Problem A. As Much As 996

Input file: `standard input`
Output file: `standard output`

Hello again. Donald, with all the judges, hope you are well, and ready for another round of challenges. Judges were divided on the difficulty of this contest, but we all believe that most of the problems are to be more interesting than usual monthly contests. Wish you good luck and enjoy the contest!

As the first problem of this contest, Donald promises you this will be the easiest problem. Well, it's up to you to skip and start from the middle, where you will find another easy problem, hopefully as easy as this one.

As a programmer, you might know that programmers have a self-invented way to represent their working hours, namely *std*, which means, start working at s a.m., end working at t p.m. and working d days a week. You might also know that recently there has been a campaign called 996.ICU which objects to long working hours and demands rights.

As a student, you might not care about any of these. You don't need to — the goal of this problem, is to write a program that will output the length of working hours per week based on the "*std*-representation".

Input

The first and the only line of input contains *std*, with no space separating them. It's guaranteed that $2 \leq s, t \leq 11$, $1 \leq d \leq 7$. Note that s and t cannot be 12 or 1 in this problem, since 12 a.m. and 12 p.m. often raises ambiguity and hopefully you won't struggle with them, and 1 a.m. or 1 p.m. is not likely to be the time when people start working or stop working, respectively.

Output

Output working hours per week. If you have enumerated all the possibilities, it's easy to see there is only one possible answer, given the constraints above.

Examples

standard input	standard output
996	72
955	40
8107	98
1127	21
2117	147

Problem B. Black Peter

Input file: `standard input`
Output file: `standard output`

Black Peter is a Victorian card game for two or more players probably deriving from an ancient gambling game in which the loser pays for the drinks, also known as Old Maid in western countries, "Zhua Wu Gui" in China.

There are retail card decks specifically crafted for playing Old Maid, but the game can just as easily be played with a regular deck of 52 cards. When using a regular deck, a card is either added or removed, resulting in one unmatchable card. The most popular choices are to remove the ace or queen of clubs or to add a single joker. It is also possible to remove one card face-down from the top of the deck before hands are dealt; if this is done, players will not know which card is unmatchable. The unmatchable card becomes the "old maid," and whoever holds it at the end of the game is the loser.

Let's discuss the situation with two players (let's call them Brett and Caoimhe) only, beginning with Brett, who plays first. Brett, who can only see Caoimhe's hand face-down, selects a card without looking and adds it to his hand. He then sees if the selected card makes pair with any of his original cards. If so, the pair is discarded face up as well. Then it's Caoimhe's turn. She offers her hand to Brett, and does the same thing, and so on. The objective of the game is to continue to take cards, discarding pairs, until no more pairs can be made. The player with the card that has no match is "stuck with the old maid" and loses.

As an observer, you know in advance the hands of both Brett and Caoimhe, and therefore want to analyze the probability that Brett will win; but first, we will do some assumptions to make your life easier.

1. When the players see any pairs of matchable cards, they immediately discard them, no matter whether it is the beginning of the game, or in the middle.
2. When one of them selects one of the cards in the opponent's hand, each card is equally likely to be chosen.

Input

The first line of the input is a positive integer t, which stands for the case numbers.

For each test case, there are three lines, the first of which is an integer n ($2 \le n \le 10^6$). The card deck, then, ranks from 1 to n and consists of exactly two suit colors, thus $2n$ cards altogether. One of the card is removed beforehand, ending up $2n - 1$.

The two lines following are the hands of Brett and Caoimhe, respectively ($b_1 b_2 \dots b_n$, $c_1 c_2 \dots c_n$, $b_i, c_i \in \{0, 1, 2\}$). Each hand is represented using a string of length n. b_i is the number of cards in rank i that Brett has; same for Caoimhe. It's guaranteed that for $1 \le i \le n$, $1 \le b_i + c_i \le 2$; and there exists only and exactly one k, such that $b_k + c_k = 1$. Initially, each player has at least one card, i.e., $\sum_{i=1}^{n} b_i > 0$, and $\sum_{i=1}^{n} c_i > 0$.

It's guaranteed that the sum of n in all t test cases is not greater than 10^6.

Output

For each test case, output the probability that Brett will win.

You answer is considered correct, if its absolute or relative error does not exceed 10^{-9}. Namely, let your answer be a, and the jury's answer be b. Your answer is considered correct, if $\frac{|a-b|}{\max(1,|b|)} \le 10^{-9}$.

Examples

standard input	standard output
3 3 100 022 10 2020202101 0202020111 7 1111112 1111100	0 0.75 0.3333333333333333

Problem C. Coprime-Free Permutation

Input file: `standard input`
Output file: `standard output`

Given $1, 2, 3, \ldots, n$, you are now challenged to select as many of them as possible and reorder them in a row, so that any adjacent pair has a greatest common divisor of at least 2.

Input

The first and only line of the input consists of an integer n ($4 \leq n \leq 10^6$).

Output

The first line of the output should be a positive integer k: the number of integers you have selected.

Now you have to output k distinct integers $a_1, a_2, \ldots, a_k$ ($1 \leq a_i \leq n$), space-separated; and for $1 \leq i < k$, $\gcd(a_i, a_{i+1}) \geq 2$.

Any solution is acceptable if there are multiple of them.

Examples

standard input	standard output
4	2 2 4
19	14 7 14 2 12 10 5 15 3 6 9 18 4 16 8

Problem D. Distinct Tautonyms

Input file: `standard input`
Output file: `standard output`

A sequence is called a tautonym if and only if it is a concatenation of two copies of another sequence of length at least 1.

For example, `1 1` and `2 1 1 2 1 1` are tautonyms, while `1` and `1 3` are not.

Count how many **distinct** subsequences (not necessarily continuous) of a given sequence are tautonyms. Output the answer modulo $10^9 + 7$.

Two subsequences $a_{i_1}, a_{i_2}, \ldots, a_{i_p}$ and $a_{j_1}, a_{j_2}, \ldots, a_{j_m}$ are considered different if $p \neq m$ or there exists at least one k $(1 \leq k \leq p)$ such that $a_{i_k} \neq a_{j_k}$.

Input

The first line of the input consists of one integer n $(2 \leq n \leq 700)$.

The second line consists of n space-separated integers $a_1, a_2, \ldots, a_n$ $(1 \leq a_i \leq n)$.

Output

Output the answer modulo $10^9 + 7$.

Examples

standard input	standard output
4 1 2 1 2	3
7 7 6 5 4 3 2 1	0
6 1 3 3 3 3 1	3

Problem E. ECNU MNIST

Input file: `standard input`
Output file: `standard output`

For those with some previous experience on Deep Learning, understanding this task might be a little easier, but Donald assures you that the implementation is not very convenient and this is NOT one of the most accessible problems of the contest. If you are still determined to take a look, here we go.

The main goal of this task is the classification of four English letters 'E', 'C', 'N', 'U'. You are challenged to receive the input of about 2000 images, decide what they are, and output your predictions. If your predictions are good enough, you will get accepted. See the specification part for details.

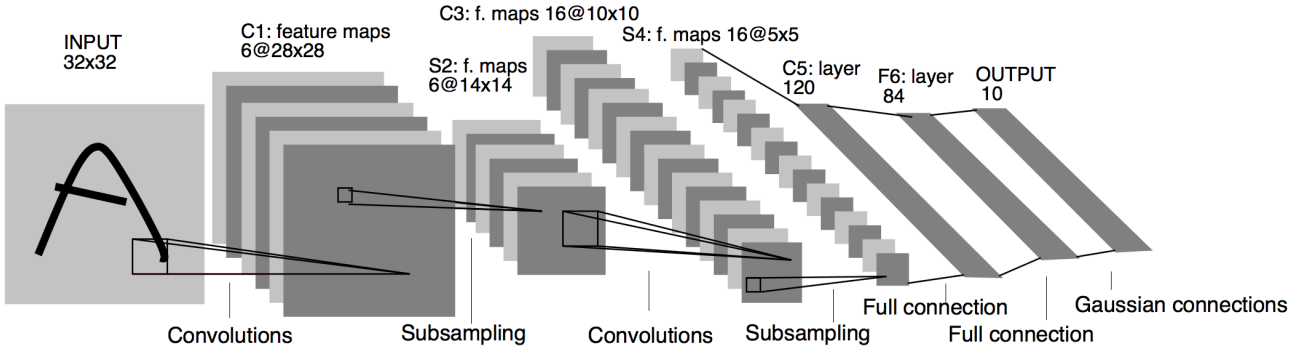


Figure 1: LeNet-5 Architecture, similar to what you are going to see, with subtle differences

Specification

The test data generation of this task is a little different from usual.

Donald builds a sequential neural network as follows:

1. Input: a binarized[1] image of size 28×28. Only one channel (black and white).
2. Conv layer 1: kernel size 5×5, 1 channels in, 4 channels out, output size is $4 \times 24 \times 24$.
3. Max pooling: pool size 2×2, reducing to $4 \times 12 \times 12$.
4. Apply ReLU.
5. Conv layer 2: kernel size 3×3, 4 channels in, 9 channels out, output size is $9 \times 10 \times 10$.
6. Max pooling: pool size 2×2, reducing to $9 \times 5 \times 5$.
7. Apply ReLU.

[1]Binarization is to replace each pixel in an image with a black pixel if the pixel intensity is less than certain value, or a white pixel if greater or equal.

8. Flatten: resize the multi-dimensional array into a 225-dimensional vector, following the lexicographical order of indices $(M_{0,0,0}, M_{0,0,1}, \ldots)$.
9. Full-connected layer 1: reducing 225 dimensions into 64 dimensions.
10. Apply ReLU.
11. Full-connected layer 2: reducing 64 dimensions into 4. And find out which one of four as the largest score. If it's 0, then prediction is 'C'; 'E' if 1; 'N' if 2 and 'U' if 3.

Then he applies softmax, calculates cross entropy and spends several minutes do the training and export all the parameters he has captured in the network.

In the Appendix section, there is a neural-network tutorial attached, which will explain all the details. For now, Donald assumes you have understood how this works and will explain the format of the input you have been given.

The first line contains a magic number; it is for test generating purposes; you can safely ignore this.

The parameters of a convolutional layer consist of two parts: weight (a multi-dimensional matrix of size $channel_{out} \times channel_{in} \times 5 \times 5$), and bias (a $channel_{out}$-dimension vector). For here and the rest of this problem, a multi-dimensional matrix of size $s_1 \times s_2 \times \cdots s_k$ is given in the input as a two-dimensional matrix of $(\prod_{i=1}^{k-1} s_i) \times s_k$, where the row order follows the convention of lexicographical order of the indices; a n-d vector is given in a line as n space-separated numbers.

The parameters of a fully-connected layer consist of weight ($D_{out} \times D_{in}$) and bias (D_{out} vector).

The parameters of Conv layer 1, Conv layer 2, FC layer 1 and FC layer 2 are given in order. So you will see:

1. Conv layer 1 weight: $4 \times 1 \times 5 \times 5$.
2. Conv layer 1 bias: 4-D vector.
3. Conv layer 2 weight: $9 \times 4 \times 3 \times 3$.
4. Conv layer 2 bias: 9-D vector.
5. FC layer 1 weight: 64×225.
6. FC layer 1 bias: 64-D vector.
7. FC layer 2 weight: 4×64.
8. FC layer 2 bias: 4-D vector.

And then, you will see images. The test data are originally drawn from the EMNIST dataset, with 5600 samples for each letter. For cross-validation purposes, Donald shuffled and split the test data into ten subsets, with 2240 samples each. Each test, except the example, has parameters trained on the other nine and presents the one subset the model has not seen before. So you will see $t = 2240$ in a line and the following $28 \cdot t$ lines contains t 28×28 01-images. Output the one you think it is in a line.

Donald guarantees you that a successful implementation of this neural network will reward you with an accuracy of over 97%. To get accepted, all you need is to reach 91.02%.

You can use any other method and go without the given parameters, which is not against any rule. You are all talented programmers, and Donald is convinced that you will figure something out, at least.

Examples

standard input	standard output
42 <20x5 matrix omitted> <1x4 matrix omitted> <108x3 matrix omitted> <1x9 matrix omitted> <64x225 matrix omitted> <1x64 matrix omitted> <4x64 matrix omitted> <1x4 matrix omitted> 4 00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000001111000000011111111000 00000011111111111111111111000 00000011111111111111111000000 00000011111111111110000000000 00000011110000000000000000000 00000011110000000000010000000 00000011100000000001110000000 00000011000000000011110000000 00001111000000011111110000000 00001111001111111111000000000 00001111111111111000000000000 00001111111111111000000000000 00011111100000000000000000000 00001110000000000000000000000 00011110000000001110000000000 00011110100000001110000000000 00011111111111111110000000000 00011111111111111100000000000 00001111111111110000000000000 00001111111100000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 <3 28x28 images omitted>	E C N U

You can find the download link here, in case you ever need it:

https://drive.google.com/open?id=1Wy--FYeNplslTr4_EwRcGeLMRB0_4AhT

Appendix

Before going into convolutional layers, let's review a few basic concepts of neural networks. If you have not learned before, this could also be a good start.

1. Tensor: you can safely take it as a vector, a matrix or a multi-dimensional matrix, in this task.
2. Fully-connected layer: assume the input is a vector t, output is another vector y, then $y = a(Wt+b)$ is known as a fully-connected layer, where W is a matrix, b is a vector and a is known as an activation function.
3. Activation function: this is to add non-linearity to the network. In our task, only one is used: ReLU. ReLU applies to a vector or a matrix, element-wise. $\text{ReLU}(x) = \max(0, x)$.

Convolutional Layers[2]

The fundamental difference between a densely connected layer and a specialized layer in the convolution operation, which we will call the convolutional layer, is that the dense layer learns global patterns in its global input space, while the convolutional layers learn local patterns in small windows of two dimensions.

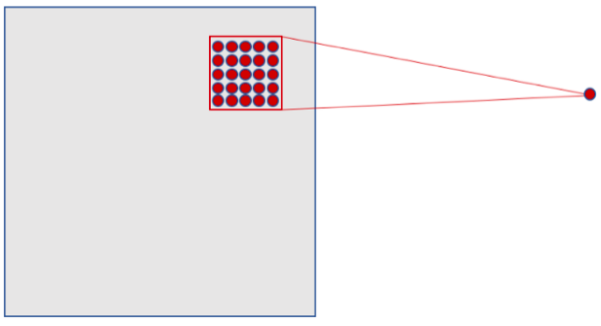
Intuitively, we could say that the main purpose of a convolutional layer is to detect features or visual features in images such as edges, lines, color drops, etc. This is a very interesting property because, once it has learned a characteristic at a specific point in the image, it can recognize it later in any part of it. Instead, in a densely connected neural network, it has to learn the pattern again if it appears in a new location of the image.

Another important feature is that convolutional layers can learn spatial hierarchies of patterns by preserving spatial relationships. For example, a first convolutional layer can learn basic elements such as edges, and a second convolutional layer can learn patterns composed of basic elements learned in the previous layer. And so on until it learns very complex patterns. This allows convolutional neural networks to efficiently learn increasingly complex and abstract visual concepts.

In general, the convolutions layers operate on 3D tensors, called feature maps, with two spatial axes of height and width, as well as a channel axis also called depth. For an RGB color image, the dimension of the depth axis is 3, because the image has three channels: red, green and blue. For a black and white image, such as the MNIST digits, the depth axis dimension is 1 (gray level).

In the case of MNIST, as input to our neural network we can think of a space of two-dimensional neurons 28×28 (height = 28, width = 28, depth = 1). The first layer of hidden neurons connected to the neurons of the input layer that we have discussed will perform the convolutional operations that we have just described. But as we have explained, not all input neurons are connected with all the neurons of this first level of hidden neurons, as in the case of densely connected neural networks; it is only done by small localized areas of the space of input neurons that store the pixels of the image.

The explained, visually, could be represented as:

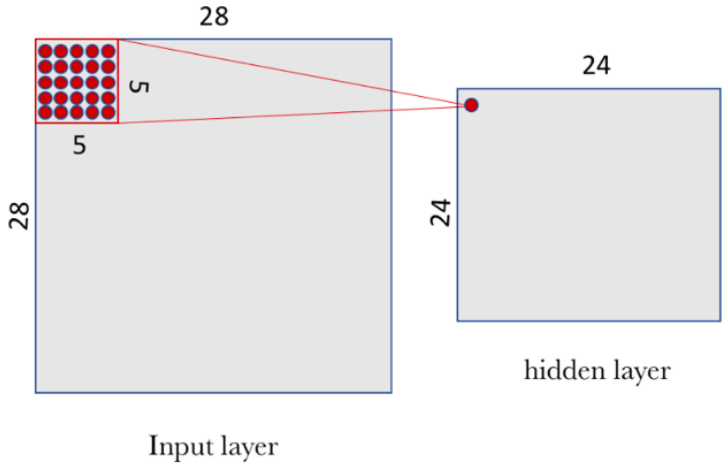


[2]Convolutional Neural Networks for Beginners, Practical Guide with Python and Keras, Jordi TORRES.AI, Sep 24, 2018, with modifications

In the case of our previous example, each neuron of our hidden layer will be connected to a small region of 5×5 neurons (i.e. 25 neurons) of the input layer (28×28). Intuitively, we can think of a 5×5 size window that slides along the entire 28×28 neuron layer of input that contains the image. For each position of the window, there is a neuron in the hidden layer that processes this information.

Visually, we start with the window in the top left corner of the image, and this gives the necessary information to the first neuron of the hidden layer. Then, we slide the window one position to the right to "connect" the 5×5 neurons of the input layer included in this window with the second neuron of the hidden layer. And so, successively, we go through the entire space of the input layer, from left to right and top to bottom.

Analyzing a little bit the concrete case we have proposed, we note that, if we have an input of 28×28 pixels and a window of 5×5, this defines a space of 24×24 neurons in the first hidden layer because we can only move the window 23 neurons to the right and 23 neurons to the bottom before hitting the right (or bottom) border of the input image.



We would like to point out to the reader that the assumption we have made is that the window moves forward 1 pixel away, both horizontally and vertically when a new row starts. Therefore, in each step, the new window overlaps the previous one except in this line of pixels that we have advanced.

In our case of study, and following the formalism previously presented, to "connect" each neuron of the hidden layer with the 25 corresponding neurons of the input layer we will use a bias value b and a W – weights matrix of size 5×5 that we will call filter (or kernel). The value of each point of the hidden layer corresponds to the scalar product between the filter and the handful of 25 neurons (5×5) of the input layer.

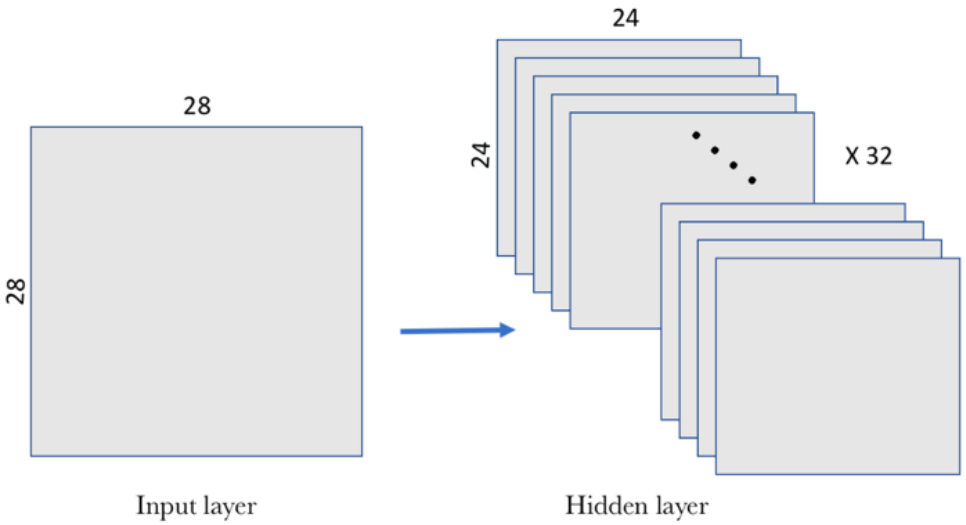
However, the particular and very important thing about convolutional networks is that we use the same filter (the same W matrix of weights and the same b bias) for all the neurons in the hidden layer: in our case for the 24×24 neurons (576 neurons in total) of the first layer. The reader can see in this particular case that this sharing drastically reduces the number of parameters that a neural network would have if we did not do it: it goes from $14,400$ parameters that would have to be adjusted ($5 \times 5 \times 24 \times 24$) to 25 ($5 \times 5$) parameters plus biases b.

This shared W matrix together with the b bias, which we have already said we call a filter in this context of convolutional networks, is similar to the filters we use to retouch images, which in our case are used to look for local characteristics in small groups of entries.

But a filter defined by a matrix W and a bias b only allows detecting a specific characteristic in an image; therefore, in order to perform image recognition, it is proposed to use several filters at the same time, one for each characteristic that we want to detect. That is why a complete convolutional layer in a convolutional neuronal network includes several filters.

A usual way to visually represent this convolutional layer is shown in the following figure, where the level

of hidden layers is composed of several filters. In our example, we propose 32 filters, where each filter is defined with a W matrix of 5×5 and a bias b.



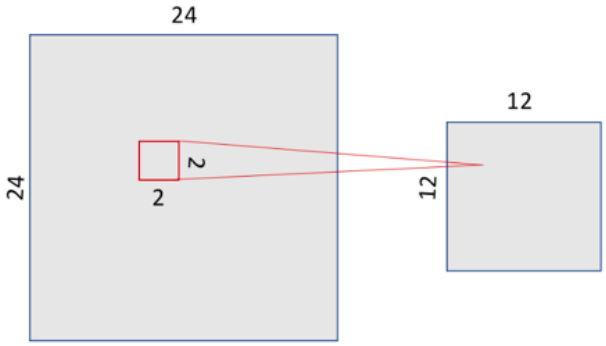
In this example, the first convolutional layer receives a size input tensor $(28, 28, 1)$ and generates a size output $(24, 24, 32)$, a 3D tensor containing the 32 outputs of 24×24 pixel result of computing the 32 filters on the input.

Also notice that if you have multiple channels/filters as your input, each filter of your output comes from the sum of each filter of your input with kernel applied. This makes $channel_{out} \times channel_{in}$ filters altogether, but there are still $channel_{out}$ biases, to be clear.

Pooling Layers

In addition to the convolutional layers that we have just described, convolutional neural networks accompany the convolution layer with pooling layers, which are usually applied immediately after the convolutional layers. A first approach to understand what these layers are for is to see that the pooling layers simplify the information collected by the convolutional layer and create a condensed version of the information contained in them.

In our MNIST example, we are going to choose a 2×2 window of the convolutional layer and we are going to synthesize the information in a point in the pooling layer. Visually, it can be expressed as follows:



There are several ways to condense the information, but a usual one, which we will use in our example, is known as max-pooling, which as a value keeps the maximum value of those that were in the 2×2 input window in our case. In this case, we divide by 4 the size of the output of the pooling layer, leaving an image of 12×12.

Problem F. Foreigners' Trouble

Input file: standard input
Output file: standard output

Foreigners like abbreviations. That's true, even if they sometimes forget about the exact meanings themselves. For example, you write "East China Normal University" as ECNU today; tomorrow, you may totally forget about it, and reinterpret ECNU as Electronic Circuit National Union, or European Central Norwich University, or anything as ridiculous as that. Sometimes you can guess what an abbreviation means according to the context, but sometimes you have permanently lost such information.

In order to investigate how serious this problem can be, Donald found a dictionary, teeming with all sorts of phrases and their abbreviations, from the very start to the very end. Donald wonders how many unordered pairs of phrases have the same abbreviations in the dictionary.

Input

The first line is a positive integer n ($1 \le n \le 5 \cdot 10^5$), which is the number of phrases in the dictionary.

Each of the following n lines contains a phrase, made up of at least 2 words; exactly one space separates adjacent words. There are no leading spaces in front of a line and no trailing spaces either. Each word is an alphabetical string, all the letters in lower case, except the first one in upper case; the word can be as short as 1 (only one uppercase letter), and as long as 11. The number of words provided altogether in all phrases does not exceed 10^6.

Phrases are all distinct, and sorted lexicographically, as dictionaries always do. Uppercase letters are first converted into lowercase letters during sorting. Following the convention of ASCII codes, space is smaller than any letter.

Output

Output a number in one line: the number of unordered pairs which have conflicted abbreviations.

Examples

standard input
5 East China Normal University Electronic Circuit National Union European Central Norwich University School Community Partnership Council Shanghai Collegiate Programming Contest
standard output
4

standard input	standard output
3 C S L O X X Orz Orz	0

Note

The input file may exceed 10 megabytes. Take care of your IO as it usually takes time and requires necessary optimizations.

Problem G. Green-Red Tree

Input file: `standard input`
Output file: `standard output`

This is an interactive problem.

Donald, who was an expert of Green-Red Tree, has recently come up with a new gambling game which has immediately received much attention, well, at least your attention.

The rules of this game are straightforward. Two players alternate their turns in the game: you and Donald. At the start of the round, Donald will give you n, which is the number of vertices available in this round, and for any two different vertices, there is an undirected edge connecting them available. Then you go first. In your turn, you will pick two different undirected edges. The edges you choose must also not coincide with edges you have selected before. Then in Donald's turn, he will decide to paint one of them red and one of them green. After $n-1$ turns, if the red edges form a tree, and the green edges also form a tree, then you win; otherwise, you lose.

Donald may know much about Green-Red Tree but doesn't have much interest in gambling. So he decides not to investigate any optimal strategies in this game. Instead, he will choose one of his two possible moves with equal probability. In other words, he will paint the first edge red with probability 0.5; if the first edge is painted red, the second one is green. If he does not do that, which is equally likely, he will paint the first edge green and the second red.

The randomness of Donald's turns should convince the players that the croupier will make random and turns, and sometimes ruin his chances to win.

You have to develop such a strategy to win in every round, before Donald decides to develop a strategy that forbids you from doing so.

Interaction Protocol

In the first line, you are given one positive integer t, which is the number of the following rounds.

At the beginning of each round, you will be told one integer n ($5 \leq n \leq 10^5$). It's also guaranteed that the sum of n in all test cases does not exceed 10^5.

After that, you will start to pick edges. Each time, you will pick two **different** edges (u_1, v_1) and (u_2, v_2) ($u_1 \neq v_1$, $u_2 \neq v_2$), which has not been picked before, and you output them in a line with the format of u_1 v_1 u_2 v_2 with space among them.

Then Donald will decide. If he outputs 1 in a line, it means that he wants the (u_1, v_1) green and (u_2, v_2) red; otherwise, he will output 0, in case he wants the first one red.

If you have violated any of the rules mentioned above, which may result in Donald's unsatisfactory, he may leave the game early and cause your program to hang. So the final verdict can be anything among "Idleness Limit Exceeded", "Wrong Answer" or "Runtime Error". "Time Limit Exceeded", which is another thing if your program exceeds the CPU time limit, usually meaning the same thing as you have seen in other problems.

Examples

standard input	standard output
1 5 1 1 0 0	 3 4 1 5 2 3 5 3 4 5 2 5 4 2 2 1

Appendix

If you have participated in the warmup round, you might have see the "Guess the Number" problem; in case you haven't, we hope the following information may help.

Sometimes you can meet interactive problems on programming contests[3].

In problems of this type, the input data given to your program may be not predetermined but is built specifically for your solution. Jury writes a special program – interactor, such that its output is transferred to the input of your solution, and the output of your program is sent to interactor's input. In the other words, your solution and the interactor exchange the data and decide what to print based on the "history of communication".

When you write the solution for the interactive problem it is important to keep in mind that if you output some data it is possible that this data is first placed to some internal buffer and may be not directly transferred to the interactor. In order to avoid such situation you have to use special flush operation each time you output some data. There are these flush operations in standard libraries of almost all languages. For example, in C++ you may use `fflush(stdout)` or `cout << flush` (it depends on what do you use for output data – `scanf/printf` or `cout`). In Java you can use method `flush` for output stream, for example, `System.out.flush()`. In Python you can use `stdout.flush()`. In Pascal you can use `flush(output)`.

There are some features for interactive problems:

- Input/output in interactive problems works much slower than in usual problems try to use `scanf/printf` instead of `cin/cout` in C++, `BufferedReader/PrintWriter` in Java and etc.
- Usually, manual testing of the solutions for interactive problems much more difficult, because the participant needed to be in the role of interactor during testing.
- Output `endl` in `cout` in C++ performs flush operation automatically.

[3]Copied from `https://codeforces.com/blog/entry/45307`, with modifications.

Problem H. Huashui Clock

Input file: standard input
Output file: standard output

For those who don't speak Chinese, huashui may not even be a legitimate word for you. You don't have to know what it is to solve this problem, but to make everything clear, huashui refers to the status of not working, especially at working hours.

Some people decide to make some plans on when to work, and when to huashui. So they invent a lockscreen app with a built-in clock. This clock is just like any other usual clock, except that it has h hours a day and m minutes an hour. Indicator of seconds is not displayed in the app, so you can safely ignore that. Therefore the time display ranges from "$0:0$" to "$h-1:m-1$".

Well, the plan works like this: if the minute number is greater than or equal to the hour number, this minute is a "huashui minute"; otherwise you are going to work.

You are a hard-worker, and you probably disdain the idea of this clock; but Donald, as a huashui-lover, is obsessed with this app and would very much like to know how much time in a day he can huashui. Help him calculate that and you will win a balloon.

Input

The only line of the input consists of two integers h and m ($2 \leq h, m \leq 10^9$), with space between.

Output

Output huashui time divided by the length of time in a day in a reduced fraction form.

Examples

standard input	standard output
2 2	3/4
2 7	13/14
7 2	3/14
13 11	6/13
100 33	17/100
100005 100009	50007/100009
1000000000 2	3/2000000000
2 999999999	1999999997/1999999998
914067307 998244353	541210700/998244353

Note

It would be interesting if you have got a "Wrong Answer" on this problem and did not do it on purpose.

Problem I. Induced Metric Space

Input file: `standard input`
Output file: `standard output`

In mathematics, a metric space is a set together with a metric on the set. The metric is a function that defines a concept of distance between any two members of the set, which are usually called points. The metric satisfies a few simple properties. In the context of this specific problem, we state the following:

- the distance from a point to itself is zero,
- the distance between two distinct points is non-negative,
- the distance from A to B is the same as the distance from B to A, and
- the distance from A to B (directly) is less than or equal to the distance from A to B via any third point C, i.e., $d(A, B) \leq d(A, C) + d(C, B)$.

There is a set S with n points. Some (possibly all, or none) distances are already known, while others are still indeterminate. Your task is to fill all the blanks, **without modifications to existing ones**, and make distances between any two points available, so that it is a metric space.

Input

The first line of the input is one positive integer t, which is the number of test cases that follows.

Then follows t test cases. Each test case starts with a line of an integer n ($2 \leq n \leq 500$), which is the number of points in S. The next n lines each contains n integers. The j-th integer in the i-th line is $d(i, j)$ ($-1 \leq d(i, j) \leq 1000$). If $d(i, j)$ is -1, it is not available yet, otherwise it is decided and should not be modified in your output.

The sum of n in all test cases does not exceed 500.

Output

For each test case, if it's not possible to complete the task, output "`NO`" in a line; otherwise output "`YES`".

If it's a "`YES`", another n lines of n space-separated integers should follow, which adheres to the input format — j-th integer in i-th line representing $d(i, j)$ ($0 \leq d(i, j) \leq 10^9$). The distances should all be non-negative and form a valid metric space.

If there are multiple solutions, any one is acceptable.

Examples

standard input	standard output
4 3 0 3 3 3 0 3 3 3 0 3 0 0 0 0 0 -1 0 -1 0 3 5 6 7 -1 -1 -1 -1 -1 -1 3 -1 3 5 -1 -1 3 -1 -1 -1	YES 0 3 3 3 0 3 3 3 0 YES 0 0 0 0 0 0 0 0 0 NO YES 0 3 5 3 0 3 5 3 0

Problem J. Josephus Problem

Input file: `standard input`
Output file: `standard output`

In computer science and mathematics, the Josephus problem (or Josephus permutation) is a theoretical problem related to a certain counting-out game.

People are standing in a circle, in the order of $1, 2, 3, \ldots, n$ clockwise, waiting to be executed. Counting begins at 1 in the circle and proceeds around the circle, in the clockwise direction. After a specified number of people are skipped, the next person is executed. The procedure is repeated with the remaining people, starting with the next person, going in the same direction and skipping the same number of people, until only one person remains, and is released.

The typical problem statement is like: given the number of people n, and number to be skipped, find out which person is so lucky to avoid execution.

Donald realized that for a talented programmer like you, you have probably seen this problem for a million times; so he proposed a more challenging version. For this to happen, let's first generalize the statement above.

For this to happen, everyone has to do counting of natural numbers. For example, the first guy shouts t; and the second guy counts $t + 1$, then $t + 2$ and etc. People are executed when what they are counting is a multiple of k. Probably, you have noticed that the first person can actually choose a nice t, so that a certain person is freed.

Donald also noticed that. What he also discovered is that you cannot always free anyone you want. Actually, in many cases it's impossible. So he modified the game rule a little bit so that people are killed not only when the counting is a multiple of k ($2 \le k \le 9$), but also it contains k in its decimal representation. For example, 19 is not a multiple of 9, but it contains a 9.

There is a rigorous proof that shows it is always possible to do so, given the constraints in this problem; of course this is left to you.

Input

The first line of the input contains an integer tc, indicating the number of test cases ($1 \le tc \le 10^4$).

Then for each test case, there is one line containing three space-separated integers n, k and x ($2 \le n \le 10^6$, $2 \le k \le 9$, $1 \le x \le n$), which is the number of people in a circle, k as explained in the statement above, and the person you want to free.

It's guaranteed that the sum of n from all test cases is no more than 10^6.

Output

For each test case, output the t ($1 \le t \le 10^{18}$) you have chosen to free x. If there are multiple t's available, you can output any of them.

Examples

standard input	standard output
40 3 2 3 3 2 2 3 2 3 3 2 2 3 2 3 3 2 2 3 2 3 3 2 2 3 2 3 3 2 2 10 3 5 10 3 4 10 3 3 10 3 5 10 3 4 10 3 3 10 3 10 10 3 9 10 3 8 10 3 1 10 3 10 10 3 9 10 3 5 10 3 9 10 3 8 7 9 7 7 9 6 7 9 5 7 9 4 7 9 3 7 9 2 7 9 1 7 9 7 7 9 6 7 9 7 7 9 6 7 9 5 7 9 4 7 9 3 7 9 2	1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Problem K. Keyboard without Binary Digits

Input file: standard input
Output file: standard output

Donald has recently received an antique typewriter from his friends, which is unquestionably not something you could have found in any retail store. The alphabetical letters are just fine, but typing numbers, or digits, can be a lot more painful. It's likely that whoever designed this keyboard is a binary hater. The "0" button is substituted with a "=", and the "1" button is replaced with a "+". So there is no way to type a number containing 0 or 1, except there is a workaround, that is, to type some numbers whose sum is the number you want.

There is a number n, which Donald indeed wants to have on his screen (or paper, or whatever), but sadly it has at least one of 0 or 1 in it. So he asked you to write a program to help him find several positive numbers, without 0 and 1, and the sum of which is the number he wants.

Input

The first line of the input contains an integer t ($1 \le t \le 1\,000$), which is the case number.

Then follows t test cases. Each test case contains a line of a positive number n ($10 \le n \le 10^{100}$). n does not have leading zeros, and it's guaranteed that at least one digit of n is 0 or 1.

Output

For each test case, output in a line k: how many numbers follows; and in the second line k space-separated positive integers $a_1, a_2, \ldots, a_k$ ($2 \le a_i \le n$ and $\sum_{i=1}^{k} a_i = n$). No 0 or 1 should appear in a_i.

k is required to be **as small as possible**. If there are still multiple solutions, any one is acceptable.

Examples

standard input	standard output
3 911 19 300	2 42 869 3 6 7 6 2 258 42

Problem L. Liwa River

Input file: `standard input`
Output file: `standard output`

Liwa River is the main river passing through East China Normal University (ECNU). With willow overhanging the rippling water, and green trees and abundant blossoms lining the river bank, the river winds its way through the campus, making itself one of the most breathtaking scenes in the University. Song Lin, a famous Chinese modern poet and a former teacher at ECNU, wrote in a letter to his friend that "If there should exist heaven, then it must undoubtedly be the Liwa River."

In the early 1900s, the place where the Liwa River is located was then part of the suburb forming the Shanghai concession for cross-border road construction and was probably a part of the Jiangnan Water Networks. It is said that the place was opened up as a commercial waterside leisure resort by some Russian immigrants and mainly served foreign nationals, Chinese middle class and the bourgeoisie.

The leisure resort was called "Rio Rita", which is of Spanish origin. The owner named it after an American musical from 1929 named Rio Rita, which was a love story about a Mexican girl Rita who lived near the Rio Grande, a river along the Mexican border. With little knowledge of the Spanish language and the film, some Chinese misunderstood "Rio" and transliterated it into "Liwa" which actually means "river".

After the leisure resort closed, Great China University was built in that district, and later incorporated into ECNU. From then on, Liwa River became synonymous with ECNU.

This might be the first time you have ever been here, or second, or even third; I'm very sure after this very contest, you will be interested in walking alongside the Liwa River, feeling the spring in the air. Meanwhile, it can be an unforgettable experience to sit on the bench by the river and make friends with all the talented programmers ("Da Lao") from universities or high schools in Shanghai, and other cities, and provinces.

You immediately realize that for all this to happen, the contest staff has to ship some benches here beforehand, so that every group of friends will have a place to seat. Specifically, people will sign up and come in groups of 1, 2, 3, 4, 5 and 6. The only kind of bench is a bench available is one that will hold six people each. A group of friends does not want to sit on two or more separate benches, as this raises unhappiness. Your task is to find the least number of benches needed to hold all the people and, meanwhile, no one is unhappy.

You are just given 5 hours to solve this problem, before it gets too late and everyone goes home.

Input

The first line contains t ($1 \leq t \leq 1\,000$), the number of test cases follows.

Each test case is a line with 6 space-separated non-negative integers, the number of groups with 1 person, 2 people, 3 people, 4, 5, and 6, respectively. The sum of people in the groups in each test case is greater than 0 and does not exceed 10 000, as it's not likely that our contest area will hold more than 10 thousand people.

Output

For each test case, output one line of an integer: the minimum number of benches needed.

Examples

standard input	standard output
6 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1	1 1 1 1 1 1
19 6 9 5 8 3 2 2 2 5 1 3 8 1 2 0 4 0 0 4 1 5 5 2 8 8 6 6 0 0 0 4 8 8 6 0 3 2 2 2 9 5 2 8 1 7 6 3 10 8 9 7 10 6 6 7 3 8 5 2 1 3 5 5 8 6 10 9 2 6 3 9 5 10 8 5 2 0 5 3 7 1 1 4 4 0 9 2 0 5 8 5 5 1 10 6 2 1 5 5 10 3 5 2 5 7 0 1 9 5 3 9 1 4 5	17 15 4 18 7 15 17 23 26 13 27 20 14 12 17 19 21 16 16