INDEX

NAME: Dhanasree . L.P. STD.: CSE SEC.: A ROLL NO.: 220701061 SUB.: POAI observation

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	09/8/24	8 Queens Problem	9 10	
2.	16/8/24	DFS	10	
3	23/8/24	DFS-Water JUG Problem	10	
4)	30/8/24	A* Search Algorithm	10	
5.	06-9/24	Implementation Of Decision tree Classification Techniques	10	
6.	27/9/24	Implementation Of ANN ~~for~~ using python regression	10	
7)	4/10/24	Implementation Of clustering techniques	10	
8)	18/10/24	Minimax Algorithm	10	
9)	25/10/24	Introduction to PROLOG	10	
10	8/11/24	PROLOG FAMILY Tree	10	

Completed

Python Programs

1) Program to count the number of occurances of an element in a list:

```
def countX(lst, x):
    count = 0
    for( ele in list :
        if (ele == x)
            count = count + 1.
        return count.

lst = [8, 6, 8, 10, 8, 20, 10, 8, 8]
x = 10
print('{} has occurred {} times'.format
    (x, countX(lst, x))).
```

output:-

10 has occured 2 times

2. Multiply two lists.

```
def multiplylist(mylist):
    result = 1
    for x in mylist :
        result = result * x
    return result

list1 = [1, 2, 3]
list2 = [3, 2, 4]
print(multiplylist(list1))
print(multiplylist(list2))
```

output:- 6
24

3) Deleting an item from the list.

```
lst = ['Iris', 'Orchids', 'Rose', 'Lavenda',
'Lily', 'Carnations']
print("Original list is:", lst)
lst.remove('Orchids')
print("After deleting the item:", lst)
```

output:-

Original list : ['Iris', 'Orchids', 'Rose', 'Lavendar', 'Lily', 'Carnations']

After deleting : ['Iris', 'Rose', 'Lavendar', 'Lily', 'Carnations']

4. Concatenation of String :

```
test_list3 = [1, 4, 5, 6, 5]
test_list4 = [3, 5, 7, 2, 5]
test_list3 = test_list3 + testlist4
print("Concatenated list using + :"
                    + str(test_list3)).
```

output:.

concatenated list using + : [1, 4, 5, 6, 5, 3, 5, 7, 2

5) Harshad Number.

```
def checkHarshad(n):
    sum = 0
    temp = n
    while temp > 0:
        sum = sum + temp % 10
        temp = temp // 10
    return n % sum == 0
```

Output

Yes

No

```
if (checkHarshad(12)):
    print ("Yes")
else:
    print ("No")

if (checkHarshad(15)):
    print("Yes")
else:
    print ("No")
```

output :- Yes
No.

6. Odd Frequency characters.

```
test_str = 'HelloWorld, Hi'
print("The original string is : "+ str(test_str))
x = set(test_str)
res = []
for i in x :
    if (test_str.count(i) % 2 != 0):
        res.append(i)
print("The odd Frequency characters
    are :"+ str(res).
```

output:-
The original string is: HelloWorld, Hi
The odd Frequency characters are :
['W', 'r', 'l', 'i', ',', 'e', 'd'].

AI - driven Fashion designer.

Domain :

AI- driven virtual fashion designer. Concepts falls within the AI and fashion technology domain. This domain integrates Both AI and ml techniques to provide suggestion for users, help the fashion designers to have a suggestion for launching the new styles and trend analysis.

Problem statement :-

The fashion industry is a dynamic and highly competitive fields where designers constantly seek innovative ways to create unique and market relevant fashion pieces Traditional fashion design processes involve extensive manual effort, creativity and travel analysis, which can be time-consuming and limited by the designer's personal style and expensive. As fashion trends evolve rapidly, staying ahead requires not only creative infiction but also the ability to analyze and respond to market changes efficiently.

To address these challenges, there is a need for an advanced solution that integrates artificial intelligence into the fashion design process. The AI-driven virtual fashion designer.

aims to leverage AI technologies to streamline and enhance the design process enabling the generation of unique fashion design based on user preferences and current trends.

Objectives:-

- Automate fashion design
- Analyse fashion trends
- Personalize design recommendations
- Streamline design process.

Users:-

- Fashion designers
- Fashion Brands and Retailers
- Fashion Enthusiasts and Consumers
- Trend Analysts and Consultants.
- E-commerce Platforms

Solution:-

N-QUEEN'S PROBLEM.

Ex no : 2

AIM :

To write the python code for the N-Queen's problem.

CODE :

```
def is_safe (board, row, col, n) :
    for i in range(col) :
        if board [row] [i] = = 1 :
            return False.
    for i, j in zip( range (row, n), range (col, -1, -1)):
        if board [i][j] == 1 :
            return False
    return True.

def solve_n_queens_util(board , col, n) :
    if ( col >= n) :
        return True
    for i in range (n) :
        if is_safe (board ,i, col ,n) :
            board [i][col] = 1.
            if solve_n_queens_util (board,col+1,n):
                return True
            board[i][col] = 0
    return False.

def print_board (board,n) :
    print ("In solution:")
    for row in board :
        for cell in row :
```

Row is fixed column is changing

Basecase

check row is safe for all column

```
        if cell == 1 :
            print ('Q',end=' ')
        else :
            print('.',end=' ')
    print()

def solve_n_queens(n):
    board[[0 for _ in range(n)] for _ in range(n)]
    if not solve_n_queens_util(board, 0, n):
        print("No solution exists")
        return False
    print_board(board,n)
    return True

# Driver code:-
    n = int(input("Enter the value of N:"))
    solve_n_queens(n):
```

OUTPUT:

ENTER the value of N: 6

SOLUTION:

```
. . . Q . .
Q . . . . .
. . . . Q .
. Q . . . .
. . . . . Q
. . Q . . .
```

Enter value of N: 4

```
. . Q .
Q . . .
. . . Q
. Q . .
```

Thus the code for n-Queen's problem is done successfully.

DFS ALGORITHM :

Ex no: 3

AIM: -

To write the python code for dfs algorithm

```
Class Node:
    def __init__(self, value):
        self.value = value
        self.neighbors = []
    def add_neighbor(self, neighbor):
        self.neighbors.append(neighbor)
Class Graph:
    def __init__(self):
        self.nodes = {}
    def add_node(self, value):
        if value not in self.nodes:
            self.nodes[value] = Node(value)
    def add_edge(self, from_value, to_value):
        if from_value in self.nodes and
    to_value in self.nodes:
            self.nodes[from_value].add_neighbor
    (self.nodes[to_value])
            self.nodes[to_value].add_neighbor
    (self.nodes[from_value])
```

```
def dfs (self, start_value):
    visited = set()
    stack = [self.nodes.get(start_value)]
    while stack:
        node = stack.pop()
        if node and node.value not in visited:
            print(node.value)
            visited.add(node.value)

            for neighbor in reversed(node.neighbors):
                if neighbor.value not in visited:
                    stack.append(neighbor).

# Example (input)
graph = Graph()
graph.add_node('1')
graph.add_node('2')
graph.add_node('3')
graph.add_node('4')
graph.add_node('5')
graph.add_node('6')

graph.add_edge('1', '2')
graph.add_edge('1', '3')
graph.add_edge('2', '4')
graph.add_edge('2', '5')
graph.add_edge('3', '6')
graph.add_edge('5', '6')

print("DFS:")
graph.dfs('1'
```

Output:

DFS :

1

2

4

5

6

3

Result:

Thus the program for DFS is executed successfully.

A* Algorithm

Ex no: 4

Date: 13/9/24

Aim:

To write a python code for A* algorithm

CODE:

```
def astar (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:
            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)

    if n == 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
    open_set.remove(n)
    closed_set.add(n)
print("Path does not exist!")
return None

def get_neighbors(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)],
}
astar('A', 'G'
```

output

Path found : ['A', 'B', 'G']

Result:

Thus the program for A* is done successfully.

WATER JUG PROBLEM USING DFS

Ex no: 5

Date: 13/9/24

Aim:

To write a python code for water jug problem using DFS

Code:

```
def water_jug_dfs(jug1, jug2, target):
    def dfs(j1, j2, seq, visited):
        if j1==target or j2==target:
            return seq
        visited.add((j1,j2))
        actions=[
("fill",1),("fill",2),("empty",1),("empty",2),
("pour",1,2), ("pour",2,1)
        for action in actions:
            if action[0]=="fill":
                if action[1]==1:
                    next_state=(jug1, j2)
            else:
                next_state=(j1, jug2)
            elif action[0]="empty":
                if action[1]==1:
                    next_state=(0, j2)
            else:
                next_state=(j1, 0)
```

```
else :
    if action[1] == 1:
        amount = min(j1, jug2 - j2)
        next_state = (j1 - amount, j2 + amount)
    else :
        amount = min(j2, jug2 - j2)
        next_state = (j1 + amount, j2 - amount)
if next_state not in visited :
        next_seq = seq + [action]
        result = dfs (next_state[0], next_state[1],
    next_seq , visited)
        if result:
            return result
return None
visited = set()
return dfs ( 0, 0, [ ], visited)

#Maincode
result = water_jug_dfs(4,3,2)
print (result)
```

Output:

[('fill', 1), ('fill', 2), ('empty', 1), ('pour', 2, 1), ('fill', 2), ('pour', 2, 1)]

Result:

Thus the program for water jug problem using DFS is executed successfully

Ex no.

DATE :

IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNIQUES.

Aim:

To implement a decision tree classification technique for gender classification techniques using python.

Explanation:

- Import tree from sklearn
- Call the function DecisionTreeClassifier() from tree.
- Assign values for x and y.
- Call the function predict for predicting on basis of given random values for each given feature
- Display the output.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
data={
    'Height': [152,155,172,185,167,180,157,180,
               164,177],
    'Weight': [45,57,72,85,68,78,22,90,66,
               88],
    'Gender': ['Female','Female','Male',
    'Male','Female','Male','Female'
```

```
'Male', 'Female', Male']
}
df = pd.DataFrame(data)
X = df[['Height', Weight']
Y = df['Gender']

classifier = DecisionTreeClassifier()
classifier.fit(X,Y)
height = float(input("Enter height (in cm)
        for prediction:"))
weight = float(input("Enter weight (in kg)
for prediction:"))
random_values = pd.DataFrame([[height,
weight]], columns = ['Height', 'Weight'])
predicted_gender = classifier.predict(random_values)
print(f "Predicted gender for height
{height}cm and weight {weight}kg:
{predicted_gender[0]}")
```

Output:

Enter the height in cm: 154
Enter the weight in kg : 39.
Predicted-gender for height 154.0 cm
and weight 39.0 kg is Female.

Result:-

Thus the python code for decision tree classification for gender classification is executed.

Ex no:

Date:

IMPLEMENTATION OF ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION USING PYTHON - REGRESSION.

Aim:

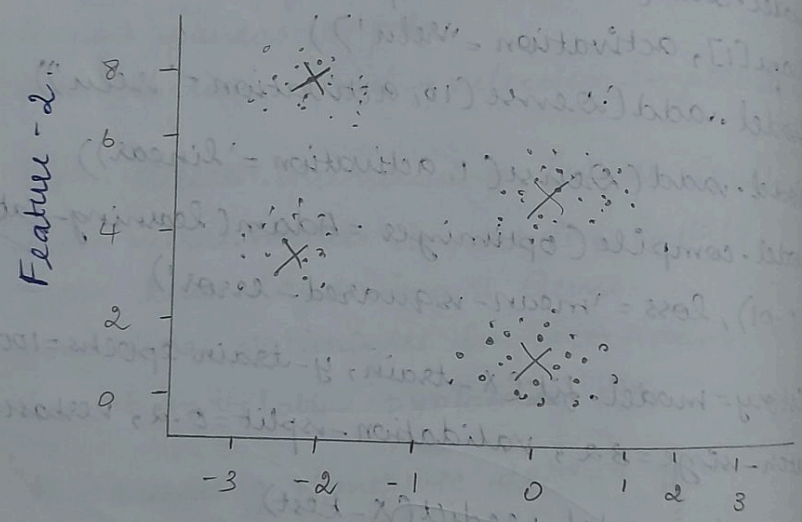
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt

# Generating synthetic dataset for demonstration
# Replace this with your own dataset
np.random.seed(42)
X = np.random.rand(1000, 3) # 100 samples, 3 features
y = 3*X[:,0] + 2*X[:,1]**2 + 1.5*np.sin(X[:,2]*np.pi) + np.random.normal(0,0.1,1000) # Non linear relationship
```

```
# Split the dataset 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)
# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Building the ANN model
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()
```

	Feature 1	Feature
0	0.836857	2.136359
1	-1.413658	7.409623
2	1.155213	7.099619
3	-1.018616	7.814915
4	1.271351	1.892542



Inertia : 212.00599621083475

Silhouttee score : 0.6819938690643478

Thus the implementation of ANN for an application using python regression is done successfully

Aim :- To implement the k-means Clustering using python language.

```
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()
```

	Feature	Target
0	3.745401	7.846204
1	9.507143	16.343597
2	7.319939	15.400275
3	5.986585	13.194341
4	1.560186	4.239954

```
Epoch 1/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step -
Loss: 143.6730
:
Epoch 100/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - Loss:
4.2248
```

Result:

Thus the implementation of K means clustering is done successfully.

Ex no:

Date:

INTRODUCTION TO PROLOG

Aim:

To learn Prolog technologies and write basic programs.

Terminologies:-

1) Atomic terms:

Atomic terms are usually strings made up of lower- and uppercase letters, digits and the underscore starting with lowercase letter.

Eg:- dog

ab_c-321.

2) Variables:

variables are string of letters, digits and the underscore, starting with a capital letter or an underscore.

Ex:- Dog

Apple-420.

3) Compound terms:

Compound terms are made up of PROLOG atom and a number of arguments. (prolog terms. i.e atoms, variables) enclosed in parantheses and seperated by commas

Eg:- is_bigger(elephant,X.)

4) Facts:

A fact is a predicate followed by a dot.

Eg:- life_is_beautiful.

5) Rules:-

A rule consists of head (a predicate)

and a body(sequence of predicates followed by commas).

Eg:- is_smaller(X, Y):- is_bigger(Y, X).

Source code:-

KB1:-

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

Output:-

```
woman(mia).
True.
woman(dhana).
False.
```

KB2

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

output

```
happy(yolanda).
True
listens2music(dhana)
False.
```

KB3

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

output

```
likes(john, brittney).
    True
likes(sally, john).
    false.
```

KB4

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

Output:

```
food(sandwich).
    True.
dinner(burger).
    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)).
```

Output :-

```
sedan(car(civic)).  -> True
truck(car(civic))   -> False.
```

Result:-

Thus the prolog terminologies and write basic programs is done successfully.

PROLOG - FAMILY-TREE

Ex. no: 9

Date:

Aim:

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

Knowledge Base:

```
/* facts */
male (peter)
male (john)
male (chris)
male (kelvin)
female (betty)
female (jeny)
female (lisa)
female (helen)
parentOf (chris, peter).
parentOf (chris, betty).
parentOf (helen, peter)
parentOf (helen, betty).
parentOf (kelvin, chris)
parentOf (kelvin, lisa).
parentOf (jeny, john).
parentOf (jeny, Helen).
/* Rules */.
```

a) father(x, y) :- male(x), parentOf(x, y).

Y	X
Peter	Chris
Peter	Helen
John	Jeny
Chris	Kevin.

b) female(Y), parentOf(X, Y).

Y	X
betty.	Chris
betty	Helen.
Lisa	Kelvin
Helen	Jeny

c) Male(Y), parentOf(X, Z), parentOf(Z, Y).

Y	X	Z.
Peter	Kelvin	Chris
peter.	jeny	Helen.

d) female(Y), parentOf(X, Z), parentOf(Z, Y)

Y	X	Z
betty	kelvin	chris
betty	Jeny	Helen.

e) Male(Y), father(X, Z), father(Y, W) Z==W

procedure father(A, B) does not exist.

f) female (Y), father (X, Z), father (Y, W) Z == W.
procedure father (A, B)' does not exist.

Result :-
Thus the prolog family tree is executed successfully.

MINIMAX ALGORITHM.

Ex no: 10

Date: 18-10-24

Aim:

- A simple example can be used to explain how the minimax algorithm works we have included in the example.

- There are 2 players in this scenario one named Maximizes and other named Minimizes.

Maximizes will strive for highest possible score and minimizes will strive for lowest score

Program:

```
import Math
def minimax (depth, node-index,
is-maximizer, scores, height)
    if depth == height
        return scores [node_index]
    if is-maximizer:
        return max (minimax (depth+1,
nodeindex * 2, False, scores, height),
minimax (depth+1, node-index * 2+1, False-
score, height))
    else :
        return min (minimax (depth+1,
node_index * 2, False, score, height),
minimax (depth+1, node-index * 2+1,
False, score, height)).
```

```
def calculate_tree_height(num_leaves):
    return math.ceil(math.log2(num_leaves))
scores = list(map(int(input("Enter the
score seperated by spaces:").split()))
tree_height = calculate_tree_height(0,0,
True, scores, tree_height).
optimal_score = minimax(0, 0, True, scores, tree_
height).
print("The optimal score is : {optimal score}")
```

Output :-

Enter the scores seperated by spaces.

-1 4 2 6 -3 -5 0 7.

The optimal score is 4

Result.

Thus the minimax algorithm successfully determinas the optimal move for players