8- QUEENS PROBLEM.

CODE:

```
N=8, N=4
def SolveQueens(board, col):
if col == N
    Print(Board):
    return True.
for i in range(N):
    if isSafe(board, i, col):
        board[i][col] = 1
    if SolveNQueens(board, col+1):
    board[i][col] = 0
return False.
def isSafe(board, row, col):
for i in range(col):
if board[row][x] == 1:
    return False
for y,x in zip(range(row, -1, -1), range(col, -1,
if board[x][y] == 1:
    return false
for[x][y] == 1
for y,x in zip(range(row, N, 1), range(col, -1,
    if board[x][y] == 1:
        return False.
```

```
board = [[0 for x in range(N)] for y
    in range(N)]
if not solveNQueens(board, 0):
    print("No solution found")
```

OUTPUT:

```
[[1,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,1,0],
 [0,0,0,0,1,0,0,0],
 [0,0,0,0,0,0,0,1],
 [0,1,0,0,0,0,0,0],
 [0,0,0,1,0,0,0,0],
 [0,0,0,0,0,1,0,0],
 [0,0,1,0,0,0,0,0]].
```

```
[[1,0,0,0],
 [0,0,1,0],
 [0,0,0,1],
 [0,1,0,0]]
```

RESULT: Thus given 8-queen Problem has been executed successfully.

DFS - DEPTH FIRST SEARCH

16/8/24

CODE:

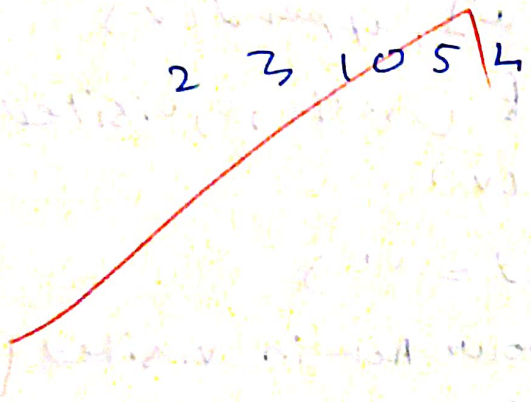
```
from collections import defaultdict
class Graph:
    def __init__(self):
        self.graph = defaultdict(list)
    def addEdge(self, u, v):
        self.graph[u].append(v)
    def DFSUtil(self, v, visited):
        visited.add(v)
        print(v, end=' ')
        for neighbour not in visited
            self.DFSUtil(neighbour, visited)
    def DFSC self, v):
            visited = set()
            self.DFSUtil(v, visited)
if __name__ == "__main__":
    g = Graph()
    g.addEdge(0,5)
    g.addEdge(0,4)
    g.addEdge(4,3)
    g.addEdge(2,3)
    g.addEdge(2,0)
    g.addEdge(3,1)
```

Print ("Following is Depth First
TRAVERSAL (Starting from vertex 2)")

g.DFS(2)

OUTPUT:

Following is Depth First Traversal
(Starting from vertex 2)



RESULT: Thus the given DFS problem
has been executed successfully.

A* ALGORITHM.

26/8/24

CODE :

```
import math
import heapq

class cell:
    def __init__(self):
        self.parent_i = 0
        self.parent_j = 0
        self.f = float('inf')
        self.g = float('inf')
        self.h = 0

ROW = 9
COL = 10

def is_valid(row, col):
    return (row >= 0) and (row < ROW) and (col >= 0)
and (col < COL)

def is_unblocked(grid, row, col):
    return grid[row][col] == 1

def is_destination(row, col, dest):
    return ((row - dest[0]) ** 2 + (col - dest[1]) ** 2) **
0.5

def tracepath(cell_details, dest):
    Print("The path is")
    Path = []
    row = dest[0]
    col = dest[1]
```

```
While not (cell_details[row][col].
    Parent_i == row and cell_details[row][col]
  Parent_j == col);

Path.append((row, col))

~~Path reverse~~
temp_row = cell_details[row][col].parent_i
temp_col = cell_details[row][col].parent_j
  row = temp_row
  col = temp_col
Path.append((row, col))
Path.reverse()
for i in path:
    Print("->", i, end=" ")
  Print()

def a_star_~~app~~search(grid, src, dest):
if not is_valid(src[0], src[1]) or not is_unblocked
    (grid, dest[0], dest[1]):
Print("We are already at destination")
 return
closed_list = [[false for _ in range(col)] for
_ in range(row)]
cell_details = [[cell() for _ in range(col)] for
  _ in range(row)]
 i = src[0]
 j = src[1]
 cell_details[i][j].f = 0
 cell_details[i][j].g = 0
 cell_details[i][j].h = 0
 cell_details[i][j].parent_i = i
```

```
cell_details[i][j].parent_j = j

open_list = []
heapq.heappush(open_list, (0.0, i, j))
found_dest = False

while len(open_list) > 0:
    p = heapq.heappop(open_list)
    i = p[1]
    j = p[2]
    closed_list[i][j] = True.

    directions = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (1, -1),
                  (-1, -1)]

    for dir in directions:
        new_i = i + dir[0]
        new_j = j + dir[1]
if is_valid(new_i, new_j) and is_unblocked
(grid, new_i, new_j) and not closed_list
[new_i][new_j]:
    if is_destination(new_i, new_j, dest):
        cell_details[new_i][new_j].parent_i = i
        cell_details[new_i][new_j].parent_j = j
    print("The destination cell is found")

    trace_path(cell_details, dest)
    found_dest = True
    return
else
    g_new = cell_details[i][j].g + 1.0
    h_new = calculate_h_value(new_i, new_j, dest)
    f_new = g_new + h_new
```

```
if cell_details[new_i][new_j].f == float('inf')
    or cell_details[new_i][new_j].f > f_new:
    heapq.heappush(open_list, (f_new, new_i, new_j))

    cell_details[new_i][new_j].f = f_new
    cell_details[new_i][new_j].g = g_new
    cell_details[new_i][new_j].h = h_new
    cell_details[new_i][new_j].parent_i = i
    cell_details[new_i][new_j].parent_j = j

if not found_dest:
    Print("Failed to find destination cell")

def main():
    grid = [
    [1, 0, 1, 1, 1, 1, 0, 1, 1, 1],
    [1, 1, 1, 0, 1, 1, 1, 0, 1, 1],
    [1, 1, 1, 0, 1, 1, 0, 1, 0, 1],
    [0, 0, 1, 0, 1, 0, 0, 0, 0, 1],
    [1, 1, 1, 0, 1, 1, 1, 0, 1, 0],
    [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
    [1, 0, 1, 1, 1, 1, 0, 1, 1, 1],
    [1, 1, 1, 0, 0, 0, 1, 0, 0, 1]]

    src = [8, 0]
    dest = [0, 0]

    a_star_search(grid, src, dest)

if __name__ == "__main__":
    main()
```

OUTPUT:

Destination cell is found

The path is

→(8,0) → (7,0) → (6,0) →(5,0) →

(4,1) → (3,2) →(2,1) →(1,0)→(0,0)

RESULT:

Thus the given A* Algorithm has been executed Successfully.

WATER JUG PROBLEM : 30/8/24

```
def waterjugProblem(capacity-jug1, capacity-jug2,
                    desired-qty):
    stack=[]
    stack.append(((0,0),[]))
    visited = set((0,0))
    while stack:
        current_state, path = stack.pop()
if current_state[0]==desired_quantity or current_st
[1]==desired_quantity:
    return current=state, path+[current_state]
    next_ states = generate NextStates(current_state, capacity
    jug1, capacity_jug2)
    for next_state in next_states:
        if next-state not in visited:
            stack.append((next_state, path+[current_state
            visited.add(next_state)
    return "No solution found"
    def generate next states(state, capacity-jug1, capacity
        jug2)):
        next_states ~~append~~ = []
        next_states.append((capacity-jug1, state[1]))
        next_states.append((state[0], capacity-jug2))
        next_states.append((0, state[1]))
        next_states.append((state[0], 0))
```

~~Pour ammount from j =~~

Pour_ammount = min [state [0], capacity - jug2 - state[

next_ states . append ((state [0] - pour_ammount, state[1]

+ pour_amount))

AO* Algorithm. 6/9/24

Aim: To implement AO* Algorithm.

Program:

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

def ao_star(grid, start, goals):
directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
open_list = [(0 + min(heuristic(start, goal) for goal
in goals), 0]
g_cost = {start: 0}, came_from = {}
while open_list:
    if any(current == goal for goal in goal
    path = []
while current in came_from:
    path.append(current)
    current = came_from[current]
    return path + [start][::-1]
for dx, dy in directions:
    neighbour = (current[0] + dx, current[
    if 0 <= neighbour[0] < len(grid) and 0
        neighbour[1])
    tentative_g = current_g + 1
        g = g_cost[neighbour] = tentative
    heapq.heappush(open_list, (g + heuristic
                               neigh
```

return None

OUTPUT:

expanding : A

Best path for A : ['B', 'I']

grid = [[0, 0, 0, 0, 0],
[0, 1, 1, 1, 0],
[0, 1, 0, 0, 0],
[0, 1, 0, 0, 0]
[0, 0, 0, 0, 0]]

start = (0, 0)

goals = [(4, 4), (2, 2)]

path = ao - star (grid, start, goals)

Print ("path found :", path)

path found : [(2, 2), (2, 1), (2, 0), (1, 0), (0, 0)]

RESULT:

Thus the AO* Algorithm has been executed successfully.

Aim:

To implement a decision tree classification technique for gender classification using Python

Program:

```
from sklearn.tree import DecisionTreeClassifier
import numpy as np
X = np.array([[181,80,44],
              [177,70,43],
              [160,60,38],
              [154,54,37],
              [166,65,40],
              [190,90,47],
              [175,64,39],
              [177,70,40],
              [159,55,37],
              [171,75,42],
              [181,85,43]])
Y = np.array(['male','male','female','female','male','male','female','female','female','male','male'])
clf = DecisionTreeClassifier()
clf = clf.fit(X,Y)
prediction = clf.predict([[190,70,43]])
print("Predicted gender: ", "male" if prediction[0] == 1 else "female")
```

OUTPUT: Predicted Gender: male

RESULT: Thus the decision tree classification technique has been executed successfully

4/10/21

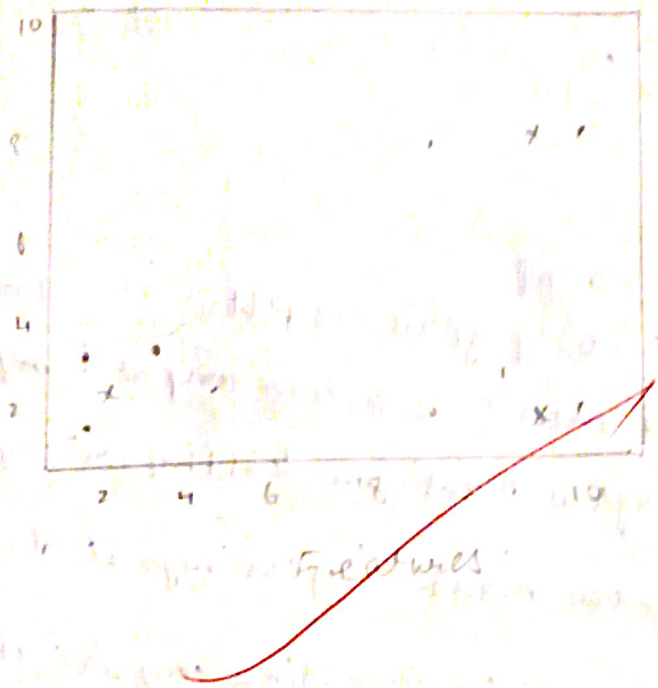
Implementing of Clusting technique k-means

Aim: To implement a k mean clustering techingue using Python language.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import kmeans
x = np.array([[1,2],[1.5,1.8],[5,6],[8,8],
    [1, 0.6],[9,11],[8,12],[10,2],[9,3]])
kmeans = kmeans(n_clusters=3)
centroids = kmeans.clusters_centers_
labels = kmeans.labels_
plt.scatter(x[:,0],x[:,1],c=labels,
    cmap='viridis')
plt.title('k-means clustering')
plt.xlabel('Feature1')
plt.ylabel('Feature 2')
plt.legend()
plt.show
```

OUTPUT :



RESULT :

Thus, implementing of Clustering technique K-means executed successfully

AIM: TO implementing Artificial neutral for an application in regression using Python

18/10/24

PROGRAM:

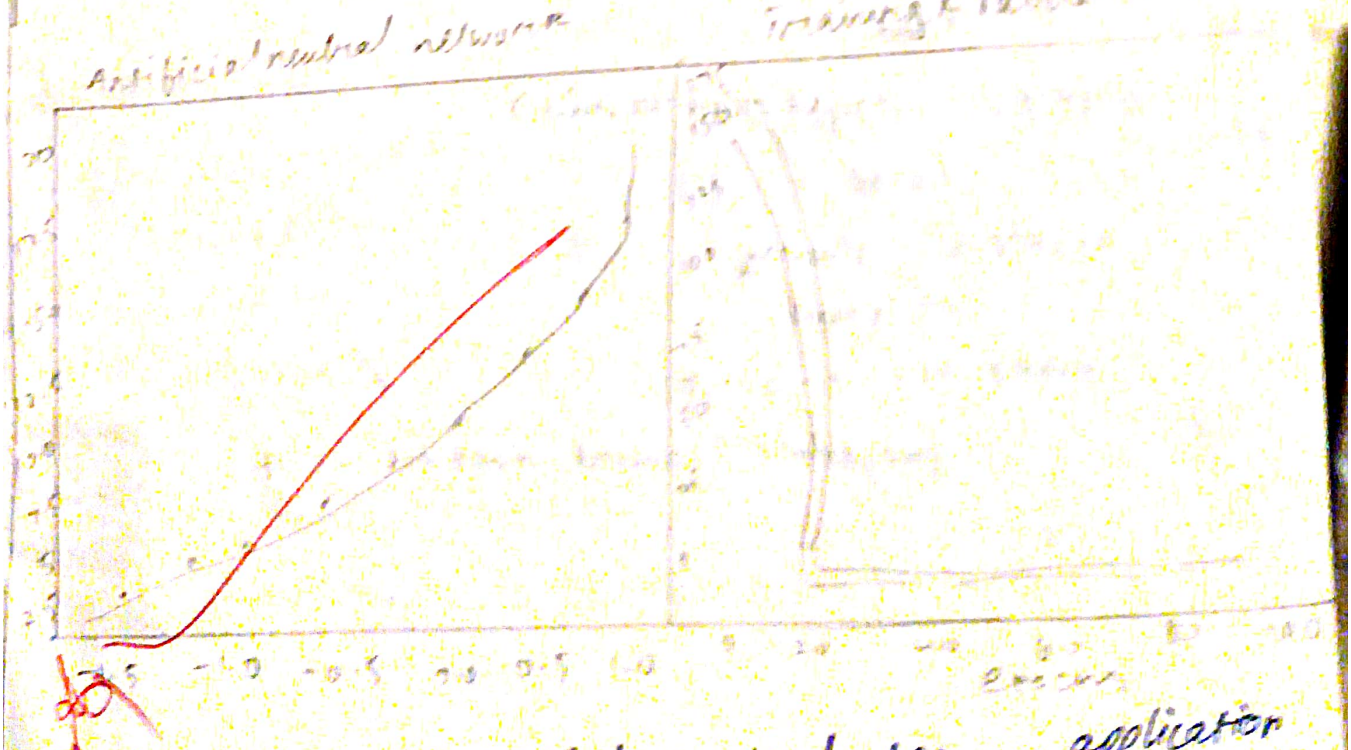
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import
from tensorflow.keras.models import
from tensorflow.keras.layers import dense
from sklearn.preprocessing import standa
scales.
np.random.seed(42)
x = np.linspace(0,10,100)
y = 2*x+1+np.random.normal(0,1
x = x.reshape(-1,1)
x_train, x_test, y_train, y_test = train_te
    -split(x, y, test size=0.2, random
                                = 42)
scaler = standard scaler()
x_train = scales.fit_transform(x_tra
x_test = scales.transform(x_test
model = sequential()
model.add(Dense(units=64, activation
                = 'relu', input)
model.add(dense(units=1))
```

history = model.fit(x_train, y_train epochs=100)

~~Plt~~

```
Plt.scatter(x_test, y_test, color='blue', label=
    'true values')
Plt.plot(x_test, y_pred, color='red', linewidth
                                    =2)
Plt.xlabel('x')
Plt.ylabel('y')
Plt.legend()
Plt.show()
Plt.plot(history.history['val_loss'], label=
'validation loss')
Plt.xlabel('epoche')
Plt.legend()
Plt.show()
```

OUTPUT:



RESULT: Thus artificial neutral for an application using Python is implemented and

INTRODUCTION TO PROLOG

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:

```
QUERY 1: woman(mia)
         True

QUERY 2: playsAirGuitar(mia)
         False

QUERY 3: Party
         True

QUERY 4: concert
         procedure 'concert' does not exist.
```

KB2

happy(yolanda)

listens2music(mia).

Query:

	OUTPUT
Listens2music(yolanda):	False
happy(yolanda):	True
playsAirGuitar(mia):	False
listens2music(mia):	True
playsairguitar(yolanda):	Yolanda = jonny
listens2music(yolanda):	False

KB3:

likes(dan, sally).

likes(sally, dan).

likes(john, britney).

Query:

married(x,y).

liked(x,y).

likes(y,x).

friends(x,y).

likes(x,y).

likes(y,x).

OUTPUT:

do not exist

x = dan y = sally true

y true

does not exist

true

y true

KB 4:

food (Burger).
food (Sandwich).
food (Pizza).
~~foo~~
lunch (sandwich).
dinner (pizza).

Query:

meal(x).

food(x).

OUTPUT

does not exist.

x = burger, true

KB 5:

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)).

RESULT:

Thus, the introduction to prolog has been executed succe

FAMILY TREE

8/4/24

AIM:

To write a program for prolog family tree.

SOURCE CODE:

```
(* Facts :: *)
male(peter)
male(john)
male(chris)
male(kevin)
female(betty)
female(jeny)
female(lisa)
female(helen)
parent of (chris, peter)
parent of (chris, betty)
parent of (helen, peter)
parent of (helen, betty)
parent of (kevin, chris)
parent of (kevin, betty)
parent of (jeny, John)
parent of (jeny, helen)
/* Rules: */
/* son. parent
```

OUTPUT :

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

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

grandfather (x, y) :- male (y), parent of (x, z), parent of (z, y)

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

RESULT :

Thus the Program has been executed successfully.