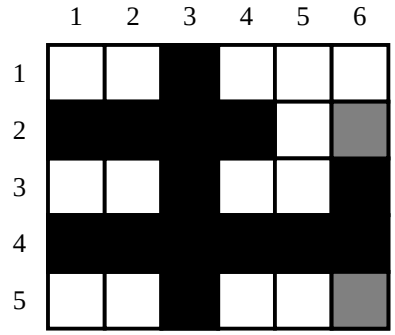
B Black Out

When you've finished solving all problems in this contest, there's no need to get bored, because we introduce a new and fun game called Black Out! Grab a piece of paper and draw a table of 5 by 6 squares. Next, choose your favorite teammate to play against. Each player in turn selects one or more squares that are adjacent and in the same row or column, and colors them all black. Some of these squares may be black already, but at least one of them must not be. The winner is the player to color the last square(s) black.

Once you got the hang of it, you might want to try and write your own AI player. You can then submit it and let it play a few games against our (= the jury's) player!

After a lengthy analysis, we have managed to prove that the game is theoretically won by the player who starts, so we'll gracefully let your player have the first move every time. If you manage to beat our player in every game, you'll be awarded another point in this contest. But be careful: the jury's player is not to be underestimated and will surely grab any chance it is offered! If you lose, you'll get a Wrong Answer. Also note that we are not that patient, so make sure your player is quick to move.



The move 2 6 5 6 as given in the sample makes all squares from (2,6) to (5,6) black. The squares (3,6) and (4,6) were black already.

Input and output

This is a problem with dynamic input and output, read the following description very carefully.

The first line of the input contains one positive number: the number of games, at most 100. Then, for each game:

- Your program writes a move on a single line to standard output. A move consists of four space-separated integers r_1, c_1, r_2 and c_2 ($1 \leq r_1 \leq r_2 \leq 5$ and $1 \leq c_1 \leq c_2 \leq 6$; also, $r_1 = r_2$ and/or $c_1 = c_2$) on a single line: the row and column number of the two squares between which all squares are colored black (including the two indicated squares). In case only one square is colored black, $r_1 = r_2$ and $c_1 = c_2$.
- After each move, our program writes one of the following two responses on a single line to standard input:

- The word "`MOVE`", followed by a single space, followed by four space-separated integers, representing the response of our program. Your program must now make another move.
- The word "`GAME`", indicating that the game has ended: either your program has won, or our program intends to make a move that ends the game in its favor.

Result

These are the possible causes for **some** of the possible results of your submission:

CORRECT Your program has won all games. Congratulations!

RUN-ERROR There was an error during the execution of your program, or your program's output did not adhere to the specified format and constraints, or all the squares that you wanted colored black were black already (invalid move).

TIMELIMIT Your program was too slow or it did not respond.

WRONG-ANSWER Your program has successfully completed all games, but it has not won all of them.

Notes

- Make sure to print a newline after your move.

C users:

- Do `setlinebuf(stdout);` at the start of your program to make sure output is flushed correctly.
- If you use `scanf` to read the moves (which we recommend), then do not read the newline character at the end. So, to read the input with `scanf`, use `scanf("%s",s)` and `scanf("%d %d %d %d", &r1, &c1, &r2, &c2);`.
  Also, do **not** use `scanf("MOVE")` or `scanf("GAME")`.

C++ users:

- Recommendation: use `std::cin` and `std::cout` to read input and write output. This will ensure output is flushed correctly.

Java/C# users:

- Do not use *buffered* output; either `System.out.println` or `System.Console.WriteLine` will do fine.

Haskell users:

- Do `hSetBuffering stdout LineBuffering` at the beginning of `main`.

Sample in- and output

Notes:

- This example is merely to illustrate the communication between your program and the jury's program; it does **not** demonstrate an example of optimal play for either player.
- The extra newlines in the example below are **for clarity only**, to demonstrate the order of events; you should print exactly one newline character after each move, and the input contains no blank lines either.

Input	Output
1 MOVE 3 6 3 6 MOVE 4 2 4 6 MOVE 2 6 5 6 MOVE 3 1 5 1 MOVE 5 2 5 5 GAME	2 1 2 4 1 3 5 3 4 1 4 6 1 1 1 6 3 2 3 5 2 5 2 5

Almost blank page