```
from collections import deque

def iterative_deepening_search(graph, start, goal):
    """
    Performs an iterative deepening search on a graph.

    Args:
        graph: A dictionary representing the graph where keys are nodes and values are lists of neighbors.
        start: The starting node.
        goal: The goal node.

    Returns:
        A list representing the path from the start node to the goal node, or None if no path exists.
    """

    depth_limit = 0
    while True:
        path = depth_limited_search(graph, start, goal, depth_limit)
        if path is not None:
            return path
        depth_limit += 1


def depth_limited_search(graph, start, goal, depth_limit):

    visited = set()
    stack = deque([(start, [start])])

    while stack:
        node, path = stack.pop()

        if node == goal:
            return path

        if depth_limit == 0:
            continue

        if node not in visited and len(path) <= depth_limit:
          visited.add(node)
          for neighbor in graph.get(node, []):
              if neighbor not in visited:
                stack.append((neighbor, path + [neighbor]))
    return None


graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

start_node = 'A'
goal_node = 'F'

path = iterative_deepening_search(graph, start_node, goal_node)

if path:
    print(f"Path from {start_node} to {goal_node}: {path}")
else:
    print(f"No path found from {start_node} to {goal_node}")
```

```
Path from A to F: ['A', 'C', 'F']
```