B.M.S. COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

Lab Record

Artificial Intelligence

Submitted in partial fulfillment for the 5th Semester Laboratory

Bachelor of Technology
in
Computer Science and Engineering

Submitted by:

Gunjal Kothari
1BM21CS274

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
Nov-Mar 2024

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Artificial Intelligence (22CS5PCAIN) laboratory has been carried out by Gunjal Kothari (1BM21CS274) during the 5th Semester Nov-March- 2024.

Signature of the Faculty Incharge:

Prof. Shravya AR
Assistant Professor
Department of Computer Science and Engineering
B.M.S. College of Engineering, Bangalore

Table of Contents

* Program - 01.
Implement Tic-Tac-Toe Game

```
board = [' '
for x in
range (10)]

def insertLetter (letter, pos):
    board[pos] = letter.

def ~~isSpaceFree~~ spaceIsFree (pos):
    return board[pos] == ' '

def printBoard (board):
    print (' |   | ')
    print (' ' + board[1] + '|' + board[2] +
           '|' + board[3])
    print ('|   |   ')
    print ('------')
    print ('|   |')
    print ('------')
    print ('------')
    print ('|   |')
    print (' ' + board[7] + '|' + board[8] +
           '|' + board[9])
    print ('|   |')

def isWinner (bo, le):
    ~~return[bo]~~
    return (bo[7] == le and bo[8] == le and
            bo[9] == le) or (bo[4] == le and
            bo[5] == le and bo[6] == le) or
```

```
( bo[1] == le and bo[2] == le and bo[3] == le)
or ( bo[1] == le and bo[4] == le and bo[7]
== le ) or (
be[2] == le and bo[5] == le and bo[8] == le) or
( bo[3] == le and bo[6] == le and bo[9] == le) or
  bo[1] == le and bo[6] == le and bo[9] == le) or
(bo[3] == le and bo[5] == le and bo[7] == le)

def playerMove():
    run = True
    while run:
    move = input ('Please select a position
    to place an 'X' (1-9): ')
    try:
    move = int(move)
    if move > 0 and move < 10:
        if spaceIsFree(move):
            run = False
        insertLetter('X', move)
else:
    print ('Sorry, this space is occupied!')
else
    print('Please try a number within
          the range!')
    except:
      print ('Please type a number!')

dif compMove():
    possibleMoves = [x for x, letter in
    enumerate(board) if letter == ' ' &
    x != 0]
    move = 0
```

```
cornersOpen=[]
for i in possibleMoves :
    if i in [1,3,7,9]
        cornersOpen.append(i)

if len(cornersOpen) > 0 :
    move = selectRandom(cornersOpen)
    return move

if len(edgesOpen) > 0 :
    move = selectRandom(edgesOpen)
    return move

def selectRandom(li) :
    import random
    r = random.randrange(0,ln)
    return li[r]

while not (isBoardFull(board)) :
    if not (isWinner(board,'O')) :
        playerMove()
        printBoard(board)
    else :
        print('Sorry, O\'s won this time')
        break

if isBoardFull(board) :
    print('Tie Game')

while True :
    answer = input('Do you want to
if answer.lower() == 'y' or answer.lower
    board = [' ' for x in range(10)]
    print('-----')
```

main()
else:
break.

* Output:

Gunjal Kothari

Do Welcome To Tic Tac Toe:

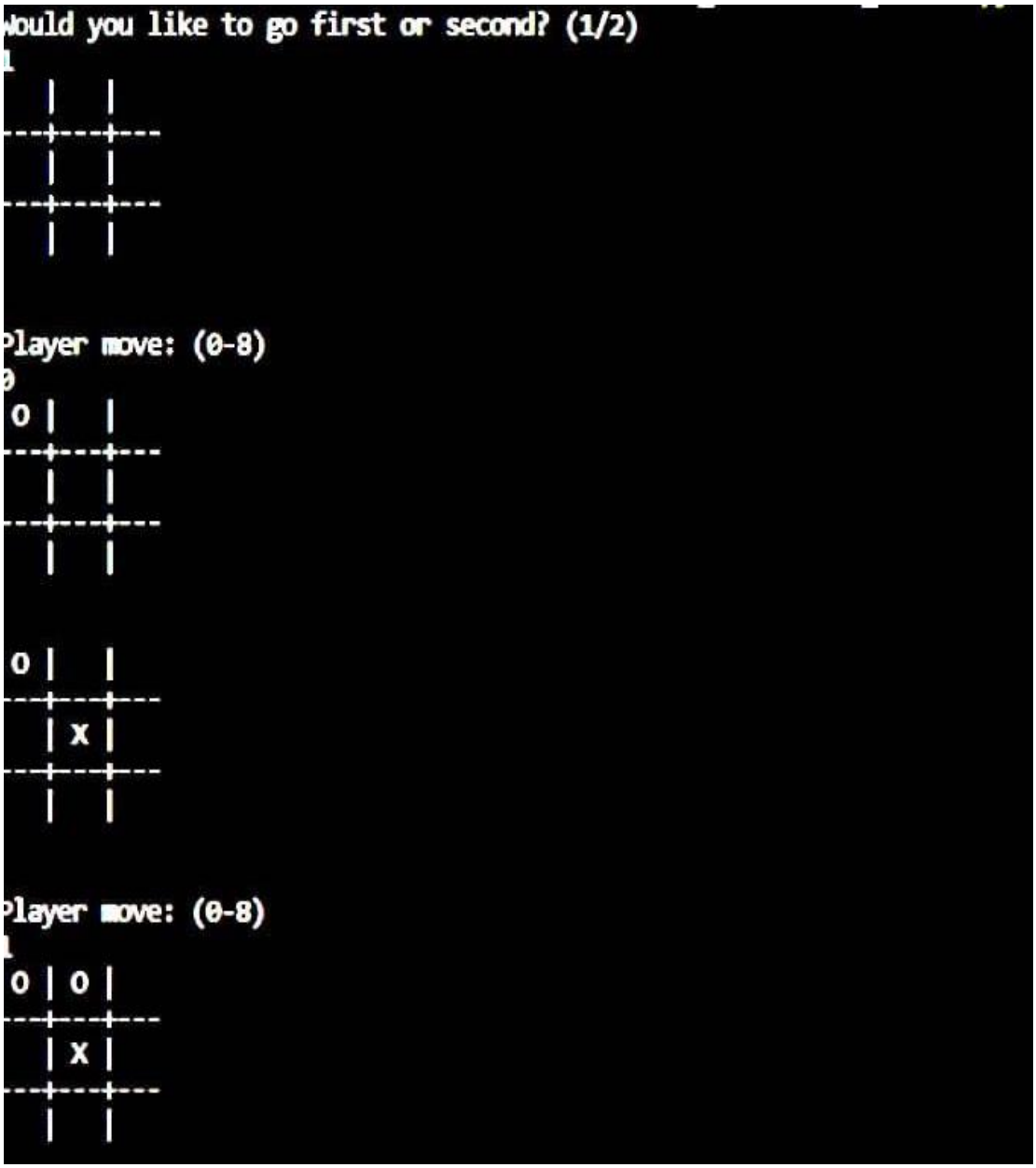
```
     |   |
     |   |
     |   |
- - - - - - - - -
```

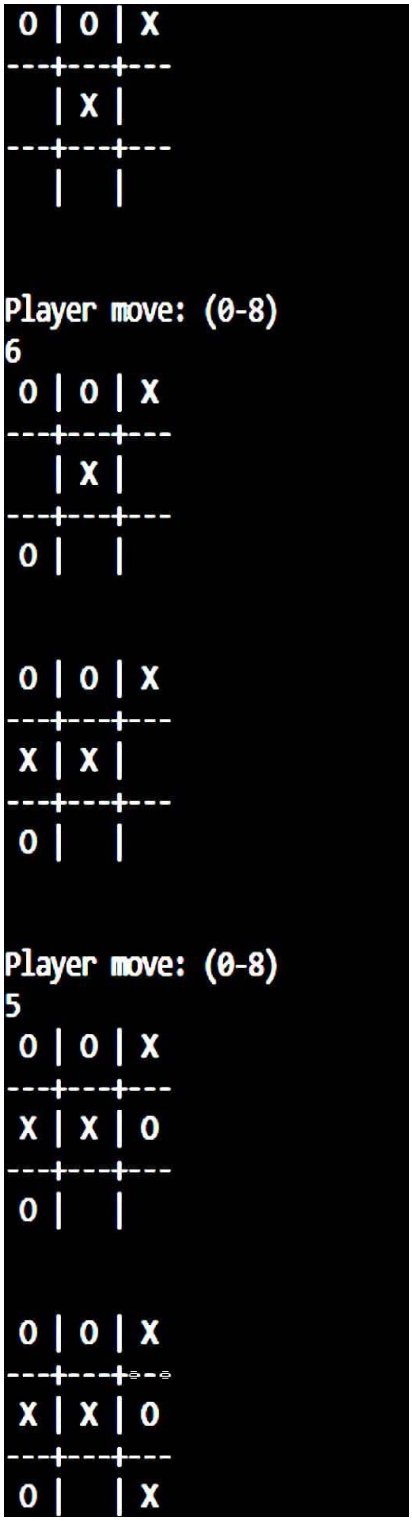
Please select a position: (0,1,2): 0

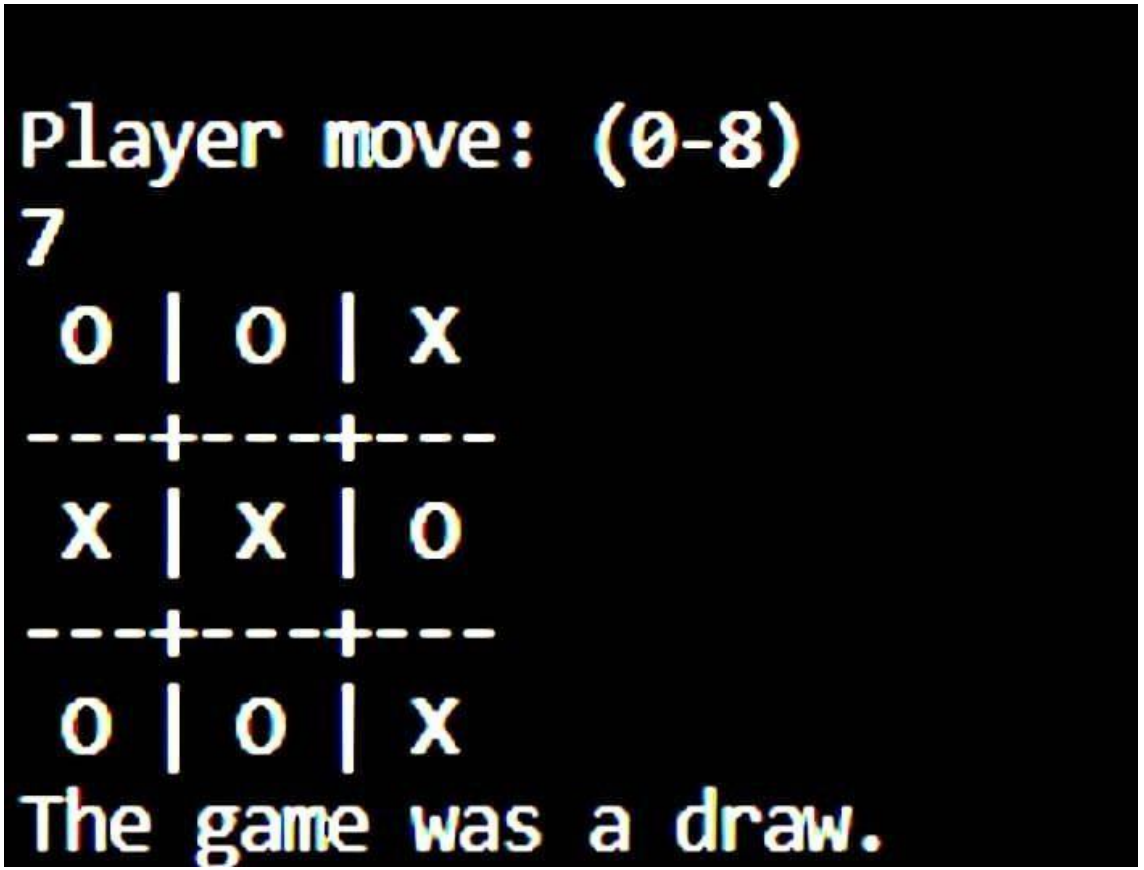
```
 0 X |   |
     |   |
     |   |
- - - - - - - - -
 X |   |
   | 0 |
   |   |
- - - - - - - - -
 X |   |
   | 0 |
   |   | X
- - - - - - - - -
 X |   |
   | 0 | 0
   |   | X
- - - - - - - - -
 X |   |
 0 | 0 | 0
   |   | X
```

Player '0' wins.

)(Y/N)?)



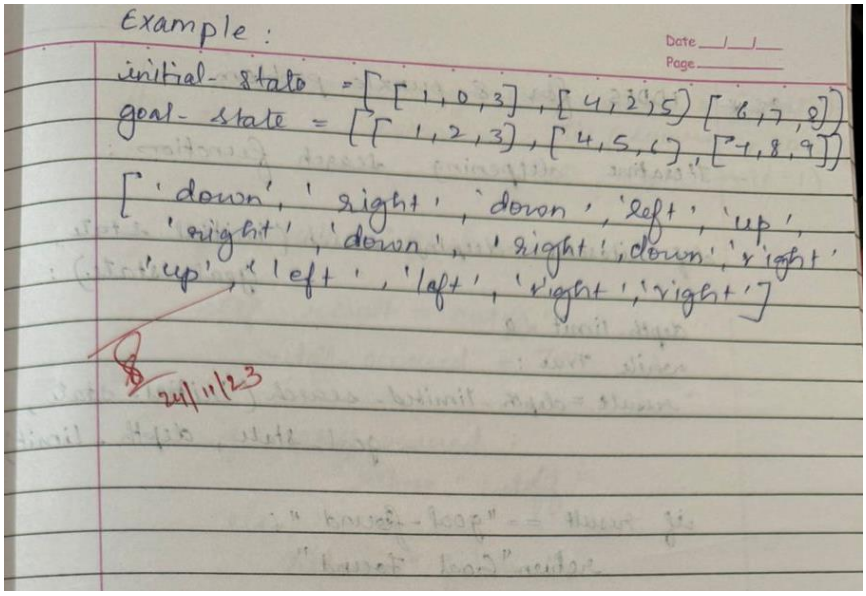




* Program - 02
8-block puzzle using breadth first search.

Algorithm:

- Define the puzzle as an array 3x3 where ~~alphabets~~ numbers represent the puzzle elements and 0 is misplaced block.
- Identify the vacant space(0) and move the elements →, ←, ↑ and ↓
- Create a new grid by moving the space. to valid position.
- Check if current grid matches over user solution
- Using BFS:
  Queue is exploring possible moves.
  Start with first block to see if it matches the solution and finalise path.
- If not generate new entry, by moving blank space.
- Continue until soln achieved.
- If solution matches print the result.


Example :

Date __/__/__
Page _____

initial- state = [[1,0,3],[4,2,5][6,7,8]]
goal- state = [[1,2,3],[4,5,6],[7,8,9]]

['down', 'right', 'down', 'left', 'up',
'right', 'down', 'right', 'down', 'right',
'up', 'left', 'left', 'right', 'right']

8
24/11/23

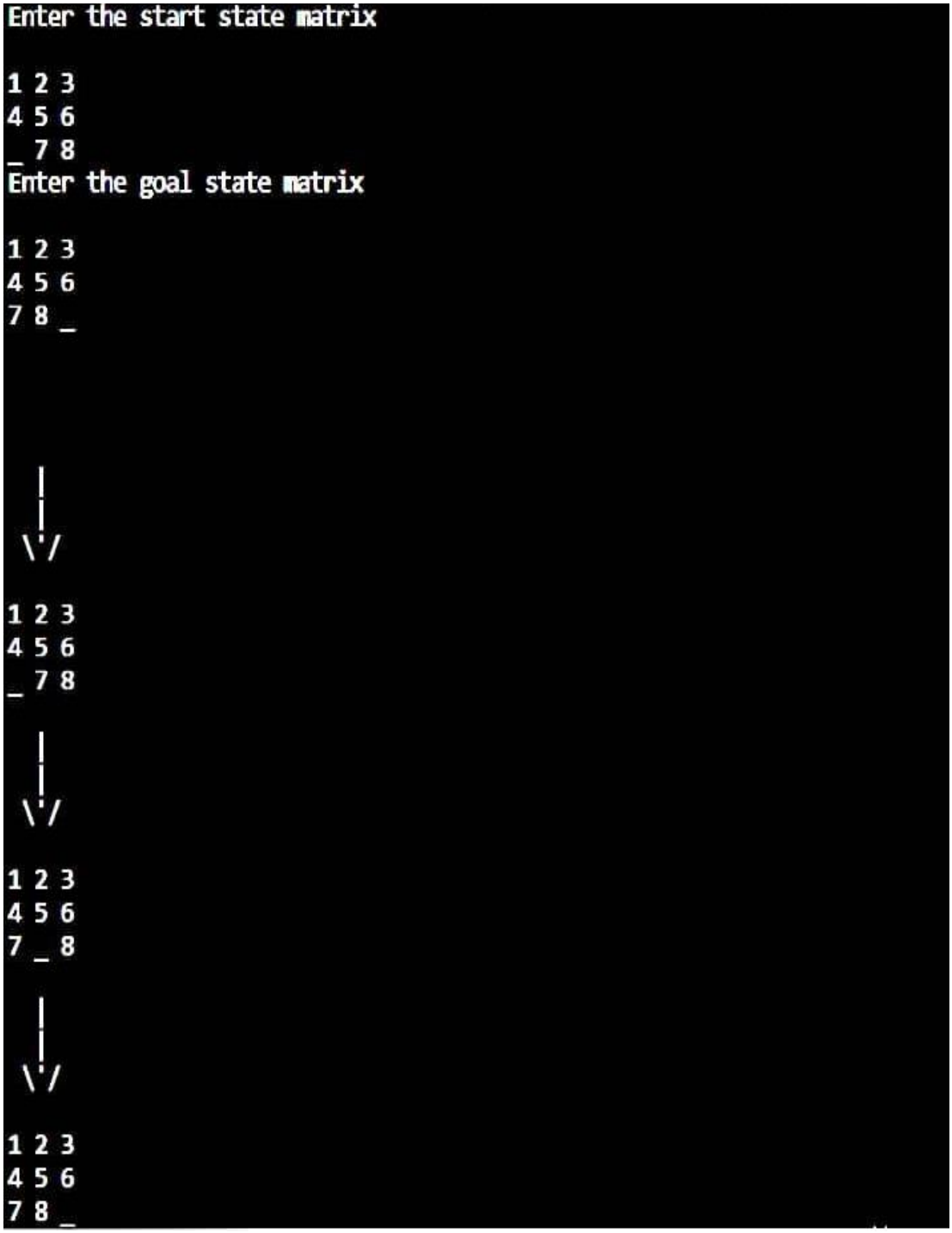
Scanned with OKEN Scanner

Q8 * IDDFS for 8 puzzle problem

- Iterative deepening search function:

```
def iterative_deeping_search(initial_state, goal_state):
    depth_limit = 0
    while True:
        result = depth_limited_search(initial_state, goal_state, depth_limit)

        if result == "goal_found":
            return "Goal Found"

        elif result == "cutoff":
            depth_limit += 1
        elif result == "failure"
            return "Goal Not Reachable"

def depth_limited_search(state, goal, depth_limit):
    return recursive_dls(state, goal_state, depth_limit)

def recursive_dls(state, goal_state, depth):
    if state == goal_state:
        return "goal_found"
    elif depth_limit == 0
        return "cutoff"
    else:
        cutoff_occured = false
```

```
for successor in generate-successor(state):
    result = recursive-dls(successor, goal-
                           state, depth_limit-1)

    if result == "goal-count":

    elif result = "cutoff"
        cutoff-occured = True.

    if cutoff ocurred:
            return "cutoff"
    else:
        return "failure"

def generate-successors(state):
pass

initial state = [[1,0,2]
                 [8,0,4]
                 [7,6,5]]

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

result = iterative-deeping-search(initial-
                                  state, goal-state)

print(result)
```



```
def is_valid_move(position):
    return 0 <= position[0] < 3 and
    0 <= position[1] < 3.

def move_tile(state, from_pos, to_pos):
    new_state = [row for row in state]
    x1, y1 = from_position
    x2, y2 = to_position
    new_state[x1][y1], new_state[x2][y2] =
    new_state[x2][y2], new_state[x1][y1]
    return
        new_state

def reconstruct_path(node):
    path = []

    while node.parent:
        path.append(node.state)
        node = node.parent
        path.append(node.state)
        path.reverse()
        return path

initial_state = [(1,2,3)
                 (8,0,4)
                 (7,6,5)]

goal_state = [(1,2,3)
              (8,0,4)
              (7,6,5)]

result = astar_search(initial_state,
                      goal_state)
print(result)
```

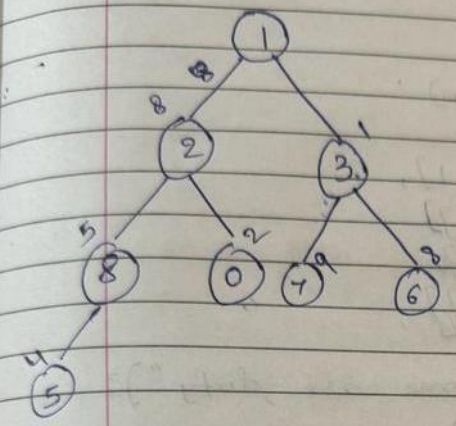
Date __/__/__
Page

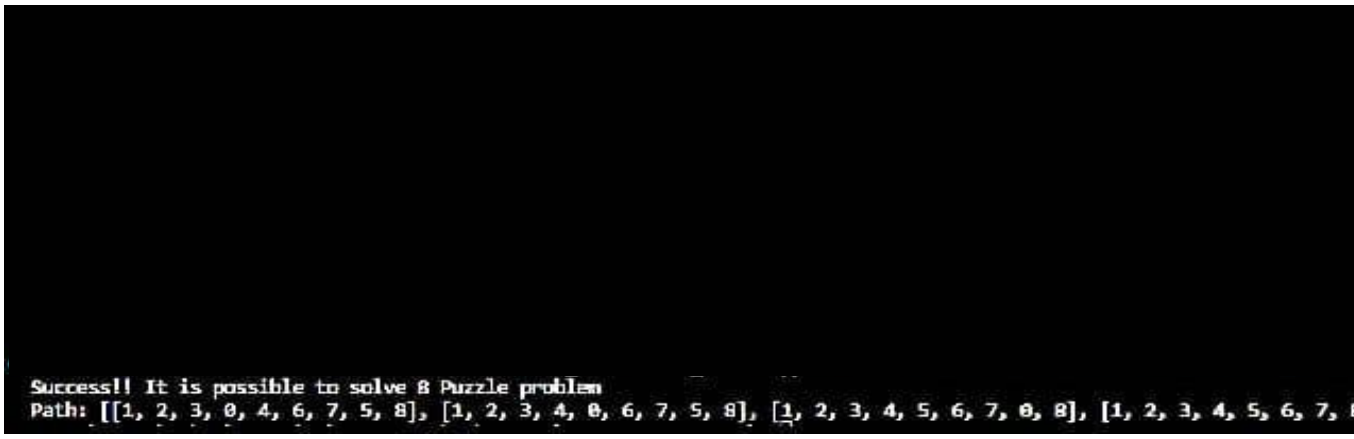
(Q9) A* Search algorithm : (8-Puzzle Problem)

```
class PuzzleNode :
    def __init__(self, state, parent=None,
                 move=None, cost=0, heuristic=0):
        self.state = state
        self.parent = parent
        self.move = move
        self.cost = cost
        self.heuristic = heuristic

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

def astar_search(initial_state, goal_state):
    open_set = [PuzzleNode(initial_state,
        None, None, 0, manhattan_distance(
            initial_state, goal_state))]

    closed_set = set()

closed_set.add(tuple(map(tuple, current_
                         node.state)))

for successor in generate_successors
    (current_node.state):
    if tuple(map(tuple, successor)) not in
        closed_set:
    heapq.heappush(open_set, PuzzleNode
    (successor, current_node, None, current
    node.cost + 1, manhattan_distance(successor,
    goal_state)))
return "Goal Not Reachable".
```

Output :

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





Scanned with OKEN Scanner

(Q6) A* algorithm for Vacuum Cleaner Problem.

```
import random
def display ( room) :
  print ( room)
  room = [ [1 , 1],
           [1 , 1]]
print ( "All rooms are dirty" )
    display ( room)
    x = 0
    y = 0

  while x < 2
  while y < 2
  room[x][y] = random. choice [ [0, 1])
    y += 1
    x += 1
    y = 0.

print ( "Before cleaning the room
      detect all of these random dirts:"

display ( room)
    x = 0
    y = 0
    z = 0
  while x < 2
  while y < 2
  if room [x][y] == 1 :
```

```
print ("Vacuum in this location now", x, y)
room [x][y] = 0
print ("Cleaned", x, y)
z += 1
x += 1
y += 1
y = 0

pro = (100 - (z/4) * 100)
print ("Room is clean, now")
display (room)
print ("performance = ", pro, "%").
```

Output:

All the rooms are dirty

[[1,1,1,1], [1,1]]

Before cleaning we detect the dirt randomly as:

[[0,1],[1,1]]

Vacuum in this location now, (0,0)
cleaned (0,0)

Vacuum in this location now, (1,0)
cleaned (1,0)

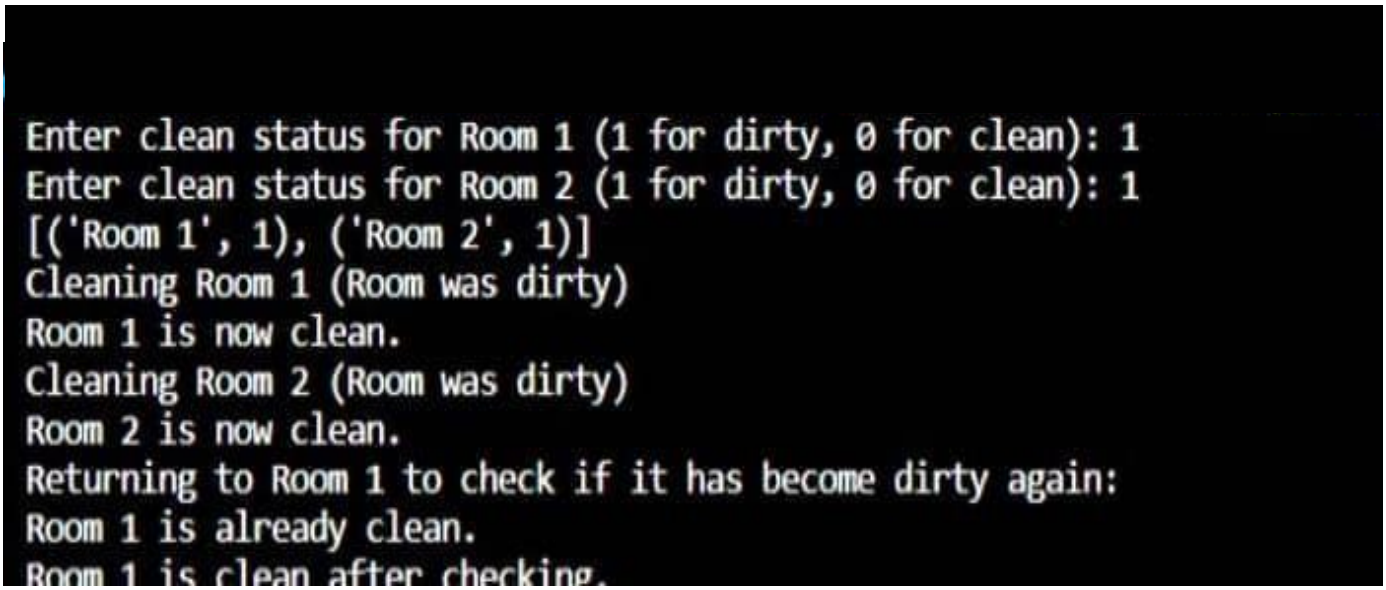
Vacuum in this location now (1,1)
cleaned (1,1)

Rooms are clean now

[[0,0],[0,0]]

Performance = 43.75%

22/12/23

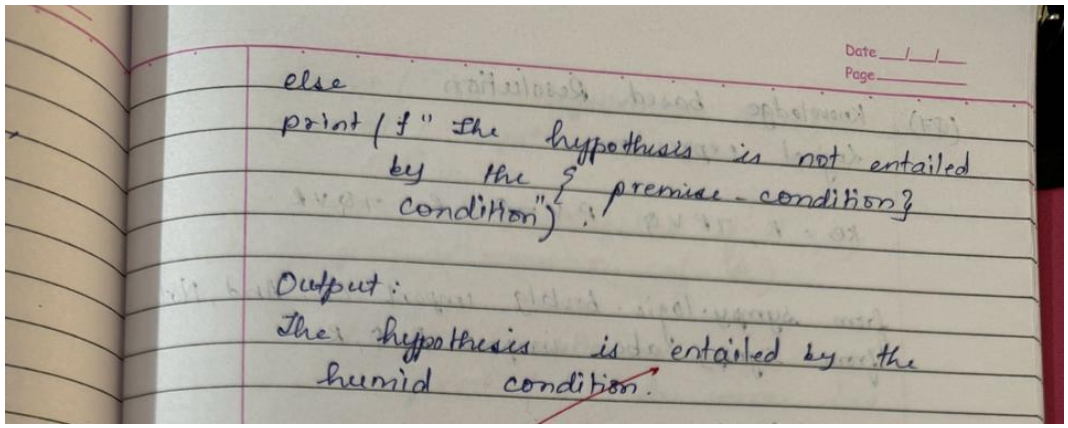


Scanned with OKEN Scanner

Date 29/12/23
Page

Q6) Knowledge based Entailment.

```
def weather_based_entailment(hypothesis, premise, a, b, c):
    if premise == "humid" and a > 70:
        return True
    elif premise == "cloudy" and b > 50:
        return True
    elif premise == "some_other_condition" and c > 30:
        return True
    else
        return False

premise_condition = "humid" or "cloudy" or "some_other_condition"

hypothesis_text = "The weather is uncomfortable"

humidity_condition = 75
cloudiness_condition = 60
some_other_condition = 40

result = weather_based_entailment(hypothesis_text, premise_condition, humidity_condition, cloudiness_condition, some_other_condition)
if result:
    print(f"The hypothesis is entailed by the {premise_condition} condition.")
```

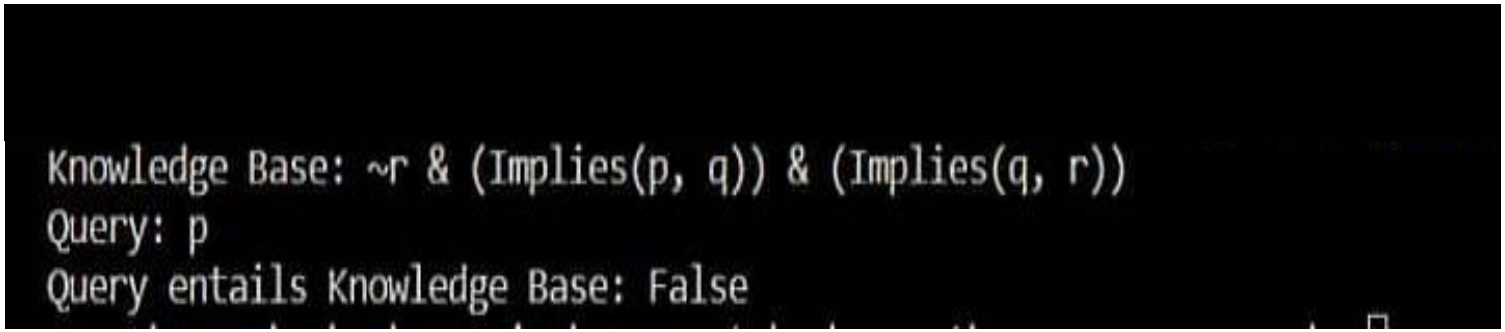


Date __/__/__
Page _____

```
else
print("The hypothesis is not entailed
      by the {premise_condition}")
```

Output:

The hypothesis is entailed by the humid condition.



(Q7) Knowledge based Resolution

Logical expression:

KB = P, ¬P ∨ Q, P ∨ ¬Q ∨ R, ¬Q ∨ R

```
from sympy.logic.boolalg import Or, And, Not
from sympy.abc import P, Q, R

def main():
    try:
        expression1 = P
        expression2 = Or(Not(P), Q)
        expression3 = Or(P, Not(Q), R)
        expression4 = Or(Not(Q), R)

        knowledge_base = And(expression1, expression2,
                             expression3, expression4)

        print("Knowledge Base:")
        print(knowledge_base)

        resolved_kb = knowledge_base.simplify()

        print("\n Resolved Knowledge Base:")
        print(resolved_kb)

        negation_of_P = Not(P)
        negation_of_Q = Not(Q)
        negation_of_R = Not(R)

        print("\n Negation of P:")
        print(negation_of_P)
```

```
print ("\n Negation of Q:")
print (negation_of_Q)
print ("\n Negation of R :")
print(negation_of_R)

except Exception as e:
    print(f"An error has
          occured : {e}:")

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

Output:

Knowledge Base:

P & (Q | ~P) & (R | ~Q) & (P | R | ~Q)

Resolved knowledge base :

True

Negation of P:

~P

Negation of Q:

~Q

Negation of R

A ∧ A = A

1) (P ∧ (¬P ∨ Q) ∧ (P ∨ ¬Q ∨ R) ∧ (¬Q ∨ R))

A ∨ A = A

2) (P ∧ (¬P ∨ Q) ∧ (P ∨ ¬Q ∨ R) ∧ (¬Q ∨ R))

3) (P ∧ (¬P ∨ Q) ∧ (P ∨ ¬Q ∨ R) ∧ (¬Q ∨

4) (P ∧ ¬P ∨ R ∧ Q ∧ P ∨ R ∧ ¬Q ∨

```
79    rules = 'Rv~P Rv~Q ~RvP ~RvQ' #(P^Q)<=>R : (Rv~P)v(Rv~Q)^(~RvP)^(~RvQ)
80    goal = 'R'
81    main(rules, goal)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Step     |Clause |Derivation
---------------------------
 1.      | Rv~P  | Given.
 2.      | Rv~Q  | Given.
 3.      | ~RvP  | Given.
 4.      | ~RvQ  | Given.
 5.      | ~R    | Negated conclusion.
 6.      |       | Resolved Rv~P and ~RvP to Rv~R, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
```

(Q8) Implement unification in first order logic

```
def unify-var ( var, x, theta)
    if var in theta :
        return unify ( theta[var], x, theta)

    elif x in theta :
        return unify ( var, theta[x], theta)

    else :
        theta [var] = x
        return theta

def unify ( x, y, theta = {} ) :

if theta is None :
    return None

elif x == y
    return theta
elif isinstance (x, str) and x(0). islower():

return unify-var ( x, y, theta)

elif isinstance ( y, str) and y[0]. islower():
    return unify-var ( y, x, theta)

elif isinstance ( x, list) and isinstance (y, list) :
    if ( len(x) ! = len(y) :
        return None

for xi, yi in zip ( x, y) :
    theta = unify ( xi, yi, theta)
```

```
    if theta is None :
        return None
    return theta
else :
    return None

x = ['p', 'a', 'x']
y = ['p', 'y', 'z']
result = unify (x, y)
print (result)
```

* Output :

Expression 1 : ['p', 'a', 'x']
Expression 2 : ['p', 'y', 'z']

Ouput :
Unification Successful!
Substitution theta : {'x': 'z', 'y': 'a'}

```
exp1 = "knows(A,x)"
exp2 = "knows(y,Y)"
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

anner

Date __/__/__
Page

(Q) Implement a given first-order logic statement into Conjunctive Normal Form (CNF)

```
from sympy import symbols, to_cnf,
    parse_expr.

def convert_to_cnf (logic statement):
parsed_statement = parse_expr (logic
                                statement)
    cnf = to_cnf (paised_element)

    return cnf

if __name__ = "__main__"

    logic statement = "(p|~q) & (~p |r)"

cnf_result = convert_to_cnf (logic_statement)

print ("Original Statement : ", logic_statement)
print ("CNF form : ", cnf_result)
```

OUTPUT :
Original Statement : (A & B) | (~C & D)
CNF form : (A | ~C) & (A | D) & (B | ~C)
& (B | D)

(Q) Create
order
quer

from

John

Paren
Gra

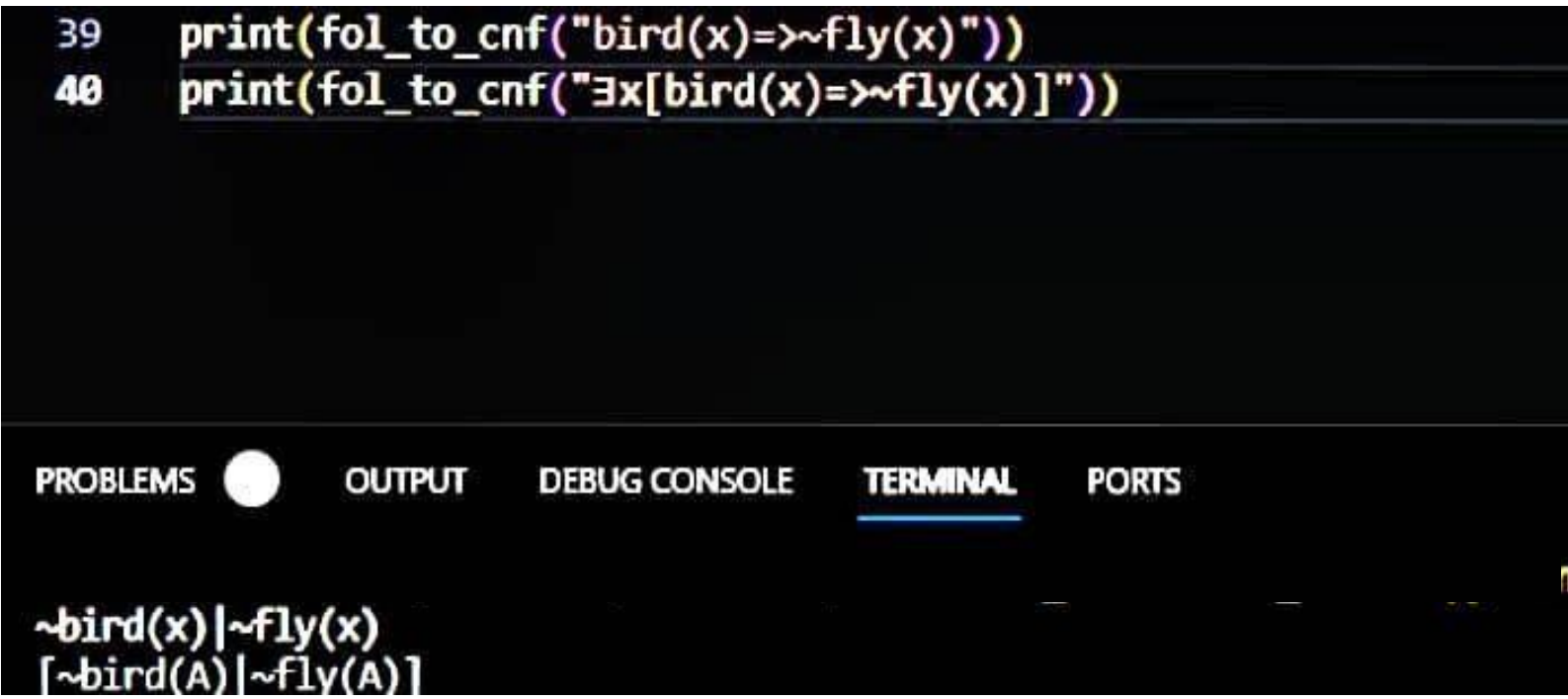
Know

]

que

def

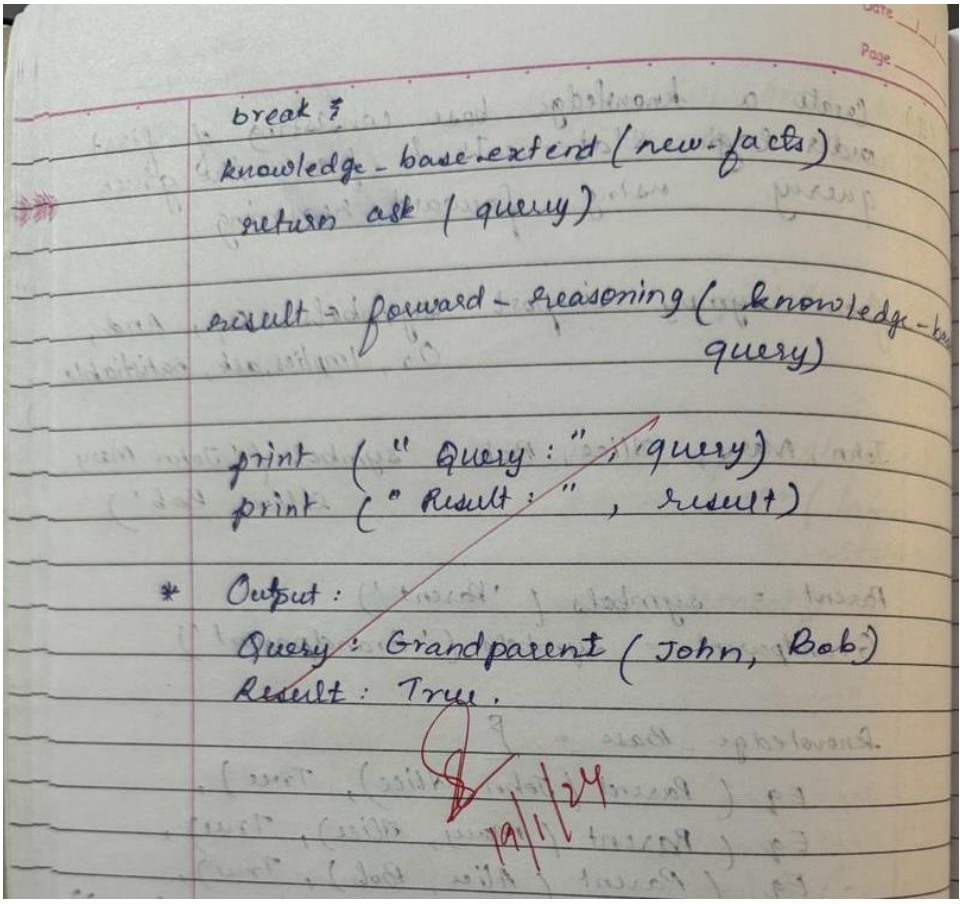
cur



Date __/__/__
Page _____

(9) Create a knowledge base consisting of first order logic statements & prove the given query using forward reasoning.

```
from sympy import symbols, Eq, And,
                 Or, Implies, ask, satisfiable

John, Mary, Alice, Bob = symbols ('John Mary
                          Alice Bob')

Parent = symbols ('Parent')
Grandparent = symbols ('Grandparent')

Knowledge_Base = [
    Eq (Parent (John, Alice), True),
    Eq (Parent (Mary, Alice), True),
    Eq (Parent (Alice, Bob), True),
    Implies (Parent (x, y), Grandparent (x, y)),
]

query = Grandparent (John, Bob)

def forward_reasoning (knowledge_base,
                       query):
    new_facts = set()

    while True:
        for fact in knowledge_base:
            if ask (fact):
                continue

            if satisfiable (fact):
                new_facts.add (fact)
        if not new_facts:
```



```
break ;
knowledge_base.extend(new_facts)
return ask(query)

result = forward_reasoning(knowledge_base,
                           query)

print("Query:", query)
print("Result:", result)
```

* Output:

Query: Grandparent(John, Bob)
Result: True.

19/1/24

```
kb = KB()
kb.tell('missile(x)=>weapon(x)')
kb.tell('missile(M1)')
kb.tell('enemy(x,America)=>hostile(x)')
kb.tell('american(West)')
kb.tell('enemy(Nono,America)')
kb.tell('owns(Nono,M1)')
kb.tell('missile(x)&owns(Nono,x)=>sells(West,x,Nono)')
kb.tell('american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)')
kb.query('criminal(x)')
kb.display()
```

```
Querying criminal(x):
        1. criminal(West)
All facts:
        1. missile(M1)
        2. weapon(M1)
        3. enemy(Nono,America)
        4. owns(Nono,M1)
        5. hostile(Nono)
        6. criminal(West)
        7. american(West)
        8. sells(West,M1,Nono)
```