INDEX

NAME: Abishek Swamthan STD.: CSE SEC.: A ROLL NO.: 008 SUB.:

S. No.	Date	Title	~~Page No.~~ marks	Teacher's Sign / Remarks
1	24/7/24	Basic Python program using Google colab	9	
2		Project - Details	9	
3.		code for DFS search N-Queens Problems	10	
4.		N. Queens Problems	10	
5.		A* Algorithm	10	
6.		Water Jug Problem	10	
7.		Implementation of decision tree classification	10	
8.		K-means clustering	10	
9.		Artificial neural network	10	
10.		Introduction to Prolog.	10	
11.		Prolog family tree	10	
12.		unification and resolution	10	

completed

Basic Python Programs

24/7/24

1) Factorial

```
def factorial(n):
if A ==0
    return 1
else:
    return n* factorial (n-1)
num = int (input ("Enter a number:")
Print ("Factorial of ", num 'is" factorial (num)
```

output

```
Enter the number:5
Factorial of 5 is 120
```

2) sum of digits:

```
def sum-of-digits (n):
    if n ==0:
        return 0
    else:
        return n%10 + sum-of-digits(n//10)
num = int (input ("Enter the number:"))
Print ("Sum of digits of", num, "is", sum-of
                                -digits(num))
```

output

```
Enter the number :45
sum of digit is :9
```

3 Palindrome

```
def is_palindrome
cleaned_s = ''.join(c.lower() for c in s if
return cleaned_s == cleaned_s[::-1]
input_str = input("Enter a string:")
if is_palindrome(input_str):
    Print (f"{input_str}" is a palidrome
else
    Print (is not palindrome)
```

output :

Enter a string: malayalam

"malayalam is a palindrome.

4. Amstrong no.

```
def is_amstrong_number(number)
  sum = 0
  while (number > 0);
    digit = number % 10
      sum += digit**3
      number = 10
  return sum
number = int (input ())
b = number
if is_amstrong_number
Print ("yes")
else
    Print ("no")
```

output

no

5, swapping of two numbers.

```
def swap(a,b
    a = a-b
    b = a+b
    a = b-a
    return a,b
a=int (input ())
b=int (input ())
print( Swap (a,b)
```

output:

10, 20
20,10

6, odd or even

```
num = int(
def check_odd or_even (number);
    if no %.2 ==0:
        return "even"
    else:
        return "odd"
result = check_odd or_even (num)
Print ("The no {num} is {result}.")
```

output

Enter of no : 7
7 is odd no.

7 area of rectangle:

```
l = int (input())
b = int (input())
Print (l+b)
```

output

```
l = 10
b = 20
  200
```

8 GCD

```
def gcd (a,b)
while b:
a, b = b, a%b
return a
num1 = int (input())
num2 = int (input())
print (gcd (n,m))
```

output:

```
Eenter num1 : 48
Enter num2 : 18
The gcd is : 6
```

9 Leap Year

```
year = 2024
if (year % 4 == 0 and year % 100 != 0  year % 400 == 0):
    print(year, "is a leap year")
else:
    print(year "is not a leap year")
```

output:

2024 is a leap year.

10 Pattern printing:

```
for i in rang(5):
    for j in range(i+1):
        print("*", end=" ")
    print()
```

output

```
*
* *
* * *
* * * *
* * * * *
```

11 simple array.

```
arr = [int(x) for x input ("Enter the number separated by space").split()]
print ("Array:", arr)
```

31/07/2024

output:

Enter the number separated by space = 8
Array: [8]

Facial Recognition Based Login System

Domine Type:

*Domain: Biometrics, Cybersecurity, Artificial Intelligence

Algorithm:

Convolutional Neural Networks, Possibly combined with feature extraction techniques like deep learning etc

use case:

Secure and convenient user authentication for system requring high levels of security, such as banking apps etc

objective:

To develop a secure and user friendly authetication system that uses facial recognition technology to replace or complement traditional login methods like password or PINs

People Benefit

. user convenience: user benifits from a faster and more convenient login experience eliminating the need to remember password.

Enhanced security: offer a higher level of security compared to traditional password system.

Accessibility: Helps users with disabilities by providing an alternative to typing passwords

Corporate Benefit: Reduces the likelihood of breaches due to weak or stolen password, thus enhancing overall cybersecurity for organizations.

Ex no: 3

Code for DFS search

Aim:

To write the python code for the DFS search

```
def dfs (graph, start, visted = None):
    if visited is none:
    visited = set()
    visited.add(start)
    Print (start, end = " ")
    for neighbour in grap [start]:
    if neighbour not in visited
        dfs (graph, neighbour, 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, the python programs for implementing DFS search algorithm was executed successfully

Ex no: 4 N-Queens

Aim:

To write the program code for the N - Queen's problem

```
def share-diagnol (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 - diagnol (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:

Found solution [5,3,1,7,4,6,0,2] in 688 tries

Found solution [5,2,6,1,7,4,0,3] in 421 tries

this is the solution for 8 queens problem

```
- - - - - Q - -        - - - - - Q - -
- - - Q - - - -        - - Q - - - -
- Q - - - - - -        - - - - - - Q -
- - - - - - - Q        - Q - - - - -
- - - - Q - - -        - - - - - - Q
- - - - - - Q -        - - - Q - - -
Q - - - - - - -        Q - - - - - -
- - Q - - - - -        - - Q - - - -
```

for 4 queens:

found solution [1,3,0,2] in 7 tries

found solution [2,0,3,1] in 32 tries

Result:

Thus, the python program for 8-queens and 4 queens problem was executed successfully

③ A* algorithm Exno: 5

Aim:

To write a python code for A* algorithm

Code:

```
import heapq
class node:
    def -init- (self, Parent = None, Position = None):
        self . parent = Parent
        self . position = Position
        self . g = 0
        self . h = 0
        self . f = 0
    def -eq- (self, other):
        return self . position = other . positin
def astar (maze, start, end):
    start - node = Node (Node, start)
    end - node = Node (None, end)
    open - list = [ ]
    closed - list = [ ]
    open - list . append (start - node)
    while len (open - list) > 0 :
        current - node = open - list [0]
        current - index = 0
    for index, item in enumerate (open - list):
        if item . f < current - node . f:
            current - node = item
            current - index = index
    open - list . POP (current - index)
    closed - list . append (current - node)
    if current - node == end - node
        Path = [ ]
```

```
current = current-node
while current is not none:
    Path. append (current. position)
    current = current. parent
return path[::-]
children = []
for new-position in [(0,-1),(0,1),(-1,0),(1,0),(-1,-1),(-1,1),
(1,-1),(1,1)]:
        node-position = (current-node. position[0]+
        new-position[0], current-no
        new-position[1])
    if node-position[0]>(len(maze)-1) or node-position
<0 or node-position[1]>(len(maze[len[maze)-1]))
continue
if maze[node-position[0]][node-position[1]]!=0:
continue
if ~~maze[node~~
newnode = Node (current-node, node-position)
children-append (new-node)
for child in children
    for closed-child in closed-list
        if child == closed-child:
continue
        child.g = current-node.g+1
        child.h = ((child. position[0]-end-node.
                                position[0])^2)
for open-node in open-list
    if child == open-node and child.g > open-node.g
continue
```

```
def main():
    mare: [[0,0,0,0,1,0,0,0,0,0],
           [0,0,0,0,1,0,0,0,0,0]
           [0,0,0,0,1,0,0,0,0,0]
           [0,0,0,0,1,0,0,0,0,0]
           [0,0,0,0,1,0,0,0,0,0]
           [0,0,0,0,0,0,0,0,0,0]
           [0,0,0,0,1,0,0,0,0,0]
           [0,0,0,0,1,0,0,0,0,0]
           [0,0,0,0,1,0,0,0,0,0]
           [0,0,0,0,0,0,0,0,0,0]]
    start = (0,0)
    end = (7,6)
    Path = astar(mare, start, end
    Print (Path)
if __name__ = "__main__"
```

output:

[(0,0), (1,1), (2,2), (3,3), (4,3), (5,4), (6,5), (7,6)]

Result:

Thus, the python program for A* algorithm is verified and executed successfully.

Ex no: 6 Water Jug:

Aim: To write a python code for water jug using DFS

code

```
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-3to-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:
        visited = set()
        stack = [start]
    while stack:
        state = stack.pop()
        x,y = state
    if state in visited:
    continue
    visted.add (state)
    print (f"Goal reached: {state}")
    return state
```

```
next_states = [fill_4_gallon (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_states 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)
               : (0,3)
               : (3,0)
               : (3,3)
               : (4,2)
               : (4,0)
               : (1,3)
               : (1,0)
               : (4,1)
               : (2,3)
               : (2,3)
```

Result:

Thus the python program for water Jng problem is executed successfully.

8 Exno:7 Implementation of Decision Tree classification Techniques

Aim:

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

Explanation:

- Import tree from sklearn
- call the function decision Tree classifier() from tree
- Assign values for x and y
- Display the output

```
import pandas as pd
from sklearn.tree import DecisionTree
                                classifier

data a={
    'Height':[152,155,172, 185, 167, 180, 157, 180,
                        184, 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 = Decision Tree classifier()
classifier. fit(x,y)
Height = float(input ("Enter height (in cm)
                for prediction:"))
```

output:

The predicted gender for the input [[5.5, 14.5]] is Female.

Result:

Thus the program to implement decision tree is executed successfully

K-means clustering

Aim:

To implement a K-means clustering technique using python language

Explanation:

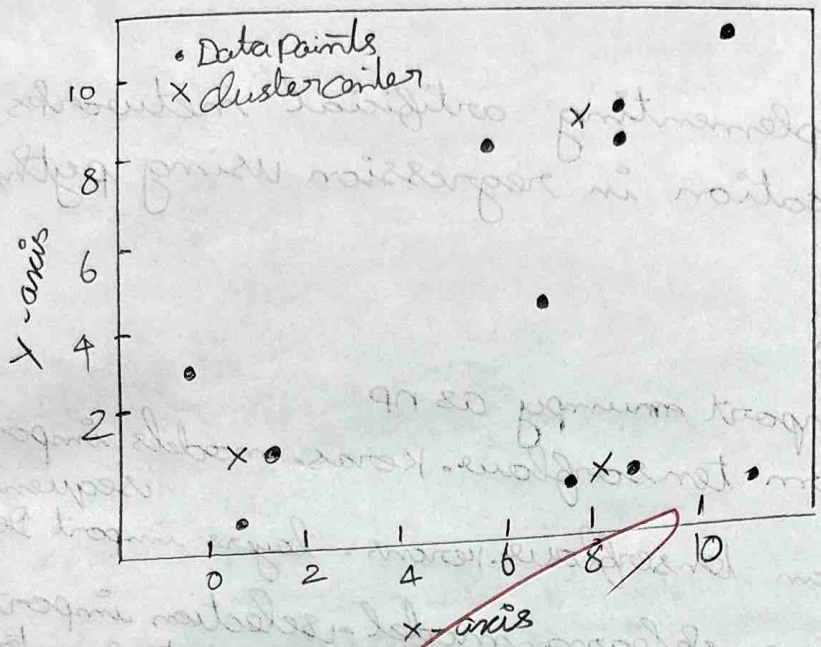
- Import K means from sklearn
- assign x and y
- call the function on K Mean x

Program:

```
From sklearn.cluster import
import numpy as np
import matplotlib pyplot as plt
x = np. array ([1,2], [1.5, 1.8], [5,8],
    [8,8], [1,0.6], [9,11], [8,2], [10,2]
    [9,3], [8,9], [0,3], [6, 4])
K means = KMeans (n-cluster=3)
K means . fit(x)
centers = K means . cluster-centers-
plt-scatter (x [:, 0], x[:,1], c = labels,
marker = 'o', label = 'Data Points')
c = 'red', marker = 'x', label = 'cluster
                                centers')

plt. x label ("x-axis")
plt . y label ("y-axis)
plt . legend()
plt . show()
```

output



Result:

Thus an algorithm using k-means was successfully executed.

Artificial Neural network

Aim:
To implementing artificial networks fo
an application in regression using python.

Program:

```
import numpy as np
from tensorflow.keras.models import
                                sequentia
from tensorflow.keras.layers import Den
from sklearn.model_selection import
                                train_test
from sklearn.preprocessing import scale
from sklearn.metrics import mean_squared
                                _err

np.random.seed(42)
x = np.random.rand(1000,3)*10
y = x[:,0]*300 + x[:,1]*500 + x[:,2]*100 + 5000
+ np.random.randn(1000)*100
x_train, y_test_size = 0.2, random_state
y_test = Scaler.Transform(x_test)
model = Sequential()
model.add(Dense(64, input_dim=x.shape
    activation='relu'))
history = model.fit(x_train y_train, epo
batch_size=32 validation_split=02, verbose
Print("Mean squared Error on testdata",m
```

output:
Mean Absolut Error on testdata: 717.356384273
Mean squared error on test data: 724649.3560191

Result Thus Artificial network was successfully implemented

Introduction to Prolog:

Aim:

To learn prolog terminologies and write basic programs.

Terminologies:

1. Atomic terms:

Atomic terms are usually strings made up of lower and upper case, digits and the underscore with a lowercase letter.

Eg: dog
ab-C-321

2. Variables:

Variables are strings of letter, digits, and the underscore, starting with a capital letter, or an underscore

Eg: Dog
Apple-420

3. Compound terms:

Compound terms are made up of a PROLOG atom and a number of arguments, variables, or other compound enclosed in parenthesis and separated by commas.

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

4. Facts:

A fact is a predicate followed by a dot

Ex: bigger-animal (whale)
life-is-beatiful

5. Rules:

A rule co-exists of a head (a predicate) and a body (a sequence of predicates separated by commas)

Source code:

KB1:

```
woman(mia).
woman(Body).
woman(yolanda).
Party.
```

Query 1: ?- woman(mia).

Query 2: ?- PlaysAirGuitar(mia)

Query 3: ?- Party

Query 4: ?- concert

output:

```
? - woman(mia)
true
? - PlaysAirGuitar
false
? - Party
true
? - concert
ERROR: unknown procedure: concert/0 (DWIM could not find
                                    correct goal)
```

KB 2:

```
happy(yolando
listen2music(mia)
listen2music(yolanda) :- happy(yolanda)
```

output:

```
? - PlaysAirGuitar(mia)
true
? - PlaysAirGuitar(yolanda)
```

KB 3:

```
likes(dan, sally).
likes(sally, dan).
likes(John, britney).
```

output:

? - likes (dan x)

x = sally

? - married (John, brittany)

KB4:

food (burger)

food (sandwich)

food (Pizza)

lunch (sandwich)

Dinner (Pizza)

meal (x) :- food (x)

output:

? - food (Pizza)

True

? - meal (x), lunch (x)

x = sandwich

? dinner (sandwich)

false

Result:

Thus the basic problems to learn prolog technologies has been executed successfully

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 (usa)
female (helen)
Parent of (Chris, peter).
Parent of (Chris, betty)
Parent of (helen, Peter)
Parent of (kevin, chris)
Parent of (Jeny, John)
/* Rules: :*/
/* son, Parent
* son, grandparent*/
father(x,y) :- male(y) Parent of (x,y)
mother (x,y) :- female(y). parent of (x,y)
grand mother (x,y) :- male (y), parent of (y,z)
                      parent of (z,y)

grand mother (x,y) :- female (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
```

Result:

Thus the prolog Problems to implement and execute family tree was successfully completed

Unification and Resolution

Aim:
To execute programs based on unification and resolution

Ex:
Let us see for below the prolog program - how instantiation take place after querying

FACTS: likes(john, jane)
likes(jane, john)

Query: ?- likes(john, x)
x = jane

Here upon asking the query first prolog to search matching terms in Facts in Top-down manner for likes predicate with arguments and it can match likes(john, ...)
x = jane.

Ex:2 at the prolog query prompt, where you write bellow query.
?- owns(x, car(bmw)) = owns(y, car(c)) you will get answer: x y.c = bmw

Here owns(x, car(bmw)) and owns(y, car(c)) unifies - because i) predicate names own are same on both side.
ii) 2nd argument with car predicate inside the brackets are same both side a {x\y} and c in insitated to bmw, --- writeen as {bmw/c} and this is called unification with

i) select two clauses that contain conflicting terms
ii) can those two clauses
iii) come out the confliting terms

For example we have following statements

i) If it is a pleasant day you will strawberry Picking

ii) If you are doing strawberry picking you are happy

i) strawberry-picking ← pleasant

ii) happy ← strawberry-picking and again these statements can be written in CNF like this

i) (strawberry-picking ∨ ¬ pleasant)

ii (happy ∨ ¬ strawberry-picking)

By resolving these two clauses and cancelling out the conflicting terms strawberry-picking and ¬ strawberry-picking, we can have one new clause

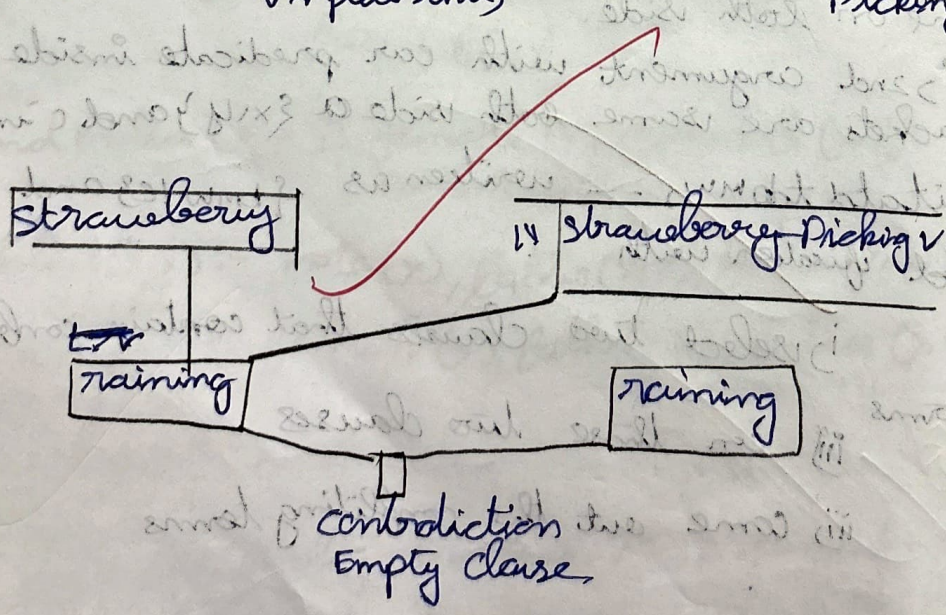
3) pleasant ∨ happy

Have ? see the figure on right

when we write above new clause in infer or implies from, we have pleasant → happy or happy ← pleasant

i.e If it is a pleasant day you are happy

i) (strawberry-picking ∨ pleasant) 2 (happy ∨ ¬ strawberry Picking)



Part (I): English sentences

1, If it is sunny and warm day you will enjoy

2, If it is warm and pleasant day you will do strawberry picking

3, If it is raining then no strawberry picking

4, If it is raining then you will get wet

5, It is warm day

6, It is sunny.

Part (II): Propositional statements

1, enjoy ← sunny ∧ warm

2, strawberry-picking ← warm ∧ pleasan.

3, warm

4, raining

5, sunny

6, raining

Part III : CNF

1, (enjoy ∨ ¬ sunny ∨ ¬ warm) ∧

2, (strawberry - picking ∨ ¬ warm)

3, (warm) ∧

4, (raining) ∧

5, (sunny)

Part (IV): other statements we want to prove by refutation:

(Goal1): you are not doing strawberry picking

(Goal2): you will enjoy

Source code:

```
enjoy:- sunny, warm
strawberry-picking:- warm, pleasant
wet :- raining
warm
raining
sunny.
```

output

```
? not strawberry-picking
true
? -enjoy
true
?-wet
true.
```

Result:

Thus the programs based on unification and resolution has been executed successfully.