Game

После открытия n планет, занумерованных последовательными целыми числами от 0 до $n-1$, Фараоны начали строительство транспортной системы между ними, основанной на **односторонних телепортах**.

Для каждого телепорта определена планета входа и планета выхода. Когда турист заходит в телепорт на планете входа, он попадает на планету выхода. При этом возможны случаи, когда для одного телепорта планета входа и планета выхода совпадают. Телепорт с планетой входа u и планетой выхода v обозначается как (u, v).

Для популяризации новой системы телепортов Фараоны создали игру, в которую туристы могут играть во время путешествия по транспортной системе. Турист может начать игру на любой планете. Планеты $0, 1, \ldots, k-1$ ($k \leq n$) называются **особенными планетами**. Каждый раз, когда турист посещает особенную планету, он получает памятный значок.

Изначально для каждого i ($0 \leq i \leq k-2$) существует телепорт $(i, i+1)$. Эти $k-1$ телепортов называются **базовыми телепортами**.

Новые телепорты добавляются по одному. Заметим, что может случиться, что после добавления нового телепорта у туриста появляется возможность получить бесконечное количество значков. Более формально, это происходит в случае, когда существует последовательность планет $w[0], w[1], \ldots, w[t]$, удовлетворяющая следующим условиям:

- $1 \leq t$
- $0 \leq w[0] \leq k-1$
- $w[t] = w[0]$
- Для каждого i ($0 \leq i \leq t-1$) существует телепорт $(w[i], w[i+1])$.

Турист может использовать базовые телепорты и **любые** телепорты, которые были добавлены к этому моменту.

Ваша задача --- помочь Фараонам после добавления очередного телепорта проверять, может ли после него турист получить возможность собрать бесконечное количество значков.

Implementation details

Вы должны реализовать следующие функции:

```
init(int n, int k)
```

- n: количество планет.
- k: количество особенных планет.
- Эта функция вызывается только один раз. Все вызовы функции `add_teleporter` идут после вызова этой функции.

```
int add_teleporter(int u, int v)
```

- u и v: планета входа и планета выхода для добавляемого телепорта.
- Функция будет вызвана не более m раз (значение m задано в разделе Constraints).
- Функция должна возвращать 1, если после добавления телепорта (u, v) турист может получить бесконечное количество значков, и 0 в противном случае.
- После того, как эта функция вернёт 1, выполнение программы будет завершено.

Examples

Example 1

Рассмотрим первый пример:

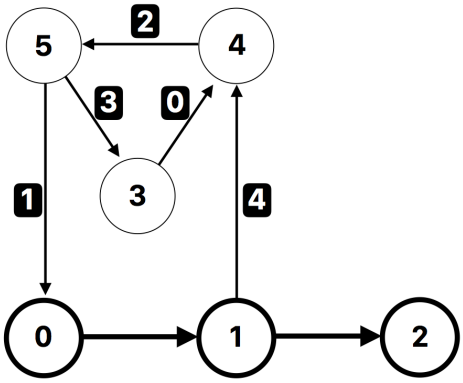
```
init(6, 3)
```

В этом примере всего 6 планет, 3 из которых являются особенными. Планеты 0, 1 и 2 являются особенными. Базовые телепорты --- $(0, 1)$ и $(1, 2)$.

Пусть идут следующие вызовы:

- (0) `add_teleporter(3, 4)`: Возвращаете 0.
- (1) `add_teleporter(5, 0)`: Возвращаете 0.
- (2) `add_teleporter(4, 5)`: Возвращаете 0.
- (3) `add_teleporter(5, 3)`: Возвращаете 0.
- (4) `add_teleporter(1, 4)`: После добавления этого телепорта можно получить бесконечное количество значков. Например, турист стартует на планете 0, после чего передвигается по планетам $1, 4, 5, 0, 1, 4, 5, 0, \ldots$ соответственно. Поэтому возвращаете 1, после чего обработка примера заканчивается.

Картинка ниже иллюстрирует пример. Особенные планеты и базовые телепорты выделены жирным. Телепорты, добавленные функцией `add_teleporter` занумерованы от 0 до 4 в порядке вызова соответствующей функции.



Example 2

Рассмотрим второй пример:

```
init(4, 2)
```

В этом примере всего 4 планеты, 2 из которых являются особенными. Планеты 0 и 1 являются особенными. Базовый телепорт --- $(0, 1)$.

Пусть добавляется телепорт:

- `add_teleporter(1, 1)`: после добавления телепорта $(1, 1)$, появляется возможность собрать бесконечное количество значков (например, стартуя на планете 1 и используя телепорт $(1, 1)$ бесконечное количество раз. Поэтому возвращаете 1, после чего обработка примера заканчивается.

Ещё примеры ввода/вывода содержатся в аттачменте к задаче.

Constraints

- $1 \leq n \leq 300\,000$
- $1 \leq m \leq 500\,000$
- $1 \leq k \leq n$

Для каждого вызова `add_teleporter`:

- $0 \leq u \leq n - 1$ и $0 \leq v \leq n - 1$
- Гарантируется, что телепорт с планеты u на планету v отсутствует перед добавлением телепорта (u, v).

Subtasks

1. (2 балла) $n = k$, $n \leq 100$, $m \leq 300$
2. (10 баллов) $n \leq 100$, $m \leq 300$
3. (18 баллов) $n \leq 1\,000$, $m \leq 5\,000$

4. (30 баллов) $n \leq 30\,000$, $m \leq 50\,000$, $k \leq 1\,000$
5. (40 баллов) Без дополнительных ограничений.

Sample grader

Пример проверяющего модуля читает данные из файла в следующем формате:

- строка 1: $n\ m\ k$
- строка $2 + i$ ($0 \leq i \leq m - 1$): $u[i]\ v[i]$

Проверяющий модуль сначала вызывает `init`, а затем последовательно вызывает `add_teleporter` от $u = u[i]$ и $v = v[i]$ для $i = 0, 1, \ldots, m - 1$.

Он выводит номер первого вызова `add_teleporter`, который возвращает 1 (число между 0 и $m - 1$ включительно), или m, если все вызовы `add_teleporter` вернули 0.

Если какой-то вызов `add_teleporter` вернул целое число, отличное от 0 или 1, проверяющий модуль выведет -1 и завершает выполнение.