#BFS

from collections import defaultdict, deque

class Graph:

def \_\_init\_\_(self):

self.graph = defaultdict(list) ## self.graph is where the connections between nodes are stored.

##Adds an edge between node u and node v.

def addEdge(self, u, v):

self.graph[u].append(v)

self.graph[v].append(u)

##visited: remembers the nodes we already saw.

##queue: remembers which nodes to visit next.

##start: starting node.

##goal: target node we want to reach.

def BFS(self, start, goal):

visited = set()

queue = deque([start])

visited.add(start)

while queue:

node = queue.popleft() ##While the queue is not empty:Take out the front node (node) to process it.

print(node, end=" ")

if node == goal:

print("\nGoal node reached!")

return

for neighbor in self.graph[node]:

if neighbor not in visited:

visited.add(neighbor)

queue.append(neighbor)

print("\nGoal node not reachable.")

# Driver code

if \_\_name\_\_ == '\_\_main\_\_':

g = Graph()

n = int(input("Enter number of edges: "))

print("Enter edges in the format: from to")

for \_ in range(n):

u, v = map(int, input().split())

g.addEdge(u, v)

start = int(input("Enter start node: "))

goal = int(input("Enter goal node: "))

print(f"BFS traversal from node {start} to find node {goal}:")

g.BFS(start, goal)