INDEX

Name S. Arun Mohan Subject POAI

Standard Section CSE-A Roll No. 220701069

School / College

S. No.	Date	Title	Page No.	Teacher's Sign
1.	31/7/24	Sample python Codes	9	
2.	9/8/24	Water Jug Problem	9	
3.	16/8/24	Depth First search	9	
4.	30/8/24	A* Algorithm	9	
5.	6/9/24	AO* Algorithm	10	
6.	13/9/24	Implementation of clustering techniques	10	
7.	24/10/24	Implementation of regression	10	
8.	18/10/24	MiniMax algorithm	10	
9.	25/6/24	Introduction to Prolog	10	
10.	8/11/24	Prolog family Tree	10	

completed

31/7/24

SAMPLE PYTHON PROGRAMS USING GOOGLE COLAB

Program No:1

Program code:

```
numb = int (Input ("enter any number"))
facts = [ ]
for a in range (1, numb + 1):
    If num % a == 0:
        facts . append (a)
print ("Factors of {} = {}". format (numb, facts))
```

Output:

```
Enter any number 5
Factors of 5 = [1, 5]
```

Program No:2

Program code:

```
a=int(input("Enter the first number of the seies"))
b=int(input("Enter the second number of the seies"))
n=int(input("Enter the number of terms needed"))
print(a,b,end=" ")
while(n-2):
    c=a+b
    a=b
    b=c
    print(a,b,end=" ")
while(n-2)
    c=a+b
    a=b
    b=c
    print(c,end=" ")
    n=n-1
```

Output:

Enter the first number of the seies 3
Enter the second number of the seies 0
Enter the number of the needed 3
3 0 3.

Program No: 3

Program code:

```
Class rectangle():
    def __init__(self, breath, length):
        self.breadth = breadth
        self.length = length

    def area(self):
        return self.breath * self.length

a = int(input("Enter length of rectangle:"))
b = int(input("Enter breath of rectangle"))
obj = rectangle(a,b)
print("Area of rectangle:", obj.area())
print()
```

Output:-

Enter length of rectangle :5
Enter breadth of rectangle :6
Area of rectangle : 30.

Program No:4

Program code:

```
def bubble_sort(alist):
    for i in range(len(alist)-1, 0, -1):
        no_swap = True
        for j in range(0, i):
            if alist[j+1] < alist[j]:
                alist[j], alist[j+1] = alist[j+1], alist[j]
                no_swap = False
    if no_swap:
        return
alist = input('Enter the list of numbers:').split()
alist = [int(x) for x in alist]
bubble_sort(alist)
print('sorted list:', end=' ')
print(alist)
```

Output:

```
Enter the numbers: 4 3 1 2
sorted list [1 2 3 4]
```

Program: 5

Program code:

```
def quicksort(arr):
    if len(arr)<=1:
        return arr
    pivot = arr[len(arr)//2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)
alist = input('Enter the list of numbers:').split()
alist = [int(x) for x in alist]
quicksort(alist, 0, len(alist))
print('sorted list:', end=' ')
print(alist)
```

Output:

```
Enter the list nos: 5 6 1 2
sorted list [1 2 5 6]
```

Program No: 8

Program code:

```
principle = float (input ("Enter the principle amount:"))
time = int (input ("Enter the time (year):"))
rate = float (input ("Enter the rate:"))
simple_interest = (principle * time * rate)/100
print ("The simple interest is:", simple_interest)
```

Output:

```
Enter the principle amount : 200
Enter the time (years) : 5
Enter the rate : 5.0
The simple interest is : 50.0
```

Program No : 9

Program code :-

```
def count_set_bits (n):
    count = 0
    while n:
        n &= n - 1
        count += 1
    return count
n = int (input ("Enter n: "))
print ("number of set bits :", count_set_bits (n))
```

Output :

Case 1:

```
Enter n: 9
number of set bit : 2
```

Case 2:

```
Enter n: 31
Number of set bits : 5
```

Program No: 10

Program code:

```
from string import ascii_lowercase as
                   asc_lower

def check(s):
    return set(asc_lower) - set(s.lower())
                        == set([])

string = raw_input ("Enter string")

if (check(string) == True):
    print ("The string is a program")

else : print ("The string isn't a pangram")
```

Output:

31/07/2024

Enter string: Hellow world

The string isn't a pangram.

Domain : Sentimental analysis

Sentiment analysis, a key application within Natural Language Processing (NLP), is an advanced computational technique aimed at deciphering the emotional tone embedded in textual data. This domain, a subset of artificial Intelligence (AI), focuses on the automated classification of text into various sentiment categories such as positive, negative and neutral, and extends to more emotional states like joy, anger or sadness.

Problem statement:

Design and implement an automated sentiment analysis system efficiently classify feedback into sentiment categories - such as as positive, negative and netural and providing actionable insights in real-time to support improved decision-making and enhance customer experience.

Applications:-

1) Business and Companies

* To know the feedback and respone of their product

2) Social media analysis

* To analysis about their post and advertisements

3) Political analysts and compaigns

Public opinion on polices candidate and political events to shape strategies and messaging.

Algorithm

Support vector Machine (SVM)

Nqueen Problem

9/8/24

AIM:

To implement Nqueens problem

Program:

```
N = int(input("Enter the number of queen:"))
board = [[0]*N for _ in range(N)]

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

```
def solve_nqueens (board, col):
    if col >= N:
        return true
    for i in range (N)
        if_safe (board, i, col)
            board [i] [col] = 1
            if solve_nqueens (board, col, 1)
                return true
        board [i] [col] = 0
    return false
if solve_nqueens (board, 0)
    for solve_nqueens (board, 0)
        for row in board
        print (' '. join ('Q' if x == 1 else '*' for
                x in row))
else
        print ("No solution exists")
```

Output:

```
Enter the number of queens: 8
Q * * * * * * *
* * * * * * Q *
* * * * Q * * *
* * * * * * * Q
* Q * * * * * *
* * * * Q * * *
* * * * * Q * *
* * Q * * * * *
```

RESULT:

Thus the NQueens program has been executed successfully.

18/9/24

Depth First Search

AIM:

To implement Depth First Search

PROGRAM:

```
def dfs_recursive(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start)
    for neighbour in graph[start]:
        if neighbor not in visited:
            dfs_recursive(graph, neighbour,
visited)
graph={
    'A': ['B','C'],
    'B': ['A','O','E'],
    'C': ['A','F'],
    'O': ['B'],
    'E': ['B','F']
    'F': ['C','E']
print("DFS recursive:")
dfs_recursive(graph,'A')
def dfs_intuative(graph, start)
    visited = set()
    stack = [start]
    while stack
    vertex = stack.pop()
    if vertex not in visited
        print(vertex)
        visited.add(vertex)
        stack.extend(neighbor for neighbour
        graph[vertex] if neighbour not in visited
graph = {
    'A' = ['B', 'C']
    'B' = ['A','O','E'],
    'C' = ['A','F'],
    'O' = ['B'],
    'E' = ['B','F'],
    'F' = ['C','E']
}
print("DFS intuative")
dfs_intuative(graph, 'A')
```

Output:

DFS iterative:

A
C
F
E
B
D

RESULT:

Thus the depth first search has been implemented successfully.

11/08/24

A* Algorithm

AIM:

To implement A* Algorithm

Program:

```
from heapq import heappop, heappush

class node:

    def __init__(self, position, parent=None):
        self.position = position
        self.parent = parent
        self.g = 0
        self.h = 0
        self.f = 0

    def __eq__(self, other):
        return self.position == other.position

    def __lt__(self, other)
        return self.f < other.f

def a_star(start, goal, grid)
    start_node = Node(start)
```

```
open_list = []
closed_list = set()

heappush (open_list, start_node)

while open_list:
    current_node = heappop (open_list)
    closed_list.add (current_node.
                     position)

    if current_node == goal_node:
        path = []
        while current_node:
            path.append (current_node.position)
            current_node = current_node.parent
        return path [::-1]

    for n in neighbours:
        neighbour_position = (current_node.position
        [0] + n [0], current_node_position [1] +
        n[1])

    if 0<= negative_position [0] < len(grid) and
    0 <= neighbour_position [1] < len (grid [0])
    and grid [neighbour] position [0]] [neighbour_
    position [1]] == 0 :

    if neighbour_node.position in closed_list:
    continue
        neighbour_node.g = current_node.g + 1
    neighbor_node.gh = abs (neighbar_node.positio
                            [0] - goal.node
        position[0]) + abs (neighbor_goal_node.position

    if all (neighbor_node != open_node for open_
    node in open_list)
        heappush (open_list, neighbor_node).
    return none.

    grid = [
        [0,1,0,0,0],
        [0,1,0,1,0],
        [0,0,0,1,0],
        [0,1,1,1,0],
        [0,0,0,0,0]
    ]
    start = (0,0)
    goal = (4,4)
```

path = a_star (start, goal, grid)
print ("path found:", path)

Output:

Pathfound = [(0,0), (1,0), (2,0), (3,0),
(4,0), (4,1), (4,2), (4,3), (4,4)]

RESULT:

Thus the A* Algorithm has been implemented and executed successfully.

18/8/24

AO* Algorithm

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.grap
[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 = Graph(graph, heuristic)
solution = graph_obj.get_solution()
print("Solution:", solution)
```

Output:

```
Expanding : A
Best path for A = ['B', 'C'] with cost 3
Expanding : B
Best path for B = ['E'] with cost 1
Expanding : E
Expanding : C
Best path C = ['G']
Expanding = G
Solution = {A: [B, C], B: ['E'], C: ['G']
```

RESULT

Thus AO* Algorithm implemented successfully.

Implementation of decision tree classification techniques

AIM:

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

Program:

```
import pandas as pd
from sklearn.tree import decision
tree (classification data ={
'Height' = [158, 155, 172, 167, 180,
            187, 180, 164, 177],
'weight' = [45, 57, 72, 68, 87, 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 = Decision Tree classifier()
classifier . fit (X,Y)
height = float (input ("Enter height
in cm) for Prl dict.:"))
Print (f" predicted gender for height
{height} cm and weight {weight} Kg
{Practical_gender}")
```

Output:

```
Enter height (in cm) for prediction:
Enter weight (in kg) for prediction:
Predicted gender for height 189.0cm a
                    weight 61.0kg:
```

RESULT:

Thus the decision tree class
is has been executed sucessfully.

Implementation of clustering techniques

K-Means

4/10/24

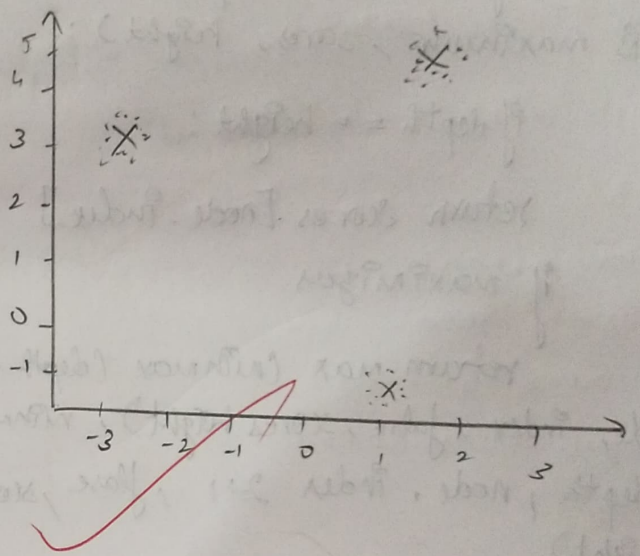
AIM:-

To implement a K-means clustering techniques 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
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='cluster')
plt.title ("K-means clustering results")
plt.ylabel ('Feature 2')
plt.legend()
plt.show()
```

Output:-



RESULT:

Thus the K-means clustering technique has been implemented successfully.

Minmax algorithm

Aim:

To implement minmax algorithm

Program:

```
import math

def minmax (depth node_index,
is maximum, scores, height):

    if depth == height:
    return scores [node_index]

    if maximizes

    return max (minmax (depth +1,
node, index, false, scores height), minmax
(depth, node. index 2+1, flase, score
height)

else
    return min (minmax (depth +1,
node_index. True, score, height,
minmax (depth+1, node. index (2+1
True, scores, height 1-1))
def calculator_Tree height (num_leaves)
```

```
scores = [5, 5, 6, 7, 1, 2, 0 -1]

Tree height = calculator_Tree height
(scores)
optimal score = minmax (0,0, True,
shores, Tree height)
Print ("The optimal score is {optimal_
                          score}")
```

Output:-

The optimal score is=5

RESULT

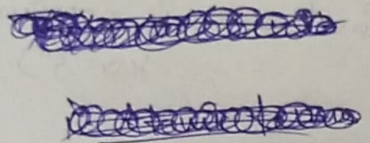
Thus the decision tree progra
has been executed successfully.

Introduction to Prolog.

25/10/24

AIM:-

To learn Prolog terminologies and write basic programs.



Source code:-

KB1

woman(Mia).
woman(Jody).
woman(yolanda).
playsAir(yolanda).
party.

Query 1 = ?- woman(Mia).
Query 2 : ?- plays Airguitar(mia).
Query 3 : ?- party.
Query 4 : ?- concert.

Output:-

?- woman(Mia).
true
?- playsairguitar(Mia).
false
?- concert.

Error = unknown procedure. concert to

KB2

happy(yolanda).
listens2music(mia).
listen2music(yolanda):- happy(yolanda).
playsairguitar(mia):- listens2music(mia).
playsairguitar(yolanda):- listens2music(yolanda).

Output:-

?- playsairguitar(mia).
true
?- 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).

Output:

```
?- likes (dan, X)
X = sally
?- married (dan, sally)
true
?- married (john, brittney)
false
```

RESULT:

Thus the basic prolog programs has been executed successfully.

8/11/24

Prolog - Family Tree

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).
parent of (chris, peter).
parent of (chris, betty).
parent of (helen, peter).
parent of (kevin, betty).
parent of (kevin, chris)
parent of (kevin, lisa)
```

parent of (jeny, john)

parent of (jeny, helen)

/* Rules : */

/* son, Parent */

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)

brother (X,Y): male (Y), father (X,Z),
father (Y,W), Z = W.

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

Output:

male. (Peter)
true
father (chris, peter)
true
father (chris, betty)
false
mother (chris, X)
X = betty
brother (chris, helen)
false

RESULT: Thus prolog for family tree program has been executed successfully.