```python
class Node:
    def __init__(self, state, parent=None, depth=0):
        self.state = state      # Current state
        self.parent = parent    # Parent node
        self.depth = depth      # Depth of the node

    def path(self):
        node, p = self, []
        while node:
            p.append(node.state)
            node = node.parent
        return p[::-1]  # Return reversed path


def depth_limited_search(start, goal, depth_limit):
    """
    Perform a depth-limited search up to a given depth limit.
    """
    return recursive_dls(Node(start), goal, depth_limit)


def recursive_dls(node, goal, depth_limit):
    """
    Recursive helper function for depth-limited search.
    """
    if node.state == goal:
        return node.path()  # Goal found, return the path

    elif node.depth == depth_limit:
        return None  # Depth limit reached, return failure

    else:
        for child_state in get_children(node.state):
            child_node = Node(child_state, node, node.depth + 1)
            result = recursive_dls(child_node, goal, depth_limit)
            if result is not None:
                return result

    return None  # Return failure if goal not found


def iterative_deepening_search(start, goal, max_depth):
    """
    Perform an iterative deepening search by gradually increasing the depth limit.
    """
    for depth in range(max_depth + 1):
        result = depth_limited_search(start, goal, depth)
        if result is not None:
            return result  # Return the found path

    return None  # If no solution is found


def get_children(state):
    """
    This function should return the list of children (neighboring states) for the given state.
    It depends on the problem you're solving, so you'll need to modify it.
    For example, this can be a graph traversal where 'state' is a node and 'get_children' gives its neighbors.
    """
    # Example for a simple graph (replace with the actual problem):
    graph = {
        'A': ['B', 'C'],
        'B': ['D', 'E'],
        'C': ['F'],
        'D': [],
        'E': ['G', 'H'],
        'F': [],
        'G': [],
        'H': []
    }
    return graph.get(state, [])


# Example usage:
start = 'A'  # Starting node
goal = 'G'   # Goal node
max_depth = 5  # Set the maximum depth limit

result = iterative_deepening_search(start, goal, max_depth)
if result:
    print("Goal found. Path:", result)
else:
    print("Goal not found within depth limit.")
```