

**Green University of Bangladesh**

**Department of Computer Science and Engineering (CSE) Faculty of Sciences and Engineering**

**Semester: (Spring, Year: 2024), B.Sc. in CSE (Day)**

**Lab Report NO- 04**

Course Title: Artificial Intelligence Lab

**Course Code: CSE 316 Section: 213-D1**

**Lab Experiment Name:** Write a program to perform topological search using IDDFS.

**Student Details**

| **Name** | | **ID** |
| --- | --- | --- |
| **1.** | Mostak Ahmmed | **213902126** |

**Lab Date : 02-04-2024**

**Submission Date : 14-05-2024**

**Course Teacher’s Name :** Fairuz Shaiara

| **Lab Report Status**  **Marks: ………………………………… Signature:..................... Comments:.............................................. Date:..............................** |
| --- |

**1. TITLE OF THE LAB REPORT EXPERIMENT**

Write a program to perform topological search using IDDFS.

**2. OBJECTIVES/AIM**

➢ To understand how to represent a graph using an adjacency list. ➢ To understand how Iterative Deepening Depth-First Search (DIDFS) works. **3.1 PROCEDURE / ANALYSIS / DESIGN**

➢ The IterativeDeepening class contains methods for performing iterative deepening and depth-limited search.

➢ iterativeDeepening: This method repeatedly performs depth-limited search (DLS) with increasing depth limits until the goal node is found or until the entire graph is explored.

➢ depthLimitedSearch: This method implements the DLS using a stack to keep track of nodes to visit within the current depth limit.

**4.1 IMPLEMENTATION**

class IterativeDeepening:

def \_\_init\_\_(self):

self.stack = []

self.numberOfNodes = 0

self.depth = 0

self.maxDepth = 0

self.goalFound = False

def iterativeDeepening(self, adjacencyMatrix, destination):

self.numberOfNodes = len(adjacencyMatrix)

while not self.goalFound:

self.depthLimitedSearch(adjacencyMatrix, 0, destination) # Start from node 0

self.maxDepth += 1

if self.goalFound:

print(f"\nGoal {destination} found at depth {self.depth}")

return

self.depth = 0

self.stack = []

def depthLimitedSearch(self, adjacencyMatrix, source, goal):

visited = [False] \* self.numberOfNodes

self.stack.append(source)

while self.stack:

element = self.stack.pop()

if not visited[element]:

visited[element] = True

print(element + 1, end=' ') # Print node index (1-based) for output

if element == goal:

self.goalFound = True

self.depth += 1

return

# Explore neigbor] == 1 and not

visited[neighbor]:

self.stack.append(neighbor)

if \_\_name\_\_ == "\_\_main\_\_":

try:

print("Enter the number of nodes in the graph\n")

number\_of\_nodes = int(input().strip())

adjacency\_matrix = []

print("Enter the adjacency matrix\n")

for i in range(number\_of\_nodes):

row = list(map(int, input().strip().split()))

adjacency\_matrix.append(row)

print("Enter the destination for the graph\n")

destination = int(input().strip())

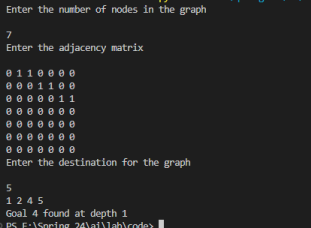
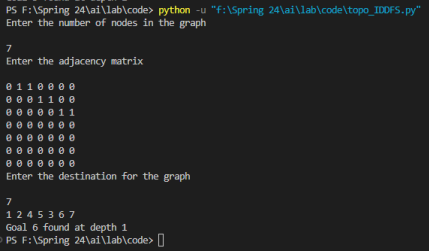
iterativeDeepening = IterativeDeepening()

iterativeDeepening.iterativeDeepening(adjacency\_matrix, destination - 1) # Adjust for 0-based indexing

except ValueError:

print("Wrong Input format")

**5.1 TEST RESULT / OUTPUT**

****

**6. ANALYSIS AND DISCUSSION**

➢ the code assumes that the graph is represented using an adjacency matrix, which is suitable for dense graphs but may not be space-efficient for sparse graphs.

➢ The graph traversal uses a depth-first approach combined with iterative deepening to ensure completeness and optimality in finding the shortest path to the destination node.

➢ Error handling is implemented to manage input-related exceptions (ValueError) to ensure the code handles unexpected input formats gracefully.

➢ Output formatting is adjusted to display node indices in a 1-based format for readability.

Overall, the code provides a structured approach to performing topological search using IDDFS on a directed graph, allowing for efficient exploration of graph nodes while respecting the defined depth limits.

**7. SUMMARY**

In this lab report, we implemented a topological search algorithm using Iterative Deepening Depth-First Search (IDDFS) on a directed graph represented by an adjacency matrix. The objective was to explore the graph in a depth-first manner while respecting a specified depth limit, aiming to find a specific destination node and determine the depth at which it was located.