PRACTICAL-1

Aim: Overview of Artificial Intelligence System.

- Artificial Intelligence (AI) is the branch of computer science that focuses on creating systems or machines capable of performing tasks that typically require human intelligence. These tasks include understanding natural language, recognizing patterns, solving problems, learning from experience, and making decisions.

Area of Artificial Intelligence:

- The **areas of Artificial Intelligence (AI)** refer to the different domains or fields of expertise within AI, each focusing on specific aspects of creating intelligent systems.

Machine Learning (ML)

- **Description:** Enables machines to learn from data and improve their performance over time without explicit programming.
- **Techniques:** Supervised Learning, Unsupervised Learning, Reinforcement Learning.
- **Applications:**
- Recommendation systems (e.g., Netflix, Amazon).
- Predictive analytics (e.g., weather forecasting).
- Fraud detection in finance.

Natural Language Processing (NLP)

- **Description:** Allows machines to understand, interpret, and generate human language.
- **Key Areas:** Sentiment analysis, text summarization, language translation, and speech recognition.
- **Applications:**

o Chatbots (e.g., ChatGPT). o Voice assistants (e.g., Siri, Alexa).

o Machine translation (e.g., Google Translate).

Robotics

- **Description:** Integrates AI with physical systems to create intelligent robots that interact with the real world.
- **Applications:**

o Industrial robots for manufacturing. o Autonomous drones.

- Personal robots (e.g., robot assistants like Roomba).

Data Mining and Knowledge Discovery

- **Description:** Extracts useful patterns and knowledge from large datasets.
- **Applications:**
  - Market basket analysis. o Social network analysis.
  - Anomaly detection.

Deep Learning

- **Description:** A subset of ML focused on neural networks with many layers to learn from vast amounts of data.
- **Applications:**
  - Generative AI models (e.g., DALL-E for image generation). o Natural language models (e.g., GPT).
  - Autonomous driving systems.

Key Features of AI:

1. **Machine Learning (ML):** A subset of AI that involves training algorithms on data so they can learn patterns and make predictions or decisions without explicit programming.
   - Example: Predicting customer behavior or recognizing objects in images.
2. **Natural Language Processing (NLP):** Enables machines to understand, interpret, and respond to human language.
   - Example: Chatbots and virtual assistants like Siri or Alexa.
3. **Computer Vision:** Allows machines to interpret and analyze visual data from the world.
   - Example: Facial recognition systems or autonomous vehicles.
4. **Reasoning and Problem-Solving:** AI systems can simulate logical reasoning and solve complex problems.
   - Example: Chess-playing AI or optimization software.
5. **Robotics:** Combines AI with physical hardware to create intelligent machines capable of interacting with their environment.
   - Example: Industrial robots or home assistants like robot vacuums.

Why we use Artificial intelligence

- We use Artificial Intelligence (AI) because it offers significant benefits across various industries, enabling us to solve complex problems, improve efficiency, and create new possibilities that were previously unimaginable.

Handling Large Volumes of Data

- **Why:** AI systems can analyze and process vast amounts of data at speeds beyond human capabilities, uncovering insights and patterns.
- **Examples:** ○ Big data analytics in business. ○ Customer behavior prediction. ○ Real-time monitoring in cybersecurity.

Cost Efficiency

- **Why:** By automating processes and improving efficiency, AI reduces operational costs over time.
- **Examples:**
  - Optimizing supply chain logistics. ○ Predictive maintenance in machinery.
  - Virtual assistants replacing certain human roles in customer service.

Applications of AI:

- **Healthcare:** Disease diagnosis, personalized treatment plans, and drug discovery.
- **Finance:** Fraud detection, risk assessment, and algorithmic trading.
- **Education:** Personalized learning experiences and automated grading systems.
- **Transportation:** Autonomous vehicles and traffic management.
- **Entertainment:** Content recommendations on platforms like Netflix or YouTube.

Practical-2

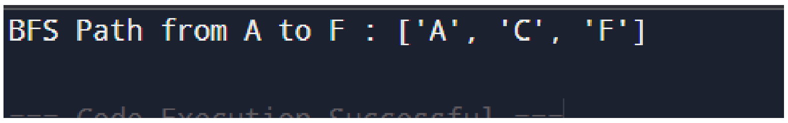
Aim: Write a program to implement BFS(water jug problem or any AI search problem) in python.

Code:

```
from collections import deque


def bfs(graph, start, goal):
    queue = deque([(start, [start])])
    visited = set()


    while queue:
        node, path = queue.popleft()
        if node == goal:
            return path


        visited.add(node)
        for neighbor in graph.get(node, []):
            if neighbor not in visited:
                queue.append((neighbor, path + [neighbor]))
                visited.add(neighbor)


    return None
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
```

```
start = 'A'

goal = 'F'

path = bfs(graph, start, goal)

print("BFS Path from", start, "to", goal, ":", path)
```

<u>Output:</u>



```
BFS Path from A to F : ['A', 'C', 'F']

=== Code Execution Successful ===
```

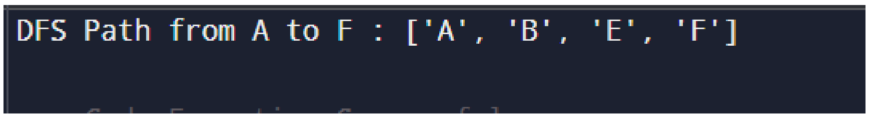
Practical-3

Aim: Write a program to implement DFS(water jug problem or any AI search problem) in python.

Code:

```
def dfs(graph, start, goal, path=None, visited=None):
    if path is None:
        path = []
    if visited is None:
        visited = set()

    path.append(start)
    visited.add(start)

    if start == goal:
        return path

    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            result = dfs(graph, neighbor, goal, path[:], visited)
            if result:
                return result

    return None
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

start = 'A'
goal = 'F'
path = dfs(graph, start, goal)
print("DFS Path from", start, "to", goal, ":", path)
```

Output:



```
DFS Path from A to F : ['A', 'B', 'E', 'F']
```

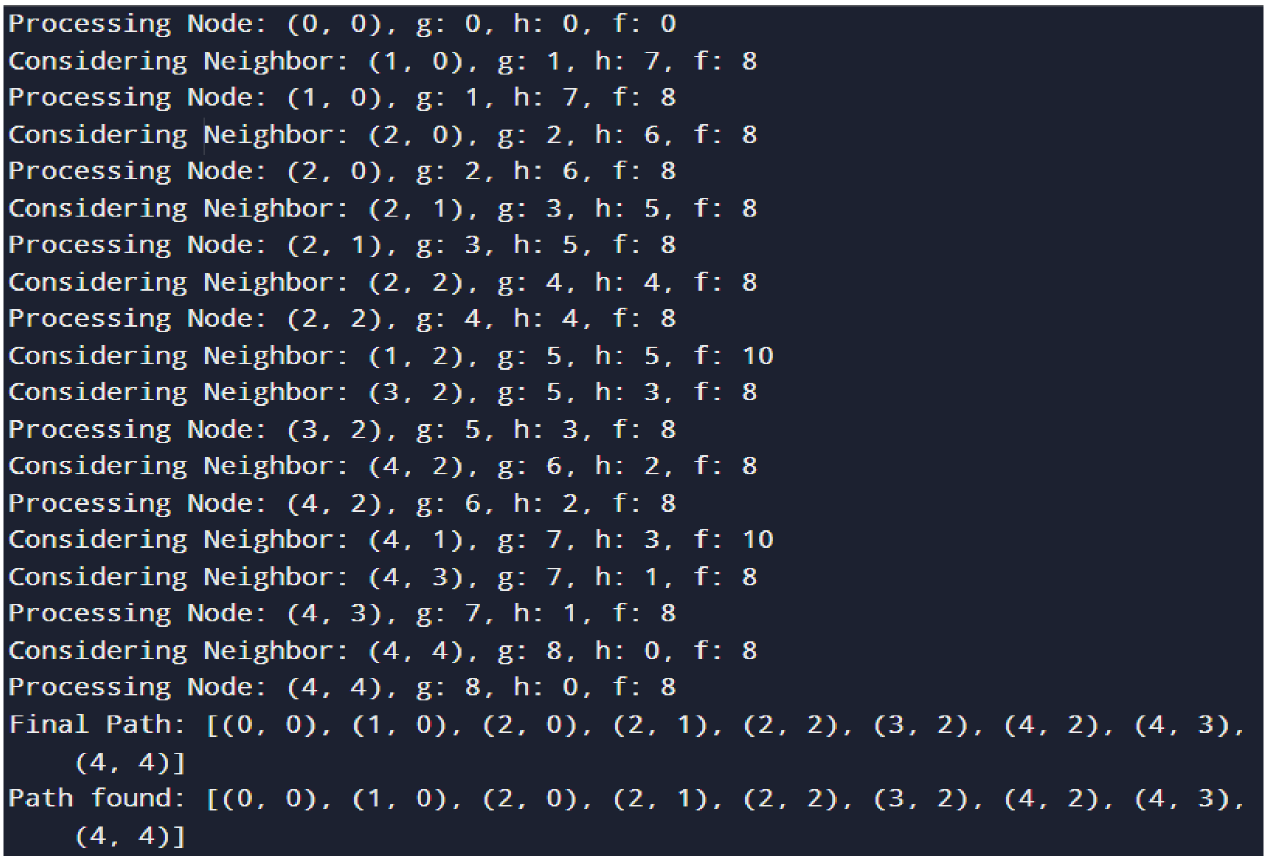
Practical-4

Aim: write a program to implement A* algorithm in python.

Code:

```
import heapq
class Node:
    def __init__(self, position, parent=None):
        self.position = position
        self.parent = parent
        self.g = 0
        self.h = 0
        self.f = 0

    def __lt__(self, other):
        return self.f < other.f

def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def astar(grid, start, end):
    open_list = []
    closed_set = set()
    start_node = Node(start)
    start_node.g = start_node.h = start_node.f = 0
    end_node = Node(end)
    heapq.heappush(open_list, start_node)

    while open_list:
        current_node = heapq.heappop(open_list)
        closed_set.add(current_node.position)
    print(f"Processing Node: {current_node.position}, g: {current_node.g}, h: {current_node.h}, f: {current_node.f}")


    if current_node.position == end_node.position:
        path = []
        while current_node:
            path.append(current_node.position)
            current_node = current_node.parent
        print("Final Path:", path[::-1])
        return path[::-1]


    neighbors = [(0, -1), (0, 1), (-1, 0), (1, 0)]
    for move in neighbors:
        node_position = (current_node.position[0] + move[0], current_node.position[1] + move[1])
        if (0 <= node_position[0] < len(grid) and 0 <= node_position[1] < len(grid[0]) and grid[node_position[0]][node_position[1]] == 0):
            if node_position in closed_set:
                continue


            neighbor = Node(node_position, current_node)
            neighbor.g = current_node.g + 1
            neighbor.h = heuristic(neighbor.position, end_node.position)
            neighbor.f = neighbor.g + neighbor.h


            print(f"Considering Neighbor: {neighbor.position}, g: {neighbor.g}, h: {neighbor.h}, f: {neighbor.f}")


            if not any(open_node.position == neighbor.position and open_node.f <= neighbor.f for open_node in open_list):
                heapq.heappush(open_list, neighbor)

    print("No Path Found")
    return None
grid = [
    [0, 1, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 1, 0],
    [1, 1, 0, 1, 0],
    [0, 0, 0, 0, 0]
]
start = (0, 0)
end = (4, 4)
path = astar(grid, start, end)
print("Path found:", path)
```

Output:



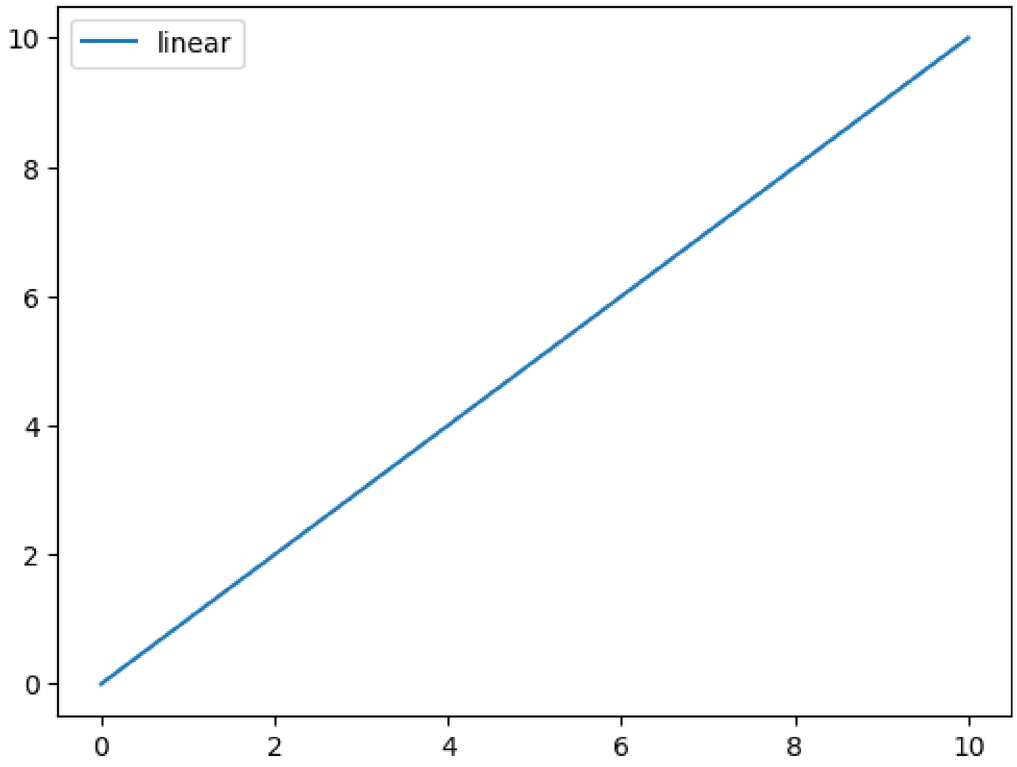
```
Processing Node: (0, 0), g: 0, h: 0, f: 0
Considering Neighbor: (1, 0), g: 1, h: 7, f: 8
Processing Node: (1, 0), g: 1, h: 7, f: 8
Considering Neighbor: (2, 0), g: 2, h: 6, f: 8
Processing Node: (2, 0), g: 2, h: 6, f: 8
Considering Neighbor: (2, 1), g: 3, h: 5, f: 8
Processing Node: (2, 1), g: 3, h: 5, f: 8
Considering Neighbor: (2, 2), g: 4, h: 4, f: 8
Processing Node: (2, 2), g: 4, h: 4, f: 8
Considering Neighbor: (1, 2), g: 5, h: 5, f: 10
Considering Neighbor: (3, 2), g: 5, h: 3, f: 8
Processing Node: (3, 2), g: 5, h: 3, f: 8
Considering Neighbor: (4, 2), g: 6, h: 2, f: 8
Processing Node: (4, 2), g: 6, h: 2, f: 8
Considering Neighbor: (4, 1), g: 7, h: 3, f: 10
Considering Neighbor: (4, 3), g: 7, h: 1, f: 8
Processing Node: (4, 3), g: 7, h: 1, f: 8
Considering Neighbor: (4, 4), g: 8, h: 0, f: 8
Processing Node: (4, 4), g: 8, h: 0, f: 8
Final Path: [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (3, 2), (4, 2), (4, 3),
    (4, 4)]
Path found: [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (3, 2), (4, 2), (4, 3),
    (4, 4)]
```

Practical-5

Aim: Python packages related to Ai.

- **Numpy:** NumPy, or Numerical Python, is a Python package that provides a multidimensional array object and tools for scientific computing and data analysis:
  - **Arrays**
    NumPy arrays are tables of elements, usually numbers, that are indexed by a tuple of positive integers. The number of dimensions in an array is called its rank, and the size of the array along each dimension is called its shape.
  - **Operations**
    NumPy provides routines for performing operations on arrays, including mathematical, logical, and statistical operations, linear algebra, and random number generation.
    Code:
    ```
    import numpy as np
    x=np.array([[1,2], [3,4],[5,6]])
    y=np.array([1,2,3])
    mean=np.mean(x, axis=0)
    print("Mean of features: ",mean)
    ```
    Output:
    ```
    Mean of features:  [3. 4.]
    ```

- **Matplotlib:** Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.
  - **Code:**
    ```
    import matplotlib.pyplot as plt
    import numpy as np
    x=np.linspace(0, 10, 100)
    plt.plot(x, x,label= 'linear')
    plt.legend()
    plt.show()
    ```
    Output:



- **Tensorflow:** TensorFlow is an open-source software library that helps developers build machine learning applications.
  - **How it works**
    TensorFlow uses data flow graphs to describe machine learning algorithms. Nodes in the graph represent mathematical operations, and edges represent the multidimensional data arrays (tensors) that flow between them.
  - **Features**
    TensorFlow can run on a variety of hardware platforms and operating environments, including GPUs, CPUs, and TPUs. It's portable and can be used on portable devices, desktops, and high-end servers.

    <u>Code:</u>

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize pixel values
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

model = tf.keras.Sequential([
   tf.keras.layers.Flatten(input_shape=(28, 28)),
   tf.keras.layers.Dense(128, activation='relu'),
   tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5, batch_size=64)

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')
```

Output:

```
Epoch 1/5
938/938 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step -
accuracy: 0.8594 - loss: 0.5084
Epoch 2/5
938/938 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step -
accuracy: 0.9569 - loss: 0.1470
Epoch 3/5
938/938 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step -
accuracy: 0.9701 - loss: 0.1011
Epoch 4/5
938/938 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step -
accuracy: 0.9790 - loss: 0.0710
Epoch 5/5
938/938 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step -
accuracy: 0.9836 - loss: 0.0568
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step -
accuracy: 0.9730 - loss: 0.0906
Test accuracy: 0.9764
```

- **Scikit-learn:** Scikit-learn is an open-source library that helps simplify the process of implementing machine learning and statistical models in Python. It provides a consistent interface for a wide range of supervised and unsupervised machine learning algorithms.
  - **How it work**
    Scikit-learn works with numeric data stored as NumPy arrays, SciPy sparse matrices, and other data types that can be converted to numeric arrays. It integrates well with other Python libraries, such as Matplotlib, Plotly, NumPy, and Pandas.
    Code:

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
iris=datasets.load_iris()
x=iris.data
y=iris.target
```

```
clf=DecisionTreeClassifier()
clf.fit(x,y)
```
Output:

```
 DecisionTreeClassifier?i
DecisionTreeClassifier()
```

- **Seaborn**: Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on top matplotlib library and is also closely integrated with the data structures from pandas.
  Seaborn aims to make visualization the central part of exploring and understanding data. It provides dataset-oriented APIs so that we can switch between different visual representations for the same variables for a better understanding of the dataset.
- **Code:**

```
import seaborn as sns

df = sns.load_dataset("iris")
print(df)
```
Output:

```
   sepal_length  sepal_width  petal_length  petal_width    species
0           5.1          3.5           1.4          0.2     setosa
1           4.9          3.0           1.4          0.2     setosa
2           4.7          3.2           1.3          0.2     setosa
3           4.6          3.1           1.5          0.2     setosa
4           5.0          3.6           1.4          0.2     setosa
..          ...          ...           ...          ...        ...
145         6.7          3.0           5.2          2.3  virginica
146         6.3          2.5           5.0          1.9  virginica
147         6.5          3.0           5.2          2.0  virginica
148         6.2          3.4           5.4          2.3  virginica
149         5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]
```