Green University of Bangladesh

Department of Computer Science and Engineering(CSE)

Faculty of Science and Engineering

Semester: (Spring, Year:2025), B.Sc. in CSE (Day)

Lab Report 02

Course Title: Artificial Intelligence Lab

Course Code: CSE 316 **Section: 221 D9**

Lab Experiment Name: Topological search using IDDFS.

Student Details

Name	ID
Shahadat Hosen Nishan	221002099

Lab Date	:	23/02/2025
Submission Date	:	26/04/2025
Course Teacher's Name	:	Md. Riad Hassan

[For Teachers use only: Don't Write Anything inside this box]

Lab Report

Status Marks: …………………………………… **Signature:…………………**

Comments:………………………………………… **Date:…………………………**

Title:

Topological search using IDDFS.

Objective:

To implement and understand Topological Search using Iterative Deepening Depth-First Search (IDDFS) in a Directed Acyclic Graph (DAG).

Problem Statement:

Given a Directed Acyclic Graph (DAG), perform a topological sort of its vertices using Iterative Deepening Depth-First Search (IDDFS). The goal is to determine a linear ordering of vertices such that for every directed edge u → v, vertex u appears before v in the ordering. This approach aims to demonstrate the feasibility of using IDDFS, typically used for search, for topological sorting.

Introduction:

Topological sorting is a linear ordering of vertices in a DAG such that for every directed edge u → v, vertex u comes before v. While traditional approaches use DFS or BFS, this lab explores the use of IDDFS, a hybrid of DFS and BFS that uses DFS repeatedly with increasing depth limits.

Algorithm:

IDDFS-based Topological Sort:

Steps:

1. For depth = 0 to maximum depth:
2. For each unvisited node:
    a. Perform Depth-Limited DFS (DLDFS) up to current depth.
    b. Push nodes to a stack when fully explored.
3. Reverse the stack to obtain topological order.

Complexity:

Time Complexity: O(D * (V + E)), where D is the maximum depth.
Space Complexity: O(V)

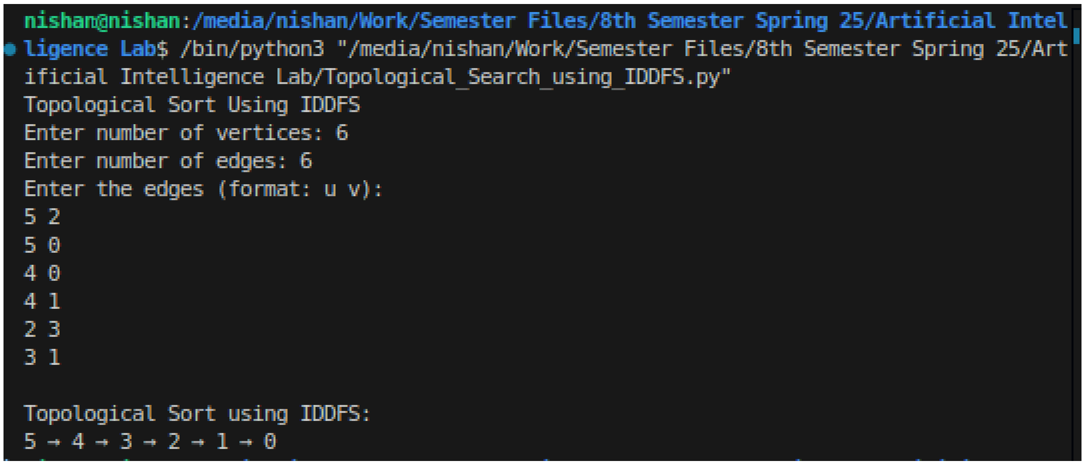
Implementation:

Code:

```
from collections import defaultdict

class TopologicalSortIDDFS:
    def __init__(self, total_vertices):
        self.graph = defaultdict(list)
        self.V = total_vertices
        self.topo_stack = []

```

```python
    def add_edge(self, start, end):
        self.graph[start].append(end)

    def depth_limited_dfs(self, node, visited, current_depth, depth_limit):
        if current_depth > depth_limit:
            return
        visited[node] = True
        for neighbor in self.graph[node]:
            if not visited[neighbor]:
                self.depth_limited_dfs(neighbor, visited, current_depth + 1, depth_limit)
        if node not in self.topo_stack:
            self.topo_stack.append(node)

    def perform_topological_sort(self):
        for depth_limit in range(self.V):
            visited = [False] * self.V
            for vertex in range(self.V):
                if not visited[vertex]:
                    self.depth_limited_dfs(vertex, visited, 0, depth_limit)


        print("\nTopological Sort using IDDFS:")
        print(" → ".join(map(str, reversed(self.topo_stack))))

if __name__ == "__main__":
    print("Topological Sort Using IDDFS")

    V = int(input("Enter number of vertices: "))
    E = int(input("Enter number of edges: "))

    ts = TopologicalSortIDDFS(V)

    print("Enter the edges (format: u v):")
    for _ in range(E):
        u, v = map(int, input().split())
        ts.add_edge(u, v)

    ts.perform_topological_sort()
```

Output:



```
nishan@nishan:/media/nishan/Work/Semester Files/8th Semester Spring 25/Artificial Intel
ligence Lab$ /bin/python3 "/media/nishan/Work/Semester Files/8th Semester Spring 25/Art
ificial Intelligence Lab/Topological_Search_using_IDDFS.py"
Topological Sort Using IDDFS
Enter number of vertices: 6
Enter number of edges: 6
Enter the edges (format: u v):
5 2
5 0
4 0
4 1
2 3
3 1

Topological Sort using IDDFS:
5 → 4 → 3 → 2 → 1 → 0
```

fig: Topological search using IDDFS

Conclusion:

This lab explored an alternate way of implementing topological sort using IDDFS. While less efficient than standard approaches, it provides educational insight into how DFS depth can influence traversal and order discovery.