INDEX

NAME: Bhuvaneshwari STD.: CSE(IoT) SEC.: A ROLL NO.: 220701046 SUB.: POAI lab.

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	6/9/24	N-queens problem	10	
2.	13/9/24	Depth First Search.	10	
3.	13/9/24	Depth First Search – Water Jug problem	10	
4.	20/9/24	A* algorithm	10	
5.	27/9/24	Implementation of decision Tree classification technique	10	
6.	27/9/24	Implementation of ANN for an application using Python - Regression	10	
7.	4/10/24	Implementation of k-Means clustering using python	10	
8.	11/10/24	Introduction to Prolog	10	
9.	18/10/24	prolog - family Tree.	10	
10.	25/10/24	Minimax algorithm	10	

completed

220701046

N-QUEENS PROBLEM.

Ex.No.1 220701046

Date: 6/9/24.

```
def is_safe(board, row, col, n):

    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, n), range(col,-1,-1)):
        if board[i][j] == 1:
            return false

    return True.

def Solve_n_Queens_Util(board, col, n):
    If col >= n:
        return True
    for i in range(n):
        If is_safe(board, i, col, n):
            board[i][col] = 1
        If solve_n_queens_Util(board, col+1, n):
            return True
        board[i][col] = 0
    return false
```

```
def Print_board(board, n):
    print("\n Solution")
    for row in board:
        for col in row:
            if col == 1:
                print('Q', end=' ')
            else:
                print('.' , end=' ')
        print()

def Solve_n_queens(n):
    board = [[0 for i in range(n)] for _ in range(n)]
    if not Solve_n_queens_util(board, 0, n):
        print("No solution exists")
        return False
    Print_board(board, n)
    return True

n = int(input("Enter the value of N: "))
Solve_n_queens(n)
```

220701046

Output:

Enter the value of N: 4

Solution:

. . Q .
Q . . .
. . . Q
. Q . .

DFS Algorithm

Aim:
To write python code for DFS algorithm

220701046

Code:

```
def __init__(self, value):
    self.value = value
    self.neighbors = []
def add_neighbors(self, neighbor):
    self.neighbors.append(neighbor)
Class Graph:
    def __init__(self):
        self.nodes = {}
    def add_node(self, value):
        if value not in self.nodes:
            self.nodes[value] = Node(value)
    def add_edge(self, from_value].add_neighbor
        if from_value in self.nodes and
            to_value in self.nodes:
            self.nodes[from_value].add_neighbor
                (self.nodes[to_value])
            self.nodes[to_value].add_neighbor
                (self.nodes[from_value])
```

220701046

```
def dfs(self, start_value):
        visited = set()
        stack = [self.nodes.get(start_value)]
        while stack:
            node = stack.pop()
            if node and node.value not in visited:
                print(node.value)
                visited.add(node.value)
            for neighbor in reversed(node.neighbors):
                if neighbor.value not in visited:
                    stack.append(neighbor)
```

```
# Example (input)
graph = Graph()
graph.add_node('1')
graph.add_node('2')
graph.add_node('3')
graph.add_node('4')
graph.add_node('5')
graph.add_node('6')
graph.add_edge('1', '2')
graph.add_edge('1', '3')
graph.add_edge('2', '4')
graph.add_edge('2', '5')
graph.add_edge('3', '6')
graph.add_edge('5', '6')
print("DFS")
graph.dfs('1')
```

Output:

```
DFS:
1
2
4
5
6
3
```

Result: Thus, program for DFS is implemented successfully

Ex. 3
13/9/24

A* algorithm

220701046

Aim:

To write python code for A* algorithm

Code

```
def astar(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] + heuristic(n):
                n = v
            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 closed_set:
                        open_set.add(m)
                        Parents[m] = n
                        g[m] = g[n] + weight
```

```
else:
    if 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 exists")
        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

    open_set.remove(n)
    closed_set.add(n)
    print("path does not exists")
    return None

def get_neighbors(v):
    if v in graph_node:
        return Graph_nodes[v]
    else:
        return None

def heuristic(n):
    H_dist = {
        'A' : 11,
        'B' : 6,
```

```
    'C' : 9,
    'D' : 1,
    'E' : 7,
    'G' : 0
}
```

220701046

Output :

path found : ['A', 'B', 'G']

Result :

Thus the program for A* algorithm has been successfully executed.

Exp : 4
20/9/24

Water Jug problem

Aim :

To write a program for water jug problem using PYTHON

220701046

Code :

```
def water_jug_dfs(jug1, jug2, target):
    def dfs(j1, j2, seq, visited):
        if j1 == target or j2 == target
            return seq
        visited.add((j1, j2))
        actions = [("fill", 1), ("fill", 2),
("empty", 1), ("empty", 2), ("pour", 1, 2), ("pour", 2, 1)]
        for actions in actions:
            if action[0] == "fill":
                if action[1] == 1:
                    next_state = (jug1, j2)
                else:
                    next_state = (j1, jug2)
            elif action[0] == "empty"
                if action[1] == 1:
                    next_state = (0, j2)
                else:
                    next_state = (j1, 0)
            else:
                if action[1] == 1:
                    amount = min(j1, jug2 - j2)
                    next_state = (j1-amount, j2+amount)
```

else:

amount = min(j2, jug2 - j2)

next_state = (j1 - amount, j2 + amount)

220701046

if next_state not in visited

next_seq = seq + [action]

result = dfs([next_state[0], next_state[1]],
next_seq, visited)

if result:

return result

return None

visited = set()

return dfs(0, 0, [], visited)

Main code

result = water_jug_dfs(4, 3, 2)

print(result)

o/p:

[('fill', 1), ('fill', 2), ('empty', 1), ('pour', 2, 1),
('fill', 2), ('pour', 2, 1)]

Result:

Thus the program for water jug problem has successfully been implemented

Ex. no: 5
Date: 21/9/24

Implementation of Decision Tree Classification Techniques.

220701046

Aim:

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

- Implement tree from skeleton
- call the fn predict for predicting on basis of given random values for each given feature.
- Assign value for x & y
- Display the output.

Program:

```
import pandas as pd
from sklearn.tree import Decision tree classifier
data = {
    'Height': [152, 155, 152, 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 = DecisionTreeClassifier()
classifier.fit(X, Y)
height = float(input("enter the height (in cm) for Prediction"))
weight = float(input("enter weight (in kg) for predictn"))
```

```
random_values = pd.DataFrame([[height, weight]],
                columns=['Height', 'weight'])
```

220701046

```
predicted_gender = classifier.predict(random_values)
print(f"predicted gender for height {height} cm
    and weight {weight} kg : {predicted_gender[0]}")
```

Output:

```
Enter the height in cm : 154
Enter the weight in kg : 39
predicted gender for height 154.0 cm & weight is 39.0kg
    is Female.
```

Result:

Thus the python code for decision tree classification for gender classification is executed successfully

Ex.no: 6 Implementation of Artificial Neural Networks
Date: 27/9/24 for an application using python - Regression.

Aim:

To implement artificial neural networks for an application using python regression.

220701046

Program:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split.
from sklearn.preprocessing import StandardScaler
from sklearn.models import Sequential.
from keras.layers import Dense
from keras.optimizers import Adam.
import matplotlib.pyplot as plt.

X = np.random.rand(1000, 3).
y = 3 * X[:,0] + 2.* X[:,1] **2 + 1.5 * np.sin(X[:,2] * np.pi)
                + np.random.normal(0.0.1,1000)

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

Scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = Scaler.transform(X_test)
model = Sequential()
model.add(Dense(10, input_dim = X_train.shape[1],
```

220701046

```
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)
history.
y_pred = model.predict(X_test)
mse = np.mean((y_test, y_pred.flatten())**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
plt.title('Training & validation Loss')          Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Epoch 1/100

20/20 ——— 7s 57ms/step loss: 0.81373 mae: 0721 -val-loss: 0.3510 - value-mae: 0.4632.

Echo 2/100

20/20 ——— 1s 8ms/step loss: 4.2982e-04 mae: 0.0165 - val-loss: 0076 val-mae: 0.059

Echo 3/100

20/20 ——— 3s 08 loss: 1.8059 - mae: 0.0106. value-mae: 0.0554.

Predicted value: [39.6765. 74.06273 -306.4476 36.98405]

Actual values: [42.6713813 75.01488257 -295.72162342 44.4323]

Result:

The Implementation of ANN for an application using Python regression is successfully executed.

Ex: 7
Date: 4/10/24

Implementation of K-Means clustering Using python

Aim :

To implement K-means clustering using python.

Explanation:

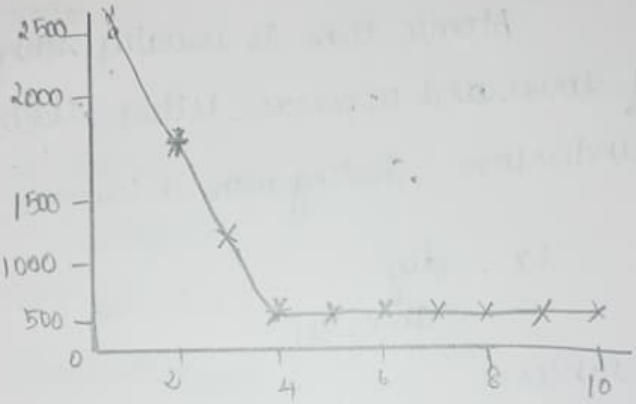
- To implement & import K-means from Sklearn.cluster ....
- Assign X & Y
- Call the fn K means.
- Perform scatter operation and display output.

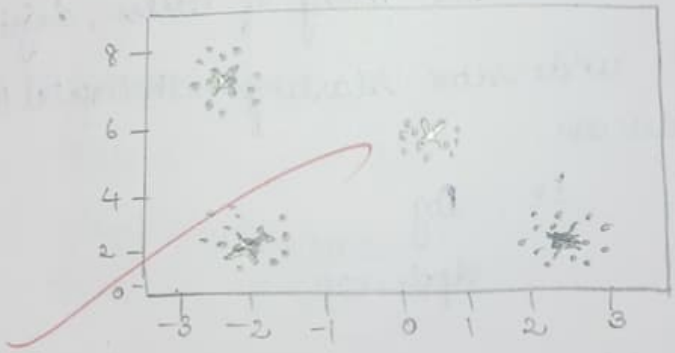
Program :

```
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, clusters_std = 0.60, random_state = 0)
K=3
Kmeans=KMeans(n_clusters =K, random_state=0)
y_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='viridus', label='cluster')
Centers= Kmeans.cluster_centers_
```

```
plt.scatter(centers[:,0], centers[:,1], c='red',
    s=200, alpha=0.75, marker='x',
    label='Centroids')
plt.title('Kmeans Clustering Results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show().
```

220701046





Result

The implementation of K-means clustering is successfully executed.

Ex: NO: 8
Date: 11/10/24

Introduction to Prolog.

220701046

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 paranthesis & separated by commas.

is_bigger(elephant, x)

4. Facts

A fact is a predicate followed by a dot.

bigger_animal(whale).

5. Rules:

A rule of a head (a predicate) & 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 concert doesn't exist
```

Source code:

KB2:

220701046

happy(yolanda).

listen2music(mia).

listen2music(yolanda) :- happy(yolanda).

playsAirGuitar(mia) :- listen2music(mia).

playsAirGuitar(yolanda) :- listen2music(yolanda).

Op:

?- 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).

Op:

?- likes(dan, X)

X = sally

?- married(dan, sally)

true

?- married(john, britney)

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(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)).
```

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 = carcchevy).

220701046

Result:

The prolog is implemented successfully

PROLOG - FAMILY TREE

Ex:no: 9
Date : 18/10/24

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 ( kelvin)

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

parentOf (chris, peter)
parentOf (chris, betty)
parentOf ( helen, peter)
parentOf ( helen, betty)
parentOf ( Kelvin, chris)
parentOf ( Kelvin, lisA)
parentOf ( jeny, 'John)
parentOf ( jeny, helen).
```

/* Rules */

```
father (X, Y) :- male (Y), ParentOf (X, Y)
```

Y	X
peter	chris
peter	helen
john	jeny
chris	kevin

220701046

(b) female(Y), parentOf(X, Y).

Y	X
betty	chris
betty	Helen
lisa	Kelvin
hean	jeny

220701046

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

Y	X	Z
peter	Kevin	chris
peter	jeny	Helen

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

Y	X	Z
betty	Kevin	chris
betty	jeny	helen.

(e) male(Y), father(X, Z), father(Y, W), Z == W.

procedure 'father(A, B)' does not exist.

(f) female(Y), father(X, Z), father(Y, W), Z == W

procedure 'father(A, B)' does not exist.

Result: The prolog family tree has been executed successfully.

MINIMAX Algorithm.

220701046

Ex.no:10

Date : 25/10/24

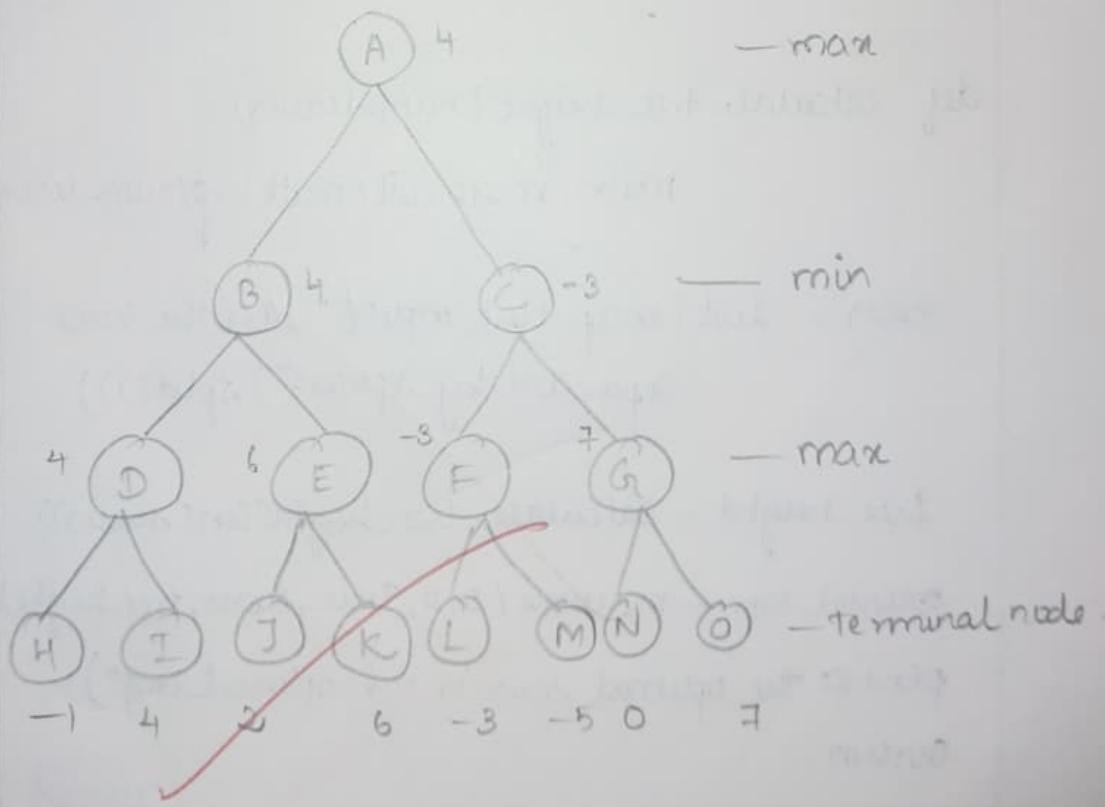
Aim:

⊙ a simple example can be used to explain how the minimax algorithm works. We have included an example below.

⊙ There are 2 players in this scenario, one named Maximizer ~~will strive~~ and other named Minimizer.

⊙ Maximizer will strive for highest possible score & Minimiser will strive for lowest possible score.

⊙ As this algorithm uses DFS we must go all the way to reach the terminal nodes



220701046

```
import math

def minimax(depth, node_index, is_maximiser, scores,
                                        height):
    if depth == height:
        return scores[node_index]
    if is_maximiser:
        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 = list(map(int, input("enter the scores
    separated by spaces:").split()))

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:

```
Enter the terminal values for leaf nodes:
H : 2
I : -1
J : 3
K : 5
L : 8
M : 0
```

N : 6

O : 9

The optimal value of maximizer is 8

Result :

Thus the minimax problem is done and executed successfully.