Lockpicking

The Hungarian State Treasury replaced the locking mechanism on one of their safes. This new lock can only be opened by custom keycards. The verification of a keycard follows a special protocol.

The **lock** has N internal **states** numbered from 0 to $N-1$. For each i from 0 to $N-1$, inclusive, there is a *bit* $A[i]$ and there are two states $S[i][0]$ and $S[i][1]$ associated with state i. A *bit* is a binary digit that is either 0 or 1. States $S[i][0]$ and $S[i][1]$ may coincide, and a state may be associated with itself.

Similarly to this, a **keycard** has M **states** numbered from 0 to $M-1$, and for each j from 0 to $M-1$, bit $B[j]$ and states $T[j][0]$ and $T[j][1]$ are associated with state j. Hereafter, lock states and keycard states are both referred to as states.

In the verification process, the lock gets paired up with a keycard. Both of them are capable of **output**ting bits, as well as reading bits from the output of the *other*. At the start of the process, the lock sets to a specific **initial state** i_0. The keycard also sets to initial state j_0, specific to the keycard. They repeat the following steps at most 10^7 times:

1. Each of them outputs a single bit according to its current state. If the lock is in state i, it outputs the bit $A[i]$. Analogously, the keycard outputs $B[j]$ if it is in state j.
2. They read the bit that was outputted by the other. The lock reads $B[j]$, and the keycard reads $A[i]$. If the two bits are different, they register an **error**.
3. The devices change their states based on the bit they read and the states associated with the current one. The lock enters state $S[i][B[j]]$, and the keycard enters state $T[j][A[i]]$.

If, at any point over the verification process, the number of errors reaches K, the verification fails, and the process terminates. Otherwise, if they complete 10^7 iterations without registering at least K errors, the verification succeeds, and the lock opens.

Upon setting up the new lock, the mechanics made a mistake: they forgot to specify state i_0, the initial state used in the verification process. As a result, whenever the lock gets paired up with a keycard, it is set to an arbitrary (unknown) initial state.

Your task is to construct a keycard capable of opening the lock despite the mistake.

Implementation Details

You should implement the following procedure.

```
void construct_card(int N, int[] A, int[][] S)
```

- N: the number of internal states of the lock.
- A: array of length N specifying the bits associated with the states of the lock.
- S: array of length N containing arrays of length 2. Specifies the two states associated with the states of the lock.
- This procedure should construct a keycard capable of opening the lock irrespective of its initial state.
- This procedure is called exactly once for each test case.

This procedure must call the following procedure exactly once to construct a keycard:

```
void define_states(int M, int[] B, int[][] T, int j0)
```

- M: the number of internal states of the keycard.
- B: array of length M specifying the bits associated with the states of the keycard.
- T: array of length M containing arrays of length 2. Specifies the two states associated with the states of the keycard.
- $j0$: a number representing the initial state of the keycard.
- This procedure constructs a keycard with M internal states. For each j from 0 to $M-1$, inclusive, bit $B[j]$ and states $T[j][0]$ and $T[j][1]$ are associated with state j. The initial state of the keycard is set to $j0$.
- This procedure should be called exactly once by the procedure `construct_card`.

Constraints on the parameters of `define_states` are given in the Constraints section below.

Example

Consider a lock that has $N = 2$ states. Bit $A[0] = 0$ and states $S[0][0] = 1$ and $S[0][1] = 0$ are associated with state 0, and bit $A[1] = 1$ and states $S[1][0] = 1$ and $S[1][1] = 0$ are associated with state 1. For this example, suppose that the value of K is 2.

The procedure `construct_card` is called the following way.

```
construct_card(2, [0, 1], [[1, 0], [1, 0]])
```

First, consider a keycard with $M = 1$ state where $B[0] = 1$, $T[0][0] = T[0][1] = 0$ and the initial state is $j_0 = 0$.

In the case when the unknown initial state of the lock is 1, the steps of the verification process are shown in the following table. The states of the lock and the keycard are denoted by i and j, respectively.

Step	i	$A[i]$	j	$B[j]$	Errors so far	$S[i][B[j]]$	$T[j][A[i]]$
0	1	1	0	1	0	$S[1][1] = 0$	$T[0][1] = 0$
1	0	0	0	1	1	$S[0][1] = 0$	$T[0][0] = 0$
2	0	0	0	1	2	$S[0][1] = 0$	$T[0][0] = 0$

Note that the last two columns show the state of the lock and the keycard for the next step.

We see that there is no error in step 0 as we have $A[i] = B[j]$. Following step 0, there is an error in step 1, as well as in step 2, so the verification fails, and the process terminates. Therefore, this keycard is not capable of opening the lock.

Consider a keycard with $M = 2$ states and suppose that $B = [0, 1]$ and $T = [[1, 1], [0, 0]]$ and the initial state is $j_0 = 0$.

If the initial state of the lock is 0, the verification goes as follows.

Step	i	$A[i]$	j	$B[j]$	Errors so far	$S[i][B[j]]$	$T[j][A[i]]$
0	0	0	0	0	0	$S[0][0] = 1$	$T[0][0] = 1$
1	1	1	1	1	0	$S[1][1] = 0$	$T[1][1] = 0$
2	0	0	0	0	0	$S[0][0] = 1$	$T[0][0] = 1$

At this point, we may deduce that the verification process will succeed without errors.

If the initial state of the lock is 0, the verification goes as follows.

Step	i	$A[i]$	j	$B[j]$	Errors so far	$S[i][B[j]]$	$T[j][A[i]]$
0	1	1	0	0	1	$S[1][0] = 1$	$T[0][1] = 1$
1	1	1	1	1	1	$S[1][1] = 0$	$T[1][1] = 0$
2	0	0	0	0	1	$S[0][0] = 1$	$T[0][0] = 1$

At this point, we may deduce that the verification process will succeed without additional errors.

Therefore, this keycard can open the lock irrespective of its initial state. The procedure may call `define_states` in the following way.

```
define_states(2, [0, 1], [[1, 1], [0, 0]], 0)
```

After `define_states` completes, procedure `construct_card` should return.

Constraints

- $2 \leq N \leq 150$
- $0 \leq A[i] \leq 1$ (for each i such that $0 \leq i < N$)
- $0 \leq S[i][0] < N$ and $0 \leq S[i][1] < N$ (for each i such that $0 \leq i < N$)
- $1 \leq M \leq 50\,000$
- $0 \leq B[j] \leq 1$ (for each j such that $0 \leq j < M$)
- $0 \leq T[j][0] < M$ and $0 \leq T[j][1] < M$ (for each j such that $0 \leq j < M$)
- $K = N$

Subtasks

1. (21 points) $N = 2$
2. (30 points) $N \leq 30$ and $S[i][0] = S[i][1]$ for each i from 0 to $N - 1$, inclusive.
3. (20 points) $N \leq 30$ and $K = N^2$
4. (29 points) No additional constraints.

Sample Grader

The sample grader reads the input in the following format:

- line 1: N K
- line $2 + i$ ($0 \leq i < N$): $A[i]$ $S[i][0]$ $S[i][1]$

If the sample grader detects a protocol violation, the output of the sample grader is `Protocol Violation: <MSG>`, where `<MSG>` is one of the error messages:

- `missing call`: the procedure `construct_card` returned without making a call to `define_states`.
- `too many calls`: the procedure `construct_card` made more than one call to `define_states`.
- `invalid number`: M is not an integer between 1 and $50\,000$.
- `invalid array`: the length of array B or the length of array T is different from M, or there exists an index j ($0 \leq j < M$) such that the length of $T[j]$ is not 2.
- `invalid bit`: there exists an index j ($0 \leq j < M$) such that $B[j]$ is not 0 or 1.
- `invalid state`: there exists an index j ($0 \leq j < M$) such that at least one of $T[j][0]$ and $T[j][1]$ is not an integer between 0 and $M - 1$.
- `invalid initial state`: $j0$ is not an integer between 0 and $M - 1$.

Otherwise, the sample grader produces two outputs.

First, the sample grader prints the constructed keycard in the following format:

- line 1: M $j0$
- line $2 + j$ ($0 \leq j < M$): $B[j]$ $T[j][0]$ $T[j][1]$

Second, the sample grader writes a file `lockpicking.bin` in the working directory. This file serves as the input of the testing tool described in the following section.

Testing Tool

The attachment package for this task contains a file named `display.py`. When invoked, this Python script simulates the verification process between a lock and a keycard. For this, the binary file `lockpicking.bin` must be present in the working directory.

The initial state of the lock is 0 in the first simulation. After the simulation completes, a simple graphical interface shows up. The main features are as follows.

- You can see the overall result of the verification (for the current initial state of the lock) at the bottom of the window.
- You can browse through the steps of the process by clicking the arrow buttons or pressing their hotkeys.
- To set the initial state of the lock, use the `Init Lock` button.
- The `Reload` button reloads the configuration file `lockpicking.bin`. It is useful if the contents of `lockpicking.bin` have changed.

The last two operations automatically rerun the simulation and update the GUI.

To invoke the script, execute the following command.

```
python3 display.py
```