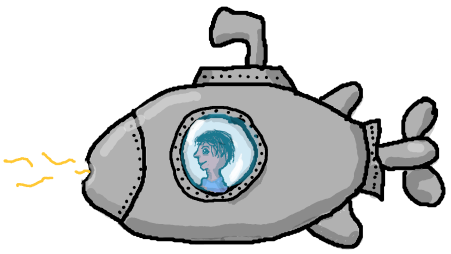
Flight to the Ford (communication)

Every break-in, even if hypothetical, needs a good escape plan. Thus, you have hired an assistant to help you in your escape from the underwater vault you discovered yesterday.

In order for your plan to work, it will be important to communicate with your assistant. More precisely, you will need to send them one of N distinct messages (conveniently numbered from 1 to N). Unfortunately, your possibilities to send a message are very limited once you're in your fancy new submarine: There are only two different types of signals you can send. Therefore, you have to encode your message as a sequence of these signals.

However, sending such a signal is a complicated process* and can even fail; in this case, the *wrong* signal is sent. *At least you can be sure that this will never happen twice in a row.* Moreover, you will always know which signal was actually sent, and will be able to react accordingly.



Obviously the signal displayed in the picture is to be interpreted as the bit 1

You already noticed that it might be impossible to unambiguously communicate a message under these circumstances. Hence, you will be happy when your assistant can *determine at most two messages that you possibly wanted to communicate*, i.e. such that your original message is one of them. Remembering that you are a talented programmer, you now want to write a program which

- you can use to decide which signals to send to your assistant, *and*
- your assistant can use to determine the two possible messages.

As sending signals from your submarine might raise suspicion,† you can send at most 250 of them. Beware moreover that your assistant will have to react to your message quickly. *Thus, they must notice when the communication ends without waiting for further signals!*

Communication

This is a communication task, in which your program is run several times for each testcase. You have to write the following functions; in each run of your submission, precisely one of them will be called, *multiple times* and possibly with different parameters:

- The function **void** *encode*(**int** N, **int** X) where as above N denotes the number of distinct messages and X is the message you want to communicate, where $1 \le X \le N$. Inside *encode* you may make up to 250 calls to the grader function **int** *send*(**int** s); s must be either 0 or 1, meaning you want to send signal s. The return value of this function tells you which signal was actually sent. This value may differ from s, but for any two consecutive calls to *send* inside the same call to *encode* this will happen at most once.
- The function **std**::**pair**⟨**int**, **int**⟩ *decode*(**int** N) where N is the same as in *encode*. For each call to *encode*, there is one call to *decode*. Inside *decode* you may call the grader function **int** *receive*() which returns the next signal sent during the corresponding call of *encode*. In the end, *decode* should return a pair of two integers $1 \le a, b \le N$ (the case $a = b$ is allowed) such that the original message was one of a or b.

* Involving the submarine's torpedo tube, a surprising and elegant application of Dijkstra's algorithm, and a family size spaghetti pack

† It *does* lead to major radio interference as well as deeply disturbed local wildlife after all.

You will automatically fail a testcase if you call *receive* inside *decode* more often than *send* was called in the respective run of *encode*. However, you are allowed to call *receive* fewer times than that.

If any of your function calls does not satisfy the above constraints, your program will be immediately terminated and judged as **Not correct** for the respective testcase. You must not write to standard output or read from standard input; otherwise you may receive the verdict **Security violation!**.

You must include the file `communication.h` in your source code. To test your program locally, you can link it with `sample_grader.cpp`, which can be found in the attachment for this task in CMS (see below for a description of the sample grader, and see `sample_grader.cpp` for instructions on how to run it with your program). Beware that for simplicity this sample grader does *not* run your program twice but instead calls both *encode* and *decode* (exactly once) as part of a single run. The attachment also contains a sample implementation as `communication_sample.cpp`.

Important technical remarks

As mentioned above, the functions *encode* and *decode* can be called several times per run. Please note the following:

1. It is *not* guaranteed that the calls to *decode* will appear in the same order as the calls to *encode*.
2. The time limit below and the runtime displayed inside CMS refer to the *average runtime* of all calls to *encode* and *decode* in a given run. Put differently, when there are K calls to *encode* or K calls to *decode* in a given run, then your program must not take more than $K \cdot 0.005\,\text{s}$ for this run. It is guaranteed that there at least 50 calls to *encode* or *decode* in each run.
3. As usual, the memory limit refers to the maximum memory consumption at any point in time during execution.

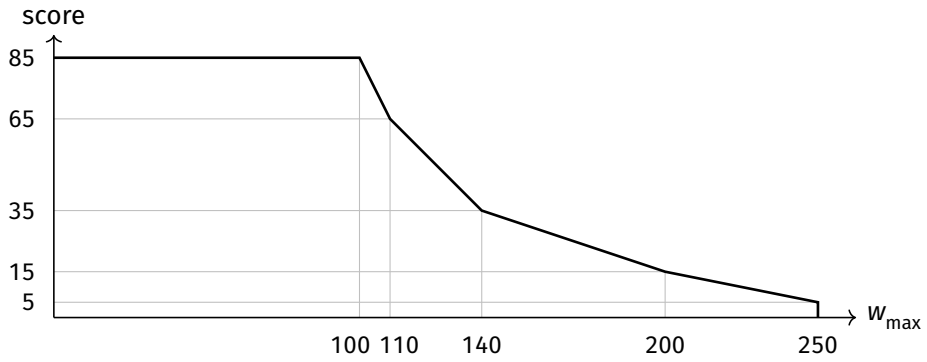
Constraints

We always have $3 \le N \le 10^9$.

Subtask 1 (15 points). $N = 3$

Subtask 2 (85 points). No further constraints.

Your actual score in Subtask 2 depends on the maximum number $w_{\max}$ of signals sent over all messages in testcases in this subtask according to the following piecewise linear function (rounded to the unique nearest integer):



In particular, to get full score you must not make more than 100 calls to *send* per testcase.

Example Interaction

Consider a testcase with $N = 1337$ and $X = 42$.

First, the grader calls your function *encode* as *encode*(1337, 42). Then, a possible interaction between your program and the grader could look as follows:

Your program	Return value	Explanation
send(1)	0	the wrong signal was sent
send(0)	0	the correct signal was sent (as was guaranteed)
send(1)	1	the correct signal was sent again
send(1)	0	the wrong signal was sent

Afterwards (in a new run of your program) the grader calls your function *decode* as *decode*(1337). Here a possible interaction could look as follows:

Your program	Return value	Explanation
receive()	0	the first call to *send* returned 0 (although this wasn't the parameter you called it with)
receive()	0	the second call to *send* returned 0
receive()	1	the third call to *send* returned 1
return {1337, 42}	—	your solution is correct and is accepted

Note that your program would have been allowed to call *receive* one more time.

Grader

The sample grader first expects on standard input the integers N and X ($1 \le X \le N$). Then, the grader calls *encode*(N, X) and writes to standard output a protocol of all calls to *send* by your program. For each call to *send* it expects the return value on standard input.

Afterwards the grader calls *decode*(N) and writes to standard output a protocol of all calls to *receive* by your program. Upon termination, it writes one of the following messages to standard output:

Invalid input. The input to the grader via standard input was not of the above form.

Invalid send. You called *send* inside *decode* or you called *send* with a parameter different from 0 or 1.

Invalid reply to send. The return value to *send* given on standard input was neither 0 nor 1, or it differed from the argument to *send* twice in a row.

Looks (and smells) fishy. You called *send* more than 250 times.

Invalid receive. You called *receive* inside *encode*.

Assistant waiting for Godot. You called *receive* more often than *send*.

Invalid answer. The function *decode* did not return a pair of integers between 1 and N.

Wrong answer. The pair returned by *decode* does not contain the original message X.

Correct: *w* signal(s) sent. The pair returned by *decode* contains the original message X and there were w calls to *send*.

In contrast, the grader actually used to judge your program will only output **Not correct** (for any of the above errors) or **Correct: *w* signal(s) sent**. Moreover, it is *adaptive*, i.e. the parameters N and X as well as the return values of *send* can depend on the behaviour of your program in the current as well as other runs. Both the sample grader and the grader used to judge your program will terminate your program automatically whenever one of the above errors occurs.

Limits

Time: 0.005 s
Memory: 8 MiB

Stranded Far From Home (island)

You just couldn't leave it alone... You actually carried out the break-in, and initially everything worked out as planned. However, the communication with your assistant went terribly wrong.* Instead of returning safely to Lübeck, you are now stranded on a small island, and your submarine is out of fuel.

To return in time for the BOI award ceremony, you now have to get to the ferry on the other side of the island. However, the local population has strange traditions. Ties are very important to them, and every village has its own preferred tie color which might change over time.

A report from the internet tells you that different villages initially preferred different tie colors. Unfortunately, the report is quite outdated. Since then, every week exactly one village convinced a neighboring village to prefer the same tie color as them. (Two villages are neighbors if they are directly connected by a road.) However, this can only happen if there were at least as many people on the *entire* island who preferred the tie color of the first village as there were people who preferred the tie color of the second village. Enough time has passed so that all islanders now prefer the same tie color.

You are almost sure that the islanders won't let you pass if you don't wear a tie matching their preference. To get to the ferry, you therefore plan to wear a tie of every color that the islanders could prefer. However, wearing too many ties will make you look suspicious. Write a program that uses a description of the island to calculate which ties you have to wear.

Input

The first line of the input contains two integers N and M where N is the number of villages and M is the number of roads on the island. The villages are numbered from 1 to N.

The next line contains N integers $s_1, \ldots, s_N$ where s_i describes the number of inhabitants of village i.

Each of the following M lines contains two integers a and b ($1 \le a, b \le N$, $a \ne b$), meaning that there is a road between villages a and b. All villages are connected to each other at least indirectly.

Output

Your program should output a string of length N consisting of `0`s and `1`s. The i-th digit should be `1` if and only if it is possible that all islanders now prefer the tie color that village i preferred initially.

Constraints

It always holds that $1 \le N \le 200\,000$, $0 \le M \le 200\,000$, and $1 \le s_i \le 10^9$.

Subtask 1 (10 points). $N \le 2\,000$ and $M \le 2\,000$.

Subtask 2 (10 points). $s_1 \ge \ldots \ge s_N$, and every village b with $b > 1$ is directly connected to exactly one village a with $a < b$.

* That was to be expected, right?

Subtask 3 (15 points). Villages a and b are directly connected if and only if $|a - b| = 1$.

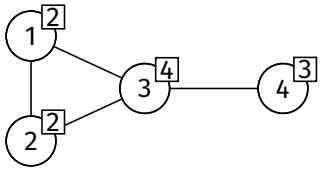
Subtask 4 (30 points). There are at most 10 distinct numbers of inhabitants.

Subtask 5 (35 points). No further constraints.

Examples

Input	Output
4 4 2 2 4 3 1 2 1 3 2 3 3 4	1110
4 3 4 2 2 1 1 2 3 2 4 1	1110

The following picture shows the first example:



The first digit of the output is 1 since it is possible that all islanders now prefer the tie color that village 1 preferred initially. This could happen as follows: In the first week, village 1 convinces village 2 that their tie color is better. After that, there are four people who prefer the tie color of village 1. Therefore, village 1 can now convince village 3 of their tie color, and if afterwards village 3 convinces village 4, everybody prefers the tie color that village 1 preferred initially.

The last digit of the output is 0 since it is impossible that village 4 convinces anyone of their tie color. This is because village 4 is only connected to village 3, but village 3 has more inhabitants.

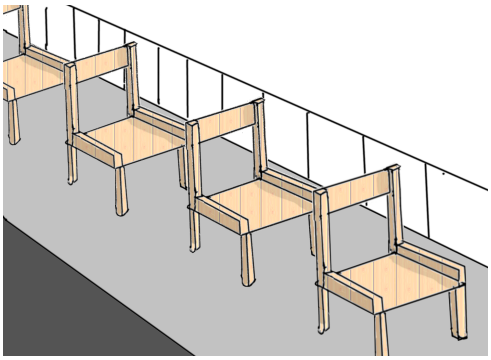
Note that the second example is a valid test case for the second subtask.

Limits

Time: 1 s
Memory: 512 MiB

Boarding Passes (passes)

After successfully navigating the local traditions, you managed to catch the ferry just in time for its departure. However, you did not expect that many people to be headed for Lübeck! Since you do not want to be late for the award ceremony,* you want to speed up the boarding process of the ferry.



The ferry has a single row of N seats which is fully booked by N passengers. Each passenger has a ticket that gives their assigned seat and one out of G boarding groups.

When boarding, these groups are called one after the other. The passengers within each boarding group will board in a random order, with all possible orders being equally likely. Each passenger can board the ferry either at the front or back of the row of seats and will then move to their assigned seat before another passenger boards.

You determined that the most time-consuming element of this process is when a passenger has to *pass* someone who is already seated.† Fortunately you found a staff uniform in a nearby locker, so you can decide the order in which the groups are boarding and tell each passenger *before the boarding begins* whether to board the ferry from the front or back of the row of seats.

Write a program that uses the ticket information to calculate the minimal expected number of passes during boarding if you choose the order of the boarding groups and the assignment of the passengers to the front and back optimally.

Note

Given an order of the boarding groups and an assignment of the passengers to the front and back, the expected number of passes is defined as

$$1 \cdot p_1 + 2 \cdot p_2 + 3 \cdot p_3 + \cdots$$

where p_k is the probability that there are exactly k passes during boarding. Put differently, this is the average number of passes among all possible orders of the passengers in each boarding group.

Input

The input consists of a string of N characters $s_1 \ldots s_N$ where s_i is one of the first G uppercase characters of the English alphabet, representing the boarding group of the passenger assigned to the i-th seat in the row (with the seat at the front of the row being seat 1).

Output

Your program should output a single number, the minimal expected number of passes if you choose the order of the boarding groups and the assignment of the passengers to the front and back optimally.

Your answer will be accepted if its absolute error is at most 0.001.

* You will also need some time to store all your stolen art in the hostel.
† The luggage containing all those ties is a considerable obstacle in the aisle.

Constraints

We always have $1 \le G \le 15$ and $1 \le N \le 100\,000$.

Subtask 1 (5 points). $G = 1$, i.e. there is only one boarding group.

Subtask 2 (25 points). $G \le 7$ and $N \le 100$

Subtask 3 (30 points). $G \le 10$ and $N \le 10\,000$

Subtask 4 (40 points). No further constraints.

Examples

Input	Output
AACCAA	1
HEHEHEHIHILOL	7.5
ONMLKJIHGFEDCBAABCDEFGHIJKLMNO	0
AAAAAAAAAAAAA...AAAAAAAAAAAAAA	100800.5

In the fourth example, the input consists of the character A repeated 899 times. All example inputs can be found in the attachment for this task in CMS.

In the first example, group C should board before group A and the passengers in the 1st, 2nd, and 3rd seats should board from the front while the rest should board from the back.

This makes it impossible for the two passengers of group C to pass each other, and they will not pass any passenger of group A since group C boards first. Moreover, the passengers of group A will not pass any passenger of group C since all passengers of group A boarding from the front are seated in front of the passengers of group C and all passengers of group A boarding from the back are seated behind the passengers of group C.

Therefore, the only possible passes are the passenger in the 2nd seat passing the passenger in the 1st seat, which will only happen if the passenger in the 1st seat boards before the passenger in the 2nd seat, and the same for the passengers in the 5th and 6th seats. Since each of these events occurs with a probability of 50%, it follows that the expected number of passes is equal to 1.

Limits

Time: 2 s
Memory: 1024 MiB