# INDEX

NAME: R. Ashmita STD.: CSE SEC.: A ROLL NO.: 220701031 SUB.: POAI Observation

| S. No. | Date | Title | ~~Page No.~~ marks | Teacher's Sign / Remarks |
|---|---|---|---|---|
| 1. | 31/07/2024 | Sample Python Programs | 10 | |
| 2. | 07/08/24 | Email Spam Detection Abstraction | 10 | |
| 3. | 04/09/2024 | n-Queens | 10 | |
| 4 | 11/9/24 | A* Search. | 10 | |
| 5. | 18/9/24 | DFS. | 10 | |
| 6. | 16/10/24 | Implementation of ANN using python regression | 10 | |
| 7. | 25/9/24 | Implementation of decision trees. Classification techniques | 10 | |
| 8. | 9/10/24 | Implementation of clustering technique using K-Means | 10 | |
| 9. | 23/10/24 | minmax | 10 | |
| 10. | 6/11/24 | Introduction to prolog | 10 | |
| 11. | 6/11/24 | Prolog family Tree. | 10 | |
| 12. | 14/9/24 | AO* | 10 | |

completed

23118 A

EXPERIMENT-1. SAMPLE PROGRAMS :

1.
```
num = int(input("Enter a number: "))
flag = False
if num == 1:
    print(num, "is not a prime number")
elif num > 1:
    for i in range(2, num):
        if (num % i) == 0:
            flag = True
            break

    if flag:
        print(num, "is not a prime number")
    else:
        print(num, "is a prime number")

else:
    print(num, "is not a valid number to check
        for prime")
```

OUTPUT:-

Enter a number : 5
5 is a prime number.

---

2)
```
word = input()
if word == word[::-1]:
    print(word, "is a pallindrome")

else:
    print(word, "is not a palindrome")
```

OUTPUT

hello
hello is not a palindrome.

3)
```
def fibonacci(n):
    fib_sequence = [0,1]
    for i in range(2,n):
        next_number = fib_sequence[-1] + fib_sequence[-2]
        fib_sequence.append(next_number)
    return fib_sequence

n = int(input("Enter the number of Fibonacci numbers to generate:"))
print(fibonacci(n))
```

OUTPUT

```
Enter the number of fibonacci numbers to generate: 7
[0, 1, 1, 2, 3, 5, 8]
```



4)
```
def factorial(n):
    if n==0:
        return 1
    else:
        return n*factorial(n-1)

number = int(input("Enter a number:"))
print(f'Factorial of {number} is {factorial(number)}.')
```

OUTPUT

```
Enter a number: 5
Factorial of 5 is 120.
```

5)
```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
    for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1],

arr[j]
    return arr.

numbers = list(map(int, input("Enter numbers
 separated by spaces: ").split()))

sorted_numbers = bubble_sort(numbers)
print(sorted_numbers).
```

6).
```
num = int(input("Enter a number: "))
sum = 0
temp = num
while temp > 0:
        digit = temp % 10.
        sum += digit ** 3
        temp //= 10

    if num == sum:
        print(num, "is an armstrong no").
    else:
        print(num, "is not an Armstrong number").
```

OUTPUT

```
Enter a number: 5
5 is not an Armstrong number.
```

7)
```
num = int (input ("Enter a number:"))
if num < 0:
    print ("Enter a positive no").
else:
    sum = 0
    while (num > 0):
        sum += num
        num -= 1
    print ("The sum is", sum)
```

8)
```
num1 = int (input ("Enter a number:"))
num2 = int (input ("Enter a number:"))
num3 = int (input ("Enter a number:"))
if (num1 >= num2) and (num1 >= num3):
    largest = num1
elif (num2 >= num1) and (num2 >= num3):
    largest = num2
else:
    largest = num3
print ("The largest number is", largest)
```

OUTPUT:-

```
Enter a number: 5
Enter a number: 7
Enter a number: 2
The largest number is 7.
```

9).
```
X = 5
Y = 10
temp = X
X = Y
Y = temp
```

```
print ("The value of x after swapping : {}".format(x))
print ('The value of y after swapping : {}'.format(y))
```

OUTPUT:

The value of x after swapping : 10

The value of y after swapping : 5

10).
```
def compute_lcm(x,y):
    if x>y:
        greater=x
    else:
        greater=y
    while(True):
        if((greater%x==0) and (greater%y==0)):
            lcm=greater
            break
        greater+=1
    return lcm
num1 = 54
num2 = 24
print ("The LCM is ", compute_lcm(num1,num2))
```

31/07/2021

OUTPUT:

The LCM is 216.

# Email Spam Detection.

## Domain :-

Email spam detection is a crucial area within the field of cybersecurity and digital communication.
It focuses on identifying and filtering out unsolicited, often malicious or irrelevant, email messages from reaching users inboxes. Effective spam detection enhances user experiences, safeguards against potential threats, and optimizes e-mail management

## Objective

The primary objective of email spam detection using machine learning (ML) is to develop a robust system that can accurately classify incoming e-mails as either "spam" or "harm" (non-spam). This involves creating a model that learns from historical email data to recognize patterns and features indicative of spam. The key goals include:

High Accuracy : Achieve a high classification accuracy to minimize false positives and false negatives.

Adaptability :- Ensure the model can adapt to new and evolving spam techniques by incorporating mechanisms.

Scalability:- Develop a solution that can handle large volumes of email data efficiently.

Target People

The target audience for this email email Spam detection.

* Individual users:- People seeking to improve their email experience by reducing spam.
* Business :-
  * Email Service Providers.
  * companies needing to protect their communication channels from spam, attempts and other email

4/09/2024

# N-Queens.

```
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), 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 solve_n_queens(board, col):
    if col >= len(board):
        return True
    for i in range(len(board)):
        if is_safe(board, i, col):
            board[i][col] = 1
            if solve_n_queens(board, col+1):
                return True
            board[i][col] = 0
    return False

def print_board(board):
    for row in board:
        print(" ".join(str(x) for x in row))

def solve():
    n = 8
    board = [[0 for _ in range(n)] for _ in range(n)]
```

```
if not Solve_n_queens(board, 0):
    print("Solution does not exist")
    return False

print_board(board)
return True

solve().
```

Output

```
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
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 1 0 0 0 0 0
True.
```

Result:

Thus n-Queens program was executed successfully

11/9/24

# A* algorithm

Program:

```
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] +
                n=v                                  heuristic (n):

        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 closedset:
                    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 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']

['A', 'E', 'D', 'G'].

Result

Thus the program for A* search was implemented successfully

18/9/2024

# DFS

Code

```
def add_edge(adj, s, t):
    adj[s].append(t)
    adj[t].append(s)

def dfs_rec(adj, visited, s):
    visited[s] = True
    print(s, end=" ")
    for i in adj[s]:
        if not visited[i]:
            dfs_rec(adj, visited, i)

def dfs(adj, s):
    visited = [False] * len(adj)
    dfs_rec(adj, visited, s)

if name_ == "main":
    V = 5
    adj = [[] for _ in range(V)]
    edges = [[1,2], [1,0], [2,0], [2,3], [2,4]]
    for e in edges:
        add_edge(adj, e[0], e[1])
    source = 1
    print("DFS from source:", source)
    dfs(adj, source)
```

Output

DFS from source: 1
1 2 0 3 4

Result:-
Thus the program was executed and output was verified.

Implementating artificial neural networks for AN application using python - Regression

AIM:

To implementing artificial neural networks for an application in Regression using python

EXPLANATION:-

1. Generate synthetic regression data using make_regression with 1000 samples, 100 features, and noise of 0.05
2. Split the data into training (80%) and testing (20%) sets using train_test_split.
3. Initialize the MLP Regressor model with a maximum of 1000 iterations
4. Fit the model to the training data using clf.fit(x_train, y_train).
5. Evaluate the model's performance by calculating and printing the $R^2$ score for both the training and testing dataset.

CODE:

```
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_regression
import numpy as np.
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
X,y = make_regression (n_samples=1000, noise=0.05,
```

```
X_train , X_test , Y_train , Y_test = train_test_split
                                    (X, y , test_size = 0.2 , shuffle = True,
                                                              random_state = 42)
clf = MLPRegressor (max_iter = 1000)
clf . fit (X_train , Y_train)
```

RESULT:

Thus the program was executed successfully and output is verified

# Implementation of Decision Tree classification Technique

AIM:

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

EXPLANATION:

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

CODE:

```
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 = DecisionTree Clasifier()
classifier . fit (x, Y)
height = float (input ("Enter height (in cm) : "))
weight = float ( input ("Enter weight (in cm) : "))
prediction_data = pd . DataFrame ([[height, weight]],
                  columns =['Height', 'Weight'])
predicted_gender = classifier . predit (prediction_data)
Printf( f" Predicted gender : { predicted_gender [0]} ")
```

OUTPUT :

Enter height (in cm) for prediction : 169
Enter weight (in kg) for prediction : 69
Predicted gender for height 109.0 and weight 61.0 kg : Female

RESULT

Thus the program was executed successfully and output is verified

# Implementation of clustering Techniques K-Means

AIM:

To implement a K-Means clustering technique using python language

EXPLANATION:

- Import KMeans from sklearn.cluster
- Assign X and Y
- Call the function KMeans()
- Performs scatter operation and display the result.

CODE:

```
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 Result')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

RESULT:

Thus the program was executed successfully and opp is verified

# AO* algorithm

AIM:

To implement AO* algorithm using Python language.

CODE:-

```
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)
```

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 for C : ['G'] with cost 3.
Expanding : G
Solution : {'A' : ['B', 'C'], 'B' : ['E'], 'C' : ['G']}
```

RESULT:

Thus the AO* algorithm was executed and implemented successfully.

# MINMAX

**AIM:**

To implement min max algorithm using python language.

**CODE:**

```
import math
def minimax(depth, node_index, is_maximizer,
            scores, height):
    if depth == height:
        return scores[nodes_index]

    if is_maximizer:
        return max(minimax(depth+1,
          node_index*2, False, scores, height),
        minimax(depth+1, node_index*2+1,
          False, scores, height))

    else:
        return min(minimax(depth+1, node_inder
          *2, True, scores, 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,-]
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:

The optimal score is : 5

RESULT:

Thus the minmax algorithm was executed and implemented successfully.

OUTPUT:

The optimal score is : 5

RESULT:

Thus the minmax algorithm was executed and implemented successfully.

# Introduction to Prolog

AIM:

To learn prolog terminologies and write basic programs

TERMINOLOGIES:

1) Atomic Terms:

Atomic term is usually string made up of lower and uppercase letters, digits and the underscore, starting with a lowercase letter

ex: dog
ab_c_321

2) Variables:-

Variables are string of letters, digits, and the underscore starting with capital letter of underscore

Ex: Dog
Apple_$420

3) Compound Terms:

Compound terms are used to made up of PROLOG atom & a no of arguments (PROLOG terms) and enclosed in parathesis and separated by commas

is_bigger(elephant, x)

4) Facts:

A fact is a ~~pred~~ pseudocode followed by a dot

bigger_animal (whale)

5) Rules:

A rule of a head (a predicate) and a body (a sequence)

is_smaller (X,Y) : is_bigger (Y, X)

SOURCE CODE:

KB1:

woman (mia)
woman (jody)
woman ( yolandia)
playsAirGuitar (jody)
party

O/P

?- woman (mia)
true

?- playsAirGuitar (mia)
false.

?- party
true

?- concert
proddure concreert doesn't exist

Source Code

KB2:

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

O/P

```
?- 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)
```

O/P

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

KB5

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

O/P

```
? - owns (john, X)
    X - car (chevy)
? - own (john, -)
    true
```

?- owns(who, car(chevy))

who = john

?- owns(jane, X), sedan(X)

false

?- owns(jane, X), truck(X)

X = car(chevy)

# PROLOG-FAMILY TREE

AIM:

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

SOURCE CODE:

KNOWLEDGE BASE:

```
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

father (X, Y) :- male (Y), parentOf (X, Y)

mother (X, Y) :- female (Y), parentOf (X, Y)

grandfather (X, Y) :- male (X), parentOf (X, Z);
parentOf (Z, Y)

grandmother (X, Y) :- female (Y), parentOf (X, Z),
parentOf (Z, Y)

brother (X, Y) :- male (Y), female (X, Z),
female (Y, W), Z == W

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

male (Y), parentOf (X, Y)

X = chris, Y = peter.

female (Y), parentOf (X, Y)

X = chris, Y = betty

male (Y), parentOf (X, Z), parentOf (Z, Y)

X = kevin, Y = peter, Z = chris

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

X = kevin, Y = betty, Z = chris