AI practical

1. BFS

```
def bfs(graph, start):
    visited=[]
    queue=[start]

    while queue:
        node=queue.pop(0)
        if node not in visited:
            print(node, end=' ')
            visited.append(node)
            queue.extend(graph[node])

graph = {
    'A': ['B','C'],
    'B': ['D','E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

bfs (graph, 'A')
```

2. IDFS

```
def iterative_dfs(graph, start):
    visited = []
    stack = [start]

    while stack:
        node = stack.pop()
        if node not in visited:
            print(node, end=" ")
            visited.append(node)
            stack.extend(reversed(graph[node]))

graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

iterative_dfs(graph, 'A')
```

3. A*Search Algorithm

```
def a_star(graph, start, goal, heuristic):
    open_list = [start]
    closed_list = []
    g = {start: 0}
    parent = {start: None}

    while open_list:
        current = min(open_list, key=lambda x: g[x] + heuristic[x])
        if current == goal:
            path = []
            while current:
                path.append(current)
                current = parent[current]
            return path[::-1]
        open_list.remove(current)
        closed_list.append(current)

        for neighbor, cost in graph[current].items():
            if neighbor in closed_list:
                continue
            temp_g = g[current] + cost
            if neighbor not in open_list or temp_g < g.get(neighbor, float('inf')):
```

```
                g[neighbor] = temp_g
                parent[neighbor] = current
                if neighbor not in open_list:
                    open_list.append(neighbor)
    return None

    graph = {
    'A': {'B': 1, 'C': 3},
    'B': {'D': 3, 'E': 1},
    'C': {'F': 5},
    'D': {'G': 2},
    'E': {'G': 1},
    'F': {'G': 2},
    'G': {}
}

heuristic = {'A':7, 'B':6, 'C':5, 'D':3, 'E':2, 'F':1, 'G':0}
print(a_star(graph, 'A', 'G', heuristic))
```

4. Implement the decision tree learning algorithm to build a decision tree for a given dataset.

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Step 1: Load dataset from a CSV file
# Replace 'data.csv' with your actual CSV file name
data = pd.read_csv('data.csv')

# Step 2: Separate features (X) and target (y)
# Example: assuming the last column is the target
X = data.iloc[:, :-1]   # all columns except last
y = data.iloc[:, -1]    # last column (target)

# Step 3: Split dataset into training and testing parts (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Create and train Decision Tree model
model = DecisionTreeClassifier(criterion='entropy')  # or use 'gini'
model.fit(X_train, y_train)

# Step 5: Make predictions
y_pred = model.predict(X_test)

# Step 6: Check accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Step 7 (optional): Visualize the decision tree
plt.figure(figsize=(12,8))
plot_tree(model, filled=True, feature_names=X.columns, class_names=True)
plt.show()
```

5. Ada Boost

```
# Import required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Step 1: Load dataset
data = pd.read_csv('data.csv')  # Replace with your dataset file

# Step 2: Separate features and target
X = data.iloc[:, :-1]   # all columns except the last one
y = data.iloc[:, -1]    # last column is the target

# (Optional) If dataset has text values, convert them to numbers
X = pd.get_dummies(X)  # One-hot encode categorical features
y = y.astype('category').cat.codes  # Convert target to numeric codes if needed
```

```
# Step 3: Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Initialize AdaBoost model
# Using Decision Tree as the base learner (stump)
base_estimator = DecisionTreeClassifier(max_depth=1)
model = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50, learning_rate=1.0, random_state=42)

# Step 5: Train the model
model.fit(X_train, y_train)

# Step 6: Make predictions
y_pred = model.predict(X_test)

# Step 7: Evaluate performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

6. Naive Bayes algorithm for classification

```
# Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Step 1: Load the dataset
data = pd.read_csv('data.csv')   # Replace with your file name

# Step 2: Separate features and target
X = data.iloc[:, :-1]   # all columns except last (features)
y = data.iloc[:, -1]    # last column (target)

# Step 3: Handle categorical data if present
# Convert categorical variables into numeric using one-hot encoding
X = pd.get_dummies(X)
# Convert target labels to numeric if they are strings
y = y.astype('category').cat.codes

# Step 4: Split the dataset into training and testing parts
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Initialize and train the Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)

# Step 6: Make predictions
y_pred = model.predict(X_test)

# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```