```
'''
    Time complexity: O(V + E)
    Space complexity: O(V^2)

    where V is the number of vertices in the input graph and
    E is the number of edges in the input graph

'''

import queue
from sys import stdin, setrecursionlimit
setrecursionlimit(10**6)
class Graph:
    def __init__(self, nVertices):
        self.nVertices = nVertices
        self.adjMatrix = [[0 for i in range(nVertices)] for j in range(nVertices)]

    def addEdge(self, v1, v2):
        self.adjMatrix[v1][v2] = 1
        self.adjMatrix[v2][v1] = 1

    def removeEdge(self):
        if self.containsEdge(v1, v2) is False :
            return
        self.adjMatrix[v1][v2] = 0
        self.adjMatrix[v2][v1] = 0

    def containsEdge(self, v1, v2):
        if self.adjMatrix[v1][v2] > 0:
            return True
        else:
            return False

    def __str__(self):
        return str(self.adjMatrix)



    def __bfs(self, sv, visited) :
        q = queue.Queue()
        q.put(sv)

        visited[sv] = True

        while q.empty() is False :
            u = q.get()
            print(u, end = ' ')
            for i in range(self.nVertices) :
                if self.adjMatrix[u][i] > 0 and visited[i] is False :
                    q.put(i)
                    visited[i] = True


    def bfs(self) :
        visited = [False for i in range(self.nVertices)]

        for i in range(self.nVertices) :
            if visited[i] is False :
                self.__bfs(i, visited)


    def __getPathBFS(self, sv, ev, visited) :
        mapp = {}
        q = queue.Queue()

        if self.adjMatrix[sv][ev] == 1 and sv == ev :
            ans = []
            ans.append(sv)
            return ans

        q.put(sv)
        visited[sv] = True

        while q.empty() is False :
            front = q.get()

            for i in range(self.nVertices) :
                if self.adjMatrix[front][i] == 1 and visited[i] is False :
                    mapp[i] = front
                    q.put(i)

                    visited[i] = True

                    if i == ev :
                        ans = []
                        ans.append(ev)
                        value = mapp[ev]

                        while value != sv :
                            ans.append(value)
                            value = mapp[value]

                        ans.append(value)
                        return ans

        return []
    def getPathBFS(self, sv, ev) :
        visited = [False for i in range(self.nVertices)]
        return self.__getPathBFS(sv, ev, visited)


    def dfs(self, sv, visited) :
        visited[sv] = True

        for i in range(self.nVertices) :
            if self.adjMatrix[sv][i] == 1 and not visited[i] :
                self.dfs(i, visited)
                visited[i] = True


    def isConnected(self) :
        visited = [False for i in range(self.nVertices)]
        self.dfs(0, visited)

        for boolVal in visited :
            if not boolVal:
                return False
        return True


# Main
li = stdin.readline().strip().split()
V = int(li[0])
E = int(li[1])
if V==0:
    print('true')
else:
    g = Graph(V)

    for i in range(E) :
        arr = stdin.readline().strip().split()
        fv = int(arr[0])
        sv = int(arr[1])
        g.addEdge(fv, sv)

    print()
    if g.isConnected() :
        print('true')
    else :
        print('false')
```