Table of Contents

Objectives

Graph traversing is a fundamental concept in computer science and plays a crucial role in various applications. It provides a foundation for solving complex problems, analyzing data, and navigating interconnected systems, making it a fundamental skill in computer science and related disciplines. And, IDS is one of the optimized graph traversing techniques. Therefore, the main objectives of this report are as follows-

- To understand the basic concept of graph traversal algorithms.
- To understand the basic mechanism behind the IDS algorithm.
- To implement the IDS search algorithm by using python and understand the concept practically.

Description

The Iterative Deepening Search algorithm is a combination of depth-first search and breadth-first search strategies. It repeatedly applies a depth-limited search, gradually increasing the depth limit with each iteration until a solution is found. The depth limit is the key parameter that balances memory usage and search space coverage. IDS explores the state space systematically, avoiding the pitfalls of excessive memory consumption associated with pure breadth-first searches. In implementation of IDS, there are two cases as follows-

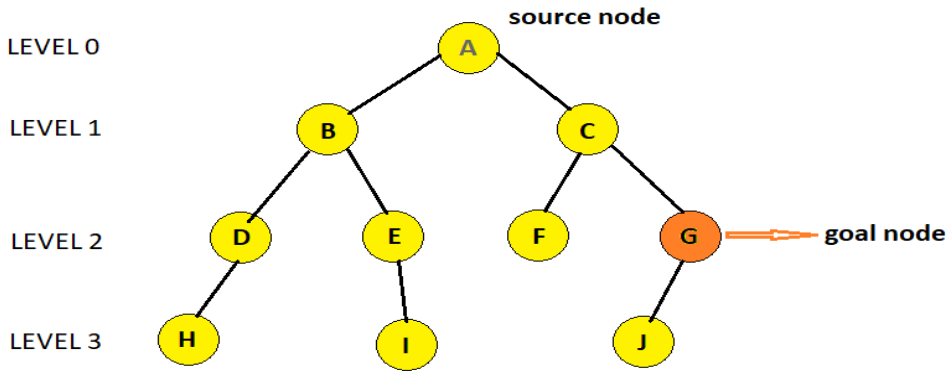
1. **Graph without cycle:** This case is simple. We can DFS multiple times with different height limits.
2. **Graph with cycle:** This is interesting as there is no visited flag in IDDFS.

However, The Iterative Deepening Depth-First Search (or Iterative Deepening search) algorithm, repeatedly applies depth-limited search with increasing limits.

It gradually increases limits from 0,1, ..., d, until the goal node is found. It terminates in following two cases-

- When the goal node is found
- The goal node does not exist in the graph/tree.

The following figure shows the overall explanation of the IDS or IDDFS algorithm where the max-depth-limit is 3 and the start node is A and the goal node is the G.



IDDFS with max depth-limit = 3
Note that iteration terminates at depth-limit=2
Iteration 0: A
Iteration 1: A->B->C
Iteration 2: A->B->D->E->C->F->G

Figure 01: Sample explanation on IDS algorithm

Source Code

```
# IDS Algorithm

# Class creation
class Node:
    def __init__(self, state, parent=None):
        self.state = state
        self.parent = parent

    def __eq__(self, other):
        return self.state == other.state

    def __hash__(self):
        return hash(self.state)

# DLS method
def depth_limited_search(node, goal_state, actions, depth_limit):
    if node.state == goal_state:
        return node
    elif depth_limit == 0:
        return None
    else:
        connect = False
        for action in actions(node.state):
            child = Node(action, parent=node)
            result = depth_limited_search(child, goal_state, actions, depth_limit
- 1)
            if result is not None:
```

```
                return result
            connect = True
        return None if connect else "Connectionless"

# IDS method
def iterative_deepening_search(root, target_state, actions, max_depth):
    for depth in range(max_depth + 1):
        result = depth_limited_search(root, target_state, actions, depth)
        if result is not None:
            return result
    return None
initial_state = "A"
target_state = "D"

def actions(state):
    if state == "A":
        return ["B", "C"]
    elif state == "B":
        return ["A", "D"]
    elif state == "C":
        return ["A", "E"]
    elif state == "D":
        return ["B"]
    elif state == "E":
        return ["C"]
root_node = Node(initial_state)
result_node = iterative_deepening_search(root_node, target_state, actions,
max_depth=1)

if result_node is not None:
    path = []
    while result_node is not None:
        path.insert(0, result_node.state)
        result_node = result_node.parent
    print("Solution Path: ", " => ".join(path))
else:
    print("No solution found.")
```

Sample Input & Output

runcell(0, 'D:/Documents_2/Level_3_Term_2/Sessional/Artificial Intelligence/Program_Folder/IDS.py')
Solution Path: A => B => D

Code Analysis

The pseudocode for the IDS algorithm is as follows-

function depth_limited_search(node, goal_state, actions, depth_limit):
if node.state == goal_state:
return node

```
    elif depth_limit == 0:
        return None
    else:
        connect = False
        for action in actions(node.state):
            child = Node(action, parent=node)
            result = depth_limited_search(child, goal_state, actions, depth_limit - 1)
            if result is not None:
                return result
            connect = True
        return None if connect else "Connectionless"

function iterative_deepening_search(root, target_state, actions, max_depth):
    for depth in range(max_depth + 1):
        result = depth_limited_search(root, target_state, actions, depth)
        if result is not None:
            return result
    return None
```

This code implements the Iterative Deepening Search (IDS) algorithm, a search strategy that systematically explores the state space by incrementally increasing the depth limit of a depth-limited search (DLS).

- **Class Creation:** A `Node` class is defined to represent a state in the search space. Each node contains the state, a reference to its parent node, and methods to check for equality and calculate a hash.
- **DLS Method:** `depth_limited_search` function performs a depth-limited search from a given node up to a specified depth limit.
    1. If the goal state is found, the function returns the node with the goal state.
    2. If the depth limit is reached, it returns `None`.
    3. Otherwise, it recursively explores the successors of the current node until the depth limit is reached.
- **IDS Method:** - `iterative_deepening_search` function iteratively applies depth-limited search with increasing depth limits until the goal state is found or the maximum depth is reached.It returns the node with the goal state if found; otherwise, it returns `None`.
- **Sample Traversal:** An example problem is defined where the initial state is "A" and the target state is "D". The `actions` function defines the possible actions from each state. A

root node is created with the initial state, and the IDS algorithm is applied with a maximum depth of 2. If a solution is found, the path from the initial state to the goal state is printed.

Discussion

The Iterative Deepening Search algorithm is a valuable technique for solving problems where the depth of the solution is uncertain. Its iterative nature ensures that it explores the state space systematically while keeping memory usage in check. By understanding the trade-offs between search strategies, one can choose the most appropriate algorithm based on the problem requirements. The major advantages of the algorithm are as follows-

- IDS is superior to other search algorithms in a number of ways. The first benefit is that it is comprehensive, which ensures that a solution will be found if one is there in the search space. This is so that all nodes under a specific depth limit are investigated before the depth limit is raised by IDS, which does a depth-limited DFS.
- IDS is memory-efficient, which is its second benefit. This is because IDS decreases the algorithm's memory needs by not storing every node in the search area in memory.

Though there are several advantages of this algorithm, there are some disadvantages also as following-

- IDS has the disadvantage of potentially visiting certain nodes more than once or the goal node can’t be searched if the max depth limit is not so enough to reach the target node, which might slow down the search.

Therefore, this report provides a comprehensive overview of the IDS algorithm, its objectives, description, pseudocode, and a discussion of its advantages and limitations.

References

1. Iterative Deepening Search or IDDFS [https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/]

2. Iterative Deepening Search (IDS) or Iterative Deepening Depth First Search (IDDFS) [https://www.javatpoint.com/iterative-deepening-search-or-iterative-deepening-depth-firstsearch]