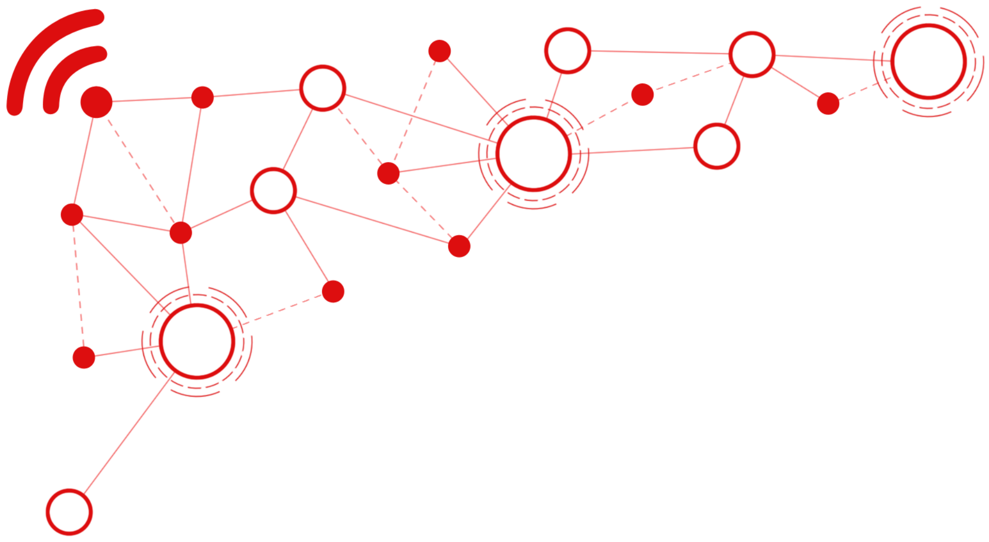
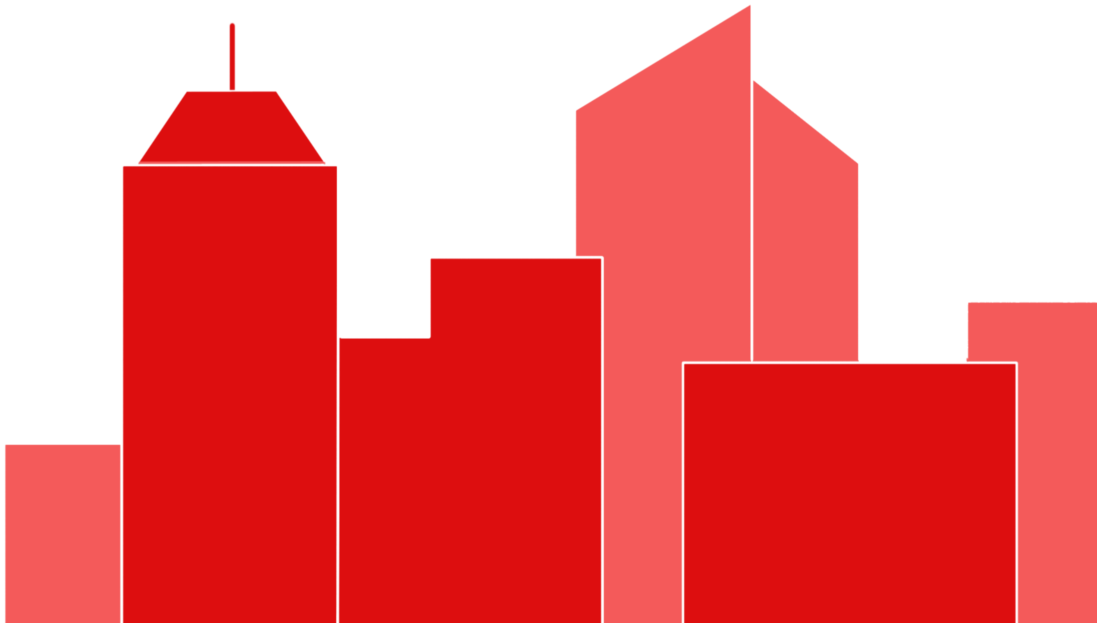


National Olympiad in Informatics

Inhouse Round



Contents

Notes

- Many problems have large input file sizes, so use fast I/O. In C++, use:
  - `ios_base::sync_with_stdio(false);` and
  - `cin.tie(nullptr);`
- On interactive problems, make sure to **flush** your output stream after printing. In C++, use `fflush(stdout);` or `cout << endl;`
- Make sure you have PyPy 3.9 in your system, and the command `pypy3.9` is working in the terminal.
- The judge is sometimes somewhat *strict*. Please terminate each line with the `'\n'` character, and please don't print trailing whitespace in any line.
- Good luck and enjoy the contest!

Problem A
Shabash!

We hope you enjoyed the 2023 NOI.PH Gaussian Prime shirts! Kevin needs your help in making the next few shirts, so maybe we should learn a bit more about what Gaussian primes are. For instance, did you know that Gaussian primes were already known to Middle-Eastern scholars during the Golden Age of Islam?[citation needed]

Complex numbers are often written in the form $a + bi$, where i is the so-called *imaginary unit*. If a and b are both integers, then we call this complex number a *Gaussian integer*. Gaussian integers are interesting because they retain many familiar properties from the normal integers—for example, factorization into "prime factors" is unique, just like with regular integers.

However, some numbers that are prime in the integers are no longer prime in the Gaussian integers. For example, 5 is prime in the integers because it cannot be broken down into smaller factors, but is not a so-called *Gaussian* prime because it can be written as $5 = (2 + i)(2 - i)$. From now on, whenever we say "prime" by itself, we are talking with respect to the integers, and will explicitly say "Gaussian prime" when working with respect to the Gaussian integers. So, 5 is prime (but not a Gaussian prime).

We can sweep most of the technical jargon under the rug for now, and instead give you a simple criterion for testing if a Gaussian integer is a Gaussian prime, discovered by Arabic mathematician Muhammad ibn al-Fermat.[citation needed] A Gaussian integer $a + bi$ is a Gaussian prime if and only if one of the following holds:

- $a^2 + b^2$ equals some prime p, or
- $a^2 + b^2$ equals the *square* of some prime p, where this p is of the form $4k + 3$ for some integer k.

Muhammad ibn al-Fermat had a lot of space in the margins of his treatise, where he also helpfully included the following extra fact that's true for Gaussian primes $a + bi$:

- If $a^2 + b^2$ equals a prime p, then either $p = 2$ or p is of the form $4k + 1$ for some integer k.
- If $a^2 + b^2$ equals the square of a prime p, then it must be true that $a = 0$ or $b = 0$.

For example, we can show that $1 + 2i$ and $2 - i$ are Gaussian primes, because $1^2 + 2^2 = 5$; here, 5 is a prime number, and indeed $5 = 4 \cdot 1 + 1$. We can also show that -7 and $-7i$ are Gaussian primes because $7^2 + 0^2 = 7^2$; here, $7 = 4 \cdot 1 + 3$ is prime, and indeed one of a or b was 0, in each case.

In addition (wow this is a really large margin), al-Fermat reminds us of the so-called "al-Fermat's Theorem on Sums of Two Squares" which he expects to be useful:

- Given a prime p, there exist positive integers a and b such that $a^2 + b^2 = p$ if and only if $p = 2$ or p can be written of the form $p = 4k + 1$, for some integer k. Furthermore, the pair (a, b) is unique (up to ordering), and $a = b$ if and only if $p = 2$.

Great! That's actually all the math background you need for this problem—which, if you've read all this carefully, you should realize is now just a matter of implementation. You will be given a list of positive integers $\ell_1, \ell_2, \ell_3, \ldots, \ell_n$. For each ℓ_k, count the number of different Gaussian primes $a+bi$ such that $a^2+b^2 \leq \ell_k$.

Oh, one last thing... Let's try to replicate how a Muslim scholar from that time would have searched for Gaussian primes. Rather than a modern library like C++'s or Python's standard libraries, we're going to use a more classic library—The Grand Library of Baghdad! For these halls once held the now-forgotten knowledge of the first computer language in the world, Shabash![citation needed]

The Shabash Programming Language

Shabash is an ancient scripting language with a limited set of instructions. A Shabash program is a sequence of lines, where each line is of the form:

- *location* `+=` *value*
- *location* `-=` *value*
- `echo` *value*
- `read` *location*
- `exit`
- *LABEL*`:`
- `if [` *value* `-gt 0 ]; then goto` *LABEL*`; fi`
- `if [` *value* `-gt 0 ]; then flip; fi`

The meaning of *location*, *value* and *LABEL* will be explained later.

Blank lines are ignored. The character # denotes the start of a *comment*; once it's found, the rest of the line is ignored.

You can download its interpreter, `shabash`, from CMS. You can also download the complete user manual `shabash_manual.pdf` as well as some sample Shabash programs (files with ending `.shsh`). Read the comments on the programs to learn what they do.

Most whitespace is insignificant, so you can also write "*location*`+=`*value*" for example. Leading and trailing whitespace is also insignificant, so for example, you can also indent some lines to help with formatting.

Memory

Shabash has variables. Variable names begin with the `$` character. Specifically, a variable is written as `$`*varname* where *varname* is any nonempty string of alphanumeric or underscore characters. All variables contain 32-bit signed integers. Variables don't need to be declared; they all implicitly exist and initially contain the value 0. All variables are global.

In addition, there's a fixed global array of 32-bit signed integers called `A`. It has a length of 1048576 and its elements are indexed by the integers 0 to 1048575. You can access the element at index *value* using `A[`*value*`]`. (Note that "`A`" doesn't begin with the `$` character.) Indices are implicitly reduced mod 1048576—even negative indices—so in particular, there are no out-of-bounds errors. The whole array is initialized to 0.

Values and Locations

We can now describe *value* and *location* expressions as follows:

- A "*value*" represents a 32-bit integer value. It is either a 32-bit integer literal or a *location*.
- A "*location*" represents a memory location containing a 32-bit integer. It is either a variable name `$`*varname* or an expression of the form `A[`*value*`]`.

In particular, you can nest array accesses, e.g., `A[A[0]]` or `A[A[A[$i]]]`. However, you can only nest up to 8 times.

We can now explain the meanings of the first four lines:

- "*location* `+=` *value*" increases the value at *location* by *value*.
- "*location* `-=` *value*" decreases the value at *location* by *value*.
- "`echo` *value*" prints the 32-bit integer *value* on a line by itself.
- "`read` *location*" reads the next 32-bit integer from input and stores it at *location* (overwriting its previous contents).

There are other versions of an `echo` or `read` line that are not needed to solve the problem but may help in debugging. They are explained in the attached Shabash manual.

Control Flow

Shabash has two ways to control the program's flow.

When a Shabash program starts, the lines are run from top to bottom in sequence.

You can jump to another part of the program using an **if-goto line**. An if-goto line is of the form "`if [ `*value*` -gt 0 ]; then goto `*LABEL*`; fi`" where *LABEL* denotes a label and is a string of *uppercase* letters, digits, or underscores that doesn't begin with a digit. When the program reaches an if-goto line, the following happens:

- If the 32-bit integer *value* is greater than 0, then the program jumps to the location denoted by the *LABEL*`:` line and proceeds from there.
- Otherwise, the program simply proceeds after the if-goto line.

Note that you cannot change the "`-gt 0`" part.

Note that a "*LABEL*`:`" must be on its own line. There must be at most one label line for each unique label name, and each label name that appears in an if-goto line must have its corresponding label line exist.

You can also change the control flow's direction using an **if-flip line**. An if-flip line is of the form "`if [ `*value*` -gt 0 ]; then flip; fi`". When the program reaches an if-flip line, the following happens:

- If *value* is greater than 0, the control flow's direction switches from top-to-bottom to bottom-to-top, or vice versa! When the control flow's direction is bottom-to-top, the lines are run in reverse order.
- Otherwise, the control flow doesn't switch directions and the program just proceeds.

As before, you cannot change the "`-gt 0`" part.

Finally, the program exits when it encounters an "`exit`" line.

Note that the program lines *wrap around*, so the next line after the last line is the first line (and vice versa when running bottom-to-top). The only way to exit is with an `exit` line.

Example Programs

The following Shabash program reads an integer input and outputs its absolute value:

```
# Reads an integer and prints its absolute value
read $input_value
if [ $input_value -gt 0 ]; then goto POSITIVE_CASE; fi
NEGATIVE_CASE:
$result -= $input_value
if [ 1 -gt 0 ]; then goto PRINT_RESULT; fi
$result += $input_value
POSITIVE_CASE:
if [ 1 -gt 0 ]; then flip; fi  # always flip!
PRINT_RESULT:
echo $result
exit
```

To run this program, first save it onto a file, say `absval.shsh`, then enter this into the terminal (the Shabash interpreter must be in the current folder):

```
./shabash absval.shsh
```

Recall that the program wraps around, so if you remove the `exit` line, the program enters an infinite loop.

The following program reads an integer input between -2 and 2 and prints its square:

```
# Prints the squares of some input values
A[-2] += 4
A[-1] += 1
A[0] += 0
A[1] += 1
A[2] += 4
read $x
echo A[$x]
exit
```

Recall that the array `A` is initialized at 0. Thus, on input "5", the program outputs "0".

Note that `A[-2]` is the same as `A[1048574]` because indices are implicitly reduced mod 1048576. In particular, on input "1048574", the program outputs "4".

Interpreter

The interpreter of Shabash is called `shabash`. Download it from CMS and then enter

```
chmod +x shabash
```

to make it executable.

To run a program saved in a file called “`prog.shsh`”, you enter this into the terminal:

```
./shabash prog.shsh
```

You can run it this way to get more details (such as the number of operations performed):

```
./shabash prog.shsh -v
```

Run the following for more information on other options:

```
./shabash --help
```

In case of technical problems, please ask for help through CMS.

Syntax Highlighting

To aid with reading and writing Shabash programs, I recommend finding an option in your code editor to highlight it as if it were a Bash script.

Input Format

Your C++ program will only receive two lines, each containing an integer. The first line contains s, the subtask number (1, 2 or 3). The second line contains n (as described in the problem statement).

Your C++ program will *not* receive the numbers $\ell_1, \ell_2, \ldots, \ell_n$. The Shabash program will receive them. Attempting to read past the first two lines may result in an *Idleness Limit Exceeded* verdict.

Output Format

Your C++ program must output a valid Shabash program with at most 1500 lines. The total size of the program must be at most 25000 bytes.

You may find C++'s “raw string literals” feature useful. Here's some sample code that uses raw string literals to print a simple Shabash program:

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
cout << R"RAW(# Shabash program follows:
read $x
echo $x
exit
)RAW";}
```

Constraints and Subtasks

For all subtasks
$s \in \{1, 2, 3\}$

Subtask	Points	Constraints
1	**0–12**	$1 \le n \le 175$ $1 \le \ell_k \le 1750$
2	**0–44**	$1 \le n \le 3500$ $1 \le \ell_k \le 35000$
3	**0–44**	$1 \le n \le 14000$ $1 \le \ell_k \le 140000$

Scoring

The judge will run your Shabash program with $n+1$ input lines containing $n, \ell_1, \ell_2, \dots, \ell_n$, respectively. Reading input beyond this results in an error. To be judged as correct, the Shabash program must `echo` a total of n integers, the kth of which must be the number of Gaussian primes $a+bi$ such that $a^2+b^2 \le \ell_k$. The Shabash program must exit within at most 10^8 operations. Each nonempty, non-label, non-exit line is counted as one operation every time it is encountered.

If your Shabash program is invalid, or if it exceeds 1500 lines or 25000 bytes, or if it runs for more than 10^8 operations, or if it doesn't `echo` exactly n times with the correct answers and then `exit` properly, then you get a score of 0 for that subtask.

Furthermore, Sultan Salah al-Dijkstra has decreed that "`goto` is evil, and no self-respecting programmer should ever use it." Thus, your Shabash program (assuming it is correct) will get more points **the fewer if-goto lines you use in it**. Let g be the maximum number of if-goto lines across all your Shabash programs for a given subtask. Then your score is computed as follows:

For subtask 1:

$$\text{score} = \begin{cases} 0 & \text{if } g > 64, \\ 12 & \text{if } g \le 64. \end{cases}$$

For subtask 2:

$$\text{score} = \begin{cases} 0 & \text{if } g > 42, \\ 7 & \text{if } 2 \le g \le 42, \\ 37 & \text{if } g = 1, \\ 44 & \text{if } g = 0. \end{cases}$$

For subtask 3:

$$\text{score} = \begin{cases} 0 & \text{if } g > 42, \\ 8 & \text{if } 28 < g \leq 42, \\ 15 - g/4 & \text{if } 12 \leq g \leq 28, \\ 36 - 2g & \text{if } 4 \leq g \leq 12, \\ 33 & \text{if } g = 3, \\ 38 & \text{if } g = 2, \\ 42 & \text{if } g = 1, \\ 44 & \text{if } g = 0. \end{cases}$$

Sample I/O

The following is some sample I/O for your Shabash program:

Input	Output
3 8 10 9	12 16 16

If this test case were on subtask 1, then your C++ submission will only receive:

Input to C++ Program
1 3

Explanation

The 16 Gaussian primes $a + bi$ such that $a^2 + b^2 \leq 9$ are as follows:

$$\begin{array}{rrrr} 1+i & 1-i & -1+i & -1-i \\ 2+i & 2-i & -2+i & -2-i \\ 1+2i & 1-2i & -1+2i & -1-2i \\ 3 & 3i & -3 & -3i \end{array}$$

Problem B
Residential Evil 2

Welcome back to Umbrella Residences, Lechon Gennady! You're here just in time... Racoon City is collapsing!

Again, Umbrella Residences has n **houses** labeled 1 to n, and m two-way **passages** that have a house on each end. Each passage also has a single *key* to a condominium in a more much affluent part of the city; each condominium (and thus its corresponding key) is assigned some positive integer value that reflects its value. You cannot travel between houses except by using these passages.

All gates have been unlocked, so you are free to travel through any of the passages connected to the house you're currently in. This is very convenient, because you are currently being chased by a horde of bloodthirsty businessmen who want to buy your house and transform it into an Airbnb. When you travel through a passage, you may choose to do the following:

- If you're not holding a key, you can pick up the key along this passage.
- If you're already holding a key, you can either throw it away and then pickup the key along this passage, or you can hold on to it and ignore the one along the passage.

Regardless of your decision, you must beware of the businessmen chasing you and gentrifying every inch of land they touch; you must exit the passage through its other endpoint, and **you may no longer use this passage until the game is reset**. (On the other hand, you may revisit houses.)

Also, each house contains a key—the key to that house. You're free to ignore it or pick it up (throwing away your current key, if you're holding one); just note that these keys have value 0, because their location location location isn't that great.

The layout of Umbrella Residences and the values of the keys remain fixed. Also, whenever you reset and start a new game, all passages are restored and become usable again. However, the following two values change in every instance:

- The starting house s
- The ending house e

You must answer q different queries, each one with its own value of s and e. For each one, find the maximum-value key that you can end the game with (starting at s, and ending at e, and with the passages collapsing behind you) assuming you take an optimal route.

Input Format

The first line of input contains the integer n. The second line contains the integer m.

This is followed by m lines. The ith of these lines contains three space-separated integers u_i, v_i, and w_i, meaning that the ith passage connects houses u_i and v_i, and the key along it has value w_i.

The next line contains the integer q.

This is followed by q lines, each containing two space-separated integers s and e, the starting and ending houses for this query.

Output Format

Output q lines, where the jth of these contains the answer to the jth query.

Constraints and Subtasks

For all subtasks

$1 \le n \le 2 \times 10^5$
$0 \le m \le 3 \times 10^5$
$1 \le q \le 2 \times 10^5$
$1 \le u_i, v_i \le n$ in each passage
$1 \le w_i \le 10^6$ in each passage
$1 \le s, e \le n$ in each query
It is possible to reach any house from any other house just by traveling through the passages.

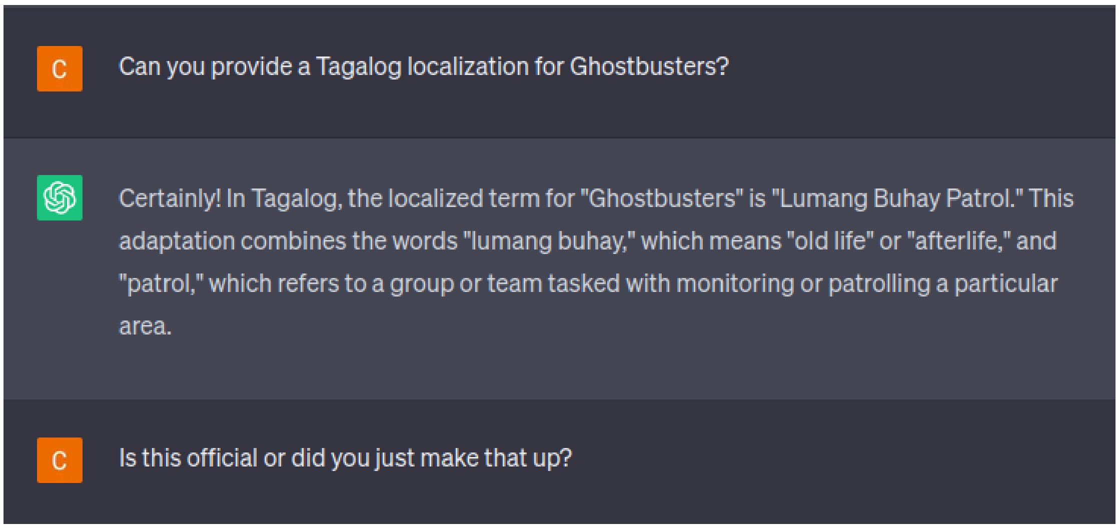
Subtask	Points	Constraints
1	**9**	$m \leq 8$, $q \leq 100$
2	**6**	$m \leq 14$, $q \leq 100$
3	**8**	$m \leq 18$, $q \leq 100$
4	**9**	There is an i with $(u_i, v_i) = (x, x+1)$ for $x = 1 \ldots n-1$. $m < n$
5	**14**	There is an i with $(u_i, v_i) = (x, x+1)$ for $x = 1 \ldots n-1$.
6	**8**	$m < n$
7	**11**	$m \leq 5000$
8	**14**	$n \leq 5000$
9	**21**	No further constraints.

Sample I/O

Input	Output

8
9
1 6 25
5 3 50
8 7 75
6 8 100
6 5 200
3 2 100
2 6 500
8 4 100
2 5 1000
3
4 7
8 1
7 6
	100
	1000
	1000

Problem C
Lumang Buhay Patrol



C Can you provide a Tagalog localization for Ghostbusters?

Certainly! In Tagalog, the localized term for "Ghostbusters" is "Lumang Buhay Patrol." This adaptation combines the words "lumang buhay," which means "old life" or "afterlife," and "patrol," which refers to a group or team tasked with monitoring or patrolling a particular area.

C Is this official or did you just make that up?

The Lumang Buhay Patrol needs help with designing a new circuit board that will control the latest model of their Proton Packs. But they don't have the money to hire a proper Electrical Engineer... or even a student in Electrical Engineering... so they've decided to hire you instead!

Locations on the circuit board can be modeled using a pair of coordinates (x, y), similar to the Cartesian plane. There are n nodes on the circuit board, such that the ith of them can be modeled as a point located at (x_i, y_i). We must solder wires onto the circuit board, each one directly connecting two different nodes, such that each node is connected to every other node by some sequence of wires. However, the Lumang Buhay Patrol have the following extra restrictions:

- A wire connecting two nodes must be a **straight line segment** with a node at each endpoint.
- Remember the Lumang Buhay Patrol's motto, "Huwag magtagpo ang mga daloy!" Any two wires **must not intersect or overlap** anywhere except for their endpoints.
- Removing any of the wires causes some pair of nodes to be disconnected.

Any configuration which satisfies all these conditions is called *valid*. The *cost* of a valid configuration is equal to

$$(\ell_1^2 + \ell_2^2 + \ldots + \ell_m^2) \bmod 998244353,$$

where m is the number of wires, and ℓ_i is the length of the ith wire. Note that the length ℓ of the wire connecting locations (x, y) and (x', y') is

$$\ell := \sqrt{(x - x')^2 + (y - y')^2},$$

so ℓ^2 is always an integer if (x, y) and (x', y') are lattice points.

Find the minimum possible cost in order to solder a valid configuration of wires onto the Lumang Buhay Patrol's circuit board.

Input Format

The first line of input contains t, the number of test cases.

The first line of each test case contains an integer n. After that, n lines follow, the ith of which contains two space-separated integers x_i and y_i.

Output Format

For each test case, output a single line containing the answer to that test case.

Constraints and Subtasks

For all subtasks
$1 \le t \le 6$ $1 \le x_i, y_i \le 10^8$ The n pairs (x_i, y_i) are distinct.

Subtask	Points	Constraints
1	**38**	$2 \le n \le 7$
1	**9**	$2 \le n \le 8$
1	**19**	$2 \le n \le 9$
1	**34**	$2 \le n \le 10$

Sample I/O

Input	Output

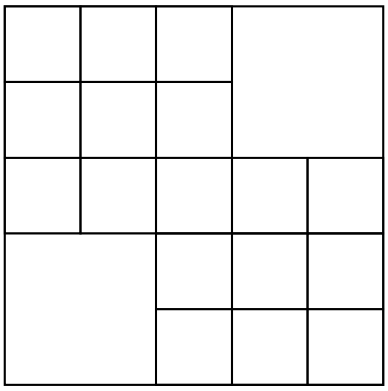
3
4
1 1
1 3
3 1
3 3
4
1 1
1 30000
30000 1
30000 30000
3
1 1
2 2
3 3
	12
	605026945
	4

Problem D
I Really Hate "IQ Tests"

If you judge a fish's IQ by its ability to solve clichéd and overused "brain teasers", then it will live its whole life thinking that being intelligent is stupid.

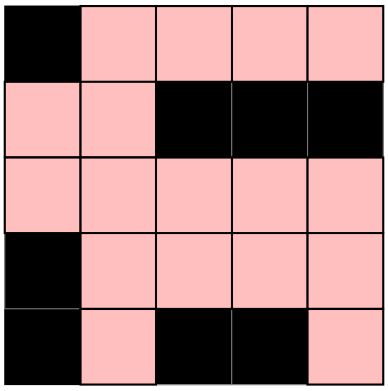
Albert Einstein

How many rectangles are in this picture?



Too easy? Alright, fine, if you're such an Einstein, why don't you solve this version? How many **pink** rectangles are there in this picture? Formally, count the number of rectangles that satisfy the following:

- Each of its sides rests entirely along the gridlines;
- All squares enclosed within it are color pink.



Still too easy, you say? Then you leave us with no choice. If you think these problems are so easy, then **why don't you make one, yourself?** Given a positive integer k, color in a square grid such that the answer to the question "How many pink rectangles are there in [your constructed grid]?" is exactly equal to k. You can choose any dimensions $n \times n$ that you want for the square grid, although keep in mind that the smaller the area, the higher your score will be.

Input Format

The first line of input contains t, the number of test cases.

Each test case consists of a single line containing the integer k.

Output Format

For each test case, first output a line containing a positive integer n. Then, output n lines, each containing a string of length n consisting of the characters `.` or `#`, denoting a black or a pink square, respectively.

Constraints and Subtasks

For all subtasks
Each k appears at most once across all *test files*. The ks are increasing in each test file.

Subtask	Points	Constraints
1	**0–25**	$t = 1500$ $1 \le k \le 1500$ There is exactly 1 test file under this subtask.
2	**0–25**	$t = 2500$ $10^4 \le k \le 10^5$ There are exactly 10 test files under this subtask.
3	**0–25**	$t = 3000$ $5 \cdot 10^5 \le k \le 10^6$ There are exactly 10 test files under this subtask.
4	**0–25**	$t = 2000$ $10^7 \le k \le 2 \cdot 10^7$ There are exactly 10 test files under this subtask.

Scoring

There is a scoring formula for each subtask. Each subtask is worth up to 25 points.

First, each test file is scored individually. For each test file, let A be the total area (sum of n^2) across all test cases, and let $A_{\min}$ be the minimum value of A across the problem setter's and testers' solution for this test file. Then your score for the test file is computed as

$$(\text{score in file}) = \begin{cases} 0 & \text{if } A > 2 \cdot 10^7; \\ 7 + \dfrac{18}{\sqrt[4]{1 + \frac{A - A_{\text{min}}}{6t}}} & \text{if } A_{\text{min}} < A \leq 2 \cdot 10^7; \\ 25 & \text{if } A \leq A_{min}, \end{cases}$$

(where t is the number of test cases in the file) rounded *down* to 3 decimal places.

Your score for each subtask is then the *minimum* among all your scores across that subtask's test files. Finally, your overall score is the *sum* of your scores across all four subtasks.

Test Data

As specified in the Constraints and Subtasks section, each subtask has a fixed number of test files.

On top of this, all test files are generated *randomly and uniformly*. Specifically, for a subtask with f test files, the following procedure was performed to generate them:

1. Choose $f \cdot t$ distinct values of k from the allowed range randomly and uniformly.
2. Shuffle these values and split them into f test files, each containing t values of k.
3. Finally, sort the ks in each test file in increasing order.

Sample I/O

Input	Output
2 42 63	5 .#..# #.##. #.### .#### .#.#. 5 .#### ##... ##### .#### .#..#

Explanation

Note that the sample test case doesn't satisfy the constraints for t for any subtask, so it will not appear as an input file.

The sample output for the second test case is illustrated by the image in the problem statement.