ゲーム (Game)

ファラオは n 個の惑星を発見し，それぞれに 0 から $n-1$ までの番号を付けた．その後，惑星間の移動を可能にするため，ファラオは **一方通行の瞬間移動装置** の建設をはじめた．それぞれの瞬間移動装置には，出発地点の惑星と到着地点の惑星が定められており，旅行者が出発地点でその装置を使うと到着地点に瞬間移動する．出発地点と到着地点は同じ惑星かもしれないことに注意せよ．出発地点が惑星 u で到着地点が惑星 v の瞬間移動装置は (u, v) で表される．

瞬間移動装置の利用を促進するため，ファラオは旅行者向けにスタンプラリーのゲームを実施することにした．旅行者は，どの惑星からゲームを開始するかを自分で決めることができる．惑星 $0, 1, \ldots, k-1$ $(k \leq n)$ は **特別な惑星** とよばれ，旅行者がこれらの惑星に入るたびにスタンプを 1 回押してもらえる．

現時点では，それぞれの i $(0 \leq i \leq k-2)$ に対して，瞬間移動装置 $(i, i+1)$ がすでに建設されている．この $k-1$ 個の瞬間移動装置は **初期時点での瞬間移動装置** とよばれる．

これから，新しい瞬間移動装置が 1 つずつ追加されていく．すると，旅行者がスタンプを無限個得ることがいずれ可能になるかもしれない．このようになる条件は，以下をすべて満たす惑星の列 $w[0], w[1], \ldots, w[t]$ が存在することである．

- $1 \leq t$.
- $0 \leq w[0] \leq k-1$.
- $w[t] = w[0]$.
- それぞれの i $(0 \leq i \leq t-1)$ に対して，瞬間移動装置 $(w[i], w[i+1])$ が存在する．

ここで，旅行者は初期時点での瞬間移動装置に加えて，その時点までに追加された **すべての** 瞬間移動装置を使えることに注意せよ．

ファラオを助けるために，新しい瞬間移動装置が追加されるたびに，旅行者がスタンプを無限個得ることが可能であるかを判定せよ．

実装の詳細

以下の関数を実装せよ．

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

- n: 惑星の数．
- k: 特別な惑星の数．
- この関数は，関数 `add_teleporter` のどの呼び出しよりも前に，最初に 1 回だけ呼び出される．

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

- u, v: 追加される新しい瞬間移動装置の出発地点が惑星 u, 到着地点が惑星 v であることを表す.
- この関数は最大で m 回呼び出される（m の範囲については「制約」の項を参照せよ）.
- 瞬間移動装置 (u, v) が追加された後, 旅行者がスタンプを無限個得ることが可能になるならば, この関数は 1 を返さなければならない. それ以外の場合, この関数は 0 を返さなければならない.
- この関数が 1 を返すと, プログラムは実行を終了する.

入出力例

入出力例 1

以下の関数呼び出しを考える.

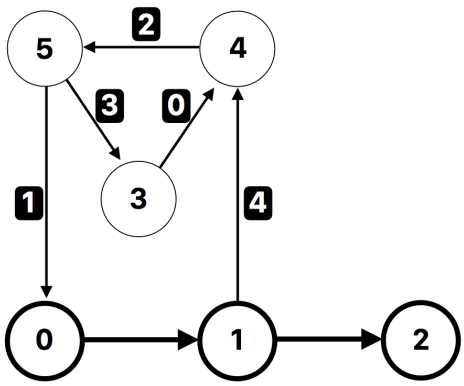
```
init(6, 3)
```

この例では 6 個の惑星があり, そのうち惑星 $0, 1, 2$ の 3 個が特別な惑星である. 初期時点での瞬間移動装置は $(0, 1)$ と $(1, 2)$ である.

その後, 以下のように関数が呼び出されることを考える.

- (0) `add_teleporter(3, 4)`: 関数は 0 を返さなければならない.
- (1) `add_teleporter(5, 0)`: 関数は 0 を返さなければならない.
- (2) `add_teleporter(4, 5)`: 関数は 0 を返さなければならない.
- (3) `add_teleporter(5, 3)`: 関数は 0 を返さなければならない.
- (4) `add_teleporter(1, 4)`: この時点で, 旅行者がスタンプを無限個得ることが可能になる. これは例えば, 旅行者が惑星 0 でゲームを開始し, その後惑星 $1, 4, 5, 0, 1, 4, 5, 0, \ldots$ と移動する, という方法で実現される. したがって, 関数は 1 を返さなければならない. その後, プログラムは実行を終了する.

この入力例は, 以下の図に対応している. 特別な惑星と初期時点での瞬間移動装置は太線で描かれている. 関数 `add_teleporter` で追加される瞬間移動装置は, 呼び出される順に 0 から 4 までのラベルが付けられている.



入出力例 2

以下の関数呼び出しを考える.

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

この例では 4 個の惑星があり, そのうち惑星 $0, 1$ の 2 個が特別な惑星である. 初期時点での瞬間移動装置は $(0, 1)$ のみである.

その後, 以下のように関数が呼び出されることを考える.

- `add_teleporter(1, 1)`: 瞬間移動装置 $(1, 1)$ を追加すると, 旅行者がスタンプを無限個得ることが可能になる. これは例えば, 旅行者が惑星 1 でゲームを開始し, 瞬間移動装置 $(1, 1)$ の使用を無限に繰り返すことで無限に惑星 1 に入り続ける, という方法で実現される. したがって, 関数は 1 を返さなければならない. その後, プログラムは実行を終了する.

入出力例はそれ以外にもあり, 添付のパッケージで得ることができる.

制約

- $1 \leqq n \leqq 300\,000$.
- $1 \leqq m \leqq 500\,000$.
- $1 \leqq k \leqq n$.

また, すべての `add_teleporter` の呼び出しについて, 以下の制約を満たす.

- $0 \leqq u \leqq n - 1$.
- $0 \leqq v \leqq n - 1$.
- 瞬間移動装置 (u, v) を追加する直前の時点で, 出発地点が惑星 u, 到着地点が惑星 v である瞬間移動装置は存在しない.

小課題

1. (2 点) $n = k$, $n \leqq 100$, $m \leqq 300$.
2. (10 点) $n \leqq 100$, $m \leqq 300$.
3. (18 点) $n \leqq 1\,000$, $m \leqq 5\,000$.
4. (30 点) $n \leqq 30\,000$, $m \leqq 50\,000$, $k \leqq 1\,000$.
5. (40 点) 追加の制約はない.

採点プログラムのサンプル

採点プログラムのサンプルは, 入力を以下の形式で読み込む.

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

採点プログラムのサンプルは, 最初に関数 `init` を呼び出し, その後 $i = 0, 1, \ldots, m - 1$ の順に関数 `add_teleporter` を $u = u[i], v = v[i]$ を引数として呼び出す.

採点プログラムのサンプルは，最初に 1 が返された `add_teleporter` の呼び出しが何番目か（0 以上 $m-1$ 以下の整数）を出力する．ただし，すべての `add_teleporter` の呼び出しが 0 を返した場合は，m を出力する．

もし `add_teleporter` が $0, 1$ 以外の値を返した場合は，採点プログラムのサンプルは -1 を出力し，プログラムはただちに実行を終了する．