Exp No: 01
Date:

N - Queen's Problem

AIM:

To write a python program for n queens problem.

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

import random
rng = random.Random()
n = int(input("Enter the Value for n:"))
bd = list(range(n))
num_found = 0
tries = 0
result = []
```

```
while num_found <10:
    rng.shuffle(bd)
    tries =0
    num_found+=1
    result.append(list(bd))
print(result)
```

Output:

Enter the Value of n: 4

Found

```
while num_found <5:
    rng.shuffle(bd)
    tries+=1
    if not has_clashed(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)
```

Output:

Enter the n Value :4

Found Solution [0,2,3,1] in 1 tries.

Found Solution [3,1,2,0] in 1 tries.

Found Solution [2,0,3,1] in 2 tries.

Found Solution [1,3,0,2] in 2 tries.

Found Solution [0,3,1,2] in 1 tries.

[[0,2,3,1], [3,1,2,0], [2,0,3,1][1,3,0,2], [0,3,1,2]]

Exp: 02

Depth First Search Algorithm

AIM:

To write a python program for DFS problem.

PROGRAM:

```
def dfs(graph, start, visited = None):
    if visited is None:
        visited = set()
        visited.add(start)
        print(start, end=" ")
        for neighbour in graph[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()
    neighbours = input("Enter neighbours of " + node + " (comma-separated): ").strip().split(',')
    neighbours = [n.strip() for n in neighbours]
    graph[node] = neighbours
print("graph:", graph)
start_node = input("Enter the starting node: ").strip()
print("strating 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
Enter node 2: B
Enter neighbors of B (Comma-Separated): D
Enter node 3: C
Enter neighbours of C (Comma-Separated): D, E
Enter node 4: D
Enter neighbours of D (Comma-Separated):
Enter node 5: E
Enter neighbours of E (Comma-Separated):
graph: {'A': ['B', 'C'], 'B': ['D'], 'C': ['D', 'E'], 'D': [''], 'E': ['']}

Enter the Starting node: A
Starting node: A
ABD CE

RESULT:

Thus the python program for n queens problem has been verified and executed Successfully.

Exp: 03

A* Algorithm

AIM:

To write a Python program for A* Algorithm.

PROGRAM:

```
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, 0, 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.g + 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', 3)],
    'B': [('A',1), ('D',1), ('E',3)],
    'C': [('A',3) ('F',2) ]
    'D': [('B',1)],
    'E': [('B',3) ('F',1)],
    'F': [('C',2), ('E',1)]
}
```

```
def heuristic (node, goal):
    return 0
Start-node = 'A'
goal-node = 'F'
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', 'C', 'F']

RESULT:

Thus the python program for A* Algorithm has been verified and executed successfully.

Exp:04

DFS - WATER JUG PROBLEM

AIM:

To write a python program for solving water jug problem using DFS

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 fill_4_gallon (x, y, x_max, y_max):
    return (0, y)
def fill_3_gallon (x, y, x_max, y_max):
    return (x, 0)
def pour_4_to_3 (x, y, x_max, y_max):
    transfer = min (x, y_max - y)
    return (x - transfer, y + transfer)
def pour_3_to_4 (x, y, x_max, y_max):
    return (x + transfer, y - transfer)
```

```
def dfs_water_jug(x_max, y_max, goalx, 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

    return None

x_max = int(input("Enter max capacity of 4:"))
y_max = int(input("Enter max capacity of 3:"))
goal_x = int(input("Enter the goal amount
                    for the 4-gallon jug:"))

result = dfs_water_jug(x_max, y_max, goal_x)
if result is None:
    print("No Solution found.")
```

OUTPUT:

Enter the max capacity of 4 : 4
Enter the max capacity of 3 : 3
Enter the goal amount of the 4-gallon jug : 2

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)
~~Visiting State : (2,3)~~
Goal Reached : (2,3)

RESULT:

Thus the above python program to perform watering problem is been executed and verified successfully.

Exp: 05

MIN-MAX 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))
        minimax(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.log 2.(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,treehigh)
print ("The optimal Score is {optimal_Score}")
```

Output:

The optimal Score is : 5

RESULT:

Thus the MinMax problem has been executed Successfully.

Exp: 06

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 classifier() from tree
* Assign values for x and y
* Call the function predict for Predicting on the basis of given random values for each given feature.
* Display the output.

PROGRAM:

```
from Sklearn.tree Import Decisiontree classifier
x = [
    [170, 65, 42],
    [160, 55, 38],
    [175, 70, 43],
    [155, 50, 37],
    [165, 60, 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 Shoe Size:"))
    Sample_data = [[height, weight, Shoe_Size]]
    predict = clf.predict(sample_data)
    gender = "Male" if prediction[0] == 1 else "Female"
    print(f"the predict gender for the input
        {sample_data[0]} is:{gender}")
except ValueError:
    print("Please enter Valid number for
    height, weight and Shoe_Size!")
```

Output:

Enter height in cm: 170

Enter weight in kg: 63

Enter Shoe Size: 41

The predicted gender for the input [170, 63, 41] is Male.

Result:

This the python program to implement the decision tree classification technique for gender Classification was executed Successfully.

Exp: 07

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 Scatter operation and display the output.

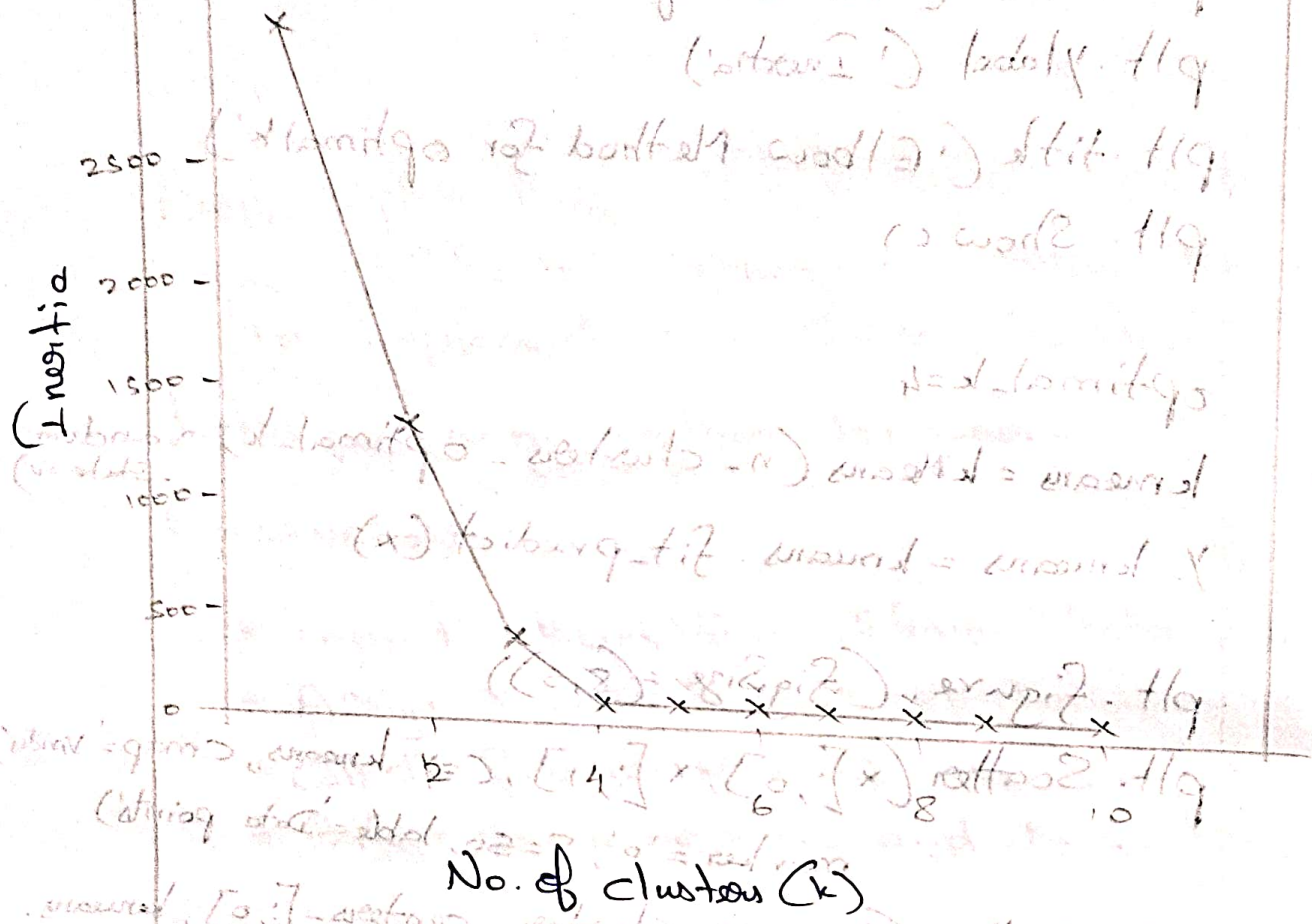
PROGRAM:-

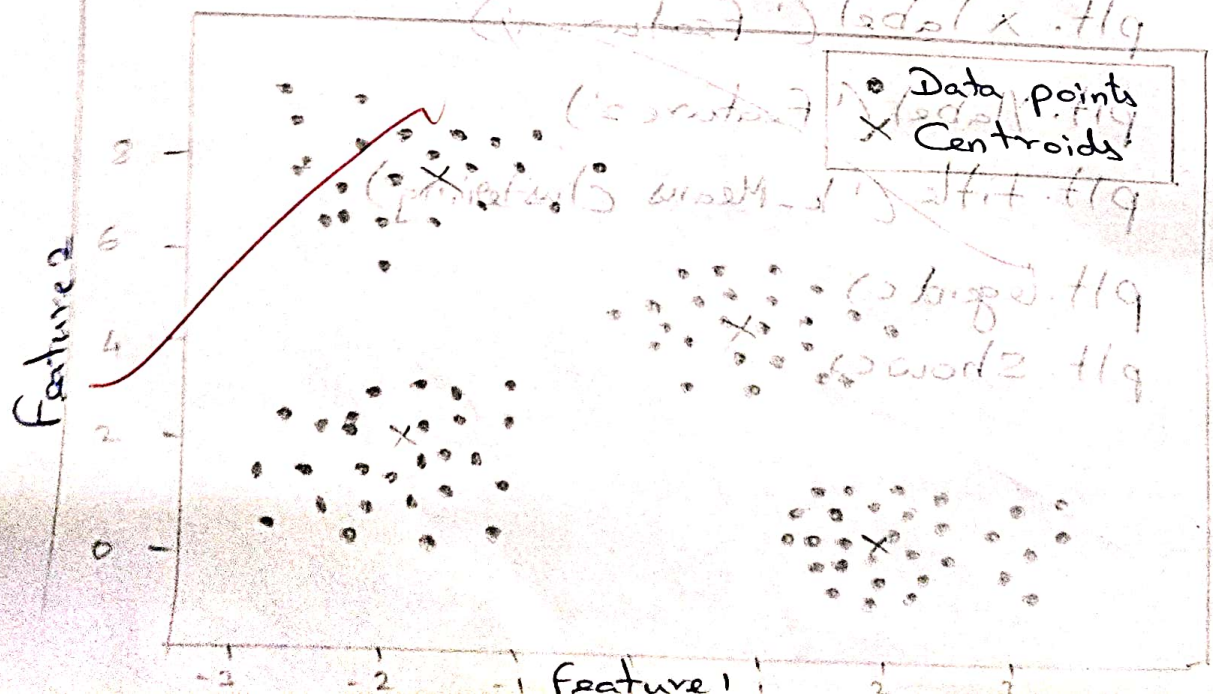
```
import numpy as np
import matplotlib.pyplot as plt
from Sklearn.cluster import kMeans
from Sklearn.dataset import make_blobs
X, _ = make_blobs(n_Samples = 300, Centers = 4, cluster_
                std = 0.6,
                random_State = 0)
inertia = []
k_range = range(1,11)
for k in krange:
    kmeans = kMeans(n_clusters=k, random_State = 0)
    kmeans.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
kmeans = kMeans(n_clusters_optimal_k, random
                                -state=0)
y_kmeans = kmeans.fit_predict(x)

plt.figure(figSize=(8,5))
plt.Scatter(x[:,0], x[:,1], c=y kmeans, cmap='viridis',
            marker='o', s=50, lable='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





RESULT:-

Thus the python program to Implementation of clustering-Techniques k-means was executed and Verified Successfully.

Exp: 08

Implementing Artificial Neural Network for An Application Using Python-Regression

AIM:

To implementing artificial neural network for an application in Regression using Python.

PROGRAM:-

```
import numpy as np
from Sklearn.model_Selection import train_test_split
from Sklearn.datasets import make_regression
from Sklearn.preprocessing import StandardScalar
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

X,Y = make_regression (n_Samples=1000, n_features=10,
noise=10.1, random_State=42)
Y=Y.reshap (-1, 1)

Scaler_X= 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 = Sequenotial()
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(optimized = 'adam', loss = mean_Squared_
                                error,
                                metrics = [mae]

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

loss, mae = model.evaluate(x_test, y_test, Veribose = 0)
print(f" Mean Absolute Error on test Set : {mae}")

y_pred = model.predict(x_test)
y_pred = scaler_y.inverse_transform(y_test)

print(" Predicted values : " y_pred[:5].flatten())
print(" Actual values : ", y_test[:5].flatten())
```

Output:-

Epoch 1/100
20/20 ——— 3s 35ms/step - loss: 0.8451 - mae: 0.7237
- val_loss: 0.4083 - val_mae: 0.4713
Epoch 2/200
20/20 ——— 1s 8ms/step - loss: 0.3895 - mae: 0.4677 - val_loss: 0.1572
- val_mae: 0.2842
Epoch 3/100
20/20 ——— 0s 5ms/step - loss: 0.1890 - mae: 0.2949 - val_loss: 0.0400
- val_mae: 0.2265

⋮

Epoch 100/100
20/20 ——— 0s 9ms/step - loss: 2.5293e-04 - mae: 0.0124 -
val_loss: 0.0073 - val_mae: 0.0875
Mean Absolute Error on Test Set: 0.06860937178134918
7/7 ——— 0s 16ms/step
Predicted Values: [43.515858 62.112823 -9.176412 -277.63498
50.05552]
Actual Values: [42.67137813 75.01408257 -4.05539077
-295.72163432 44.43248324]

Result:-

Thus the python program to implement ANN was verified and executed successfully.

Exp: 09

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 a 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 no. of arguments (PROLOG terms) enclosed in parentheses and separated by commas.

Ex:

is_bigger(elephant, x)
f(g(x, _), 7)

4. Facts:-

A Fact is a predicate followed by a dot.

Ex:

bigger_animal (whale)
life_is_beautiful.

5. Rules:-

A rule Consists of a head (a predicate) and a body (a sequence of predicates separated by Commas).

Ex:

is_Smaller (x, y) :- is_bigger (y, x)
aunt (Aunt, child) :- Sister (Aunt, Parent), Parent (Parent, child).

Source Code:-

KB1:-

```
woman (mia).
woman (jody)
woman (yolanda).
plays AirGuitar (jody).
party.

Query1 : ?- woman (mia).
Query2 : ?- plays AirGuitar (mia).
Query3 : ?- party.
Query4 : ?- Concat.
```

Output:-

```
?- woman (mia).
true
?- playAirGuitar (mia).
false.
?- party.
true
?- Concert
Error: Unknown procedure: Concert/0
        (DWIM could not correct goal)
?-
```

KB 2:-

```
happy (Yolanda).
listens2 music (mia).
Listens 2 music (Yolanda) :- happy (Yolanda)
playAirGuitar (mia) :- listens 2 music (mia)
playAirGuitar (Yolanda) :- listen2 music (Yolanda).
```

Output:-

```
?- playsAirGuitar (mia).
true
?- playsAirGuitar (yolanda).
true
?-
```

KB 3:

```
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)
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)

x = Sandwich.

?- dinner (Sandwich)

false.

?-

KBS:-

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 (~~jane~~ who, car(chevy)).

who = john

?- owns (jane, x). Sedan (x).

false.

?- owns (jane, x), truck (x).

x = car (chevy).

Result:-

The learning of PROLOG terminologies and the execution of basic programs has been done successfully.

Exp: 10 DATE: 08/4/24

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)
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)
```

```
parentof (kevin, lisa)
parent of (jeny, john)
parent of (jeny, helen).

/*RULES::*/
/* Son, parent
* Son . grandparent*/
father (x,y) :- male(y) , parent of (x,y).
mother (x,y) :- female (y) , parent of (x,y)
grandfather (x,y) :- male (y), 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
```

OUTPUT:-

male(y), parent of (x,y)

y	x
peter	chris
peter	helen
john	jeny
chris	kevin