Dausend Inselen

D'Dausend Insele sinn eng wonnerschéi Grupp vun Inselen um Stauséi. Se besteet aus N Inselen, nummeréiert vun 0 bis $N-1$.

Et ginn M Kanuen, nummeréiert vun 0 bis $M-1$, mat deenen en kann tëschent zwou Insele fueren. Fir all i mat $0 \leq i \leq M-1$, ka Kanu i entweder un der Insel $U[i]$ oder $V[i]$ ugedockt sinn, an et ka benotzt gi fir tëschent den Inselen $U[i]$ an $V[i]$ ze fueren. Dat heescht, wann de Kanu un der Insel $U[i]$ ugedockt ass, kann en et huele fir vun Insel $U[i]$ op d'Insel $V[i]$ ze fueren. Dann ass d'Kanu un der Insel $V[i]$ ugedockt. D'selwecht, wann de Kanu un der Insel $V[i]$ ugedockt ass, kann en et huele fir vun der Insel $V[i]$ op d'Insel $U[i]$ ze fueren. Duerno ass d'Kanu un der Insel $U[i]$ ugedockt. Um Ufank, ass de Kanu un der Insel $U[i]$ ugedockt. Et ass méiglech datt méi Kanuen tëschent de selwechten Insele kennen hin an hier fueren. Et ass och méiglech datt méi Kanuen un der selwechter Insel ugedockt sinn.

Aus Sécherheetsgrënn, muss en Kanu no all Faart an d'Maintenance, an dowéinst kann et net zweemol hannertenee benotzt ginn. Also, muss nodeems Kanu i benotzt ginn ass, en anere Kanu benotzt ginn éier Kanu i erëm benotzt ka ginn.

D'Ketti well en Tour duerch e puer Insele plangen. Séin Tour ass nëmmen dann **an der Rei** wa folgend Konditiounen erfëllt sinn.

- Et fänkt séin Tour op der Insel 0 un an en hält och do op.
- Et besicht wéinstens eng Insel déi net d'Insel 0 ass.
- Um Enn vu sengem Tour, muss all Kanu op der Insel sinn op der en wuer éier säin Tour ugefaangen huet. Dat heescht, Kanu i muss un der Insel $U[i]$ ugedockt sinn, fir all i mat $0 \leq i \leq M-1$.

Hëllef dem Ketti en Tour den an der Rei ass ze fannen, wou et maximal $2\,000\,000$ mol en Kanu benotzt, oder fann eraus op keen sou en Tour existéiert.

It can be proven that under the constraints specified in this task (see Constraints section), if a valid journey exists, there also exists a valid journey that does not involve sailing more than $2\,000\,000$ times.

Implementation Details

You should implement the following procedure:

```
union(bool, int[]) find_journey(int N, int M, int[] U, int[] V)
```

- N: the number of islands.
- M: the number of canoes.
- U, V: arrays of length M describing the canoes.
- This procedure should return either a boolean or an array of integers.
    - If no valid journey exists, the procedure should return `false`.
    - If a valid journey exists, you have two options:
        - To be awarded the full score, the procedure should return an array of at most $2\,000\,000$ integers representing a valid journey. More precisely, the elements of this array should be the numbers of the canoes that are used in the journey (in the order they are used).
        - To be awarded a partial score, the procedure should return `true`, an array of more than $2\,000\,000$ integers, or an array of integers not describing a valid journey. (See the Subtasks section for more details.)
- This procedure is called exactly once.

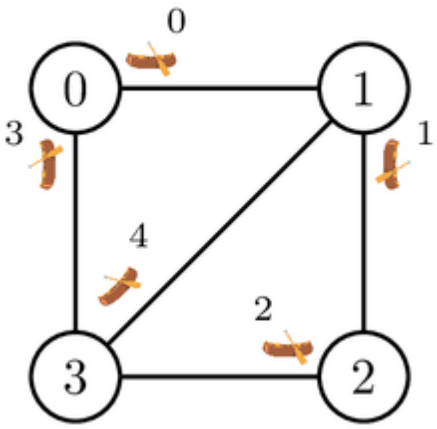
Examples

Example 1

Consider the following call:

```
find_journey(4, 5, [0, 1, 2, 0, 3], [1, 2, 3, 3, 1])
```

The islands and canoes are shown in the picture below.



One possible valid journey is as follows. Bu Dengklek first sails canoes 0, 1, 2, and 4 in that order. As a result, she is at island 1. After that, Bu Dengklek can sail canoe 0 again as it is currently docked at island 1 and the last canoe she used is not canoe 0. After sailing canoe 0 again, Bu Dengklek is

now at island 0. However, canoes 1, 2 and 4 are not docked at the same islands as they were before the journey. Bu Dengklek then continues her journey by sailing canoes 3, 2, 1, 4, and 3 again. Bu Dengklek is back at island 0 and all the canoes are docked at the same islands as before the journey.

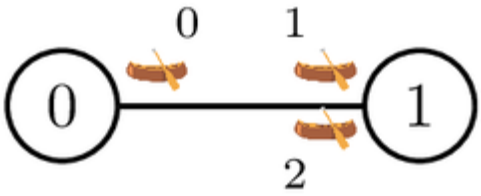
Therefore, the returned value $[0, 1, 2, 4, 0, 3, 2, 1, 4, 3]$ represents a valid journey.

Example 2

Consider the following call:

```
find_journey(2, 3, [0, 1, 1], [1, 0, 0])
```

The islands and canoes are shown in the picture below.



Bu Dengklek can only start by sailing canoe 0, after which she can sail either canoe 1 or 2. Note that she cannot sail canoe 0 twice in a row. In both cases, Bu Dengklek is back at island 0. However, the canoes are not docked at the same islands as they were before the journey, and Bu Dengklek cannot sail any canoe afterwards, as the only canoe docked at island 0 is the one she has just used. As there is no valid journey, the procedure should return `false`.

Constraints

- $2 \leq N \leq 100\,000$
- $1 \leq M \leq 200\,000$
- $0 \leq U[i] \leq N - 1$ and $0 \leq V[i] \leq N - 1$ (for each i such that $0 \leq i \leq M - 1$)
- $U[i] \neq V[i]$ (for each i such that $0 \leq i \leq M - 1$)

Subtasks

1. (5 points) $N = 2$
2. (5 points) $N \leq 400$. For each pair of distinct islands x and y ($0 \leq x < y \leq N - 1$), there are exactly two canoes that can be used to sail between them. One of them is docked at island x, and the other one is docked at island y.

3. (21 points) $N \leq 1000$, M is even, and for each **even** i such that $0 \leq i \leq M-1$, canoes i and $i+1$ can both be used to sail between islands $U[i]$ and $V[i]$. Canoe i is initially docked at island $U[i]$ and canoe $i+1$ is initially docked at island $V[i]$. Formally, $U[i] = V[i+1]$ and $V[i] = U[i+1]$.
4. (24 points) $N \leq 1000$, M is even, and for each **even** i such that $0 \leq i \leq M-1$, canoes i and $i+1$ can both be used to sail between islands $U[i]$ and $V[i]$. Both canoes are initially docked at island $U[i]$. Formally, $U[i] = U[i+1]$ and $V[i] = V[i+1]$.
5. (45 points) No additional constraints.

For each test case in which a valid journey exists, your solution:

- gets full points if it returns a valid journey,
- gets 35% of the points if it returns `true`, an array of more than $2\,000\,000$ integers, or an array that does not describe a valid journey,
- gets 0 points otherwise.

For each test case in which a valid journey does not exist, your solution:

- gets full points if it returns `false`,
- gets 0 points otherwise.

Note that the final score for each subtask is the minimum of the points for the test cases in the subtask.

Sample Grader

The sample grader reads the input in the following format:

- line 1: N M
- line $2+i$ ($0 \leq i \leq M-1$): $U[i]$ $V[i]$

The sample grader prints your answers in the following format:

- If `find_journey` returns a `bool`:
  - line 1: 0
  - line 2: 0 if `find_journey` returns `false`, or 1 otherwise.
- If `find_journey` returns an `int[]`, denote the elements of this array by $c[0], c[1], \ldots c[k-1]$. The sample grader prints:
  - line 1: 1
  - line 2: k
  - line 3: $c[0]$ $c[1]$ $\ldots$ $c[k-1]$