Pusat Sains Matematik	**SUBJECT: BSD2513** **ARTIFICIAL INTELLIGENCE**	**MARKS**: 25(5%)
	TOPIC: Chapter 1: Introduction to Artificial Intelligence Chapter 2: Search Algorithms	
	LAB REPORT 1	

CLO	Description	PLO Mapping	Percentage	Marks
CLO1	Acquire the artificial intelligence concepts and methodologies in data science.	PLO1: Knowledge and Understanding C3: Application	1%	5
CLO2	Demonstrate critical thinking ideas of artificial intelligence knowledge in problem-solving situation.	PLO2: Cognitive Skills and Functional work skills with focus on Numeracy skills CLO3: Application	1%	5
CLO3	Develop an artificial intelligence system prototype using appropriate software.	PLO3: Functional work skills with focus on Practical, and Digital skills P4: Mechanism	3%	15

Laboratory Report Objectives

By the end of this lab, students should be able to:

1. articulate AI capability types and their relevance to real deployments;
2. reason about and trace Breadth-First Search (BFS) and Depth-First Search (DFS) on given graphs; and
3. implement a minimal, reliable web app (Streamlit or similar) that runs these traversals, explains their complexity, and is reproducible from a GitHub repository (with a live deployment link).

CASE STUDY:

Modern organizations (e.g., logistics, telecom, and public services) routinely face search and routing problems under time and resource constraints. Artificial Intelligence (AI) methods, particularly search algorithms where are used to explore large state spaces to find feasible or optimal solutions. Typical use cases include:

- Network traversal & connectivity: discovering reachable components or shortest hops across campus or city networks.
- Work allocation & incident response: exploring task graphs to sequence actions under precedence constraints.
- Navigation and routing: traversing road graphs to plan visits while honouring constraints (e.g., fuel, time windows).

Question 1
General Knowledge

Explain types of AI by capability and functionality. For each capability and functionality, give one realistic application example in data-driven operations. Conclude with a short note on ethical and governance considerations relevant to deployment (e.g., transparency, safety, bias).

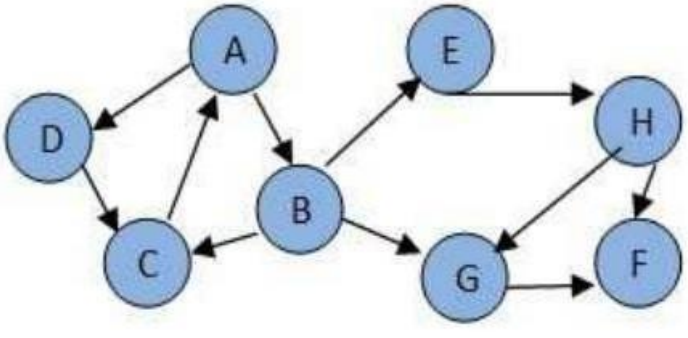
(5 Marks)
CO1 PO1)

Question 2
Python: Search Algorithms

Consider the directed graphs provided below. When expanding nodes, assume alphabetical tie-breaking, i.e., when multiple adjacent nodes are available, always visit the node with the name closest to the beginning of the alphabet first using:

a. breadth-first search (BFS) and,
b. depth-first search (DFS).

Discuss the level and the process path clearly.



(5 Marks)
(CO2PO2)
(15 Marks)
(CO3PO3)

Save your work in both .py and PDF formats. Name your files using the following format: StudentID_LabX. Submit both files through the Kalam platform by 10th October 2025, 11:59 PM. In addition, deploy your Streamlit application and include the public URL to your deployed app and GitHub repository link inside your report. Late submissions will only be considered with prior approval.

Pusat Sains Matematik	**SUBJECT: BSD3513** **INTRODUCTOION TO ARTIFICIAL INTELLIGENCE**	**MARKS**: 25(5%)
	TOPIC: Chapter 1: Foundations of Artificial Intelligence Chapter 2: Search Algorithms	
	LAB REPORT 1	

NAME: ZAHIN ZIKRI BIN ZAWAWI **STUDENT ID: SD23019** **SECTION: 02G**

Mark for CO1: /5

Rubric for CO2.

Instruction: For CO2, assess each item using the given scales.

Demonstrate critical thinking ideas of artificial intelligence knowledge in problem-solving situation.								
Item Assessed (Cognitive)	**Very Poor 0**	**Poor 1**	**Fair 2**	**Good 3**	**Very Good 4**	**Excellent 5**	**Weightage**	**Score**
Apply and analyse relevant artificial intelligence knowledge.	The work has not done.	Poorly applied and analysed relevant artificial intelligence knowledge and results.	Applied and analysed relevant artificial intelligence knowledge but failed to achieve successful results.	Applied and analysed relevant artificial intelligence knowledge but arrive at satisfactory results.	Applied and analysed relevant artificial intelligence knowledge to arrive at successful results.	Applied and analysed relevant artificial intelligence knowledge to arrive at excellent results.	0.5	
Using logical, rational or problem-solving appropriate to the artificial intelligence problems.	The work has not done.	The work needs to demonstrate logical, rational or problem-solving understanding appropriate to the artificial intelligence problems.	The work has demonstrated some logical, rational or problem-solving understanding appropriate to the artificial intelligence problems.	The work has demonstrated logical, rational or problem-solving understanding appropriate to artificial intelligence problems.	The work has demonstrated a thorough logical, rational or problem-solving understanding appropriate to artificial intelligence problems.	The work has demonstrated a thorough and classy logical, rational or problem-solving understanding appropriate to artificial intelligence problems.	0.5	
						Total Score	**1**	**/5**

Rubric for CO3.

Instruction: For CO3, assess each item using the given scales.

CO3: Develop an artificial intelligence system prototype using appropriate softwar					
Item Assessed (Cognitive)	**Very Poor 0**	**Poor 1**	**Fair 2**	**Good 3**	**Very Good 4**
Utilizing the appropriate tools / software effectively	No relevant tool used.	Tools used but did not enhance solution or information clarity.	Tools used but with limited enhancement; minimal functionality demonstrated.	Tools used appropriately to produce a functional solution with clear output.	Tools used effectively to enhance clarity, performance, and solution quality.
Code functionality, clarity & structure	No code constructed.	Code incomplete or mostly non-functional; unclear and poorly structured.	Partially functional code; errors present; structure somewhat difficult to follow.	Mostly functional code with minor errors; clear structure and readable.	Fully functional and well-structured code; clearly commented and readable.
Deployment & Version Control (GitHub + Streamlit or etc)	No deployment and no GitHub repository.	GitHub repo exists but incomplete OR app deploy attempt failed.	GitHub repo available with basic files; deployment page exists but app not functioning correctly.	Working deployment provided; GitHub repo contains main code files.	Working deployment with complete repository (README, code, requirements); clearly accessible.

BSD3513 – ARTIFICIAL INTELLIGENCE LAB REPORT 1

Question 1

Explain types of AI by capability and functionality. For each capability and functionality, give one realistic application example in data-driven operations. Conclude with a short note on ethical and governance considerations relevant to deployment (e.g., transparency, safety, bias).

Answer :

Types of AI by Capability and Functionality

Artificial Intelligence (AI) can be categorised based on **capability** and **functionality**, each describing how intelligent and how autonomous a system is. Understanding these categories helps organisations choose suitable AI solutions for real-world, data-driven operations.

1. AI by Capability

a) Artificial Narrow Intelligence (ANI)

ANI refers to AI systems designed to perform a **single, specialised task**.
They cannot generalise their intelligence beyond their specific domain.
Example Application:
Logistics forecasting systems that use historical delivery data to predict demand and schedule trucks efficiently.

b) Artificial General Intelligence (AGI)

AGI represents AI that can understand, learn, and apply knowledge across **any domain** similar to a human being. AGI is still theoretical and not yet achieved.
Example Application (future concept):
A fully autonomous operations manager that can plan routes, optimise schedules, analyse risks, and take decisions across all departments without human intervention.

c) Artificial Superintelligence (ASI)

ASI refers to AI that surpasses human intelligence in **all aspects**, including reasoning, planning, creativity, and decision-making. This remains hypothetical and is associated with philosophical and safety discussions.
Example Application (theoretical):
A system that autonomously optimises entire national transport networks, weather prediction, economic planning, and emergency responses simultaneously.

2. AI by Functionality

a) Reactive Machines

These systems can only react to current inputs; they **do not store memory**.

Example Application:

Real-time traffic light controllers that change signals based solely on current vehicle density.

b) Limited Memory

These systems use **past data** to make better decisions. Most modern AI systems fall into this category.

Example Application:

Ride-sharing apps (Grab/Uber) that store short-term data (location, traffic, fares) to optimise driver–rider matching.

c) Theory of Mind (Future AI)

This category describes machines that can understand **human emotions, beliefs, and intentions**.

Currently research-stage.

Example Application (future concept):

Customer service robots that emotionally adapt their tone during conversations with frustrated customers.

d) Self-Aware AI

Represents AI that has **consciousness and self-awareness**. This stage does not exist yet.

Example Application (hypothetical):

Fully self-governing AI systems that make independent decisions about mission goals and self-correction.

3. Ethical and Governance Considerations

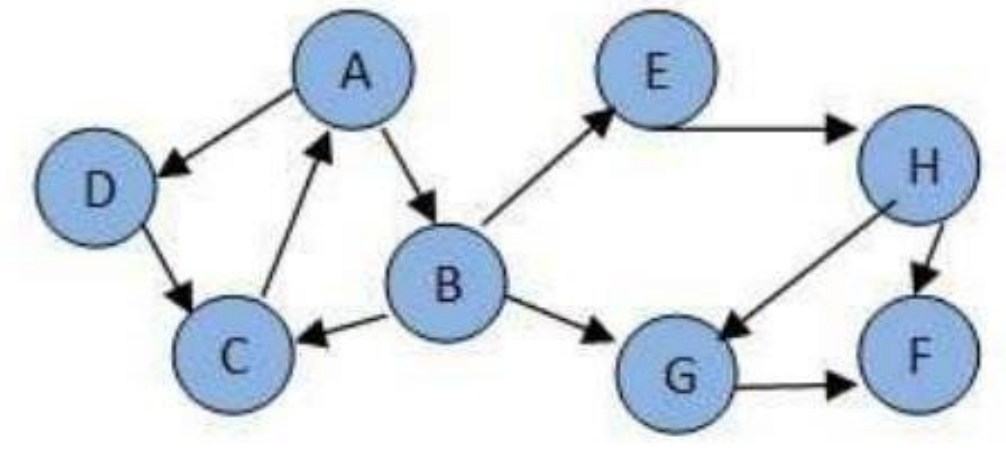
When deploying AI in real-world environments, especially data-driven operations, several governance factors must be considered:

Transparency: Organisations must ensure AI decisions are explainable and auditable.
Bias & Fairness: AI should not discriminate; training data must be free from harmful bias.
Safety: AI must be tested to avoid unsafe actions that may harm users or infrastructure.
Privacy: Only necessary data should be collected; sensitive data must be protected.
Accountability: Organisations deploying AI must have clear responsibility structures for failures or misuse.

Question 2

Consider the directed graphs provided below. When expanding nodes, assume alphabetical tie-breaking, i.e., when multiple adjacent nodes are available, always visit the node with the name closest to the beginning of the alphabet first using:

a. breadth-first search (BFS) and,
b. depth-first



Discuss the level and the process path clearly.

```
graph = {

    "A": ["B", "D"],
    "B": ["C", "E", "G"],
    "C": ["A"],
    "D": ["C"],
    "E": ["H"],
    "F": [],
    "G": ["F"],
    "H": ["G", "F"]
}
```

Python code :

```
import streamlit as st
from collections import deque
from PIL import Image, ImageDraw, ImageFont
from pathlib import Path

# Streamlit App Header
st.set_page_config(page_title="BFS & DFS Visualizer", layout="centered")
st.title(" BFS and DFS Graph Traversal Visualizer")
st.markdown("### BSD3513 – Lab Report 1")
st.markdown("*Name:* ZAHIN ZIKRI BIN ZAWAWI | *Student ID:* SD23019 | *Section:*
02G")
```

```
# Display Graph Image
st.subheader("Graph Reference Image")
st.info("Below is the sample graph used for BFS and DFS traversal
demonstrations.")

script_dir = Path(".")      # Safer for Streamlit Cloud
candidates = ["LabReport_BSD2513_#1.jpg"]



img_path = None
for name in candidates:
    p = script_dir / name
    if p.exists():
        img_path = p
        break

if img_path:
    try:
        image = Image.open(img_path)
        st.image(image, caption=f"Graph used ({img_path.name})",
use_column_width=True)
    except Exception:
        st.warning(f"⚠ Found '{img_path.name}' but failed to open it.")
else:
    # Placeholder image when file is missing
    W, H = 800, 400
    placeholder = Image.new("RGB", (W, H), color="white")
    draw = ImageDraw.Draw(placeholder)
    font = ImageFont.load_default()
    text = "Image not found\nPlace graph image in this folder"
    lines = text.split("\n")
    y_offset = 150

    for line in lines:
        bbox = draw.textbbox((0, 0), line, font=font)
        w = bbox[2] - bbox[0]
        h = bbox[3] - bbox[1]
        draw.text(((W - w) / 2, y_offset), line, fill="black", font=font)
        y_offset += h + 5

    st.image(placeholder, caption="(Placeholder) Graph not found",
use_column_width=True)

# Graph Definition
graph = {
```

```
    "A": ["B", "D"],
    "B": ["C", "E", "G"],
    "C": ["A"],
    "D": ["C"],
    "E": ["H"],
    "F": [],
    "G": ["F"],
    "H": ["G", "F"]
}

st.subheader("1. Graph Structure")
st.json(graph)

# Ensure deterministic traversal (alphabetical order)
for node in graph:
    graph[node] = sorted(graph[node])

# BFS WITH LEVEL TRACKING
def bfs_with_levels(graph, start):
    visited = []
    queue = deque([(start, 0)])  # store (node, level)
    levels = {}

    while queue:
        node, level = queue.popleft()

        if node not in visited:
            visited.append(node)

            # save node by level
            if level not in levels:
                levels[level] = []
            levels[level].append(node)

            for neighbour in graph[node]:
                if neighbour not in visited:
                    queue.append((neighbour, level + 1))

    return visited, levels
# DFS WITH STEP TRACKING
def dfs_with_steps(graph, start, visited=None, order=None, steps=None, depth=0):
    if visited is None:
        visited = set()
        order = []
        steps = []
```

```
    steps.append(f"{' ' * depth * 2}➡ Enter {start} (depth {depth})")

    if start not in visited:
        visited.add(start)
        order.append(start)

        for neighbour in graph[start]:
            dfs_with_steps(graph, neighbour, visited, order, steps, depth + 1)

    steps.append(f"{' ' * depth * 2}⬅ Backtrack from {start} (depth {depth})")
    return order, steps

# Streamlit Interface
st.subheader("2. Choose Traversal Type")
start_node = st.selectbox("Select Starting Node:", list(graph.keys()))
algorithm = st.radio("Choose Algorithm:", ["Breadth-First Search (BFS)", "Depth-
First Search (DFS)"])

if st.button("Run Traversal"):
    st.markdown("---")

    # --- BFS ---
    if algorithm == "Breadth-First Search (BFS)":
        order, levels = bfs_with_levels(graph, start_node)

        st.success("Traversal Order (BFS): " + " → ".join(order))

        st.markdown("###  BFS Levels")
        for lvl in sorted(levels.keys()):
            st.write(f"**Level {lvl}:** {', '.join(levels[lvl])}")

    # --- DFS ---
    else:
        order, steps = dfs_with_steps(graph, start_node)

        st.success("Traversal Order (DFS): " + " → ".join(order))

        st.markdown("###  DFS Step-by-Step Process")
        for s in steps:
            st.write(s)

# Explanation Section
st.markdown("---")
st.subheader("3. Algorithm Explanation")

st.markdown("""
###  Breadth-First Search (BFS)
```

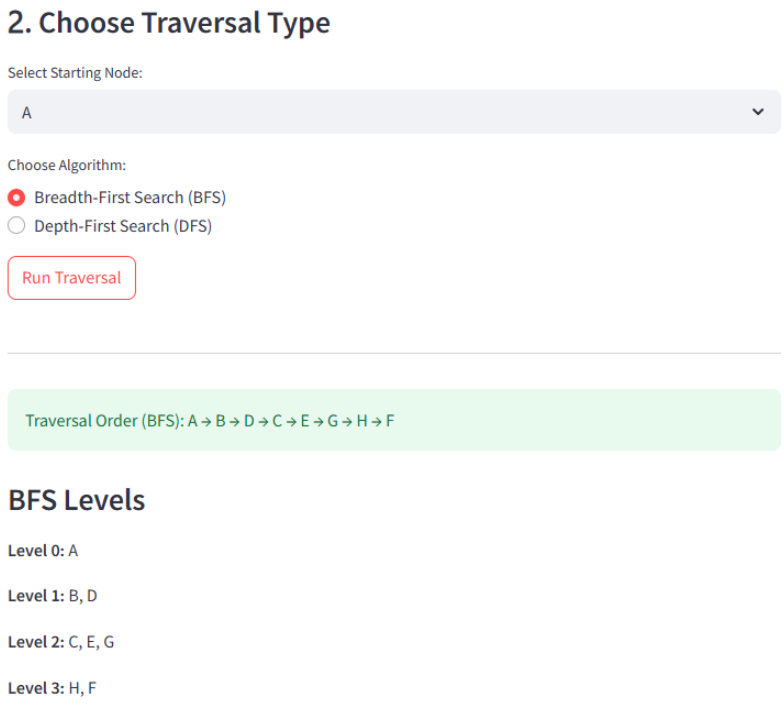
```
- Explores neighbors **level-by-level**.
- Uses a **queue (FIFO)**.
- Good for shortest path in unweighted graphs.

###  Depth-First Search (DFS)
- Explores as **deep as possible** before backtracking.
- Uses **recursion (implicit stack)**.
- Good for topological sorting, component detection.
""")


st.markdown("###  Complexity Summary")
st.table({
    "Algorithm": ["BFS", "DFS"],
    "Time Complexity": ["O(V + E)", "O(V + E)"],
    "Space Complexity": ["O(V)", "O(V)"]
})

st.markdown("---")
st.caption("Developed with Streamlit for BSD3513 Lab Report 1 – AI Search
Algorithms.")
```

Streamlit Output :



Breadth-First Search (BFS) was performed on the directed graph starting from node **A**, following alphabetical tie-breaking when multiple neighbours were available. The traversal explores nodes level by level, systematically visiting all nodes at a given depth before moving to the next. Starting at **A**, its neighbours **B** and **D** were discovered and added to the queue, forming Level 1. Expanding Level 1, nodes **B** and **D** led to the discovery of **C**, **E**, and **G**, which constituted Level 2. Continuing the process, Level 2 nodes **C**, **E**, and **G** revealed **H** and **F** as Level 3 nodes. Throughout the traversal, previously visited nodes were not revisited, ensuring an efficient exploration of the graph.

The final BFS traversal order was **A → B → D → C → E → G → H → F**, reflecting the systematic level-wise exploration. The levels of the graph are clearly defined as Level 0: **A**, Level 1: **B, D**, Level 2: **C, E, G**, and Level 3: **H, F**. This traversal demonstrates BFS's capability to capture the hierarchical structure of a directed graph while adhering to a deterministic tie-breaking rule, providing a clear and organized overview of node connectivity and discovery order.

Depth-First Search (DFS) was performed on the directed graph starting from node **A**, following alphabetical tie-breaking when multiple neighbours were available. DFS explores nodes by diving as deep as possible along each branch before backtracking. Starting at **A**, the traversal first visited **B**, then **C**, exploring the deepest path along **B** → **C**. Backtracking from **C**, the search continued to **E**, then **H**, from which **F** and **G** were visited in alphabetical order. Finally, after exhausting all paths from **H** and backtracking through previous nodes, the algorithm visited **D**, completing the traversal.
The final DFS traversal order is **A** → **B** → **C** → **E** → **H** → **F** → **G** → **D**, reflecting the depth-first exploration of the graph while respecting the alphabetical tie-breaking rule. This traversal demonstrates DFS's ability to explore each branch fully before moving to the next, producing a path that highlights the hierarchical depth and connectivity of nodes within the directed graph.

Deployment :
Github link: https://github.com/zahinzikrizawawi/Lab_Report_1
Streamlit app : https://labreport1sd23019.streamlit.app/