Artificial Intelligence (01CE1702) Lab Manual 24-25

**Name:** Dalsaniya Jay

**ER No.:** 92100103336

**Calss:** 7TC4

|  |  |  |  |
| --- | --- | --- | --- |
| **Lab** | **Program** | **Signature** | **Marks** |
| 1. | Write a prolog Program to understand the concept of facts and queries. |  |  |
| 2. | Write a prolog program to implement the following:   1. Factorial of a given number 2. Fibonacci of a given number |  |  |
| 3 | Write a Prolog program to perform the following operations of the list, i) To display the element of the given list, ii) To check given element is in the list or not, iii) To print the last element of the list,  Iv) To print the sum of the elements of the given list. |  |  |
| 4. | Implement a Family Tree and define the following predicates: 1)parent(X,Y)   1. Father(X,Y) 2. Mother(X,Y) 3. Sister(X,Y) 5)Brother(X,Y)   6)Grandfather(X,Y) 7)Grandmother(X,Y) |  |  |
| 5. | Assume given a set of facts of the form father(name1,name2) (name1 is the father of name2)  Define a predicate cousin(X,Y) which holds iff X and Y are cousins. Define a predicate grandson(X,Y) which holds iff X is a grandson of Y.  Define a predicate descendent(X,Y) which holds iff X is a descendent of Y. Define a predicate grandparent(X,Y) which holds iff X is a grandparent of Y.  Consider the following genealogical tree:  father(a,b).  father(a,c).  father(b,d).  father(b,e).  father(c,f).  Say which answers, and in which order, are generated by your definitions for the following queries in Prolog:  ?- cousin(X,Y).  ?- grandson(X,Y).  ?- descendent(X,Y).  ?-grandparent(X,Y). |  |  |
| 6. | Write a program to solve Tower of Hanoi problem |  |  |
| 7. | Write a program to implement BFS for Water Jug problem/ 8 Puzzle problem  or any AI search problem |  |  |
| 8. | Write a program to implement DFS for Water Jug problem/ 8 Puzzle problem or any AI search problem |  |  |
| 9. | Write a program to implement Single Player Game (Using Heuristic Function) |  |  |
| 10 | Write a program to Implement A\* Algorithm. |  |  |
| 11. | Implement the Mini Max algorithm for game playing |  |  |
| 12. | Write a program to solve N-Queens problem |  |  |
| 13 | Develop an NLP application |  |  |
| 14 | Implement Library for visual representations of text data |  |  |

**Practical 1** : Write a prolog Program to understand the concept of facts and queries.

**Program:**

parent(john, mary).

parent(john, mike).

parent(susan, mary).

parent(susan, mike).

parent(mary, sophia).

parent(mary, james).

parent(paul, sophia).

parent(paul, james).

male(john).

male(mike).

male(paul).

male(james).

female(susan).

female(mary).

female(sophia).

**Output :**

****

**Practical 2 :** Write a prolog program to implement the following:

a.Factorial of a given number

b.Fibonacci of a given number

**program:**

**a) Factorial of a given number**

factorial(0, 1).

factorial(N, F) :-

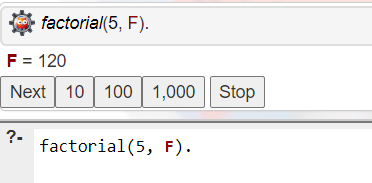
N > 0,

N1 is N - 1,

factorial(N1, F1),

F is N \* F1.

**output :**

****

**b) Fibonacci of a given number**

fibonacci(0, 0).

fibonacci(1, 1).

fibonacci(N, F) :-

N > 1,

N1 is N - 1,

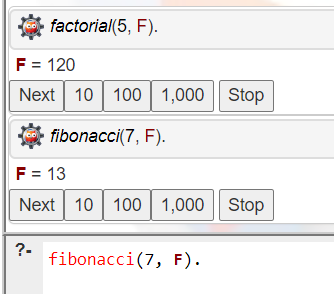
N2 is N - 2,

fibonacci(N1, F1),

fibonacci(N2, F2),

F is F1 + F2.

**output :**

****

**Practical 3 :** Write a Prolog program to perform the following operations of the list,

i) To display the element of the given list,

ii) To check given element is in the list or not,

iii) To print the last element of the list,

Iv) To print the sum of the elements of the given list.

**Program:**

1. To display the element of the given list

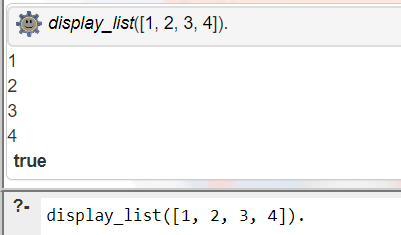
display\_list([]).

display\_list([H|T]) :-

write(H), nl,

display\_list(T).

**output**:



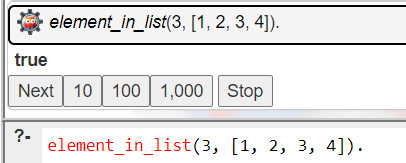
1. To check given element is in the list or not

element\_in\_list(X, [X|\_]).

element\_in\_list(X, [\_|T]) :-

element\_in\_list(X, T).

**output**:



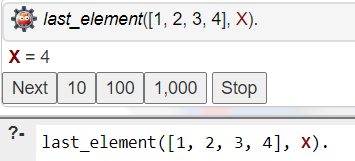
1. To print the last element of the list

last\_element([X], X).

last\_element([\_|T], X) :-

last\_element(T, X).

**output**:



1. To print the sum of the elements of the given list.

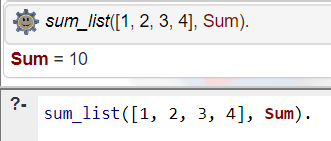
sum\_list([], 0).

sum\_list([H|T], Sum) :-

sum\_list(T, TempSum),

Sum is H + TempSum.

**Output:**

****

**Practical 4:** Implement a Family Tree and define the following predicates:

1)parent(X,Y)

2)Father(X,Y)

3)Mother(X,Y)

4)Sister(X,Y)

5)Brother(X,Y)

6)Grandfather(X,Y)

7)Grandmother(X,Y)

**Program:**

parent(john, mary).

parent(john, mike).

parent(susan, mary).

parent(susan, mike).

parent(mary, sophia).

parent(mary, james).

parent(paul, sophia).

parent(paul, james).

male(john).

male(mike).

male(paul).

male(james).

female(susan).

female(mary).

female(sophia).

father(X, Y) :- parent(X, Y), male(X).

mother(X, Y) :- parent(X, Y), female(X).

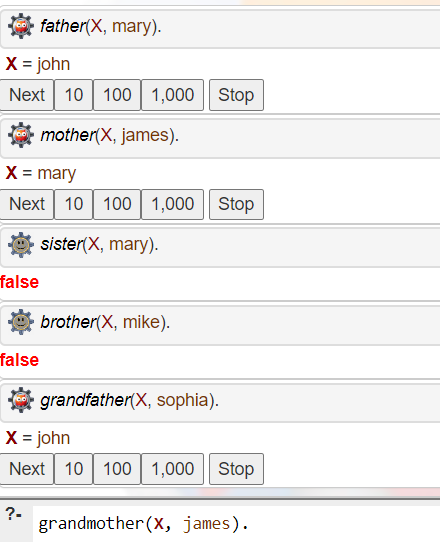
sister(X, Y) :- parent(Z, X), parent(Z, Y), female(X), X \= Y.

brother(X, Y) :- parent(Z, X), parent(Z, Y), male(X), X \= Y.

grandfather(X, Y) :- parent(X, Z), parent(Z, Y), male(X).

grandmother(X, Y) :- parent(X, Z), parent(Z, Y), female(X).

**Output:**

****

**Practical 5:** Assume given a set of facts of the form father(name1,name2) (name1 is the father of name2)

Define a predicate cousin(X,Y) which holds iff X and Y are cousins. Define a predicate grandson(X,Y) which holds iff X is a grandson of Y.

Define a predicate descendent(X,Y) which holds iff X is a descendent of Y. Define a predicate grandparent(X,Y) which holds iff X is a grandparent of Y.

Consider the following genealogical tree:

father(a,b).

father(a,c).

father(b,d).

father(b,e).

father(c,f).

Say which answers, and in which order, are generated by your definitions for the following queries in Prolog:

?- cousin(X,Y).

?- grandson(X,Y).

?- descendent(X,Y).

?-grandparent(X,Y).

**Program:**

father(a, b).

father(a, c).

father(b, d).

father(b, e).

father(c, f).

cousin(X, Y) :-

father(P1, X),

father(P2, Y),

father(GP, P1),

father(GP, P2),

P1 \= P2.

grandson(X, Y) :-

father(Y, P),

father(P, X).

descendent(X, Y) :-

father(Y, X).

descendent(X, Y) :-

father(Y, Z),

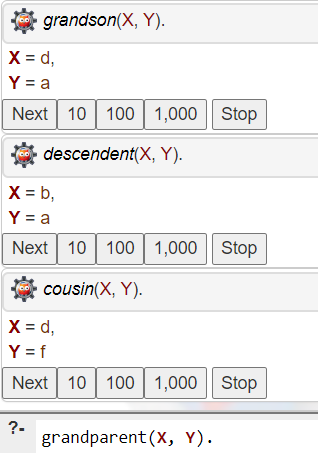
descendent(X, Z).

grandparent(X, Y) :-

father(X, P),

father(P, Y).

**Outout:**

****

**Practical 6 :** Write a program to solve Tower of Hanoi problem

**Program:**

move(1, X, Y, \_) :-

write('Move top disk from '), write(X), write(' to '), write(Y), nl.

move(N, X, Y, Z) :-

N > 1,

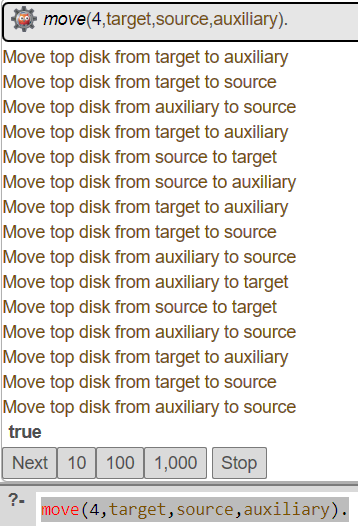
M is N - 1,

move(M, X, Z, Y), % Move N-1 disks from Source to Auxiliary using Target as auxiliary

move(1, X, Y, \_), % Move the remaining disk from Source to Target

move(M, Z, Y, X). % Move the N-1 disks from Auxiliary to Target using Source as auxiliary

**Output:**



**Practical 7 :** Water jug problem using BFS

**Program:**

import java.util.\*;

class Pair {

int j1, j2;

List<Pair> path;

Pair(int j1, int j2) {

this.j1 = j1;

this.j2 = j2;

path = new ArrayList<>();

}

Pair(int j1, int j2, List<Pair> \_path) {

this.j1 = j1;

this.j2 = j2;

path = new ArrayList<>(\_path);

path.add(new Pair(this.j1, this.j2));

}

}

public class WaterJugProblem {

public static void main(String[] args) throws java.lang.Exception {

int jug1 = 4;

int jug2 = 3;

int target = 2;

getPathIfPossible(jug1, jug2, target);

}

private static void getPathIfPossible(int jug1, int jug2, int target) {

boolean[][] visited = new boolean[jug1 + 1][jug2 + 1];

Queue<Pair> queue = new LinkedList<>();

// Initial State: Both Jugs are empty so, initialise j1 j2 as 0 and put it in the path list

Pair initialState = new Pair(0, 0);

initialState.path.add(new Pair(0, 0));

queue.offer(initialState);

while (!queue.isEmpty()) {

Pair curr = queue.poll();

// Skip already visited states and overflowing water states

if (curr.j1 > jug1 || curr.j2 > jug2 || visited[curr.j1][curr.j2]) {

continue;

}

// Mark current jugs state as visited

visited[curr.j1][curr.j2] = true;

// Check if current state has already reached the target amount of water or not

if (curr.j1 == target || curr.j2 == target) {

if (curr.j1 == target) {

// If in our current state, jug1 holds the required amount of water, then we

// empty the jug2 and push it into our path.

curr.path.add(new Pair(curr.j1, 0));

} else {

// else, If in our current state, jug2 holds the required amount of water,

// then we empty the jug1 and push it into our path.

curr.path.add(new Pair(0, curr.j2));

}

int n = curr.path.size();

System.out.println("Path of states of jugs followed is:");

for (int i = 0; i < n; i++)

System.out.println(curr.path.get(i).j1 + " , " + curr.path.get(i).j2);

return;

}

// If we have not yet found the target, then we

// have three cases left:

// I. Fill the jug and Empty the other

// II. Fill the jug and let the other remain untouched

// III. Empty the jug and let the other remain untouched

// IV. Transfer amounts from one jug to another

// I. Fill the jug and Empty the other

queue.offer(new Pair(jug1, 0, curr.path));

queue.offer(new Pair(0, jug2, curr.path));

// II. Fill the jug and let the other remain untouched

queue.offer(new Pair(jug1, curr.j2, curr.path));

queue.offer(new Pair(curr.j1, jug2, curr.path));

// III. Empty the jug and let the other remain untouched

queue.offer(new Pair(0, curr.j2, curr.path));

queue.offer(new Pair(curr.j1, 0, curr.path));

// IV. Transfer water from one to another until one jug becomes empty or until

// one jug becomes full in this process

// Transferring water form jug1 to jug2

int emptyJug = jug2 - curr.j2;

int amountTransferred = Math.min(curr.j1, emptyJug);

int j2 = curr.j2 + amountTransferred;

int j1 = curr.j1 - amountTransferred;

queue.offer(new Pair(j1, j2, curr.path));

// Transferring water form jug2 to jug1

emptyJug = jug1 - curr.j1;

amountTransferred = Math.min(curr.j2, emptyJug);

j2 = curr.j2 - amountTransferred;

j1 = curr.j1 + amountTransferred;

queue.offer(new Pair(j1, j2, curr.path));

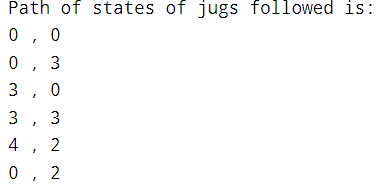
}

System.out.println("Not Possible to obtain target");

}

}

**Output:**

****

**Practical 8 :** Write a program to implement DFS for Water Jug problem/ 8 Puzzle problem or any AI search problem

**Program:**

def is\_goal(state, target):

return target in state

def get\_successors(state, capacities):

successors = []

jug1, jug2 = state

max1, max2 = capacities

# Fill Jug1

if jug1 < max1:

successors.append((max1, jug2))

# Fill Jug2

if jug2 < max2:

successors.append((jug1, max2))

# Empty Jug1

if jug1 > 0:

successors.append((0, jug2))

# Empty Jug2

if jug2 > 0:

successors.append((jug1, 0))

# Pour Jug1 to Jug2

if jug1 > 0 and jug2 < max2:

pour\_amount = min(jug1, max2 - jug2)

successors.append((jug1 - pour\_amount, jug2 + pour\_amount))

# Pour Jug2 to Jug1

if jug2 > 0 and jug1 < max1:

pour\_amount = min(jug2, max1 - jug1)

successors.append((jug1 + pour\_amount, jug2 - pour\_amount))

return successors

def dfs\_water\_jug(start, capacities, target):

stack = [start]

visited = set()

parent\_map = {}

while stack:

state = stack.pop()

if state in visited:

continue

visited.add(state)

if is\_goal(state, target):

path = []

while state:

path.append(state)

state = parent\_map.get(state)

return path[::-1]

for successor in get\_successors(state, capacities):

if successor not in visited:

stack.append(successor)

parent\_map[successor] = state

return None

# Example usage

start\_state = (0, 0) # Both jugs are empty initially

jug\_capacities = (4, 3) # Capacity of jug1 is 4 liters, jug2 is 3 liters

target = 2 # The goal is to measure exactly 2 liters

solution\_path = dfs\_water\_jug(start\_state, jug\_capacities, target)

if solution\_path:

print("Solution path found:")

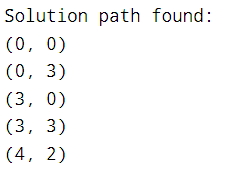
for state in solution\_path:

print(state)

else:

print("No solution found.")

**Output:**

****

**Practical 9 :** Write a program to implement Single Player Game (Using Heuristic Function)

**Program:**

import heapq

class PuzzleState:

def \_\_init\_\_(self, board, moves=0, previous=None):

self.board = board

self.moves = moves

self.previous = previous

self.blank\_pos = self.find\_blank()

def find\_blank(self):

for i in range(3):

for j in range(3):

if self.board[i][j] == 0:

return (i, j)

def \_\_lt\_\_(self, other):

return self.priority() < other.priority()

def priority(self):

return self.moves + self.manhattan\_distance()

def manhattan\_distance(self):

distance = 0

for i in range(3):

for j in range(3):

if self.board[i][j] != 0:

x, y = divmod(self.board[i][j] - 1, 3)

distance += abs(x - i) + abs(y - j)

return distance

def is\_goal(self):

goal = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

return self.board == goal

def generate\_successors(self):

successors = []

x, y = self.blank\_pos

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

for dx, dy in directions:

new\_x, new\_y = x + dx, y + dy

if 0 <= new\_x < 3 and 0 <= new\_y < 3:

new\_board = [row[:] for row in self.board]

new\_board[x][y], new\_board[new\_x][new\_y] = new\_board[new\_x][new\_y], new\_board[x][y]

successors.append(PuzzleState(new\_board, self.moves + 1, self))

return successors

def print\_board(board):

for row in board:

print(" ".join(str(num) if num != 0 else "\_" for num in row))

print()

def a\_star\_search(initial\_board):

start\_state = PuzzleState(initial\_board)

open\_set = []

heapq.heappush(open\_set, start\_state)

closed\_set = set()

while open\_set:

current\_state = heapq.heappop(open\_set)

if current\_state.is\_goal():

return current\_state

closed\_set.add(tuple(map(tuple, current\_state.board)))

for successor in current\_state.generate\_successors():

if tuple(map(tuple, successor.board)) not in closed\_set:

heapq.heappush(open\_set, successor)

return None

def reconstruct\_path(state):

path = []

while state:

path.append(state.board)

state = state.previous

return path[::-1]

def main():

print("Enter the initial state of the 8-puzzle, using 0 for the blank space:")

initial\_board = []

for \_ in range(3):

row = list(map(int, input().split()))

initial\_board.append(row)

print("\nInitial board:")

print\_board(initial\_board)

solution = a\_star\_search(initial\_board)

if solution:

path = reconstruct\_path(solution)

print(f"\nSolved in {len(path) - 1} moves.\n")

for i, step in enumerate(path):

print(f"Step {i}:")

print\_board(step)

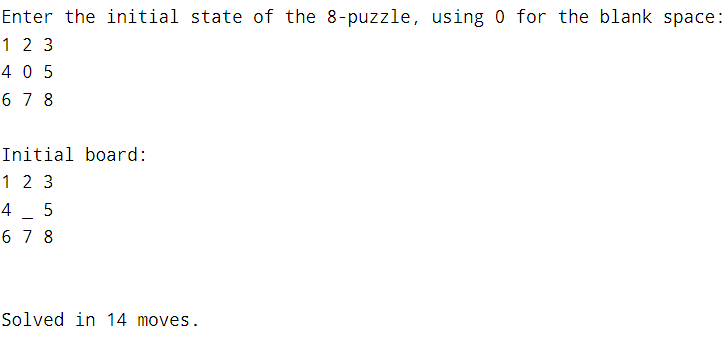
else:

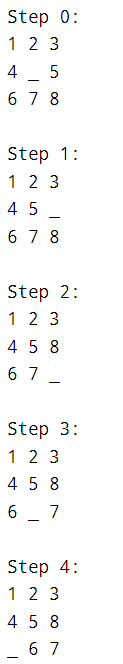
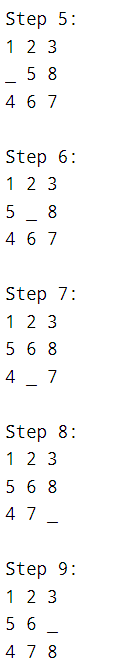
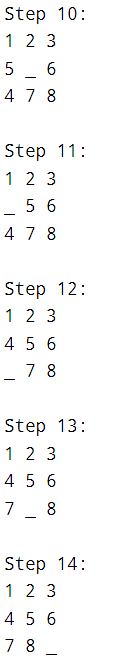
print("No solution found.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output :**

****

****  

**Practical 10 :** Write a program to Implement A\* 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 # Cost from start to node

self.h = h # Heuristic estimate of cost from node to goal

self.f = g + h # Total cost

def \_\_lt\_\_(self, other):

return self.f < other.f

def a\_star\_search(start, goal, graph, heuristic):

open\_list = []

closed\_list = set()

start\_node = Node(start, None, 0, heuristic[start])

goal\_node = Node(goal, None)

heapq.heappush(open\_list, start\_node)

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] # Return reversed path

closed\_list.add(current\_node.name)

for neighbor, cost in graph[current\_node.name].items():

if neighbor in closed\_list:

continue

g = current\_node.g + cost

h = heuristic[neighbor]

neighbor\_node = Node(neighbor, current\_node, g, h)

if add\_to\_open(open\_list, neighbor\_node):

heapq.heappush(open\_list, neighbor\_node)

return None # Return None if no path is found

def add\_to\_open(open\_list, neighbor\_node):

for node in open\_list:

if neighbor\_node.name == node.name and neighbor\_node.f >= node.f:

return False

return True

def main():

# Input the graph

graph = {}

num\_edges = int(input("Enter the number of edges: "))

print("Jay Dalsaniya")

print("92100103336")

print("Enter each edge in the format 'node1 node2 cost':")

for \_ in range(num\_edges):

node1, node2, cost = input().split()

cost = int(cost)

if node1 not in graph:

graph[node1] = {}

if node2 not in graph:

graph[node2] = {}

graph[node1][node2] = cost

graph[node2][node1] = cost # Assuming undirected graph

# Input the heuristic values

heuristic = {}

print("Enter the heuristic values for each node:")

for node in graph:

h\_value = int(input(f"Heuristic value for {node}: "))

heuristic[node] = h\_value

# Input the start and goal nodes

start = input("Enter the start node: ")

goal = input("Enter the goal node: ")

# Perform A\* search

path = a\_star\_search(start, goal, graph, heuristic)

# Output the result

if path:

print(f"Path from {start} to {goal}: {path}")

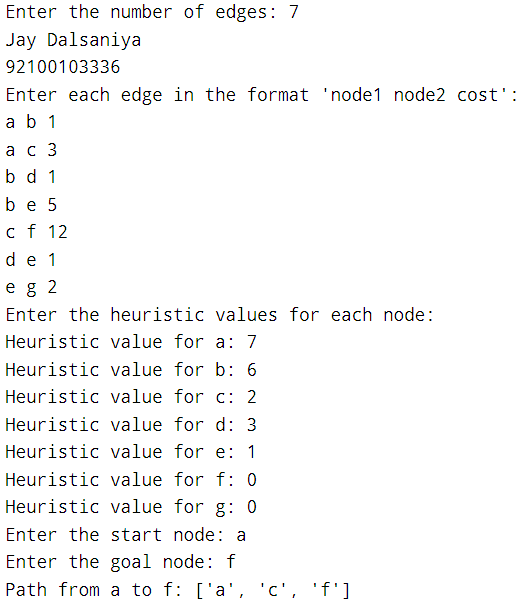
else:

print(f"No path found from {start} to {goal}.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output :**



**Practical 11 :** Implement the Mini Max algorithm for game playing

**Program:**

import math

# Display board

def display\_board(board):

for i in range(0, 9, 3):

print(f"{board[i]} | {board[i+1]} | {board[i+2]}")

if i < 6:

print("--+---+--")

print()

# Check winner

def check\_winner(board, player):

win\_conditions = [(0, 1, 2), (3, 4, 5), (6, 7, 8),

(0, 3, 6), (1, 4, 7), (2, 5, 8),

(0, 4, 8), (2, 4, 6)]

for condition in win\_conditions:

if board[condition[0]] == player and board[condition[1]] == player and board[condition[2]] == player:

return True

return False

# Minimax algorithm

def minimax(board, is\_max):

if check\_winner(board, 'O'):

return 10

if check\_winner(board, 'X'):

return -10

if ' ' not in board:

return 0

best\_score = -math.inf if is\_max else math.inf

for i in range(9):

if board[i] == ' ':

board[i] = 'O' if is\_max else 'X'

score = minimax(board, not is\_max)

board[i] = ' '

best\_score = max(best\_score, score) if is\_max else min(best\_score, score)

return best\_score

# AI move

def ai\_move(board):

best\_move = -1

best\_score = -math.inf

for i in range(9):

if board[i] == ' ':

board[i] = 'O'

score = minimax(board, False)

board[i] = ' '

if score > best\_score:

best\_score = score

best\_move = i

if best\_move != -1:

board[best\_move] = 'O'

# Player move

def player\_move(board):

move = -1

while move not in range(1, 10) or board[move-1] != ' ':

try:

move = int(input("Enter your move (1-9): "))

except ValueError:

pass

board[move-1] = 'X'

# Game loop

def play\_game():

board = [' '] \* 9

while True:

display\_board(board)

if check\_winner(board, 'X'):

print("You win!")

break

if check\_winner(board, 'O'):

print("AI wins!")

break

if ' ' not in board:

print("It's a tie!")

break

player\_move(board)

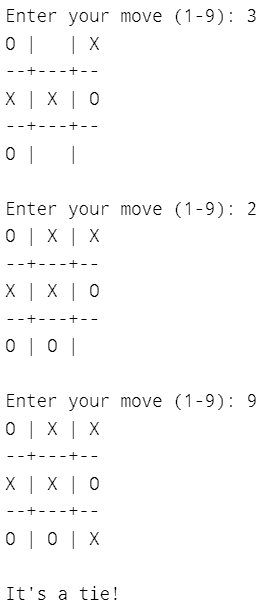
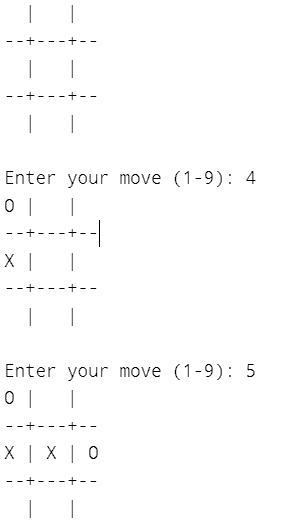
if ' ' in board:

ai\_move(board)

if \_\_name\_\_ == "\_\_main\_\_":

play\_game()

**Output :**

****

**Practical 12 :** Write a program to solve N-Queens problem

**Program:**

# N is the size of the chessboard (N x N)

N = 4

# Function to print the solution

def printSolution(board):

for i in range(N):

for j in range(N):

if board[i][j] == 1:

print("Q", end=" ")

else:

print(".", end=" ")

print()

# Function to check if a queen can be placed on board[row][col]

def isSafe(board, row, col):

# Check the current row on the left side

for i in range(col):

if board[row][i] == 1:

return False

# Check upper diagonal on the left side

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

if board[i][j] == 1:

return False

# Check lower diagonal on the left side

for i, j in zip(range(row, N, 1), range(col, -1, -1)):

if board[i][j] == 1:

return False

return True

# Recursive utility function to solve the N-Queens problem

def solveNQUtil(board, col):

# Base case: If all queens are placed, return True

if col >= N:

return True

# Try placing the queen in each row of the current column

for i in range(N):

if isSafe(board, i, col):

# Place the queen

board[i][col] = 1

# Recur to place the rest of the queens

if solveNQUtil(board, col + 1):

return True

# If placing the queen does not lead to a solution, backtrack

board[i][col] = 0

# If the queen cannot be placed in any row in this column, return False

return False

# Function to solve the N-Queens problem using backtracking

def solveNQ():

# Initialize the board with all 0's (empty board)

board = [[0 for \_ in range(N)] for \_ in range(N)]

if not solveNQUtil(board, 0):

print("Solution does not exist")

return False

printSolution(board)

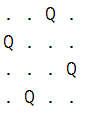
return True

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

solveNQ()

**Output:**

****

**Practical 13 :** Develop an NLP application

**Program:**

from nltk.sentiment.vader import SentimentIntensityAnalyzer import nltk sia = SentimentIntensityAnalyzer()

# Tweets about AI tweets = [

"Artificial Intelligence is transforming the world in unimaginable ways!",

"AI can help solve complex problems but it must be handled responsibly.",

"I'm really excited to see how AI is being used in healthcare.",

"AI in education is going to make learning more personalized and accessible.",

"The future of AI is bright but we need to ensure it doesn't replace jobs.",

"AI technology is advancing faster than we can keep up with."

]

def analyze\_sentiment(tweets): for

tweet in tweets: print(f"Tweet:

{tweet}") score =

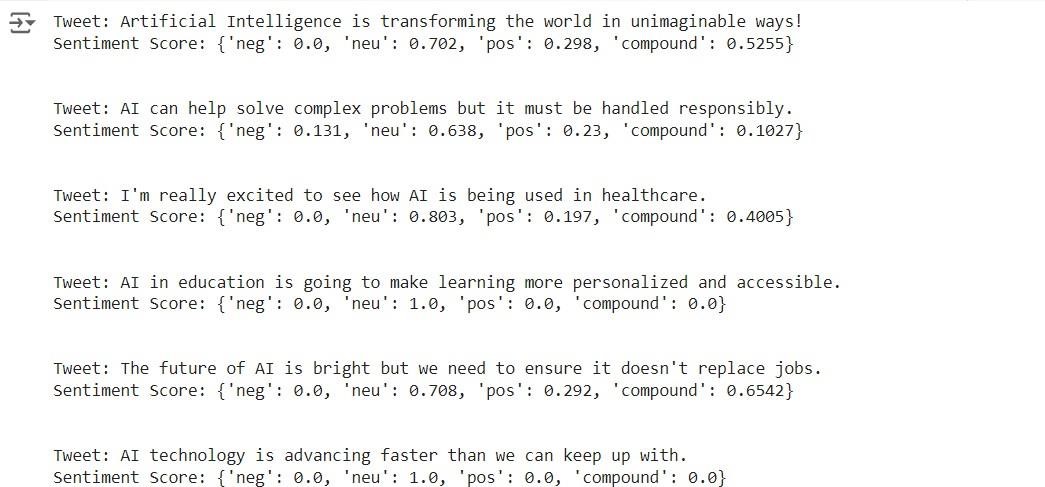
sia.polarity\_scores(tweet)

print(f"Sentiment Score: {score}")

print("\n")

analyze\_sentiment(tweets)

**Output:**



**Practical 14 :** Implement Library for visual representations of text data

**Program:**

import matplotlib.pyplot as plt import seaborn as sns

from wordcloud import WordCloud import networkx as nx from collections import Counter import numpy as np

from sklearn.decomposition import PCA

#import pyLDAvis #import pyLDAvis.sklearn

def plot\_wordcloud(text,max\_words=100,colormap='viridis'):

wordcloud=WordCloud(max\_words=max\_words,colormap=colormap,background\_color='white').generat e(text)

plt.figure(figsize=(10,5))

plt.imshow(wordcloud,interpolation='bilinear') plt.axis('off') plt.show()

def plot\_barchart(text,top\_n=20): words=text.split() word\_counts=Counter(words)

most\_common=word\_counts.most\_common(top\_n)

labels,values=zip(\*most\_common) plt.figure(figsize=(10,6)) sns.barplot(x=labels,y=values)

plt.title(f"Top {top\_n} Words by Frequency.") plt.show()

def plot\_heatmap(data\_matrix,x\_labels,y\_labels):

plt.figure(figsize=(10,6))

sns.heatmap(data\_matrix,annot=True,cmap='coolwarm',xticklabels=x\_labels,yticklabels=y\_labels) plt.title("Heatmap of Text Features") plt.show()

def plot\_graph(edges):

G=nx.Graph()

G.add\_edges\_from(edges) plt.figure(figsize=(10,6)) pos=nx.spring\_layout(G) nx.draw(G,pos,with\_labels=True,node\_color='lightblue',edge\_color='gray',node\_size=1500,font\_size=10

)

plt.show()

def plot\_scatterplot(embeddings,labels=None):

pca=PCA(n\_components=2)

reduced\_embeddings=pca.fit\_transform(embeddings) plt.figure(figsize=(10,6))

plt.scatter(reduced\_embeddings[:,0],reduced\_embeddings[:,1],s=50) plt.xlabel("Principal Component 1") plt.ylabel("Principal Component 2") plt.title("PCA projection of text embeddings.") plt.show()

if name == " main ": text = "machine learning models can predict outcomes based on historical data." plot\_wordcloud(text) plot\_barchart(text) data=np.random.rand(10,10)

plot\_heatmap(data,x\_labels=[f'word{i}'for i in range(10)],y\_labels=[f'word{i}'for i in range(10)]) edges=[('A','B'),('B','C'),('C','D'),('D','E'),('E','A')] plot\_graph(edges)

embeddings=np.random.rand(100,100) plot\_scatterplot(embeddings)

**Output:**

