String from Substrings

PROBLEM

Every DNA string consists of only 4 letters, `A`, `T`, `G` and `C`. You have an unknown DNA string and must determine the string. The only way you can access information about the hidden string is through an oracle. The oracle, when given a query string *S*, answers whether the hidden string contains *S* as a substring. For example, let the hidden string be S_o= "`ATTGCGCGATCG`". Then "`ATTG`" and "`CGCG`", "`T`", "`AT`" are substrings of S_o. But neither of "`TGG`" or "`GCGATG`" is not a substring of S_o. When a string S_o is represented as S_o= $(s[1], s[2], s[3], \ldots, s[N])$, where $s[i]$ is the i-th character of S_o, then a substring of S_o is a consecutive subsequence represented as $(s[i], s[i+1], s[i+2], \ldots, s[j])$, where $1 \le i \le j \le N \le 255$.

You are to write a program, which determines the hidden string using as few oracle queries as possible.

LIBRARY

You are given a library in the following.

GNU C/C++ Library: (`oracle.h, oracle.o`)

The C/C++library has the following three functions:

`void start_string()`: The call to start. It should be called only once at the beginning.

`int oracle_call(char *S)`: If `S` is a substring of the hidden string, this function returns 1. Otherwise, this function returns 0. The query string `S` should not be an empty string, and the length of `S` should be equal or less than 255.

`void answer_string(char *S)`: This function will terminate your program. Your program passes the string `S` as an answer. This should be called only once at the end of the program.

Instruction: To compile your string.c or string.cpp, use the include statement

```
 #include "oracle.h"
```

in the source code and compile it as:

```
 gcc -O2 -static string.c oracle.o -lm
 g++ -O2 -static string.cpp oracle.o -lm
```

`lib_test.c` shows how to use the GNU C/C++ library.

FreePascal Library: (`oracle.ppu`, `oracle.o`)

The corresponding pascal library functions are

```
procedure start_string;
function oracle_call(S: string): integer;
procedure answer_string(S: string);
```

Instruction: To compile your `string.pas`, include the import statement

```
 uses oracle;
```

in the source code and compile it as

```
 fpc -So -O2 -XS string.pas
```

`lib_test.pas` shows how to use the FreePascal library.

The libray generates a file named `string.out` automatically in a call to `answer_string`. The file `string.out` has two lines. The integer in the first line shows the number of calls to `oracle_call` made by your program and the second line contains the hidden string your program has given as a solution.

EXAMPLE INPUTS AND OUTPUTS

The length L of the hidden string satisfies $1 \leq L \leq 255$. You can experiment with the library by creating a text file `string.in` with a single line which contains the hidden DNA string. Note that the `string.out` in the following example is not necessarily optimal. It merely shows the file format of input and output.

`Example1:`   `string.in`

```
ATTGCGCGATCG
```

`string.out`

```
41
ATTGCGCGATCG
```

SCORING

If your program violates one of constraints (e.g. calling too many function calls), then you get 0 point.

If your solution is not correct, then the score is 0. When the output solution is correct, then your score depends on the number of library function calls for each testing data. For each data if the number of function calls is less than a bound B (that is fixed independently for each data), then you get full score. Otherwise you will get 0 points.