Problem A: Advice for Olivia

Olivia is going to work in the candy shop during the summer. However, she is afraid she'll have to work at the cash register. Whenever the cash register tells her to return some money to the customer, Olivia panics, because she can't decide which denominations to use. And if she takes too long (i.e., uses more than p pieces), a long checkout queue of nervous people will soon form.

Problem specification

In this problem we shall consider the Euro, as this is the currency used where Olivia lives.

The cash register holds an *infinite supply* of each of the following denominations: 1c, 2c, 5c, 10c, 20c, 50c, 1 Euro, 2 Euro, 5 Euro, 10 Euro, 20 Euro, 50 Euro, and 100 Euro. (The "c" denotes cents. There are 100 cents in 1 Euro.)

For the given sum s, find one way of paying s using at most p pieces of currency.

In the easy subproblem A1, p equals 10^6.
In the hard subproblem A2, p equals 200. (Using fewer pieces is harder!)
In both subproblems, each s will be between 0 and 100 Euro, inclusive.

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 consists of a single line containing two space-separated integers: e and c. The sum s for this test case is "e Euro c cents". You may assume that $0 \leq c \leq 99$.

Output specification

For each test case, output one line containing 13 space-separated integers n_1 through n_{13}, where each n_i represents the number of pieces of the i-th currency type in the list above. That is, n_1 is the number of 1c coins, ..., and n_{13} is the number of 100 Euro banknotes Olivia should use.

The total number of pieces of currency you use ($n_1 + \cdots + n_{13}$) must be less than or equal to p. Any such output that pays exactly the desired sum s will be considered correct.

Example

input

```
3

10 50

0 1

18 47
```

output

```
0 0 0 0 0 1 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 0 2 0 1 1 1 1 0 0 0
```

In the first test case, Olivia pays "10 Euro 50 cents" using a 10-Euro banknote and a 50-cent coin. In the second test case, she pays a single cent using a 1-cent coin. In the last test case, she pays the sum "18 Euro 47 cents" as 10 Euro + 5 Euro + 2 Euro + 1 Euro + 20 cents + 20 cents + 5 cents + 2 cents.

Note: Please do NOT submit any programs.
For each subproblem, just produce and submit a correct output file.

Problem B: Boredom buster

Gillian is normally a very lively child. Most of the time she plays with her friends and tries to indulge in some mischief. But today is different, today Gillian woke up with the flu so she has to stay in bed – still and bored. To entertain her, her brother came up with the following game.

When Gillian has an integer x greater than 1, she can split it up into two positive integers y and z such that $x = y + z$. After performing this operation, her brother gives her $y \cdot z$ hazelnuts. However, not all pairs of y, z are valid – there are some rules Gillian must comply with. These rules differ between the easy and hard subproblems; they are listed in the problem specification section.

Numbers that are obtained as a result of this operation can be also split up. At the beginning of the game, Gillian starts with a single integer n. She performs a series of operations described above until she is left with n copies of number 1. What is the maximum number of hazelnuts she can win if she chooses her moves wisely?

Problem specification – easy subproblem B1

For any $x > 1$ there is exactly one valid way of splitting:

- if x is divisible by 3, then $y = x/3$ and $z = 2x/3$;
- if x is divisible by 2 (but not by 3), then $y = z = x/2$;
- otherwise, $y = 1$ and $z = x - 1$.

Problem specification – hard subproblem B2

Gillian can pick any integer k satisfying $1 < k \leq x$ and split up number x into $y = \lfloor x/k \rfloor$ and $z = x - \lfloor x/k \rfloor$.

Input specification

The first line of the input file contains an integer $t \leq 1000$ specifying the number of test cases. Each test case is preceded by a blank line.

Each test case contains a single line with Gillian's initial integer $n > 1$. You may assume that $n \leq 10^6$ in the easy subproblem B1, and $n \leq 10^9$ in the hard subproblem B2.

Output specification

For each test case, output a single line with the maximum number of hazelnuts.

Example

input	output
1 5	10

This answer happens to be correct for both subproblems.

Problem C: Code Inception

In each subproblem of this task you are given a piece of code. Ultimately, the code produces (somehow, somewhen) a single readable English word.

Problem specification

Your only goal: recover the word.
And remember: you may need to go deeper.

Input specification

For each subproblem, you are given two files, each containing the same program, written in a different language. (One is in C++, the other in Python3. We did our best to make the programs as similar to each other as possible.) Each program produces the same single English word.

Difficulty is subjective. You may find subproblem C2 easier to solve than C1. But solving C1 is still worth 1 point and solving C2 is worth 2, just as in the other tasks.

Output specification

Submit a text file containing the recovered word, in UPPERCASE.

Example

input

```
for x in "olleh"[::-1]:
    print(x)

------------------------------

#include <iostream>
#include <string>
std::string s = "olleh";
int main() {
    for (auto c=s.rbegin();
         c!=s.rend();
         ++c)
            std::cout << *c;
}
```

output

```
HELLO
```

Note that the answer is given in uppercase.

Problem D: Do the grading

Did you take part in the practice session? If you did, you will be able to read this problem statement faster. But if you missed the practice session, don't worry, we will tell you all you need to know.

One of the practice tasks was the task *Rearranged alphabet*. In this task we asked the contestants to find a short string of lowercase letters such that each of the 26! permutations of `a` through `z` occurs in it as a (not necessarily contiguous) subsequence. For example, if the alphabet only consisted of `a`, `b`, and `c`, `abcabac` would be a valid answer, but `abccba` would not (both `bac` and `cab` is missing).

Solving the practice task was simple enough: you just have to find a pattern and print the corresponding string. On the other hand, *grading* the practice task is much more complicated: you have to read a string and check whether it actually contains all possible permutations.

Preparing the grader for the practice problem was quite fun. In fact, it was so much fun that we wanted to share it with you.

Problem specification

You are given a string of lowercase English letters.

In the easy subproblem D1, check whether each of 26! permutations of `a` through `z` occurs in the string as a subsequence.

In the hard subproblem D2, count the number of permutations of `a` through `z` that **do not** occur in the given string as a subsequence. As this number may be quite big, output it modulo 65 521.

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 consists of a single line containing a nonempty string of lowercase English letters.

In the easy subproblem D1, $t \leq 150$ and none of the strings is longer than 2 500 characters.

In the hard subproblem D2, $t \leq 20$ and the sum of lengths of all strings does not exceed 1 000.

Output specification

For each test case, output a single line with the answer.

That is, in the easy subproblem D1, output "`YES`" (without quotes) if the given string contains all permutations, or "`NO`" if it doesn't.

In the hard subproblem D2, output the number of missing permutations modulo 65 521.

Example

input	output for subproblem D1

`1`
 	
`abc`
	`NO`

	output for subproblem D2
	`8297`

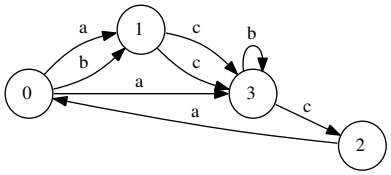
In D2, note that 26! mod 65521 = 8297.

Problem E: Exploring the cave

After the success of "open sesame!", Ali Baba experimented with various other crops. Most of them didn't do anything out of the ordinary, until suddenly "open sugarcane!" caused one of the rocks to shift and reveal the entrance to a peculiar cave.

The cave consisted of several chambers. The entrance lead directly into one of these chambers, we will call it the starting chamber. Some pairs of chambers were connected by *one-way* tunnels. Each of the tunnels was of one of three types: some tunnels had abrasive walls, others had battered walls, and the rest had calcified walls. As you have probably already guessed, we will denote the tunnel types `a`, `b`, and `c`.

For any chamber, there could have been arbitrarily many tunnels entering it, and arbitrarily many tunnels leaving it – including multiple tunnels of the same type, or no tunnels at all. Also, there could have been tunnels that start and end in the same chamber.



An example of a cave with 4 chambers and 8 tunnels.

Of course, it's not really a good idea to explore a cave with one-way tunnels on your own. Luckily, Ali Baba can enlist the help of the forty thieves (and their infinitely many friends, if necessary). One round of cave exploration looks as follows:

1. Ali Baba chooses a finite (possibly empty) sequence of tunnel types (a string of letters).
2. One after another, the thieves repeat the following procedure:
   (a) The thief takes a long piece of rope and fastens one of its ends to his waist.
   (b) He enters the starting chamber.
   (c) He tries to follow a sequence of tunnels that 1. corresponds exactly to the sequence of types selected by Ali Baba, and 2. has not been traveled (as a whole) by any of the previous thieves.
   (d) If successful, the thief remains waiting in the final chamber reached by his walk. (We assume that each chamber is large enough to accommodate all the thieves that end their walks there.)
3. As soon as a thief is unable to perform his task (each possible sequence of tunnels has already been traversed by someone), the exploration round stops. The last, unsuccessful thief is removed from the cave – Ali Baba uses the thief's rope to pull him out.

   At this moment, consider the set of chambers that contain at least one thief. The set of chambers will be called *significant*. (Note that sometimes the significant set may even be empty.)
4. Ali Baba uses the ropes to pull all the thieves out of the cave.

Of course, different choices of the sequence in step 1 can lead to different significant sets of chambers in step 3. Consider the example above. If Ali Baba chooses the sequence `ac`, he will discover the significant set $\{2, 3\}$: there will be one thief going $0 \to 3 \to 2$ and two other thieves going $0 \to 1 \to 3$ (each of these two using a different tunnel to get from 1 to 3). The sequence `bcb` produces the significant set $\{3\}$, the empty sequence produces the significant set $\{0\}$, and the sequence `ccc` produces an empty significant set.

Problem specification

You are given the total number n of chambers in the cave. Ali Baba has also told you that they tried to explore the cave using all possible sequences of tunnel types (even though there's infinitely many of them!) and that they were able to find exactly d different significant sets of chambers.

Find whether such a cave system exists. If yes, find one example.

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 consists of a single line with the two numbers: n and d.

In the easy subproblem E1, $1 \leq n \leq 10$ and $1 \leq d \leq 100$.

In the hard subproblem E2, $1 \leq n \leq 22$ and $1 \leq d \leq 10^9$.

Output specification

For each test case, there are two possible outputs.

If there is a cave with the given parameters n and d, output the description of one cave as a sequence of tunnels. In the first line, output the number $m \leq 5000$ of tunnels in your cave. (If there is a valid cave, there is always one with much less than 5000 tunnels.) In each of the following m lines, output one tunnel in the form "x y z", where x is the chamber where the tunnel starts, y is the chamber where it ends, and z is one of `a`, `b`, and `c`. (The chambers in the cave are numbered 0 through $n-1$, where chamber 0 is the starting chamber.)

If there is no such cave, output a single line with the integer -1 instead.

You may output additional whitespace. (Note that we do so in the example output for clarity.)

Example

input

```
3

2 3

4 7

1 100
```

output

```
1
0 1 a

8
0 1 a
0 1 b
0 3 a
1 3 c
1 3 c
3 3 b
3 2 c
2 0 a

-1
```

In the first test case the cave we produced has three significant sets of chambers: $\emptyset$, $\{0\}$, and $\{1\}$.
In the second test case our answer is the cave shown on the previous page.
In the third test case it is obvious that there is no such cave.

Problem F: Feeling lucky?

Last year the IPSC was so successful that we earned n coins. And they are no ordinary coins: they are perfectly identical coins made of solid gold.

Sadly, there are some problems with our coins. First of all, we don't actually have them. The coins are locked in a vault in Absurdistan. And second, we just got word that one of our coins has been stolen and replaced by a fake one. The fake coin slightly differs from the real ones in weight, but we do not know whether it is heavier or lighter.

So far, our situation looks like one of those weighing puzzles, doesn't it? We bet you would love to take balance scales and start comparing the weights of some coins in order to identify the fake one as quickly as possible.

Well, it kind of does look like a weighing puzzle, but the weighing part is not the major issue here. Remember that all our coins are in a country far far away? We will not be the ones weighing the coins, we can only send our request to the natives.

Why does this change anything, you ask? Well, for a start, there is no Internet in Absurdistan. Each time we want to make some weighings, we write down a list of requests, send it by regular post and wait a week or so for the answers.

To add insult to injury, the natives in Absurdistan are very lazy. Each time a native is asked to weigh some coins, with probability $p = 0.7$ he will ignore the request and just give you a random answer instead. That is, only 30% of your requests will actually be executed, the other 70% will receive random answers. On the other hand, the natives are precise. If the native decides to perform the requested weighing, he will always get and report the correct result.

The scales used by the natives are extremely precise balance scales. They consist of two pans (we will call them "left" and "right") that are connected by a beam with a fulcrum in the middle. One may place some coins into the left pan, some other coins into the right pan, and then read off one of three possible results: either one of the pans is heavier, or the scales balance. (It only makes sense to place the same number of coins onto each pan. If you ask to place more coins into one pan than the other, the pan with more coins will always be heavier. Of course, even if this is the case, the native may still skip the weighing and report a random answer.)

Problem specification

There are n coins, labeled 1 through n. The labeling was chosen uniformly at random. Out of the coins, $n-1$ are real and one coin is fake. The fake coin is either lighter or heavier than the real coins. As the counterfeiter was trying his best, both options are equally likely (probability 50%).

You will interact with our grader using multiple submissions. Each submission represents one letter sent to Absurdistan. You will first send some letters that require the natives to perform some weighings, and finally one letter announcing *which coin is the fake one and also whether it is heavier or lighter than the real ones*. In each letter in which you request weighings, you must request exactly k of them.

In each subproblem of this task there are two criteria you need to satisfy in order to solve the subproblem:

- You may send at most s letters. (That is, you may make at most $s-1$ submissions, each requesting k weighings, and then you must submit your answer.)
- Are you feeling lucky? Well, today your luck has run out. If you are thinking about just taking a random guess, you can forget it right now.

  Your answer will only be accepted if *you can be at least 99% certain that your answer is correct*, based on the weighings you made, their outcomes you received, and the assumption that all random events were independent and had the stated probabilities.

If you fail (by guessing incorrectly, guessing without 99% certainty, or using up all s submissions without guessing at all), **the whole subproblem is restarted** and you can try again from the beginning. A new fake coin in chosen, and you get to make another s submissions in the "new game".

Constraints

In both subproblems, the probability of a particular native being lazy is $p = 0.7$. (That's a lot!)

Easy subproblem F1: The number of coins is $n = 81$. You may send at most $s = 6$ submissions (per restart), and in each submission that requests weighings, you have to request exactly $k = 50$ of them.

Hard subproblem F2: The number of coins is $n = 250$. You may send at most $s = 11$ submissions (per restart), and in each submission that requests weighings, you have to request exactly $k = 15$ of them.

A different rule replaces the standard limit of at most 10 submissions per subproblem. Here, only *Wrong answer*s count towards the limit. In each subproblem, you may only receive a *Wrong answer* message at most 9 times. If you receive a 10th *Wrong answer*, all further submissions will be rejected.

Submission specification

The first line of your file should contain a single letter: either '`G`' (a guess) or '`W`' (a list of k weighing requests).

If your submission is a guess, the second line of your submission should contain the number of the fake coin (between 1 and n, inclusive), a space, and a letter. The letter should be '`L`' if the fake coin is lighter than the real ones, and it should be '`H`' if the fake coin is heavier.

If your submission is a list of k weighing requests, it should contain exactly k more lines. Each of those lines should contain a string of n characters that describes one weighing request. The i-th character of a request should be '`L`' if coin i should be placed on the left pan of the scales, '`R`' for the right pan, and '`-`' if the i-th coin should remain off the scales.

Evaluation result specification

If your submission is syntactically incorrect, you will receive a *Wrong answer* with an explanation. A syntax error does not cause a restart, nor is it counted in the s allowed submissions.

If your submission is a successful guess, you will receive an *OK*, thus solving the subproblem. If your guess fails, you will receive a *Wrong answer* with an explanation, and the game is restarted.

If your submission is a list of weighing requests, you will receive a *Continue* and a string of k characters. The i-th character is the result of your i-th weighing request: '`L`' if the native claims the left pan is heavier, '`R`' if he claims the right pan is heavier, and '`=`' if he claims both pans are exactly equally heavy.

Note that each request on your list is handled by a different native, and (independently of each other) each of those natives generates his reply uniformly at random with probability p.

Continue messages do not affect the team's rank. They are not worth any points, nor do they add penalty time. *Wrong answer*s are scored as usual.

Good advice

If at first you don't succeed, try again!

As there are probabilities involved, even the best strategy might sometimes fail. If you trust that your game strategy makes sense, give it another attempt if the first one doesn't make it.

(Of course, if your strategy is bad, your chance to solve this problem would be zero even if we granted you a thousand attempts.)

Example

In the example below, we have $n = 6$ coins and we request $k = 3$ weighings at a time.
One possible first submission:

submission

```
W
LLRR--
-L---R
--LRLR
```

This is what you may receive as our response:

response

```
Continue: LL=
```

This means that you got the following responses:

- Coins 1+2 are heavier than coins 3+4.
- Coin 2 is heavier than coin 6.
- Coins 3+5 are exactly as heavy as coins 4+6.

If we trusted these answers, we could now conclude that coin 2 is the fake one, and it is heavier than the real coins. (Coins 3, 4, 5, 6 have to be real from the third answer, and then coin 2 is fake and heavier from the second answer.) We could then submit the corresponding guess:

submission

```
G
2 H
```

However, in this problem we have to be certain enough before making our guess.

And right now we shouldn't be too certain yet. After all, each of those three responses has probability 70% of being the result of a random choice. The submission would be evaluated as a *Wrong answer*, and the subproblem would be restarted.

At the moment (assuming $p = 0.7$) the actual probability that "`2 H`" is the correct answer is only about 36.96%. The second most likely answer is currently "`1 H`" with probability about 16.17%. (The answer "`1 H`" corresponds, among other possibilities, to the situation when the response to `-L---R` was generated at random and the other two are correct.)

Problem G: Grid

You are given a rectangular grid consisting of $r \times c$ points. The lower left corner has coordinates $(1, 1)$, the upper right corner has coordinates (c, r). The neighbors of a point (x, y) are the points $(x - 1, y)$, $(x + 1, y)$, $(x, y - 1)$, and $(x, y + 1)$, if they exist. A path is a sequence of points such that subsequent points are neighbors and each point appears on the path at most once.

Problem specification

Given two distinct points (x_s, y_s) and (x_f, y_f), find one longest path from (x_s, y_s) to (x_f, y_f).

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 consists of a single line with six integers – r, c, x_s, y_s, x_f, y_f.

For both the easy subproblem G1 and the hard subproblem G2, you may assume that $1 \leq r, c \leq 100$ and $rc \geq 2$. Additionally, for the easy subproblem G1 you may assume that $r \leq 5$.

Output specification

For each test case, output a single string describing one possible longest path. If there are multiple longest paths, output any one of them.

A path $a_1, a_2, \ldots, a_k$ is described by a string consisting of $k - 1$ letters `U`, `D`, `L`, `R`. The i-th letter in the string describes the move from point a_i to a_{i+1}:

If $a_i = (x, y)$ and $a_{i+1} = (x, y + 1)$, the i-th letter should be `U`.
If $a_i = (x, y)$ and $a_{i+1} = (x, y - 1)$, the i-th letter should be `D`.
If $a_i = (x, y)$ and $a_{i+1} = (x + 1, y)$, the i-th letter should be `R`.
If $a_i = (x, y)$ and $a_{i+1} = (x - 1, y)$, the i-th letter should be `L`.

Example

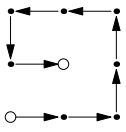
input

```
2

1 10 2 1 4 1

3 3 1 1 2 2
```

output

```
RR
RRUULLDR
```

first example:

second example: 

Problem H: Histiaeus

Sometimes, you need to send someone a message without anyone knowing about the message's existence. This is the general principle of steganography. One of the early users of steganography was an Ancient Greek ruler named Histiaeus.

Histiaeus needed to send a secret message to Aristagoras, but worried that the slave carrying the message would be intercepted. So the slave was given some innocent letters to fool the enemy spies, but also carried another message, hidden in a clever manner devised by Histiaeus. The enemy didn't notice the secret message was there, but Aristagoras knew how to find it.

Inspired by Histiaeus, we decided to send you a secret message, hiding it the same way he did. We will play the role of Histiaeus, you'll be Aristagoras, and the problem statement of *Problem H – Histiaeus* is the slave we sent to you, carrying the secret message.

(However, *this* problem statement – the one that you are reading right now – does not hold any secrets. Searching for the secret message here would be a waste of time. Now, where else could it be?)

Problem specification

Do what Aristagoras did, and find the secret message.

Input specification

There is no input.

Output specification

For both subproblems, the correct output is a single English word written in UPPERCASE. The secret message will tell you which word it is.

Problem I: Invisible cats

You are given a number of small grayscale images, each exactly 32×32 pixels in size. The images have been encrypted. The encryption is different for the easy and the hard subproblem. Both encryption types are described below.

There is a cat in the 21st image. There are also exactly ten other images with cats among the first 20 images. Find those ten cats!

Problem specification – easy subproblem

The pictures are encrypted in the following way: We picked a single random permutation on 32 elements. Then, for each picture we shuffled its columns using this permutation. That is, each column of pixels is still in its original order from top to bottom, but the order of columns is now different. (Note that we used the same permutation for all pictures.)

Problem specification – hard subproblem

The pictures are encrypted in the following way: We picked a single random permutation on 32×32 elements. Then, for each picture we shuffled its pixels using this permutation. That is, the set of pixels is now the same, they are just in different locations. (Again, note that we used the same permutation for all pictures.)

Input specification

For each subproblem you are given one set of encrypted images. Each set of images is provided in two different formats:

The first format is a ZIP archive that contains each encrypted image as a separate PGM file.

The second format is a single file that is formatted as follows: The first line contains a single integer: the number of images. For each image, you are then given 1024 integers, each in range from 0 (black) to 255 (white). The first 32 of these integers are the colors of the first row (left to right), the next 32 is the second row, and so on.

Output specification

Print ten whitespace-separated integers – the numbers of first ten pictures with cats, in ascending order. Do not use leading zeroes, even though the filenames in the ZIP archive have them.

Example

input	output
(a bunch of pictures)	1 2 3 4 5 6 7 8 9 20

Problem J: Just a single gate

Surely you have heard about logic gates, such as AND, NOT, XOR, and many others. A logic gate is a tiny device with some inputs and outputs that implements a Boolean function. That is, the inputs and outputs are boolean values (0 or 1), and for each particular gate each output is uniquely determined by its inputs.

For example, the traditional $NAND$ gate has two inputs (let's call them x and y) and one output (z). All possible outputs of this gate are given in the truth table shown below. This gate computes the "not and" function: the output is true if and only if the logical "and" of both inputs is false.

```
NAND:  x | 0 0 1 1
       y | 0 1 0 1
      ---+---------
       z | 1 1 1 0
```

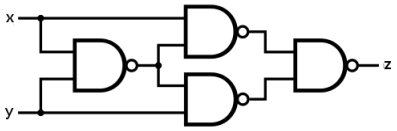
It is a well-known fact that the $NAND$ gate is *universal*: You can construct any other gate using only a finite set of suitably interconnected $NAND$s.

For example, consider the unary NOT gate – a gate that outputs 0 if the input is 1, and vice versa. This gate can be constructed using a single $NAND$ gate: $NOT(x)$ is the same as $NAND(x, x)$.

Of course, sometimes the construction is more involved. For example, to construct the binary XOR gate (a gate that returns 1 if the inputs are different and 0 if they are equal) we need at least four $NAND$ gates. One possible construction:

$$XOR(x, y) = NAND(\ NAND(x, NAND(x, y)),\ NAND(y, NAND(x, y))\ ).$$

The above expression has five $NAND$s, but $NAND(x, y)$ occurs twice, and can be implemented by a single gate in hardware, as shown in the figure below.



There are no universal unary gates, and only two universal binary gates: the gate $NAND$ described above, and the gate NOR that implements the Boolean function "not or".

Problem specification

In this problem we are interested in ternary gates: gates with three inputs and one output. An example of a ternary gate is the MAJ gate that returns the majority element – i.e., it returns 1 if at least two inputs are 1, and 0 if at least two inputs are 0. Below is the truth table of MAJ with inputs w, x, y and output z:

```
MAJ:  w | 0 0 0 0 1 1 1 1
      x | 0 0 1 1 0 0 1 1
      y | 0 1 0 1 0 1 0 1
     ---+-----------------
      z | 0 0 0 1 0 1 1 1
```

Easy subproblem J1: Find **at least seven** universal ternary gates.

Hard subproblem J2: Find **all** universal ternary gates.

Input specification

There is no input.

Output specification

Each line of your output file must describe a universal ternary gate, written as a space-separated list of zeroes and ones: the output row of its truth table, in order.

For the easy subproblem, the output must contain *at least* seven distinct rows, for the hard subproblem it has to contain all universal ternary gates. (I.e., any correct output for the hard subproblem will also be accepted if you submit it as your answer to the easy subproblem.)

Example output

```
0 0 0 1 0 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
```

This is a syntactically correct output file. It describes three ternary gates: the first row is MAJ, the second row is a gate that always returns 0, and the third row is the $AND3$ gate (ternary and): its output is the logical "and" of all three inputs.

(This is an incorrect output: it contains too few gates, and the gates it contains are not universal.)

Problem K: Knee problems

You wander through a dark dungeon. All around you there are doors of different shapes and colors. You pick one, open it and enter.

"I knew you would come," said a voice in the dark. You come closer and see an old man with a long white beard sitting on the floor.

"I used to be a problem solver like you," he says, "but then I took an arrow in the knee."

"Seriously?" you ask him.

"Well... not really. It's just what all the kids were saying the last time I saw daylight."

"So what happened to you?" you ask and sit beside him.

"The truth is, I destroyed my kneecaps on the stairs. When I was younger, I did a lot of programming contests. And in one of them was a really nasty task. I had to determine the number of ways in which one can go up and down a staircase with n steps. Of course, there were some constraints: when going up, you can take two steps at a time, and when going down, you can take up to four steps at once."

He sighs deeply. "I had no idea how to solve the task, so I found a staircase and attempted to try every possibility. But there were so many of them that I overloaded my knees and now I can't even walk. So I'm sitting here and still wondering about a solution for that problem. Can you help me to finally put a close on this?"

Problem specification – easy subproblem K1

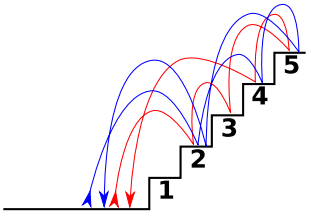
The staircase consists of n steps. Count the ways of going up and then down the staircase, given the following constraints:

- On the way up, you can take either 1 or 2 steps at a time.
- On the way down, you can take 1, 2, 3, or 4 steps at a time.

As the actual number of ways can be huge, compute the remainder it gives when divided by $10^9 + 9$.

For example, for $n = 5$ one valid way of going up and down the staircase looks as follows: Start on the ground, ascend to step 2, continue to step 3, and then go to step 5. Having now reached the top of the staircase, you turn around and walk down, first descending to step 4 and then going directly to the ground (which is, at that moment, 4 steps below).

The figure on the right shows two valid ways of going up and down the stairs for $n = 5$. The one described above is shown in red.



Problem specification – hard subproblem K2

The staircase consists of n steps. Count all ways of going up and then down the staircase, given the following constraints:

- On the way up, you can take either 1 or 2 steps at a time.
- On the way down, you can take 1, 2, 3, or 4 steps at a time.
- On the way down, you can only walk on the steps you used on the way up.

Again, your task is to compute the number of valid paths modulo $10^9 + 9$.

In the figure above, the red path is not valid for this subproblem: on the way down we walk on step 4, which was not used on the way up. The blue path ($0 \to 2 \to 4 \to 5 \to 2 \to 0$) is valid.

Input specification

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

Each test case consists of a single line with the integer n ($1 \leq n \leq 100,000$) – the number of steps.

Output specification

For each test case print a single line with one integer – the number of valid paths modulo $10^9 + 9$.

Example

input

```
2

3

5
```

output

```
12
120
```

This output is correct for the easy subproblem K1. For example, when $n = 3$ there are 3 ways to go up, and for each of them there are 4 ways to go back down.

input

```
2

3

5
```

output

```
8
52
```

This output is correct for the hard subproblem K2.

Problem L: Labyrinth

To alleviate the stress you surely experience during programming competitions, we invite you to play a fun browser-based game.

Problem specification

Your objective in the game will be to find your way through a labyrinth. You win when you reach the finish tile.

The labyrinth is full of doors that will often block your way, and switches that can be used to open and close the doors. Every door and switch is marked by a letter (or a pair of letters) and a color. Pressing a switch toggles all doors that have the same marking.

Each of the two subproblems consists of 7 levels. Send us your solutions to all 7 levels to solve the subproblem. Good luck!

JavaScript application

The game is a browser-based JavaScript application. You can either open it from the online problem statement, or open `l/game.html` in the downloadable archive.

To move, use the arrows on your keyboard or the numeric keypad. To push a button you're standing on, press either "P", the numeric keypad "5", or Enter.

You'll need a reasonably modern browser to play. Old versions of Internet Explorer probably won't work.

Input specification

There is no input.

Output specification

Your steps through the labyrinth will be recorded as a string of letters 'U', 'D', 'L', 'R', and 'P' (meaning "up", "down", "left", "right", and "push", respectively). Collect the solution strings for all 7 levels in a text file and submit it.

The output file must contain 7 whitespace-separated strings. The i-th string must be a solution for the i-th level in the subproblem.

Your output file must not contain more than 1,000,000 characters.

Problem M: Morning hassle

Peter was supposed to catch a morning train at 7:30. But as usual, he overslept the alarm set to 6:30 and he just woke up with 7:29 on the clock. He only managed a single "Oh crap!" before the train was gone. Luckily for Peter, everything can still be saved. He can take his old car out of the garage and drive it to his destination.

Peter lives in the middle of an abandoned countryside. There is a single long straight road going across the countryside. Peter's home and his destination both lie on this road.

Still, Peter has a valid reason to prefer the train. The whole countryside is covered by train tracks, and thus the road is riddled with railroad crossings. And as trains have priority over cars, you could easily end up waiting for a long time at some of those crossings.

Moreover, even if you are not waiting for a train to pass, you still need to approach the railroad crossings carefully – they are in pretty bad shape and if Peter were to drive carelessly, the crossing could easily break his old car.

Problem specification

For the purpose of this task, the road is an infinite straight line. Peter's home, his destination, each of the railroad crossings, and Peter's car should all be considered points. Peter's car is the only point that will be moving, all the other ones are stationary. The movement of Peter's car is continuous (not discrete).

We will be using a linear coordinate system on the road, with Peter's home being at 0 and his destination at some $x_{end} > 0$. All coordinates are in meters, all speeds are in meters per second, all accelerations are in meters per second squared.

All the railroad crossings lie strictly between Peter's home and his destination, at pairwise different coordinates x_i. In the input, their descriptions are ordered by their coordinate.

There are no other cars on the road. Peter's car can move freely along the line, including the parts that are not between his home and his destination. However, the movement of his car is subject to the following constraints:

- The car's acceleration (change of velocity over time) has to be between $-a_{max}$ and a_{max}, inclusive. (E.g., if your current speed is $v = 20$ and $a_{max} = 1.2$, after 0.5 seconds your speed can be anything between 19.4 and 20.6, inclusive.)
- The car cannot enter a crossing when the crossing is closed because of a passing train.
- In general, the speed of the car has no upper bound – it can go arbitrarily fast.

  However, the railroad crossings are special: The maximum allowed speed at a railroad crossing is v_{max} (an integer).

  But even when driving slower than v_{max}, the bumping while crossing the railroad sometimes tends to resonate parts of Peter's car and Peter fears that the car might break. He has already tested that this does not happen if the speed of his car is an integer less than or equal to v_{max}. He now refuses to drive over a railroad crossing using any other speed.

  Therefore, whenever the car crosses a railroad, its speed at that moment has to be a positive integer.

  (Note that zero is not allowed. Stopping at a railroad crossing is forbidden by law.)

Peter's car starts stationary ($v = 0$) at his home. Calculate the shortest time in which it is possible to park the car (i.e., have $v = 0$ again) at Peter's destination (x_{end}).

Input specification

The first line of the input file contains an integer $t \leq 500$ specifying the number of test cases. Each test case is preceded by a blank line.

Each test case starts with a single line containing four numbers:

- a floating-point number x_{end} ($0.5 \leq x_{end} \leq 1500.0$): the destination
- a floating-point number a_{max} ($0.1 \leq a_{max} \leq 10.0$): the maximum acceleration
- an integer v_{max} ($1 \leq v_{max} \leq 40$): the maximum speed over a crossing
- an integer n: the number of railroad crossings (constraints are given below)

Next n lines describe railroad crossings, one per line. Each line starts with two numbers. The first one a floating-point number x_i: the coordinate of this crossing. The second one is a nonnegative integer m_i: the number of trains that will be passing through the crossing. Then, $2m_i$ floating-point numbers follow: for each train the start $s_{i,j}$ and the end $e_{i,j}$ of the time interval when the crossing is blocked by the train.

You may assume the following things about the crossings:

- Their coordinates are in sorted order: $0 < x_1 < \cdots < x_n < x_{end}$.
- The intervals when the crossing is blocked are given in chronological order, they do not overlap, and they do not even touch (i.e., the end of one interval is always strictly less than the start of the next one). The first interval starts at 0 or later, the last interval ends at 10^6 or sooner.
- Assume that the intervals are open – if you arrive precisely at their start or end, you are still able to cross in either direction.

There should be no numerically unstable test cases in the test data. More precisely, for each test case we used: 1) if we make small changes to the values $s_{i,j}$ and $e_{i,j}$, the optimal path remains essentially the same; and 2) if we make a small change to a_{max}, the optimal path remains essentially the same.

Subproblem-specific constraints

In the easy subproblem M1, $0 \leq n \leq 1$ (i.e., there is at most one crossing) and $0 \leq m_i \leq 2$ (there are at most two trains per crossing). In the hard subproblem M2, $0 \leq n \leq 30$ and $0 \leq m_i \leq 25$.

Output specification

For each test case, output a single line with a floating-point number on it – the earliest time Peter can be parked ($v = 0$) at position x_{end}. Output sufficiently many decimal places. Answers with absolute or relative error up to 10^{-6} will be accepted.

Example

input

```
1

10 1 3 0

10 1 30 1
5 1 2 3
```

output

```
6.32455532034
6.32882800594
```

In the first case, there is no crossing and so Peter may drive directly to his destination. The optimal strategy is to accelerate until he reaches $x = 5$, and then to brake for the rest of the way. Note that his maximum speed during this trip will exceed v_{max}.

In the second case the train will leave the crossing before Peter can possibly reach it. Still, the crossing limits the car speed. In the optimal solution Peter will cross the railroad crossing having speed $v = 3$.

