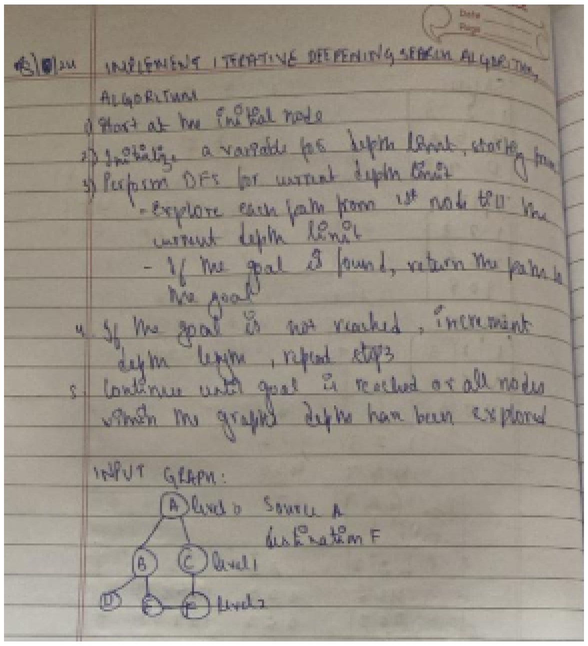
Date:8/10/24

Program Title:Implement Iterative Deepening Search Algorithm

Algorithm - (NEXT PAGE)



8/10/24 IMPLEMENT ITERATIVE DEEPENING SEARCH ALGORITHM

ALGORITHM

1) Start at the initial node
2) Initialize a variable for depth limit, starting from
3) Perform DFS for current depth limit
   - Explore each path from 1st node till the current depth limit
   - If the goal is found, return the path to the goal
4. If the goal is not reached, increment depth length, repeat step 3
5. Continue until goal is reached or all nodes within the graph depth have been explored

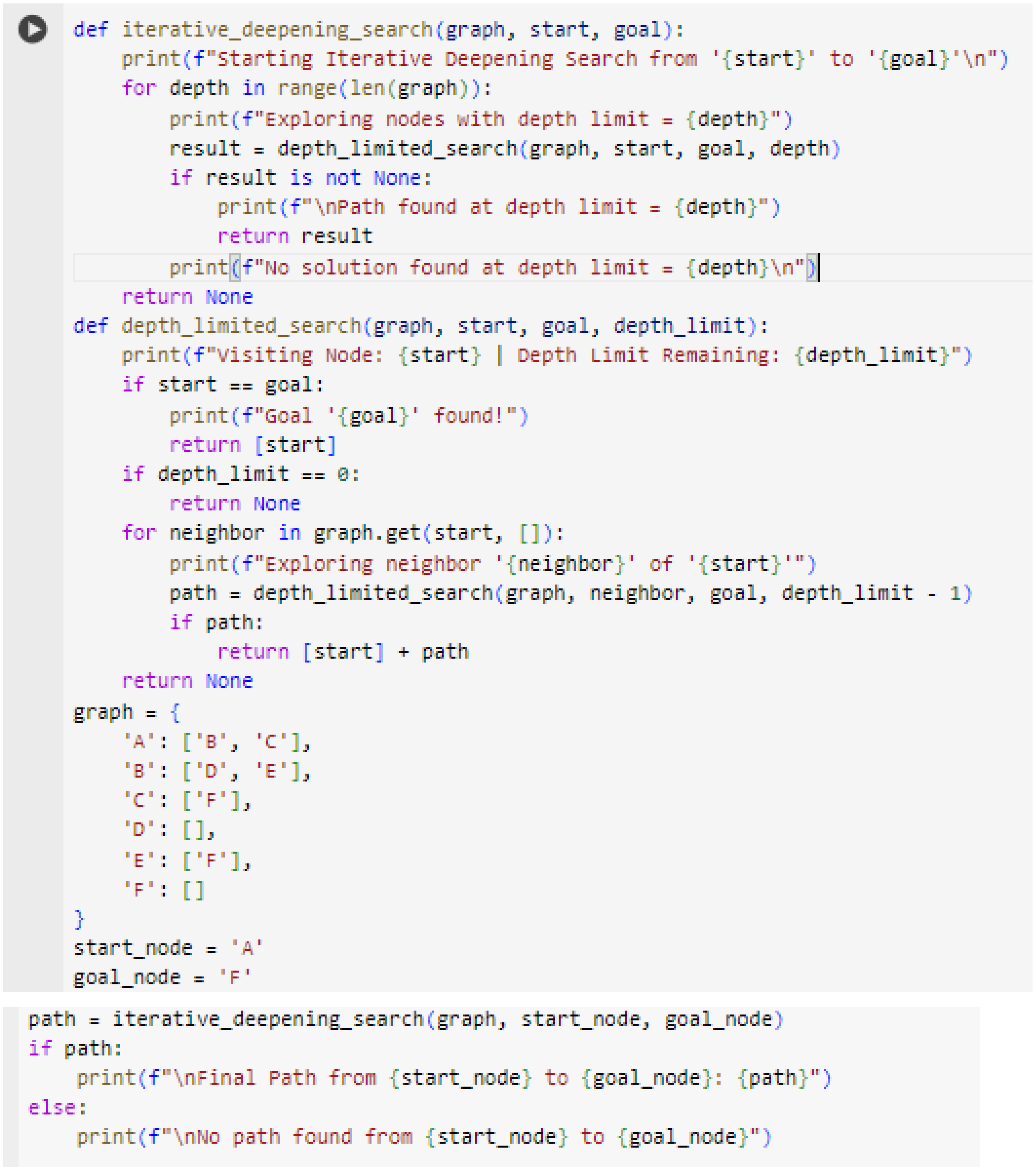
INPUT GRAPH:

$$\begin{tikzcd}
 & & A \arrow[dl, dash] \arrow[d, dash] & \text{level 0} \\
 & B \arrow[dl, dash] \arrow[d, dash] & C \arrow[d, dash] & \text{level 1} \\
D & E \arrow[r, dash] & F & \text{level 2}
\end{tikzcd}$$

Source A

destination F

Code:

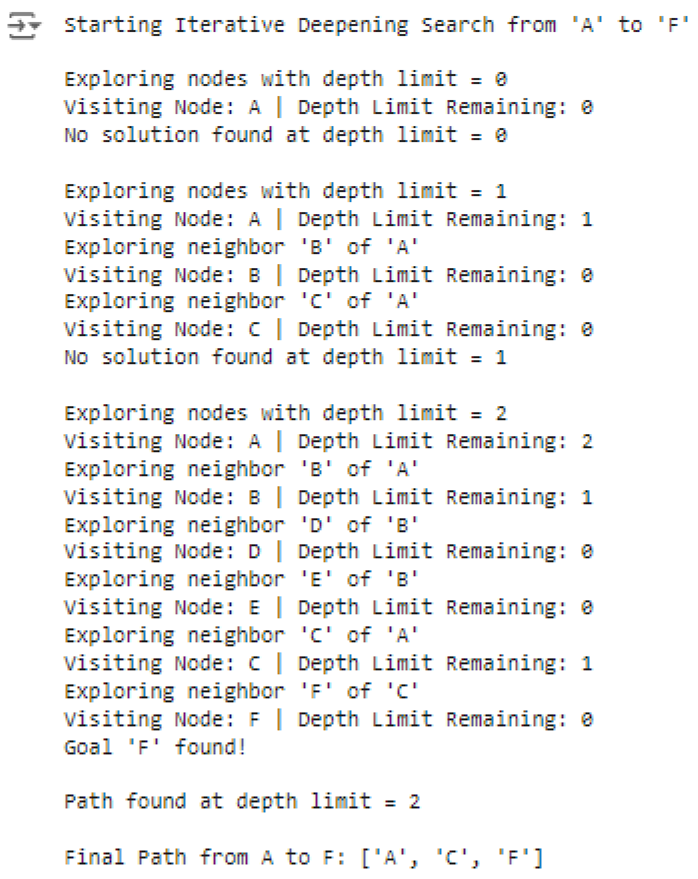


```
def iterative_deepening_search(graph, start, goal):
    print(f"Starting Iterative Deepening Search from '{start}' to '{goal}'\n")
    for depth in range(len(graph)):
        print(f"Exploring nodes with depth limit = {depth}")
        result = depth_limited_search(graph, start, goal, depth)
        if result is not None:
            print(f"\nPath found at depth limit = {depth}")
            return result
        print(f"No solution found at depth limit = {depth}\n")
    return None
def depth_limited_search(graph, start, goal, depth_limit):
    print(f"Visiting Node: {start} | Depth Limit Remaining: {depth_limit}")
    if start == goal:
        print(f"Goal '{goal}' found!")
        return [start]
    if depth_limit == 0:
        return None
    for neighbor in graph.get(start, []):
        print(f"Exploring neighbor '{neighbor}' of '{start}'")
        path = depth_limited_search(graph, neighbor, goal, depth_limit - 1)
        if path:
            return [start] + path
    return None
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
start_node = 'A'
goal_node = 'F'
```

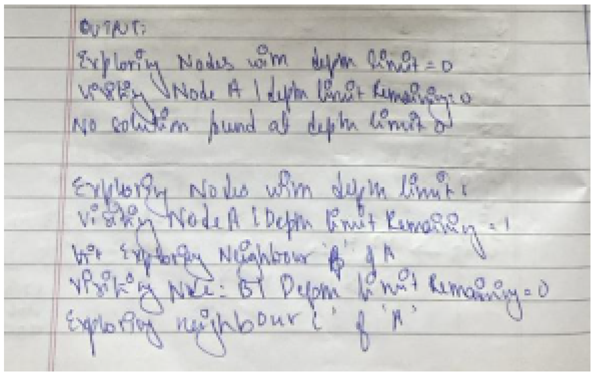
```
path = iterative_deepening_search(graph, start_node, goal_node)
if path:
    print(f"\nFinal Path from {start_node} to {goal_node}: {path}")
else:
    print(f"\nNo path found from {start_node} to {goal_node}")
```

```
def iterative_deepening_search(graph, start, goal):
   print(f"Starting Iterative Deepening Search from '{start}' to '{goal}'\n")
   for depth in range(len(graph)):
      print(f"Exploring nodes with depth limit = {depth}")
      result = depth_limited_search(graph, start, goal, depth)
      if result is not None:
         print(f"\nPath found at depth limit = {depth}")
         return result
      print(f"No solution found at depth limit = {depth}\n")
   return None
def depth_limited_search(graph, start, goal, depth_limit):
   print(f"Visiting Node: {start} | Depth Limit Remaining: {depth_limit}")
   if start == goal:
        print(f"Goal '{goal}' found!")
        return [start]
    if depth_limit == 0:
        return None
    for neighbor in graph.get(start, []):
        print(f"Exploring neighbor '{neighbor}' of '{start}'")
        path = depth_limited_search(graph, neighbor, goal, depth_limit - 1)
        if path:
            return [start] + path
    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"\nFinal Path from {start_node} to {goal_node}: {path}")
else:
    print(f"\nNo path found from {start_node} to {goal_node}")
```

Output(NEXT PAGE)

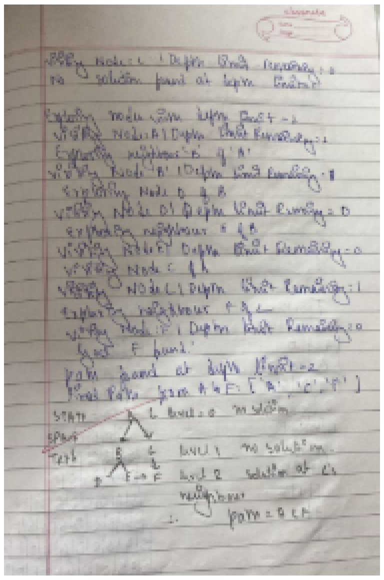


```
Starting Iterative Deepening Search from 'A' to 'F'

Exploring nodes with depth limit = 0
Visiting Node: A | Depth Limit Remaining: 0
No solution found at depth limit = 0

Exploring nodes with depth limit = 1
Visiting Node: A | Depth Limit Remaining: 1
Exploring neighbor 'B' of 'A'
Visiting Node: B | Depth Limit Remaining: 0
Exploring neighbor 'C' of 'A'
Visiting Node: C | Depth Limit Remaining: 0
No solution found at depth limit = 1

Exploring nodes with depth limit = 2
Visiting Node: A | Depth Limit Remaining: 2
Exploring neighbor 'B' of 'A'
Visiting Node: B | Depth Limit Remaining: 1
Exploring neighbor 'D' of 'B'
Visiting Node: D | Depth Limit Remaining: 0
Exploring neighbor 'E' of 'B'
Visiting Node: E | Depth Limit Remaining: 0
Exploring neighbor 'C' of 'A'
Visiting Node: C | Depth Limit Remaining: 1
Exploring neighbor 'F' of 'C'
Visiting Node: F | Depth Limit Remaining: 0
Goal 'F' found!

Path found at depth limit = 2

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



OUTPUT:
Exploring Nodes with depth limit = 0
visiting Node A | depth limit Remaining = 0
No solution found at depth limit 0

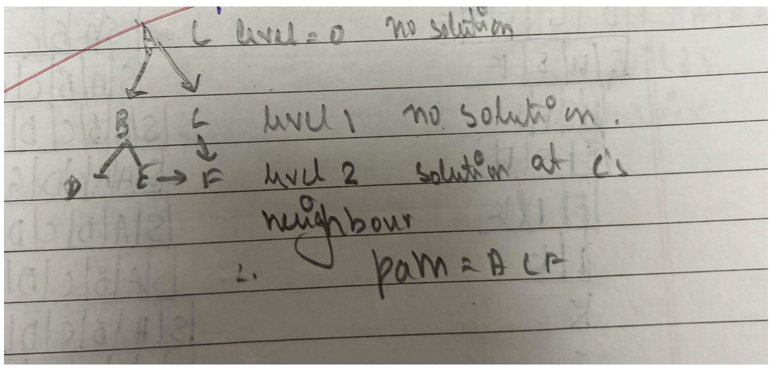
Exploring Nodes with depth limit 1
visiting Node A | Depth limit Remaining = 1
[illegible] Exploring Neighbour 'B' of A
visiting Node: B | Depth limit Remaining = 0
Exploring neighbour 'C' of 'A'



visiting Node: C | Depth limit Remaining: 0
No solution found at depth limit 1

Exploring nodes with depth limit = 2
visiting Node: A | Depth limit Remaining: 2
Exploring neighbour 'B' of 'A'
visiting Node 'B' | Depth limit Remaining: 1
Exploring Node D of B
visiting Node D | Depth limit Remaining = 0
Exploring neighbour E of B
visiting Node E | Depth limit Remaining = 0
visiting Node C of A
visiting Node C | Depth limit Remaining: 1
Exploring neighbour F of C
visiting Node F | Depth limit Remaining: 0
Goal F found
Path found at depth limit = 2
Final Path from A to F: ['A', 'C', 'F']

STATE SPACE TREE:



$$\begin{tikzcd}
 & A \arrow[dl] \arrow[dr] & \\
B \arrow[d] \arrow[dr] & & C \arrow[d] \\
D & E \arrow[r] & F
\end{tikzcd}$$

level = 0 no solution

level 1 no solution.

level 2 solution at C's neighbour

∴ path = A C F