Political Patricians (patricians)

A roman censor is monitoring N patricians with dubious political allegiances. Each patrician belongs to **exactly one** secret political faction, and each faction has one or more patricians. The censor wants to determine which patricians belong to which factions.

In order to do so, the censor will repeatedly organize banquets in which all of the N patricians are invited. In each banquet, the patricians can be seated at different tables, and the censor controls the seating arrangement. A patrician is *lonely* if there is no other patrician of the same political faction seated at the same table.

As the master of ceremonies, the censor will know after each banquet which patricians were lonely. Your task is to help the censor determine the political factions of the patricians without organizing too many banquets.

Implementation

You will have to submit a single `.cpp` source file.

Among this task's attachments you will find a template `patricians.cpp` with a sample implementation.

You will have to implement the following function:

C++
```
vector<int> find_factions(int N)
```

- Integer N represents the number of patricians. Patricians are numbered from 0 to $N-1$.
- The function should return an array P of N integers representing the political factions of the patricians. $P[i]$ should be equal to $P[j]$ if and only if patricians i and j belong to the same political faction. There are no other constraints on the values of the integers of P.

Your program can call the following function, which is already defined in the grader:

C++
```
vector<bool> organize_banquet(const vector<int>& Q)
```

- The array Q of N integers represents the seating arrangement: patrician i will be seated at table $Q[i]$. There are no constraints on the values of the integers of Q: tables can be given any numbering, and the only relevant information is that if $Q[i] = Q[j]$, then patrician i and patrician j will be seated at the same table.
- The function will return an array A of N booleans. $A[i]$ will be true if and only if the patrician i is lonely at the banquet.

The grader is **not adaptive** and each submission is graded on the same set of test cases. That is, the political factions of each patrician are fixed beforehand in each test case and do not depend on the calls to `organize_banquet` made by your program.

Sample Grader

Among this task's attachments you will find a simplified version of the grader used during evaluation, which you can use to test your solutions locally. The sample grader reads data from `stdin`, calls the function `find_factions` and writes back on `stdout` using the following format.

The input file is formed by 2 lines, containing:

- Line 1: the integer N.
- Line 2: N integers describing the political faction of each patrician.

The output is made up of one line, containing the score obtained by your solution in the test case. If the score is 0, it prints a brief message to `stderr` with the reason (incorrect factions given as answer, invalid query, or too many queries).

Constraints

- $4 \leq N \leq 2000$.

You can call `organize_banquet` at most 30000 times.

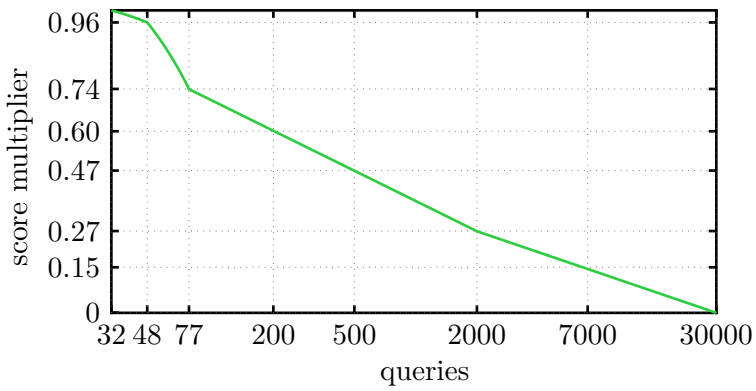
Scoring

Your program will be tested against several test cases grouped in subtasks. The score multiplier associated to a subtask will be the minimum of the score multipliers obtained in each of the test cases.

- **Subtask 0 [0 points]**: Sample test cases.
- **Subtask 1 [19 points]**: There are exactly two political factions.
- **Subtask 2 [29 points]**: Each political faction contains exactly two patricians.
- **Subtask 3 [52 points]**: No additional constraint.

The score multiplier obtained in a test case is determined as follows.

Number of queries	Score multiplier
≤ 32	1.0
$33 \leq x < 48$	$1.08 - 0.0025x$
48	0.96
$48 < x < 77$	$1.32413 - 0.00758x$
77	0.74
$77 < x < 2000$	$1.366 - 0.1 \log_2(x)$
2000	0.27
$2000 < x \leq 30000$	$1.027 - 0.0691 \log_2(x)$



The score obtained on a subtask is the minimum score multiplier obtained over its test cases, multiplied by the base score of the subtask.

Examples

Grader	Solution
`find_factions(5)`	
	`organize_banquet([0,0,1,1,2])`
`return [1,1,1,1,1]`	
	`organize_banquet([0,0,0,1,1])`
`return [0,1,0,1,1]`	
	`organize_banquet([0,1,2,1,1])`
`return [1,0,1,0,1]`	
	`return [0,1,0,1,2]`

Grader	Solution
`find_factions(4)`	
	`organize_banquet([1,1,0,0])`
`return [1,1,1,1]`	
	`organize_banquet([1,1,1,2])`
`return [0,1,0,1]`	
	`organize_banquet([0,1,1,1])`
`return [1,0,1,0]`	
	`return [0,1,0,1]`

Explanation

In the first example, patricians 0 and 2 are in one political faction, patricians 1 and 3 are in a different one, and patrician 4 is in its own faction. In the first banquet, all patricians are lonely. In the second banquet, patricians 0 and 2 are not lonely because they are seated together. In the third banquet, patricians 1 and 3 are not lonely because they are seated together.