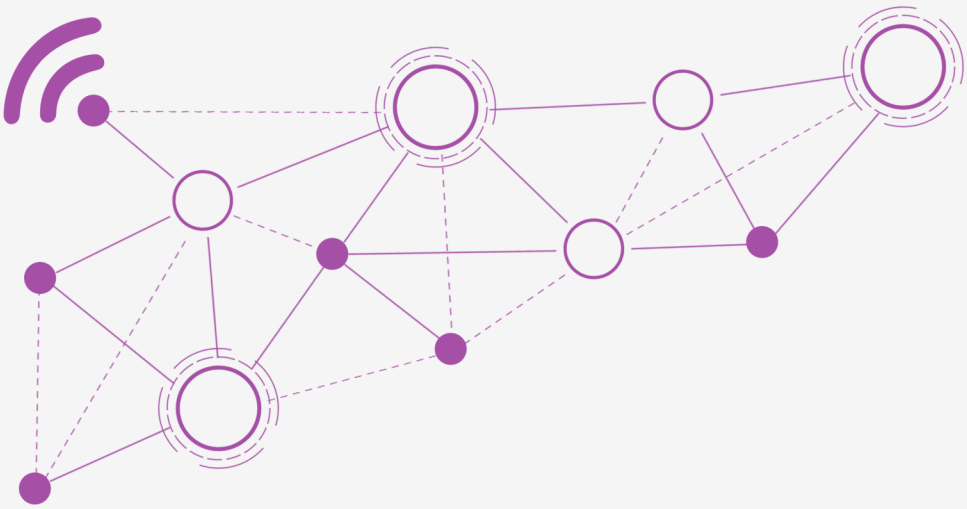


National Olympiad in Informatics

Finals Round 2

Contents

Notes

- Many problems have large input file sizes, so use fast I/O. For example:
  - In Java, use `BufferedReader` and `PrintWriter`.
  - In C/C++, use `scanf` and `printf`.
- Good luck and enjoy the problems!

Important! Read the following:

- Your solution will be checked by running it against one or more (usually several) hidden test cases. You will not have access to these cases, but a correct solution is expected to handle them correctly.
- The output checker is **strict**. Follow these guidelines strictly:
  - It is **space sensitive**. Do not output extra leading or trailing spaces. Do not output extra blank lines unless explicitly stated.
  - It is **case sensitive**. So, for example, if the problem asks for the output in lowercase, follow it.
  - Do not print any tabs. (No tabs will be required in the output.)
  - Do not output anything else aside from what's asked for in the Output section. So, do not print things like "Please enter t".

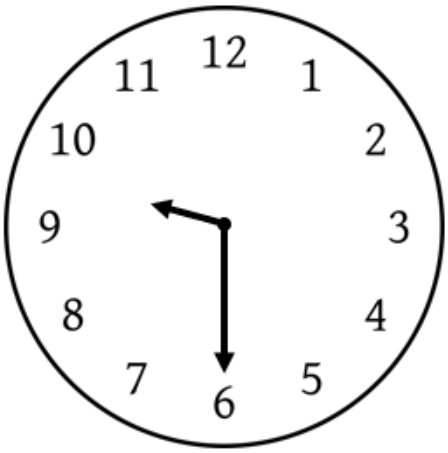
  Not following the output format strictly and exactly will likely result in the verdict "*Output isn't correct*".
- Do not read from, or write to, a file. You must read from the standard input and write to the standard output.
- For Java, do not add a package line at the top of your code. Otherwise, your submission will receive the verdict "*Execution failed because the return code was nonzero*" or "*Compilation failed*".
- Only include one file when submitting: the source code (.cpp, .java, .py, etc.) and nothing else.
- Only use letters, digits and underscores in your filename. Do not use spaces or other special symbols.
- Many problems have large input file sizes, so use fast I/O. For example:
  - In Java, use `BufferedReader` and `PrintWriter`.
  - In C/C++, use `scanf` and `printf`.

  We recommend learning and using these functions during the Practice Session.
- Good luck and enjoy the contest!

Problem A

Eddie the Nerdie

Time Limit: 2 seconds



Eddie is a nerdie. Eddie is bored. Eddie is so bored that he stares at the clock as its hands rotate. Eddie is disgusted by the second hand, so he only looks at the hour hand and the minute hand. He makes the following observations about his clock.

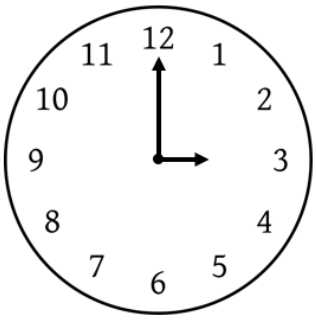
The hands do not "snap" at exact values, i.e., Eddie's clock is the type where each hand moves clockwise continuously with a constant rotational velocity.

The clock is a 12-hour clock, meaning the hour hand makes one full rotation around the clock every 12 hours.

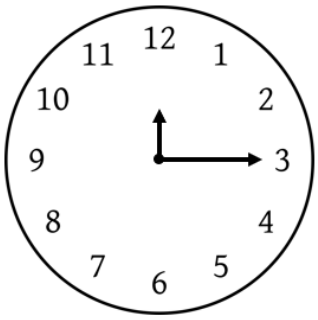
The clock is a 60-minute-per-hour clock, meaning the minute hand makes one full rotation around the clock every 60 minutes.

The clock is labeled with the integers from 1 to 12, in order, evenly spaced, and such that 12 is at the northmost part of the clock. At noontime, the hour hand and the minute hand are at exactly the same place, pointing north.

Eddie later noted that not all configurations of the clock hand and the minute hand are "valid". For instance, you can convince yourself that whenever the hour hand is pointing at any of the integer labels, the minute hand *must* be pointing at 12. In the image below, the clock on the left shows a "valid" position, while the clock on the right shows an "invalid" position.



03:00
valid



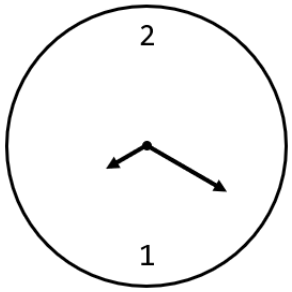
??:??
invalid

Eddie wonders: Is there any time `HH:MM` so that when he interchanges the hour hand and the minute hand, the resulting position is also valid? Eddie thinks about it and can only think of one solution: `12:00`. Perhaps there are solutions where the time is not exact to the minute? Actually, while we're at it, let's be even more general.

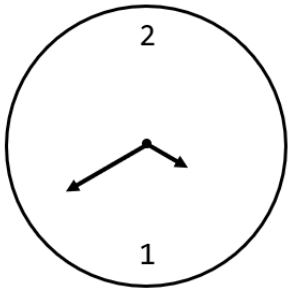
Eddie disassembles the clock and messes with its settings. Now, it is an h-hour and m-minute-per-hour clock, meaning the hour hand makes one full rotation every h "hours", and the minute hand makes one full rotation every m "minutes". He sets both the hour and minute hand to be pointing north (in what *used to be* the `12:00` position), and lets the clock hands run with their new angular velocities.

With this new system, let's explain how the time format `HH:MM` works, for **integers** $1 \leq \texttt{HH} \leq h$ and $0 \leq \texttt{MM} < m$. The clock is relabeled with the integers from 1 to h, in order, evenly spaced, and such that h is at the northmost part of the clock. Both clock hands begin at the h mark at time h`:00`. The time `HH:MM` now corresponds to the *valid* position where the minute hand has made $\frac{\texttt{MM}}{m}$ of a full rotation around the clock, and where the hour hand is between the marks `HH` and $\texttt{HH} + 1$ (or between h and 1, if $\texttt{HH} = h$).

For example, in an $h = 2$ hour and $m = 60$ minute-per-hour clock, the times `1:20` and `2:40` correspond to the positions below.



1:20
valid



2:40
valid

Note that in this case, interchanging the clock hands in the `2:40` position *actually also corresponds* to a valid position, the one corresponding to `1:20`. Eddie thinks there are more, so he asks his friend Luke the Duke to solve this more general problem. Given an h-hour, m-minute-per-hour clock, is there any time *other than* h`:00` so that interchanging the hour and minute hands also results in a valid position? Eddie is interested in both *counting* the number of solutions, and finding the *next* time after h`:00` that has this property.

Input Format

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

Then, t lines follow, each describing a test case. Each test case contains two space-separated integers h and m, meaning this test case has an h-hour and m-minute-per-hour clock.

Note that h is not necessarily smaller than m. It can be equal or even larger.

Output Format

For each test case, output a single line containing an integer c and a string s separated by a space.

The integer c is the number of solutions (not counting h`:00`).

The string s describes the earliest possible time (after h`:00`) which is a solution to this problem. The string s should be:

- "`exactly` x" (without the quotes), where x is a time in the `HH:MM` format, if this earliest possible time is exactly x;
- "`between` x `and` y" (without the quotes), where x and y are **consecutive** times in the `HH:MM` format, if this earliest possible time is strictly between x and y.

Important note: Pad the time with *just enough zeroes* to be able to write all the times that can be represented with the clock. For example, for $h = 10$, $m = 1000$, the time must be written as `HH:MMM` (e.g., `02:069`), while for $h = 9$, $m = 1001$, the time must be written `H:MMMM`.

Constraints and Subtasks

For all subtasks
$1 \leq t \leq 10^4$ $2 \leq h \leq 10^5$ $2 \leq m \leq 10^5$

Subtask	Points	Constraints
1	5	$t = 1$ $h = 12$ $m = 60$
2	10	$t = 9$ $h = 2$ $2 \leq m \leq 9$
3	5	$t = 90$ $h = 2$ $11 \leq m \leq 99$
4	15	$h = 2$
5	15	It is guaranteed that every solution time (not just the earliest one after h`:00`) can be represented exactly in the `HH:MM` format. $t = 90$ $h \leq 12$ $11 \leq m \leq 99$
6	20	$h \leq 12$
7	15	$h \leq 1000$
8	15	No additional constraints.

Sample I/O

Input	Output
2 2 60 2 200	2 exactly 2:40 2 between 2:133 and 2:134

Problem B

Sussus Amongus

Time Limit: 1.5 seconds

The Philippines' expedition to the moon on the gossip-powered rocketship was so successful that someone decided to make an internationally-famous game *Sa Atin*, which borrows that setting! In fact, *Sa Atin* is so popular that just saying the words *Sa Atin* automatically leaves everyone laughing uproariously for several minutes. This joke is timeless and will never *ever* feel old or dated.

Sussus Amongus is the internet's most popular *Sa Atin* streamer, along with his good friend Higa da Drip. They are very serious players! They are so serious that the game studio that created *Sa Atin* has invited them as beta testers to try out all the new features in *Sa Atin 2*.

In *Sa Atin*, the Filipinauts that are not the *manlilinlang* must play various minigames to keep the ship functioning. Sussus Amongus is frustrated with one of the new minigames introduced in *Sa Atin 2*—it makes him want to say, "Sus! Ano ba 'to!"

In the minigame, you are presented with an $r \times c$ grid. Some cells of the grid contain mini-Filipinauts, with one mini-Filipinaut per cell; all other cells are empty. However, the mini-Filipinauts still do not have any colors! The minigame-player is tasked with assigning one of k di□erent colors to each mini-Filipinaut, such that if two mini-Filipinauts are in cells that share an edge, then they $\mathbf{do\ not}$ share the same color (since that would be confusing, and thus pretty dubious).

Higa da Drip told Sussus Amongus that not only was the game easy, but there were likely *billions* of possible solutions (if not more!) Sussus Amongus wasn't convinced by this, and said that Higa da Drip's claim was quite "hins" (short for *kahina-hinala*). Can you help prove or disprove the claim by counting the number of di□erent solutions to the minigame?

Input Format

The first line of input contains two space-separated integers k and t, the number of colors and the number of grids to process. The descriptions of the t grids follow.

The first line of each grid description contains two space-separated integers r and c, the dimensions of the grid.

Then, r lines follow, each containing a string of c characters. This encodes the grid. Cells with mini-Filipinauts are represented by the asterisk (‘`*`’) character, while empty cells are represented by the dot (‘`.`’) character.

Output Format

For each test case, output a line containing a single integer, the number of ways to assign colors to the mini-Filipinauts such that if two mini-Filipinauts are in cells that share an edge, then they **do not** share the same color. Two ways are considered different if there exists a mini-Filipinaut that is colored differently between the two ways. Since this number can be quite large, output it modulo 353442899.

Constraints and Subtasks

For all subtasks

$1 \le r, c \le 143$
The sum of the rc's is at most 143.
$2 \le k \le 4$
There is at least one cell with a mini-Filipinaut in every grid.

Subtask	Points	Constraints
1	**19**	$k = 2$
2	**33**	$k \le 3$ The sum of the rc's is at most 15.
3	**12**	$k \le 3$ $c \le 8$
4	**12**	$k \le 3$
5	**12**	$c \le 8$
6	**12**	No additional constraints.

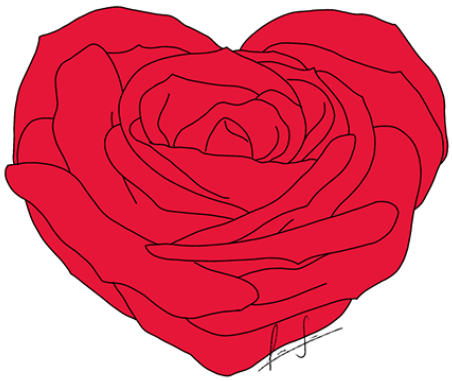
Sample I/O

Input 1	Output 1
2 2 1 1 * 2 3 .** **.	2 2

Input 2	Output 2
3 2 1 1 * 2 3 .** **.	3 24

Problem C
A Walk to Remember

Time Limit: 1 second



Romeo and Julia are in love. Crazy, stupid love, you could call it. When Romeo met Julia, it was a lovely night, and he wrote down in his notebook that he vowed to love this pretty woman forever. For Julia, she wasn't sure if this was love actually, but when comparing Romeo to all the boys she's loved before, she knew that notting would compare. Romeo and Julia have already gone on fifty dates, but Romeo plans for their fifty-first date to be extra special.

Romeo wants to construct an exhibit, showcasing all the things he loves about Julia. The exhibit shall have r rooms, and c **one-way** corridors each connecting one room to another (but he has not yet decided on r and c). The rooms are to be labeled $1, 2, 3, \ldots, r$. A one-way corridor from room a to room b allows for passage from a to b, but not necessarily from b to a. Romeo *cannot* construct a corridor that connects a room to itself, and there can only be at most one corridor going from some room a to room b. However, if he builds a corridor from room a to room b, he *is* permitted to build a corridor from room b to room a.

A *walk* is a sequence of rooms such that a corridor exists leading from each room to the next room in the sequence; in other words, Romeo and Julia could visit these rooms in that order by just passing through the corridors. Note that unlike a *path*, you are allowed to visit the same room multiple times in a walk.

Romeo first chooses some s and t, then builds the entrance in room s, and the exit in room t. He considers a walk to be a *walk to remember* if it starts at s, ends at t, and visits **exactly** n rooms; since Romeo wants to spend a lot of time with Julia, note that n will not be very small (see the constraints). Also, Romeo wants to take Julia through this exhibit multiple times, but wants their walks to be different each time. So, Romeo should construct an exhibit such that there are **exactly** m different walks to remember (note that the chosen rooms s and t should be *the same* across all walks to remember). Two walks are considered different if there exists some i such that the ith rooms in each sequence are different.

But Romeo is quite stupid right now, probably because of his aforementioned crazy, stupid love for Julia. He can handle creating the romantic content, but, given the values of n and m, do you think you can construct the layout of the exhibit for him? Also, Romeo does not have an infinite amount of money, so we he will impose restrictions on how big r and c are allowed to be. We know now, you're Romeo's only hope!

Input Format

The first line of input contains two space-separated integers n and m, the number of rooms in a walk to remember, and the number of di□erent walks to remember.

The second line of input contains two space-separated integers R and C, meaning your construction may only contain at most R rooms and at most C corridors.

Output Format

First, output a line containing four space-separated integers r, c, s, and t, which are the number of rooms, the number of one-way corridors, the room with the entrance, and the room with the exit. This must satisfy $1 \leq r \leq R$ and $1 \leq c \leq C$, as well as $1 \leq s, t \leq r$.

Then, output c lines, with each line containing two space-separated integers a and b, meaning you wish to construct a one-way corridor from room a to room b. You cannot connect a room to itself, so $a \neq b$ must hold for all lines. You also cannot construct more than one corridor from room a to room b, so each a b must appear on at most one line; you are permitted, though, to have a b on one line and b a on another.

Your output will be accepted if and only if there are exactly m walks to remember, i.e., m walks from room s to room t that visit exactly n rooms.

Constraints and Subtasks

For all subtasks
$143 \leq n \leq 143{,}143{,}143$ $1 \leq m \leq 143{,}143{,}143$ $C = 5000$

Subtask	Points	Constraints
1	**11**	$R = 1000$, $n = 143$, $m = 1$
2	**11**	$R = 1000$, $n = 143$, $m \leq 6$
3	**9**	$R = 1000$, $n = 143$, $m \leq 600$
4	**9**	$R = 1000$, $n = 143$, $m \leq 60000$
5	**10**	$R = 1000$, $n = 143$
6	**9**	$R = 200$, $m \leq 143{,}143$
7	**9**	$R = 150$, $m \leq 143{,}143$
8	**32**	$R = 199$, has partial scoring (see **Scoring** below)

Scoring

There is a partial scoring rule for Subtask 8. Your score depends on the largest r you output across all cases in that subtask. If $r_{\max}$ is this maximum, then your score for Subtask 8 is calculated as follows:

$$\text{score} = \begin{cases} 0 & \text{if } 200 \leq r_{\max} \\ 8 & \text{if } 150 < r_{\max} < 200 \\ 14 & \text{if } 120 < r_{\max} \leq 150 \\ 19 + 11 \cdot \frac{120 - r_{\max}}{120 - 32} & \text{if } 32 \leq r_{\max} \leq 120 \\ 32 & \text{if } r_{\max} < 32 \end{cases}$$

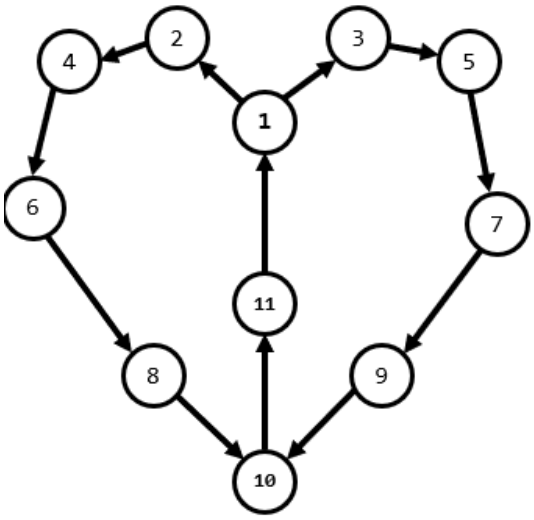
Sample I/O

Input	Output
7 2 1000 5000	11 12 1 11 1 2 2 4 4 6 6 8 8 10 1 3 3 5 5 7 7 9 9 10 10 11 11 1

Explanation

Note that this sample case will not appear in the test data because n is too small.

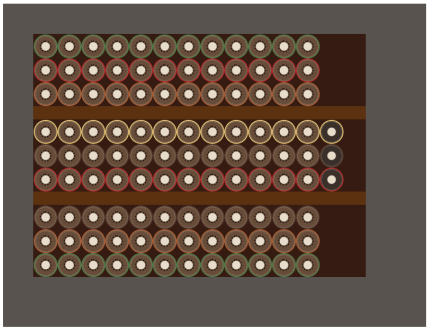
The exhibit, when drawn, has the following shape.



We can see that there are exactly 2 walks to remember.

1. $1 \to 2 \to 4 \to 6 \to 8 \to 10 \to 11$
2. $1 \to 3 \to 5 \to 7 \to 9 \to 10 \to 11$

Problem D
Pering the Platypus
Time Limit: 11 seconds



Many people recognize Pering the Platypus as a semi-aquatic, egg-laying mammal of action. He's got more than just mad skill—he's got a beaver tail and a bill. But what most do not know, is that more than just a secret agent, Pering the Platypus is actually an incredible computer scientist! For this reason, he was assigned to analyze the behavior of one of Dr. Oof's latest inventions—the *Flashback-inator*! What does it do?

Anyone zapped by the *Flashback-inator* is transported into a flashback of Dr. Oof's miserable childhood in Godelschtump. Dr. Oof's family fell on hard times, and he was forced into labor at a lawn gnome factory in order to make ends meet.

Pering the Platypus zapped himself with the device, and was brought to a corridor of rooms that extends bi-infinitely to the left and to the right. Each room has a lawn gnome, a black paint can, and a white paint can; initially, all lawn gnomes in the factory are color black.

Pering the Platypus also noticed he had $\mathbf{five}$ instruction manuals on his person, labeled A to E; initially, he is holding the one labeled A. Each manual actually only contains two instructions. If Pering is in a room with a black lawn gnome, he should perform the first instruction. Otherwise, he is in a room with a white lawn gnome, and he should perform the second instruction.

Each instruction is of the following specific form.

- First, an integer x, which is either 0 or 1. If $x = 0$, Pering must paint the lawn gnome black; if $x = 1$, he must paint the lawn gnome white. Note that this overwrites whatever its previous color was.
- Second, a symbol *dir*, which is either $\leftarrow$ or $\rightarrow$. If dir is $\leftarrow$, Pering must then move to the room to his left; if *dir* is $\rightarrow$, he instead moves to the room to his right. Note that this moving is done *after* painting the gnome.

- Third, a character *book*, which is one of A, B, C, D, E, or a special character ! (exclamation mark). If a letter, this means Pering should change his instruction manual to the one whose letter is *book*. If it's the ! character, then Pering is forcefully ejected from the flashback.

All manuals' instructions share this form, but of course the content of the instructions will differ from manual to manual. Pering must carry out the sequence of operations—painting lawn gnomes, moving between rooms, and switching between manuals—until he reaches the ! instruction and is ejected from the flashback.

Why would Dr. Oof do such a thing? Well, his original intention was to create a *Universal-inator*, which should have been able to simulate the effects of every *-inator* he has ever built, and even every *-inator* that he could possibly ever build! Unfortunately, sometimes he messes up and loads it with instructions that cause people to stay in the flashback *forever* (until someone from the outside saves them by hitting the self-destruct button).

Dr. Oof asked Pering the Platypus if he knew of a general algorithm that could tell whether a given set of instructions would leave someone trapped in the flashback forever, or if they would *eventually* end up ejected from the flashback. Pering the Platypus wasn't able to solve this, as it was an incredibly difficult task. (Though maybe you, dear reader, can think of a solution! Please tell us if you do!)

He *was* able to answer the question for some easy cases, but others were more difficult. For your task, Pering the Platypus decided to focus on just *one* set of instructions for the *Flashback-inator*, described in the following table. We still do not know if one can escape from the flashback induced by this set of instructions.

Manual	Black Gnome	White Gnome
A	$1, \leftarrow,$ C	$1, \rightarrow,$ D
B	$1, \leftarrow,$!	$0, \leftarrow,$ C
C	$1, \rightarrow,$ A	$1, \leftarrow,$ C
D	$1, \rightarrow,$ E	$0, \rightarrow,$ A
E	$1, \leftarrow,$ B	$0, \leftarrow,$ E

Your task is to write a program that computes the state of the factory after some finite k number of instructions have been carried out.

You will prove that your program performed its simulation correctly by outputting the colors of the gnomes in each room after k instructions have been carried out. Actually, that's too much to output, so we will only ask for a *hash* of that state instead, described below.

Note: For this problem, there is exactly one input file, and it is available for download:

- `pering-the-platypus-input.txt`

Input Format

The first line of input contains an integer T, the number of test cases. Each test case consists of a single line containing a single positive integer k.

Output Format

For each test case, output a single line containing either the string "PASS" (without quotes), or a single integer, the hash of the state of the factory after exactly k instructions were carried out. The hash is computed as follows.

If there are no white gnomes, then the hash is 0. Otherwise, locate the leftmost white gnome, and delete all black gnomes to its left; also, locate the rightmost white gnome, and delete all black gnomes to its right. This leaves us with a finite sequence of black and white gnomes. If we replace each black gnome with a `0` and each white gnome with a `1`, we get a bitstring, which we can interpret as a binary number. Output the value of this binary number, modulo $(10^9 + 8)^2 - 1$ (or $1{,}000{,}000{,}016{,}000{,}000{,}063$).

Constraints

- There is only one test file.
- $T = 72$
- $1 \leq k < 10^{15}$
- Pering the Platypus has ensured that for all k in the input file, someone zapped by the *Flashback-inator* would have not yet been ejected from the flashback after k instructions.

Scoring

There is a special scoring rule.

You may choose to not answer a test case by printing "PASS" (without the quotes).

If you answer any test case incorrectly, you get a score of 0. (A PASS is considered neither correct nor incorrect.)

Otherwise, if you answered A out of T cases correctly, then your score will be

$$\left\lceil 100 \left(\frac{A}{T}\right)^{\pi} \right\rceil.$$

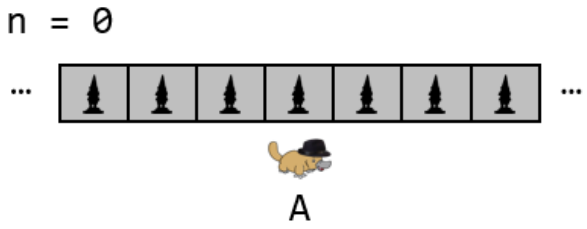
Sample I/O

Input	Output
72 1 2 3 5 8 13 21 34 ... 64 lines omitted ...	PASS PASS PASS PASS 13 PASS PASS PASS PASS ... 63 lines omitted (all PASS) ...

Explanation

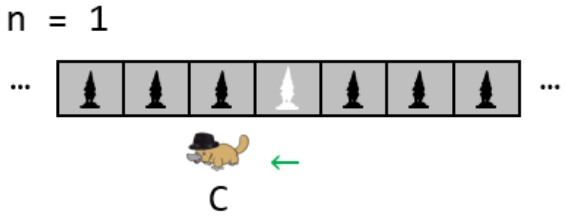
The sample output is all PASS except for the 5th test case, where $k = 8$.

Here is the initial set-up of the flashback.



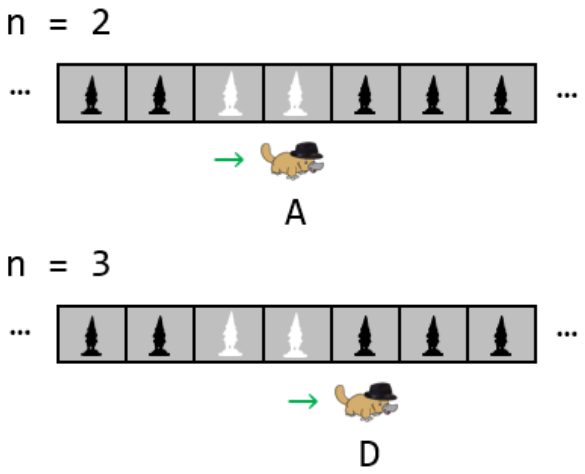
For his first instruction, Pering the Platypus is in a room with a black gnome, and using instruction manual A. So, he follows the corresponding instruction of $1, \leftarrow,$ C, which tells him to...

- First, paint the gnome white.
- Then, move to the room to the left.
- Finally, swap to instruction manual C.

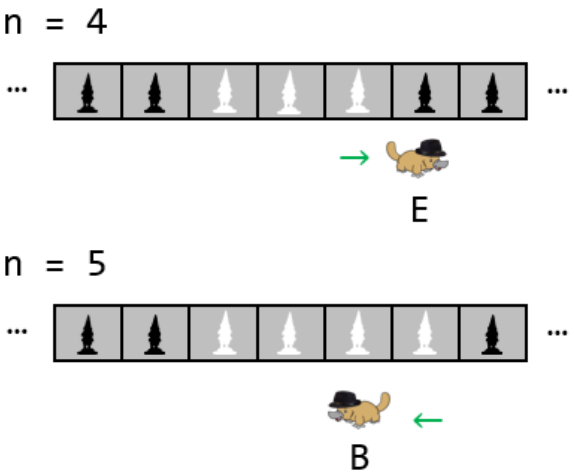


If he keeps doing this for the next few instructions, we get the following behavior.

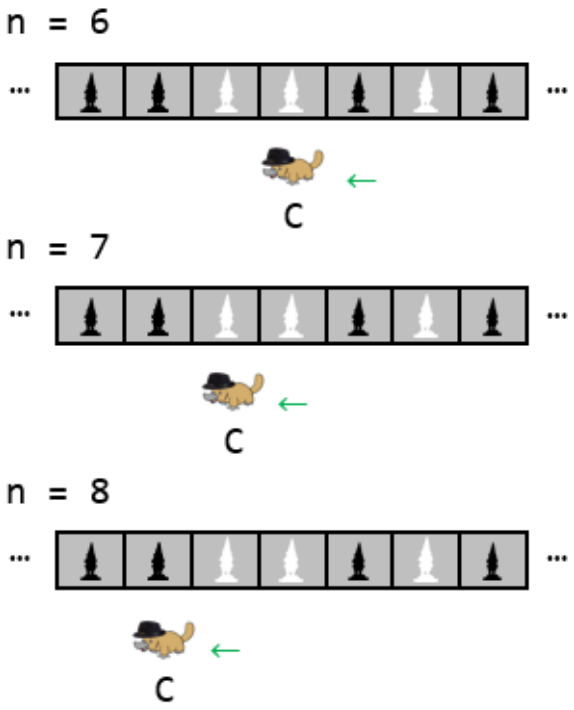
After 2 and 3 instructions...



After 4 and 5 instructions...



After 6, 7, and 8 instructions...



We see that the hash of the state after the 8th instruction is $1101_2 = 13$ modulo $(10^9 + 8)^2 - 1$, which is 13.

Since $T = 72$ and $A = 1$, the sample output gets a score of

$$\left\lceil 100 \left(\frac{A}{T}\right)^{\pi} \right\rceil = \left\lceil 100 \left(\frac{1}{72}\right)^{\pi} \right\rceil = \lceil 0.00014622\ldots \rceil = 1.$$