

## Episodio IV: tracce di DNA (dna)

After the contest and the buffet, the staff happily goes back to the control room to arrange the award ceremony. But a nasty surprise is waiting for them: someone tried to hack their database to change the results! Even though the attempt failed, thanks to the unbreakable security measures used by the staff, the culprit of the crime shouldn't go unpunished.

A careful analysis detects organic samples on the server keyboard. The first attempt, a traditional DNA sequencing, fails: the DNA thus belongs to the Proxima B aliens! Suspicion immediately falls on  $\text{Q}\text{z}\text{I}\text{m}\text{V}\text{I}\Delta\text{M}\text{I}$ , who has already tried to bribe the staff using a suspicious amount of souvenirs... but he can't be accused without proof.




Figure 1: The experimental procedure of alien DNA test.

The alien DNA, unlike the Earth's one, only contains two nucleotide types: 0 and 1. Its study is at the very beginning, so it can't be directly sequenced. However, a new test has recently been developed: given a sequence made of characters 0 and 1, the test can determine whether it is contained (as a contiguous substring) in the alien DNA sample. Since this test is quite expensive, the staff would like to sequence the DNA sample using as few tests as possible: help them plan the sequencing optimally!

## Implementation

You should submit a single file, with a `.cpp` extension.

 Among the attachments in this task you will find a template `dna.cpp` with a sample implementation.

You will have to implement the following function:

```
C++ | string analizza(int N);
```

- The integer  $N$  represents the length of the detected alien DNA sample.
- The function must return a string of length  $N$ , consisting only of characters 0 and 1, corresponding to the DNA sample sequencing.

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

```
C++ | bool test(string T);
```

- The string  $T$  must have length between 1 and  $N$  (included) and it must consist only of the characters 0 and 1.
- The function returns **true** if the string  $T$  is a contiguous substring of the sample, **false** otherwise.

The grader will call the function **analizza** and print the return value on the output file.

## Sample Grader

Among this task's attachments you will find a simplified version of the grader used during the evaluation, which you can use to test your solutions locally. The sample grader reads data from **stdin**, calls the functions that you should implement 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: the string that represents the DNA sample.

The output file is formed by  $Q + 1$  lines, with  $Q$  being the number of calls to **test** made by your program. The lines contains:

- Line  $1 + i$  ( $0 \leq i < Q$ ): the string  $T$  used as a parameter in the  $i$ -th call to **test**.
- Line  $1 + Q$ : the string returned by the function **analizza**.

## Constraints

- $1 \leq N \leq 10\,000$ .
- The function **test** can be called at most 30 000 times.
- Unlike the sample grader, the official grader is *adaptive*. Thus, the DNA sample is not predefined: instead, it's determined throughout the calls to the function **test**.

## Scoring

Your program will be tested on a variety of test cases. Example cases are included in the correction but aren't considered in the scoring. The score of a test case is 0 if the program doesn't produce an answer within the time limit or if the answer is wrong. Otherwise, let  $Q$  be the number of queries made by your program. The score of the test case is as follows:

Q	Score
> 30 000	0
30 000	10
20 000	20
15 000	40
11 000	60
10 200	70
10 035	85
10 025	90
$\leq 10\,015$	100

If  $Q$  is between two of the values above, the score will also be in between, assigned through linear interpolation. The score assigned to the problem is the *worst* score among all test cases.

## Examples

stdin	stdout
101010	Correct! 5 queries.

You can find more examples cases among this task's attachments.

## Explanation

The following is a possible interaction that solves the first example:

- `test("0")` returns `true`;
- `test("00")` returns `false`;
- `test("1")` returns `true`;
- `test("11")` returns `false`;
- at this point there can't be two consecutive equal characters, so the DNA is either "010101" or "101010";
- `test("010101")` returns `false`, so the answer is "101010".