Problem: Contact!

Input file: `standard input`
Output file: `standard output`
Time limit: 2 seconds for 10 moves
Memory limit: 64 megabytes

There will be no legends, wars or any introduction for this problems. Let's get to the problem itself. You have a board of 20×20 cells. Two players capture cells in turns. First player's goal is to connect top and bottom sides of the board, second player's goal is to connect left and right sides. Two sides are considered conected if and only if there's a path from first side to second using only cells of corresponding player. Two neighbouring cells in path should be adjacent by side.

Input

Before every move you will receive current field state into stdin. You will get 20 lines with 20 numbers in each. You will get a blank line after each state. Every number a_{ij} describes one field cell. If $a_{ij} = 1$, this cell belongs to you. If $a_{ij} = 2$, then cell belongs to your opponent. $a_{ij} = 0$ denotes an empty cell. The field will be transposed in such a way, that you will need to connect top and bottom sides (and your cells will be denoted by 1).

Output

You are to output row and column numbers of cell you want to capture. Rows and columns are numbered from zero, rows are numbered from top to bottom, columns are numbered from left to right. Output an empty line after your move. Don't forget to flush output buffer.

Please note, that your program should not terminate by itself. If you know, that there's no free cells in the field (so the game will now end with a draw), do not exit for the program. You should hang without consuming CPU time. It could easily be done by reading something from stdin.

Examples

standard input

```
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
```

standard output

```
11 9
```

In this example the player doing move wins.

Sample program for C++ (making random move each turn):

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <cstdlib>

using namespace std;

const int SZ = 20;
int field[SZ][SZ];
int numfree;
int freerow[SZ * SZ];
int freecol[SZ * SZ];

int main()
{
    srand((unsigned)time(NULL));
    int i, j;
    while (1)
    {
        for (i = 0; i < SZ; i++)
        {
            for (j = 0; j < SZ; j++)
            {
                scanf("%d", &field[i][j]);
            }
        }
        numfree = 0;
        for (i = 0; i < SZ; i++)
        {
         for (j = 0; j < SZ; j++)
         {
         if (field[i][j] == 0)
         {
         freerow[numfree] = i;
         freecol[numfree] = j;
         numfree++;
         }
         }
        }
        if (numfree == 0) break;
        int pos = rand() % numfree;
        printf("%d %d\n\n", freerow[pos], freecol[pos]);
        fflush(stdout);
    }
    // this is needed to prevent early termination of program
    scanf("%d", &i);
    return 0;
}
```

Sample program for Pascal (doing random move each turn):

```
const SZ = 20;
var field : array[1..SZ, 1..SZ] of integer;
i, j : integer;
freerow, freecol : array[1..SZ * SZ] of integer;
numfree, pos : integer;
begin
randomize;
while true do begin
        for i := 1 to SZ do begin
         for j := 1 to SZ do begin
         read(field[i, j]);
         end;
        end;
        numfree := 0;
        for i := 1 to SZ do begin
         for j := 1 to SZ do begin
         if (field[i, j] = 0) then begin
         inc(numfree);
         freerow[numfree] := i;
         freecol[numfree] := j;
         end;
         end;
        end;
        if (numfree = 0) then break;
        pos := trunc(random() * 1000) mod numfree + 1;
        writeln(freerow[pos] - 1, ' ', freecol[pos] - 1);
        writeln;
        flush(output);
    end;
    read(i);
end.
```