**Problem: Sorting and Graph**

Alice and Bob are playing with a graph.  
Bob gives Alice a challenge to find the lexicographically smallest topological sorting of the given graph.  
As you are the best friend of Alice help Alice to solve this challenge.  
You are given a directed graph of V vertices ( Numbered from 1 to V ) and E edges.  
find the lexicographically smallest topological sorting of the given graph.  
Note:- Print -1 if the topological sort does not exists for the graph.

**Input Format**

* The first line of input contains two integers V and E — the number of vertices and the number of edges respectively.
* The following E lines contain two integers ui and uj, indicating that there is a directed edge from ui to uj.

**Constraints**

* 2 <= V <= 10000
* 1 <= E <= 2V(V−1)​
* 1 <= ui,uj <= V
* The given graph is a directed graph.

**Output Format**

Print V vertices of the graph in the lexicographically smallest topological sorting. Print -1 if the topological sort does not exist for the graph.

**Sample Input**

4 3

2 1

3 4

2 4

**Sample Output**

2 1 3 4

**Explanation**

The following five topological sorts exist for the given graph: (2,1,3,4), (2,3,1,4), (2,3,4,1), (3,2,1,4), (3,2,4,1). The lexicographically smallest among them is (2,1,3,4).

**Solution**

To solve this problem, we use Kahn's algorithm for topological sorting with a priority queue to ensure the lexicographically smallest order.

Here is the Python implementation:

python

import heapq

from collections import defaultdict, deque

def lexicographical\_topological\_sort(V, E, edges):

# Create adjacency list and in-degree array

graph = defaultdict(list)

in\_degree = [0] \* (V + 1)

for u, v in edges:

graph[u].append(v)

in\_degree[v] += 1

# Min-heap for nodes with in-degree 0

min\_heap = []

for i in range(1, V + 1):

if in\_degree[i] == 0:

heapq.heappush(min\_heap, i)

result = []

while min\_heap:

u = heapq.heappop(min\_heap)

result.append(u)

for v in graph[u]:

in\_degree[v] -= 1

if in\_degree[v] == 0:

heapq.heappush(min\_heap, v)

if len(result) == V:

return result

else:

return [-1]

# Input reading

V, E = map(int, input().split())

edges = [tuple(map(int, input().split())) for \_ in range(E)]

# Processing and Output

result = lexicographical\_topological\_sort(V, E, edges)

print(" ".join(map(str, result)))

**Test Cases**

**Test Case 1**

**Input**

4 3

2 1

3 4

2 4

**Output**

2 1 3 4

**Test Case 2**

**Input**

2 3

1 2

1 2

2 1

**Output**

-1

**Test Case 3**

**Input**

6 6

1 2

2 3

4 3

4 5

5 6

6 4

**Output**

diff

-1

**Test Case 4**

**Input**

10 20

5 10

10 3

9 10

6 2

8 6

7 6

2 5

8 4

8 5

2 3

4 1

9 3

7 10

1 3

5 9

7 2

5 3

1 9

6 9

8 3

**Output**

7 8 4 1 6 2 5 9 10 3

**Test Case 5**

**Input**

10 20

2 7

1 10

9 5

9 7

10 2

6 10

2 9

1 9

1 4

4 5

8 5

10 5

1 3

1 2

3 6

5 7

9 4

8 9

10 7

6 2

**Output**

1 3 6 8 10 2 9 4 5 7