```
'''
    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
'''

from sys import stdin
import queue
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 __hasPath(self, sv, ev, visited) :
        if sv == ev :
            return True

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

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

            for i in range(self.nVertices) :
                if self.adjMatrix[u][i] == 1 and not visited[i]:
                    if i == ev :
                        return True

                    q.put(i)
                    visited[i] = True
        return False

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



# Main
li = stdin.readline().strip().split()
V = int(li[0])
E = int(li[1])

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)

li = stdin.readline().strip().split()
sv = int(li[0])
ev = int(li[1])

if g.hasPath(sv, ev) :
    print('true')
else :
    print('false')
```