ARTIFICIAL INTELLIGENCE PRACTICAL

NAME : Sumit Popat Dige.
ROLL NO. : ST.23.152

PRACTICAL - 01.A

Aim : Implement depth first search algorithm.

Code : BFS.py

```
graph1={
      'A' :set(['B','C']),
      'B' :set(['A','D','E']),
      'C' :set(['A','F']),
       'D' :set(['B']),
       'E' :set(['B','F']),
       'F' :set(['C','E'])
      }

def DFS(graph,node,visited):
      if node not in visited:
            visited.append(node)
            for n in graph[node]:
                  DFS(graph,n,visited)
      return visited

visited=DFS(graph1,'A',[])
print(visited)
```

OutPut :

```
>>>
= RESTART: C:/Users/Sumit/OneDrive/Desktop/AI practical/DFS.py
['A', 'B', 'E', 'F', 'C', 'D']
>>>
```

PRACTICAL - 01.B

Aim : Implement breadth first search algorithm.

Code : BFS.py

```
graph={'A':set(['B','C']),
       'B': set(['A','D','E']),
       'C': set(['A','F',]),
       'D': set(['B']),
       'E': set(['B','F']),
       'F': set(['C','E'])
       }
def BFS(start):
    queue=[start]
    levels={}
    levels[start]=0
    visited=set(start) #v=
    while queue:
        node=queue.pop(0)
        ns=graph[node] #ns=neighbours
        for ng in ns:
            if ng not in visited: #ng=neighbor
                queue.append(ng)
                visited.add(ng)
                levels[ng]=levels[node]+1
    print(levels)
    return visited
print(str(BFS('A')))
```

OutPut :

```
Python 3.12.5 (tags/v3.12.5:ff3bc82, Aug  6 2024, 20:45:27) [MSC
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more infor
>>
= RESTART: C:/Users/Sumit/OneDrive/Desktop/AI practical/BFS.py
{'A': 0, 'B': 1, 'C': 1, 'E': 2, 'D': 2, 'F': 2}
{'F', 'D', 'A', 'E', 'C', 'B'}
>>
```

PRACTICAL - 02.A

Aim : Simulate 4-Queen / N-Queen problem.

Code : 4QueenP.py

```
global N
N = 4

def printSolution(board):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end=' ')
        print()

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

def solveNQUtil(board, col):
    if col >= N:
        return True
    for i in range(N):
        if isQSafe(board, i, col):
            board[i][col] = 1
            if solveNQUtil(board, col + 1):
                return True
            board[i][col] = 0
    return False

def solveNQ():
    board = [[0, 0, 0, 0],
             [0, 0, 0, 0],
             [0, 0, 0, 0],
             [0, 0, 0, 0]]
```

```
    if not solveNQUtil(board, 0):
            print("Solution does not exist")
            return False
        printSolution(board)
        return True

    solveNQ()
```

OutPut :

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: C:/Users/Sumit/OneDrive/Desktop/AI practical/4QueenP.py =======
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

PRACTICAL - 02.B

Aim : Solve tower of Hanoi problem.

Code : Hanoi.py

```
def t(h,s,aux,e):
    if h>=1:
        print("line1")
        t(h-1,s,e,aux)
        print("line2")
        print("moving disk from",h,s,"to",e)
        print("line3")
        t(h-1,aux,s,e)

t(3,"A","B","C")
```

OutPut :

```
>>
= RESTART: C:/Users/Sumit/OneDrive/Desktop/AI practical/Hanoi.py
line1
line1
line1
line2
moving disk from 1 A to C
line3
line2
moving disk from 2 A to B
line3
line1
line2
moving disk from 1 C to B
line3
line2
moving disk from 3 A to C
line3
line1
line1
line2
moving disk from 1 B to A
line3
line2
moving disk from 2 B to C
line3
line1
line2
moving disk from 1 A to C
line3
```

PRACTICAL - 03.A

Aim : Implement alpha beta search.

Code : .py

OutPut :

PRACTICAL - 03.B

Aim : Implement hill climbing problem.

Code : Hill_Climbing.py

```
SuccList={'A':[['B',3],['C',2]],'B':[['D',2],['E',3]],'C':[['F',2],['G',4]],'D':[['H',1],['I',99]],'F':[['J',1]],'
G':[['K',99],['L',3]]}
Start='A'

Closed=list()
SUCCESS=True
FAILURE=False

def MOVEGEN (N):
    New_list=list()
    if N in SuccList.keys():
        New_list=SuccList[N]

    return New_list

def SORT(L):
    L.sort(key=lambda x:x[1])
    return L

def heu(Node):
    return Node[1]

def APPEND(L1,L2):
    New_list=list(L1)+list(L2)
    return New_list

def Hill_Climbing(Start):
    global Closed
    N=Start
    CHILD=MOVEGEN(N)
    SORT(CHILD)
    N=[Start,5]
    print("\nStart=",N)
    print("Sorted Child List=",CHILD)
    newNode=CHILD[0]
    CLOSED=[N]
```

```
    while (heu(newNode) < heu(N)) and (len(CHILD) !=0):
        print("\n-------------------------")
        N=newNode
        print("N=",N)
        CLOSED = APPEND (CLOSED,[N])
        CHILD = MOVEGEN (N[0])
        SORT(CHILD)
        print("Sorted Child List=",CHILD)
        print("CLOSED=",CLOSED)
        newNode=CHILD[0]

    closed=CLOSED

Hill_Climbing(Start) #call search algorithm
```

OutPut :

```
>>
= RESTART: C:/Users/Sumit/OneDrive/Desktop/AI practical/Hill_Climbing.py

Start= ['A', 5]
Sorted Child List= [['C', 2], ['B', 3]]

-------------------------
N= ['C', 2]
Sorted Child List= [['F', 2], ['G', 4]]
CLOSED= [['A', 5], ['C', 2]]
>>
```

PRACTICAL - 04.A

Aim : Implement A* algorithm.

Code : AStar.py

```
Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('H', 5)],
    'F': [('G', 1), ('H', 7)],
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('H', 3)],
}

def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

def h(n):
    H_dist = {
        'A': 10,
        'B': 8,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[n]

def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}
    parents = {}
```

```
        g[start_node] = 0
        parents[start_node] = start_node

        while len(open_set) > 0:
            n = None

            for v in open_set:
                if n is None or g[v] + h(v) < g[n] + h(n):
                    n = v

            if n == stop_node or get_neighbors(n) is None:
                pass
            else:
                for (m, weight) in get_neighbors(n):
                    if m not in open_set and m not in closed_set:
                        open_set.add(m)
                        parents[m] = n
                        g[m] = g[n] + weight
                    elif m in open_set:
                        if g[m] > g[n] + weight:
                            g[m] = g[n] + weight
                            parents[m] = n

            if n == stop_node:
                path = []
                while parents[n] != n:
                    path.append(n)
                    n = parents[n]
                path.append(start_node)
                path.reverse()
                print('Path found: {}'.format(path))
                return path

            open_set.remove(n)
            closed_set.add(n)

        print('Path does not exist!')
        return None
    aStarAlgo('A', 'I')
```

OutPut :

```
= RESTART: C:/Users/Sumit/OneDrive/Desktop/AI practical/AStar.py
Path found: ['A', 'F', 'G', 'I']
```

PRACTICAL - 04.B

Aim : Solve water jug problem.

Code : .py

```
j1 = int(input("Capacity of jug 1: "))
j2 = int(input("Capacity of jug 2: "))
q = int(input("Amount of water to be measured: "))

def apply_rule(ch, x, y):
    # Rule 1: Fill jug 1
    if ch == 1:
        if x < j1:
            return j1, y
        else:
            print("Rule cannot be applied")
            return x, y

    # Rule 2: Fill jug 2
    elif ch == 2:
        if y < j2:
            return x, j2
        else:
            print("Rule cannot be applied")
            return x, y

    # Rule 3: Transfer all water from jug 1 to jug 2
    elif ch == 3:
        if x > 0 and x + y <= j2:
            return 0, x + y
        else:
            print("Rule cannot be applied")
            return x, y

    # Rule 4: Transfer all water from jug 2 to jug 1
    elif ch == 4:
        if y > 0 and x + y <= j1:
            return x + y, 0
        else:
            print("Rule cannot be applied")
            return x, y
```

```
    # Rule 5: Transfer some water from jug 1 to jug 2 until jug 2 is full
    elif ch == 5:
        if x > 0 and x + y >= j2:
            return x - (j2 - y), j2
        else:
            print("Rule cannot be applied")
            return x, y

    # Rule 6: Transfer some water from jug 2 to jug 1 until jug 1 is full
    elif ch == 6:
        if y > 0 and x + y >= j1:
            return j1, y - (j1 - x)
        else:
            print("Rule cannot be applied")
            return x, y

    # Rule 7: Empty jug 1
    elif ch == 7:
        if x > 0:
            return 0, y
        else:
            print("Rule cannot be applied")
            return x, y

    # Rule 8: Empty jug 2
    elif ch == 8:
        if y > 0:
            return x, 0
        else:
            print("Rule cannot be applied")
            return x, y

    else:
        print("INVALID CHOICE")
        return x, y

# Initialize capacities of both jugs as 0
x = y = 0

while True:
    if x == q or y == q:
        print('GOAL ACHIEVED!')
        break
    else:
```

```
        print("================================== Rules
==================================")
        print("Rule 1: Fill jug 1")
        print("Rule 2: Fill jug 2")
        print("Rule 3: Transfer all water from jug 1 to jug 2")
        print("Rule 4: Transfer all water from jug 2 to jug 1")
        print("Rule 5: Transfer some water from jug 1 to jug 2 until jug 2 is full")
        print("Rule 6: Transfer some water from jug 2 to jug 1 until jug 1 is full")
        print("Rule 7: Empty jug 1")
        print("Rule 8: Empty jug 2")
        ch = int(input("Enter rule to apply: "))
        x, y = apply_rule(ch, x, y)
        print("=========== STATUS ===========")
        print("CURRENT STATUS:", x, y)
```

OutPut :

```
Python 3.12.5 (tags/v3.12.5:ff3bc82, Aug  6 2024, 20:45:27) [MSC v.19
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more informati
>>>
= RESTART: C:/Users/Sumit/OneDrive/Desktop/AI practical/jugP.py
Capacity of jug 1: 3
Capacity of jug 2: 2
Amount of water to be measured: 5
================================== Rules ==============================
Rule 1: Fill jug 1
Rule 2: Fill jug 2
Rule 3: Transfer all water from jug 1 to jug 2
Rule 4: Transfer all water from jug 2 to jug 1
Rule 5: Transfer some water from jug 1 to jug 2 until jug 2 is full
Rule 6: Transfer some water from jug 2 to jug 1 until jug 1 is full
Rule 7: Empty jug 1
Rule 8: Empty jug 2
Enter rule to apply:
```

PRACTICAL - 05.A

Aim : Simulate tic – tac – toe game using min-max algorithm.

Code : tic tac toe.py

```
import os
import time

board = [' '] * 10   # Using index 1-9 for the game board
player = 1

# Constants for game states
Win = 1
Draw = -1
Running = 0

Game = Running
Mark = 'X'

def DrawBoard():
    print("%c | %c | %c" % (board[1], board[2], board[3]))
    print("__|___|__")
    print("%c | %c | %c" % (board[4], board[5], board[6]))
    print("__|___|__")
    print("%c | %c | %c" % (board[7], board[8], board[9]))
    print("   |    |   ")

def CheckPosition(x):
    return board[x] == ' '

def CheckWin():
    global Game

    # Horizontal winning conditions
    if board[1] == board[2] == board[3] and board[1] != ' ':
        Game = Win
    elif board[4] == board[5] == board[6] and board[4] != ' ':
        Game = Win
    elif board[7] == board[8] == board[9] and board[7] != ' ':
        Game = Win

    # Vertical winning conditions
    elif board[1] == board[4] == board[7] and board[1] != ' ':
        Game = Win
```

```
    elif board[2] == board[5] == board[8] and board[2] != ' ':
        Game = Win
    elif board[3] == board[6] == board[9] and board[3] != ' ':
        Game = Win

    # Diagonal winning conditions
    elif board[1] == board[5] == board[9] and board[1] != ' ':
        Game = Win
    elif board[3] == board[5] == board[7] and board[3] != ' ':
        Game = Win

    # Check for draw
    elif all(cell != ' ' for cell in board[1:]):
        Game = Draw
    else:
        Game = Running

print("Tic-Tac-Toe Game")
print("Player 1 [X] -- Player 2 [O]\n")
print("Please wait...")
time.sleep(1)

while Game == Running:
    os.system('cls' if os.name == 'nt' else 'clear')
    DrawBoard()
    if player % 2 != 0:
        print("Player 1's Chance")
        Mark = 'X'
    else:
        print("Player 2's Chance")
        Mark = 'O'

    try:
        choice = int(input("Enter the position between [1-9] where you want to mark: "))
        if choice < 1 or choice > 9 or not CheckPosition(choice):
            print("Invalid move. Try again.")
            continue
        board[choice] = Mark
        player += 1
        CheckWin()
    except (ValueError, IndexError):
        print("Invalid input. Please enter a number between 1 and 9.")

os.system('cls' if os.name == 'nt' else 'clear')
DrawBoard()
```

```
    if Game == Draw:
        print("Game Draw")
    elif Game == Win:
        player -= 1
        if player % 2 != 0:
            print("Player 1 Won")
        else:
            print("Player 2 Won")
```

OutPut :

```
===== RESTART: C:\Users\Sumit\OneDrive\Desktop\AI practical\tic tac toe.py
Tic-Tac-Toe Game
Player 1 [X] -- Player 2 [O]

Please wait...
   |   |
___|___|___
   |   |
___|___|___
   |   |
   |   |
Player 1's Chance
Enter the position between [1-9] where you want to mark: 1
X  |   |
___|___|___
   |   |
___|___|___
   |   |
   |   |
Player 2's Chance
Enter the position between [1-9] where you want to mark: 3
X  |   |  O
___|___|___
   |   |
___|___|___
   |   |
   |   |
Player 1's Chance
Enter the position between [1-9] where you want to mark: 5
X  |   |  O
___|___|___
   | X |
___|___|___
   |   |
   |   |
Player 2's Chance
Enter the position between [1-9] where you want to mark: 6
X  |   |  O
___|___|___
   | X |  O
___|___|___
   |   |
   |   |
Player 1's Chance
Enter the position between [1-9] where you want to mark: 9
X  |   |  O
___|___|___
   | X |  O
___|___|___
   |   |  X
   |   |
Player 1 Won
>>
```

PRACTICAL - 05.B

Aim : Shuffle deck of cards.

Code : .py

OutPut :

PRACTICAL - 06.A

Aim : Design an application to simulate number puzzle problem.

Code : puzzle.py

```
def print_in_format(matrix):
    for i in range(9):
        if i%3==0 and i>0:
            print("")
        print(str(matrix[i])+" ",end=" ")
def count(s):
    c=0
    ideal=[1,2,3,
           4,5,6,
           7,8,0]
    for i in range(9):
        if s[i]!=0 and s[i]!=ideal[i]:
            c+=1
    return c

def move(ar,p, st):
    store_st=st.copy()
    for i in range(len(arr)):
        dup1_st=st.copy()
        tmp=dup1_st[p]
        dup1_st[p]=dup1_st[ar[i]]
        dup1_st[ar[i]]=tmp

        trh=count(dup1_st)
        store_st=dup1_st.copy()
    return store_st,trh
state=[1,2,3,
       0,5,6,
       4,7,8]
h=count(state)
Level = 1

print("\n--------Level"+str(Level)+"-----------")
print_in_format(state)
print("\n Heuristic Value (Misplaced):" +str(h))

while h>0:
    pos=int(state.index(0))
    print('pos',pos)
```

```
Level +=1

    if pos==0:
        arr=[1,3]
        state,h=move(arr,pos,state)
    elif pos==1:
        arr=[0,2,4]
        state,h=move(arr,pos,state)
    elif pos==2:
        arr=[1,5]
        state,h=move(arr,pos,state)
    elif pos==3:
        arr=[0,4,6]
        print(arr)
        state,h=move(arr,pos,state)
    elif pos==4:
        arr=[1,3,5,7]
        state,h=move(arr,pos,state)
    elif pos==5:
        arr=[2,4,8]
        state,h=move(arr,pos, state)
    elif pos==6:
        arr=[3,7]
        state,h=move(arr,pos,state)
    elif pos==7:
        arr=[4,6,8]
        state,h=move(arr,pos,state)
    elif pos==8:
        arr=[5,6]
        state,h=move(arr,pos,state)

    print("\n----------Level"+str(Level)+"---------")
    print_in_format(state)
    print("\n Heuristic Value(Misplaced):"+str(h))
```

OutPut :

```
= RESTART: C:/Users/Sumit/OneDrive/Desktop/AI practical/puzzle.py

--------Level1----------
1  2  3
0  5  6
4  7  8
 Heuristic Value (Misplaced):3
pos 3
[0, 4, 6]

----------Level2---------
1  2  3
4  5  6
0  7  8
 Heuristic Value(Misplaced):2
pos 6

----------Level3---------
1  2  3
4  5  6
7  0  8
 Heuristic Value(Misplaced):1
pos 7

----------Level4---------
1  2  3
4  5  6
7  8  0
 Heuristic Value(Misplaced):0
```

PRACTICAL - 07.A

Aim : Solve constraint satisfaction problem.

Code : .py

OutPut :

PRACTICAL - 08.B

Aim : Derive the expressions based on Associative Law.

Code : Associative**.**pl

```
associative_law(A, B, C, Result1, Result2) :-
    Result1 is A + (B + C),
    Result2 is (A + B) + C.

expression1(2, 3, 4).
expression2(5, 6, 7).

derive_results :-
    expression1(A, B, C),
    associative_law(A, B, C, Result1A, Result2A),
    expression2(X, Y, Z),
    associative_law(X, Y, Z, Result1B, Result2B),
    write('Result of expression 1 using associative law is : '), nl,
    write('A + (B + C) = '), write(Result1A), nl,
    write('(A + B) + C = '), write(Result2A), nl,
    write('Result of expression 2 using associative law is : '), nl,
    write('X + (Y + Z) = '), write(Result1B), nl,
    write('(X + Y) + Z = '), write(Result2B), nl.
```

OutPut :

```
C:/Users/Sumit/OneDrive/Desktop/AI practical/Associative.pl

yes
| ?- derive_results.
Result of expression 1 using associative law is :
A + (B + C) = 9
(A + B) + C = 9
Result of expression 2 using associative law is :
X + (Y + Z) = 18
(X + Y) + Z = 18

yes
| ?- |
```

PRACTICAL - 08.B

Aim : Derive the expressions based on Distributive Law.

Code : Distributive.pl

```
distributive_law(A, B, C, Result1, Result2) :-
    Result1 is A * (B + C),
    Result2 is A * B + A * C.

expression1(2, 3, 4).
expression2(5, 6, 7).

derive_results :-
    expression1(A, B, C),
    distributive_law(A, B, C, Result1A, Result2A),
    expression2(X, Y, Z),
    distributive_law(X, Y, Z, Result1B, Result2B),
    write('Result of expression 1 using distributive law is : '), nl,
    write('A * (B + C) = '), write(Result1A), nl,
    write('A * B + A * C = '), write(Result2A), nl,
    write('Result of expression 2 using distributive law is : '), nl,
    write('X * (Y + Z) = '), write(Result1B), nl,
    write('X * Y + X * Z = '), write(Result2B), nl.
```

OutPut :

```
C:/Users/Sumit/OneDrive/Desktop/AI practical/Distributive.pl

yes
| ?- derive_results.
Result of expression 1 using distributive law is :
A * (B + C) = 14
A * B + A * C = 14
Result of expression 2 using distributive law is :
X * (Y + Z) = 65
X * Y + X * Z = 65

yes
```

PRACTICAL - 09.A

Aim : Derive the predicate. (for e.g.: Sachin is batsman, batsman is cricketer) - > Sachin is Cricketer.

Code : batsman**.**pl

```
batsman(sachin).
batsman(dhoni).
footballer(ronaldo).
cricketer(X) :- batsman(X).
```

OutPut :

```
C:/Users/Sumit/OneDrive/Desktop/AI practical/batsman.pl

yes
| ?- cricketer(X).

X = sachin ? ;

X = dhoni

yes
| ?- |
```

PRACTICAL - 10.A

Aim : Write a program which contains three predicates: male, female, parent. Make rules for following family relations: father, mother, grandfather, grandmother, brother,sister, uncle, aunt, nephew and niece, cousin. Question: i. Draw Family Tree. ii.Define:Clauses, Facts, Predicates and Rules with conjunction and disjunction.

Code : family.pl

```
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(pat,jim).
parent(bob,peter).
parent(peter,jim).
parent(pam,liz).

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 :

```
compiling C:/Users/Sumit/OneDrive/Desktop/AI practical/
C:/Users/Sumit/OneDrive/Desktop/AI practical/family.pl

yes
| ?- mother(X,Y).

X = pam
Y = bob ? ;

X = pat
Y = jim ? ;

X = pam
Y = liz

yes
| ?- wife(X,Y).

X = pam
Y = tom ? ;

X = pat
Y = peter ? ;

X = pam
Y = tom ? ;

no
| ?- father(X,Y).

X = tom
Y = bob ? ;

X = tom
Y = liz ? ;

X = bob
Y = ann ? ;

X = bob
Y = peter ? ;

X = peter
Y = jim ? ;

no
```