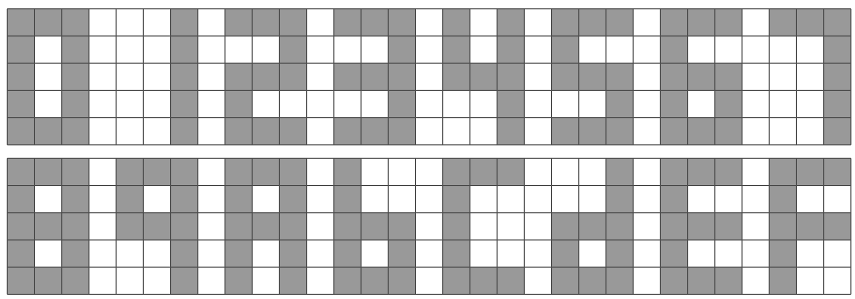
3 Digits

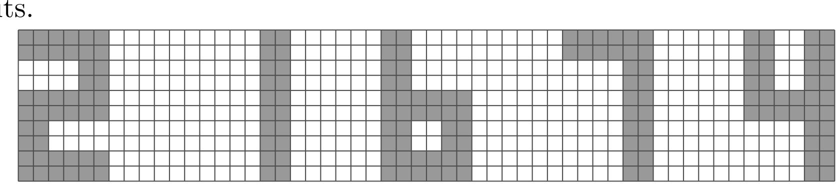
Orange has released their latest new shiny electronic device, equipped with their cutting-edge Cornea display with zillions of tiny subatomic pixels, a carbon fibre body, and hardware support for Facebook integration. Crazyman has won one such gadget in a programming contest, but he is using it simply to display a number in hexadecimal, even though the 7-segment display on a calculator would have sufficed for this purpose. As a challenge, you have decided to find the number written on the screen by querying a few pixels of the screen.

We first describe how a sequence of digits is represented as a rectangular grid of *blocks*, each of which can be white (the background), or black (the foreground). Each digit occupies a grid of blocks with 3 columns and 5 rows. Two consecutive digits have *three* empty columns of blocks between them. The following two grids show "01234567" and "89abcdef" respectively, *except that due to space constraints, only one empty column between two digits instead of three is shown.*

There are only 16 possible digits, and they are all shown here.



Finally, to display a given sequence of digits on the screen using font size k ($k \geq 1$), we first translate the sequence of digits into a grid of blocks as described above, and then translate this to a grid of pixels by using a $k \times k$ square of pixels for every block. For example, the following shows "21b74" displayed with font size $k = 2$. Note that this correctly has three columns of blocks between two consecutive digits.



As you can calculate, a sequence of m digits displayed with font size k will require a rectangle (called its *display rectangle*) occupying $5k$ rows and $6mk - 3k$ columns of pixels on the Cornea display.

Your task is the following : a sequence of digits is displayed somewhere on the screen, in some font size. You will be given the location of one black pixel on the screen. By querying the black/white status of various pixels on the screen, you have to determine the sequence of digits and the top-left corner of the display rectangle used (note: in some cases, the top-left corner may be a white pixel). You may make at most 5000 queries.

Library interaction

For this problem, you must *not* write a `main` function and must *not* perform any input or output. Instead you should write a function `solve` which plays the role of your main function, with the following signature:

```
void solve(long long R, long long C, long long br, long long bc,
    int &ans_N, char *ans_digits, long long &ans_r, long long &ans_c);
```

The grader will call this function with appropriate arguments. `R` and `C` describe the screen resolution: the number of rows and columns of pixels respectively. The rows are numbered from 0 to $\texttt{R} - 1$ from top to bottom, and the columns are numbered from 0 to $\texttt{C} - 1$. The pixel `(br, bc)` is black. It is guaranteed that $0 \leq \texttt{br} < R$ and $0 \leq \texttt{bc} < C$.

Your code can call the function `pixel` to find out the colour of a particular pixel:
`bool pixel(long long r, long long c);`
`pixel(r, c)` returns `true` if the pixel `(r,c)` is black, and `false` otherwise, for $0 \le r < R$ and $0 \le c < C$. If (r, c) is not within range, your program will be terminated.
Finally when the function `solve` is ready to report the answers, it should write the answers to the following variables:

- `ans_N` : The number of digits displayed.
- `ans_digits[0]`, ..., `ans_digits[N-1]` : The digits displayed. Each element must be one of `'0'` ...`'9'`, `'a'` - `'f'`, or `'A'` - `'F'`. Both lowercase and uppercase letters will be accepted and can be used interchangeably.
- `ans_r`, `ans_c` : The row and column coordinate of the top-left corner of the display rectangle. Note that this may be a white pixel or a black pixel.

You are given a template file with a `solve` function in which you can fill in your code.

Compiling your program

In the beginning of your solution, add the following line:
`bool pixel(long long, long long);`
The template provided includes this. If your file is called `digits.cpp`, you may compile as follows:
`g++ digits.cpp digitslib.o -W -Wall -O2 -o digits`

Test data

In all subtasks, $5 \le R \le 10^{18}$ and $3 \le C \le 10^{18}$, where R and C are the number of rows and columns of pixels respectively. $1 \le N \le 100$, where N is the number of digits displayed. You can make at most 5000 calls to `pixel`.

- Subtask 1 (15 marks) : The font size is 1.
- Subtask 2 (25 marks) : The font size is between 1 and 16 inclusive.
- Subtask 3 (15 marks) : $N = 1$, and the digit displayed is `6`.
- Subtask 4 (20 marks) : $N = 1$.
- Subtask 5 (25 marks) : No further constraints.

Experimentation

This section describes how the library works, so that you can experiment with your solution yourself. The information in this section is *not* needed to write your solution or to submit your solution on the grader. The library reads data from the standard input and then calls your `solve` function. The input format is as follows:

- Line 1 : A sequence of hexadecimal digits, of length between 1 and 100. Both lowercase and uppercase letters are accepted, and they are equivalent.
- Line 2 : Two integers, the number of rows and columns of pixels that the Cornea display has.
- Line 3 : Two integers, the row and column coordinates of the top-left corner of the display rectangle respectively.
- Line 4 : A single integer, the font size.
- Line 5 : Two integers, the row and column coordinates of a black pixel, to be passed to `solve`.

Finally, the library checks the correctness of your answers and reports it on standard output. The library writes a log of the interaction to `digits.log`. This is for your information only and you are not required to examine this file.

Limits

- *Memory limit* : 128 MB
- *Time limit* : 1s