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:

Harini N
1BM21CS256

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 Harini N (1BM21CS256) 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

venus
DATE / /20

Q1) Write a python programme using if-else statement for age

Q2) Print integers in reverse order

Q3) i) Print pattern 1, 2, 2, 3, 3, 3, 4, 4, 4, 4
5, 5, 5, 5, 5, upto - n.

ii) 1, 1, 2, 1, 2, 3, 1, 2, 3, 4, . . .

Q4) Power of a number → x^n

Q5) switch case problem.

1)
```
age = int (input ("enter your age: "))

if age < 0:
    print ("Invalid age")
elif age < 18:
    print ("You are a minor")
elif age < 65:
    print ("You are an adult")
else:
    print("You are a senior citizen")
```

2)
```
num = int (input ("enter the number to be reversed"))

reverse = 0

while number != 0:
        digit = number % 10
        reverse = reverse * 10 + digit
        number = number // 10
print (reverse)
```

Output:
Enter the number to be reversed: 1234
4321.

Q3) a) n = int(input("Enter the value of n"))

```
for i in range(1, n+1):
    for j in range(i):
        print(i, end=' ')
```

Output:
1 2 2 3 3 3 4 4 4 4 5 5 5 5-5

b) n = int(input("Enter the value of n"))

```
for i in range(n+1):
    for j in range(1, i+1):
        print(j, end=' ')
```

Enter the value of n: 4
1 1 2 1 2 3 1 2 3 4

Q4) power = int(input("Enter the power"))
base = int(input("Enter the base"))

result = base ** power

print(f'The power is {result}')

(Q5)

```
lang = int (input ("Language"))

match lang:
    case "Java":
        print (" You can become a coder")
    case "PHP"
        print ("Backend")
    case "solidity"
        print ("Blockchain")
    case _:
        print ("default")
```

8/10/11/12

Tic-Tac-Toe game

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

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

    def printBoard board
    def ~~spaceIsFree~~(~~pos~~):
        print('   |   |')
    ~~print~~ print('' + board[1] + '|' +
                    board[2] + '|' +
                    board[3])
        print('   |   |')
        print('------------')
        print('   |   |')
        print('' + board[4] + '|' +
                  board[5] + '|' +
                  board[6])
        print('   |   |')
        print('------------')
        print('   |   |')
        print('' + board[7] + '|' +
                  board[8] + '|' +
                  board[9])
        print('   |   |')
```

```
def main():
    print('Welcome to Tic Tac Toe!')
    printBoard(board)

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

    if not(isWinner(board,'X')):
        move = compMove()

    else:
        insertLetter('O', move)
        print(position, move, ':')
        printBoard(board)

    else:
        print('X's won this time! Good Job!')
        break

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

Algorithm

1. Initialize the board
   board = [' ' for in range(10)]

2. Insert a letter ('x' or 'O') into the board
   def insertLetter(letter, pos):

3. Check if space one the board is free:

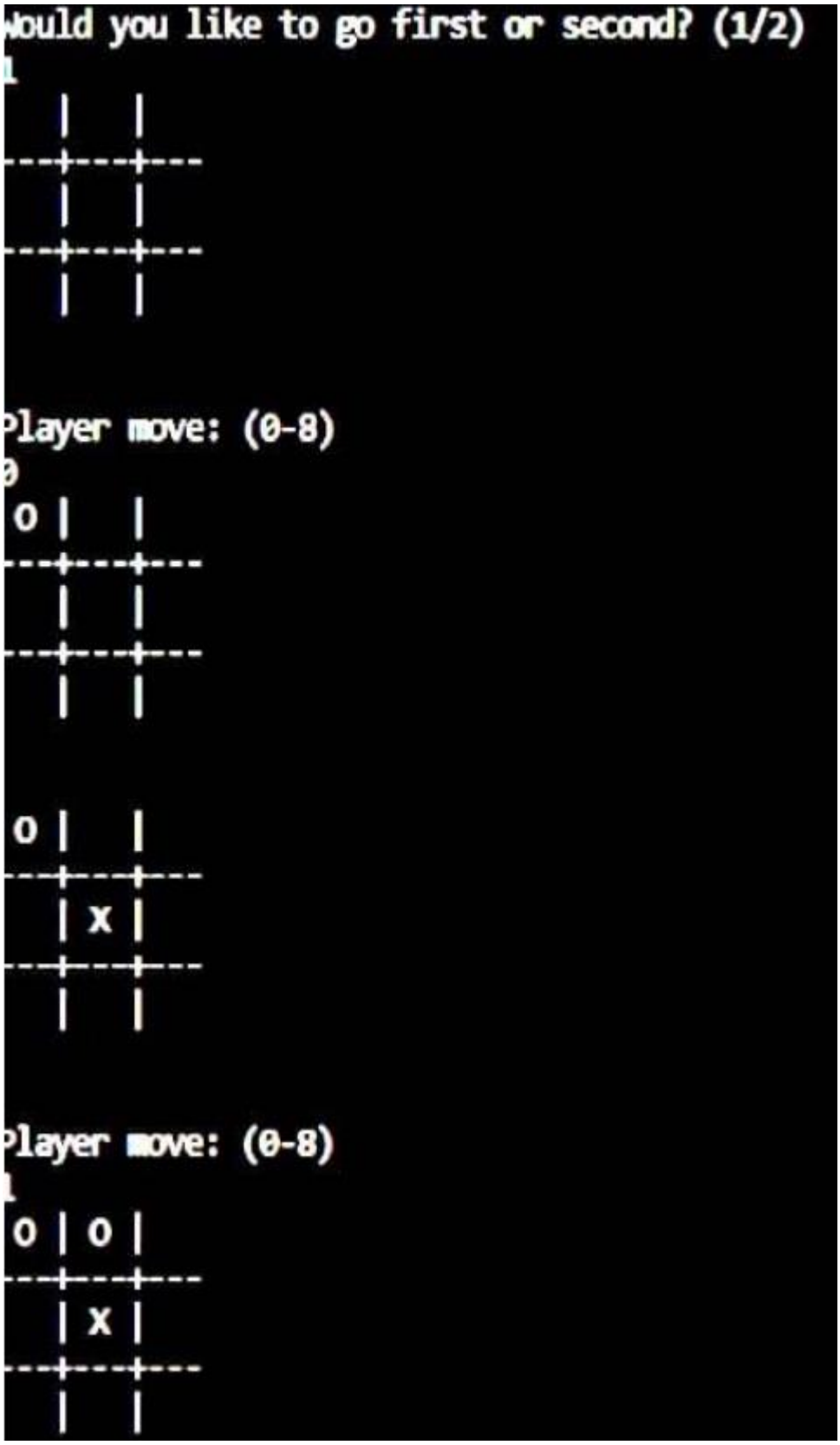
   def spaceIsFree(pos):
       return board[pos] == ' '

4. print the board def printBoard(board):

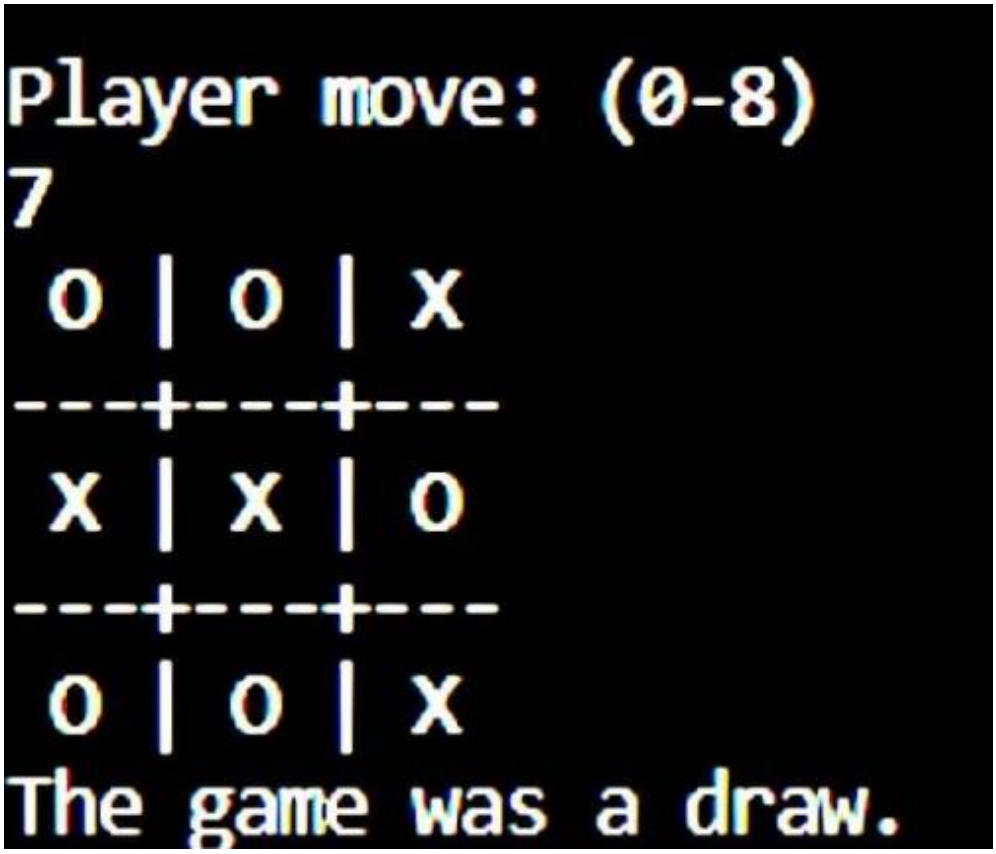
5. Check for a winner.
   def isWinner(bo, le):

6. Player's move
   def playerMove():
   Ask the player to choose a position to place their 'X' on the board.

7. Computer's move
   Implement the logic for the computers move
   def compMove


Would you like to go first or second? (1/2)
1
Player move: (0-8)
0
Player move: (0-8)
1


O | O | X
---+---+---
| X |
---+---+---
| |
Player move: (0-8)
6
O | O | X
---+---+---
| X |
---+---+---
O | |
O | O | X
---+---+---
X | X |
---+---+---
O | |
Player move: (0-8)
5
O | O | X
---+---+---
X | X | O
---+---+---
O | |
O | O | X
---+---+---
X | X | O
---+---+---
O | | X


Player move: (0-8)
7
O | O | X
---+---+---
X | X | O
---+---+---
O | O | X
The game was a draw.

venus
DATE / /20

Lab-3

TO Implement 8-Puzzle Game Using BFS

1. Initialize Puzzle:
   → Create instance of Puzzle8 class.
   → The initial board is generated randomly.

2. Loop until Puzzle is solved.
   → ~~while~~ Display the current board state.
   → The user inputs up/↓/→/←.
   → Move function does this move

3. In Move fn:
   → Find idx of blank tile (0).
   → Move accordingly.
   → If move is invalid, then print error.

4. If Puzzle is solved, then is_solved fn is used to compare current board ~~state~~ with goal
   If they match, puzzle is solved.
   ~~The~~ If not loop will continue

5. Once it is solved display the final ans.

I/P:

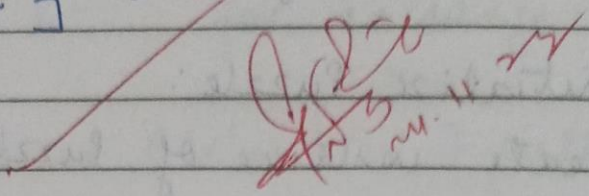
1	2	3
4	5	0
6	7	8

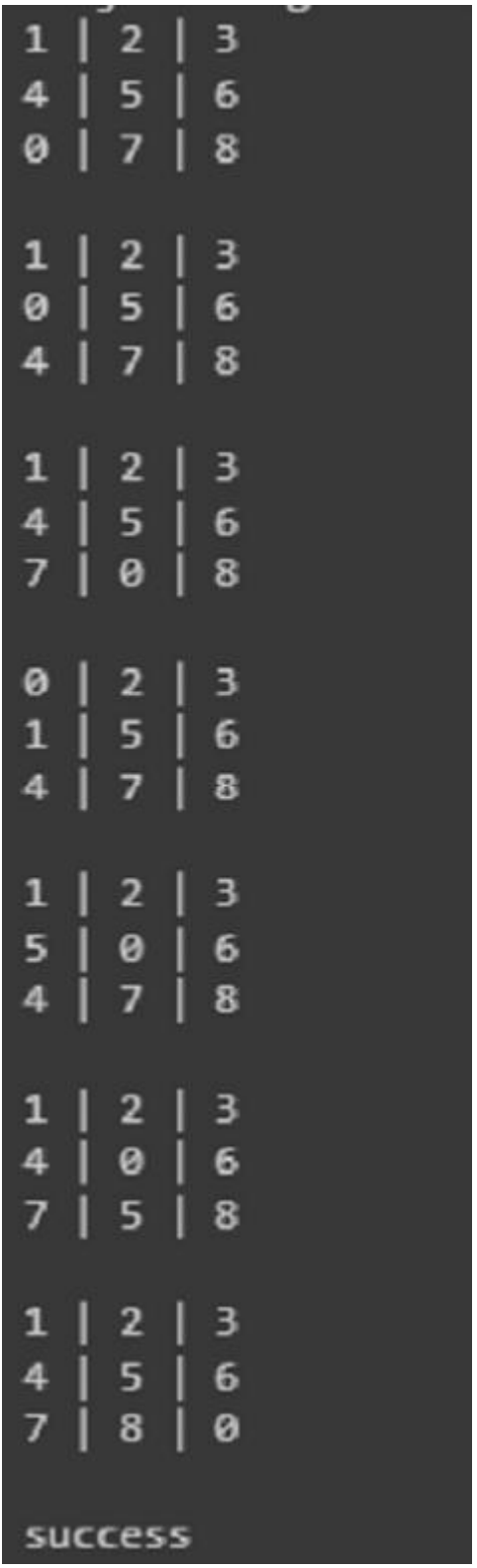
O/P:

1	2	3
4	5	6
7	8	9

O/P:

The moves are [down, left, left, up right, down, right, up, left, down, right, right]




1 | 2 | 3
4 | 5 | 6
0 | 7 | 8
1 | 2 | 3
0 | 5 | 6
4 | 7 | 8
1 | 2 | 3
4 | 5 | 6
7 | 0 | 8
0 | 2 | 3
1 | 5 | 6
4 | 7 | 8
1 | 2 | 3
5 | 0 | 6
4 | 7 | 8
1 | 2 | 3
4 | 0 | 6
7 | 5 | 8
1 | 2 | 3
4 | 5 | 6
7 | 8 | 0
success

3) 8 - Puzzle Iterative deepening search algorithm.

1. Define PuzzleNode class:
→ Each node represents a state of the puzzle.
→ The state is a 3x3 grid.
→ Store the parent node, the action taken to reach the current state, the depth, and other necessary information.

2. Define helper functions
get blank position (state)
get neighbours (node)
apply_action (state, action)
print solution (solution)

Step 3: Depth limited search function
depth limited search (node, goal state, depth, limit)

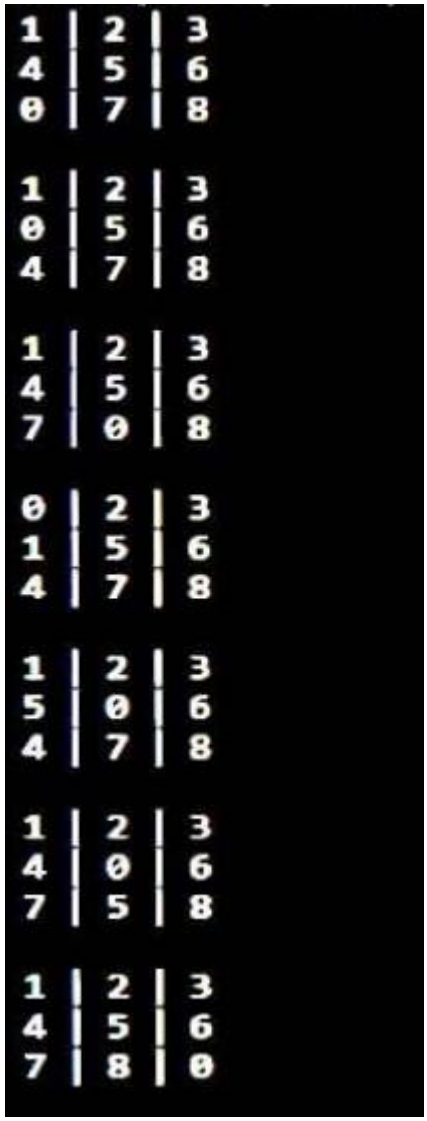
Step 4: Iterative deepening search function
iterative_deepening search [initial, state, goal state].

Step 5: Main function.

```
import copy

GOAL_STATE = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]
MOVE_UP, MOVE_DOWN, MOVE_LEFT, MOVE
RIGHT = 'U', 'D', 'L', 'R'
MOVES = [MOVE_UP, MOVE_DOWN, MOVE_LEFT,
         MOVE_RIGHT]

def print_puzzle(state):
    for row in state:
        print(row)
    print()

def find_blank(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j

def move(state, direction):
    i, j = find_blank(state)
    new_state = copy.deepcopy

def is_goal(state):
        return state == GOAL_STATE

def depth_limited_search(state, depth, path):
    if depth == 0
        return None
```

```
if is_goal(state):
    return path

for move_direction in MOVES:
    newstate = move(state, move_direction)
    if new_state not in path:
        new_path = path + [new_state]
        result = depth_limited_search
        (newstate, depth-1, newpath)

return None

def iddfs iterative_deepeing_search(initial state
    depth = 0
    while True:
        result = depth_limited_search(
        initial state, depth, (initial state])
        if result is not None:
            return result
            depth = 1

initial state = [[1,2,3], [0,8,4], [7,6,5]]
```

8/12/23


1 | 2 | 3
4 | 5 | 6
0 | 7 | 8
1 | 2 | 3
0 | 5 | 6
4 | 7 | 8
1 | 2 | 3
4 | 5 | 6
7 | 0 | 8
0 | 2 | 3
1 | 5 | 6
4 | 7 | 8
1 | 2 | 3
5 | 0 | 6
4 | 7 | 8
1 | 2 | 3
4 | 0 | 6
7 | 5 | 8
1 | 2 | 3
4 | 5 | 6
7 | 8 | 0

4) 8 Puzzle A* search algorithm.

1) Define PuzzleNode class
→ Each node represent a state of the puzzle
→ The state is a 3×3 grid.

2) Define PuzzleNode methods.
calculate heuristic(): calculate the heuristic value for the current state
__lt__(self, other)

3. Define helper fns.
get_blank_position(state)
find the position of the blank (0)
get_neighbours(node)
get valid neighbours for a given table
apply_action(state, action)
Apply the given action

Step 4: A search algorithm *
Initialize the initial state as a puzzlenode
" a set to store explored state

Step 5: Print solution
Trace back from goal state to initial state

Step 6: Main function
Define the initial state of puzzle call the A* algorithm with the initial state

DATE 7/20

Code:

```
class Node:
    def __init__(self, data, level, fval):
        """ Initialize the node with the data,
        level of the node with the data,
        level of the node and the calculated
        f value """
        self.data = data
        self.level = level
        self.fval = fval.

def generate_child(self):
    """ Generate child nodes from the give node
    by moving the blank space either in the
    four directions {up, down, left,
    right} """
    x, y = self.find(self.data, x, y,
                     i[0], i[1])

if child is not None:
    child_node = Node(child, self.level+
                      1, 0)

def shuffle(self, puz, x1, y1, x2, y2)
    """ Move the blank space in the given
    direction"
    if x2 >= 0 and x2 < len(self.data)
    and y2 >= 0 and y2 < len(self.data)

def copy(self, root)
    """ Copy function to create a
    similar matrix of the given node"""
```

```
temp = []

def process(self):
    """Accept start & goal puzzle state"""
    print("Enter the start state matrix")
    start = self.f(start, goal)
    while True:
        cur = self.open[0]
        print("  |")
        print("  |")
        print("  |")
        for i in cur.data:
            for j in i:
                print(j, end=" ")
            print("")

puz = Puzzle(3)
puz.process()
```

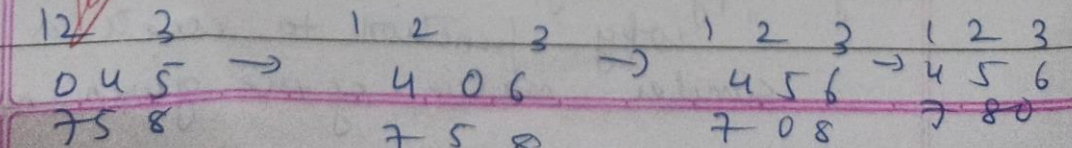
O/P:

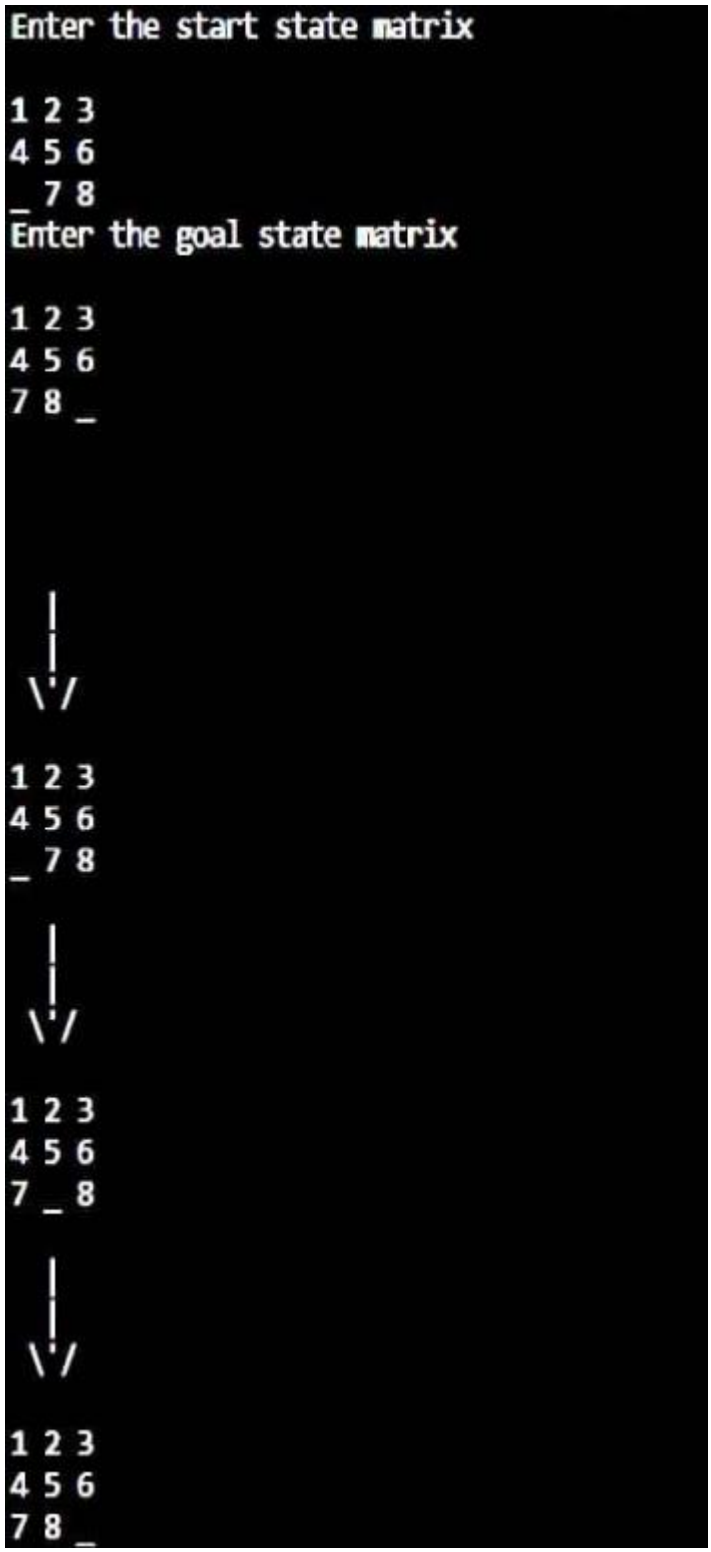
Enter the start state matrix

1 2 3
0 4 6
7 5 8

Enter the goal state matrix

1 2 3
4 5 6
7 8 0




Enter the start state matrix
1 2 3
4 5 6
_ 7 8
Enter the goal state matrix
1 2 3
4 5 6
7 8 _
|
|
\'/
1 2 3
4 5 6
_ 7 8
|
|
\'/
1 2 3
4 5 6
7 _ 8
|
|
\'/
1 2 3
4 5 6
7 8 _

Vacuum cleaner problem

Algorithm:

function Reflex_Vacuum_Agent ([location, status])
returns
action
if status = Dirty then return suck
else if location = A Then return Right
else if location = B then return left

Code:

```
def clean(floor):
    i, j, row, col = 0, 0, len(floor), len(floor[0])
    for i in range(row):
        if (i % 2 == 0):
            for j in range(col):
                if floor[i][j] == 1:
                    print_floor(floor, i, j, "Right" if
                        j < col-1 else "Stay")
        else:
            for j in range(col-1, -1, -1):
                if floor[i][j] == 1:
                    print_floor(floor, i, j, "clean")
                    floor[i][j] = 0
                print_floor(float, i, j, "Left"
                    if j > 0 else "stay").

def print_floor(floor, row, col, direction):
    print(f"The Floor matrix is as below: ")
```

```
for r in range(len(floor)):
    for c in range(len(floor[r])):
        if r==row and c==col:
            print(f">{floor[r][c]}<", end=' ')
        else:
            print(f" {floor[r][c]} ", end=' ')
    print(end='\n')
print(f"{direction}")
print(end='\n')

def main():
    floor = []
    m = int(input("Enter the No. of rows:"))
    n = int(input("Enter the No. of columns:"))
    print("Enter clean status for each cell
          (1-dirty, 0-clean)")
    for i in range(m):
        f = list(map(int, input().split(" ")))
        floor.append(f)
    print()
    clean(floor)

main()
```

o/p:

```
Enter No. of rows = 1
      No. of columns = 2
Clean status for each cell (1-dirty, 0-clean)
1 1
```

Floor matrix is as below:

>1< 1

clean

>0< 1

Right

0 >1<

clean

0 >0<

Stay

22/12/23

```
Enter clean status for Room 1 (1 for dirty, 0 for clean): 1
Enter clean status for Room 2 (1 for dirty, 0 for clean): 1
[('Room 1', 1), ('Room 2', 1)]
Cleaning Room 1 (Room was dirty)
Room 1 is now clean.
Cleaning Room 2 (Room was dirty)
Room 2 is now clean.
Returning to Room 1 to check if it has become dirty again:
Room 1 is already clean.
Room 1 is clean after checking.
```

Knowledge Base Entailment

Algo:

1. Tokenization
   - Tokenize both sentence 1 & 2

2. Normalization:
   Convert words to lowercase
   Remove punctuation

3. Comparison:
   Initialize bool var entailmentHolds, to true.
   For each word in sentence 1.
   Check if word is present in set of normalized words in sentence 2.
   If any word is not found, set entailmentHolds to false, exist the loop & break.
   → Return the value of entailmentHolds as the result

4. End:
   End the algorithm.

Code:

```
from sympy.logic.boolalg import Implies, Not
from sympy.abc import p, q

class PropositionalKnowledgeBase:
    def __init__(self):
        self.knowledge_base = set()

def add_statement(self, statement):
    self.knowledge_base.add(statement)
```

```
def check-entailment (self, query):
    return query.simplify() in self.knowledge_base

Kb= PropositionalKnowledgeBase()

Kb.add_statement(Implies(p, q))
Kb.add_statement(Not(q))

query 1 = p
query 2 = Not(p)

result 1 = Kb.check_entailment(query 1)
result 2 = Kb.check_entailment(query 2)
```

Output:

Does 'p' logically follow from the knowledge base : True

Does '~p' logically follow from the knowledge base : True

```
Knowledge Base: ~r & (Implies(p, q)) & (Implies(q, r))
Query: p
Query entails Knowledge Base: False
```

Knowledge Base Resolution

Algo:

1. Initialize Knowledge Base.
Create a class knowledge base to represent the knowledge base.

Populate the knowledge base with a set of predefined questions & answers.

2. User interaction loop.
Enter a loop to interact with the user.

Prompt the user to ask a question or type exit to end the program.

3. User input
Receive user input for the question

4. Knowledge resolution
Uses the resolve-query method in the 'KnowledgeBase' class to find the answer to the users question.

5. If an answer is found, display the answer.

6. Exit.

venus
DATE / /20

Code

```
def resolution (Knowledge_base, query):
    clauses = Knowledge_base + [frozenset ({~
                                query})]

while True:
    new_clauses = set()

    for i in range (len (clauses)):
        for j in range (i+1, len (element)):
            resolvents = resolve (clauses[i],
                                  clauses[j])

        if frozenset () in resolvents:
            new_clauses.update (resolvents)

    if new_clauses.issubset (clauses)
        return false

    clauses.update (new_clauses)

def resolve (ci, cj):
    resolvents = set()

    for li in ci:
        for lj in cj:
            if li == ~lj or ~li == lj
            resolvent = (ci - {lj}) | (cj - {lj})
            resolvents.add (frozenset (resolvent))
    return resolvents
```

```
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.
```

Unification for first order logic

Algorithm:

1. Initialize: start with an empty substitution 'theta = {}'.

2. Comparison of Atoms: Compare the 2 atoms ~~and~~
   The 1st element "p" are the sum
   The 2nd element "x" and 'A' are diff so we need to unify them.

3. Unify Variables

4. Recursive unification: Now, recursively unify the remaining the variables.

5. Result: The final ~~substitution~~ theta = {theta = {"x": "A", "y": "B"}}.

Code:

```
Class UnificationError (Exception):
    pass

def print step (step, atom1, atom 2, theta):
    print (f"\n step {step}:")
    print (f" Unifying {atom1} and {atom2}")
    print (f" current Substitution Theta: {theta}")
```

```
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(atom1, atom2, theta=None, step=1):
    if theta is None:
        theta = {}

print_step(step, atom1, atom2, theta)
    if atom

if atom1 == atom2:
    return theta
elif isinstance(atom1, str) and atom1.isalpha():
    return unify_var(atom1, atom2, theta)
elif isinstance(atom2, str) and atom2.isalpha():
    return unify_var(atom2, atom1, theta)

    if len(atom1) != len(atom2):
        raise UnificationError("lists have
                                different length")
    for a1, a2 in zip(atom1, atom2):
        raise UnificationError("lists have
                                diffenet lengths")

    for a1, a2 in zip(atom1, atom2):
        theta = unify(a1, a2, theta, step+1)
        return theta
else:
    raise UnificationError("cannot Unify")
```

try:
    theta = unify(["p", "x", "y"], ["p", "A", "B"])
    print("\n Unification successful. Substitution theta: ", theta)
except UnificationError as e:
    print("\n Unification failed:", e)

O/P:

Step 1:

Unifying ['p', 'x', 'y'] and ['p', 'A', 'B']
Current substitution Theta: {}

Step 2:

Unifying p and p
Current substitution Theta: {}

Step 3:

Unifying x and A
Current Substitution: {}

Step 4:

Unifying y and B
Current substitution theta: {'x': 'A'}

Unification successful. Substitution theta:
{'x': 'A', 'y': 'B'}

8/11/24


exp1 = "knows(A,x)"
exp2 = "knows(y,Y)"
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
python
Substitutions:
[('A', 'y'), ('Y', 'x')]

venus
DATE / /20

Convert a first order logic to conjuctive normal form

Algorithm

1. Eliminate implications
Replace implication with their equivalent forms using the rule $P \Rightarrow Q \equiv \neg P \vee Q$

2. Move negations inwards Apply De Morgan's law to move negation inwards.
For eg:
$\neg (P \wedge Q) \equiv \neg P \vee \neg Q$
$\neg (P \vee Q) \equiv \neg P \wedge \neg Q$

3. Distribute disjunctions over conjunctions
eg:
$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$

4. Convert to CNF form. Ensure that formula is in CNF by applying any additional simplifications if needed

Code:

```
from sympy import symbols And, Or, Implies
                Not, to-cnf

def eliminate-implications (formula):
    return formula.subs (Implies (P, Q),
                         Or (Not (P), Q))

def move-negations-inwards (formula):
    return formula.simplify ()
```

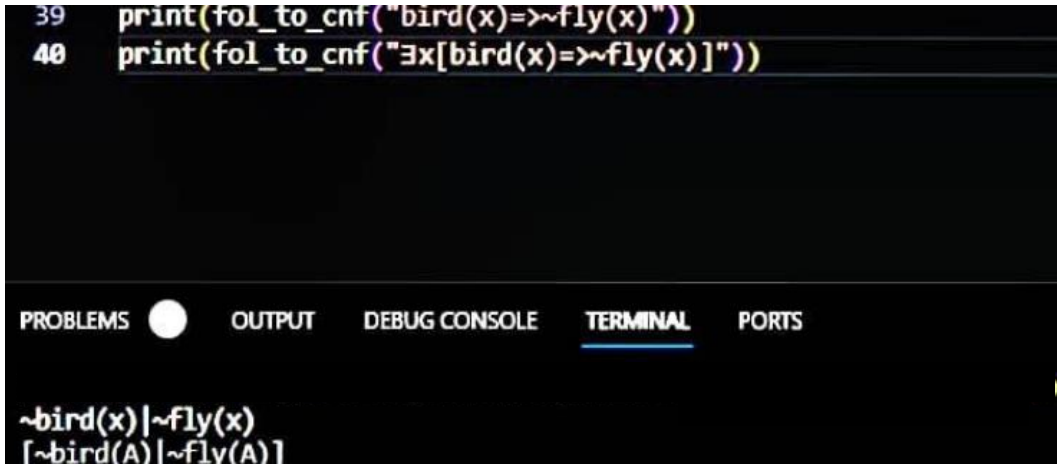
```
def distribute_disjunctions_over_conjunctions
    (formula):
    return formula.simplify()

def fol_to_cnf (fol_formula):
    formula_step1 = eliminate_implications
                    (fol_formula)

    formula_step2 = move_negations_inwards
                    (formula_step1)

    formula_step3 = distribute_disjunctions
                    over_conjunctions
                    (formula_step2)
    cnf_formula = to_cnf (formula_step3)

    return cnf_formula

user_input = input ("enter a first order
                    logic formula:")
P, Q, R = symbols ('P Q R')
fol_formula = eval (user_input, {'P': P
                    'Q': Q, 'R': R})

cnf_formula = fol_to_cnf (fol_formula)
print ("\n FOL formula", fol_formula)
print ("CNF formula:", cnf_formula)
```

Output:

Enter a first order logic: pf (~q|r)

FOL formula: And (P, or (Not (q), r))

CNF-formula: or (And (P, r) AND (P, ~q)) V (∧(P, r) ∧ (P, ~q)


print(fol_to_cnf("bird(x)=>~fly(x)"))
print(fol_to_cnf("∃x[bird(x)=>~fly(x)]"))
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
~bird(x)|~fly(x)
[~bird(A)|~fly(A)]

venus
DATE / /20

Create a Knowledge Base consisting of first order logic statements and prove the given query using forward reasoning

Algorithm:

1. Initialize Knowledge Base (KB)
start with an empty knowledge base

2. Add FOL statement to KB
Add relevant FOL statements represent facts & rules about the domain

3. Define forward reasoning rules specify the rules for forward reasoning

4. Initialize working memory
Create a working memory to store the current state

5. Ask Query

6. Forward reasoning loop.

Repeat the foll. steps until the query

a. Iterate through each rule

b. Apply rules when condition match the current state.

c. Add the conclusions

d.

7. check query in working memory

8. Output result.

Code:

```
class KnowledgeBase:
    def __init__(self):
        self.rules = []

    def add_rule(self, conditions, conclusion):
        self.rules.append({"conditions": set(conditions), "conclusion": conclusion})

    def forward_reasoning(self, query):
        working_memory = set()
        unchanged = False
        kb = KnowledgeBase()
        kb.add_rule(
        kb.add_rule(["P", "Q"], "R")
        kb.add_rule(["R", "S"], "T")
        kb.add_rule(["T", "U"], "V")

        query = "V"

result = kb.forward_reasoning(query)

if result:
    print("Query is true")
else:
    print("Query is false")
```

Output:

Query is false. 24/11/24

```
95    kb = KB()
96    kb.tell('missile(x)=>weapon(x)')
97    kb.tell('missile(M1)')
98    kb.tell('enemy(x,America)=>hostile(x)')
99    kb.tell('american(West)')
100   kb.tell('enemy(Nono,America)')
101   kb.tell('owns(Nono,M1)')
102   kb.tell('missile(x)&owns(Nono,x)=>sells(West,x,Nono)')
103   kb.tell('american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)
104   kb.query('criminal(x)')
105   kb.display()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
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)
```