INDEX

NAME : Ashwanth . E STD. : CSE SEC. : A ROLL NO. : 2203010312 SUB. :

S. No.	Date	Title	~~Page No.~~ marks	Teacher's Sign/Remarks
1.	31/7/24	Sample python programs	9	
2.	7/8/24	Project report	9	
3.	4/9/24	A- queen problem	10	
4.	4/9/24	Depth first search	10	
5.	11/9/24	A* algorithm	10	
6.	18/9/24	Ao* algorithm	10	
7.	25/9/24	Decision Tree	10	
8.	9/10/24	K-means	10	
9.	16/10/24	Artificial neural network	10	
10.	23/10/24	Minimax	10	
11.	30/10/24	Introduction to prolog	10	
12.	6/11/24	Prolog - family tree	10	

completed

1). Find factorial of a number

Program:

```
def factorial (n)
    return 1 if (n==1 or n==0) else n * factorial (n-1)
[] num = 5
print ("factorial of" num, "is", factorial (num))
```

Output:

Factorial of 5 is 120

2) TO find area of triangle

Program

```
a = float (input ('enter first side:'))
b = float (input ('enter second side:'))
c = float (input ('enter third side:'))
s = (a+b+c)/2
area = (s * (s-a) * (s-b) * (s-c)) ** 0.5
print ('the area of triangle is %0.2f' %area)
```

O/P:

```
enter first side : 3
Enter second side : 2
Enter third side : 4
the area of triangle is 2.90
```

3). To Generate

Program
```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a+b

count = 10
print("fibanocci sequence");
fibanocci(count)
```

O/P fibanocci sequence 0 1 1 2 3 5 8 13 21 34

4) Display calendar

Program
```
import calendar
yy = 2014
mm = 11
```

O/P

November 2024

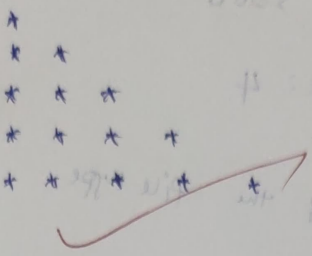
M	T	W	T	F	S	S
				1	2	
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

5). Printing patterns

Program
```
[ ] rows = int(input("Enter no. of rows"))
for i in range(rows):
    for j in range(i+1):
```

9p

enter no. of rows : 5



6). matrix addition

Program

```
x = [[12,7,3], [4,5,6], [7,8,9]]

y = [[5,8,1], [6,7,3], [4,5,9]]

result = [[0,0,0], [0,0,0], [0,0,0]]

for i in range (len(x)):
    for j in range (len(x[0])):
        result[i][j] = x[i][j] + y[i][j]
for r in result
    print(r).
```

O/P

```
[17, 15, 4]
[10, 12, 9]
[11, 13, 18]
```

7). To find no. of digits

```
a = int(input("enter the number"))
count = 0
i = a
while (i>0):
    count = count + 1
    i = i//10
print("count")
```

print (delta . days)

6/8

31/01/2024

Ex - 2
7/8/24

TRAIN TICKET BOOKING SYSTEM USING MENU-BASED CHATBOT

DOMAIN

1. Information technology
2. Artificial intelligence

OBJECTIVES

1. Automate online ticket booking

- To conquer the crowd in the counter to catch the train on time

2. Real-time updates

- real-time information of available train at platform

3. multilingual Support

- Avoid miscommunication in ticket counter

4. Simplify Booking

- Easy UI with like human chatting interface and facilate easy booking.

5. Payment

- various payment options to pay fast and secure

6. Personalization

- Maintain a report and track

Problem Statement

→ Current online ticket booking often involve in long queues at ticket counters, while some systems offer online ticket booking many users face difficult in navigation of web interface. Traditional system may require manual inputs, and lack immediate assistance There is a need for more intutive & user-friendly solution that simplifies booking process, provides real-time information.

- challenge for accurate valuation
- Feature diversity
- Data availability & quality
- user needs
- Integration & usability

Algorithm

Initialization

→ starts with a welcome message

Main Menu

→ Select book a ticket

→ Ask for travel details
  - departure station
  - Arrival station
  - travel date & time

→ validate input

→ check availability

Error Handling

→ Invalid input - re-enter correct information

→ System errors

Integration with Backend Service

→ Train schedule & availability

→ payment gateway

→ Booking system - interface with Booking system

Testing & Refinement

→ Test scenarios - Run various test cases to ensure handling

→ Refine Responses - Improve user feedback responses & test it

Deployment

→ Launch chatbot in google, telegram or web

→ Monitor - make necessary adjustments & monitor performance

People who benefits

→ Peoples who travel

→ Business organization

→ train operators & Service provider

→ government & regulatory bodies

Ex.no: 3 n- Queen problem

4/9/24

code:

```
def printsolution(board):
    for i in range(len(board)):
        for j in range(len(board)):
            print(board[i][j], end=' ')
        print()

def isSafe(board, row, col):
    for i in range(col):
        if board[row][i] == 1:
            return false

    for i,j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return false

    for i,j in zip(range(row, len(board), 1), range(col, -1, -1)):
        if board[i][j] == 1:
            return false.

    return True

def solveNQutil(board, col):
    if col >= len(board):
        return true.

    for i in range(len(board)):
        if isSafe(board, i, col):
            board[i][col] = 1

            if solveNQutil(board, col+1):
                return true

            board[i][col] = 0
```

```
    return False

def solveNQ():
    N = int(input("Enter the size of board(N):"))
    board = [[0 for _ in range(N)] for _ in range(N)]
    if not solveNQUtil(board, 0):
        print("Solution doesn't exist")
        return False
    printSolution(board)
    return True

solveNQ()
```

Output

```
>> Enter size of board (N)
   6

>> 0 0 0 1 0 0
   1 0 0 0 0 0
   0 0 0 0 1 0
   0 1 0 0 0 0
   0 0 0 0 0 1
   0 0 1 0 0 0

   True
```

Ex: no: 4
4/9/24

Depth first search algorithm

Aim:

To write a python program for DFS problem

PROGRAM

```
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, neighbour, visited)

num nodes = int (input ("Enter no. of nodes: "))
graph = {}
for i in range (num_nodes):
    node = input ("Enter node " + str (i+1) + ": ").
        strip()
    neighbours = input ("Enter neighbour of " + node +
        " (comma - seperate): ").strip().
        split (',')
    graph [node] = neighbours
print ("graph:", graph)
start_node = input ("Enter the starting node:")
        .strip()
print ("starting node :", start_node)
dfs (graph, start_node)
```

OUTPUT

enter the no. of nodes : 5

enter node 1 : A

enter neighbour of A (comma - seperated) : B, C

Enter node 2 : B

Enter neighbour of B (comma - seperated) : D

Enter node 3 : C

Enter neighbour of C (comma - seperated) : D, E

enter node 4 : D

enter neighbour of D (comma - seperated) :

Enter node 5 : E

enter neighbour of E (comma - seperated) :

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

enter starting node : A

Starting node : A

A B D C E

RESULT :

Thus the python program for n queen's problem has been verified and executed successfully.

ex: 5
11/9/24 A* algorithm

AIM:
to write a python program for A* algo

PROGRAM:

```
import heap
class node
    def __init__(self, name, parent=None, g=0, h=0):
        self.name = name
        self.parent = parent
        self.g = g
        self.f = g+h
    def __lt__(self, other):
        return self.f < other.f
def a_star(graph, start, goal, h_func):
    open_list = []
    heapq.heappush(open_list, Node(start, None,
                                   (start, goal)))
    closed_list = set()
    while open_list:
        current_node = heapq.heappop(open_list)
        if (current_node.name == goal:
            path = []
            while current_node:
                path.append(current_node.name)
                current_node = current_node.parent
            return path[::-1]
        closed_list.add(current_node.name)
```

```
        if neighbour in closed_list:
            continue
        g_new = current_node.get_cost()
        h_new = h_func(neighbour, goal)
        f_new = g_new + h_new
        neighbour_node = Node(neighbour, current_node,
                              g_new, h_new)
        heapq.heappush(open_list, neighbour_node)
    return None

graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('A', 1), ('D', 1), ('E', 3)],
    'C': [('A', 3), ('F', 2)],
    'D': [('B', 1)],
    'E': [('B', 3), ('F', 1)],
    'F': [('C', 2), ('E', 1)]
}

def heuristic(node, goal):
    return 0

start_node = 'A'
goal_node = 'F'
path = a_star(graph, start_node, goal_node, heuristic)
if path:
    print(f"Path found: {path}")
else:
    print("No path found")
```

OUTPUT

Path Found : ['A', 'C', 'F']

RESULT

Thus python program for A* algorithm has been verified & executed successfully

Ex.No:10

AIM

To write a python program for solving water jug problem using DFS

PROGRAM:

```
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 fill_4_gallon(x, y, x_max, y_max):
    return (0, y)
def fill_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):
    return (x + transfer, y - transfer)
def dfs_water_jug(x_max, y_max, goal_x,
                  visited = None, start = (0,0)):
    if visited is None:
        visited = set()
    stack = [start]
    while stack:
        state = stack.pop()
        x, y = state
        if state in visited
```

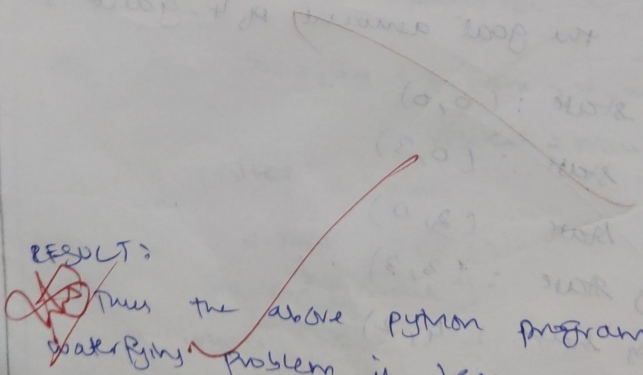
```
        visited.add(state)
        print(f"visiting state: {state}")
        if x == goal_x:
            print(f"goal reached: {state}")
            return state
    return None

x_max = int(input("Enter max capacity of 4:"))
y_max = int(input("enter max capacity 3:"))
goal_x = int(input("enter the goal amount
                    for 4-gallon jug:"))
result = dfs_water_jug(x_max, y_max, goal_x)
if result is None:
    print("No solution found:")
```

OUTPUT:

```
enter max capacity of 4: 4
enter the max capacity of 3: 3
Enter the goal amount of 4-gallon jug: 2
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)
visiting state : (2, 3)



RESULT:

Thus the above python program to perform Water Pouring Problem is been executed and verified successfully.

Aim:

To implement AO* algorithm

PROGRAM:

```
class graph
def __init__(self, graph, heuristic)
    self.graph = graph
    self.heuristic = heuristic
    self.solution = {}

def ao_star(self, node)
    print("f* expanding : {node}")
if node not in self.graph or not self.graph(node):
return
    children = self.graph[node]
    best_path = None
    min_cost = float("inf")
for group in children
    cost = sum(self.heuristic[child] for child in
                group)
    if cost < min_cost
        min_cost = cost
        best_path = group
    self.solution[node] = best_path
print(f"Best path for {node} = {best_path}
        with cost {min_cost}")
for child in best_path
    self.ao_star(child)
def get_solution(self):
    return self.solution
graph = {
```

'A' : [['B', 'C'], ['D']]

'B' = [['E']]

'C' = [['G']]

'D' = [['H']]

'E' = []

'G' = []

'H' = []

}

heuristic = {

'A' = 0, 'B' = 1, 'C' = 2, 'D' = 4, 'E' = 1,

'G' = 3, 'H' = 5 }

graph_obj = graph(graph, heuristic)

graph_obj = ao_star('A')

solution = graph_obj.get_solution()

print("solution :", solution)

o/p:

expanding : A

best path jos A = ['B', 'C'] with cost }

expanding : B

Best path for B = ['E'] with cost)

expanding : E

Expanding : C

Best pat C = ['G']

expanding : G

solution of A = {'A', ['B', 'C'], 'B' = ['E'], 'C' = ['G']}

RESULT:

Thus AO* algorithm implemented successfully.

Exp: NO. 7 Implementation of decision tree classification
25/9/24

Aim:

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

EXPLANATION.

* Input the tree
* Call the function from tree
* Assign value for x & y
* Call the function on basis of given random values for each given feature.
* Display the o/p

```
import pandas as pd
from sklearn import tree
data = {'height': [152, 155, 172, 185, 167, 180, 164, 177],
        'weight': [45, 54, 72, 85, 68, 78, 22, 88],
        'gender': ['female', 'female', 'male', 'male', 'female', 'male', 'female', 'male']}
df = pd.DataFrame(data)
x = df[['height', 'weight']]
y = df['gender']
classifier = DecisionTreeClassifier()
classifier = 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'])

print(f"predicted gender for height {height}
      and weight {weight} kg :
      {predictable_gender[0]}")
```

o/p:

predicted gender : female

RESULT:

Thus the python to implement a decision tree classification technique for gender classification is executed successfully.

Ex.no:8 Implementation of ANN for an application using python regression

9/10/24

AIM:

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

PROGRAM CODE:

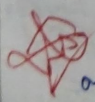
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import standard_scale
from keras.optimizers import adam

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)
x_train = scalar.fit_transform(x_train)
x_test = scalar.transform(x_test)
model = sequential()
model.add(Dense(10, input_dim = x_train, shape[1],
activation = 'rel'))
model.add(Dense(1, activation = 'linear'))
y_pred = model.predict(x_test)
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 & validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

o/p

predicted: 0.06, Actual: 250000



RESULT:

Thus the python program to implement artificial neural networks for an application in regression is executed successfully.

Ex.no: 09
16/10/24

Implementation of K-means clustering technique

Aim:
To implement a K-means clustering technique using python language

Explanation

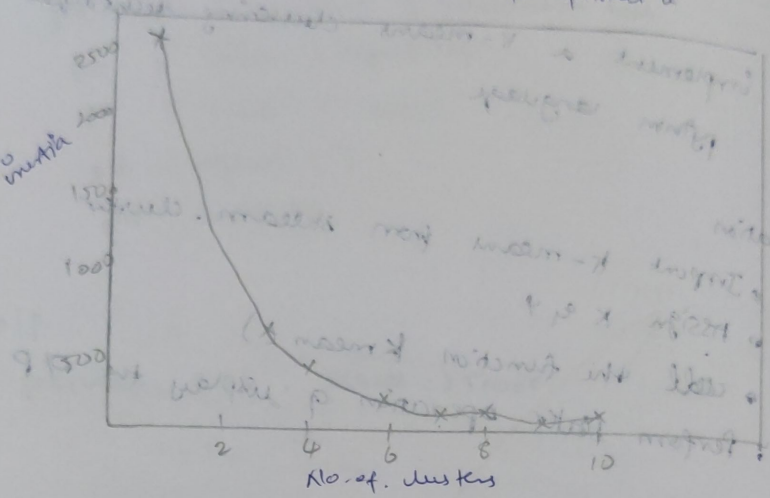
- Import K-means from sklearn.cluster
- Assign X & Y
- Call the function kmean()
- Perform scatter operation & display the O/P

Program code:

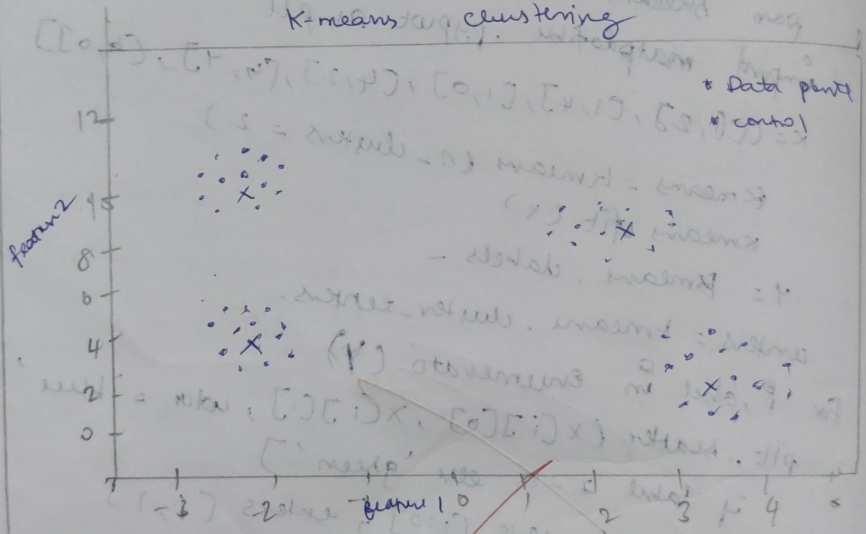
```
from sklearn.cluster import Kmeans
import matplotlib.pyplot as plt
X = [[1,2], [1,4], [1,0], [4,2], [4,4], [4,0]]
Kmeans = kmeans(n_clusters = 2)
kmeans.fit(X)
Y = kmeans.labels_
centers = kmeans.cluster_centers_
for i, label in enumerate(Y):
plt.scatter(X[i][0], X[i][1], color = 'blue'
if label == 0 else 'green')
plt.scatter(centers[:,0], centers[i,1]
color = 'red', marker = 'x', s2 = 100,
label = 'centroids')
plt.xlabel('feature1')
plt.ylabel('feature2')
plt.title('K means clustering')
plt.legend()
```

output

Elbow method for optimal k



K-means clustering



RESULT:

Thus the program to implement K-means clustering technique is executed successfully

Ex: no: 10 Minimax
23/10/24

Aim:

To implement minimax algorithm

Program:

```
import math

def minimax (depth, node_index, maximizes, scores,
             height):

    if depth == height:
        return scores[node_index]

    if is_maximizes:
        return max(minimax(depth-1, node_index*2, False,
                   scores, height), minimax(depth-1, node_
                   False, scores, height))

    else
        return min(minimax(depth-1, node_index*2, True,
                   S(res, height), minimax(depth+1, node,
                   index*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, tree_height)
print(f"The optimal score is {optimal_score}")
```

output

Output:

The optimal value is : 5

Result:

Thus the decision tree program has been executed successfully

30/10/24

Exp-no 11 Introduction to prolog

Aim:

To learn prolog terminologies & write basic program

TERMINOLOGIES:

1) Atomic terms:

They are usually strings made up of lower & uppercase letters, digits & the underscore, starting with a lowercase letter

eg: dog, ab-c, 321

2) variables

They are strings of letter, digits & the underscore, starting with a capital letters or an underscore

eg: Dog, Apple_420

3). compound terms

Compound terms are made up of a prolog atom & a no. of. arguments enclosed in parenthesis & seperated by commas

eg: is-bigger (elephant, X), f(g)(X, _)7)

4) Fact

A fact is predicate followed by a dot

eg: bigger-animal (whale). life-a-beautiful.

5) Rules

A rule contains of head & body.

eg: is_smaller (X, Y, is_bigger), aunt (aunt-child),,
sister (Aunt, parent).

Source code :

KB 1

woman (mia).

woman (jody).

woman (yolanda).

playsAirGuitar(jody)

Party

Query 1:2 - woman(mia)

Query 2: ? - plays, Airguitar(mia

Query 3: ? - party

Query 4: ? concat

o/p:

? - woman(mia)

true.

? - playsairguitar(mia)

false

? - concat.

error : unknown procedure : concat/0

KB 2

happy(yolanda).

listen2music(mia)

listen2music(yolanda) :- happy(yolanda)

playsairguitar(mia) :- listen2music(mia)

playsairguitar(yolanda) :- listen2music(yolanda)

o/p:

? plays airguitar (mia)

true

? - plays airguitar (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)

o/p

?- likes (dan, X)

X = sally

? - married (john, britney)

false

KB4

food (burger)

food (sandwich)

food (pizza)

lunch (sandwich)

dinner (pizza)

meal (X) :- food (X)

o/p :

?- food (pizza)

true

? - meal (X), lunch (X)

X = sandwich

KBS

owns (jane, car(bmw)).

owns (john, car(chevy)).

owns (olivia, car(civic)).

owns(jane, car(chevy))

sedan(car(bmw)).

sedan(car(civic)).

truck(car(chevy)).

O/P

?-owns(john, X)

X = car(chevy)

?-owns(john, _)

true

RESULT:

Thus the basic prolog programs has been executed successfully

Lab No: 12
6/11/24

Prolog Family Tree

Aim:

To develop a family tree program using prolog with all possible facts, rules & queries

Source code:

knowledge-base:

```
/* facts : : */
male (peter).
male (john)
male (chris)
male (kwin)

female (betty)
female (jeny)
female (lisa)
female (keen)
parent of (chris, peter)
parent of (chris, betty)
parent of (kwin, chris)
parent of (kwin, lisa)
parent of (jeny, john)
parent of (jeny, keen)

/* RULES : : */
/* son, parent */
/ Son, grandparent */
father (X,Y) :- male (Y), parent of (X, Y)
mother (X, Y) : - female(Y), Parent of (X, Y)
grandfather (X, Y) : - male (Y), parent of (X, Z),
                       parent of (Z, Y)
```

grandmother (X,Y) : female (Y), parent of (X,Z),
parent of (Z,Y)

brother (X,Y) : - male (Y), father (X,Z), father (Y,

sister (X,Y) : - female (Y), father (X,Z), father (Y,
Z = = W

op -

male (peter)

true

father (chris - peter)

true

father (chris, letty)

false

mother (chris, X)

X = betty

brother (chris, betty)

false.

Result:

Thus Prolog for family tree program has been executed successfully