

Aesthetic Islands

PAIO organizers are planning an excursion to Lake Kivu, which is known for its many beautiful islands. To plan properly, they need to know the exact number of islands in the lake. Fortunately, they have just received a satellite image of the map of the lake.

The map is represented as an $N \times M$ grid. The rows of the map are numbered from 0 to $N - 1$ from top to bottom, and the columns of the map are numbered from 0 to $M - 1$ from left to right. The cell in row i and column j ($0 \leq i < N, 0 \leq j < M$) is denoted by (i, j) . Each cell contains 1 if it is land and 0 if it is water.

Two land cells are considered adjacent if they share a side. In other words, two land cells are adjacent if one is directly above, below, to the left or to the right of the other. Cells touching only at a corner are not adjacent.

An island is a maximal set of land cells such that every pair of cells in the set can be connected by a sequence of adjacent land cells. Here, maximal means that no other land cell can be added to the set while preserving this property.

For example, in the following map:

```
1 0 1
0 1 0
1 0 1
```

there are 5 islands, because diagonal contact does not connect land cells.

In the following map:

```
1 1 0 1
0 1 0 1
0 0 0 0
1 1 1 0
```

there are 3 islands.

We also know that in Lake Kivu, all islands are aesthetic. An island is called aesthetic if for any land cells belonging to the island and lying in the same row or the same column, every cell between them is also land. In other words, within a single island, each row and each column occupied by the island contains one continuous block of land cells.

For example, the following are possible maps for Lake Kivu (i.e. containing only aesthetic islands):

0	0	1	0	0	1	1	1	1	1
0	1	1	1	0	1	0	0	0	0
1	1	1	1	1	0	1	1	1	0
0	1	1	0	0	1	0	1	0	1
0	1	0	0	0	1	1	0	1	1

The second map contains 4 islands, and each of them is aesthetic.

However, the following are not possible maps for Lake Kivu:

					1	1	1	0	1
1	1	1			1	0	1	0	1
0	1	0			1	0	1	0	0
1	1	1			1	0	1	0	1
					1	1	1	0	1

The organizers have access to a special computer with a memory chip of K bits, numbered from 0 to $K - 1$. The map is already stored in this memory. Specifically, the value of cell (i, j) is stored in bit $i \cdot M + j$ of the memory.

The organizers cannot read or modify the memory directly, so they don't know how the map looks like. Instead, with your help, they can submit a program to the computer.

Your program does not know the values stored in memory. It only knows N , M and K . Its task is to describe a fixed sequence of bitwise operations that the computer will later perform on the memory containing the map.

After your program finishes, the described operations are executed on the hidden memory contents. At the end, your program must choose some occupied memory bits. These bits, read from least significant to most significant, must represent the number of islands in binary after executing your program on the hidden map stored in the memory.

You need to implement a program that describes the operations the computer should perform on the memory.

Initially, bits $0, 1, \dots, N \cdot M - 1$ contain the map. These bits are already occupied and cannot be overwritten. All remaining bits are initially empty.

The computer has 6 types of instructions that can be used in your program to perform operations on memory bits. Each instruction is a bitwise operation on one or two input bits, and the result is stored in a new memory bit.

Whenever an instruction produces a new bit, the computer stores it in the first available empty bit of the memory. This bit then becomes occupied and may be used as an input bit in later instructions. Each memory bit can be written to at most once.

The available 6 instructions are described below. After receiving each instruction, the computer first creates a new bit C ($0 \leq C < K$), which is the first empty memory bit. For a bit index X , let $memory[X]$ denote the value stored in bit X of the memory.

- $and(A, B)$: Takes the bitwise-AND of bits with indices A and B and stores it in bit C of the memory. In other words, it sets $memory[C] := 1$ if both $memory[A]$ and $memory[B]$ are 1, and sets $memory[C] := 0$ otherwise. The function returns C .
- $or(A, B)$: Takes the bitwise-OR of bits with indices A and B and stores it in bit C of the memory. In other words, it sets $memory[C] := 1$ if at least one of $memory[A]$ and $memory[B]$ is 1, and sets $memory[C] := 0$ otherwise. The function returns C .
- $xor(A, B)$: Takes the bitwise-XOR of bits with indices A and B and stores it in bit C of the memory. In other words, it sets $memory[C] := 1$ if exactly one of $memory[A]$ and $memory[B]$ is 1, and sets $memory[C] := 0$ otherwise. The function returns C .
- $not(A)$: Takes the bitwise-NOT of the bit with index A and stores it in bit C of the memory. In other words, it sets $memory[C] := 1 - memory[A]$. The function returns C .
- $nand(A, B)$: Takes the bitwise-NAND of bits with indices A and B and stores it in bit C of the memory. In other words, it sets $memory[C] := 0$ if both $memory[A]$ and $memory[B]$ are 1, and sets $memory[C] := 1$ otherwise. The function returns C .
- $nor(A, B)$: Takes the bitwise-NOR of bits with indices A and B and stores it in bit C of the memory. In other words, it sets $memory[C] := 0$ if at least one of $memory[A]$ and $memory[B]$ is 1, and sets $memory[C] := 1$ otherwise. The function returns C .

The computer will execute the described operations in the order specified by your program. At the end, some occupied memory bits chosen by your program must have the binary representation of the number of islands in the map.

Implementation Details

C++/Python Interface

For C++/Python submissions, you should implement the following procedure:

```
int32[] construct_instructions(int32 N, int32 M, int32 K)
```

- N : number of rows
- M : number of columns
- K : number of bits in the memory chip
- This procedure is called exactly once and should construct a sequence of instructions to perform the required task.

- The procedure should return an array V containing the memory indices that store the answer, from least significant bit to most significant bit. In other words, let l be the length of V . The number of islands in the map must be equal to $2^0 \cdot \text{memory}[V[0]] + 2^1 \cdot \text{memory}[V[1]] + \dots + 2^{l-1} \cdot \text{memory}[V[l-1]]$.
- Every bit index in V must refer to an occupied bit.
- The length of V must not exceed 20.

This procedure should call the following procedures to construct the sequence of instructions:

```
int32 append_and(int32 A, int32 B)
int32 append_or(int32 A, int32 B)
int32 append_xor(int32 A, int32 B)
int32 append_not(int32 A)
int32 append_nand(int32 A, int32 B)
int32 append_nor(int32 A, int32 B)
```

- Each procedure appends an *and*, *or*, *xor*, *not*, *nand* or *nor* instruction to the program, respectively.
- For `append_not`, A should be an occupied bit. For each of the other five procedures, both A and B should be occupied bits.
- Each procedure returns the memory index where the produced bit will be stored.

You may also call the following procedure to help test your solution:

```
void append_print(string label, int32[] A)
```

- Every index passed in A to `append_print` must refer to an occupied bit.
- Any call to this procedure will be ignored during the grading of your solution and will not be counted as an instruction.
- In the sample grader, this procedure appends a *print*($label, A$) operation to the program.
- When the sample grader encounters a *print*($label, A$) operation during the execution of a program, it prints $label$ followed by the contents of the bit indices of A on a single line in the same order they appear in A (see "Sample Grader" section for details).

C Interface

For C submissions, include the provided `islands_c.h` header and implement:

```
int construct_instructions(int N, int M, int K, int answer_bits[]);
```

The grader supplies `answer_bits`, an array with capacity `ISLANDS_MAX_ANSWER_BITS` (20). Write the indices of V , from least significant bit to most significant bit, into `answer_bits[0]`, `answer_bits[1]`, and so on, and return the number l of indices written. Thus,

$V[i] = \text{answer_bits}[i]$ for $0 \leq i < l$. The returned value must satisfy $0 \leq l \leq 20$. Do not write beyond the capacity of `answer_bits` and do not free this grader-owned array.

In C, the exact signature of the debugging procedure is:

```
void append_print(const char *label, const int A[], int length);
```

Here, `label` must point to a null-terminated string, and `length` is the number of elements in `A` and must be nonnegative. `A` may be `NULL` only when `length` is 0.

The grading of your solution may result in one of the following error messages:

- `Invalid index`: an incorrect (possibly negative) bit index was provided as parameter `A` or `B` for some call of one of the procedures or was included in the answer produced by `construct_instructions(N, M, K)`.
- `Too many instructions`: there is not enough memory bits to store the results of the instructions. In other words, you have used more than $K - N \cdot M$ instructions.
- `Too many bits in the answer`: `construct_instructions` has produced more than 20 bits for the answer.

If the program does not result in any error messages and calculates the correct number of islands, your program will be judged as Accepted and your score is calculated according to the number of instructions used (see Subtasks). Otherwise, it will be judged as Wrong Answer.

Examples

Example 1

Suppose $N = 1$, $M = 2$ and $K = 10^7$.

There are only four possible input maps to the program:

- Case 1: 0 0
- Case 2: 0 1
- Case 3: 1 0
- Case 4: 1 1

We can notice that for all cases, the number of islands is equal to 1 if either cell $(0,0)$ or $(0,1)$ is a land cell. Therefore, a possible solution for $N = 1$ and $M = 2$ is to construct a program by using a single `or` instruction:

- `append_or(0, 1)`, which would take the bitwise-OR of cells $(0,0)$ and $(0,1)$ of the map stored in bits 0 and 1 of the memory, respectively, and would store it in bit 2 of the memory.

Then, the answer sequence produced by `construct_instructions` is `[2]`, since bit 2 of the memory would contain 1 if there is exactly 1 island in the map and 0 otherwise.

Example 2

For $N = 5$, $M = 5$, $K = 10^7$ and the following map:

```

1  1  1  1  1
1  0  0  0  0
0  1  1  1  0
1  0  1  0  1
1  1  0  1  1

```

The number of islands is 4. Therefore, when your fixed program is executed on these values, the bits whose indices are produced by `construct_instructions` must contain 0, 0 and 1 in this order since the binary representation of 4 is 100_2 .

Constraints

- $1 \leq N, M \leq 32$
- $K = 10^7$
- Each cell of the map is either 0 or 1
- All islands are aesthetic. In other words, for any land cells belonging to the same island and lying in the same row or the same column, every cell between them is also land.

Subtasks

1. (40 points) Each island consists of exactly one land cell.
2. (20 points) For each island, there exist rows r_1, r_2 and columns c_1, c_2 such that the island consists exactly of all cells (i, j) with $r_1 \leq i \leq r_2$ and $c_1 \leq j \leq c_2$. In other words, each island forms a filled rectangle of land cells.
3. (40 points) No additional constraints.

Assume your program is judged as Accepted, and uses I instructions. Then, your score P for the testcase, depending on its subtask number is calculated as follows:

- Subtask 1:
 - If $I \leq 5\,030$, $P = 40$.
 - If $5\,030 < I \leq 10\,760$, $P = 22 + 18 \cdot \frac{10\,760 - I}{5\,730}$.
 - If $10\,760 < I \leq 22\,530$, $P = 12 + 10 \cdot \frac{22\,530 - I}{11\,770}$.
 - Otherwise, $P = 8$.
- Subtask 2:
 - If $I \leq 62\,000$, $P = 20$.

- Otherwise, $P = 8$.
- Subtask 3:
 - If $I \leq 9\,100$, $P = 40$.
 - If $9\,100 < I \leq 15\,880$, $P = 22 + 18 \cdot \frac{15\,880 - I}{6\,780}$.
 - If $15\,880 < I \leq 27\,650$, $P = 13 + 9 \cdot \frac{27\,650 - I}{11\,770}$.
 - Otherwise, $P = 10$.

Sample Grader

The sample grader reads the input in the following format:

- line 1: $N\ M\ K$

This is followed by some test cases. Each test case consists of N lines:

- line i ($0 \leq i < N$): $g[i][0]\ g[i][1]\ \dots\ g[i][M - 1]$ (separated by a space)

where $g[i][j]$ ($0 \leq i < N$, $0 \leq j < M$) denotes the contents of cell (i, j) of the map.

The description of all testcases is followed by a single line containing solely -1 .

The sample grader first calls `construct_instructions` using the signature for the submission language. If this call violates some constraint described in the problem statement, the sample grader prints one of the error messages listed at the end of the "Implementation Details" section and exits.

Otherwise, the sample grader processes test cases in order. For each test case, it runs the program constructed by `construct_instructions` on the input testcase.

For each `print(label, A)` instruction, let l be the length of A , the grader prints the contents of the bit indices in A in the following format:

- *label: memory[A[0]] memory[A[1]] ... memory[A[l - 1]]*

Once all instructions have been executed, the sample grader prints the binary number constructed by the answer sequence V produced by your program for each testcase. In other words, for each testcase, let l be the length of V , the grader prints $2^0 \cdot \text{memory}[V[0]] + 2^1 \cdot \text{memory}[V[1]] + \dots + 2^{l-1} \cdot \text{memory}[V[l - 1]]$.

After executing all test cases, the grader prints `number of instructions: X` where X is the number of instructions in your program.