INDEX

NAME: Bhuvaneshwari R STD.: _____ SEC.: A ROLL NO.: _____ SUB.: _____

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	09/08/24	8 QUEENS PROBLEM	9	
2.	16/08/24	DEPTH FIRST SEARCH	9	
3	23/08/24	DEPTH FIRST SEARCH WATER JUG PROBLEM	10	
4	30/08/24	MINMAX PROBLEM	10	
5	06/09/24	A* SEARCH ALGORITHM	10	
6	27/09/24	IMPLEMENTATION OF DECISION TREE CLASSIFICATION	10	
7	04/10/24	IMPLEMENTATION OF CLUSTERING TECHNIQUES K-MEANS	10	
8	18/10/24	IMPLEMENTING ARTIFICIAL NEUTRAL NETWORK FOR AN APPLICATION USING PYTHON - REGRESSION	10	
9	25/10/24	INTRODUCTION TO PROLOG	10	
10	08/11/24	PROLOG FAMILY TREE	10	

completed

Ex: 1
09/05/24

8 QUEENS PROBLEM.

AIM:

To write a python progeam for N queens problem.

CODE:

```
def share_diagonal(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_diagonal(i, bs[i], c, bs[c]):
return True

return False

def has_clashes(the_board)
for col in range(the_board, col):
return True
return False

def main():
import random
rng = random.Random()

bd = list(range(8))
num_found = 0
tries = 0
result = []
whil 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:

```
Enter the number of queens: 4
Found solution [3,0,4,1,5,2] in 2 tries
Found solution [4,2,0,5,3,1] in 1 tries
Found solution [1,3,5,0,2,4] in 4 tries
Found solution [2,5,1,4,0,3] in 2 tries
```

Result:

Thus the program for N-queens problem has been executed successfully.

Ex: 2
16/08/24

DEPTH FIRST SEARCH

AIM:

Write a python program for Depth - First Search

CODE:

```
def dfs(graph, start, visited = None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end=" ")
    for neighbour in graph.get(start, []):
        if neighbour not in visited:
            dfs(graph, neighbour, visited)

num_nodes = int(input("Enter the no of nodes:"))
graph = {}
for i in range(num_nodes):
    node = input("Enter node " + str(i+1) + ":").strip()
    neighbors = input("Enter neighbors of " + node + "(comma-sep):")
                .strip().split(",")
    neighbors = [n.strip() for n in neighbors]
    graph[node] = neighbors

print("Graph:", graph)
start_node = input("Enter the starting node:").strip()
print("Starting node:", start_node)
dfs(graph, start_node)
```

~~Result:~~

Output:

Enter the number of nodes : 4

Enter node 1 : a

Enter neighbors of a (comma-seperated) : b,c

Enter node 2 : b

Enter neighbors of b (comma-seperated) : d

Enter node 3 : c

Enter neighbors of c (comma-seperated) : d

Enter node 4 : d

Enter neighbors of d (comma-seperated) : a

Graph : {'a' : ['b', 'c'], 'b' : ['d'], 'c' : ['d'], 'd' : ['a']}.

Result:

Thus the program for Depth First Search is Executed Successfully.

EX: 3

23/08/24

DEPTH FIRST SEARCH – WATER JUG PROBLEM

AIM:

To write a python program for Water Jug Problem

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_max, y)

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(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
        visited.add(state)
        print(f"visiting state: {state}")
```

```
if x == goal_x
    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_state 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)

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)

Goal reached : (2, 3)

RESULT:

Thus the water Jug program is successfully executed

Ex: 4

30/08/24

MINMAX ALGORITHM

AIM:

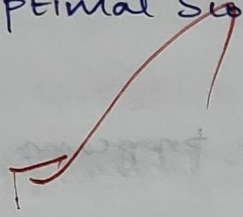
To write a python program for MinMax problem

Code:

```
import math
def minmax(depth, node_index, is_minimizer, scores,
                                                height)

    if depth == height:
        return scores [node_index]

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

                   minmax (depth+1, node_index*2+1, False, scores,
                                                height)

    else:
        return min (minmax (depth+1, node_index*2, False,
                                    scores, height),

                    minmax (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 = minmax (0,0, True, scores, tree_height)
print (f"The optimal score is {optimal_score}")
```

Output:

The optimal Score is : 5



RESULT:

Thus the MinMax problem has been executed Successfully.

Ex: 5

06/09/24

A* SEARCH ALGORITHM

AIM:

To write a python program for A* Algorithm.

CODE:

```
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,
                                   h_func(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)
for neighbor, cost in graph[current_node.name]:
    if neighbor in closed_list:
        continue

    g_new = current_node.gt_cost
    h_new = h_func(neighbor, goal)
    f_new = g_new + h_new
    neighbor_node = Node(neighbor, current_node,
                         g_new, h_new)

    heapq.heappush(open_list, neighbor_node)

return None.

graph = {
    A: [('B', 1), ('C', 4)],
    B: [('A', 1), ('C', 2), ('D', 5)]
    C: [('A', 4), ('B', 2), ('D', 1)]
    D: [('B', 5), ('C', 1), ('E', 3)]
    E: [('D', 3)]
}

def heuristic(node, goal):
    h_values = {
        'A': 4,
        'B': 3,
        'C': 2,
        'D': 1
        'E': 0
```

```
}
return h_values.get(node, float('inf'))

start_node = 'A'
goal_node = 'E'

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', 'B', 'C', 'D', 'E']
```

RESULT:

Thus the python program to for A* Algorithm is executed successfully

Ex: 6
27/09/24

Implementation of Decision Tree Classification Techniques

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 predicting on the basis of given random values for each given feature
* Display the Output

CODE:

```
from sklearn.tree import DecisionTreeClassifier
X=[
[170, 65,42],
[160, 55,38],
[175,70,43],
[155,50,33],
[165,65,39],
[180,80,44],
]
Y=[1,0,1,0,0,1]
clf = DecisionTreeClassifier()
clf.fit(X,Y)
```

```
try:
    height = int(input("Enter height in cm : "))
    weight = int(input("Enter weight in kg :"))
    shoe_size = int(input("Enter shoesize :"))
    sample_data = [[height, weight, shoe_size]]

    prediction = clf.predict(sample_data)
    gender = "Male" if prediction[0] == 1 else "Female"
    print(f"the predicted gender for the input
          {sample_data[0]} is : {gender}")
except ValueError :
    print("Please enter valid numbers for
          height, weight and shoe size.")
```

Output:

Enter Height in cm: 160
Enter Weight in kg: 55
Enter Shoe Size : 38

The Predicted Gender for the input [160, 55, 38] is Female.

RESULT:

The decision tree implementation for gender classification using python is computed successfully

Ex:7
04/10/24

Implementation of Clustering Techniques

K-Means

AIM:

To implement a K-Means clustering technique using python language.

EXPLANATION:

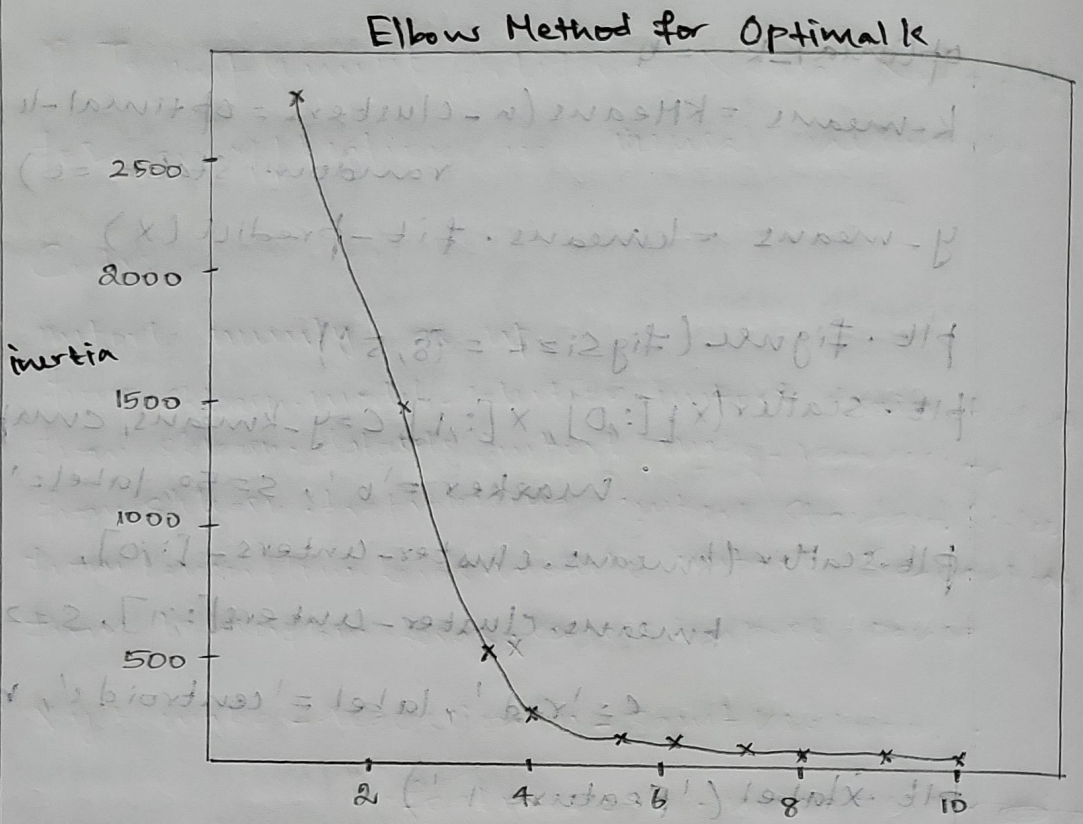
* Import KMeans from sklearn cluster
* Assign x and y
* Call the function kMeans()
* Perform the scatter operation and display the output.

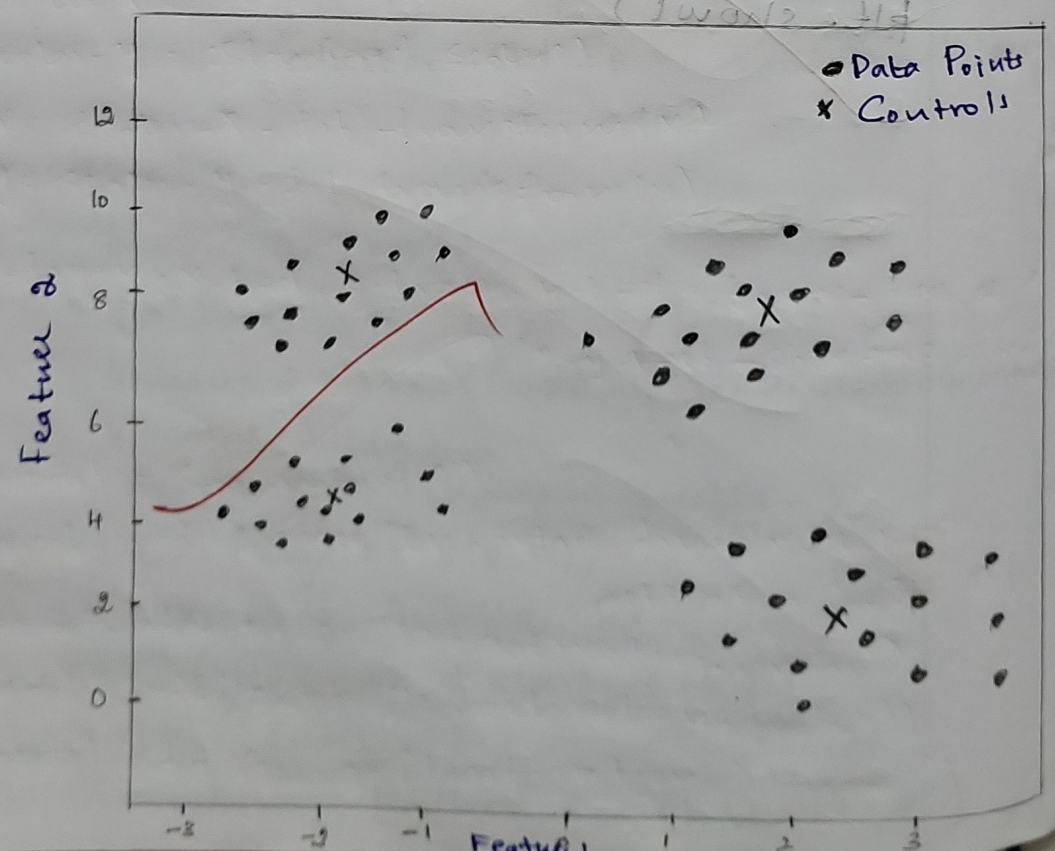
CODE:

```
import numpy as np
import matplotlib.pylot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

X,_ = make_blobs (n_samples = 300, centres = 4,
                  cluster_std = 0.6, random_state = 0)

inertia = []
k_range = range (1,11)
for k in k_range:
    kmeans = KMeans (n_clusters = k, random_state =
    k means.fit(X)
    inertia.append (kmeans.inertia_)
plt.figure (figsize = (8,5))
plt.plot [k_range, inertia, 'bx-'
plt.xlabel ('Number of clusters (k)')
plt.ylabel ('Inertia')
plt.title ('Elbow Method for optimal k')
```

```
plt.show()

optimal_k = 4
k_means = KMeans(n_clusters = optimal_k,
                 random_state = 0)
y_means = kmeans.fit_predict(X)

plt.figure(figSize = (8, 5))
plt.scatter(X[[:,0], X[:,1], c=y_kmeans, cmap='viridis',
            marker='o', s=50, label='Data Points')
plt.scatter(kmeans.cluster_centers_[:,0],
            kmeans.cluster_centers_[:,1], s=200,
            c='red', label='centroids', marker='x')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K Means Clustering')
plt.legend()
plt.show()
```

Output:


Elbow Method for Optimal k
2500
2000
inertia
1500
1000
500
2
4
6
8
10
No of Clusters


K-Means Clustering
Data Points
Centroids
12
10
8
6
4
2
0
Feature 2
-2
-1
1
2
3

RESULT:

Implementation of K-Means Clustering technique using python language is executed successfully.

Ex: 8
18/10/24

Implementing Artificial Neural Networks for an Application using Python - Regression

AIM:

To implement Artificial Neural Networks for an application in neural networks

Code:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_regression
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

X,y = make_regression(n_samples=1000,
                      n_features=10, noise=0.1,
                      random_state=42)

y = y.reshape(-1,1)

scalerX = StandardScaler()
scaler_y = StandardScaler()
X = scaler_X.fit_transform(X)
y = scaler_y.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split
                    (X, y, test_size=0.2, random_state=42)

model = Sequential()
model.add(Dense(64, input_dim=X.shape[1],
                activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='linear'))
```

```
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

history = model.fit(X_train, y_train, epochs=100,
                    batch_size=32, validation_split=0.2,
                    verbose=1)

loss, mae = model.evaluate(X_test, y_test,
                           verbose=0)

print(f"Mean Absolute Error on Test Set: {mae}")

y_pred = model.predict(X_test)
y_pred = scaler_y.inverse_transform(y_pred)
y_test = scaler_y.inverse_transform(y_test)
print("Predicted values:", y_pred[:5].flatten())
print("Actual values:", y_test[:5].flatten())
```

OUTPUT:

Predicted Values : [39.676956 74.06273
-8.287243 -306.44778
86.984655]

Actual Values : [42.67137813 75.01408257
-4.055390 77 -295.72163432
44.43243324]

RESULT:

* Implementation of Artificial Neural Network for an application in Neural Network is successfully executed

Ex 9
25/10/24

Introduction to Prolog

AIM:

To learn ~~prolog~~ PROLOG terminologies and write basic programs.

TERMINOLOGIES:

1. Atomic Terms:

* Atomic terms are usually strings are made up of lower- and upper case letters

Eg: dog, ab-c-321

2- Variables:

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

Eg: Dog, Appk-420

3. Compound Terms:

* Compound Terms are made up of PROLOG atom and a number of arguments (PROLOG terms, ie. atoms, numbers, variables or other compound terms) enclosed in the parenthesis and seperated by commas.

Ex: is_bigger(elephant, X)
f(g(X, _), 7)

4. Facts:

* A fact is a predicate followed by a dot.

Eg: bigger_animal(whale)
life_is_beautiful.

5. Rules:

* A rule consist of a head (a predicate) and a body (a sequence of predicate separated by commas)

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

SOURCE CODE:

KB1:

```
woman(mia).
woman(jody)
woman(yolanda)
playsAirGuitar(jody)
party.
```

Query 1: ?- woman(mia).
Query 2: ?- playsAirGuitar(mia)
Query 3: ?- party.
Query 4: ?- concert.

OUTPUT:

```
? - woman(mia)
true

?.- plays AirGuitar(mia)
false

? - party
true

? concert
ERROR: Unknown procedure: concert/0
        (DWIM could not correct goal)

?-
```

KB2:

```
happy('yolanda)
listens2music(mia).
Listens2music(yolanda) :- happy(yolanda).
playsAirGuitar(mia) :- listens2music(mia
playsAirGuitar(Yolanda) :- listens2music
                            (yolanda)
```

OUTPUT

```
likes(dan, sally)
likes(sally, dan).
likes('john, brittney
likes(
married(X,Y) :- likes(X,Y), likes(Y,X)
```

OUTPUT:

```
?- likes(dan,x)
x=sally

?- married(dan,sally)
true.

?- married (john, brittney)
false.
```

KB4:

```
food(bueger).
food (sandwich).
food (pizza)
lunch (sandwich)
dinner (pizza)
meal(x) :- food (x)
```

OUTPUT:

```
?-
1 · food(pizza)
true

?- meal(x), lunch(x)
x=sandwich

?- dinner (sandwich)
false
?-
```

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
?- owns(who, car(chevy)).
who = john
?- owns(jane, x), sedan(x).
false
?- owns(jane, x), truck(x)
x = car(chevy)
```

RESULT:

* Thus the basic terminologies of PROLOG is learnt and a program is completely written.

Ex 10
08/11/24

PROLOG FAMILY TREE

AIM:

To develop a family tree program using PROLOG with all possible facts and queries.

SOURCE CODE:

KNOWLEDGE BASE:

```
/* FACTS : : */
male(peter).
male(john).
male(chris).
male(kevin).

female(betty).
female(jeny).
female(lisa)
female(helen)

parentof(chris, peter)
parent of (Chris, betty)
parent of (helen, peter)
parent of (helen, betty)
parent of (kevin, chris)
parent of (kevin, Lisa)
parent of (jeny, john)
parent of (jeny, helen)
```

/* RULES :: */

/* son, parent
   son, grandparent */

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

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

grandfather(X,Y) :- male(Y), parentof(X,Z),
parentof(Z,Y)

grandmother(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:

female(Y), parentof(X,Y)

· false

female(Y), parentof(X,Z), parentof(Z,Y)

false

mah(Y), father(X, Z), father(Y, W),
Z == W.

RESULT:

Thus the family tree progeam has been developed successfully using PROLOG