# index

Rahmath

Name : ABILASH. M Std. : Sec. :

Roll No.: Sub :

| S.No. | Date | Title | Page No. | Teacher's Sign / Remarks |
|---|---|---|---|---|
| 1 | 31/07/24 | sample programs | 9 | |
| 2 | 7/8/24 | Problem statement, domain | 10 | |
| 3 | 4/8 | N Queens Problem | 10 | |
| 4 | 4/9 | DFS | 10 | |
| 5 | 11/9/24 | A* algorithm | 10 | |
| 6 | 18/9/24 | AO* Algorithm | 10 | |
| 7 | 25/9 | decision tree | 10 | |
| 8 | 8/10 | KMeans | 10 | |
| 9 | 16/10 | Artificial neural network | 10 | |
| 10 | 23/10 | Minimax | 10 | |
| 11 | 30/10 | introduction to prolog | 10 | |
| 12 | 6/11 | prolog - family tree | 10 | |

completed

EX:01 SAMPLE PROGRAMS USING GOOGLE COLAB

1. Add 2 numbers

```
num1 = 1.5
num2 = 6.3
sum = num1 + num2
Print ("The sum of {0} and {1} is {2}
        format(num1, num2, sum))
```

2. To check armstrong number

```
n = int (input ("Enter a number:"))
s = 0
t = n
while t > 0;
 digit = t % 10
 s += digit ** 3
 t // = 10
 if n == s:
  print (n, "is an Armstrong number")
 else:
  print (n, "is not an Armstrong number")
```

3. Even odd program in python

```
num = int (input ("Enter any number to test
whether it is odd or even:"))
if (num % 2) == 0:
    Print ("The number is even")

else:
    Print ("The provided number is odd")
```

4. To find factorial of a number using recursive function

```
def recursive_factorial(n):
 if n==1:
  return n
 else:
  return n*recursive_factorial(n-1)
number = int(input("User Input:"))
Print("The factorial of", number, "is",
    recursive_factorial(number))
```

5. To find factors of a number

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

6. To count the number of each vowel

```
ip_string = input("Enter a string:")
ip_str = ip_str.casefold()
count = {x: sum([1 for char in ip_str if
    char == x]) for x in 'aeiou'} Print(count)
```

7. To find the LCM of 2 numbers

```
def calc_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 = int(input("enter the first no:"))
num2 = int(input("enter the second no:"))
Print("The LCM of the provided 2 numbers is",
    calc_lcm(num1, num2))
```

8. To find leap year

```
year = int(input("Enter any year:"))
if (year % 4) == 0:
    if (year % 100) == 0:
        if (year % 400) == 0:
            Print("The given year is leap year")
    else
        Print("It is not a leap year")
    else:
        Print("It is not a leap year")
else:
    Print("It is not a leap year")
```

9. To swap two variables

```
var1 = input("Enter the first variable")
var2 = input("Enter the second variable")
temp = var1
var1 = var2
var2 = temp
print("The value of first variable after swapping is:", var1)

print("The value of second variable after swapping is:" var2)
```

10. To calculate area of a triangle

```
p = float(input("enter the length of first side:"))
q = float(input("enter the length of second side:"))
r = float(input("enter the length of third side:"))
s = (p+q+r)/2
area_tri = (s*(s-p)(s-q)(s-r))**0.5
Print("The area of the triangle is %0.2f"
      % area_tri)
```

31/07/2024

Domain Type: Travel and Transportation analysis

Travel and Transportation analytics is a domain that focuses on using data set analysis and predictive modelling to improve decision-making and operational efficiency within the travel and transportation sector.

=> Travel analytics: Involves analysing data related to travel behaviour, booking trends, and customer preferences. It aims to enhance the travel experience, understand travel patterns.

=> Transportation analytics: Focuses on data related to modes of transportation and improve logistics.

Beneficiries:

=> Travellers: Gain access to more accurate prediction.

=> Airline: Benefit from optimized fare prediction.

Problem statement

develop a machine learning model to forecast future flight fares by analyzing historical fare data, booking trends, seasonality, and external factors such as economic conditions, regional events and fuel prices. The objective is to provide accurate fare predictions to assist travellers in finding the best deals, support airlines in optimizing their pricing strategies, and help travel agencies offer competitive and personalized pricing options.

objectives:

1. Accurate prediction: Develop a model that provides precise forecasts of flight fares based on various influencing factors.

2. Dynamic pricing optimization: Assist airlines in setting competitive prices that maximize revenue and occupancy.

3. cost savings for travellers: Enable travellers to identify the best times to book flights to secure lower fares.

EX - 2

AIM: To implement N-Queens problem

```
def print_solution(board):
    for row in board:
        print(" ".join("Q" if cell else "." for cell in row))
def is_safe(board, row, col, N):
    for i in range(row):
        if board[i][col]:
            return False
    for i, j in zip(range(row,-1,-1), range(col,-1,-1)):
        if board[i][j]:
            return False
    for i, j in zip(range(row,-1,-1), range(col,N)):
        if board[i][j]:
            return False
    return True
def solve_n_queens_util(board, row, N):
    if row >= N:
        return True
    for col in range(N):
        if is_safe(board, row, col, N):
            board[row][col] = True
            if solve_n_queens_util(board, row+1, N):
                return True
            board[row][col] = False
    return False
def solve_n_queens(N):
    board = [[False]*N for _ in range(N)]
```

```
if not solve_n_queens_util(board, 0, N):
    print("No solution exists")
    return
print_solution(board)
return True

N=4
solve_n_queens(N)
```

OUTPUT:

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

RESULT:

Thus the above code has been executed successfully and output is verified.

# EXERCISE - DFS

AIM:
To implement DFS search algorithm

```
def dfs(Graph, start, visited Node):
 if visited is None:
   Visited = set()
 Visited.add(start)
 Print(start)
  for neighbour in graph[start]:
    if neighbour Not in visited:
      dfs(graph, neighbour, visited)
    return visited
graph = {
 'A': ['B', 'C'] 'B': ['A', 'D']
 'C': ['A'], 'D': ['B'], }
dfs(graph, 'A')
```

OUTPUT:

A
B
C
D

{'A', 'B', 'C', 'D'}

RESULT:
Thus the above code has been executed successfully and output is verified.

11(a) AIM: To implement A* algorithm in python

```
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) <
          n == v
      if n == stop_node or Graph_nodes[n] == Ne
        Pass
      else:
        for (m, weight) in get_neighbors(n):
        if m not in open_set and m not in
          open_set.add(m)
          Parents [m] = n
          g[m] = g[m] + 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 == start_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_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)],
}
astarAlgo('A', 'G')
```

OUTPUT:

Path found: {'A', 'E', 'D', 'G'}

RESULT:

Thus the above code has been executed successfully and the output is verified.

AIM: To implement AO* algorithm in python.

```
class AONode:
  def __init__(self, name):
    self.name = name
    self.successors = []
    self.cost = float('inf')
    self.is_solved = False
    self.best_child = None
  def add_successor(self, successors, cost):
    self.successors.append((successor, cost))
def ao_star(node, solved):
  if node.is_solved:
    return node.cost
  for successors, cost in node.successors:
    total_cost = cost
    all_solved = True
    for child in successors:
      child_cost = ao_star(child, solved)
      total_cost += child_cost
      if not child.is_solved:
        all_solved = False
    if all_solved and total_cost < node.cost:
```

```
node.cost = total_cost
node.best_child = successors
node.is_solved = any(all([child.is_solved for child in
    successors]) for successors, _ in node.in_node.successors])
solved[node.name] = node.cost
return node.cost
def get_user_input():
num_nodes = int(input("Enter the number of nodes:"))
    nodes = {}
    for i in range(num_nodes):
        node_name = input(f"Enter the name of node {i+1}:")
        nodes[node_name] = AONode(node_name)
    for node_name, node in nodes.items():
        num_successors = int(input(f"how many successor groups
                                does {node_name} have?"))
        for j in range(num_successors):
            successors_input = input(f"Enter space-separated successors
                                for group {j+1} of {node_name}:").split()
            cost = int(input(f"Enter the cost to reach this group of
                                successors from {node_name}:"))
            successors = [nodes[successor] for successor in successors_input]
            node.add_successor(successors, cost)
    start_node_name = input("Enter the start
                                node:")
    return nodes, start_node_name
nodes, start_node_name = get_user_input()
solved = {}
```

```
start_node = node[start_node_name]
optimal_cost = ao_star(start_node, solved)
print(f"\n optimal cost from {start_node_name} is
    {optimal cost}")
print("solved nodes with their costs", solved)

def print_solution_path(node):
    if node.best_child:
        print(f"{node.name} => {', '.join(child.name for
        child in node.best_child)}")
        print_solution_path(child)
print("\n solution path:")
print_solution_path(start_node)
```

OUTPUT:

```
Enter the number of nodes: 5
Enter the name of node 1: A
Enter the name of node 2: B
Enter the name of node 3: C
Enter the name of node 4: D
Enter the name of node 5: E
How many successor groups does A have? 2
Enter space-seperated successors for group 1 of A: C
Enter the cost to reach this group of successors from A: 1
Enter space-seperated successors for group 2 of A: D
Enter the cost to reach this group of successors from A: 1
How many successor groups does B have? 1
Enter space-seperated successors for group 1 of B: E
Enter the cost to reach this group of successors
from B: 3
```

How many successor does C have? 0
How many successor groups does D have? 1
Enter space separated successors for group 1 of D: E
Enter the cost to reach this group of successors from D: 2
How many successor groups does E have? 0
Enter the start node: A
Optimal cost from A is: 2
Solved nodes with their cost: {'D': 2, 'A': 2}

OUTPUT:
Thus the above program has been executed successfully and output is verified.

25/9 Implementation of decision tree classification techniques.

AIM:-
To implement a decision tree classification technique for gender classification technique using python.

Explanation:
☆ Import tree from sklearn
☆ call the function decision tree classifier() from tree
☆ Assign value 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 decision Tree classifier
data = {
'height': [152, 155, 172, 185, 167, 180, 157, 180, 184, 172]
'Height': [45, 57, 72, 85, 68, 78, 22, 90, 66, 88],
'Gender': ['Female', 'Female', 'Male', 'Male', 'Female',
'Male', 'Male']
```

```
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]}")
```

Result: Thus the python programs implement a decision tree classification technique for gender classification is executed successfully.

10/10

Implementation of artificial neural networks for an application using python - regression

AIM:-

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

Program Code:-

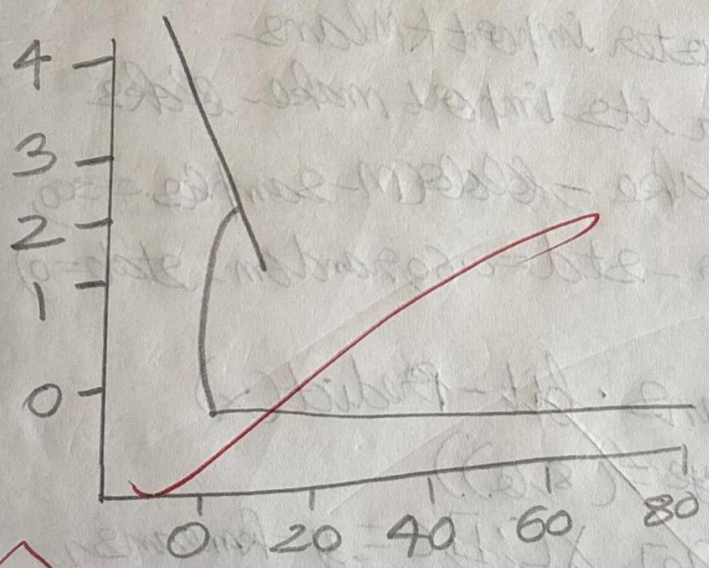
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import standard scale
from keras.models import sequential.
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt.
# Generating synthetic dataset for demonstration.
# replace this with your own dataset
np.randomized(42)
X = np.random.rand(1000, 3) #100 samples
3 features.
if = 3 X[:,0] + 2 * X] = ]** 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(np.array(y-test-y-pred)**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()
```

output:

```
Epoch 1/100
 20/20
   :
Epoch 100/100
 20/20
```




Result:-

Thus the artificial neural networks for regression has been implemented successfully.

8/10 Implementation of clustering technique K-means

AIM:-

To implement a K Means clustering technique using python language.

Explanation:

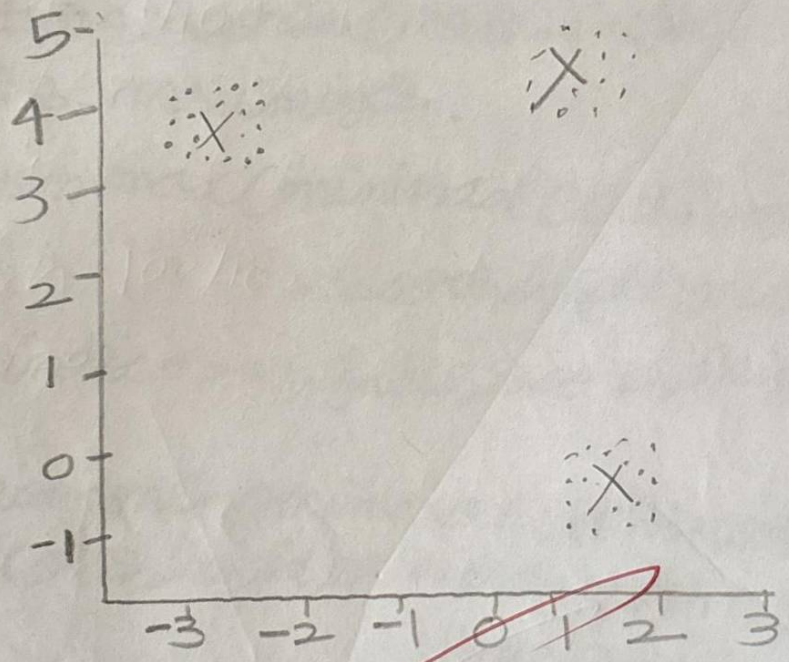
* Import KMeans from sklearn.cluster
* Assign x and y
* Call the function KMeans ()
* Perform scatter operation and display the output.

Program:-

```
import numpy as np
import matplotlib as plt
from sklearn.cluster import KMeans
from sklearn.data ists import make_blobs
X, y_true = make_blobs(n_samples=300,
centers = 3, cluster_std=0.60,random_state=0)
K=3
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.Clusters_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('Feature1')
plt.ylabel('Feature2')
plt.legend()
plt.show()
```

output :-




Result :

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

23/10

# MINIMAX ALGORITHM

## AIM:-

To implement minimax algorithm.

## 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, node_index*2, False,
            *scores, height), minimax(depth+1, node_
   index*2+1, False, scores, height))
 else
 return min(minimax(depth+1, node_index*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,-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:-

The optimal score is: 5

~~Re~~

Result :-

Thus the decision tree program has been executed successfully.

30/10 Introduction to Prolog

AIM:
To learn prolog terminologies and write basic programs.

Terminologies:

1. Atomic terms:-
They are usually strings made up of lower and uppercase letter, digits and the underscore and uppercase letter, digits and the underscore starting with a lowercase letter.
eg. dog, abc_C_321

2. Variables:
They are strings 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 a proper atom and a number of arguments enclosed in parenthesis and separated by commas.
eg: is_bigger(elephant, X). f(g(X, _), 7)

4. Fact
A fact is a predicate followed by a dot.
eg: bigger_animal(whale). life_a_beautiful.

5. Rules
A rules consists of a head and body
eg: is_smaller(X, Y, Yes_bigger(Y, X) aunt(Aunt, Child)
sister(Aunt, Parent), Parent(Parent Child).

Source code :-

KB1

woman(mia).

woman(jody).

woman(yolanda).

Playsair guitar(jody)

Party

Querty1 = ?_woman(mia)

Querty2 = ?_playsAI guitar(Mia)

Querty3 = ?_Party

Querty4 : ? - Concat

Output :-

?_woman(mia)

true

?_playsair guitar(mia)

false

?_Concat

Error = unknown procedure: concat/0

KB2

happy(yolanda)

listens2music(mia)

listen2music(yolanda) :- happy(yolanda)

playsair guitar(mia) = listens2music(mia)

Playsair guitar(yolanda) listens2music(yolanda)

output :-

? Playsair guitar(mia)

true

?- playsairguitar(yolanda)

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(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)

output:

?.. food(Pizza)
true
?- meal(X), lunch(X)
X = sandwich

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 :-

?- owns(john, X)

X = car(chevy)

?- owns(john, _)

true

Result :-

Thus the basic prolog programs has been executed successfully.

6/11 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 (jon)
male( chris)
male (Lumis)
female (betty)
female (jane)
female (lora)
female (helen)
parent of (chris, Peter)
Parent of (chris, betty)
Parent of (Kelvin, lisa)
Parent of (jay, helen)
father(x,y) . male (y) parent (x,y)
mother (x,y) . female (y), Parent (x,y)
grand-father (x,y) :- male (y), parent of (x,z)
parent (z,y)
grand mother (x,y) . female (y) parent of (x,z)
parent (z,y)
brother (x,y) . male (y), father (x,z), father
(y,w). z=w( sister (x,y) grand (x,z)
father (x,z), father (y,z) z=w
```

Output:

male (Peter)
true
father (Chris - Peter)
true
father (Chris, Patrick)
father
mother (Chris, X)
X = Bella
brother (Chris, Helen)

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