DNA dictionary

Time limit:	1 second
Memory limit:	256 megabytes

To help in your research, Bakhyt has given you a DNA dictionary - a data structure that stores N DNA-strings. DNA-strings are strings of length K consisting only of uppercase letters 'A', 'C', 'G', and 'T'. Unfortunately, this dictionary has a weird interface, and you need to find a way to use it properly.

This dictionary allows you to access its data using templates. A template is a string of length K that can have three types of characters - uppercase letters ACGT, question marks and underscores. For any given template the dictionary will check all of its DNA-strings whether they match the template and tell the information about the strings that do. A string S matches the template T if there is a match at every position i ($1 \le i \le K$) except the positions when T_i is an underscore. Match at a position is determined according to the following rules:

- `'A', 'C', 'G', 'T'` — if T_i is one of the letters, this position is a match only if S_i is equal to T_i
- `'?'` — if T_i is a question mark, this position is a match regardless of S

The dictionary has only one function

- *get_min_max(T)* — Find the lexicographically minimal and maximal strings that match the given template T

An underscore in a template means that this position will be **completely ignored** during the check. The strings returned by *get_min_max(T)* will also have underscores at the same positions that T does.

You need to find a way to determine for any given DNA-string the string from the dictionary that would stay right after it in alphabetical order (or that is equal to it).

Interaction Protocol

YOUR SUBMISSIONS MUST NOT READ FROM THE STANDARD INPUT, PRINT TO THE STANDARD OUTPUT OR INTERACT WITH ANY OTHER FILE.

Your task is to implement the following function:

```
string find_next(string P)
```

- P is a DNA-string of length K
- The function must return the lexicographically minimal string from the dictionary that is lexicographically greater or equal to P.
- If no such string exists the function must return an empty string.

You can call the following function

```
pair <string, string> get_min_max(string T):
```

- T is a template string of length K
- The function will return the pair of lexicographically minimal and maximal strings in the dictionary that match the template T, with underscores at the same positions as in T.
- If no strings match the template, the function will return a pair of empty strings.

You can make no more than 2500 function calls in total in each test case. If any of the above conditions are violated, your program will get the **Wrong Answer** verdict. Otherwise, your program will get the **Accepted** verdict and your score will be calculated based on the total number of calls of the functions *get_min_max* and the total number of questions marks in the parameters of the function across all calls (Refer to the section "Scoring").

Scoring

In this task, the grader is NOT adaptive. It means that the content of the dictionary is fixed at the beginning of the run and does not depend on calls from your program.

This task consists of two subtasks:

1. (10 points) $K = 4$.
2. (90 points) $K = 256, N \leq 2 \times K$ In this subtask, your score is calculated in the following manner. Let q be the total number of calls of the function *get_min_max* and let m be the total number of question marks used in the parameters of *get_min_max* across all calls. Then your score is calculated in the following manner.
   - If $q + m \geq 2500$, your score is 0.
   - If $q + m \leq 550$, your score is 90.
   - If $550 < q + m < 2500$, your score is calculated according to the formula $\frac{2500-(q+m)}{2500-550} \cdot 90$.

   Note that your score for this subtask is equal tp the minimum score among the scores on all of its test cases.

Note

The sample grader reads the input in the following format:

- Line 1: N, K
- Line 2: $S_1, S_2, ..., S_n$

YOU CAN DOWNLOAD `dnadict.zip` in the system that contains examples for languages Java and C++11.

All the examples of calling the functions can be found above.

`dnadict.zip` contains examples of solutions for each language.

For the solutions in `Java` language, file and class name have to be named as `Dnadict.java` and `Dnadict` respectively.