INDEX

NAME: Caroline Suja. J S STD.: CSE SEC.: A ROLL NO.: 220701048 SUB.: POAI

S. No.	Date	Title	~~Page No.~~ marks	Teacher's Sign / Remarks
1.	9/8/24	8 Queens	9	
2	16/8/24	Depth First Search	9	
3	13/9/24	DFS - Water Jug	9	
4	20/9/24	A* Algorithm	10	
5	4/10/24	MinMax Algorithm	10	
6	20/9/24	Artificial Neural	10	
	11/10/24	Network - Regression	10	
7	18/10/24	Decision Tree	10	
8	18/10/24	K Means	10	
9	25/10/24	Introduction to Prolog	10	
10	25/10/24	Prolog Family Tree	10	

completed

EXP: 1 8 QUEENS PROBLEM

9/8/24

Write a program to solve 8 Queens problem

CODE:

```
def share_diagonal(x0, y0, x1, y1):
    dx = abs(x0 - x1)
    dy = abs(y0 - y1)
    return dy == dx

def col_clashes(bs, c):
    for i in range(c):
        if share_diagonal(i, bs[i], c, bs[c]):
            return True
    return False

def has_clashes(the_board):
    for col in range(1, len(the_board)):
        if col_clashes(the_board, col):
            return True
    return False

def main():
    import random
    rng = random.Random()
    bd = list(range(8))
    num_found = 0
    tries = 0
    result = []
    while num_found < 10:
        rng.shuffle(bd)
        tries += 1
        if not has_clashes(bd) and bd not in result:
            print("Found solution {0} in {1} tries".format(bd, tries))
            tries = 0
            num_found += 1
            result.append(list(bd))
    print(result)

main()
```

OUTPUT:

4

Found solution [1, 3, 0, 2] in 4 tries
Found solution [2, 0, 3, 1] in 4 tries

. . Q .
Q . . .
. . . Q
. Q . .

8

Found solution [2, 5, 1, 4, 7, 0, 6, 3] in 115 tries
Found solution [4, 1, 3, 5, 7, 2, 0, 6] in 2 tries
Found solution [3, 1, 6, 4, 0, 7, 5, 2] in 930 tries
Found solution [5, 2, 0, 7, 4, 1, 3, 6] in 317 tries
Found solution [2, 6, 1, 7, 5, 3, 0, 4] in 1849 tries
Found solution [4, 2, 0, 6, 1, 7, 5, 3] in 56 tries
Found solution [4, 2, 1, 3, 7, 0, 2, 5] in 50 tries
Found solution [3, 1, 7, 4, 6, 0, 2, 5] in 451 tries
Found solution [4, 6, 7, 3, 1, 6, 2, 5] in 836 tries
Found solution [6, 3, 1, 7, 5, 0, 2, 4] in 151 tries

Q . . . . . .
. . . . . Q .
. . . Q . . .
. . . . . Q
. Q . . . . .
. . . Q . . .
. . . . . Q .
. . Q . . . .

RESULT:

Thus the 8 Queens problem is executed successfully

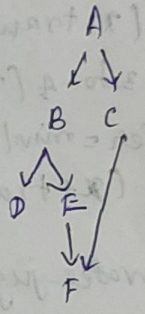
EXP: 2 DEPTH FIRST SEARCH

16/8/24

Solve any problem using Depth First Search

CODE:

```
def dfs(graph, start, visited = None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end = " ")
    for neighbour in graph[start]:
        if neighbour not in visited:
            dfs(graph, notebook, visited)

graph = {
'A' : ['B', 'C'],
'B' : ['D', 'E'],
'C' : ['F']
'D' : [],
'E' : ['F'],
'F' : []
}
dfs(graph, 'A')
```



OUTPUT:

A B D E F C

RESULT:

Thus DFS program is executed successfully

Exp: 3 WATER JOG USING DFS

4/10/24

```
def fill_4_gallon(x, y, x_max, y_max):
    return (x_max, y)

def fill_3_gallon(x, y, x_max, y_max):
    return (x, y_max)

def empty_4_gallon(x, y, x_max, y_max):
    return (0, y)

def empty_3_gallon(x, y, x_max, y_max):
    return (x, 0)

def pour_4_to_3(x, y, x_max, y_max):
    transfer = min(x, y_max - y)
    return (x - transfer, y + transfer)

def pour_3_to_4(x, y, x_max, y_max):
    transfer = min(y, x_max - x)
    return (x + transfer, y - transfer)

def dfs_water_jug(x_max, y_max, goal_x, visited=None, start=(0,0)):
    if visited is None:
        visite = set()
    stack = [start]
    while stack:
        state = stack.pop()
        x, y = state
        if state in visited:
            continue
        visited.add(state)
        print(f"visited state: {state}")
        if x == goal_x:
            print(f"Goal reached: {state}")
            return state

    next states =
[ fill_4gallon(x, y, x_max, y_max),
  fill_3_gallon(x, y, x_max, y_max),
  empty_4_gallon(x, y, x_max, y_max),
  empty_3_gallon(x, y, x_max, y_max),
  pour_4_to_3(x, y, x_max, y_max),
  pour_3_to_4(x, y, x_max, y_max)]
```

```
    for new_state in next_states:
        if new_state not in visited:
            stack.append(new_state)
    return None

x_max = 4
y_max = 3
goal_x = 2
dfs - water_jug(x_max, y_max, goal_x)
```

Output:

```
Visiting state: (0,0)
Visiting state: (0,3)
Visiting state: (3,0)
Visiting state: (3,3)
Visiting state: (4,2)
Visiting state: (4,0)
Visiting state: (1,3)
Visiting state: (1,0)
Visiting state: (0,1)
Visiting state: (4,1)
Visiting state: (2,3)
Goal reached: (2,3)
(2,3)
```

RESULT

Thus the Waterjug problem using DFS is executed successfully

EXP : 4 A* SEARCH ALGORITHM

13/9/24

CODE:

```
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 len(open_set) > 0:
        n = None
        for v in open_set:
            if n = None or g[v] + heuristic(v) <
                           g[n] + heuristic(n):
                n = v
        if n == stop_node or graph_nodes[n] == None:
            pass
        else
            if g[m] > g(n] + weight:
                g[m] = g(n] + weight
                parents[m] = n
            if m in closedset:
                closed_set.remove(m)
                open_set.add(m)
            if n == None:
                print("path does not exist")
                return none
            if n == stopnode:
                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)
    print("Path does not exist!")
    return None

def get_neighbours(v):
    if v in graph_nodes:
        return graph_nodes[v]
    else:
        return None

def heuristic(n):
    H = dist = {
        'A': 11
        'B': 6,
        'C': 99,
        'D': 1,
        'E': 7,
        'G': 0 }
    return H.dist[n]

graph_nodes = {
    'A': [('B', 2), ('E', 3)]
    'B': [('C', 1), ('G', 9)],
    'C': None,
    'E': [('D', 6)]
    'D': [('G', 1)], }

astaralgo ('A', G)
```

OUTPUT

Path found: ['A', 'E', 'D', G']

RESULT:

Thus the program is verified and executed successfully

EXP: 5 ARTIFICIAL NEURAL NETWORKS

20/9/24

AIM:

To implement artificial neural networks for an application in regression using python

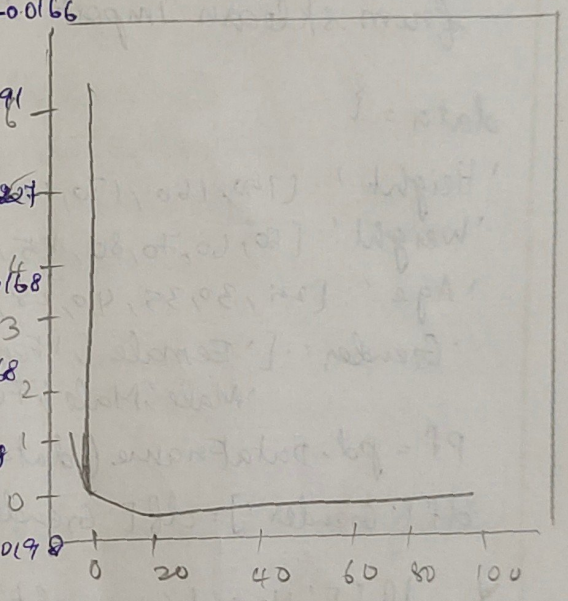
CODE:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt

np.random.seed(42)
X = np.random.rand(1000,3)
y = 3*X[:,0] + 2*X[:,1]**2 + 1.5*np.sin(X[:,2]*np.pi)
    + np.random.normal(0,0.1,1000)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,
                                   random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = Sequential()
model.add(Dense(10,input_dim=X_train.shape[1],activation='relu'))
model.add(Dense(10,activation='relu'))
model.add(Dense(1,activation='linear'))
model.compile(optimizer=Adam(learning_rate=0.01),loss='mean_squared
                                                       error')
history = model.fit(X_train,y_train,epochs=100,batch_size=32,
                    validation_split=0.2,verbose=1)
y_pred = model.predict(X_test)
```

```
mse = np.mean(y_test - y_pred.flatten()) ** 2)
print(f'Mean Squared Error : {mse:.4f}')

plt.figure(figsize = (12, 6))
plt.plot(history.history['loss'], label: 'Training Loss')
plt.plot(history.history['val_loss'], label = 'Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

OUTPUT:

```
20/20 ———— 0s loss: 0.0137 - val_loss 0.0166
Epoch 86/100
20/20 ———— 0s loss: 0.0128 val_loss - 0.0166
Epoch 87/100
20/20 ———— 0s loss - 0.0130 val_loss : 0.0191
Epoch 88/100
20/20 ———— 0s loss: 0.0147 val_loss 0.0197
Epoch 89/100
20/20 ———— 0s loss: 0.0135 val_loss: 0.0168
Epoch 90/100
20/20 ———— 0s loss: 0.0135 val_loss: 0.0168
Epoch 91/100
20/20 ———— 0s loss 0.0140 val_loss 0.0198
Epoch 92/100
20/20 ———— 0s loss : 0.0146 val_loss: 0.0198
Epoch 93/100
20/20 ———— 0s loss: 0.0149 val_loss 0.0152
Epoch 94/100
20/20 ———— 0s loss: 0.0140 val_loss: 0.0154
Epoch 96/100
20/20 ———— 0s loss : 0.0159 val_loss: 0.0198
Epoch 98/100
20/20 ———— 0s loss : 0.0123 - val_loss: 0.0170
Epoch 99/100
20/20 ———— 0s loss: 0.0135 val_loss .0187
Epoch 100/100
20/20 ———— 0s loss: 0.0126 val_loss: 0.0180
7/7 ———— 0s
```



OUTPUT:

Thus ANN for an application in regression using python is executed successfully

DECISION TREE

EXP: 6
4/10/24

AIM:
To implement a decision tree classification technique for gender classification using python

CODE:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
        classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn import tree

data = {
'Height': [150, 160, 170, 180, 155, 165, 175, 185, 140, 145],
'Weight': [50, 60, 70, 80, 55, 65, 75, 85, 45, 50],
'Age': [25, 30, 35, 40, 28, 32, 37, 42, 24, 29],
'Gender': ['Female', 'Female', 'Male', 'Male', 'Female', 'Female',
           'Male', 'Male', 'Female', 'Female']}

Pf = pd.DataFrame(data)
df['Gender'] = df['Gender'].map({'Female': 0, 'Male': 1})

X = df[['Height', 'Weight', 'Age']]
y = df['Gender']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                   random_state=42, stratify=y)

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred, zero_division=0)

print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
plt.figure(figsize=(12,8))
tree.plot_tree(clf, feature_names=X.columns,
               class_names=['Female', 'Male'], filled=True)
plt.title("Decision Tree for Gender classification")
plt.show()
```

OUTPUT:

Enter height in cm for prediction: 169
Enter weight in cm for prediction: 61
Predicted gender for height 169.0 cm
and weight 61.0 kg : Female

RESULT:

Thus decision tree is executed successfully

K-MEANS

EXP 7
4/10/24

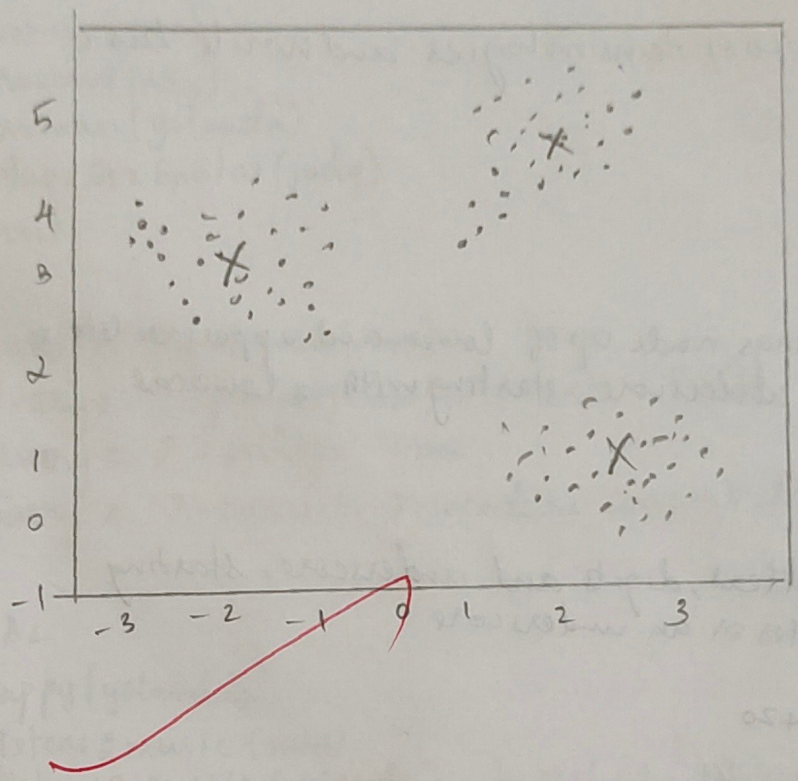
AIM:

To implement a k-means clustering technique using python language

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import kMeans
from sklearn.datasets import make_blobs

X, y_true = make_blobs(n_samples=300, centers=3, cluster_std=0.60,
                       random_state=0)

K=3
kmeans= kMeans(n_clusters= K , random_state=0)
y_kmeans= kmeans.fit_predict(X)

plt.figure (figsize =(8,6))
plt.scatter( X [:,0], X [:,1] c =y_kmeans, s=30, cmap='viridis'
                                label = 'Clusters')

Centers = kmeans.cluster_centers_
plt.scatter (centers[:,0] centers[:,1], c='red', s=200,
             alpha =0.75, marker ='x';
label = 'Centroids')
plt.title ('K-means Clustering Results')
plt.xlabel ('Feature 1')
plt.ylabel ('Feature 2')
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

Thus K means clustering technique

EXP: 8 INTRODUCTION TO PROLOG

18/10/24

AIM:

To learn PROLOG terminologies and write basic programs

TERMINOLOGIES

1. Atomic Terms
   Usually strings made up of lower-and uppercase letters, digits and the underscore starting with a lowercase
   Eg: dog
   ab_c-521

2. Variables
   Strings of letters, digits and underscore, starting with capital letter or an underscore
   Eg: Dog
   Apple-420

3 Compound Terms
   Made up of a PROLOG atom and a number of arguments enclosed in parantheses and separated by comas
   Eg: is_bigger(elephant,X)
   f(g(X,_),7)

4. Facts
   predicate followed by a dot.
   Eg: bigger_animal(whale)
   life_is_beautiful

5. Rules
   consist of head and a body
   Eg: is_smaller(X,Y):-is_bigger(Y,X)
   aunt(Aunt,Child):-sister(Aunt,Parent),parent(Parent,Child).

SOURCE CODE

KB1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

Query 1 : ?- woman(mia). True
Query 2 : ?- playsAirGuitar(mia). False
Query 3 : ?- party. True
Query 4 : ?- concert. procedure concert doesnt exist

KB2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda) :- happy(yolanda).
playsAirGuitar(mia) :- listens2music(mia).
playsAirGuitar(yolanda) :- listens2music(yolanda).
```

Query 1 : ?- playsAirGuitar(mia) true
Query 2 : ? playsAirGuitar(yolanda) true

KB3

```
likes(dan, sally).
likes(sally, dan).
likes(john, britney).
married(X, Y) :- likes(X, Y), likes(Y, X).
friends(X, Y) :- likes(X, Y); likes(Y, X).
```

Query 1 : ? likes(dan, X).
X = sally
Query 2 : ? married(dans, sally). false
Query 3 : ? married(john, britney). false

KB4

```
food(burger).
food(sandwich).
food(pizza).
lunch(sandwich).
dinner(pizza).
meal(X) :- food(X).
```

Query 1 : ? meal(X), lunch(X)
X = sandwich
Query 2 : ? food(pizza) true
Query 3 : ? dinner(sandwich) false

KB5

```
owns(jack, car(bmw)).
owns(john, car(chevy)).
owns(olivia, car(civic)).
owns(jane, car(chevy)).
sedan(car(bmw)).
sedan(car(civic)).
truck(car(chevy)).
```

Query 1 : ? owns(john, X)
X = car(chevy)
Query 2 : ? owns(john, _)
true
Query 3 : ? owns(who, car(chevy))
who = john
Query 4 : ? owns(jane, X), sedan(X)) false

RESULT :

Thus the prolog programs are executed successfully

EXP: 9 MINMAX ALGORITHM

18/10/24

AIM:

To implement minmax algorithm

CODE:

```
import math
def minmax(depth, node_index, ismaximizes,
           scores, height):
    if depth == height:
        return scores[node_index]
    if is_maximizes:
        return max(minmax(depth+1, nodeindex*2,
                          false, scores, height),
                   minmax(depth+1, nodeindex*2+1,
                          false, scores, height))
    else:
        return min(minmax(depth+1, node_indx*2,
                          True, scores, height),
                   minmax(depth+1, node*2+1,
                          true, scores, height))
def calculate_tree_height(num_leaves):
    return math.ceil(math.log2(num_leaves))
scores = [3, 5, 6, 9, 1, 2, 0, -1]
tree_height = calculate_tree_height(len(scores))
optimal_score = minimax(0,0, True, scores, treeheight)
print(f"the optimal score is: {optimal_score}")
```

OUTPUT:

The optimal score is : 5

RESULT:

Thus minmax is algorithm is executed successfully

EXP 10 PROLOG FAMILY TREE

25/10/24

AIM:

To develop a family tree program using PROLOG with all possible facts, rules and queries

SOURCE CODE:

KNOWLEDGE BASE:

Facts

```
male(peter)
male(john)
male(chris)
male(kevin)

female(betty)
female(jeny)
female(lisa)
female(helen)

parentof(chris, peter)
parentof(chris, betty)
parentof(helen, peter)
parentof(helen, betty)

parentof(kevin, chris)
parentof(kevin, lisa)
parentof(jeny, john)
parentof(jeny, helen)
```

Rules

son, parent
son, grand parent

father(x,y) :- male(y), parent of(x,y)

output
X = chris
Y = peer
mother(x,y) - female(y), Parent of(x,y)
output
x = chris
y = betty
grandfather(x,y) :- male(y) parent of(x,z) parent of(z,y)
output
X = kevin
Y = petel
Z = chris
grand mother(x,y) :- female(y) parent of(x,z)(parent of(z,y)
Output
X = kevin
Y = betty
Z = chris
brother(x,y) : male(y) father(x,z) father(y,w), z == w
output
procedure father(A,B) doesnot exist
sister(x,y) : female(y) father(x,z) father(y,N) z == w
output
procedure father of(A,B) doesnot exist

RESULT:
Thus the program was executed successfully