**NAME OF THE EXPERIMENT**

Implementation of IDS (Iterative Deepening Search).

**THEROY**

Iterative Deepening Search: In computer science, iterative deepening search or more specifically iterative deepening depth-first search is a state space/graph search strategy in which a depth-limited version of depth-first search is run repeatedly with increasing depth limits until the goal is found. The algorithm of IDS is given below:

bool IDS (src, target, max\_depth)

for limit from 0 to max\_depth

if DLS(src, target, limit) == true

return true

return false

bool DLS (src, target, limit)

if (src == target)

return true;

if (limit <= 0)

return false;

for each adjacent i of src

if DLS (i, target, limit?1)

return true

return false

**SOURCE CODE:**

class Graph:

    def \_\_init\_\_(self):

        self.graph = {}

        for i in range(20):

            self.graph[i] = []

    def Print(self):

        print(self.graph)

    def add\_edge(self, node, neighbour):

        if node not in self.graph:

            self.graph[node] = [neighbour]

        else:

            self.graph[node].append(neighbour)

    def dls\_util(self, node, goal, limit, visited):

        if node == goal:

            return True

        if limit <= 0:

            return False

        visited.add(node)

        for child in self.graph[node]:

            if child not in visited and self.dls\_util(child, goal, limit - 1, visited):

                return True

        visited.remove(node)

        return False

    def dls(self, start, goal, limit):

        return self.dls\_util(start, goal, limit, set())

    def IDS(self, start, goal):

        depth = 0

        while True:

            result = self.dls(start, goal, depth)

            if result == True:

                return depth

            depth += 1

g = Graph()

g.add\_edge(1, 2)

g.add\_edge(1, 3)

g.add\_edge(2, 4)

g.add\_edge(2, 5)

g.add\_edge(3, 6)

g.add\_edge(3, 7)

g.add\_edge(4, 8)

g.add\_edge(4, 9)

g.add\_edge(5, 10)

g.add\_edge(5, 11)

g.add\_edge(6, 12)

g.add\_edge(6, 13)

g.add\_edge(7, 14)

g.add\_edge(7, 15)

g.add\_edge(8, 16)

g.add\_edge(9, 18)

g.add\_edge(9, 19)

g.add\_edge(16, 17)

result = g.IDS(1, 17)

print(result)

**SOURCE CODE EXPLANATION**

This code defines a class called Graph that represents a graph. The graph is represented as a dictionary, where the keys are the nodes in the graph and the values are the neighbors of each node.

The Graph class has the following methods:

• \_\_init\_\_(): Initializes the graph.

• Print(): Prints the graph.

• add\_edge(node, neighbour): Adds an edge between two nodes.

• dls\_util(node, goal, limit, visited): Performs a depth-limited search with a specified limit.

• dls(start, goal, limit): Calls the dls\_util function to perform a depth-limited search.

• IDS(start, goal): Performs an iterative deepening search.

The dls function takes a start node, a goal node, and a limit as input. The function performs a depth-limited search starting at the start node and trying to find the goal node. The limit parameter specifies the maximum depth of the search. If the goal node is not found within the limit, the function returns False. Otherwise, the function returns True.

The IDS function takes a start node and a goal node as input. The function performs an iterative deepening search, which is a type of search that combines the depth-first search and breadthfirst search algorithms. The function starts with a limit of 0 and then increases the limit by 1 on each iteration. The function stops when the goal node is found or when the limit reaches the number of nodes in the graph.

The code first creates a graph object and then adds edges to the graph. The code then calls the dls function to perform a depth-limited search with a limit of 2. The code then calls the IDS function to perform an iterative deepening search.

The output of the code is:

Iterative Deepening Search:

3

**CONCLUSION**

In conclusion, the presented lab report outlines the implementation of the IDS salgorithm using Python, encapsulated within a graph class. The algorithm efficiently navigates through graph structures. The provided code not only illustrates the fundamental components of the IDS algorithm but also demonstrates its practical application through an example graph. Overall, this lab report serves as a valuable exploration of non-heuristic guided IDS search, shedding light on its significance in artificial intelligence and problem-solving domains optimization, thorough testing, and user-friendliness should be prioritized to enhance the overall effectiveness of the proposed solution.