it in half several times. For each fold, you pick up one side and place it over the opposite side. There are four sides to pick, so there are four ways to perform each fold. (Two of the folds are horizontal and two are vertical.)

Once you finish the last fold, you pick a straight line and cut along it, through all layers of the folded paper. How many pieces of paper will you have at the end?

Input specification

The first line of the input file contains an integer $t = 100$ specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of three lines.

The first line contains a number n ($1 \leq n \leq 10^5$) denoting the number of folds.

The second line consists of n characters specifying the folds in the order they are applied: `T`, `B`, `L` and `R` represent folds for which you pick up the top, bottom, left, and right side, respectively. (Hence, `T` represents the fold that places the top side over the bottom side.) Note that for $n = 0$ the second line is empty.

The last line describes the cut. To make your life easier, the cut will never pass through a corner of the square. As it can be shown that the number of pieces does not depend on the exact points where the cut intersects the sides of the square, we can specify the cut simply by giving the labels of the two sides it intersects. For example, `TR` is a cut that intersects the top and the right side of the folded paper.

In the **easy subproblem B1** each cut can be made horizontally or vertically. That is, the two letters that describe each cut are either `T`+`B` or `L`+`R`.

Output specification

For each test case, output a single line containing x modulo $10^9 + 7$, where x is the number of paper pieces we're left with after cutting the paper.

Example

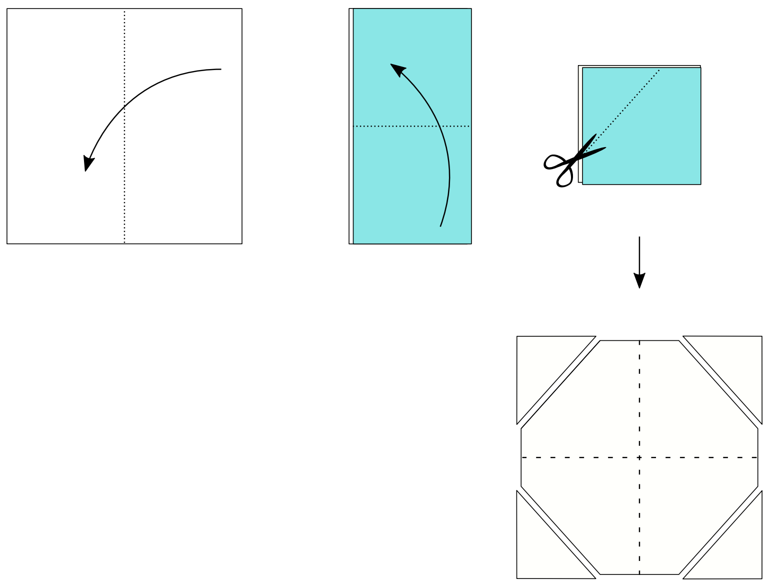
input

```
2

2
LB
TB

2
RB
LT
```

output

```
3
5
```

The second test case is shown in the figure below. This test case will not appear in subproblem B1.



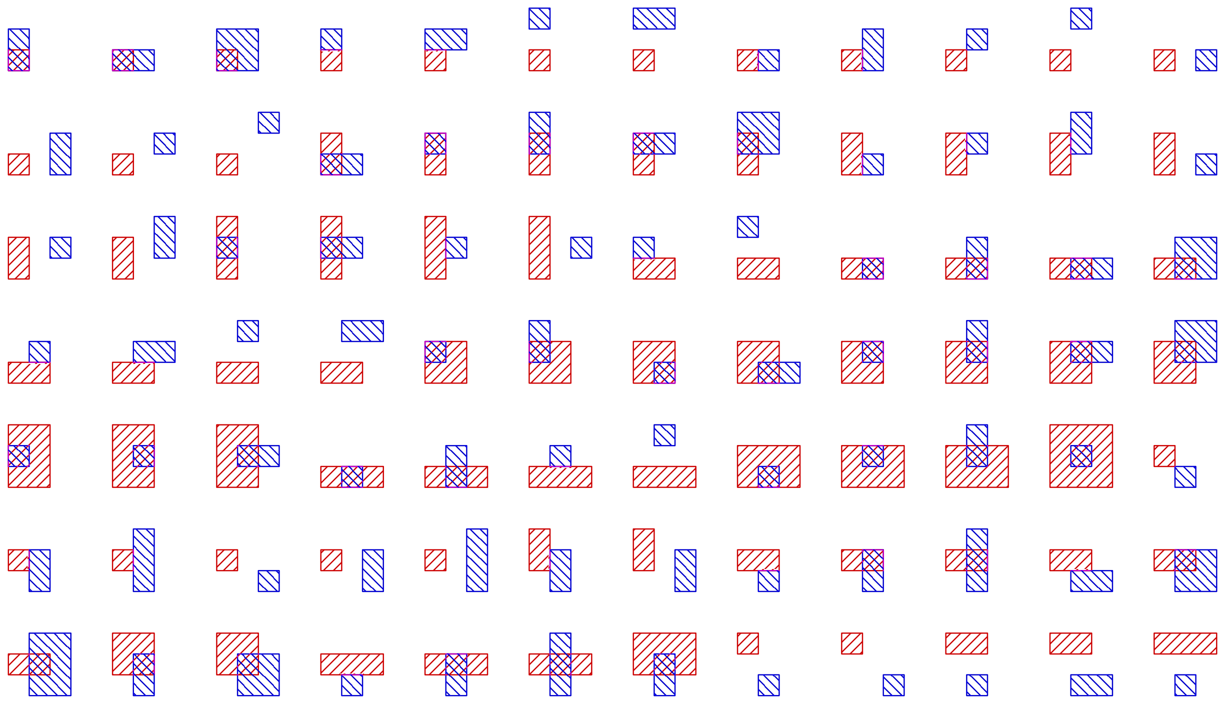
Problem C: Counting rectangles

This entire problem takes place in a standard two-dimensional plane.

Your task is very simple: count all *fundamentally different* sets of n axes-parallel rectangles.

Informally, two sets are fundamentally different if one cannot be transformed into the other by horizontally and vertically stretching some parts of the plane. A precise formal definition is given below.

For $n = 2$ there are 84 such sets. All of them are shown in the following figure. (The two rectangles are indistinguishable but in our figure we show one in red and the other in blue to make the figures where they overlap easy to read.)



Formal definitions

Given an ordered pair of points (P, Q), their relative position is $rp(P, Q) = (sgn(P.x - Q.x), sgn(P.y - Q.y))$. Hence, there are nine possible relative positions, including "Q is somewhere to the left and down of P", "Q is exactly below P", and "P and Q are the same point".

An axes-parallel rectangle is a rectangle such that each of its sides has a positive length and lies parallel to one of the coordinate axes. For an axes-parallel rectangle R we will use $R[0]$ through $R[3]$ to denote its bottom left, bottom right, top left, and top right corner.

Two sets of axes-parallel rectangles $\mathcal{S}$ and $\mathcal{T}$ are called similar if there is a one-to-one map φ from $\mathcal{S}$ to $\mathcal{T}$ such that

$$\forall S_1, S_2 \in \mathcal{S} : \ \forall i, j \in \{0, 1, 2, 3\} : \ rp(S_1[i], S_2[j]) = rp(\varphi(S_1)[i], \varphi(S_2)[j])$$

In human words, if $\mathcal{S}$ and $\mathcal{T}$ each have n rectangles, we call them similar if we can number the rectangles in each set 1 through n in such a way that the relative positions of all their pairs of corners are the same in both cases.

Problem specification

Given n, find the maximum number of sets of n rectangles such that no two sets are similar.

(Note that the order of rectangles in a set does not matter. Also note that sets cannot contain duplicates, hence the n rectangles in each set must be distinct.)

Input specification

There is no input.

Output specification

Let c_n be the number of fundamentally different sets of n rectangles, and let $m_n = c_n \bmod (10^9 + 7)$.
For the **easy subproblem C1** submit an output file with 4 lines, containing m_1 through m_4.
For the **hard subproblem C2** submit an output file with 5000 lines, containing m_1 through m_{5000}.

Example

output for C1

```
1
84
some_bigger_number
an_even_bigger_number
```

(Un)helpful note

Don't bother looking for this sequence in the Online Encyclopedia of Integer Sequences. It's not there :)

Problem D: Delightful

You have found a weird box in your grandma's basement. A careful inspection revealed that it is an old-school computer. You tried to run some simple programs but they all behaved very weirdly. It took you some time before you were able to pinpoint the reason: the computer *uses ternary logic and arithmetic!*

A *trit* is the ternary equivalent of a bit. Our trits can store the values 0, 1, 2. When doing logical operations with trits we imagine that 0 represents false, 1 represents unknown, and 2 represents true.

The computer has 26 registers, labeled `A` through `Z`. Each register is a **40-trit** unsigned integer variable.

Operators

The computer supports the standard arithmetic operators: `+`, `-`, `*`, `/` (integer division), and `%` (modulo). All computations are done modulo 3^{40}. Division and modulo by zero cause a runtime error.

The computer also supports the following tritwise operators:

- `&` (and) returns false if either operand is false, true if both are true, and unknown otherwise
- `|` (or) returns true if either operand is true, false if both are false, and unknown otherwise
- `^` (xor) is addition (without carry) modulo 3
- `~` (unary not) is subtraction from 2
- `<<` (shift left) and `>>` (shift right) behave as expected, new trits are set to 0

Examples (with leading zeros omitted):

- 210_3 `&` $111_3 = 110_3$
- 210_3 `|` $111_3 = 211_3$
- 210_3 `^` $111_3 = 021_3$
- `~` $210_3 = 222\ldots222012_3$ (i.e., 37 twos followed by 012)
- 210_3 `<<` $2 = 21000_3$
- 210_3 `>>` $2 = 2_3$.

Programs

You suspect that there are some instructions for loops and jumps and such, but so far you weren't able to discover any. Thus, in this problem you will have to do without such fancy tools. A program is therefore a finite sequence of commands. Each command will be executed exactly once, in the given order. Each command has one of three forms:

- `[register] = [operand]`
- `[register] = [operand] [operator-with-two-operands] [operand]`
- `[register] = [operator-with-one-operand] [operand]`

Here, operators are the ones listed above and each `[operand]` is either a register or an integer constant that fits into a register.

Constants can be given either in base-10 or in base-3. Base-10 constants have no prefix, base-3 constants have a prefix "`0t`" (zero, lowercase t). For example, `47` and `0t001202` represent the same number.

At least one space between each two tokens is mandatory.

Example

At the beginning of our program's execution, the register `X` contains the input number and all other registers contain zeros. We want to find the last non-zero digit of `X` (if any) and set it to zero.

One possible program that does this looks as follows:

```
A = X - 1
B = ~ A
C = B ^ X
X = X & C
```

The explanation is left as an exercise for the reader.

Subproblem D1

Write a program that computes *ndp*: the length of the longest non-decreasing prefix of a number. For example, for any number x that has the form $000122012\ldots_3$ we have $ndp(x) = 6$ because the first six trits of x are in sorted order but the first seven aren't.

Formally, let the ternary representation of x be $x_{39}x_{38}\ldots x_0$. We define $ndp(x)$ as the largest i such that $x_{39} \leq x_{38} \leq \cdots \leq x_{39-i+1}$.

At the beginning of your program's execution, the register `X` contains the input number and all other registers contain zeros. At the end of your program's execution the register `Y` must contain the value ndp(`X`), all other registers may contain any value.

Your program must contain **at most 100 commands**.

Subproblem D2

At the beginning of your program's execution, the register `X` contains the input number and all other registers contain zeros. It is guaranteed that `X` is between 0 and 10^9, inclusive.

Write a program that will set `Y` to 1 if `X` is prime, and leave it at 0 if it isn't. (Recall that the numbers 0 and 1 are not prime.)

Your program must contain **at most 5,000 commands**.

Input specification

There is no input.

Output specification

Your output file must contain a program that meets the above specifications and solves the given subproblem.

Note that empty lines and extra whitespace on each line are ignored, feel free to use those to format your program for better human readability. Only non-empty lines are counted towards the limit on the number of instructions. Remember that at there must be at least one space between any two tokens in your program, and at least one newline between any two commands.

Testing

A very basic interpreter is provided on a best effort basis on an external site. The interpreter will start in a state with zeros in all registers, execute up to 5000 commands of the program you provide, and report either an error that occurred, or the number of commands executed and final values in all registers.

Problem E: Encrypted romance

Alice and Bob are two young lovers who frequently send each other hearts and kisses over the Internet. However, they (rightfully) suspect that Eve might be interested in hearing all their private messages. Alice and Bob are both untouched by modern cryptography, so they have designed their own encryption scheme. As you probably already expect, it's not that good.

In order to be "extra secure", Alice and Bob have chosen n different symmetric ciphers. Whenever one of them wants to send a message to the other, he or she splits it into n pieces (called blocks) and encrypts each block using a different cipher.

Of course, they need some secret keys in order to do all that encryption and decryption. On the last day of summer Alice and Bob picked one shared secret key k. The key k was then used to compute the keys k_i used for the individual ciphers.

Eve has intercepted a message Alice sent to Bob. Eve already knows all ciphers they use, and also the way in which they compute the keys k_i from the original key k. The only information she's missing is the key k itself.

Estimate Eve's chances of successfully decrypting at least one block of the message!

Problem specification

The secret key is an integer k between 0 and $p-1$, inclusive, where p is a prime number.

The blocks of each message are numbered 1 through n. The key Alice and Bob use to encrypt and decrypt block i is denoted k_i. The key k_i was computed from k using the following formula:

$$k_i = ((a_i \cdot k + b_i) \bmod p) \bmod m_i$$

Luckily for our two lovers, Eve is also woefully bad at cryptography. When everything she tried failed, she decided that she will simply try finding the right k using brute force.

You are given all parameters of the encryption scheme, **including** the secret key k that Eve does not know. Estimate the number of keys from the key space that allow Eve to decrypt at least one block of the message.

Formally, you are asked for the number of integers $k' \in [0, p)$ that have the following property: For at least one $i \in \{1, 2, \ldots, n\}$ we have $k_i = ((a_i \cdot k' + b_i) \bmod p) \bmod m_i$. (In words, if Eve uses the key k' instead of the correct key k, she will correctly decrypt at least one block of the message.)

Since the number of such keys may be very large, we are only interested in a rough approximation of the correct number. (Refer to the output specification.)

Input specification

The first line of the input file contains an integer t specifying the number of test cases. Each test case is preceded by a blank line.

Each test case begins with a line consisting of three integers p, k and n, where

- p is the prime number specifying the size of the keyspace.
- k is the secret key selected by Alice and Bob.
- n is the number of blocks in a message.

Then n lines follow, each describing the key generation scheme for one block of the plaintext. Each line contains three space-separated integers a_i, b_i, m_i that are used in the formula to compute the key k_i.

In both subproblems we have $1 \leq n \leq 100$ and $0 \leq k < p$.
In both subproblems we also have $\forall i \in \{1, 2, \ldots, n\}$: $1 \leq a_i < p$, $0 \leq b_i < p$, and $1 \leq m_i \leq p$.

In the **easy subproblem E1** we have $t = 400$, $p \leq 10^9$ and $\forall i \in \{1, 2, \ldots, n\}$: $m_i \leq 1000$.
In the **hard subproblem E2** we have $t = 100$ and $p \leq 10^{100}$.

Output specification

For each test case, output a single line with a floating-point number m: the amount of keys k' from the key space that share at least one block encryption key with the actual key k.

Make sure that your output contains at least four **most significant decimal digits** of m (but preferably print more). The value of m may be printed in scientific notation (“`1.2345678E-90`”).

In the **easy subproblem E1** your answer will be accepted if the **relative** error is at most 10^{-1}.
In the **hard subproblem E2** your answer will be accepted if the **relative** error is at most 10^{-2}.

Example

input

```
4

13 10 2
2 0 5
3 1 6

19 18 2
3 4 8
2 2 6

1632899 12351 5
12453 232 789
73829 112 650
11234 893 998
788920 128492 706
123892 99109 551

5228506301 5128935514 5
516969294 4322317375 63664
65456316 100106652 423344
121168466 121978447 442230
616682668 556300002 185418
1631488462 1712350840 628094
```

output

```
4
5
11461
1.42e5
```

In the first sample, we have:

- $k_1 = ((2 \cdot k + 0) \bmod 13) \bmod 5 = 2$
- $k_2 = ((3 \cdot k + 1) \bmod 13) \bmod 6 = 5$.

The set of useful keys is $\{1, 5, 10, 12\}$, as for $k' \in \{1, 5\}$ we correctly decrypt the first block and for $k' = 12$ we correctly decrypt the second block.

In the second sample, the key $k = 18$ leads to $k_1 = 1$ and $k_2 = 0$. There are five good keys: 2, 5, 8, 17, 18. Even though with $k' = 8$ Eve deciphers both blocks correctly, the value 8 is only counted once.

In the third case, any answer in the range $[10315, 12607]$ would be accepted for the easy subproblem. For the hard subproblem, only values within the range $[11347, 11575]$ would be considered correct.

The last sample cannot appear in the easy subproblem, as both $p > 10^9$ and $m_i > 1000$. The correct answer is 142816.

Problem F: Farmer's code

Bob is a farmer. Among other endeavours, he grows apple trees. Not all of his trees are perfect, though. Some produce sour fruit, some are too small, and some are just crooked. His dream is a perfectly symmetrical tree with extremely sweet apples. However, he isn't very fluent in genetics, so he needs your help!

Problem specification

In this problem, you will interact with an online farm dashboard page. You can find the link in the online version of this problem statement.

You have a population of n trees, sorted from oldest to youngest. In the beginning all trees have distinct genotypes (their genetic code) and phenotypes (their appearance) but this may change based on your actions. You are also given an image of your goal – the tree that Bob wants to grow.

You may select any two living trees to cross them. When you do this, the following happens:

- The genetic code of these two trees is crossed and a new tree grows. The new tree is considered the youngest.
- The oldest tree of the whole population dies. As there is one addition and one removal, the size of the population stays the same.
- If you create the tree that is your goal, you will win and the subproblem will be marked as **OK**.

You will see the same set of trees if you connect from two different browser windows.

You can use the *Reset* button if you think that your genetic diversity became insufficient to produce the requested phenotype, or you simply want to start over. The farm will go back to the initial population and you'll receive a **Wrong answer**. You can reset as often as you want – you are not limited to 10 submissions per subproblem.

However, you can only cross trees a limited number of times. The limit is separate for each subproblem and is displayed on the farm page. The number **will not reset** when you click *Reset*. If you run out, you won't be able to solve that subproblem.

Input specification

In the **easy subproblem F1** we have $n = 4$.
In the **hard subproblem F2** we have $n = 18$.

Output specification

You don't have to submit anything. You will automatically get **OK** or **Wrong answer** submissions by interacting with the farm page.

Word of advice

Knowledge of real-life tree genetics probably *won't* help you solve this problem.

Problem G: Git gud

We did it! We finally created a fully-functional time machine. It can only send information back and forth, but that's more than enough if you know how to take advantage of it.

Obviously, the first thing we did was to establish communications with ourselves 50 years in the future, so that future-us can send present-us a copy of all the IPSC problems they made and we won't have to do any more work. They immediately agreed and sent us 50 years' worth of problems. And what's even better, the data was not in some crazy future file format that hasn't been invented yet. It was just a Git repository.

But when we tried to open it and find out what's inside, disaster struck. Our computer crashed and the time machine got disconnected. Please help us!

Problem specification

You are given a valid Git repository. Print a checksum of the list of files it contains.

Input specification

The input is a Git repository. Every file and directory in it has a name consisting only of lowercase English letters, numbers, dashes ("`-`") and periods ("`.`"). The repository can be downloaded with this command:

```
git clone https://ipsc.ksp.sk/2018/real/problems/g.git
```

The repository has three branches: `example`, `easy` and `hard`. When you clone it, Git will rename them to `origin/example`, `origin/easy` and `origin/hard` in your copy.

Instead of getting the repository from `ipsc.ksp.sk`, you can use `g_local.py` from the downloadable archive. This script will start a local server containing the same repository. Use it as follows:

```
python3 g_local.py                      # (in one window)
git clone http://localhost:8000/g.git   # (in another window while g_local.py is running)
```

Output specification

Submit a file with a single number: the checksum of the file list, as defined below.

The file list contains one line for each file in the repository. Each file is printed with its full path, using "`/`" (ASCII code 47) as the directory separator. The lines of the file list are sorted lexicographically and each line is followed by a UNIX newline (ASCII code 10). There is no extra whitespace.

The checksum is produced as follows. Interpret the file list as a sequence of ASCII codes. Multiply each of these ASCII codes by its 1-based position in the sequence. The checksum is their sum modulo $10^9 + 7$. For example, the checksum of "`Cat`⟨*newline*⟩" is $(1 \cdot 67 + 2 \cdot 97 + 3 \cdot 116 + 4 \cdot 10) \bmod (10^9 + 7) = 646$.

Example

sorted file list for the "example" branch

```
12/r
12/r.h
3
ll
lll/v.txt
lll/v/-
lll/v0
llll
```

output

```
87437
```

Problem H: Hats of various colors

You and your fellow revolutionaries have been captured by the evil archduke, and now you're all waiting for execution in the archduke's dungeons. Fortunately, a thousand year old tradition states that all prisoners must be given a last chance to go free if they can solve a logic puzzle.

The warden explains what will happen:

- Every prisoner will receive a red or blue hat. The prisoners will be seated so that they can see everyone else's hat, but not their own hat. The prisoners must stay completely silent.
- When the warden shouts "Now!", all prisoners must immediately and simultaneously choose whether to raise their hand or not. Everyone will see everyone else, but they must still stay silent.
- The warden will (privately) ask every prisoner what is the color of their own hat. If all prisoners answer correctly, the archduke has to let them go. If anyone makes a mistake, they will all be executed.

In both subproblems, there are $N = 13$ prisoners. In the **easy subproblem H1**, the hats are of $C = 2$ different colors (red and blue). In the **hard subproblem H2**, there are $C = 4000$ available hat colors.

Problem specification

Write a program that describes the prisoners' strategy.

This problem is special. Usually, you can write a program in any programming language, and you only submit the computed output data, not your source code. This task is the exact opposite. You only submit source code – specifically, a program written in the Lua 5.3 language. (See below for an introduction.)

We will run N copies of your program and allow them to communicate according to the rules. If all N programs answer correctly, you win.

Your program can use these variables:

- `N`: the number of prisoners.
- `C`: the number of different colors.
- `myself`: your own ID – a number between 1 and N, inclusive.
- `colors`: an array of visible hat colors. For every i between 1 and N, `colors[i]` will be a number between 1 and C, inclusive, except for `colors[myself]` which will be `nil`.
- `raise`: a function which takes a single boolean argument (whether you raise your hand or not), and returns an array of N booleans (for each prisoner, whether they raised their hand). You must call `raise` exactly once.
- `answer`: initially unset. You must set it to a number between 1 and C, inclusive – the number you want to announce as the color of your hat.

Limits

These functions won't exist: `debug.debug`, `io.*`, `loadfile`, `math.random`, `math.randomseed`, `os.*`, `package.*`, `print`, `require`.

We will run 300 test cases. (In total, your program will run $300 \times N$ times.) The memory limit is 256 MB (for the sum of your N programs). The time limit is 10 seconds (for the sum of your $300 \times N$ runs).

Testing your program

If you'd like, you can test your solution with our testing program: `lua h-test.lua solution.lua` .

Introduction to Lua

If you don't know the Lua language, here's a quick introduction:

- Variables: `foo_bar = a + b`
- Literals: `nil`, `true`, `false`, `123`, `0xFF`, `"a string"`
- Arithmetic: `1+2 == 3`, `1-2 == -1`, `2*3 == 6`, `3/2 == 1.5`, `3//2 == 1`, `7%5 == 2`, `2^3 == 8`
- Comparison: `a == b`, `a ~= b`, `a < b`, `a <= b`, `a > b`, `a >= b`
- Logic: `a and b`, `a or b`, `not a` (note that `nil` and `false` count as false, and everything else counts as true, including `0` and `""`)
- Math: `math.abs(x)`, `math.floor(x)`, `math.max(a, b, c, d)`, `math.pi`, etc.
- Binary: `a & b` (AND), `a | b` (OR), `a ~ b` (XOR), `a >> b` (right shift), `a << b` (left shift), `~a` (NOT)
- "If" statement: `if ??? then ... elseif ??? then ... else ... end`
- "While" loop: `while ??? do ... end`
- Numeric "for" loop: `for i = 1, 128 do ... end`
- Functions: `function some_name(a, b, c) local d = 7; return a + d; end`
- Local variables (in functions and control structures): `local l = 123`
- Arrays (tables): `arr[idx]` (indexed from 1), `{ 1, 2, 3 }` (new array), `#arr` (array length)
- Comments: `--[[ multiple lines ]]`, `-- until end of line`
- Newlines *and* semicolons are optional: `a = 1 b = 2 + 3 c = 4 * 5`

For more details, refer to the Lua manual and the language grammar, or other on-line resources – though you probably won't need most parts.

Example

```
my_bool = colors[1] == 1 or myself > 6

hands = raise(my_bool)

if hands[1] or not hands[2] then
  answer = 1
else
  answer = C
end
```

Problem I: Incorrect expression

Number Man is one of the world's most powerful superheroes, thanks to his superhuman mathematical powers. With a single thought he can forecast market movements, factor large numbers, extrapolate object trajectories, compute discrete logarithms, estimate probabilities of catastrophic events, and even calculate his tax rate. His favorite hobbies are knitting and high-frequency trading.

But despite his amazing powers, Number Man is very frustrated right now. His ultra-accurate weather prediction formula started giving ultra-inaccurate results, and he spent the last few hours searching for the cause. Did he really make a mistake? He never makes mistakes! And why does everyone else keep giggling? His teammates probably pranked him and made a tiny change on the blackboard while he was distracted. But what symbol did they change? There are so many options.

Problem specification

First, let's recursively define what is a *valid expression*:

- A sequence of one or more digits is a valid expression. Leading zeros are allowed.
- If `A` is a valid expression, then `(A)` is a valid expression.
- If `A` and `B` are valid expressions, then `A+B`, `A-B` and `A*B` are valid expressions.
- Nothing else is a valid expression. No division, no unary minus, etc.

Number Man has a valid expression E. Every valid expression that has the exact same length as E and differs from E in exactly one character is a *potentially correct expression.* Note that E itself is **not** a potentially correct expression.

To calculate the *weather value* of a valid expression, calculate the expression's value normally (standard precedence rules apply) and then find the nonnegative remainder that value gives modulo $10^9 + 7$. E.g. the weather value of "`0-2*1`" is $10^9 + 5$.

Find the weather value of every potentially correct expression. How many different numbers did you get?

Input specification

The first line of the input file contains an integer t specifying the number of test cases. Each test case is preceded by a blank line and consists of a single line containing the *incorrect but valid expression* E.

In the **easy subproblem I1**, the length of each expression is at most 10 000.

In the **hard subproblem I2**, the length of each expression is at most 1 000 000.

Output specification

For each test case, print a single number: the number of distinct weather values among all potentially correct expressions.

Example

input

```
2

0+0

5*6*7
```

output

```
19
49
```

Problem J: Jumping over walls

You have always been an avid fan of mazes, and a bit of a cheater. No wonder you came up with a peculiar device that can be used to cheat in a maze. You will simply select a direction and the device will help you jump over all walls in your way, all the way to the nearest empty space in the chosen direction.

You have built two prototypes of the device. Prototype J1 only allows you to choose the 4 cardinal directions for jumping. Prototype J2 also lets you jump diagonally, so you have 8 available directions for each jump.

You would like to start manufacturing more of these devices – many maze fans would love to have one! But before you sell them, you need to test each of them to make sure you don't damage your brand by selling defective pieces. You decided that you'll build a test maze for those tests.

Problem specification

You will be constructing two separate test mazes: one for the J1 device and one for the J2 device. The purpose of the test maze is to make sure that the device jumps correctly in all supported directions, so it must be impossible to solve the maze without using every direction at least once. Constructing mazes costs money, so you must make them as small as possible.

While in the maze, you will only move using the device, never by walking. But note that the device is also able to jump over zero walls – if the very next cell in that direction is empty, you will just jump to that cell.

Each test maze must have the following properties:

- The maze is on a rectangular grid. The area of the rectangle must be as small as possible.
- Some cells are walls (denoted `#`), others are empty (denoted `.`).
- There is exactly one start cell (denoted `A`) and exactly one target cell (denoted `B`). Both are considered empty. They may be located anywhere in the maze.
- We assume that the maze is surrounded by an infinite grid of walls and jumping outside is impossible.
- It must be possible to travel from `A` to `B`.
- It must be impossible to travel from `A` to `B` without using each of the available directions at least once.

The **easy subproblem J1** is to construct a testing maze for the J1 prototype (4 directions), the **hard subproblem J2** is to do the same for the J2 prototype (8 directions).

Input specification

There is no input.

Output specification

The first line of your output should contain two integers: the number of rows r and the number of columns c in your maze. The value $r \times c$ must be as small as possible.

The rest of your output should contain the map of the maze: r lines, each with c characters as specified above.

Example

output

```
5 6
......
.#.#.B
###...
#A#...
#.#...
```

This is a syntactically correct output but it doesn't describe a good test maze. This is because you could solve this maze without ever making a jump down. (For example, you can make three jumps right followed by two jumps up.)

Note that if you had the J2 prototype and started this maze at `A`, you would have exactly four possibilities for your first jump:

- Select the direction "down" and jump by 1.
- Select the direction "up" and jump by 3.
- Select the direction "right" and jump by 2.
- Select the direction "diagonally up and right" and jump by 3.

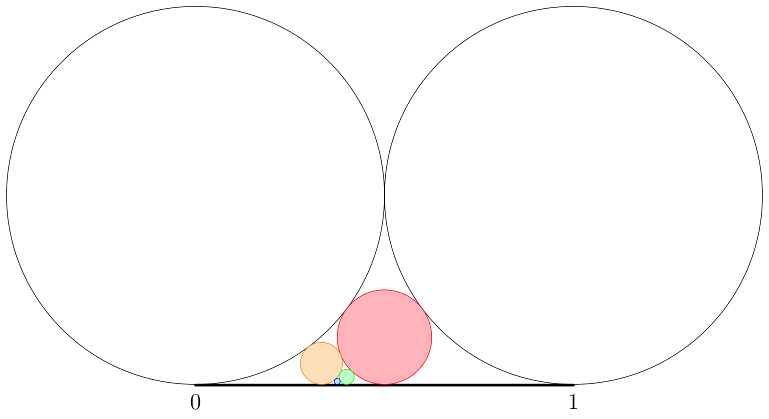
Note that you cannot choose a jump direction that doesn't lead to any empty cells.

Problem K: Kids draw the darndest things

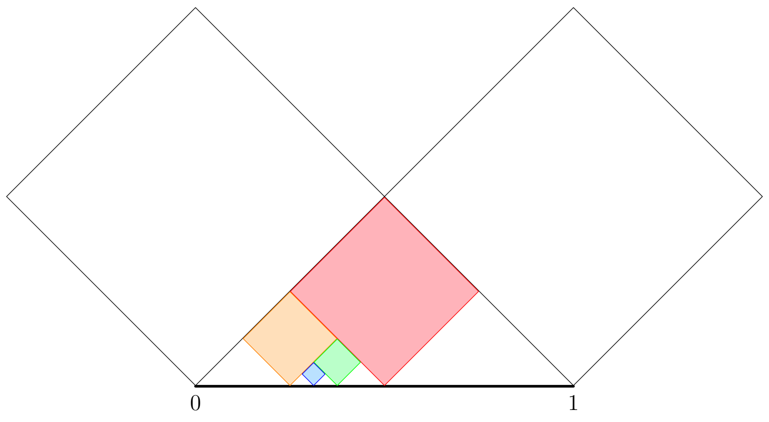
Kelly is drawing pictures with circles. In order to draw a picture, she does the following:

1. She draws a horizontal line segment of length 1 and she assigns the coordinates 0 and 1 to its left and right end, respectively.
2. She draws two equally big circles that touch the line of the segment from above (one at each end of the segment) and touch each other above the middle of the segment. These two circles become her active circles.
3. She draws a third circle that touches the line segment and both active circles.
4. She chooses a sequence of $n \geq 0$ directions, each being either "left" or "right".
5. Exactly n times she repeats the following:
    1. She looks at the next direction. Only one of the two active circles remains active: the one that touches the segment farther in that direction.
    2. The most recently drawn circle becomes active.
    3. She draws a new circle that touches the segment and both active circles.

For example, this is the picture she would draw for the sequence "left, right, left". Circles were drawn in the order red, orange, green, blue. (The blue one is quite tiny, don't miss it!)



As boys often do, Kevin forgot his compass at home. But he also likes pretty pictures, so he decided that he will do the same with squares (rotated 45 degrees, so that their diagonals are parallel to the coordinate axes). For example, for the sequence "left, right, left" Kevin would draw this:



Problem specification

The "circles-to-squares" function c is defined as the only continuous function on $[0, 1]$ with the following property: For any sequence of directions, if Kelly drew her picture and the last circle touches the segment at x_c, and Kevin drew his picture and the last square touches the segment at x_s, then let $c(x_c) = x_s$.

You will be given a collection of inputs for c. Compute the corresponding outputs.

Input specification

The first line of the input file contains an integer $t \leq 100$ specifying the number of test cases. Each test case is preceded by a blank line and consists of a single line.

Each test specifies the number x_c for which you must compute $c(x_c)$. Decimal numbers are not exact, so the inputs are specified as roots of polynomials. Each test case has the form "p k", where p is a polynomial with integer coefficients (see below) and k is a positive integer. Find the k-th smallest distinct real root of p and use that value as x_c. It is guaranteed that it exists and lies in $[0, 1]$.

In the **easy subproblem K1** each p is linear. (Therefore, every x_c is a rational number.)

In the **hard subproblem K2** each p is either linear or quadratic.

Additionally, the input will be such that each line of correct output will have fewer than 40,000 characters.

Output specification

For each test case, output a single line. If the corresponding $x_s = c(x_c)$ is an algebraic number (i.e., it can be expressed as a root of a polynomial with integer coefficients), output it using the same formatting as in the input. You must output the polynomial with the *smallest possible positive degree*. If the corresponding x_s is transcendental, output the line "`transcendental`" instead.

Polynomial formatting for input and output

- The description of a polynomial does not contain any spaces.
- The powers of x in a polynomial are printed in descending order.
- Every term is written as `a*x^n` where `a` and `n` are integers, except that `a*x^1` is written as just `a*x`, and `a*x^0` as just `a`.
- All powers of x are explicitly mentioned, even if their coefficient is zero. (E.g., `2*x^2+3` is invalid.)
- The leading term has a positive coefficient. (E.g., `0*x^2+1*x+1` and `-1*x+2` are invalid.)
- All coefficients are always explicitly mentioned, even if they are 0 or ± 1. (E.g., `x^3-x^2+x+1` is invalid.)
- There is no $d > 1$ that divides all coefficients of the polynomial. (E.g., `2*x+2` is invalid.)

Examples of correctly formatted polynomials include "`1*x^3+0*x^2+0*x-0`" and "`7*x^2-47*x+123456`". For more examples see the input files directly.

Example

input

```
2

2*x-1 1

3*x-1 1
```

output

```
2*x-1 1
4*x-1 1
```

Problem L: Lethargic foe

Alice entered a local chess tournament where she will play Bob a total of seven times, wielding white pieces every time. Bob is not a very good chess player and he is notorious for using a lazy strategy. He always mirrors his opponent's moves. For example, when Alice starts by moving her pawn `e2` to `e4` (denoted `1. e4` in Standard Algebraic Notation), Bob will respond by moving his pawn from `e7` to `e5` (`1. ... e5`). When she follows with a queen move `2. Qh5`, he will counter with `2. ... Qh4`. Of course, such a mirroring move is not always possible. For example, if after the said two moves Alice captures the queen by playing `3. Qxh4`, Bob can no longer reply the same way, as he has no queen left. Similarly, if Alice played `3. Qxf7+`, Bob would be in check and the move `3. ... Qxf2+` would be illegal. When such a situation occurs, Bob will just stare at the chessboard until his time runs out.

Problem specification

Alice considers games with Bob a waste of time, so she wants to win them as fast as possible. This means that the number of moves has to be as small as possible, and Bob has to be able to mirror her every move until the move that delivers the checkmate.

However, Alice wants to challenge herself at least a little bit. She wants to play seven different games, and in each of them use a different piece in the move that delivers the checkmate. That is, there has to be a game where the last move is performed by a pawn, a knight, a bishop, a rook, a queen, a king, and any promoted piece, respectively. Your task is to suggest such games to Alice.

Note that the piece considered to deliver the mate is the one that moves in Alice's last turn, even if it's just moving out of the way (a discovered attack). Delivering the mate with a promoted piece counts as a promoted piece victory regardless of whether you promoted it to a queen, a rook, a bishop or a knight. When the promotion itself is a checkmate, it is considered to be a promoted piece move and not a pawn move. Castling is considered to be a king move, not a rook move.

Output specification

Submit a valid file in Portable Game Notation (PGN) containing at most 7 games. We will ignore all comments, tags, and variations (but they still have to be syntactically correct if present) and only consider the main line. So the simplest way is to submit a file containing **only the `movetext` for each game**, separated by blank lines.

Each game has to be valid and follow the above rules. It has to be legal, end with the black king in a checkmate, and every black move has to mirror the immediately previous white move.

For each piece type you can score a point. In order to determine whether you do, we count the number of moves in your shortest game (defaulting to 100 if no game of that type has been submitted) and then we do the same for our best solution. To score a point, your solution has to use at most as many moves as ours.

In the **easy subproblem L1** you need to score at least 3 points.

In the **hard subproblem L2** you need the full 7 points.

Example

Below is an example of a file you might submit. The file contains four consecutive games in PGN, separated by a blank line.

```
1. f3 e5
2. g4?? Qh4#

1. e4 e5 2. Bc4 Bc5 3. Qh5 Nf6 4. Qxf7#

1.c3 c6 2.Qc2 Qc7 3.Qxh7 Qxh2 4.g3 g6 5.Bg2 Bg7 6.Qxg7 Qxg2 7.Qxh8 Qxh1 8.Qxg8#

[FEN "8/3RP2k/6pp/3N4/3n4/8/3rp2K/8 w - - 0 1"]
[SetUp "1"]
1.e8=Q#
```

This submit would get a Wrong answer for the following reasons:

- The first example game is incorrect because it ends with white king in checkmate.
- The second example game is incorrect because the black's third move is not a mirror image of the third white's move.
- The third game is correct. However, you will not get a point because there is a better solution for the queen giving the checkmate.
- The fourth game is incorrect as you didn't use the standard starting position. Note that the mating move would be only considered a checkmate by a promoted piece for the purposes of this task, and not pawn, queen or rook.

Problem M: McDroid's

The trend is clear: the number of robots in the world is growing and growing. You decided to become an entrepreneur and open the first robot restaurant in the world. Not a restaurant staffed by robots – a restaurant *for* robots. After a long preparation, everything is finally ready for the grand opening.

You have a long menu with many different kinds of delicious robot food. Most humans would probably just be confused because robot recipes don't have names, only numbers. And of course, robots only use binary. You're sure the restaurant will become very popular and it will attract a big crowd. As the owner, your biggest challenge is managing your time and making sure you don't leave anybody (any*boty*?) waiting. Your second biggest challenge is that it's not always easy to understand what food the robots want.

Problem specification

In this problem, you will interact with an online dashboard page. You can find the link in the online version of this problem statement.

The page has a button to open the restaurant. When you press the button, your restaurant will open and it will start getting robot customers. Your team has just one restaurant. You will see the same restaurant if you access the dashboard from multiple browsers. Note that time continues to run even if you close the page.

Every customer will tell you what they want, and you must choose what food to give them. Some robots might just tell you "`HELLO I WOULD LIKE TO ORDER FOOD ITEM 1101.`", in which case you'd obviously give them `1101`. For other customers figuring out what they want may be more complicated. You'll see.

Robots will arrive in real time. Each robot is only willing to wait for some predetermined period of time (at least a minute, sometimes more). You may have multiple robots waiting in your restaurant at the same time. For each of them you will see the time left to serve them.

When you first open your restaurant, you'll have 10 reputation points.

There are two things that can go wrong at your restaurant:

- If you **serve incorrect food** to a customer, you lose one reputation point and **that customer** leaves.
- If you **fail to serve** a customer in time, you lose one reputation point and **everybody who was waiting** gives up and leaves.

Initially, the rules for solving **subproblems M1 and M2** are as follows:

- If the total time your restaurant was open reaches t_1 = **60 minutes**, you will solve subproblem M1.
- If the total time your restaurant was open reaches t_2 = **180 minutes**, you will solve subproblem M2.

Whenever your reputation reaches zero, you go bankrupt. You can then reopen the restaurant and continue trying to solve this problem. Whenever you reopen your restaurant, the following things will happen:

- You are given another 5 reputation points.
- If you didn't solve **subproblem M1** yet, the threshold t_1 is increased by 20 minutes.
- If you didn't solve **subproblem M2** yet, the threshold t_2 is increased by 20 minutes.
- As an additional penalty, your restaurant now also attracts human customers and you need to serve those as well. Humans are usually much more annoying than robots.

Notes

- The problem uses normal scoring rules. The sooner you begin, the smaller will be your time penalty when you solve a subproblem.

- Please be merciful on our servers. The page auto-updates itself, you don't have to press F5 every millisecond. If you do, the server might temporarily block you and you might fail to serve some customers.
- As always, please contact us if you encounter any technical issues.
- One final word of caution: You should expect that over time the restaurant will become more popular and you *might (wink, wink)* start getting more picky customers.

Input and output specification

There is no input and no output. You will automatically get an **OK** verdict for a subproblem the moment the total time your restaurant was open reaches the corresponding threshold.