```
def dfs(graph, start, goal, max_depth, depth=0, visited
=None, path=None):
    if visited is None:
        visited = set()
    if path is None:
        path = []

    visited.add(start)
    path = path + [start]

    if start == goal:
        return path

    if depth >= max_depth:
        return None

    for neighbor in graph[start]:
        if neighbor not in visited:
            new_path = dfs(graph, neighbor, goal,
max_depth, depth + 1, visited, path)
            if new_path:
                return new_path

    return None

def dfid(graph, start, goal):
    max_depth = 0
    while True:
        result = dfs(graph, start, goal, max_depth)
        if result is not None:
            return result
        max_depth += 1

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

#FOR TREE
graph = {
    'A': ['B', 'C','D'],
    'B': ['E', 'F'],
    'C': ['G','H'],
    'D':[],
    'E':[],
    'F':[],
    'G':[],
    'H':[]
}

start_node = input("Enter the start node: ").strip().
upper()
goal_node = input("Enter the goal node: ").strip().
upper()

print("DFID Path:")
if start_node not in graph or goal_node not in graph:
    print("Start node or goal node not found in the
graph.")
else:
    path = dfid(graph, start_node, goal_node)
    if path:
        print("Path from", start_node, "to", goal_node
, ":", ' -> '.join(path))
    else:
        print("Path from", start_node, "to", goal_node
, "not found.")
```