# AI Lab 04: Uninformed Search Ops

## Mission 1: The Time-Bomb Labyrinth

A hero is trapped in a labyrinth with a ticking time bomb. You must find if a path to the exit exists within a specific number of moves (depth limit) before it's too late! Implement Depth-Limited Search to solve this.

### The Intel (Graph)

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

### Sample Inputs & Expected Outputs

* **Input:** start='A', goal='F', limit=2
* **Expected Output:** ['A', 'C', 'F']
* **Input:** start='A', goal='F', limit=1
* **Expected Output:** None (or a message like "Goal not reachable within limit")

### The Approach (Your Blueprint)

def depth\_limited\_search(graph, start, goal, limit):...  
 # This is the main function that initiates the search.  
 # It should call a recursive helper function to perform the DLS.  
 # It will return the path if found, otherwise None.  
 pass  
  
def dls\_recursive(node, goal, graph, limit, path):  
 # This recursive function explores the graph.  
 # 1. Add the current node to the path.  
 # 2. Check if the current node is the goal. If yes, return the path.  
 # 3. Check if the depth limit is reached. If yes, stop exploring this path.  
 # 4. Iterate through the neighbors of the current node.  
 # 5. Recursively call this function for each neighbor with a decremented limit.  
 # 6. If a recursive call returns a valid path, propagate it up.  
 # 7. If no path is found from any neighbor, return None.

## Mission 2: Escape the Hydra Base

You've infiltrated a Hydra base, represented as a graph of interconnected rooms. Find any path from your entry point to the exit. The base is vast, so you need a systematic way to explore every possible corridor to its end before backtracking. Use Depth-First Search.

### The Intel (Graph)

graph = {  
 'Entry': ['Hall A', 'Stairs'],  
 'Hall A': ['Lab', 'Armory'],  
 'Stairs': ['Control Room'],  
 'Lab': [],  
 'Armory': ['Exit'],  
 'Control Room': [],  
 'Exit': []  
}

### Sample Input & Expected Output

* **Input:** start='Entry', goal='Exit'
* **Expected Output:** ['Entry', 'Hall A', 'Armory', 'Exit'] (Note: Another valid DFS path could be found depending on neighbor order)

### The Approach (Your Blueprint)

def depth\_first\_search(graph, start, goal):  
 # Implements the iterative version of DFS using a stack.  
 # 1. Initialize a stack and add a tuple: (start\_node, [path\_to\_start\_node]).  
 # 2. Initialize a set to keep track of visited nodes to avoid cycles.  
 # 3. Loop while the stack is not empty.  
 # 4. Pop a node and its path from the stack.  
 # 5. If the node has not been visited:  
 # a. Mark it as visited.  
 # b. If the node is the goal, return its path.  
 # c. For each neighbor of the node, push a new tuple onto the stack: (neighbor, new\_path).  
 # 6. If the loop finishes without finding the goal, return None.  
 pass