D. Data Structure

In compute science, a stack s is a data structure maintaining a list of elements with two operations:

- s.`push`(e) appends an element e to the right end of the list,
- s.`pop`$()$ removes the rightmost element in the list and returns the removed element.

For convenience, Bobo denotes the number of elements in the stack s by `size`(s), and the rightmost element by `right`(s).

Bobo has m stacks $s_1, \dots, s_m$. Initially, the stack s_i contains k_i elements $a_{i,1}, \dots, a_{i,k_i}$ where $a_{i,j} \in \{1, \dots, n\}$. Furthermore, for each $e \in \{1, \dots, n\}$, the element e occurs in the m stacks **exactly twice**. Thus, $k_1 + \dots + k_m = 2n$.

A sorting plan of length l consists of l pairs $(f_1, t_1), \dots, (f_l, t_l)$. To execute a sorting plan, for each $i \in \{1, \dots, l\}$ in the increasing order, Bobo performs s_{t_i}.`push`$(s_{f_i}$.`pop`$())$.

A sorting plan is *valid* if the length does not exceed $\lfloor \frac{3n}{2} \rfloor$, and for each $i \in \{1, \dots, l\}$, $1 \le f_i, t_i \le m$, $f_i \ne t_i$. Before the i-th operation,

- `size`$(s_{f_i}) > 0$,
- `size`$(s_{t_i}) < 2$,
- either `size`$(s_{t_i}) = 0$ or `right`$(s_{f_i}) =$ `right`(s_{t_i}).

Also, after the execution of a *valid* sorting plan, each of the m stacks either is empty or contains the two copies of the same element.

Find a *valid* sorting plan, given the initial configuration of the m stacks.

Input

The input consists of several test cases terminated by end-of-file. For each test case,

The first line contains two integers n and m.

For the next m lines, the i-th line contains an integer k_i, and k_i integers $a_{i,1}, \dots, a_{i,k_i}$.

- $1 \le n \le m \le 2 \times 10^5$
- $0 \le k_i \le 2$ for each $1 \le i \le m$
- $1 \le a_{i,j} \le n$ for each $1 \le i \le m$, $1 \le j \le k_i$
- For each $1 \le e \le n$, there exists exactly two (i, j) where $1 \le j \le k_i$ and $a_{i,j} = e$.
- In each input, the sum of m does not exceed 2×10^5.

Output

For each test case, if there exists a *valid* sorting plan, output an integer l, which denotes the length of the sorting plan. Followed by l lines, the i-th line contains two integers f_i and t_i. Otherwise, output `-1`.

If there are multiple *valid* sorting plans, any of them is considered correct.

Sample Input

```
2 3
2 1 2
2 1 2
0
1 1
2 1 1
3 4
2 1 3
2 2 3
1 1
1 2
```

Sample Output

```
3
1 3
2 3
2 1
0
-1
```

Note

For the first test cases,

- Initially, $s_1 = [1, 2]$, $s_2 = [1, 2]$, $s_3 = [\,]$.
- After $s_3.\texttt{push}(s_1.\texttt{pop}())$. $s_1 = [1]$, $s_2 = [1, 2]$, $s_3 = [2]$.
- After $s_3.\texttt{push}(s_2.\texttt{pop}())$, $s_1 = [1]$, $s_2 = [1]$, $s_3 = [2, 2]$.
- After $s_1.\texttt{push}(s_2.\texttt{pop}())$, $s_1 = [1, 1]$, $s_2 = [\,]$, $s_3 = [2, 2]$.

For the second test case, the initial configuration is already sorted.