Task 5: `Lemon`

This is a communication task. Note that only C++ is allowed for this task.

Takina and Chisato are at a fruit convention. After a day of taking photos with their favourite fruit cosplayers, they came across a game booth.

The rules of the game are as follows:

- There are n fruits, each with a distinct label from 1 to n.
- Exactly one of the fruits is a lemon, but neither Takina nor Chisato know which one it is yet.
- Takina will be given all n fruits one by one. Her goal is to communicate the lemon's label to Chisato (who is blindfolded during this process).

Before receiving any fruits, Takina is given an array p, which represents the order in which the fruit labels will appear. $p[1]$ will be the label of the first fruit, $p[2]$ will be the label of the second fruit, and so on. Then, Takina will write down a binary string b containing only `0`s and `1`s. The string must not be longer than 5000 characters, but it can be empty. Let x denote the length of b.

Afterwards, the fruits are given to Takina one by one in the aforementioned order. Upon receiving each fruit, Takina is informed whether it is the lemon.

- If the fruit is **not** the lemon, Takina may choose whether or not to eat it. This decision must be made before receiving the next fruit and cannot be changed.
- If the fruit **is** the lemon, Takina must not eat it.

Let y denote the total number of fruits eaten by Takina.

Finally, Chisato receives the string b and the **sorted list of labels** of fruits that were uneaten. Using this information, Chisato must determine the label of the fruit which is the lemon.

Takina and Chisato have decided to play this game t times. Design a strategy for them to correctly identify the lemon while minimising both x and y.

Implementation Details

This is a communication task. Do not read from standard input or write to standard output.

You need to implement **three** procedures.

For Takina, you should implement:

```
std::string init(int subtask, int n, std::vector<int> p)
```

- `subtask`: the index of the subtask to which the test case belongs.
- n: the number of fruits.
- p: an array of length $n + 1$ where:
  - $p[0] = 0$, and
  - for each $1 \leq i \leq n$, $p[i]$ is the label of the i-th fruit that will be given to Takina.
- This procedure is called t times per test case, once at the start of each game.

This procedure should return string b, with length between 0 and 5000 inclusive, consisting of only `0`s and `1`s. If a string of invalid length or format is returned, you will receive a `Wrong Answer` verdict.

```
bool receive_fruit(int id, bool is_lemon)
```

- `id`: the label of the fruit given to Takina.
- `is_lemon`: `true` if the fruit given is the lemon, `false` otherwise.
- This procedure is called n times for each of the t games, once for each fruit.

This procedure should return `true` if the fruit should be eaten, and `false` otherwise. If `true` is returned when `is_lemon` is `true`, you will receive a `Wrong Answer` verdict.

For Chisato, you should implement:

```
int answer(int subtask, int n, std::string b,
        std::vector<int> uneaten)
```

- `subtask`: the index of the subtask to which the test case belongs.
- n: the number of fruits.
- b: the string returned by `init`.
- `uneaten`: the sorted vector of length $n - y + 1$ of labels of fruits not eaten by Takina, where:
  - `uneaten[0]` $= 0$, and
  - `uneaten[i]` is the i-th smallest label among uneaten fruits.
- This procedure is called t times per test case, once at the end of each game.

This procedure should return an integer, the label of the lemon. If the return value does not match the correct label, you will receive a `Wrong Answer` verdict.

In the actual grading, a program that calls the above procedures is run **twice**:

1. In the first run, the following steps are performed t times:
   - `init` is called once.
   - `receive_fruit` is called n times, following the order of fruits given to Takina.
   - Your program can store and retain information across successive calls.
2. In the second run, **the order of the games may be different from the first run**. For each of the t games:
   - `answer` is called once. Other than the parameters passed into `answer`, your program may not access information from the first run.

Since the procedure can be called more than once, you should pay attention to the effect of the remaining data from the previous call on the current call.

Subtasks

For all test cases, the input will satisfy the following bounds:

- $1 \leq t \leq 10\,000$
- $n = 500$
- $1 \leq p[i] \leq n$ for all $1 \leq i \leq n$
- There is exactly one lemon.

For each subtask, your program will be scored differently based on the length of the string Takina writes, x, and the number of fruits she eats, y. Your score for each test case is calculated using the maximum value of x in all t games and the maximum value of y in all t games.

Subtask	Score
1	If $y > 2$, your score is 0. Otherwise, your score is $10 \times \min\left(\frac{288}{x}, 1\right)$.
2	If $y > 9$, your score is 0. Otherwise, your score is $30 \times \min\left(\frac{30}{x}, 1\right)$.
3	Your score is $60 \times \min\left(\frac{20}{x+y}, 1\right)$.

Sample Interaction

Consider a single game ($t = 1$) with $n = 4$. Note that this does not satisfy the input constraints and is only used here for demonstration purposes.

Suppose that Takina is given the order of fruits $p = [0, 3, 1, 4, 2]$.

This means that the fruits will be presented in the following order of labels: $3, 1, 4, 2$.

Assume that the fruit with label 4 is the lemon.

First, `init` is called. Takina receives the parameters `subtask`, n, and p, and returns string b.

Then, `receive_fruit` is called once for each fruit in the given order. Each time, Takina is informed whether the fruit is the lemon and decides whether to eat it.

Finally, Chisato receives the string b and the sorted set of labels of fruits that were not eaten, and the procedure `answer` is called.

One possible sequence of calls and return values is shown below.

Step	Procedure Call	Parameters	Return Value
1	`init`	(`subtask`, 4, [0, 3, 1, 4, 2])	`"101"`
2	`receive_fruit`	(3, `false`)	`true`
3	`receive_fruit`	(1, `false`)	`false`
4	`receive_fruit`	(4, `true`)	`false`
5	`receive_fruit`	(2, `false`)	`true`
6	`answer`	(`subtask`, 4, `"101"`, [0, 1, 4])	4

In this example, Takina eats the fruits with labels 3 and 2, so the uneaten fruits are $\{1, 4\}$. The vector `uneaten` passed to `answer` is therefore $[0, 1, 4]$.

Using b and `uneaten`, the procedure `answer` returns 4, which is the label of the lemon. This strategy was able to correctly identify the lemon with $x = 3$ and $y = 2$.

Testing

You may download the sample grader (`grader.cpp`), the header file (`lemon.h`) and a solution template (`lemon.cpp`) under the Attachments. Two input files are provided for your testing, `sample1.txt` and `sample2.txt`. You may use the scripts `compile.sh` to compile and `run.sh` to run your solution for testing.

CMS user tests are not supported for this problem.

Sample Grader

A sample grader is provided to help test your implementation locally. Unlike the official grader, the sample grader runs your program only **once** and does not alter the order of tests for Takina and Chisato.

Input Format

The first line of input contains one integer `subtask`.

The second line of input contains one integer t.

The next t lines of input each describe one game. Each game is described by:

- a line containing two space-separated integers n and l, representing the number of fruits and the label of the lemon, and
- a line containing n space-separated integers $p[1], p[2], \ldots, p[n]$.

```
subtask
t
n_1 l_1
p_1[1] p_1[2] ... p_1[n_1]
n_2 l_2
p_2[1] p_2[2] ... p_2[n_2]
...
n_t l_t
p_t[1] p_t[2] ... p_t[n_t]
```

Output Format

The sample grader outputs a single real number representing the score fraction calculated from the values of x and y over all games.

Note

Additional diagnostic messages may be printed to standard error. The sample grader does **not** simulate the behaviour of the official grader.