PRACTICAL: 1

Practical: 1

Aim: Write a program to implement Tic-Tac-Toe game problem.

Code:

```
import numpy as np
import random
from time import sleep

def create_board():
	return(np.array([[0, 0, 0],
                                        [0, 0, 0],
                                        [0, 0, 0]]))

def possibilities(board):
	l = []

	for i in range(len(board)):
		for j in range(len(board)):

			if board[i][j] == 0:
				l.append((i, j))
	return(l)

def random_place(board, player):
	selection = possibilities(board)
	current_loc = random.choice(selection)
	board[current_loc] = player
	return(board)

def row_win(board, player):
	for x in range(len(board)):
		win = True

		for y in range(len(board)):
			if board[x, y] != player:
				win = False
				continue

		if win == True:
			return(win)
	return(win)

def col_win(board, player):
	for x in range(len(board)):
		win = True

		for y in range(len(board)):
			if board[y][x] != player:
				win = False
				continue

		if win == True:
```

```
                        return(win)
        return(win)

def diag_win(board, player):
        win = True
        y = 0
        for x in range(len(board)):
                if board[x, x] != player:
                        win = False
        if win:
                return win
        win = True
        if win:
                for x in range(len(board)):
                        y = len(board) - 1 - x
                        if board[x, y] != player:
                                win = False
        return win

def evaluate(board):
        winner = 0

        for player in [1, 2]:
                if (row_win(board, player) or
                                col_win(board, player) or
                                diag_win(board, player)):

                        winner = player

        if np.all(board != 0) and winner == 0:
                winner = -1
        return winner

def play_game():
        board, winner, counter = create_board(), 0, 1
        print(board)
        sleep(2)

        while winner == 0:
                for player in [1, 2]:
                        board = random_place(board, player)
                        print("Board after " + str(counter) + " move")
                        print(board)
                        sleep(2)
                        counter += 1
                        winner = evaluate(board)
                        if winner != 0:
                                break
        return(winner)

print("Winner is: " + str(play_game()))
```

Output:

Case 1:

```
"C:\Users\HP DR0006TX\PycharmProjects\aipratical\venv\Scripts\python.exe" "C:\Users\HP DR0006TX\PycharmProjects\aipratical\main.py"
  |   |
---------
  |   |
---------
  |   |
---------
Player X, enter row and column (e.g., 1 2): 1 1
X |   |
---------
  |   |
---------
  |   |
---------
Player O, enter row and column (e.g., 1 2): 1 2
X | O |
---------
  |   |
---------
  |   |
---------
Player X, enter row and column (e.g., 2 1
X | O |
---------
X |   |
---------
  |   |
---------
```

```
Player O, enter row and column (e.g., 1 2): 2 2
X | O |
---------
X | O |
---------
  |   |
---------
Player X, enter row and column (e.g., 1 2): 3 1
X | O |
---------
X | O |
---------
X |   |
---------
Player X wins!

Process finished with exit code 0
```

Case 2:

```
"C:\Users\HP DR0006TX\PycharmProjects\aipratical\venv\Scripts\python.exe" "C:\Users\HP DR0006TX\PycharmProjects\aipratical\main.py"
  |   |
---------
  |   |
---------
  |   |
---------
Player X, enter row and column (e.g., 1 2): 2 1
  |   |
---------
X |   |
---------
  |   |
---------
Player O, enter row and column (e.g., 1 2): 2 2
  |   |
---------
X | O |
---------
  |   |
---------
Player X, enter row and column (e.g., 1 2): 2 3
  |   |
---------
X | O | X
---------
  |   |
---------
```

```
Player O, enter row and column (e.g., 1 2): 1 2
  | O |
---------
X | O | X
---------
  |   |
---------
Player X, enter row and column (e.g., 1 2): 1 1
X | O |
---------
X | O | X
---------
  |   |
---------
Player O, enter row and column (e.g., 1 2): 3 2
X | O |
---------
X | O | X
---------
  | O |
---------
Player O wins!

Process finished with exit code 0
```

Case 3:

```
"C:\Users\HP DR0006TX\PycharmProjects\aipratical\venv\Scripts\python.exe" "C:\Users\HP DR0006TX\PycharmProjects\aipratical\main.py"
  |   |
---------
  |   |
---------
  |   |
---------
Player X, enter row and column (e.g., 1 2): 2 2
  |   |
---------
  | X |
---------
  |   |
---------
Player O, enter row and column (e.g., 1 2): 1 1
O |   |
---------
  | X |
---------
  |   |
---------
Player X, enter row and column (e.g., 1 2): 1 3
O |   | X
---------
  | X |
---------
  |   |
---------
```

```
Player O, enter row and column (e.g., 1 2): 3 1
O |   | X
---------
  | X |
---------
O |   |
---------
Player X, enter row and column (e.g., 1 2): 2 1
O |   | X
---------
X | X |
---------
O |   |
---------
Player O, enter row and column (e.g., 1 2): 2 3
O |   | X
---------
X | X | O
---------
O |   |
---------
Player X, enter row and column (e.g., 1 2): 3 2
O |   | X
---------
X | X | O
---------
O | X |
---------
```

```
Player O, enter row and column (e.g., 1 2): 1 2
O | O | X
---------
X | X | O
---------
O | X |
---------
Player X, enter row and column (e.g., 1 2): 3 3
O | O | X
---------
X | X | O
---------
O | X | X
---------
It's a tie!

Process finished with exit code 0
```

PRACTICAL: 2

Practical: 2

Aim: - Write a program to implement BFS (for 8 puzzle problem or Water Jugproblem or any AI search problem).

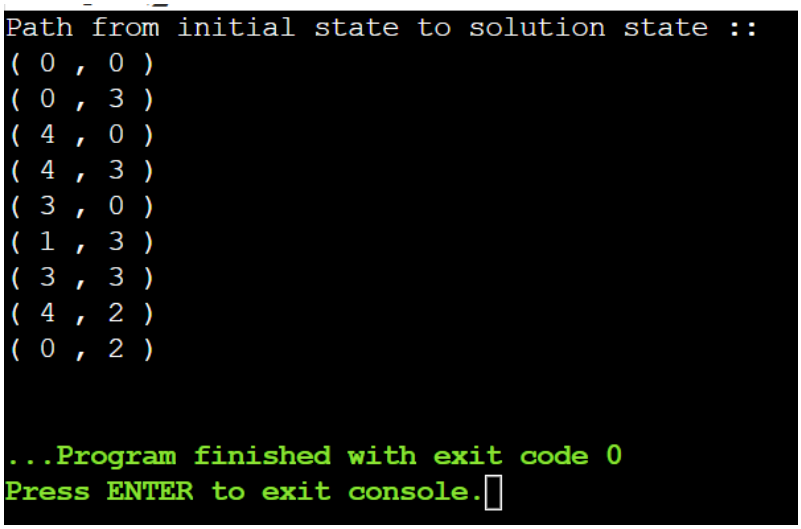
Code:

```
from collections import deque

def BFS(a, b, target):

m = {}

isSolvable = False

path = []

q = deque()

q.append((0, 0))

while (len(q) > 0):

        u = q.popleft()

        if ((u[0], u[1]) in m):

                continue

        if ((u[0] > a or u[1] > b or

                u[0] < 0 or u[1] < 0)):

                continue

        path.append([u[0], u[1]])

        m[(u[0], u[1])] = 1

        if (u[0] == target or u[1] ==

                target):isSolvable = True

                if (u[0] == target):

                        if (u[1] != 0):
```

```
                path.append([u[0], 0])
            else:
                if (u[0] != 0):
                    path.append([0, u[1]])
            sz = len(path)
            for i in range(sz):
                print("(", path[i][0], ",",
                    path[i][1], ")")
            break
        q.append([u[0], b])
        q.append([a, u[1]])
        for ap in range(max(a, b) + 1):
            c = u[0] + ap
            d = u[1] - ap
            if (c == a or (d == 0 and d >=
                0)):q.append([c, d])
            c = u[0] - ap d
              = u[1] + ap

            if ((c == 0 and c >= 0) or d ==
                b):q.append([c, d])
                q.append([a, 0])
                q.append([0, b])
 if (not isSolvable):
```

```
        print("No solution")

if__name__== '_main_':

Jug1, Jug2, target = 4, 3, 2

print("Path from initial state " "to solution state ::")

BFS(Jug1, Jug2, target)
```

Output:



PRACTICAL: 3

Practical: 3

Aim: Write a program to implement DFS (for 8 puzzle problem or Water Jug problem or any AI search problem).

Code:

```
def water_jug_dfs(jug1_capacity, jug2_capacity, target_amount, jug1=0, jug2=0,
    visited=set()): if jug1 == target_amount and jug2 == 0:
     return [(jug1,
jug2)]
visited.add((jug1,
jug2))
if jug1 < jug1_capacity and (jug1_capacity, jug2) not in visited:
  path = water_jug_dfs(jug1_capacity, jug2_capacity, target_amount, jug1_capacity,
jug2, visited) if path:
   return [(jug1, jug2)] + path
if jug2 < jug2_capacity and (jug1, jug2_capacity) not in visited:
  path = water_jug_dfs(jug1_capacity, jug2_capacity, target_amount, jug1,
jug2_capacity, visited) if path:
   return [(jug1, jug2)] +
path if path:
   return [(jug1, jug2)] + path
if jug2 > 0 and (jug1, 0) not in visited:
   path = water_jug_dfs(jug1_capacity, jug2_capacity, target_amount, jug1, 0,
visited) if path:
    return [(jug1, jug2)] + path
if jug1 > 0 and jug2 < jug2_capacity:
pour_amount = min(jug1, jug2_capacity - jug2)
path = water_jug_dfs(jug1_capacity, jug2_capacity, target_amount, jug1 - pour_amount,
jug2 + pour_amount, visited)
if path:
  return [(jug1, jug2)] + path
```

```
pour_amount = min(jug2, jug1_capacity - jug1)
path = water_jug_dfs(jug1_capacity, jug2_capacity, target_amount, jug1 + pour_amount,
jug2 - pour_amount, visited)
  if path:
   return [(jug1, jug2)] + path return []
jug1_capacity = 4 jug2_capacity= 3 target_amount = 2
solution = water_jug_dfs(jug1_capacity, jug2_capacity,
target_amount) if solution:
   for step, state
   inenumerate(solution):
   print(f"Step{p+1}: {state}")
else:
   print("No solution found.")
```

Output:

```
Step 1: (0, 0)
Step 2: (4, 0)
Step 3: (4, 3)
Step 4: (0, 3)
Step 5: (3, 0)
Step 6: (3, 3)
Step 7: (4, 2)
Step 8: (0, 2)
Step 9: (2, 0)
```

PRACTICAL: 4

Practical-4

Aim: - Write a program to implement Single Player Game (Using any Heuristic Function)

Code:

```
import random
def heuristic_guess(low, high):
    return (low + high) // 2
def play_game():
    print("Welcome to the Guessing
    Game!") target_number =
    random.randint(1, 100) low, high = 1,
    100
    attempts = 0
    while True:
guess = heuristic_guess(low, high)
print(f"I guess {guess}")
if guess == target_number:
        print(f"Congratulations! I guessed the number {target_number} in {attempts}
attempts.")
        break
    elif guess < target_number:
        print("Too low!")
  low = guess + 1 else:
        print("Too
        high!") high =
        guess - 1
attempts += 1
if __name__ == "__main__":
    play_game()
```

Output:

```
Welcome to the Guessing Game!
I guess 50
Too high!
I guess 25
Too low!
I guess 37
Too high!
I guess 31
Too low!
I guess 34
Too high!
I guess 32
Too low!
I guess 33
Congratulations! I guessed the number 33 in 6 attempts.
```

PRACTICAL: 5

Practical 5

AIM: Implement A* algorithm.

Code:

```
import heapq
class Node:
    def __init__(self, state, parent=None, action=None, cost=0, heuristic=0):
        self.state = state
        self.parent = parent
        self.action = action
        self.cost = cost
        self.heuristic = heuristic

    def __lt__(self, other):
        return (self.cost + self.heuristic) < (other.cost + other.heuristic)

def astar(start_state, goal_state, get_neighbors, heuristic):
    open_list = []
    closed_set = set()
    start_node = Node(start_state, None, None, 0, heuristic(start_state, goal_state))
    heapq.heappush(open_list, start_node)
    while open_list:
        current_node = heapq.heappop(open_list)
        if current_node.state == goal_state:
            path = []
            while current_node:
                path.append((current_node.state, current_node.action))
                current_node = current_node.parent
            return list(reversed(path))
        closed_set.add(current_node.state)
        for neighbor_state, action, step_cost in get_neighbors(current_node.state):
            if neighbor_state in closed_set:
                continue
            g_score = current_node.cost + step_cost
            h_score = heuristic(neighbor_state, goal_state)
            f_score = g_score + h_score
            neighbor_node = Node(neighbor_state, current_node, action, g_score, h_score)
            # Check if the neighbor is already in the open list
            found = False
            for node in open_list:
                if node.state == neighbor_state:
                    found = True
                    if g_score < node.cost:
                        open_list.remove(node)
                        heapq.heappush(open_list, neighbor_node)
                    break
            if not found:
                heapq.heappush(open_list, neighbor_node)
```

```
    return None  # No path found
    def get_neighbors(state):
        x, y = state
        neighbors = []
        for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
            new_x, new_y = x + dx, y + dy
         if 0 <= new_x < len(grid) and 0 <= new_y < len(grid[0]) and grid[new_x][new_y] == 0:
                neighbors.append(((new_x, new_y), f"Move to ({new_x}, {new_y})", 1))
        return neighbors

    def manhattan_distance(state, goal):
        x1, y1 = state
        x2, y2 = goal
        return abs(x1 - x2) + abs(y1 - y2)

    path = astar(start, goal, get_neighbors, manhattan_distance)
    if path:
        for state, action in path:
            print(f"Action: {action}, State: {state}")
    else:
        print("No path found.")
```

Output:

```
"C:\Users\HP DR0006TX\PycharmProjects\pythonProject11\venv\Scripts\python.exe"
Shortest path from A to F: ['A', 'C', 'F']


Process finished with exit code 0
```

PRACTICAL: 6

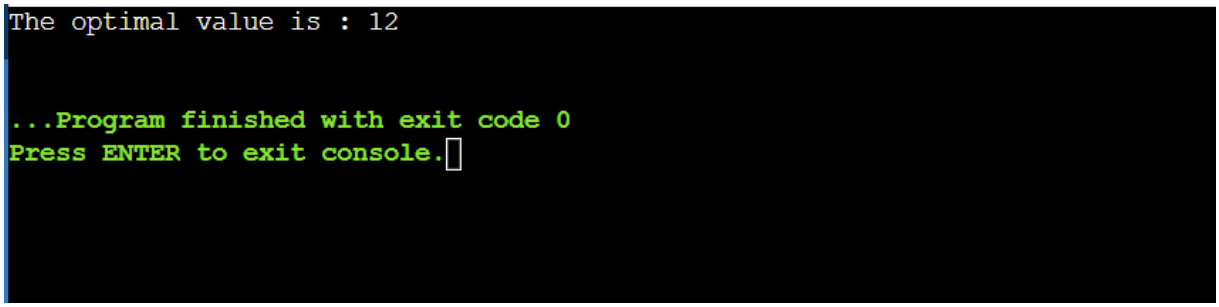
Practical: 6

Aim: Write a program to implement mini-max algorithm for any game development.

Code:

```
# A simple Python3 program to
find# maximum score that
# maximizing player can
getimport math

def minimax (curDepth, nodeIndex,
                    maxTurn,
                    scores,
                    targetDepth):

# base case : targetDepth
reachedif (curDepth ==
targetDepth):
        return scores[nodeIndex]

if (maxTurn):
        return max(minimax(curDepth + 1, nodeIndex * 2,
                                False, scores, targetDepth),
                           minimax(curDepth + 1, nodeIndex * 2
                               + 1,False, scores, targetDepth))

else:
        return min(minimax(curDepth + 1, nodeIndex * 2,
                                True, scores, targetDepth),
                           minimax(curDepth + 1, nodeIndex * 2
                               + 1,True, scores, targetDepth))

# Driver code
scores = [3, 5, 2, 9, 12, 5, 23, 23]

treeDepth = math.log(len(scores), 2)

print("The optimal value is : ", end = "")
print(minimax(0, 0, True, scores,
treeDepth))

# This code is
contributed# by
rootshadow
```

Output:


The optimal value is : 12
...Program finished with exit code 0
Press ENTER to exit console.

PRACTICAL: 7

Practical: 7

Aim: Assume given a set of facts of the form father (name1, name2) (name1is the father of name2).

Code:

```
female(pam).
female(liz).
female(pat).
female(ann).

male(jim).
male(bob).
male(tom).
male(peter).

parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).

parent(bob,pat).
parent(pat,jim).
parent(bob,peter).
parent(peter,jim).

mother(X,Y):- parent(X,Y),female(X).
father(X,Y):-parent(X,Y),male(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.
grandparent(X,Y):-parent(X,Z),parent(Z,Y).
grandmother(X,Z):-mother(X,Y),parent(Y,Z).
grandfather(X,Z):-father(X,Y),parent(Y,Z).
wife(X,Y):-
parent(X,Z),parent(Y,Z),female(X),male(Y).
uncle(X,Z):-brother(X,Y),parent(Y,Z).
```

Output:

PRACTICAL: 8

Practical: 8

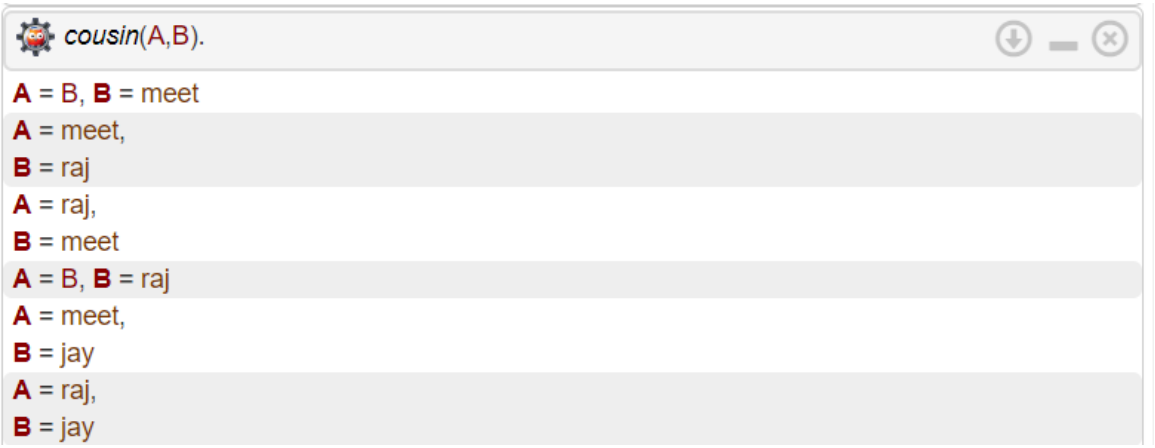
Aim : Define a predicate brother(X,Y) which holds iff X and Y arebrothers.
Define a predicate cousin(X,Y) which holds iff X and Y are cousins.
Define a predicate grandson(X,Y) which holds iff X is a grandson of Y.
Define a predicate descendent(X,Y) which holds iff X is a descendent of Y.
Consider the 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:
?- brother(X,Y).
?- cousin(X,Y).
?- grandson(X,Y).
?- descendent(X,Y).

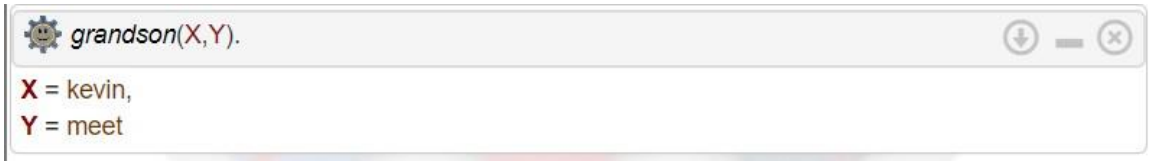
Code:

```
father(kevin,milu).

father(kevin,yash).

father(milu,meet).

father(milu,raj).

father(yash,jay).


brother(X,Y):-father(K,X),father(K,Y).

cousin(A,B):-father(K,X),father(K,Y),father(X,A),father(Y,B).

grandson(X,Y):-father(X,K),father(K,Y).

descendent(X,Y):-father(K,X),father(K,Y).

descendent(X,Y):-father(K,X),father(K,Y),father(X,A),father(Y,B).
```

Output:


brother(X,Y).
X = Y, Y = milu
X = milu,
Y = yash


cousin(A,B).
A = B, B = meet
A = meet,
B = raj
A = raj,
B = meet
A = B, B = raj
A = meet,
B = jay
A = raj,
B = jay


grandson(X,Y).
X = kevin,
Y = meet


descendent(X,Y).
X = Y, Y = milu
X = milu,
Y = yash
X = yash,
Y = milu

PRACTICAL: 9

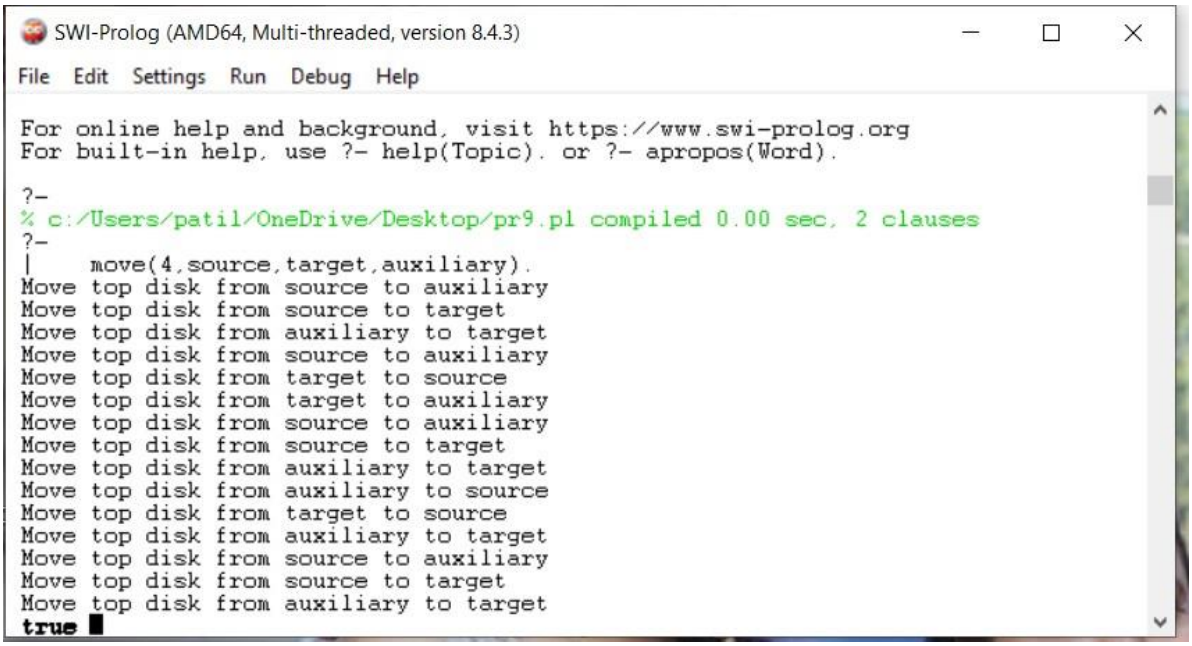
Practical-9

Aim: - Write a program to solve Tower of Hanoi problem using Prolog.

Code:

```
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(1,X,Y,_),
move(M,Z,Y,X).
```

Output:



PRACTICAL: 10

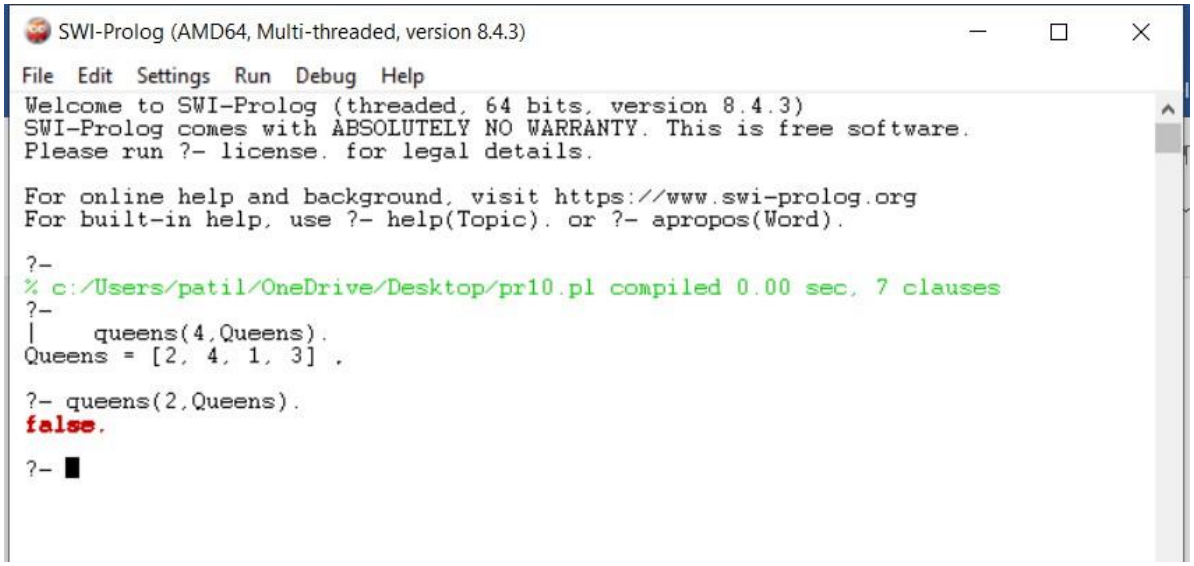
Practical-10

Aim: - Write a program to solve N-Queens problem using Prolog.

Code:

```
% render solutions nicely. use_rendering(chess).

queens(N, Queens) :- length(Queens, N), board(Queens, Board, 0, N, _, _),
queens(Board, 0, Queens).

board([], [], N, N, _, _). board([_|Queens],

[Col-Vars|Board],

Col0, N, [_|VR], VC) :- Col is Col0+1, functor(Vars, f, N),

constraints(N, Vars, VR, VC),

board(Queens, Board, Col, N, VR, [_|VC]). constraints(0, _, _, _) :- !.

constraints(N, Row, [R|Rs], [C|Cs]) :- arg(N, Row, R-C),

M is N-1,

constraints(M, Row, Rs, Cs).

queens([], _, []).

queens([C|Cs], Row0, [Col|Solution]) :- Row is Row0+1,

select(Col-Vars, [C|Cs], Board), arg(Row, Vars, Row-Row), queens(Board, Row,
Solution).
```

Output:



PRACTICAL: 11

Practical-11

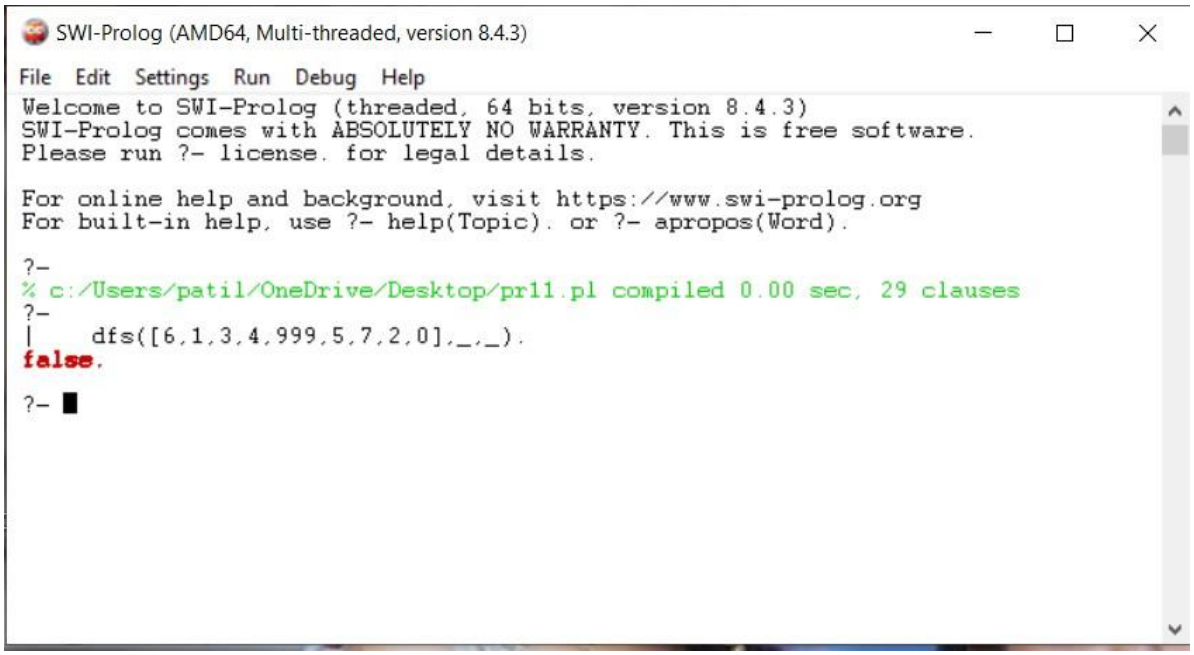
Aim: - Write a program to solve 8 puzzle problem using Prolog.

Code:

```
goal([1,2,3, 4,0,5, 6,7,8]).
move([X1,0,X3, X4,X5,X6, X7,X8,X9],
[0,X1,X3, X4,X5,X6, X7,X8,X9]).
move([X1,X2,0, X4,X5,X6, X7,X8,X9],
[X1,0,X2, X4,X5,X6, X7,X8,X9]).
move([X1,X2,X3, X4,0,X6,X7,X8,X9],
[X1,X2,X3, 0,X4,X6,X7,X8,X9]).
move([X1,X2,X3, X4,X5,0,X7,X8,X9],
[X1,X2,X3, X4,0,X5,X7,X8,X9]).
move([X1,X2,X3, X4,X5,X6, X7,0,X9],
[X1,X2,X3, X4,X5,X6, 0,X7,X9]).
move([X1,X2,X3, X4,X5,X6, X7,X8,0],
[X1,X2,X3, X4,X5,X6, X7,0,X8]).
move([0,X2,X3, X4,X5,X6, X7,X8,X9],
[X2,0,X3, X4,X5,X6, X7,X8,X9]).
move([X1,0,X3, X4,X5,X6, X7,X8,X9],
[X1,X3,0, X4,X5,X6, X7,X8,X9]).
move([X1,X2,X3, 0,X5,X6, X7,X8,X9],
[X1,X2,X3, X5,0,X6, X7,X8,X9]).
move([X1,X2,X3, X4,0,X6, X7,X8,X9],
[X1,X2,X3, X4,X6,0, X7,X8,X9]).
move([X1,X2,X3, X4,X5,X6,0,X8,X9],
[X1,X2,X3, X4,X5,X6,X8,0,X9]).
move([X1,X2,X3, X4,X5,X6,X7,0,X9],
[X1,X2,X3, X4,X5,X6,X7,X9,0]).
```

```
move([X1,X2,X3, 0,X5,X6, X7,X8,X9],
[0,X2,X3, X1,X5,X6, X7,X8,X9]).
move([X1,X2,X3, X4,0,X6, X7,X8,X9],
[X1,0,X3, X4,X2,X6, X7,X8,X9]).
move([X1,X2,X3, X4,X5,0, X7,X8,X9],
[X1,X2,0, X4,X5,X3, X7,X8,X9]).
move([X1,X2,X3, X4,X5,X6, X7,0,X9],
[X1,X2,X3, X4,0,X6, X7,X5,X9]).
move([X1,X2,X3, X4,X5,X6, X7,X8,0],
[X1,X2,X3, X4,X5,0, X7,X8,X6]).
move([X1,X2,X3, X4,X5,X6, 0,X8,X9],
[X1,X2,X3, 0,X5,X6, X4,X8,X9]).
move([0,X2,X3, X4,X5,X6, X7,X8,X9],
[X4,X2,X3, 0,X5,X6, X7,X8,X9]).
move([X1,0,X3, X4,X5,X6, X7,X8,X9],
[X1,X5,X3, X4,0,X6, X7,X8,X9]).
move([X1,X2,0, X4,X5,X6, X7,X8,X9],
[X1,X2,X6, X4,X5,0, X7,X8,X9]).
move([X1,X2,X3, 0,X5,X6, X7,X8,X9],
[X1,X2,X3, X7,X5,X6, 0,X8,X9]).
move([X1,X2,X3, X4,0,X6, X7,X8,X9],
[X1,X2,X3, X4,X8,X6, X7,0,X9]).
move([X1,X2,X3, X4,X5,0, X7,X8,X9],
[X1,X2,X3, X4,X5,X9, X7,X8,0]). dfsSimplest(S, [S]) :- goal(S).
dfsSimplest(S, [S|Rest]) :- move(S, S2), dfsSimplest(S2, Rest). dfs(S, Path, Path) :-
goal(S). dfs(S, Checked, Path) :- move(S, S2), \+member(S2, Checked), dfs(S2,
[S2|Checked], Path).
```

Output:


SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.
For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
?-
% c:/Users/patil/OneDrive/Desktop/pr11.pl compiled 0.00 sec, 29 clauses
?-
| dfs([6,1,3,4,999,5,7,2,0],_,_).
false.
?-

PRACTICAL: 12

Practical-12

Aim: - Write a program to solve 8 puzzle problem using Prolog.

Code:

```
edge(a, b, 3).
edge(a, c, 4).
edge(a, d, 2).
edge(a, e, 7).
edge(b, c, 4).
edge(b, d, 6).
edge(b, e, 3).
edge(c, d, 5).
edge(c, e, 8).
edge(d, e, 6).
edge(b, a, 3).
edge(c, a, 4).
edge(d, a, 2).
edge(e, a, 7).
edge(c, b, 4).
edge(d, b, 6).
edge(e, b, 3).
edge(d, c, 5).
edge(e, c, 8).
edge(e, d, 6).
edge(a, h, 2).
edge(h, d, 1).
len([], 0).
len([H|T], N):- len(T, X), N is X+1 . best_path(Visited, Total):- path(a, a, Visited, Total).
path(Start, Fin, Visited, Total) :- path(Start, Fin, [Start], Visited, 0, Total). path(Start,
```

```
Fin, CurrentLoc, Visited, Costn, Total) :-

edge(Start, StopLoc, Distance), NewCostn is Costn + Distance, \+ member(StopLoc,

CurrentLoc),

path(StopLoc, Fin, [StopLoc|CurrentLoc], Visited, NewCostn, Total).

path(Start, Fin, CurrentLoc, Visited, Costn, Total) :-

edge(Start, Fin, Distance), reverse([Fin|CurrentLoc], Visited), len(Visited, Q), (Q\=7 ->

Total is 100000; Total is Costn + Distance).

shortest_path(Path):-setof(Cost-Path, best_path(Path,Cost), Holder),pick(Holder,Path).

best(Cost-Holder,Bcost-_,Cost-Holder):- Cost<Bcost,!.

best(_,X,X).

pick([Cost-Holder|R],X):- pick(R,Bcost-Bholder),best(Cost-Holder,Bcost-Bholder,X),!.

pick([X],X).
```

Output: