INDEX

NAME: Bharath Kumar STD: CSE SEC: 'D' ROLL NO: 22951A0563 SUB: Principles of Artificial Intelligence

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1)	01-08-2024	8QUEENS PROBLEM	9	
2)	16-08-2024	DEPTH FIRST SEARCH	9	
3)	23-08-2024	DEPTH FIRST SEARCH - WATER JUG PROBLEM	10	
	~~30-08-2024~~	~~MINIMAX ALGORITHM~~		
4)	30-08-2024	A* SEARCH ALGORITHM	10	
5)	06-09-2024	IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNOLOGIES	10	
6)	27-09-2024	IMPLEMENTING ANN FOR AN APPLICATION USING PYTHON - REGRESSION.	10	
7)	04-10-2024	IMPLEMENTATION OF CLUSTERING TECHNIQUES	10	
8)	18-10-2024	MINIMAX ALGORITHM	10	
9)	25-10-2024	INTRODUCTION TO PROLOG.	10	
10)	08-11-2024	PROLOG FAMILY TREE.	10	

Completed

26/07/2024

10 Python Simple Programs

1) Harshad number:-

```
def checkharshad (n):
    sum=0
    temp=n
    while temp>0:
        sum =sum+ temp %10
        temp = temp //10
    return n %. sum==0
a= int (input ("Enter a number:"))
if (check harshad (a)):
    print ("Yes")
else:
    print ("No")
```

OUTPUT:-

Enter a number 54
Yes

2) CONVERTING A LIST to numarray:

```
import numpy as np
ls = []
print ("Enter the no.of elements of a list")
for i in range
n = int (input("Enter a number of elements of a list"))
print ("Enter the elements of the list:")
for i in range (n):
    a = int (input( ))
    ls.append(a)
arrays= np.array (ls)
print(arrays)
```

OUTPUT:-

Enter number of elements of a list 3
Enter the elements of the list: 7 9 11
[7, 9, 11]

3) CHECK IF A NUMBER IS PRIME:

```
def is_prime(n):
    if (n<=1):
        return false
    for i in range(2, int(n**0.5)+1):
        if (n%i==0):
            return false
    return true
num= int(input("Enter a number: "))
if is_prime(num):
    print(num, "is a prime number.")
else:
    print(num, "is not a prime number")
```

OUTPUT:

Enter a number: 5
5 is a prime number.

---

4) LIST OF SQUARES:

```
def squares_list(n):
    return [i**2 for i in range(n)]
num = int(input("Enter a number: "))
print("List of squares:", squares_list(num))
```

OUTPUT:-

Enter a number: 5
List of squares: 1 4 9 16 25

13/09/2024

8- QUEENS PROBLEM

EX-NO: 1 DATE: 03/08/2024

AIM:

To implement an 8-Queens problem using python

PROGRAM:

```
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
dff 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
    n=int(input("Enter the number of queens"))
    rng= random.Random()
    bd= list(range(n))
    num_found = 0
    tries =0
    result = []
    while num_found <5:
        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(col))
        print(result)
main()
```

RESULT:

Thus, the 8 Queens problem was implemented successfully using backtracking Algorithm.

DEPTH FIRST SEARCH.

EX-NO:- 2

DATE: 18/08/2024

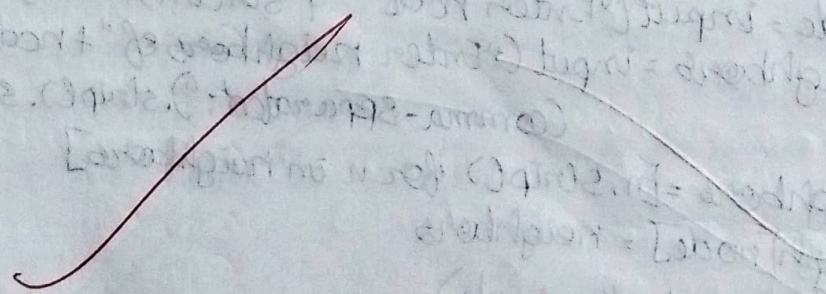
To implement a depth-first search problem using python.

PROGRAM:-

```
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 neember of nodes:"))
graph = {}
for i in range(num_nodes):
    node = input("Enter node " + str(i+1) + ": ").strip()
    neighbors = input("Enter neighbors of " + node +
                      " (comma-separated): ").strip().split(",")
    neighbors = [n.strip() for n in neighbors]
    graph[node] = neighbors
print("Graph:", graph)
start_node = input("Enter starting node:").strip()
print("Starting node:", start_node)
dfs(graph, start_node)
```

OUTPUT:-

```
Enter the number of nodes: 5
Enter node 1: A
Enter neighbors of A (comma-separated): B,C,D
Enter node 2: B
Enter neighbors of B (comma-separated): C,D
Enter node 3: C
Enter neighbors of C (comma-separated): D
Enter node 4: D
Enter neighbors of D (comma-separated):
Enter node 5: E
Enter neighbors of E (comma-separated): F,G
Graph: {'A': ['B', 'C', 'D'], 'B': ['C', 'D'], 'C': ['D'], 'D': [], 'E': ['F', 'G']}
Enter the starting node: A
Starting node: A
A B C D
```



RESULT:-

Thus, a searching technique using Depth-first search algorithm was implemented successfully.

DEPTH FIRST SEARCH - WATER JUG PROBLEM.

EX-NO: 3 DATE: 23-08-2024

AIM:

To implement water-jug problem using depth first search algorithm.

PROGRAM:

```
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_3gallon(x, y, x_max, y_max):
    return (x, 0)
def pour_4_to_3(x, y, x_max, y_max):
    return (x, y_max - y)
def pour_
    transfer = min(x, y_max - y)
    return (x - transfer, y + transfer)
def pour_3_to_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
        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_4to_3(x, y, x_max, y_max),
            pour_3_to_4(x, y, xmax, 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)
(2, 3)
```

RESULT:

Thus, the water jug problem is implemented successfully using depth-first search algorithm.

A* SEARCH ALGORITHM.

EX-NO: 4 DATE: 30-08-2024

AIM:
To implement a searching technique using the A* heuristic algorithm.

PROGRAM:

```
import heapq
class Node:
    def __init__(self, name, parent=None, g=0, h=0):
        self.name = name
        self.parent = parent
        self.g = g
        self.h = h
        self.f = g+h
    def __lt__(self, other):
        return self.f < other.f

def a_star(graph, start, goal, h_values):
    open_list = []
    heapq.heappush(open_list, Node(start, None, 0, h_values[start]))
    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.get(current_node.name, []):
            if neighbor in closed_list:
                continue
            g_new = current_node.g + cost
            h_new = h_values[neighbor]
```

```
            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 = {}
h_values = {}
print("Enter heuristic values for each node. Type 'nil' to stop")
while True:
    node = input("Enter node name (or 'nil' to finish): ")
    if node.lower() == 'nil':
        break
    h_value = int(input(f"Enter heuristic value for {node}: "))
    h_values[node] = h_value
print("Enter edges and their costs: Type 'nil' to stop")
while True:
    node1 = input("Enter the start node (or 'nil' to finish)
    if node1.lower() == 'nil':
        break
    node2 = input("Enter the end node")
    cost = int(input(f"Enter the cost from {node1} to
                                    {node2}: "))
    if node1 not in graph:
        graph[node1] = []
    graph[node1].append((node2, cost))
start_node = input("Enter the start node: ")
python code
goal_node = input("Enter the goal node.
path = a_star(graph, start_node, goal_node, h_values.
if path:
    print(f"Path found: {path}")
else:
    print("No path found")
```

OUTPUT:

Enter edges and their costs.
Enter the start node (or 'nil' to finish): A
Enter the end node: B
Enter the cost from A to B: 2
Enter the start node (or 'nil' to finish): B
Enter the end node: C
Enter the cost from B to C: 1
Enter the start node (or 'nil' to finish): B
Enter the end node: Cr
Enter the cost from B to Cr: 9
Enter the start node (or 'nil' to finish): D
Enter the end node: E
Enter the cost from D to E: 6
Path found: ['A', 'B', 'Cr'].

RESULT:

Thus, a searching technique was implemented successfully using A* heuristic search algorithm.

IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNOLOGIES.

EX NO: 5 DATE: 06-09-2024

AIM:

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

PROGRAM:

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

X, y = make_regression(n_samples=1000, n_features=5,
                       noise=0.1)
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(units=64, activation='relu',
                       input_dim=X_train.shape[1]))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=1, activation='linear'))
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=1)
loss = model.evaluate(X_test, y_test)
print(f"Model Loss: {loss}")
y_pred = model.predict(X_test)
print("Predictions for the first 5 test samples:", y_pred[:5])
```

OUTPUT:-

Epoch 1/5
25/25 ———— 2s 5ms/step - loss: 11171.3555
Epoch 2/5
25/25 ———— 0s 2ms/step - loss: 10910.3125
Epoch 3/5
25/25 ———— 0s 2ms/step - loss: 12157.5947
Epoch 4/5
25/25 ———— 0s 2ms/step - loss: 10762.9805
Epoch 5/5
25/25 ———— 0s 2ms/step - loss: 9546.4512
7/7 ———— 0s 2ms/step - loss: 9276.8145
Model Loss: 9698.2236328125
7/7 ———— 0s 2ms/step
Predictions for the first 5 test samples: [[-2.0036604]
[-6.901868]
[-14.406294]
[4.5987067]
[6.8129045]]

RESULT:-

Thus, we have successfully implemented Artificial Neural Networks for an application using python - regression.

IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNIQUES

EX-NO: 6 DATE: 27-09-2024

AIM:

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

PROGRAM:

```
from sklearn.import tree
X= [[150,50,37],[160,60,38],[170,70,39],[180,80,40],[165,55,36]]
Y= [0,0,1,1,0]
clf=tree.DecisionTreeClassifier()
clf=clf.fit(X,Y)
prediction=clf.predict([[175,75,41]])
print("Predicted Crender (0=Female,1=Male):"prediction[0])
```

OUTPUT:

Predicted Crender (0=Female, 1=Male): 1

RESULT:

Thus, we have successfully implemented a decision tree classifier technique for gender classification using python.

IMPLEMENTATION OF CLUSTERING TECHNIQUES - K-MEANS

EX-NO: 7

DATE: 04-10-2024

AIM:

To implement a k-means clustering technique using python language.

PROGRAM:

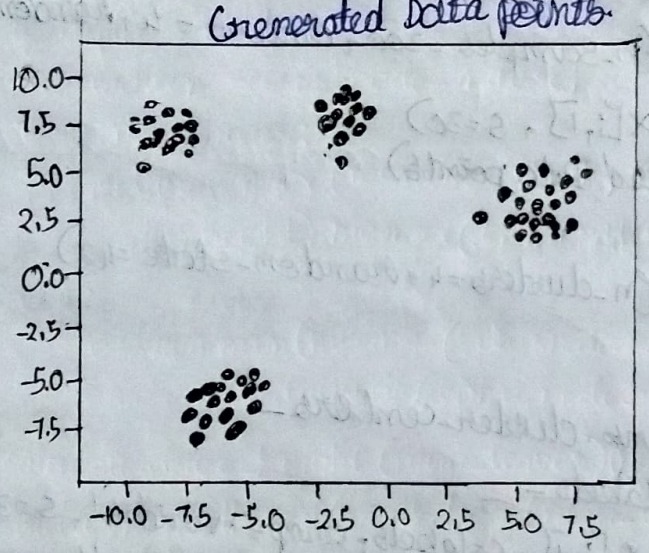
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import kMeans
from sklearn.datasets import make_blobs.
X, _ = make_blobs(n_samples=300, centers = 4, random_state=42)
plt.scatter(X[:,0], X[:,1], s=30)
plt.title("Generated Data points)
plt.show()
kmeans =kMeans(n_clusters=4, random_state=42)
kmeans.fit(X)

centroids = kmeans.cluster_centers_
labels = kmeans.labels_
plt.scatter(X[:,0], X[:,1], c=labels, cmap='viridis', s=30)
plt.scatter(centroids[:,0], centroids[:,1], c='red', marker='X',
                                    s=200, label='centroids')
plt.title("K-means Clustering")
plt.legend()
plt.show()
print(end="\n")
print("Centroids of the clusters:")
print(centroids, end="\n\n")
inertia =[]
```

```
for k in range(1,11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

plt.plot(range(1,11), inertia, marker='o')
plt.title("Elbow Method to find optimal K")
plt.xlabel("Number of clusters (k)")
plt.ylabel("Inertia")
plt.show()
```

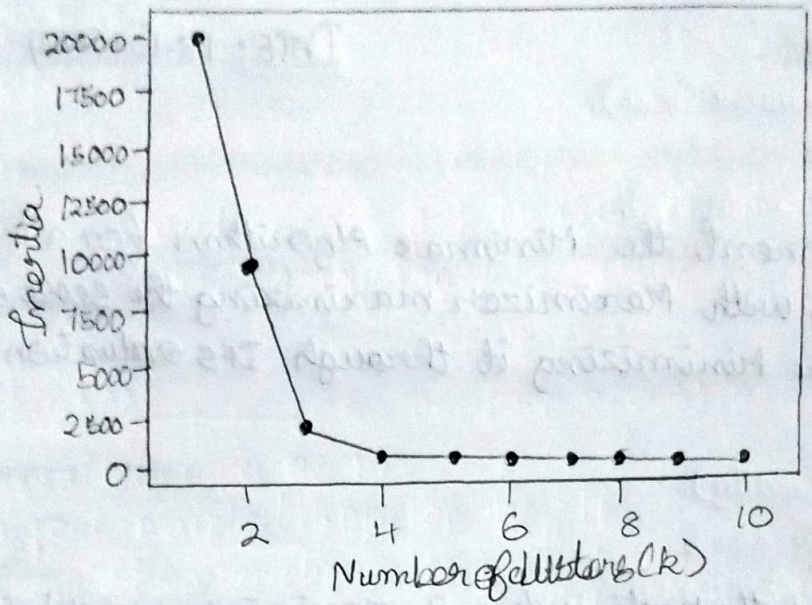
OUTPUT:-



Centroids of the Clusters:

```
[[-2.70781136  8.97143336]
 [-6.83235205 -6.85345748]
 [ 4.71820419  2.04179676]
 [-8.87357218  7.17458342]]
```

ELBOW METHOD TO FIND OPTIMAL K.



RESULT:-

Thus, we have successfully implemented a K-Means clustering technique using python.

MINIMAX ALGORITHM.

EX. NO: 8 DATE: 18-10-2024

AIM:

To implement the Minimax Algorithm for a two-player game, with Maximizer maximizing the score and the Minimizer Minimizing it through DFS evaluation.

~~=>A sim~~

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, False, scores, height),
                minimax(depth+1, node_index*2+1, False, 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(0, 0, True, scores, tree_height)
optimal_score = minimax(0, 0, True, scores, tree_height)
print(f"The optimal score is: {optimal_score}")
```

OUTPUT:

Enter the scores separated by spaces:- -1 4 2 6 -3 -5 0 7
The optimal score is: 4

RESULT:

Thus, the Minimax algorithm successfully determines the optimal moves for both players.

INTRODUCTION TO PROLOG.

EX-NO: 9 DATE: 25-10-2024

AIM:

To learn PROLOG terminologies and write basic programs:

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
?- playsAirGuitar(mia).
        false
?-party.
        true
?- concert
        procedure "concert" does not exist.
```

KB2:

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

OUTPUT:-

```
?- happy(yolanda).
    true
?- listens2music(mia).
    true
?- playsAirGuitar(X).
    mia.
    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)
    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)
    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)

John	X
jack	car(bmw)
john	car(chevy)
olivia	car(civic)
jane	car(chevy)

?- owns(John, _)

John
jack
john
olivia
jane

?- owns(Who, car(chevy))

Who
john
jane

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

false

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

car(chevy)

RESULT:

Thus, we have written basic programs to learn prolog terminologies.

PROLOG FAMILY TREE

EX. NO: 10 DATE: 08-11-2024

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(lisa).
female(helen).

parentof(chris,peter).
parentof(chris,betty).
parentof(helen,peter).
parentof(helen,betty).
parentof(kevin,chris).
parentof(kevin,lisa).
parentof(jeny,john).

/*RULES::*/
/* son,parent
   son,grandparent*/
father(X,Y):-male(X),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):-female(X),parentof(X,Z),parentof(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
```

OUTPUT:

- ?- male(Y), parent of(X, Y)

Y	X
peter	chris
peter	helen
john	jeny
chris	kevin

?- female(Y), parentof(X, Y)

Y	X
betty	chris
betty	helen
lisa	kevin
helen	jeny

?- male(Y), parentof(X, Z), parentof(Z, Y)

Y	X	Z
peter	kevin	chris
peter	jeny	helen

?- female(Y), parent(X, Z), parentof(Z, Y)

Y	X	Z
betty	kevin	chris
betty	jeny	helen

?- male(Y), father(X, Z), father(Y, W), Z == W
procedure 'father(A, B)' does not exist

?- female(Y), father(X, Z), father(Y, W), Z == W
procedure 'father(A, B)' does not exist

RESULT:-

Thus, we have developed a family tree program using PROLOG with all possible facts, rules and queries.