INDEX POAI

NAME: S.AJITNASH STD.: _____ SEC.: _____ ROLL NO.: 034 SUB.: _____

S. No.	Date	Title	Page No. Marks	Teacher's Sign / Remarks
1.	22/7/24	Basic python programs	9	
2.	26/7/24	Documentation	9	
3.	29/7/24	Depth first Search	10	
4.	6/7/24	A* Search	10	
5	9/8/24	8-Queens problem	10	
6	30/8/24	Minimax algorithm	10	
7	25/10/24	Introduction to prolog	10	
8	08/11/24	PROLOG Family tree	10	
9	18/10/24	Implementation of Neural Network for an application using python - Regression	10	
10	27/9/24	implementation of decision tree classifier techniques	10	
11	04/10/24	Implementation of clustering techniques Mean	10	

completed

```
def reconstruct_path(came_from, current):
    path = [current]
    while current in came_from:
        current = came_from[current]
        path.append(current)
    return path[::-1]

def heuristic(a, b):
    return abs(a[0]-b[0]) + abs(a[1]-b[1])

def distance(a, b):
    return ((a[0]-b[0])**2 + (a[1]-b[1])**2)**0.5

def neighbours(node):
    x, y = node
    return [(x+1, y), (x-1, y), (x, y+1),
            (x, y-1)]

start = (0, 0)
goal = (5, 5)
print(path)
```

OUTPUT

```
[(0,0), (0,1), (0,2), (0,3), (0,4), (0,5), (1,5)
(2,3) (5,5)]
```

RESULT: The the program for A* search has been executed successfully.

9/08/24

EXNO:1 **8-QUEENS PROBLEM**

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

def 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 neg_clashes 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 queen: 8
Found solution {1,6,4,7,0,3,5,2} in 94 tries.
Found solution {3,0,4,7,5,2,6,1} in 175 tries.
Found solution {2,0,6,4,7,1,3,5} in 196 tries.
Found solution {5,0,4,1,7,2,6,3} in 439 tries.
Found solution {6,2,7,1,4,0,5,3} in 156 tries.
{{1,6,4,7,0,3,5,2}, {3,0,4,7,5,2,6,1}, {2,0,6,4,7,1
3,5}, {5,0,4,1,7,2,6,3}, {6,2,7,1,4,0,5,3}}
```

RESULT :-

Thus the 8 queens problem was implemented successfully using backtracking algorithm.

Ex No: 2 DEPTH FIRST SEARCH 16/08/24

AIM:-

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 number 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 the 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~~:

Th

RESULT:

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

EX NO: 4 MINIMAX ALGORITHM 30/8/24

AIM:

To implement the minimax algorithm for a two-player game, with maximizer maximizing the score and minimizer minimizing it through DFS evaluation.

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)

else:
    return min ( minimax (depth +1, node - index * 2, true
                scores, height)
        minimax (depth +1, node - index * 2 + 1, true, scores,
                height)
    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("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 players by evaluating the game-tree and selecting the best possible scores for maximizer and minimizer.

Exno:6 INTRODUCTION TO PROLOG 25/10/24

INTRODUCTION TO PROLOG

AIM:

To learn PROLOG terminologies and write basic programs.

TERMINOLOGIES:

1. Atomic Terms:

Atomic terms are usually strings made up of lower- and uppercase letters, digits, and the underscore, starting with lowercase letter.

Ex:

dog

ab_c_321

2. variables:

variables 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 PROLOG atom and a number of arguments PROLOG terms, i.e atoms, numbers, variables, or other compound terms)

Ex:

```
is- bigger (elephant, X)
f(g(X--) 7)
```

4. Facts:

A fact is a predicate followed by a dot.

Ex:

```
bigger- animal (whale)
like- is- beautiful
```

5. Rules:

A rule consists of a head and a body

Ex:

```
is. smaller (X,Y) :- is bigger (Y, X)
aunt (Aunt, Child) :- sister (Aunt, parent)
                      (parent, child)
```

SOURCE CODE

KB1:

```
woman (mia)
woman (jody)
woman (yolanda)
plays Air Guitar (jody)

party
query 1: ?- woman
query 2: ? plays
query 3: ? party
query 4: ? concat
```

KB2:

```
happy
listens 2 music
listens 2 music - happy
plays Air Guitar - listen 2 music
plays Air Guitar - listens 2 music
```

KB3:

```
likes (dan, Sally)
likes (sally, dan)
likes (john, britney)
married (X,Y) :- like (X,Y) likes (Y,X)
```

friends (x,N) :- likes (x,N) like (Y,X)

KB4:

food (burger)

food (sandwich)

food (pizza)

lunch (sandwich)

dinner (pizza)

meal (x) :- food (x)

KB5:

owns (jack, car (bmw))

owns (john, car (chevy))

owns (olivia, car, (civic))

owns (jane, car (chevy))

sedan car (bmw)

truck car (chevy)

RESULT:

Thus we have written basic programs to learn prolog terminologies.

EX NO: 7

8/11/24

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 (john)

male (chris)

female (betty)

female (jeny)

female (lisa)

parent of (chris, peter)

parent of (chris, betty)

parent of (peter, betty)

parent of (kevin, lisa)

parent of (jeny, john)

parent of (jeny, pete))

RULES:

son. parent

son. grand parent

father (x,y) :- male (y) parent of (x,y)

mother (x,y) :- female (y) parent of (x,y)

grand mother (x,y) :- female (y) parent of (x,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

RESULT:

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

Ex.No: 8

18/10/24

IMPLEMENTING ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION USING PYTHON - REGRESSION.

AIM:

To implement artifical neural networks for an application in Regression using Python.

PROGRAM:

```
import numpy as np
from keras.models import Sequential
from sklearn.datasets import make_regression
x, y = make_regression (n_samples=1000)
x_train, x_test y_train y_test= train_test_split
(x,y, test_state=42)
x_train = scalar.fit_transform (x_train)
x_test = scalar.transform (x_test)
model = Sequential()
model.compile (optimizer = 'adam', loss='mean_
squared_error')
```

```
model.fit(x_train, y_train, epochs=5,
          batch_size=32, verbose=1)
loss = model.evaluate(x_test, y_test)
print("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 1ms/step - loss: 15025.5996
Epoch 2/5
25/25 ———— 0s 1ms/step - loss: 14661.4590
Epoch 3/5
25/25 ———— 0s 2ms/step - loss: 12759.1582
Epoch 4/5
Epoch 5/5
7/7 ———— 0s 7ms/step
Model Loss: 9905.695312
7/7
Predictions for the first 5 test samples
: [[-14.904316]
 [ 9.110759]
 [-11.914794]
 [-11.280347]
 [-1.4355274]]
```

RESULT :-

Thus, we have successfully implemented artificial neural networks for an application in regression using python.

Ex. No 9

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 decission tree from tree
- Assign value for x and y
- Call the function predict for given random values for each given feature.
- display the output.

PROGRAM:

```
from sklearn import tree
x = {[150, 50, 37], [160, 60, 38], [170, 70, 39]
     [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 Gender(0 = Female, 1 = Male):"
       prediction[0])
```

OUTPUT:

predicted Gender(0) = Female, 1 = Male): 1

RESULT:

Thus, we have successfully implemented a decission tree classification techniques for gender classification.

EX NO: 9

04/10/24

IMPLEMENTATION OF CLUSTERING TECHNIQUES. K-MEANS

AIM:-

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

EXPLANATION:

- Import k means from sklearn.cluster.
- Assign x and y.
- Call the function k means().
- Perform scatter operation and display the output.

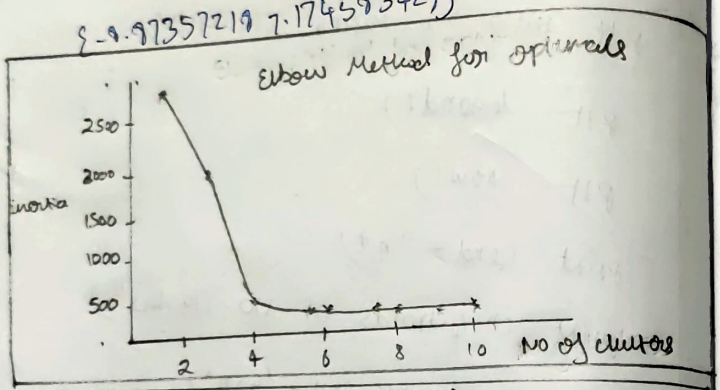
PROGRAM:

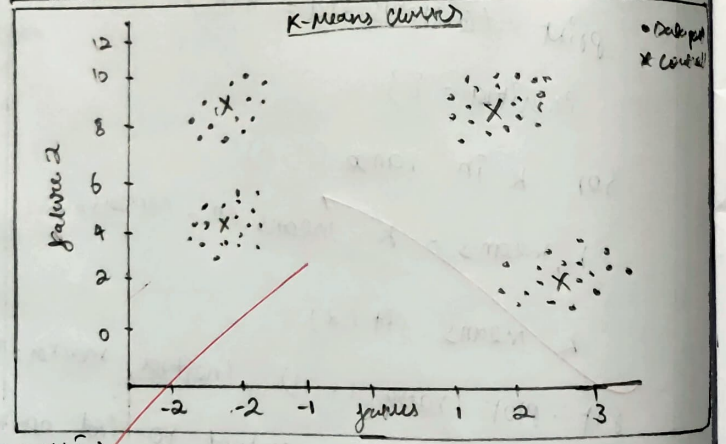
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import k means
from sklearn.datasets import make_blobs
plt.title ("Generated data points")
plt.show()
k means = k means (n_clusters=4, random_state=42)
labels = k means.labels
plt.scatter(x[:0], x[:1], c= labels, cmap=viridis
            s=30)
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):
  k means = k means (n_clusters = k
  k means.fit(x)
plt.plot (range (1,11), Inertia, marker="o")
plt.title ("Elbow method to find optimal k")
plt.ylable ("Intertia")
plt.show()
```

OUTPUT :-

Centroids of the clusters :

[[-2.70991136 0.97143336]
[-6.93232051 -6.93643548]
[4.7192049 2.04179676]
[-0.97357219 7.17459342]]





RESULT :-

Thus we have successfully implemented a K-means clustering technique using Python language.