26/7/24

EX 1: PYTHON PROGRAMS

```
# pgm 1
# add two matrices
rows = int (input ('Enter no of rows: '))
cols = int (input ('Enter no of columns: '))
print ()
print ('Enter values for matrix A')
matrix A = [[int (input (f"column {j+1} → Enter {i+1} element: '))
            for j in range (cols)] for i in range (rows)]
print ()
print ('Enter values for matrix B')
matrix B = [[int (input (f"column {j+1} → Enter {i+1} element: ")) for j
        in range (cols)] for i in range (rows)]
result = [[0 for j in range (cols)] for j in range (rows)]
for i in range (rows):
    for j in range (cols):
        result[i][j] = matrix_A[i][j] + matrixB [i][j]
print (result)
```

OUTPUT:

```
Enter no of rows 2
Enter no of columns 2

Enter values for matrixA
  2
  2
  2
  2

Enter values for matrixB
  2
  2
  2
  2

[4, 4] [4, 4]
```

```
#pgm 2
# largest in array
def largest(a):
    max_val = a[0][0]
    for i in range(len(a)):
        for j in range (len(a[0])):
            if max_val < a[i][j]:
                max_val = a[i][j]

    return max_val.

a = [[2,7], [5,6]]
print (largest(a))
```

OUTPUT :

```
7
```

```
# pgm 3
# even and odd
def evenorodd():
    e = []
    o = []
    for i in range (len(a)):
        if a[i] % 2 == 0:
            e.append (a[i])
        else:
            o.append (a[i])
    return e,o.
a = [1,2,3,4,5,6,7]
print (evenorodd())
```

OUTPUT

```
[1,3,5,7]
[2,4,6]
```

pgm 4

```
def fib(n):
    fib_s = [0,1]
    while len(fib_s) < n:
        fib_s.append(fib_s[-1] + fib_s[-2])
    return fib_s

n=10
print fib(n)
```

OUTPUT :

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

pgm 5

```
def is_arm(n):
    num_str = str(n)
    num_d = len(num_str)
    sum = sum(int(digit) ** num_d for digit in num_str
    return n == sum.

num = 371
if is_arm(num):
    print ("Yes")
else
    print ("No")
```

OUTPUT :

Yes

9.8.24

PROJECT : IMAGE-BASED MUSIC RECOMMENDATION SYSTEM

DOMAIN: MULTIMEDIA CONTENT RECOMMENDATION

PROBLEM STATEMENT:

Develop a music recommendation system that suggests songs based on user-uploaded images. The system will analyze the images to detect objects, scenes and emotions and then match these with suitable tracks. Using advanced computer vision techniques, the system aims to deliver personalized and relevant music recommendations with an easy to use interface for real-time suggestions.

TARGET AUDIENCE:

PRIMARY USERS : Users of streaming platforms.

SECONDARY USERS : Users of social media who want personalized music recommendation based on images they upload.

OBJECTIVES:

→ Analyse user Images
→ Detect Mood and theme
→ Generate recommendations
→ Good User Interface
⊛ → Real-Time Performance

ALGORITHMS:

1. Image Feature Extraction
2. Object and Scene Detection
3. Emotion and Mood Analysis
4. Recommendation Mapping
5. Enhanced Recommendations.

6.9.24

EXPERIMENT 2:

```
# N Queen
def printSolution(board)
    for row in board:
        print(' '.join(str(cell) for cell in row))
    print()

def isSafe(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 ~~range~~ zip(range(row, N, 1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True.

def solveNQutil(board, col, N, solutions):
    if col >= N:
        solutions.append([row[:] for row in board])
        return
    for i in range(N):
        if isSafe(board, i, col, N):
            board[i][col] = 1
            solveNQutil(board, col+1, N, solutions)
            board[i][col] = 0

def solveNQ():
    N = int(input("Enter no of queens"))
    board = [[0] * N for _ in range(N)]
    solutions = []
    solveNQutil(board, 0, N, solutions)
```

```
if solutions:
    print(f"Total Solutions: {len(solutions)}")
    for index, solution in enumerate(solutions):
        print(f"\nSolution {index+1}:")
        print_solution(solution)
else
    print("Solution does not exist")

solve_NQ()
```

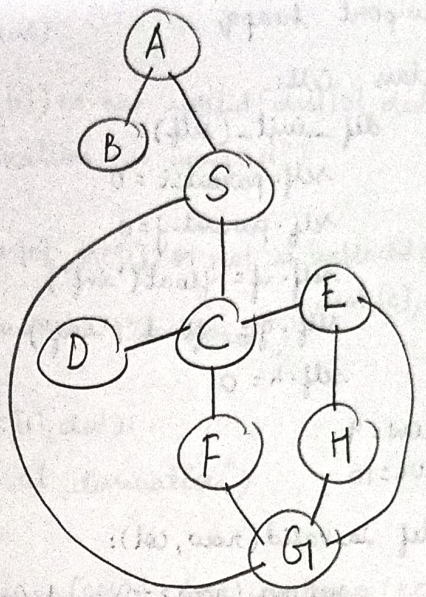
Sample Output :

```
Enter the number of queens: 4
Total solutions: 2

Solution 1:
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0

Solution 2:
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0
```

RESULT: Thus a python program was executed to implement N-Queen problem's solution successfully

EXPERIMENT 3:

```
# Depth first search
graph = {
    'A': ['B', 'S'],
    'B': ['A']
    'C': ['D','E', 'F', 'S'],
    'D': ['C'],
    'E' = ['C', 'H'],
    'F' : ['C', 'G'],
    'G' : ['F', 'S'],
    'H' : ['E', 'G'],
    'S' : ['A', 'C', 'G']
}
```




```
def dfs(graph, node, visited= None):
    if visited is None:
        visited = []
    if node not in visited:
        visited.append(node)
        for neighbor in graph[node]:
            if neighbor not in visited:
                dfs(graph, neighbor, visited)

    return visited

visited_nodes = dfs(graph1, 'A')
print(visited_nodes)
```

Sample Output

['A', 'B', 'S', 'C', 'D', 'E', 'H', 'G', 'F']

RESULT: Thus a python program was executed to implement Depth First Search successfully.

EXPERIMENT- 4

A* Algorithm implementation.

```
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] and col == dest[1]

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

def trace_path(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))
        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_search(grid, src, dest):
    if not is_valid(src[0], src[1]) or not is_valid(dest[0], dest[1]):
        print("Source or destination is invalid")
        return
    if not is_unblocked(grid, src[0], src[1]) or not is_unblocked(grid,
                                                    dest[0], dest[1]):
        print("Source or destination is blocked")
        return
    if is_destination(src[0], src[1], dest):
        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].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), (-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_j][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 dest")
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 1 1 1 1 0 1 0 0]
         [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()
```

Sample OUTPUT

The 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)

EXPERIMENT – 5

```
# water jug problem

def bfs_water_jug_pblm (capacity_jug1, capacity_jug2, target):
    queue = [(0,0)]
    visited = set()
    visited.add = ((0,0))
    parent = {}
    parent [(0,0)] = None
    action = {}
    while queue:
        jug1, jug2 = queue.pop(0)

        if jug1 == target or jug2 == target:
            return reconstruct_path (parent, action, (jug1, jug2))

        possible_states = []
        new_state = (capacity_jug1, jug2)
        if new_state not in visited:
            visited.add (new_state)
            queue.append (new_state)
            parent [new_state] = (jug1, jug2)
            action [new_state] = f" Fill jug1 (3 litres)"

        new_state = (jug1, capacity_jug2)
        if new_state not in visited:
            visited.add (new_state)
            queue.append (new_state)
            parent [new_state] = (jug1, jug2)
            action [new_state] = f" Fill jug2(5 litres)"

        new_state = (0, jug2)
        if new_state not in visited:
            visited.add (new_state)
            queue.append (new_state)
            parent [new_state] = (jug1, jug2)
            action [new_state] = f" Empty jug1"
```

```
        new_state = (jug1, 0)
        if new_state not in visited:
            visited.add(new_state)
            queue.append(new_state)
            parent[new_state] = (jug1, jug2)
            action[new_state] = f"Empty jug2"

    pour_to_jug2 = min(jug1, capacity_jug2 - jug2)
    new_state = (jug1 - pour_to_jug2, jug2 + pour_to_jug2)
    if new_state not in visited:
            visited.add(new_state)
            queue.append(new_state)
            parent[new_state] = (jug1, jug2)
            action[new_state] = f"Pour jug1 into jug2"

    pour_to_jug1 = min(jug2, capacity_jug1 - jug1)
    new_state = (jug1 + pour_to_jug1, jug2 - pour_to_jug1)
    if new_state not in visited:
            visited.add(new_state)
            queue.append(new_state)
            parent[new_state] = (jug1, jug2)
            action[new_state] = f"Pour jug2 into jug1"

~~ret~~
return None

def reconstruct_path(parent, action, end_state):
    path = []
    actions = []
    while end_state is not None:
        path.append(end_state)
        if end_state in action:
            actions.append(action[end_state])
        end_state = parent[end_state]
    path.reverse()
    actions.reverse()
    return list(zip(path, actions))
```

```
def main():
    capacity_jug1 = 3
    capacity_jug2 = 5
    target = 4

    path_with_actions = bfs_water_jug_problem(capacity_jug1,
                                              capacity_jug2, target)

    if path_with_actions:
        print("Solution Path with Actions:")
        for state, act in path_with_actions:
            print(f"{state}- {act}")
    else:
        print("No sol^ found")

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

Sample Output:

Solution Path with Actions:

(0,0) - Fill jug2 (5 litres)
(0,5) - Pour jug2 into jug1
(3,2) - Empty jug1
(0,2) - Pour jug2 into jug1
(2,0) - Fill jug2 (5 litres)
(2,5) - Pour jug2 into jug1

EXPERIMENT-6

AIM: To implement a program using decision tree.

decision tree classification

```
from sklearn.tree import DecisionTreeClassifier
import numpy as np

#Step 1: features & target
# Example : [ht, wt, shoe size]

X = np.array([
        [170, 65, 42],
        [180, 75, 44],
        [160, 50, 38],
        [175, 70, 43],
        [165, 55, 39],
        [185, 80, 45]
    ])

# Target labels
Y = np.array([0, 1, 0, 1, 0, 1])

clf = DecisionTreeClassifier()  # Step 2: Initialise

clf.fit(X, Y)  # Step 3: Train

# Step 4: predict
new_data = np.array([[168, 52, 38]])

prediction = clf.predict(new_data)

print("Predicted gender:", "Male" if prediction[0] == 1 else "Female")
```

Sample Output:

Predicted Gender: Female.

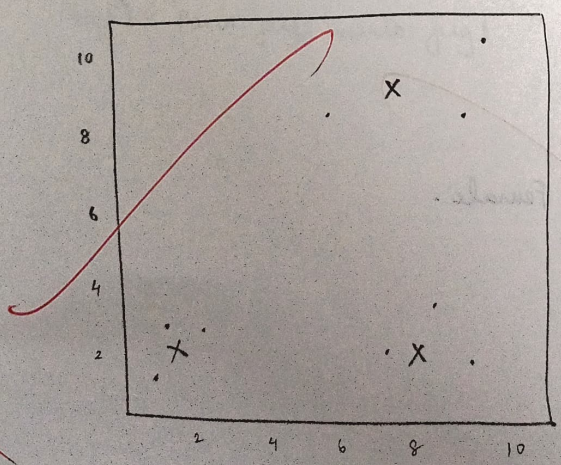
RESULT: Thus a program was implemented succesfully using DecisionTree.

EXPERIMENT - 7

AIM: To implement a program using KMeans

Kmeans

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Step 1: Generate or assign data points
X = np.array([
  [1,2],[1.5,1.8],[5,8], [8,8], [1,0.6], [9,11], [8,2], [10,2], [9,3]])
Kmeans = KMeans(n_clusters=3) # Step 2: Initialise algm
kmeans.fit(X) # step 3: fit kmeans to data
centroids = kmeans.cluster_centers_ # step 4: get cluster centres
labels = kmeans.labels_
# Step 5: Plot the graph
plt.scatter(centroids[:,0], centroids[:,1], c='red', s=200, marker='x'
            label='centroids')

plt.title('Kmeans Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```



RESULT: Thus an algorithm using Kmeans was successfully executed.

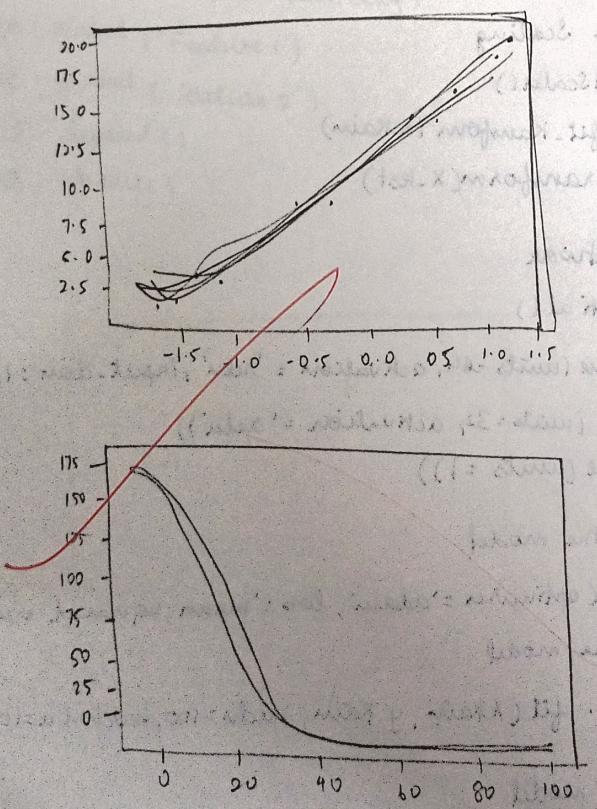
EXPERIMENT - 8

AIM: To implement an artificial neural network.

```
# artificial neural networks

import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import StandardScaler.

# Step 1: generate synthetic data
np.random.seed(42)

X = np.linspace(0,10,100)

y = 2 * X + 1 + np.random.normal(0,1,100)

#X = X.reshape(-1,1)

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

# Step 2: Feature Scaling
Scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 4: Build network
model = Sequential()
model.add(Dense(units =64, activation = 'relu', input_dim = 1))
model.add(Dense(units = 32, activation = 'relu'))
model.add(Dense(units = 1))

# Step 5: compile the model
model.compile(optimizer ='adam', loss ='mean_squared_error')
# Step 6: Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation
                                                  _split =0.2)

#Step 7: Train the model
y_pred = model.predict(X_test)
```

Step 8: Plot the results.

```
plt.scatter(X_test, y_test, color='blue', label='True values')
plt.scatter(X_test, y_pred, color='red', label='Predictions')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.title('Artificial Neural N/w Regn')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



RESULT: Thus a artificial neural network was successfully implemented.

EXPERIMENT - 9

AIM: To learn PROLOG terminologies and write basic programs

TERMINOGIES:

Atomic Terms : strings made up of lowercase, uppercase letters digits and underscore.
Starts with a lowercase letter

dog ab_c_321

Variables - variables are strings starting with capital letters

Dog Apple_420

Compound Terms : are made up of PROLOG atom and a no. of arguments enclosed in parantheis and separated by commas

is_bigger(elephat, X)
f(g(X,_),7)

Facts : A fact is a predicate followed by a dot.

bigger_animal(whale)
life_is_beautifule

Rules : A rule consists of a head (a predicate) and a body (a sequence of predicats seperated by commas).

is_smaller(X,Y) :- is_bigger(Y,X)
aunt(Aunt, child) :- sister(Aunt, Parent), parent(Parent, child)

CODE :

KB1 :

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

Query:

```
? woman(mia).
- true
? playsAirGuitar(mia).
- false
? party.
- true
? concert
- Error
```

KB2

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

Output :

```
?- 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).
```

Output :

```
?- 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)
```

Output :

```
? 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)).
```

Output :

```
? owns (john, X)
X = car (chevy)
? owns (john, _)
true
? - owns (who, car (chevy))
  who = john
```

RESULT: Thus the experiments to learn about PROLOG and execute basic programs was done successfully.

EXPERIMENT - 10

AIM: To develop a family tree program using PROLOG with all possible facts, rules and queries.

CODE:

```
/* FACT :: */
male(peter).
male(john).
male(chris).
male(kevin).

female(betty).
female(jenny).
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).
parentOf(jeny, helen).

/* RULES :: */
/*son, parent
*son, grandparent*/
father(X,Y) :- male(Y), 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(Y), 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 :

? parentOf (kevin, X)

X = chris

? father (X, chris)

X = kevin

? sister (X, chris)

false.

RESULT : Thus the PROLOG program to implement and execute family tree was succesfully completed.