```
'''

    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, setrecursionlimit
setrecursionlimit(10**6)
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 __getPathDFS(self, sv, ev, visited) :
        if sv == ev :
            return list([sv])

        visited[sv] = True

        for i in range(self.nVertices) :
```

```
            if self.adjMatrix[sv][i] == 1 and not visited[i] :
                li = self.__getPathDFS(i, ev, visited)

                if li != None :
                    li.append(sv)
                    return li

        return None



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

li = g.getPathDFS(sv, ev)

if li != None :
    for element in li :
        print(element, end = ' ')
```