EX.NO: 1 IMPLEMENTATION OF UNINFORMED SEARCH ALGORITHM (BFS,DFS)

BREADTH FIRST SEARCH:

PROGRAM

```
graph = '5': ['3','7'],
'3': ['2', '4'], ‘7’: ['8'],
‘2’:[],
'4': ['8'].
8:[];
visited = []
queue=[]
def bfs(visited. node):
visited.append(node)
queue.append(node)
while queue:
mqueue.pop(0)
print (m, end="") for neighbour in graph[m]:
if neighbour not in visited: visited.append(neighbour)
queue.append(neighbour)
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')
```

OUTPUT

Following is the Breadth-First Search 537248

DEPTH FIRST SEARCH:

PROGRAM

```
graph = [
'5': ['3',7'], '3': ['2', '4'].
7': ['8'].
2:[]
'4': ['8'].
'8': []
1
the SHA ENGINEERING COLLEGE
visited = set()
def dfs(visited, graph, node):
if node not in visited:
print (node)
visited.add(node) for neighbour in grap
dfs(visited, graph, neighbour)
print("Following the Depth-First Search")
dfs(visited, graph, '5')
```

OUTPUT

```
    Following is the Depth-First Search
5
3
2
4
8
7
```

EX.NO: 2 IMPLEMENTATION OF INFORMED SEARCH ALGORITHM

PROGRAM

```
def aStarAlgo(start_node, stop_node):
    open_set = set([start_node])
    closed_set = set()
    g = {}
    parents = {}
    g[start_node] = 0
    parents[start_node] = start_node

    while open_set:
        n = None
        for v in open_set:
            if n is None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v

        if n is None:
            print('Path does not exist!')
            return None

        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path

        open_set.remove(n)
        closed_set.add(n)

        for m, weight in get_neighbors(n):
            if m not in open_set and m not in closed_set:
                open_set.add(m)
                parents[m] = n
```

```
                g[m] = g[n] + weight
            else:
                if g[m] > g[n] + weight:
                    g[m] = g[n] + weight
                    parents[m] = n
                    if m in closed_set:
                        closed_set.remove(m)
                        open_set.add(m)

    print('Path does not exist!')
    return None

def get_neighbors(v):
    return Graph_nodes.get(v, None)

def heuristic(n):
    H_dist = {
        'A': 10, 'B': 8, 'C': 5, 'D': 7, 'E': 3,
        'F': 6, 'G': 5, 'H': 3, 'I': 1, 'J': 0
    }
    return H_dist[n]

Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('J', 5)],
    'F': [('G', 1), ('H', 7)],
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('J', 3)],
}

aStarAlgo('A', 'J')
```

OUTPUT

Path found: ['A', 'F', 'G', 'I', 'J']

EX.NO: 3 IMPLEMENTATION CANDIDATE ELIMINATION ALGORITHM

PROGRAM

```
import csv

file_path = r"C:\ai\trainingexamples.csv"

try:
    with open(file_path, mode='r') as f:
        csv_file = csv.reader(f)
        data = list(csv_file)

        specific = data[1][:-1]
        general = [['?' for _ in range(len(specific))] for _ in range(len(specific))]

        for i in data[1:]:
            if i[-1].strip().lower() == "yes":
                for j in range(len(specific)):
                    if i[j] != specific[j]:
                        specific[j] = "?"
                        general[j][j] = "?"
            elif i[-1].strip().lower() == "no":
                for j in range(len(specific)):
                    if i[j] != specific[j]:
                        general[j][j] = specific[j]
                    else:
                        general[j][j] = "?"

            print(f"\nStep {data.index(i)} of Candidate Elimination Algorithm")
            print("Specific Hypothesis:", specific)
            print("General Hypothesis:", genera
        gh = []
        for i in general:
            if any(j != '?' for j in i):
                gh.append(i)

        print("\nFinal Specific Hypothesis:\n", specific)
        print("\nFinal General Hypothesis:\n", gh)
except OSError as e:
    print(f"Error opening file: {e}")
```

OUTPUT

Step 1 of Candidate Elimination Algorithm
Specific Hypothesis: ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
General Hypothesis: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 2 of Candidate Elimination Algorithm
Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
General Hypothesis: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 3 of Candidate Elimination Algorithm
Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
General Hypothesis: [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Step 4 of Candidate Elimination Algorithm
Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
General Hypothesis: [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific hypothesis:
['Sunny', 'Warm', '?', 'Strong', '?', '?']

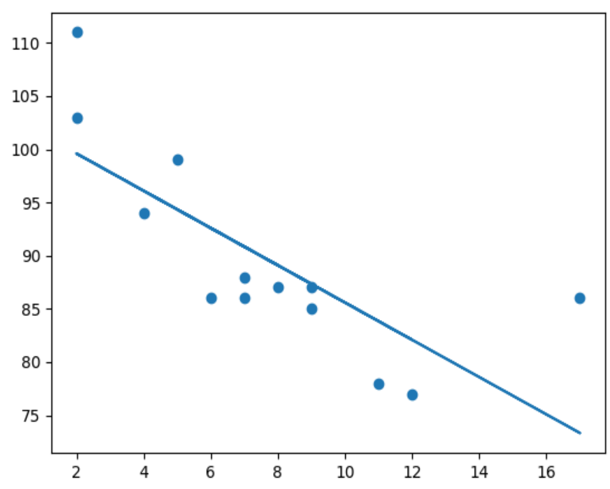
Final General hypothesis:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

EX.NO: 4 IMPLEMENT LINEAR REGRESSION

PROGRAM

```
import matplotlib.pyplot as plt
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
    return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

OUTPUT



EX.NO:5 IMPLEMENT BACK-PROPOGATION ALGORITHM

PROGRAM

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X / np.amax(X, axis=0)  # Maximum of X array longitudinally
y = y / 100

# Sigmoid Function
def sigmoid(x):
return 1 / (1 + np.exp(-x))

# Derivative of Sigmoid Function
def derivatives_sigmoid(x):
return x * (1 - x)

# Variable initialization
epoch = 5000  # Setting training iterations
lr = 0.1      # Setting learning rate
inputlayer_neurons = 2  # Number of features in data set
hiddenlayer_neurons = 3 # Number of hidden layer neurons
output_neurons = 1      # Number of neurons at output layer

# Weight and bias initialization
wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))
bh = np.random.uniform(size=(1, hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons, output_neurons))
bout = np.random.uniform(size=(1, output_neurons))

# Training
for i in range(epoch):
# Forward Propagation
hinp1 = np.dot(X, wh)
hinp = hinp1 + bh
hlayer_act = sigmoid(hinp)
outinp1 = np.dot(hlayer_act, wout)
outinp = outinp1 + bout
```

```
output = sigmoid(outinp)

# Backpropagation
EO = y - output
outgrad = derivatives_sigmoid(output)
d_output = EO * outgrad
EH = d_output.dot(wout.T)

# How much hidden layer weights contributed to error
hiddengrad = derivatives_sigmoid(hlayer_act)
d_hiddenlayer = EH * hiddengrad

# Update weights and biases
wout += hlayer_act.T.dot(d_output) * lr
wh += X.T.dot(d_hiddenlayer) * lr

print("Input:\n" + str(X))
print("Actual Output:\n" + str(y))
print("Predicted Output:\n", output)
```

OUTPUT

Input:

```
[[0.66666667 1.        ]
[0.33333333 0.55555556]
[1.         0.66666667]]
```

Actual Output:

```
[[0.92]
[0.86]
[0.89]]
```

Predicted Output:

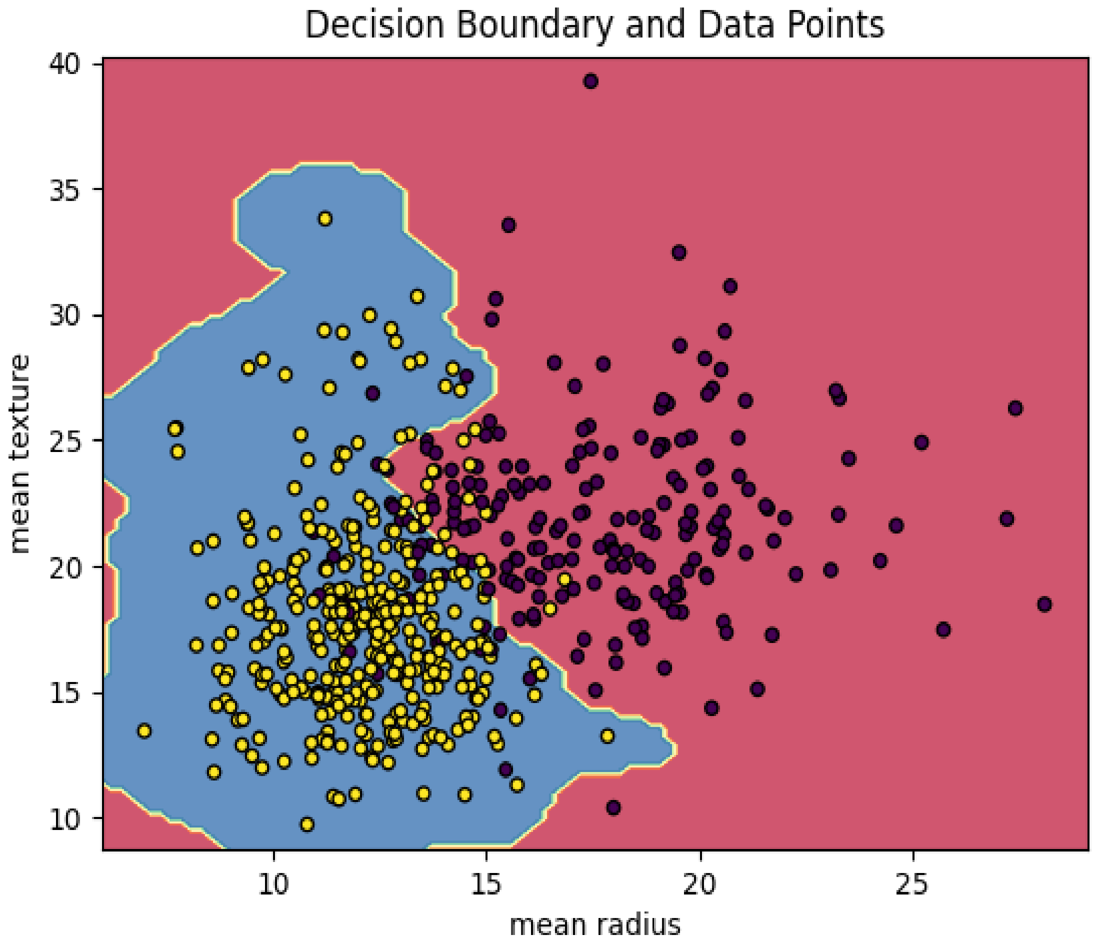
```
[[0.89597569]
[0.87889218]
[0.89500469]]
```

EX.NO:6 IMPLEMENT SUPPORT VECTOR MACHINE ALGORITHM

PROGRAM

```
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

# Load the datasets
cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

# Build and train the model
svm = SVC(kernel=”rbf”, gamma=0.5, C=1.0)
svm.fit(X, y)

# Plot Decision Boundary
DecisionBoundaryDisplay.from_estimator(
    svm,
    X,
    response_method=”predict”,
    cmap=plt.cm.Spectral,
    alpha=0.8,
    xlabel=cancer.feature_names[0],
    ylabel=cancer.feature_names[1]
)

# Scatter plot
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolors=”k”)
plt.title(‘Decision Boundary and Data Points’)
plt.xlabel(cancer.feature_names[0])
plt.ylabel(cancer.feature_names[1])
plt.show()
```

OUTPUT



Decision Boundary and Data Points
40
35
30
25
20
15
10
mean texture
10
15
20
25
mean radius

EX.NO:7 IMPLEMENT DECISION TREE ALGORITHM

PROGRAM

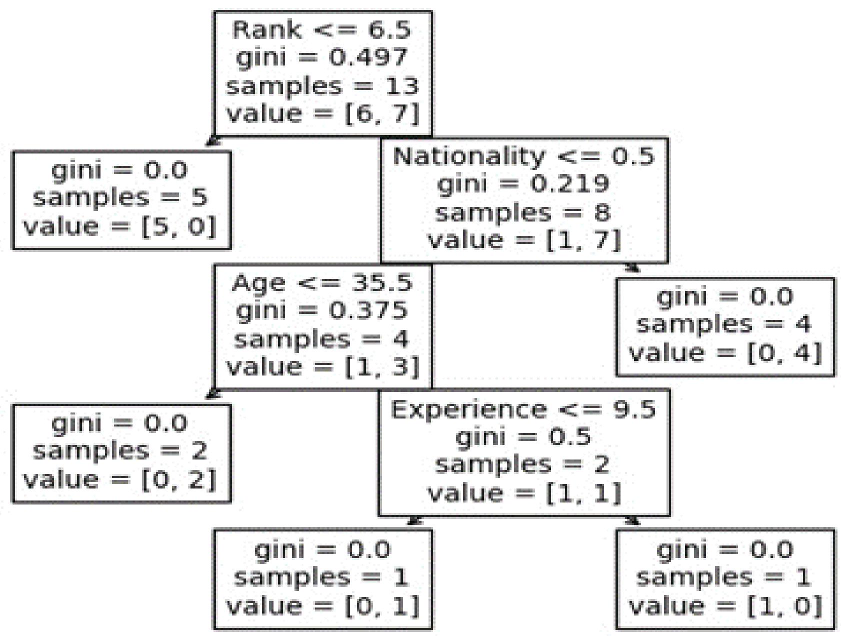
```
import sys
import matplotlib
matplotlib.use('Agg')

import pandas
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

df = pandas.read_csv("data.csv")

d = {'UK': 0, 'USA': 1, 'N': 2}
df['Nationality'] = df['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)

features = ['Age', 'Experience', 'Rank', 'Nationality']

X = df[features]
y = df['Go']

dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, y)

tree.plot_tree(dtree, feature_names=features)

plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

data.csv:

```
Age,Experience,Rank,Nationality,Go
36,10,9,UK,NO
42,12,4,USA,NO
23,4,6,N,NO
```

52,4,4,USA,NO
43,21,8,USA,YES
44,14,5,UK,NO
66,3,7,N,YES
35,14,9,UK,YES
52,13,7,N,YES
35,5,9,N,YES
24,3,5,USA,NO
18, 3,7,UK,YES
45, 9,9,UK,YES

OUTPUT

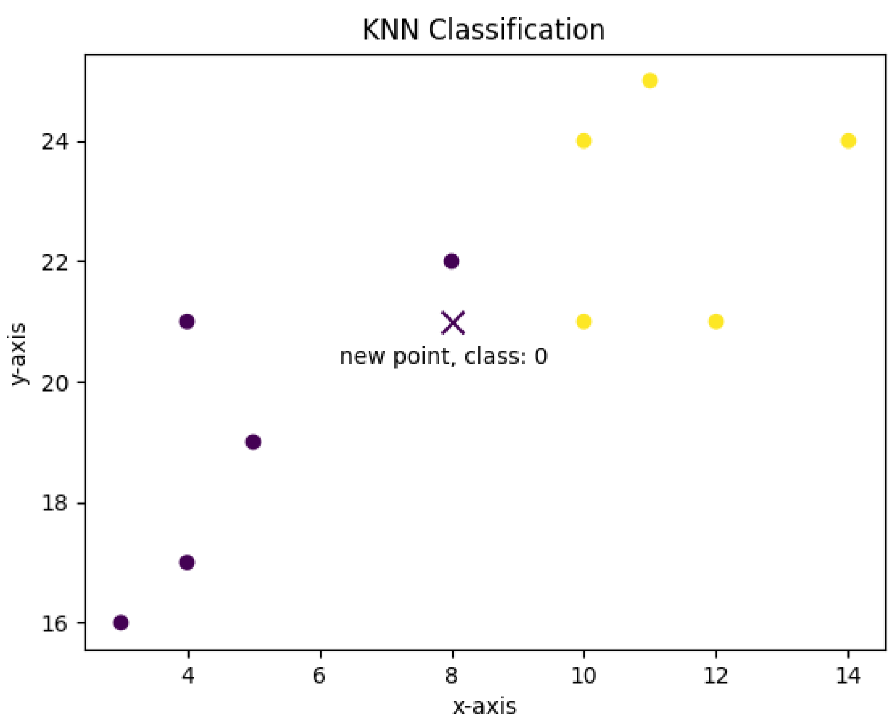


EX.NO:8 IMPLEMENT K-NEAREST NEIGHBORS ALGORITHM

PROGRAM

```
import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14, 8, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]

plt.scatter(x, y, c=classes)
new_x = 8
new_y = 21
new_point = [(new_x, new_y)]

prediction = knn.predict(new_point)

plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```

OUTPUT



EX.NO:9 IMPLEMENT K- MEANS CLUSTERING ALGORITHM

PROGRAM

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

data = list(zip(x, y))
inertias = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1, 11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

OUTPUT