VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Artificial Intelligence (23CS5PCAIN)

Submitted by

Sohan R (1BM23CS336)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Aug 2025 to Dec 2025

B.M.S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Artificial Intelligence (23CS5PCAIN)" carried out by **Sohan R (1BM23CS336),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Artificial Intelligence (23CS5PCAIN) work prescribed for the said degree.

Dr. Seema Patil Associate Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link: https://github.com/SohanR-1BM23CS336/AI-LAB.git

INDEX

Name Johan.R Std 5th Sec 'F'

Roll No. 336 Subject AI Lab School/College BMSCE

School/College Tel. No. — Parents Tel. No. —

	Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
Week1	1.	18/8/25	Tic-Tac-Toe problem	10	
Week2	2.	25/8/25	Vaccum Cleaner problem.	10	
Week3	3.	25/8/25	8-puzzle problem. (non-heuristic) using BFS	10	
	4.	31/8/25	8-puzzle using DFS (Non-heuristic)		
	5.	1/9/25	8-puzzle (Non-heuristic) using IDS	10	
Week4	6.	8/9/25	8-puzzle (heuristic) A* (Misplaced Tiles)	10	15/9/25
	7.	8/9/25	8-puzzle (heuristic) A* (Manhattan distance)		
Week5	8.	15/9/25	Hill-Climbing Algorithm 4-Queens	10	
	9.	15/9/25	Simulated Annealing: 8-Queens.	10	15/9/25
Week6	10.	22/9/25	Propositional Logic	10	22/9/25
Week7	11.	29/9/25	Unification Algorithm for FOL		
Week8	12.	13/10/25	First Order Logic - Forward reasoning		13/10/25
Week9	13.	20/10/25	First Order Logic	10	
Week10	14.	20/10/25	Adversial Search		6/11/25

Completed

Program 1

Implement Tic –Tac –Toe Game
Implement vacuum cleaner agent

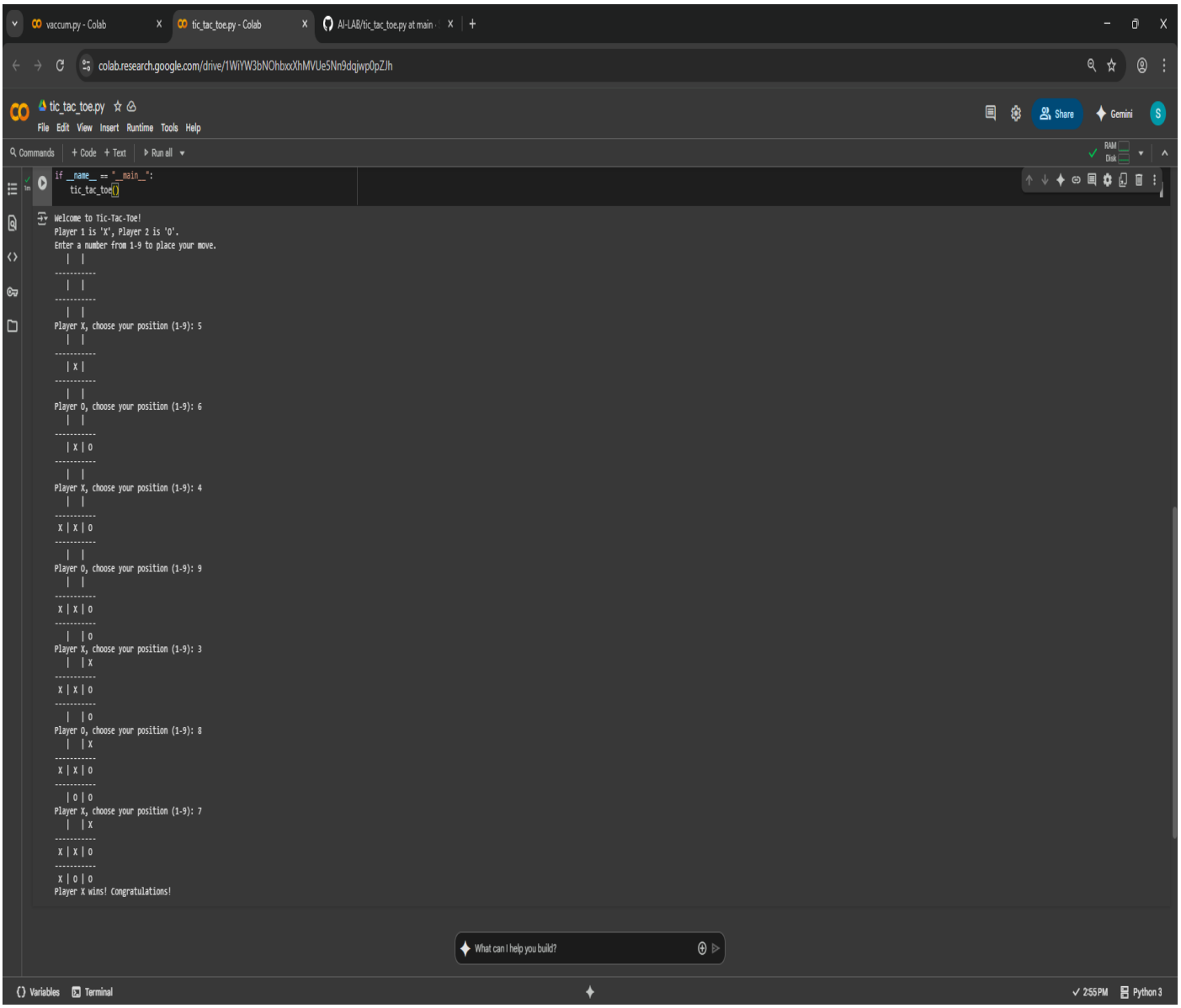
Algorithm:
Implement Tic –Tac –Toe Game

Implement vacuum cleaner agent

Code:
Implement Tic –Tac –Toe Game

```
def print_board(board):
    print(f" {board[0]} | {board[1]} | {board[2]} ")
    print("-----------")
    print(f" {board[3]} | {board[4]} | {board[5]} ")
    print("-----------")
    print(f" {board[6]} | {board[7]} | {board[8]} ")

def check_win(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]] == board[condition[1]] == board[condition[2]]
== player:
            return True
    return False

def is_draw(board):
    return ' ' not in board

def tic_tac_toe():
    board = [' '] * 9
    current_player = 'X'
    game_on = True

    print("Welcome to Tic-Tac-Toe!")
    print("Player 1 is 'X', Player 2 is 'O'.")
```

```
    print("Enter a number from 1-9 to place your move.")

    while game_on:
        print_board(board)
        try:
            position = int(input(f"Player {current_player}, choose your
position (1-9): "))
            index = position - 1

            if 0 <= index < 9 and board[index] == ' ':
                board[index] = current_player

                if check_win(board, current_player):
                    print_board(board)
                    print(f"Player {current_player} wins! Congratulations!")
                    game_on = False
                elif is_draw(board):
                    print_board(board)
                    print("It's a draw!")
                    game_on = False
                else:
                    current_player = 'O' if current_player == 'X' else 'X'
            else:
                print("This position is already taken or invalid. Please try
again.")
        except ValueError:
            print("Invalid input. Please enter a number.")

if __name__ == "__main__":
    tic_tac_toe()
```


vaccum.py - Colab
tic_tac_toe.py - Colab
AI-LAB/tic_tac_toe.py at main
colab.research.google.com/drive/1WiYW3bNOhbxoXhMVUe5Nn9dqjwp0pZJh
tic_tac_toe.py
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
if __name__ == "__main__":
tic_tac_toe()
Welcome to Tic-Tac-Toe!
Player 1 is 'X', Player 2 is 'O'.
Enter a number from 1-9 to place your move.
| |

| |

| |
Player X, choose your position (1-9): 5
| |

| X |

| |
Player O, choose your position (1-9): 6
| |

| X | O

| |
Player X, choose your position (1-9): 4
| |

X | X | O

| |
Player O, choose your position (1-9): 9
| |

X | X | O

| | O
Player X, choose your position (1-9): 3
| | X

X | X | O

| | O
Player O, choose your position (1-9): 8
| | X

X | X | O

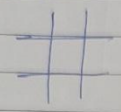
| O | O
Player X, choose your position (1-9): 7
| | X

X | X | O

X | O | O
Player X wins! Congratulations!
What can I help you build?
Variables
Terminal
2:55PM
Python 3

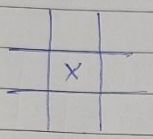
Output:

Welcome to tic tac toe



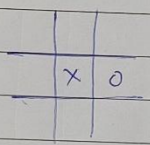
x's turn

Enter your move : 1, 1



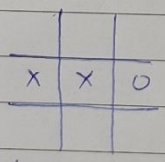
O's turn

Enter your move : 1, 2



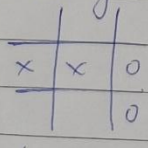
X's turn

Enter your move 1, 0



O's turn

Enter your move 2, 2



X's turn

Enter your move 0, 2

		X
X	X	O
		O

O's turn
Enter your move 2, 1

		X
X	X	O
	O	O

X's turn
Enter your move 2, 0

		X
X	X	O
X	O	O

O's turn
Enter your move 0, 1

	O	X
X	X	O
X	O	O

X's turn
Enter your move 0, 0

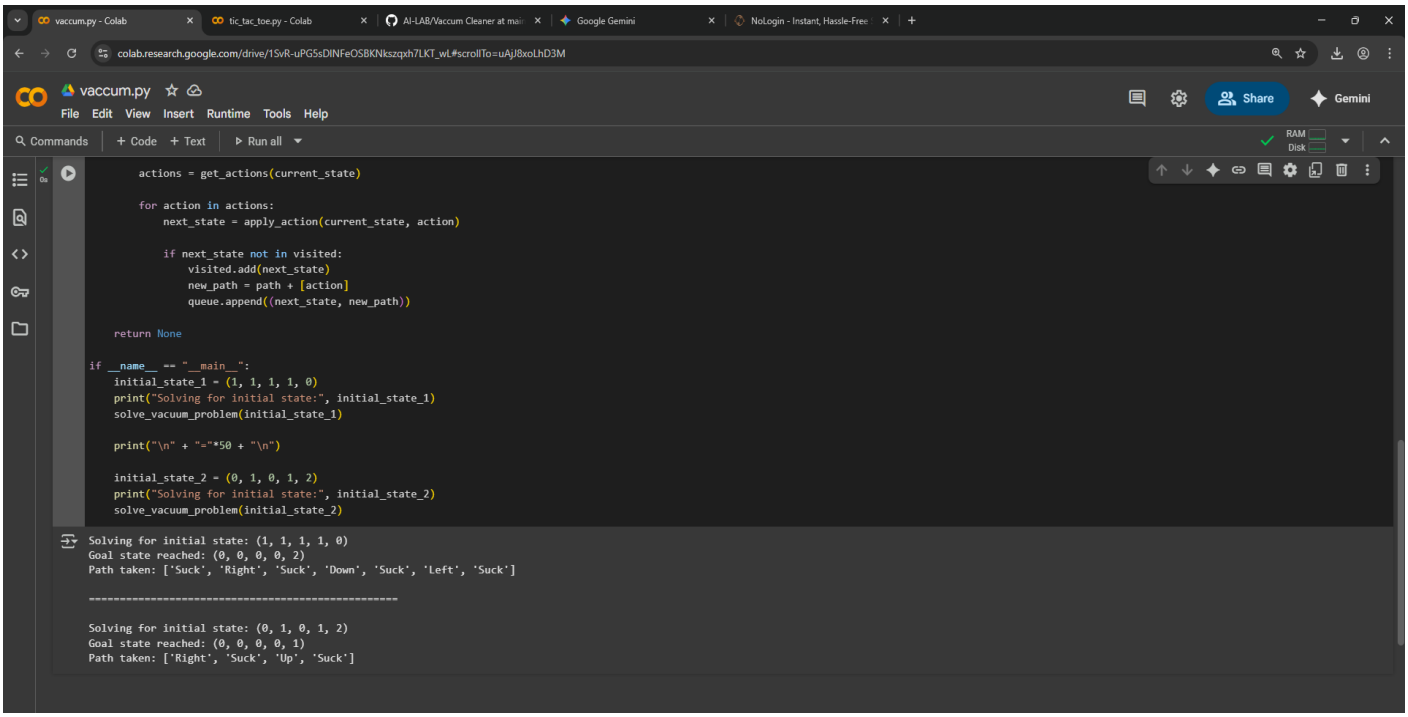
X	O	X
X	X	O
X	O	O

X won

Implement vacuum cleaner agent

```
import collections

def get_actions(state):
    actions = []
    room_status = list(state[:4])
    location = state[4] - 1

    if room_status[location] == 1:
        actions.append("Suck")

    if location < 3:
        actions.append("Right")

    if location > 0:
        actions.append("Left")

    return actions

def apply_action(state, action):
    room_status = list(state[:4])
    location = state[4] - 1

    if action == "Suck":
        room_status[location] = 0
    elif action == "Right":
        location += 1
    elif action == "Left":
        location -= 1

    return tuple(room_status + [location + 1])

def solve_vacuum_problem(initial_state):
    queue = collections.deque([(initial_state, [])])
    visited = {initial_state}

    while queue:
        current_state, path = queue.popleft()
```

```
        if all(room == 0 for room in current_state[:4]):
            print("Goal state reached:", current_state)
            print("Path taken:", path)
            return path

        actions = get_actions(current_state)

        for action in actions:
            next_state = apply_action(current_state, action)

            if next_state not in visited:
                visited.add(next_state)
                new_path = path + [action]
                queue.append((next_state, new_path))

    return None

if __name__ == "__main__":
    initial_state_1 = (1, 1, 1, 1, 1)
    print("Solving for initial state:", initial_state_1)
    solve_vacuum_problem(initial_state_1)

    print("\n" + "="*50 + "\n")

    initial_state_2 = (0, 1, 0, 1, 3)
    print("Solving for initial state:", initial_state_2)
    solve_vacuum_problem(initial_state_2)
```


vaccum.py - Colab
tic_tac_toe.py - Colab
AI-LAB/Vaccum Cleaner at mai
Google Gemini
NoLogin - Instant, Hassle-Free
colab.research.google.com/drive/1SvR-uPG5sDINFeOSBKNkszqxh7LKT_wL#scrollTo=uAjJ8xoLhD3M
vaccum.py
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
actions = get_actions(current_state)
for action in actions:
next_state = apply_action(current_state, action)
if next_state not in visited:
visited.add(next_state)
new_path = path + [action]
queue.append((next_state, new_path))
return None
if __name__ == "__main__":
initial_state_1 = (1, 1, 1, 1, 0)
print("Solving for initial state:", initial_state_1)
solve_vacuum_problem(initial_state_1)
print("\n" + "="*50 + "\n")
initial_state_2 = (0, 1, 0, 1, 2)
print("Solving for initial state:", initial_state_2)
solve_vacuum_problem(initial_state_2)
Solving for initial state: (1, 1, 1, 1, 0)
Goal state reached: (0, 0, 0, 0, 2)
Path taken: ['Suck', 'Right', 'Suck', 'Down', 'Suck', 'Left', 'Suck']
==
Solving for initial state: (0, 1, 0, 1, 2)
Goal state reached: (0, 0, 0, 0, 1)
Path taken: ['Right', 'Suck', 'Up', 'Suck']

Date __/__/__
Page _____

Output:-

Enter status of room A (C for clean, D for dirty): d
Enter status of room B (C for clean, D for dirty): d
Which room to move to? (A or B): a
Room A is dirty cleaning now.
Which room to move to? (A or B): b
Room B is dirty cleaning now.
All rooms are clean. Exiting...
Total cost of moves $2

$8 x8

Program 2

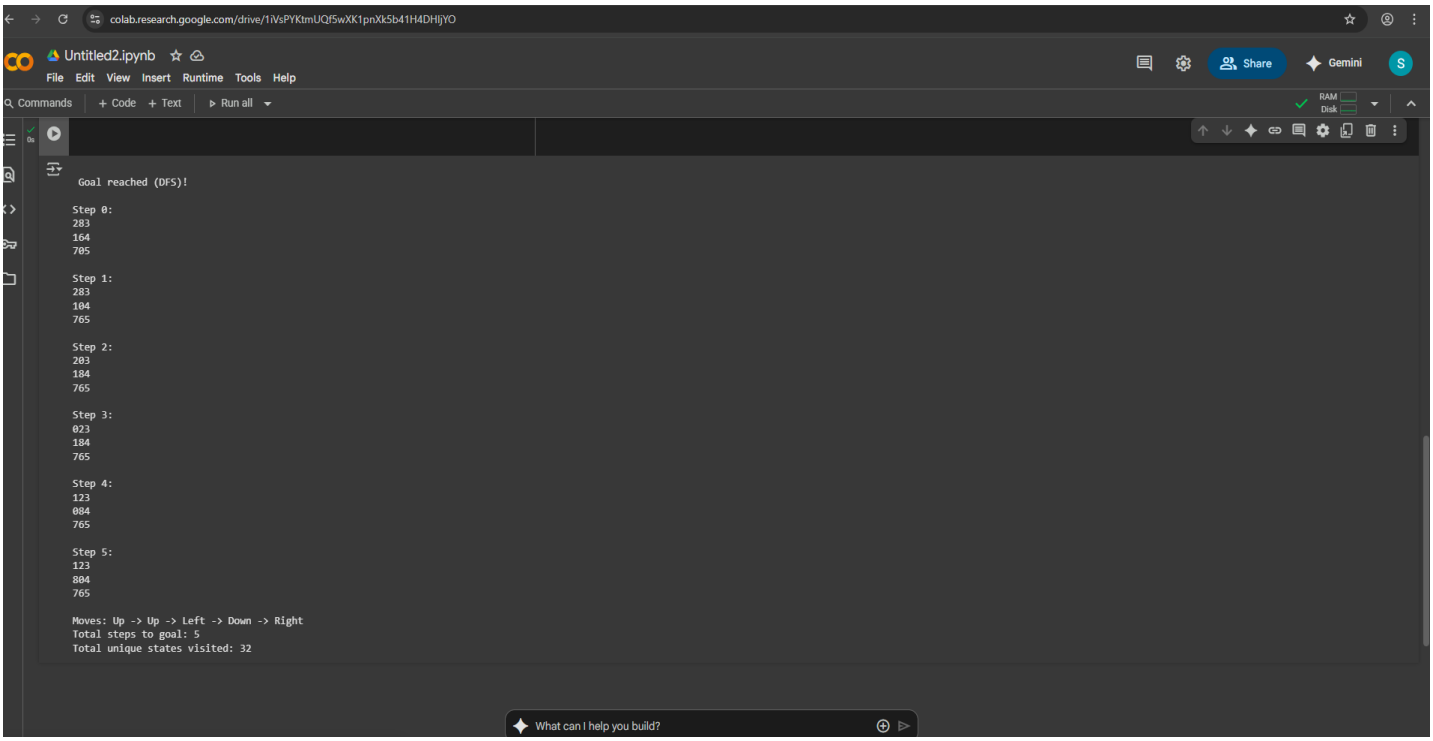
Implement 8 puzzle problems using
a) Depth First Search (DFS)
b) Iterative deepening search

a) Depth First Search (DFS):

Code:

```
def print_state(state):
    for i in range(0, 9, 3):
        print(state[i:i+3])
    print()

def is_goal(state):
    return state == "123804765"

def get_neighbors(state):
    neighbors = []
    moves = {'Up': -3, 'Down': 3, 'Left': -1, 'Right': 1}
    zero_index = state.index('0')

    for move, pos_change in moves.items():
        new_index = zero_index + pos_change
        if move == 'Left' and zero_index % 3 == 0:
            continue
        if move == 'Right' and zero_index % 3 == 2:
            continue
        if move == 'Up' and zero_index < 3:
            continue
        if move == 'Down' and zero_index > 5:
            continue
        state_list = list(state)
        state_list[zero_index], state_list[new_index] = state_list[new_index],
state_list[zero_index]
        neighbors.append(("".join(state_list), move))
    return neighbors

def dfs(start_state, max_depth=20):
    visited = set()
    stack = [(start_state, [], [], 0)]  # (state, path, moves, depth)
```

```
    while stack:
        current_state, path, moves, depth = stack.pop()

        if current_state in visited:
            continue
        visited.add(current_state)

        if is_goal(current_state):
            steps = path + [current_state]
            print("\n Goal reached (DFS)!\n")
            for i, step in enumerate(steps):
                print(f"Step {i}:")
                print_state(step)
            print("Moves:", " -> ".join(moves))
            print("Total steps to goal:", len(steps) - 1)
            print("Total unique states visited:", len(visited))
            return

        if depth < max_depth:
            for neighbor, move in reversed(get_neighbors(current_state)):
                if neighbor not in visited:
                    stack.append((neighbor, path + [current_state], moves + [move], depth + 1))

    print(f"No solution found within depth limit {max_depth}!")

start = "283164705"
dfs(start, max_depth=10)
```

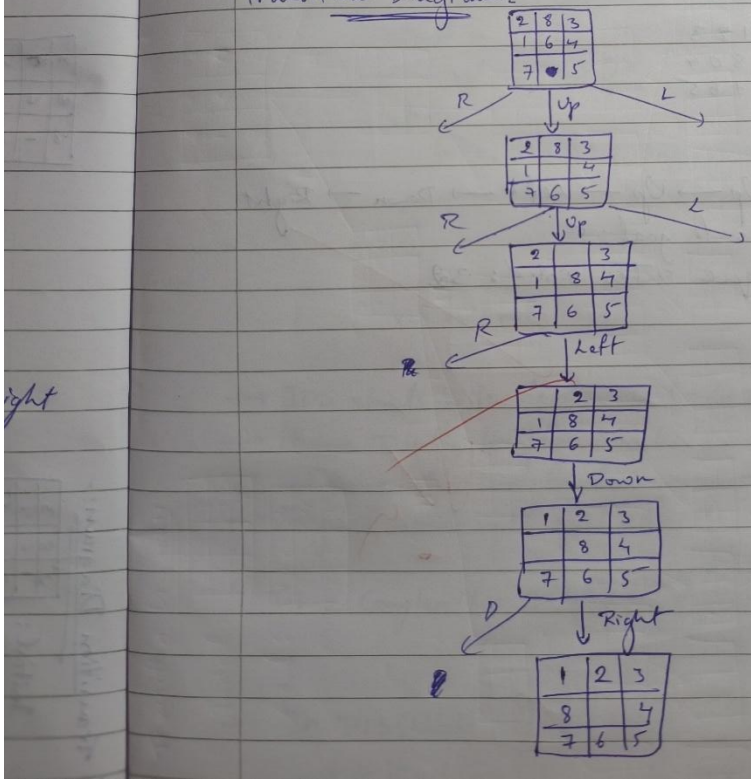

colab.research.google.com/drive/1iVsPYKtmUQf5wXK1pnXk5b41H4DHljYO
Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
Goal reached (DFS)!
Step 0:
283
164
705
Step 1:
283
104
765
Step 2:
203
184
765
Step 3:
023
184
765
Step 4:
123
084
765
Step 5:
123
804
765
Moves: Up -> Up -> Left -> Down -> Right
Total steps to goal: 5
Total unique states visited: 32
What can I help you build?

4. 8- puzzle problem (non-heuristic using DFS)

Algorithm:

- Start from Initial state and compare with the goal matrix.
- In DFS move the space to the up, down, right and left
- Then move the space to available space to the left most & check if it matches the solution or else backtrace to started matrix then compare.
- If found return the matrix or No of iterations.
- We must set the depth limit. (10)

Transition Diagram:



(Depth limit = 10)

Outputs:

Goal Reached (DFS)!

Step 0: 2 8 3
1 6 4
7 0 5

Step 1: 2 8 3
1 0 4
7 6 5

Step 2: 2 0 3
1 8 4
7 6 5

Step 3: 0 2 3
1 8 4
7 6 5

Step 4: 1 2 3
0 8 4
7 6 5

Step 5: 1 2 3
8 0 4
7 6 5

Moves: Up → Up → Left → Down → Right

Total steps to goal: 5

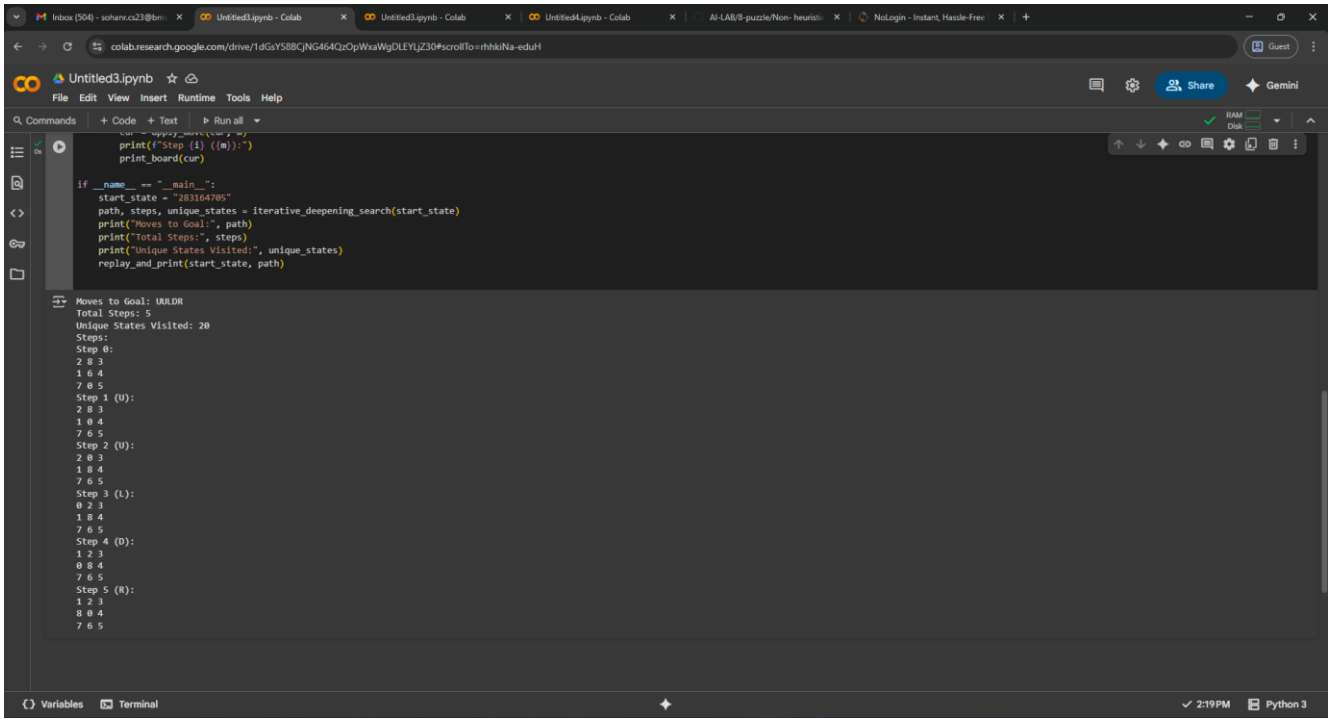
Total unique states visited: 32.

b) Iterative deepening search

Code:

```
GOAL = "123804765"
MOVE_OFFSETS = {'U': -3, 'L': -1, 'R': 1, 'D': 3}

def is_valid_move(z, m):
    if m == 'L' and z % 3 == 0: return False
    if m == 'R' and z % 3 == 2: return False
    if m == 'U' and z < 3: return False
    if m == 'D' and z > 5: return False
    return True

def apply_move(state, m):
    z = state.index('0')
    s = z + MOVE_OFFSETS[m]
    a = list(state)
    a[z], a[s] = a[s], a[z]
    return "".join(a)

def depth_limited_search(state, limit, visited, path):
    if state == GOAL: return path
    if limit == 0: return None
    visited.add(state)
    z = state.index('0')
    for m in ['U','L','R','D']:
        if is_valid_move(z, m):
            ns = apply_move(state, m)
            if ns not in visited:
                r = depth_limited_search(ns, limit - 1, visited, path + m)
                if r is not None: return r
    return None

def iterative_deepening_search(start_state):
    d = 0
    total_visited = set()
    while True:
        visited = set()
        r = depth_limited_search(start_state, d, visited, "")
        total_visited |= visited
        if r is not None:
            return r, len(r), len(total_visited)
        d += 1

def print_board(state):
    for i in range(0, 9, 3):
        print(state[i], state[i+1], state[i+2])
```

```
def replay_and_print(start_state, path):
    print("Steps:")
    print("Step 0:")
    print_board(start_state)
    cur = start_state
    for i, m in enumerate(path, 1):
        cur = apply_move(cur, m)
        print(f"Step {i} ({m}):")
        print_board(cur)

if __name__ == "__main__":
    start_state = "283164705"
    path, steps, unique_states = iterative_deepening_search(start_state)
    print("Moves to Goal:", path)
    print("Total Steps:", steps)
    print("Unique States Visited:", unique_states)
    replay_and_print(start_state, path)
```



5. 8-puzzle problem (non-heuristic) using iterative deepening search (IDS)

Ans

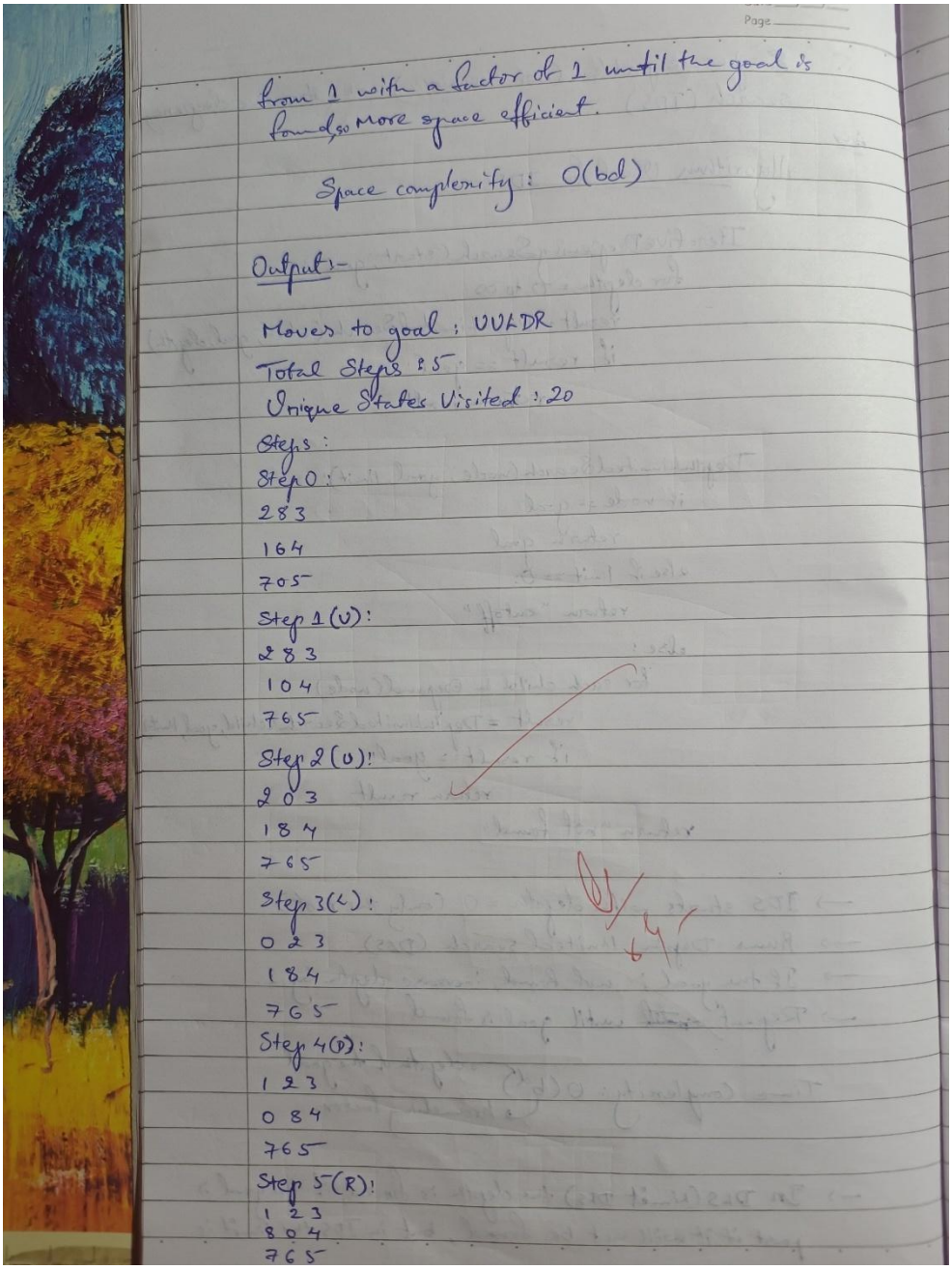
Algorithm Used for IDS :-

```
IterativeDeepeningSearch(start, goal):
    for depth = 0 to ∞:
        result = DepthLimitedSearch(start, goal, depth)
        if result == goal:
            return result

DepthLimitedSearch(node, goal, limit):
    if node == goal:
        return goal
    else if limit == 0:
        return "cutoff"
    else:
        for each child in Expand(node):
            result = DepthLimitedSearch(child, goal, li
            if result == goal:
                return result
        return "not found"
```

→ IDS starts with depth = 0 (only the root)
→ Runs Depth-limited search (DLS).
→ If the goal is not found, increase depth by 1.
→ Repeat ~~until~~ until goal is found.

Time Complexity : $O(b^d)$ → depth of the goal
→ branching factor.

→ In DLS (Limit DFS) the depth is fixed, if the goal is ... it if it will not be found but in IDS the limit is



from 1 with a factor of 1 until the goal is found so More space efficient.

Space complexity : O(bd)

Outputs:-

Moves to goal : UULDR
Total Steps : 5
Unique States Visited : 20
Steps :
Step 0 :
2 8 3
1 6 4
7 0 5
Step 1 (U) :
2 8 3
1 0 4
7 6 5
Step 2 (U) :
2 0 3
1 8 4
7 6 5
Step 3 (L) :
0 2 3
1 8 4
7 6 5
Step 4 (D) :
1 2 3
0 8 4
7 6 5
Step 5 (R) :
1 2 3
8 0 4
7 6 5

Program 3

Implement A* search algorithm

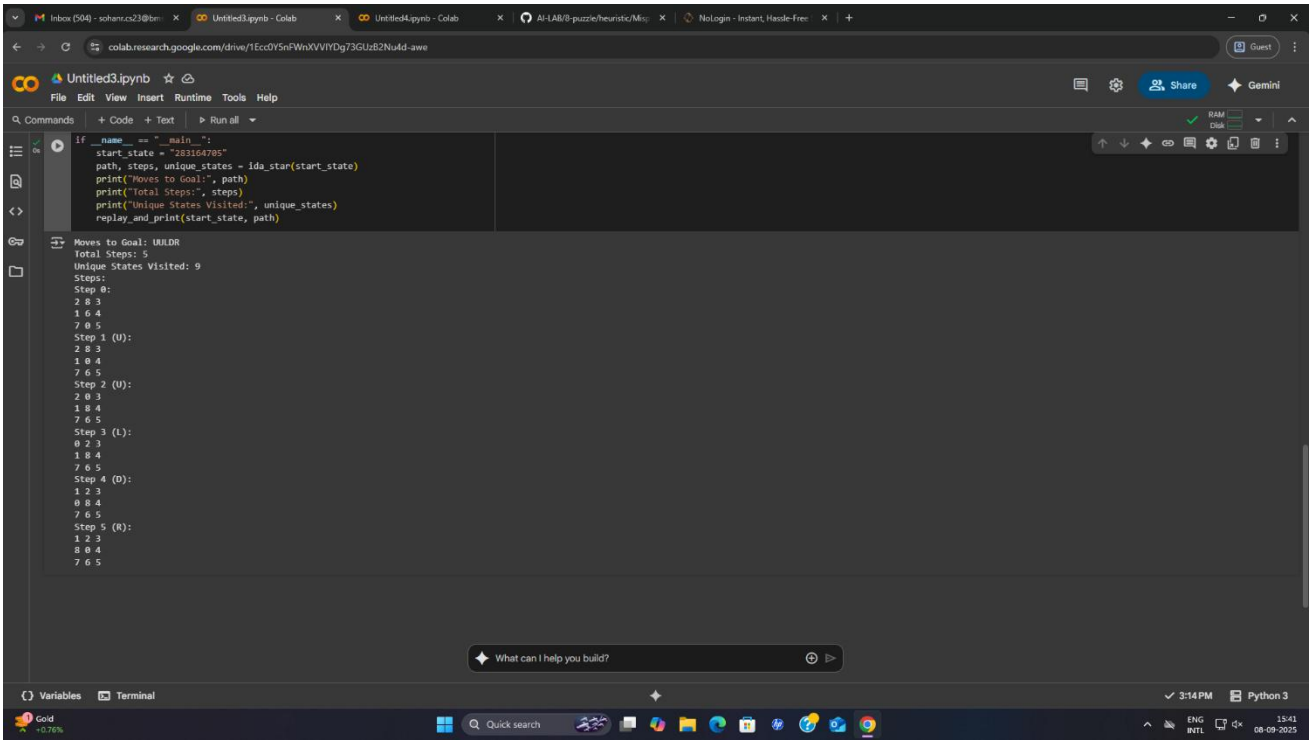
a) Misplaced Tiles
b) Manhattan Distance

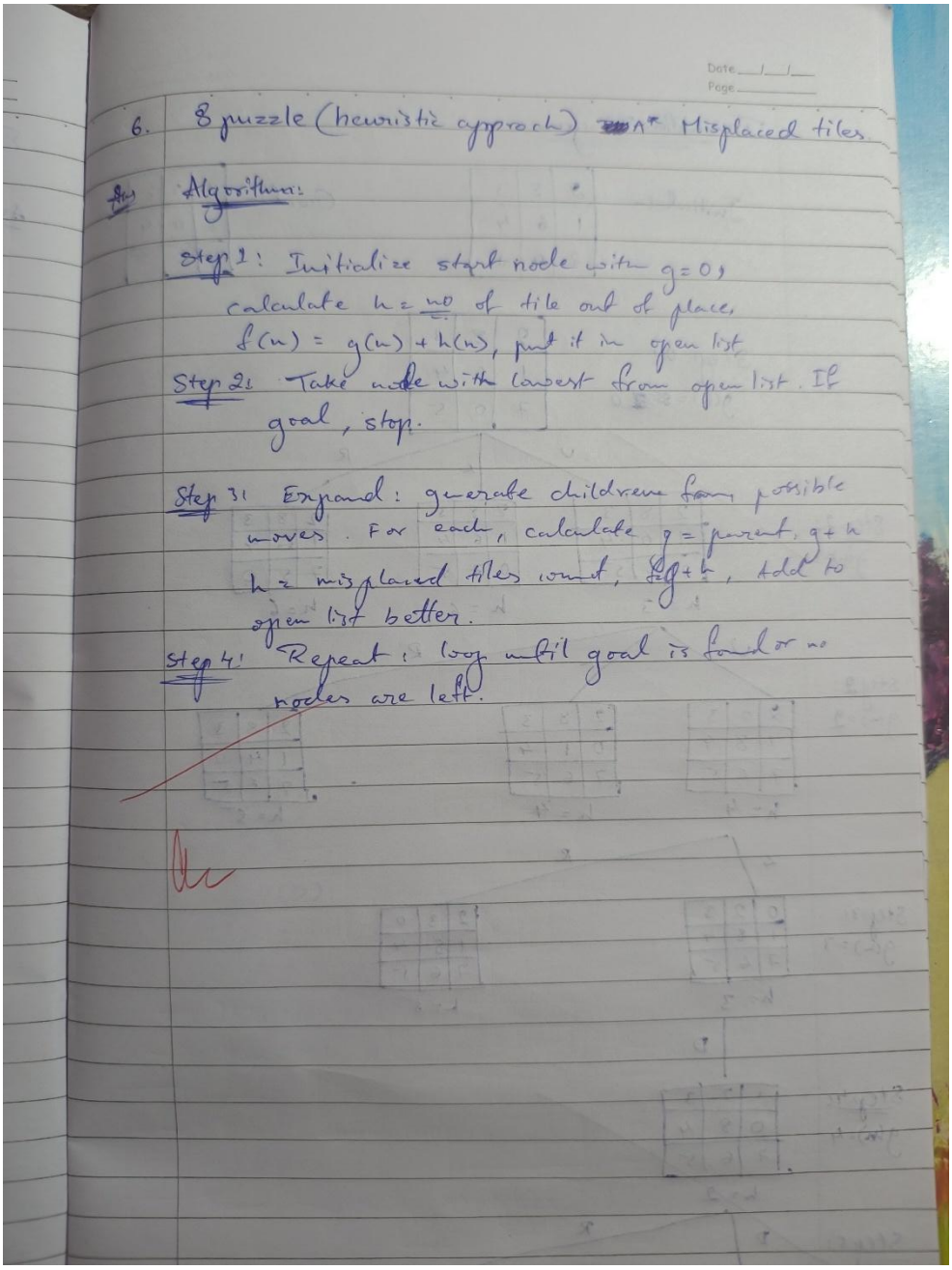
Misplaced Tiles

Code:

```
GOAL = "123804765"
MOVE_OFFSETS = {'U': -3, 'L': -1, 'R': 1, 'D': 3}

def is_valid_move(z, m):
    if m == 'L' and z % 3 == 0: return False
    if m == 'R' and z % 3 == 2: return False
    if m == 'U' and z < 3: return False
    if m == 'D' and z > 5: return False
    return True

def apply_move(state, m):
    z = state.index('0')
    s = z + MOVE_OFFSETS[m]
    a = list(state)
    a[z], a[s] = a[s], a[z]
    return "".join(a)

def misplaced_tiles(state):
    return sum(1 for i in range(9) if state[i] != '0' and state[i] != GOAL[i])

def ida_star(start):
    bound = misplaced_tiles(start)
    path = [(start, "")]
    visited = set([start])
    total_visited = set()

    def search(path, g, bound):
        state, moves = path[-1]
        f = g + misplaced_tiles(state)
        if f > bound:
            return f, None
        if state == GOAL:
            return True, moves
        min_bound = float("inf")
        z = state.index('0')
        for m in ['U','L','R','D']:
            if is_valid_move(z, m):
                new_state = apply_move(state, m)
```

```
            if new_state not in {s for s,_ in path}:
                path.append((new_state, moves + m))
                total_visited.add(new_state)
                t, sol = search(path, g+1, bound)
                if t == True: return True, sol
                if isinstance(t, int) and t < min_bound:
                    min_bound = t
                path.pop()
        return min_bound, None

    while True:
        t, sol = search(path, 0, bound)
        if t == True:
            return sol, len(sol), len(total_visited)
        if t == float("inf"):
            return None, 0, len(total_visited)
        bound = t

def print_board(state):
    for i in range(0, 9, 3):
        print(state[i], state[i+1], state[i+2])

def replay_and_print(start_state, path):
    print("Steps:")
    print("Step 0:")
    print_board(start_state)
    cur = start_state
    for i, m in enumerate(path, 1):
        cur = apply_move(cur, m)
        print(f"Step {i} ({m}):")
        print_board(cur)

def is_solvable(state):
    s = [int(c) for c in state if c != '0']
    inv = 0
    for i in range(len(s)):
        for j in range(i+1, len(s)):
            if s[i] > s[j]: inv += 1
    return inv % 2 == 0

if __name__ == "__main__":
    start_state = "283164705"
    path, steps, unique_states = ida_star(start_state)
    print("Moves to Goal:", path)
    print("Total Steps:", steps)
    print("Unique States Visited:", unique_states)
    replay_and_print(start_state, path)
```


Untitled3.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
if __name__ == "__main__":
start_state = "283164705"
path, steps, unique_states = ida_star(start_state)
print("Moves to Goal:", path)
print("Total Steps:", steps)
print("Unique States Visited:", unique_states)
replay_and_print(start_state, path)
Moves to Goal: URLDR
Total Steps: 5
Unique States Visited: 9
Steps:
Step 0:
2 8 3
1 6 4
7 0 5
Step 1 (U):
2 8 3
1 0 4
7 6 5
Step 2 (U):
2 0 3
1 8 4
7 6 5
Step 3 (L):
0 2 3
1 8 4
7 6 5
Step 4 (D):
1 2 3
0 8 4
7 6 5
Step 5 (R):
1 2 3
8 0 4
7 6 5
What can I help you build?
Variables
Terminal
3:14 PM
Python 3



Date __/__/__
Page _____

6. 8 puzzle (heuristic approach) ~~A*~~ Misplaced tiles

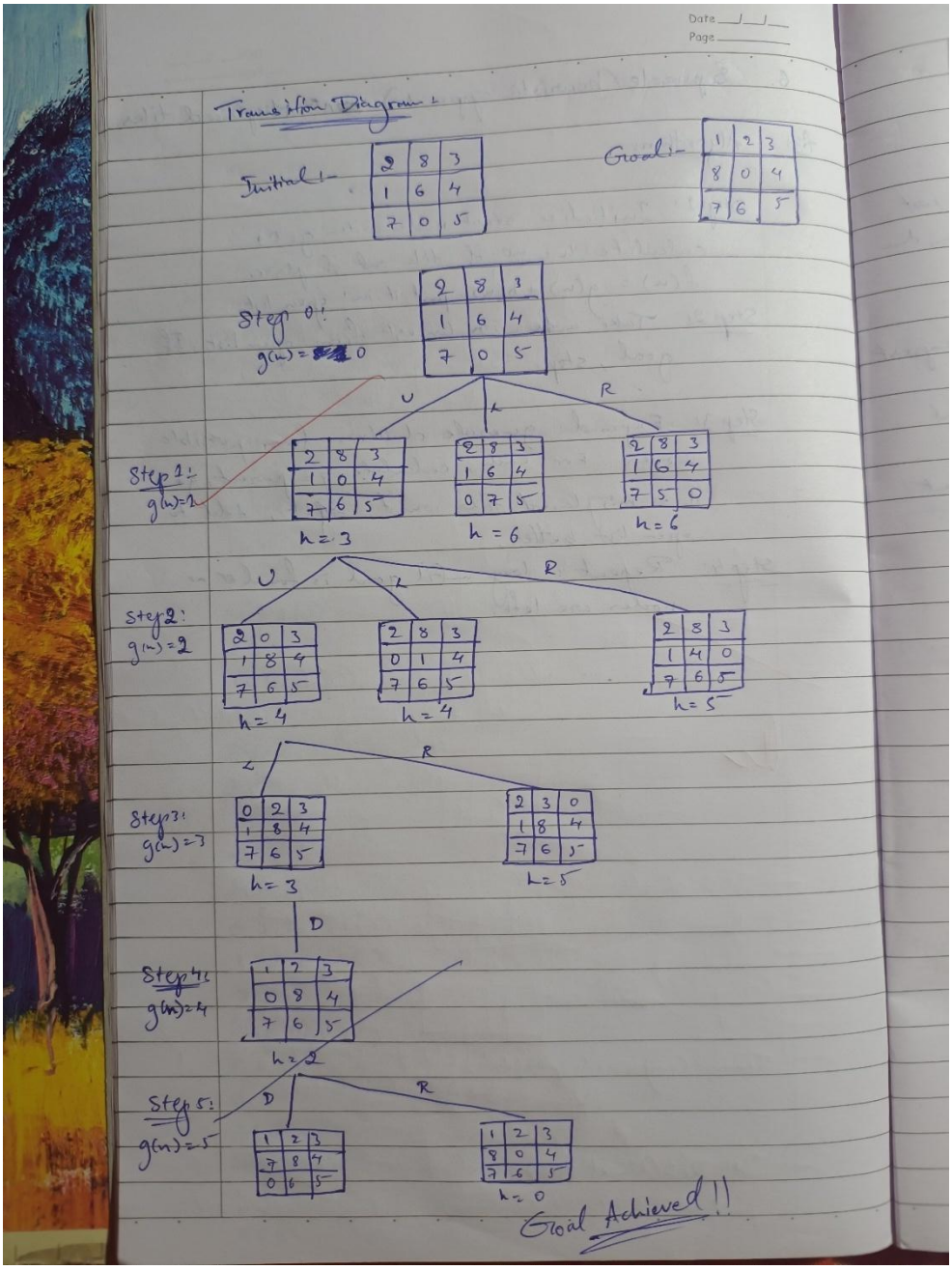
Algorithm:

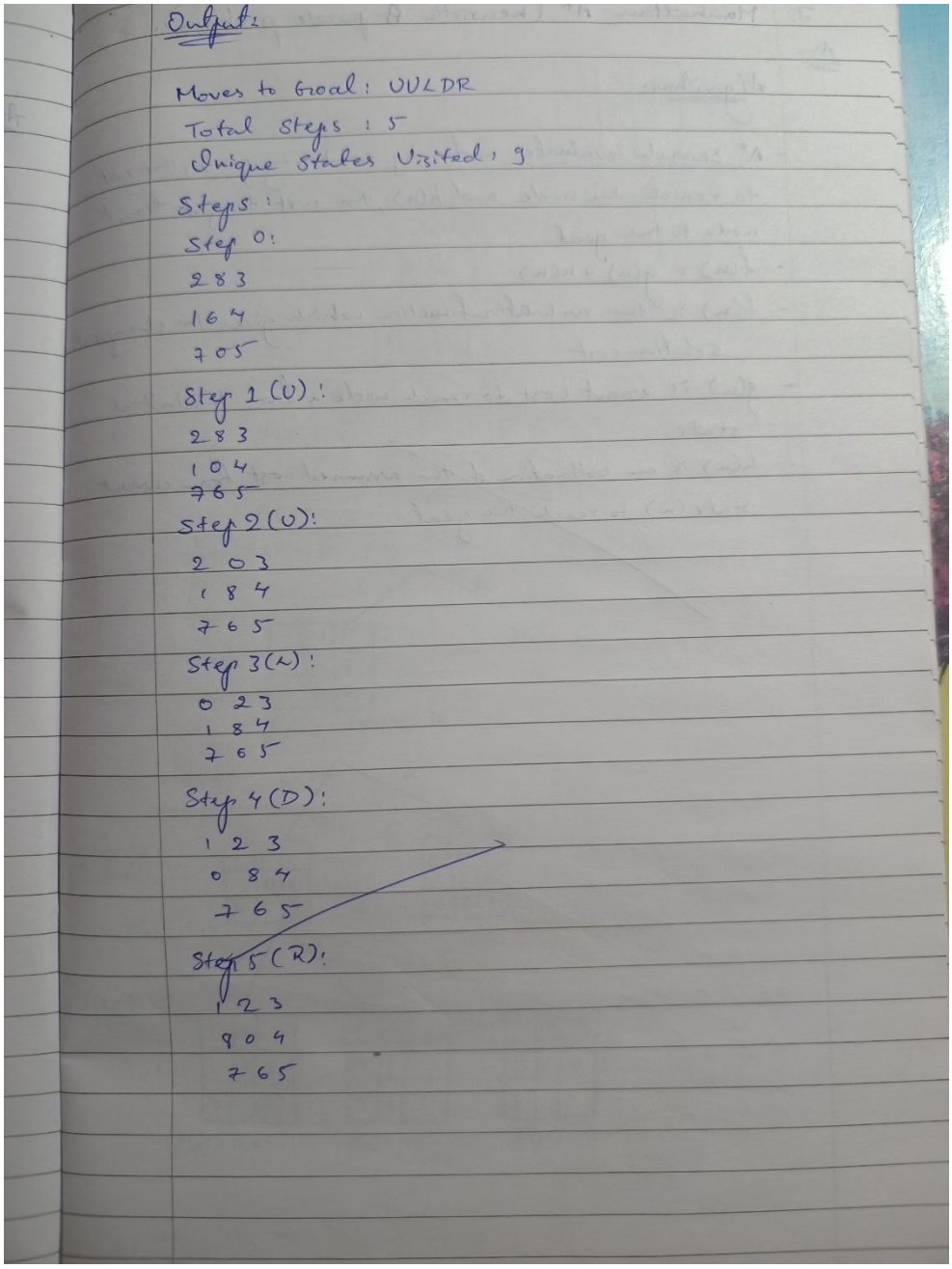
Step 1: Initialize start node with g=0, calculate h = no of tile out of place, f(n) = g(n) + h(n), put it in open list

Step 2: Take node with lowest from open list. If goal, stop.

Step 3: Expand: generate childrens from possible moves. For each, calculate g = parent g + 1, h = misplaced tiles count, f = g+h, add to open list better.

Step 4: Repeat: loop until goal is found or no nodes are left.


Transition Diagram :-
Initial :-
Goal :-
Step 0 :
g(n) = 0
U
L
R
Step 1 :-
g(n)=1
h = 3
h = 6
h = 6
Step 2 :
g(n) = 2
h = 4
h = 4
h = 5
Step 3 :
g(n) = 3
h = 3
h = 5
D
Step 4 :
g(n) = 4
h = 2
Step 5 :
g(n) = 5
h = 0
Goal Achieved !!

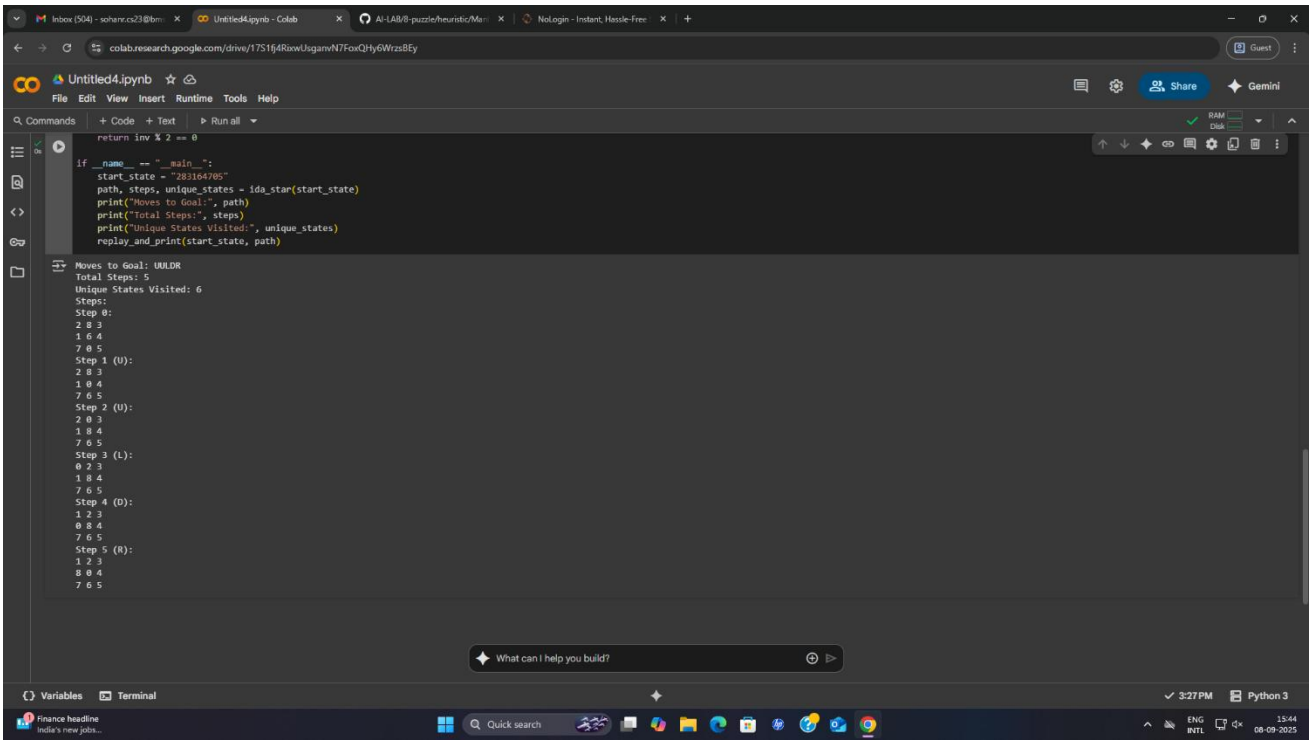

Output:

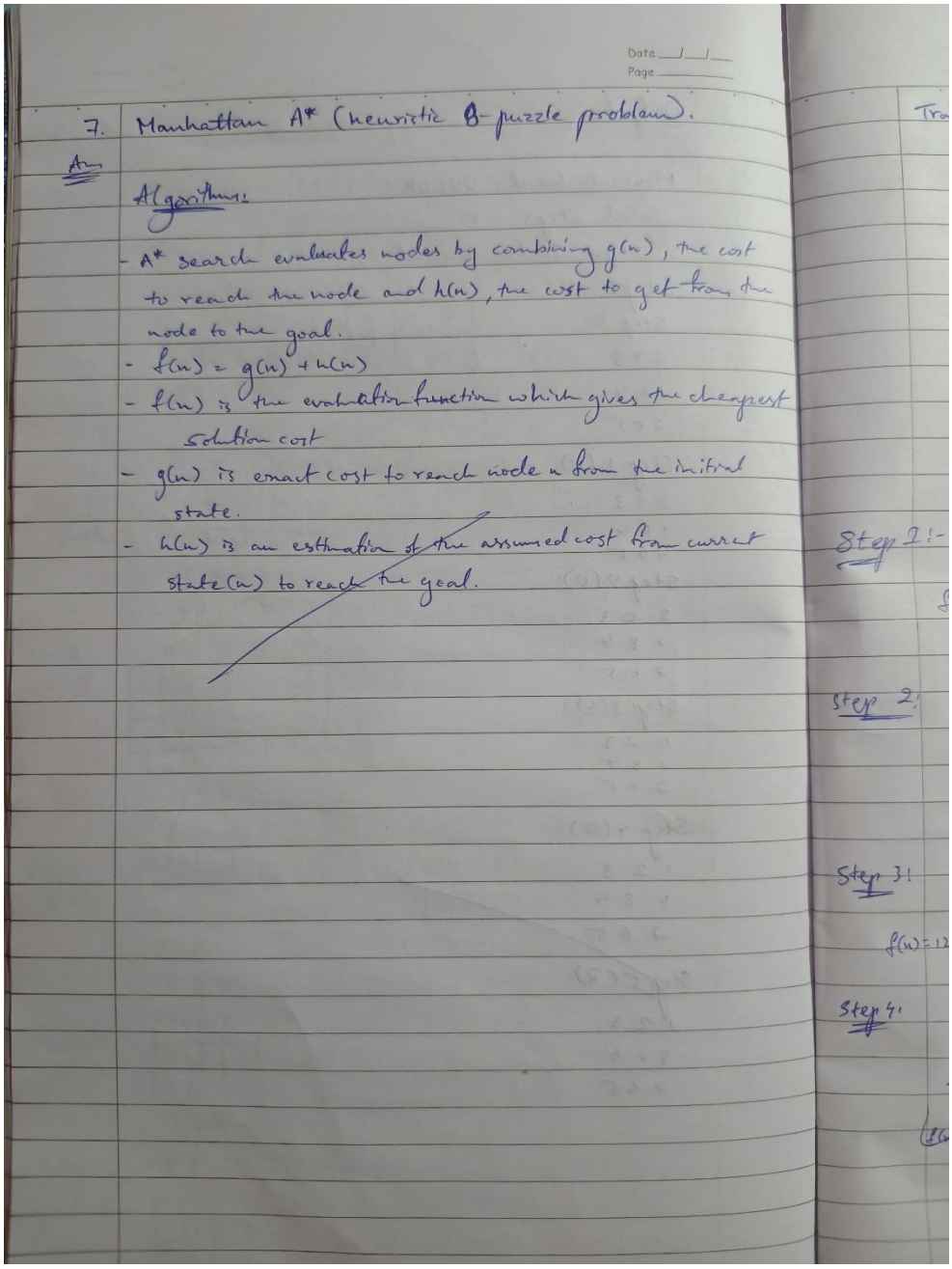
Moves to Goal: UULDR
Total Steps : 5
Unique States Visited: 9
Steps :
Step 0:
2 8 3
1 6 4
7 0 5
Step 1 (U):
2 8 3
1 0 4
7 6 5
Step 2 (U):
2 0 3
1 8 4
7 6 5
Step 3 (L):
0 2 3
1 8 4
7 6 5
Step 4 (D):
1 2 3
0 8 4
7 6 5
Step 5 (R):
1 2 3
8 0 4
7 6 5

Manhattan Distance:
Code:

```
GOAL = "123804765"
MOVE_OFFSETS = {'U': -3, 'L': -1, 'R': 1, 'D': 3}

def is_valid_move(z, m):
    if m == 'L' and z % 3 == 0: return False
    if m == 'R' and z % 3 == 2: return False
    if m == 'U' and z < 3: return False
    if m == 'D' and z > 5: return False
    return True

def apply_move(state, m):
    z = state.index('0')
    s = z + MOVE_OFFSETS[m]
    a = list(state)
    a[z], a[s] = a[s], a[z]
    return "".join(a)

def manhattan_distance(state):
    dist = 0
    for i, val in enumerate(state):
        if val != '0':
            goal_pos = GOAL.index(val)
            x1, y1 = divmod(i, 3)
            x2, y2 = divmod(goal_pos, 3)
            dist += abs(x1 - x2) + abs(y1 - y2)
    return dist

def ida_star(start):
    bound = manhattan_distance(start)
    path = [(start, "")]
    total_visited = set([start])

    def search(path, g, bound):
        state, moves = path[-1]
        f = g + manhattan_distance(state)
        if f > bound:
            return f, None
        if state == GOAL:
            return True, moves
        min_bound = float("inf")
        z = state.index('0')
        for m in ['U','L','R','D']:
            if is_valid_move(z, m):
                new_state = apply_move(state, m)
                if new_state not in {s for s,_ in path}:
```

```
                path.append((new_state, moves + m))
                total_visited.add(new_state)
                t, sol = search(path, g+1, bound)
                if t == True:
                    return True, sol
                if isinstance(t, int) and t < min_bound:
                    min_bound = t
                path.pop()
        return min_bound, None

    while True:
        t, sol = search(path, 0, bound)
        if t == True:
            return sol, len(sol), len(total_visited)
        if t == float("inf"):
            return None, 0, len(total_visited)
        bound = t

def print_board(state):
    for i in range(0, 9, 3):
        print(state[i], state[i+1], state[i+2])

def replay_and_print(start_state, path):
    print("Steps:")
    print("Step 0:")
    print_board(start_state)
    cur = start_state
    for i, m in enumerate(path, 1):
        cur = apply_move(cur, m)
        print(f"Step {i} ({m}):")
        print_board(cur)

def is_solvable(state):
    s = [int(c) for c in state if c != '0']
    inv = 0
    for i in range(len(s)):
        for j in range(i+1, len(s)):
            if s[i] > s[j]: inv += 1
    return inv % 2 == 0

if __name__ == "__main__":
    start_state = "283164705"
    path, steps, unique_states = ida_star(start_state)
    print("Moves to Goal:", path)
    print("Total Steps:", steps)
    print("Unique States Visited:", unique_states)
    replay_and_print(start_state, path)
```


Untitled4.ipynb - Colab
colab.research.google.com/drive/17S1fj4RxwUsganvN7FoxQHy6WrzsBEy
Untitled4.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
return inv % 2 == 0
if __name__ == "__main__":
start_state = "283164705"
path, steps, unique_states = ida_star(start_state)
print("Moves to Goal:", path)
print("Total Steps:", steps)
print("Unique States Visited:", unique_states)
replay_and_print(start_state, path)
Moves to Goal: UULDR
Total Steps: 5
Unique States Visited: 6
Steps:
Step 0:
2 8 3
1 6 4
7 0 5
Step 1 (U):
2 8 3
1 0 4
7 6 5
Step 2 (U):
2 0 3
1 8 4
7 6 5
Step 3 (L):
0 2 3
1 8 4
7 6 5
Step 4 (D):
1 2 3
0 8 4
7 6 5
Step 5 (R):
1 2 3
8 0 4
7 6 5
What can I help you build?
Variables Terminal
3:27PM Python 3
Finance headline
India's new jobs...
Quick search
ENG
INTL
15:44
08-09-2025



Date __/__/__
Page _____

7. Manhattan A* (heuristic 8-puzzle problem).

Ans

Algorithm:

- A* search evaluates nodes by combining g(n), the cost to reach the node and h(n), the cost to get from the node to the goal.
- f(n) = g(n) + h(n)
- f(n) is the evaluation function which gives the cheapest solution cost
- g(n) is exact cost to reach node n from the initial state.
- h(n) is an estimation of the assumed cost from current state(n) to reach the goal.

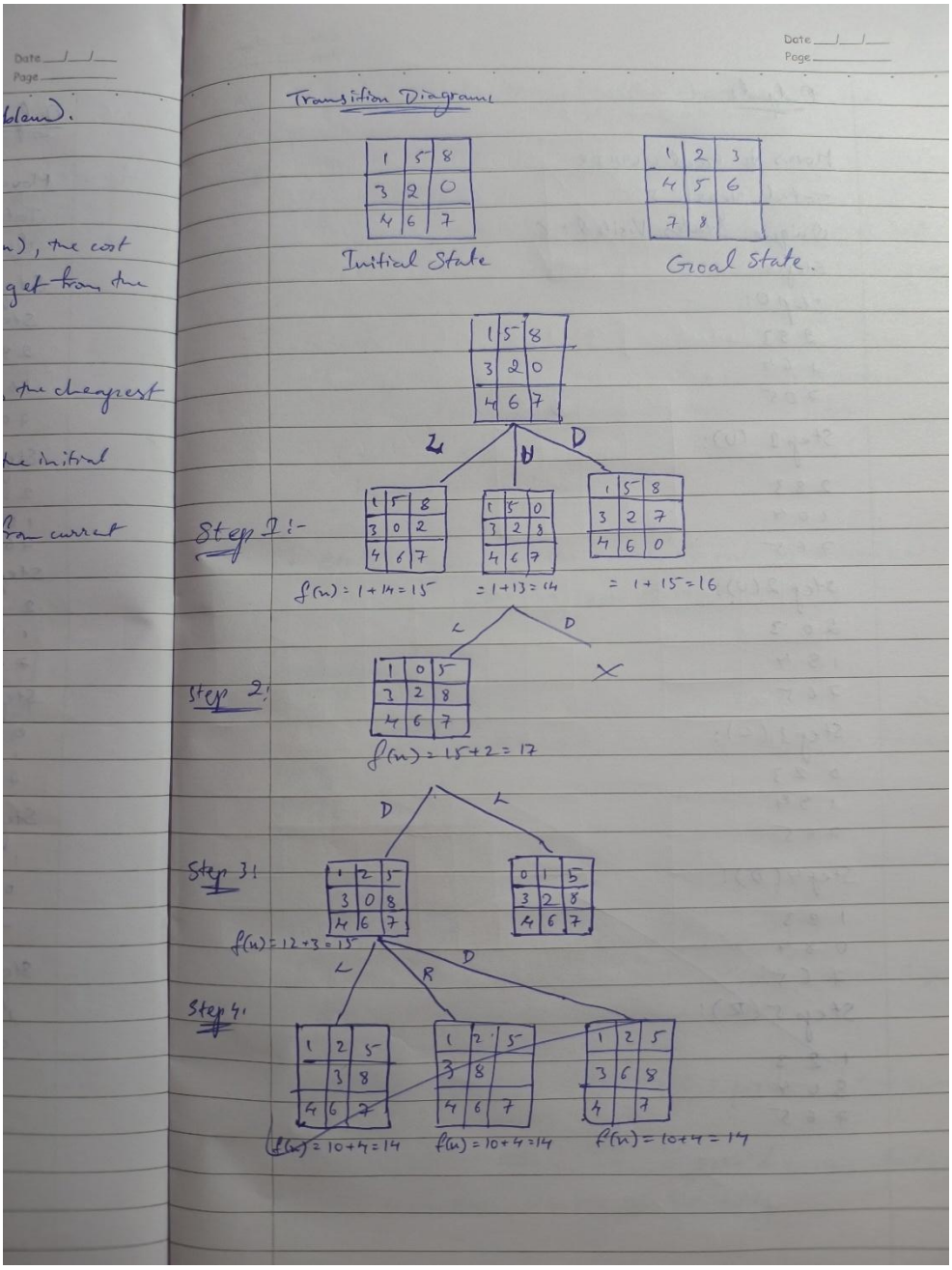
Step 1:-

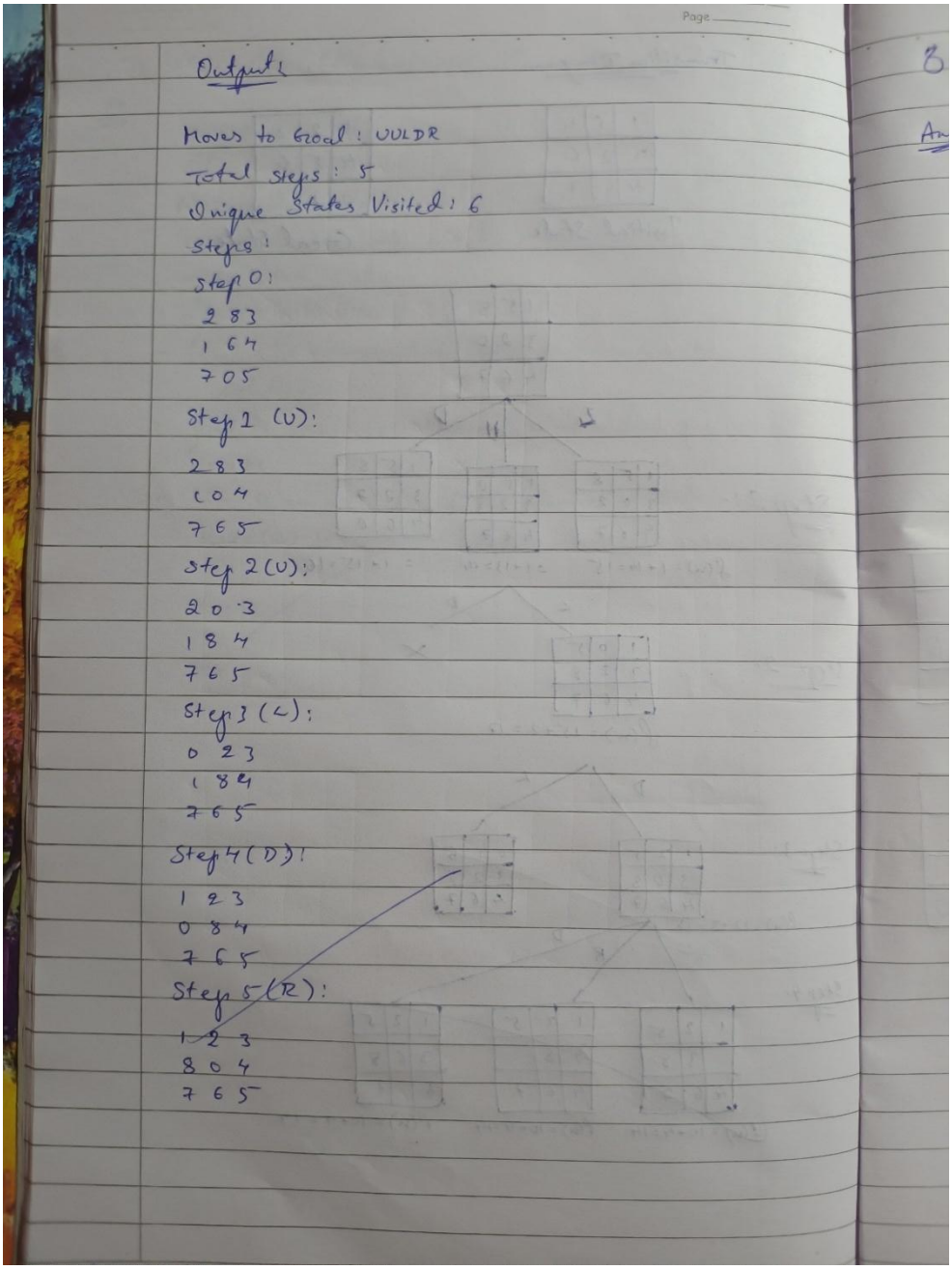
Step 2:

Step 3:

f(n) = 12

Step 4:


Transition Diagram
Initial State
Goal State.
Step 1:-
f(n) = 1+14 = 15
= 1+13 = 14
= 1+15 = 16
Step 2:
f(n) = 15+2 = 17
Step 3:
f(n) = 12+3 = 15
Step 4:
f(n) = 10+4 = 14
f(n) = 10+4 = 14
f(n) = 10+4 = 14



Output:

Moves to Goal : UULDR
Total steps : 5
Unique States Visited : 6
Steps :
Step 0:
2 8 3
1 6 4
7 0 5
Step 1 (U):
2 8 3
1 0 4
7 6 5
Step 2 (U):
2 0 3
1 8 4
7 6 5
Step 3 (L):
0 2 3
1 8 4
7 6 5
Step 4 (D):
1 2 3
0 8 4
7 6 5
Step 5 (R):
1 2 3
8 0 4
7 6 5

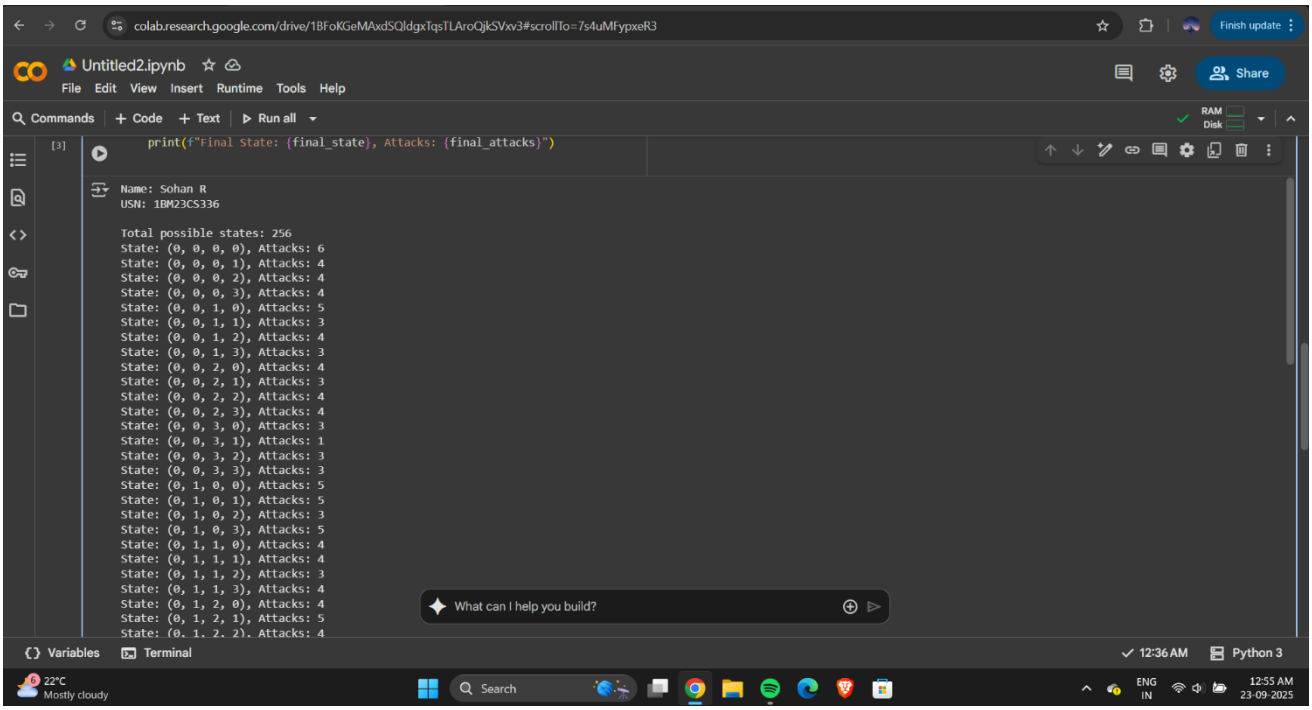
Program 4

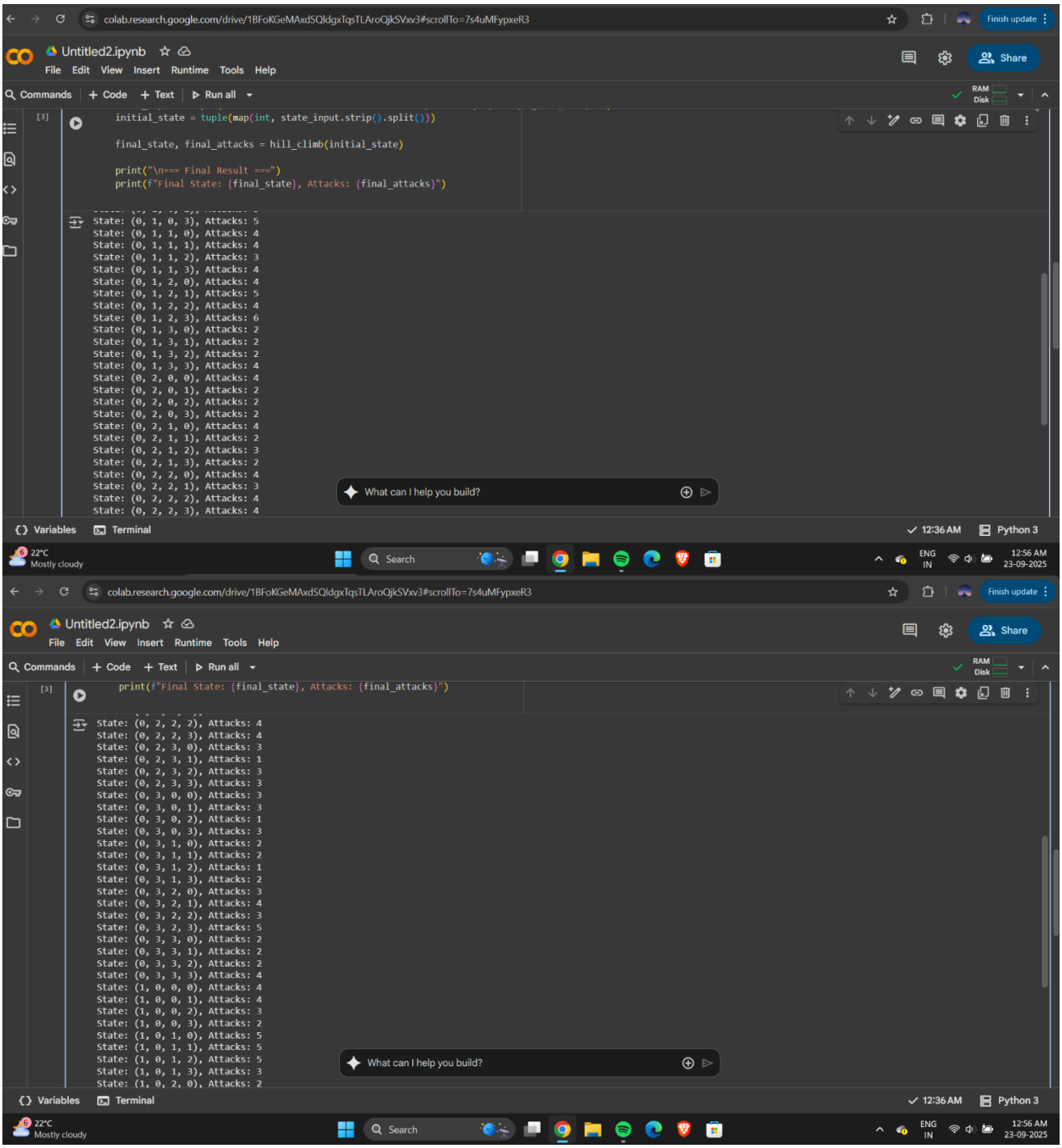
Implement Hill Climbing search algorithm to solve N-Queens problem

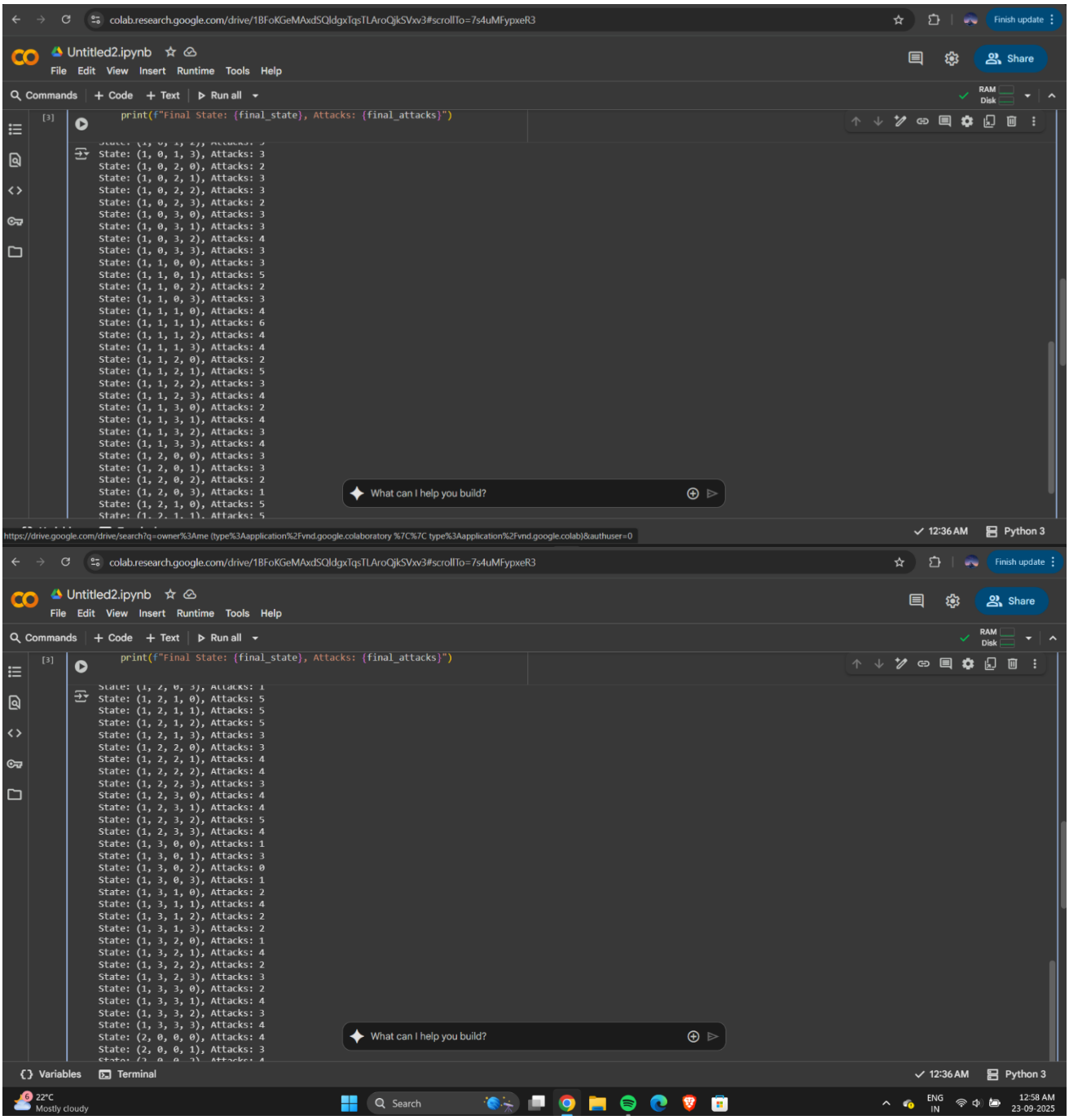
Code:

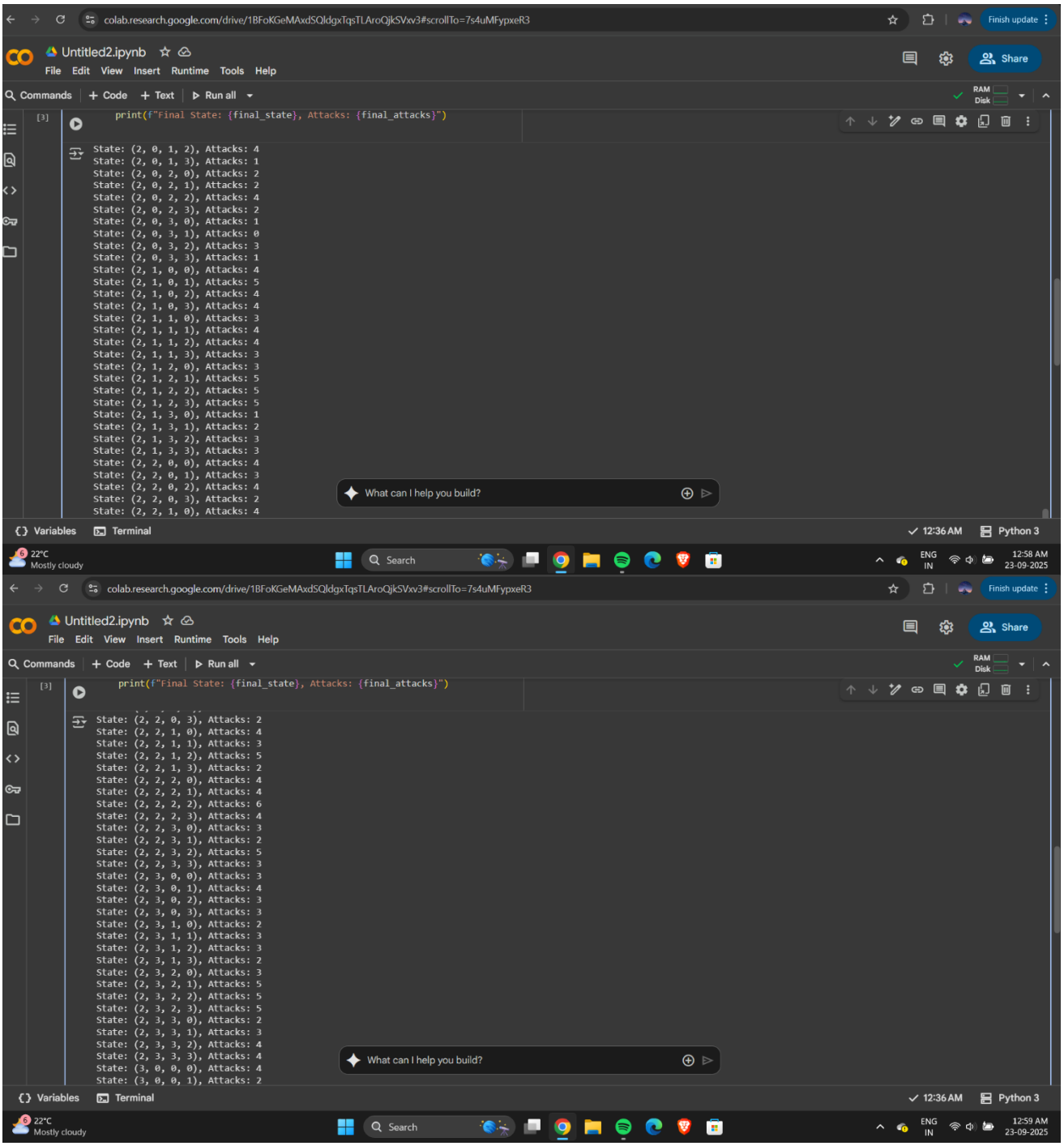
```
import itertools

def calculate_attacks(state):
    attacks = 0
    n = len(state)
    for i in range(n):
        for j in range(i + 1, n):
            if state[i] == state[j]:
                attacks += 1
            if abs(state[i] - state[j]) == abs(i - j):
                attacks += 1
    return attacks

def get_neighbors(state):
    neighbors = []
    n = len(state)
    for col in range(n):
        for row in range(n):
            if state[col] != row:
                new_state = list(state)
                new_state[col] = row
                neighbors.append(tuple(new_state))
    return neighbors

def hill_climb(initial_state):
    current = initial_state
    current_attacks = calculate_attacks(current)

    print(f"\nStarting Hill Climb from: {current}, Attacks: {current_attacks}")
    while True:
        neighbors = get_neighbors(current)
        if not neighbors:
            break
        neighbor = min(neighbors, key=lambda x: calculate_attacks(x))
        neighbor_attacks = calculate_attacks(neighbor)
        if neighbor_attacks >= current_attacks:
            break
        current, current_attacks = neighbor, neighbor_attacks
        print(f"Move to: {current}, Attacks: {current_attacks}")
    return current, current_attacks

if __name__ == "__main__":
```

```
print("Name: Sohan R")
print("USN: 1BM23CS336")
print(f"\nTotal possible states: {4**4} ")

all_states = itertools.product(range(4), repeat=4)
for state in all_states:
    print(f"State: {state}, Attacks: {calculate_attacks(state)}")

state_input = input("\nEnter initial state as 4 numbers separated by space (e.g. 0 1 2 3): ")
initial_state = tuple(map(int, state_input.strip().split()))

final_state, final_attacks = hill_climb(initial_state)

print("\n=== Final Result ===")
print(f"Final State: {final_state}, Attacks: {final_attacks}")
```

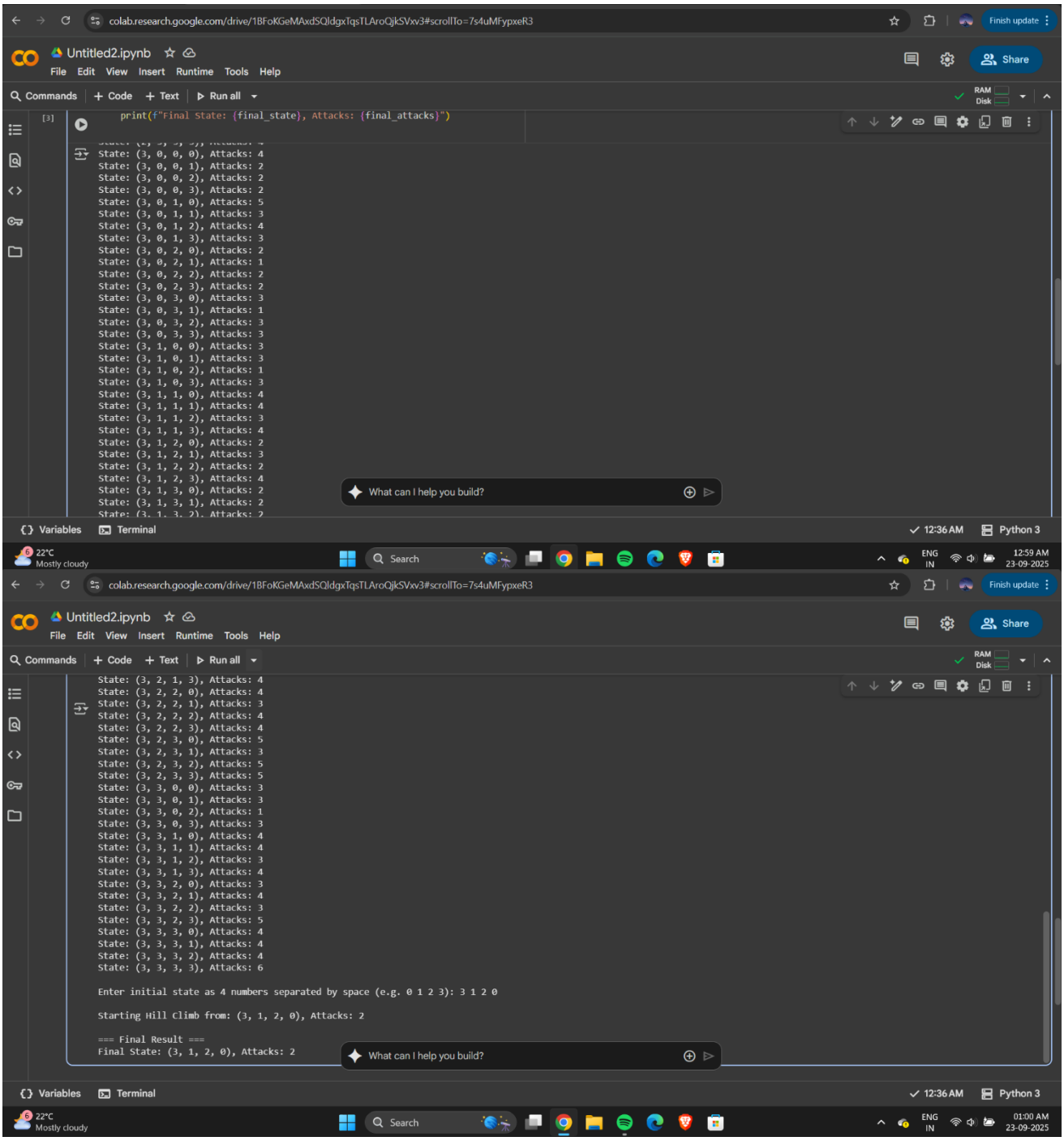



colab.research.google.com/drive/1BFoKGeMAxdSQldgxTqsTLAroQjkSVxv3#scrollTo=7s4uMFypxeR3
Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
initial_state = tuple(map(int, state_input.strip().split()))
final_state, final_attacks = hill_climb(initial_state)
print("\n=== Final Result ===")
print(f"Final State: {final_state}, Attacks: {final_attacks}")
State: (0, 1, 0, 3), Attacks: 5
State: (0, 1, 1, 0), Attacks: 4
State: (0, 1, 1, 1), Attacks: 4
State: (0, 1, 1, 2), Attacks: 3
State: (0, 1, 1, 3), Attacks: 4
State: (0, 1, 2, 0), Attacks: 4
State: (0, 1, 2, 1), Attacks: 5
State: (0, 1, 2, 2), Attacks: 4
State: (0, 1, 2, 3), Attacks: 6
State: (0, 1, 3, 0), Attacks: 2
State: (0, 1, 3, 1), Attacks: 2
State: (0, 1, 3, 2), Attacks: 2
State: (0, 1, 3, 3), Attacks: 4
State: (0, 2, 0, 0), Attacks: 4
State: (0, 2, 0, 1), Attacks: 2
State: (0, 2, 0, 2), Attacks: 2
State: (0, 2, 0, 3), Attacks: 2
State: (0, 2, 1, 0), Attacks: 4
State: (0, 2, 1, 1), Attacks: 2
State: (0, 2, 1, 2), Attacks: 3
State: (0, 2, 1, 3), Attacks: 2
State: (0, 2, 2, 0), Attacks: 4
State: (0, 2, 2, 1), Attacks: 3
State: (0, 2, 2, 2), Attacks: 4
State: (0, 2, 2, 3), Attacks: 4
What can I help you build?
Variables Terminal
12:36 AM Python 3
22°C Mostly cloudy
Search
ENG IN
12:56 AM 23-09-2025
colab.research.google.com/drive/1BFoKGeMAxdSQldgxTqsTLAroQjkSVxv3#scrollTo=7s4uMFypxeR3
Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
print(f"Final State: {final_state}, Attacks: {final_attacks}")
State: (0, 2, 2, 2), Attacks: 4
State: (0, 2, 2, 3), Attacks: 4
State: (0, 2, 3, 0), Attacks: 3
State: (0, 2, 3, 1), Attacks: 1
State: (0, 2, 3, 2), Attacks: 3
State: (0, 2, 3, 3), Attacks: 3
State: (0, 3, 0, 0), Attacks: 3
State: (0, 3, 0, 1), Attacks: 3
State: (0, 3, 0, 2), Attacks: 1
State: (0, 3, 0, 3), Attacks: 3
State: (0, 3, 1, 0), Attacks: 2
State: (0, 3, 1, 1), Attacks: 2
State: (0, 3, 1, 2), Attacks: 1
State: (0, 3, 1, 3), Attacks: 2
State: (0, 3, 2, 0), Attacks: 3
State: (0, 3, 2, 1), Attacks: 4
State: (0, 3, 2, 2), Attacks: 3
State: (0, 3, 2, 3), Attacks: 5
State: (0, 3, 3, 0), Attacks: 2
State: (0, 3, 3, 1), Attacks: 2
State: (0, 3, 3, 2), Attacks: 2
State: (0, 3, 3, 3), Attacks: 4
State: (1, 0, 0, 0), Attacks: 4
State: (1, 0, 0, 1), Attacks: 4
State: (1, 0, 0, 2), Attacks: 3
State: (1, 0, 0, 3), Attacks: 2
State: (1, 0, 1, 0), Attacks: 5
State: (1, 0, 1, 1), Attacks: 5
State: (1, 0, 1, 2), Attacks: 5
State: (1, 0, 1, 3), Attacks: 3
State: (1, 0, 2, 0), Attacks: 2
What can I help you build?
Variables Terminal
12:36 AM Python 3
22°C Mostly cloudy
Search
ENG IN
12:56 AM 23-09-2025


colab.research.google.com/drive/1BFoKGeMAxdSQldgxTqsTLAroQjkSVxv3#scrollTo=7s4uMFypxeR3
Finish update
Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM
Disk
[3]
print(f"Final State: {final_state}, Attacks: {final_attacks}")
State: (1, 0, 1, 3), Attacks: 3
State: (1, 0, 2, 0), Attacks: 2
State: (1, 0, 2, 1), Attacks: 3
State: (1, 0, 2, 2), Attacks: 3
State: (1, 0, 2, 3), Attacks: 2
State: (1, 0, 3, 0), Attacks: 3
State: (1, 0, 3, 1), Attacks: 3
State: (1, 0, 3, 2), Attacks: 4
State: (1, 0, 3, 3), Attacks: 3
State: (1, 1, 0, 0), Attacks: 3
State: (1, 1, 0, 1), Attacks: 5
State: (1, 1, 0, 2), Attacks: 2
State: (1, 1, 0, 3), Attacks: 3
State: (1, 1, 1, 0), Attacks: 4
State: (1, 1, 1, 1), Attacks: 6
State: (1, 1, 1, 2), Attacks: 4
State: (1, 1, 1, 3), Attacks: 4
State: (1, 1, 2, 0), Attacks: 2
State: (1, 1, 2, 1), Attacks: 5
State: (1, 1, 2, 2), Attacks: 3
State: (1, 1, 2, 3), Attacks: 4
State: (1, 1, 3, 0), Attacks: 2
State: (1, 1, 3, 1), Attacks: 4
State: (1, 1, 3, 2), Attacks: 3
State: (1, 1, 3, 3), Attacks: 4
State: (1, 2, 0, 0), Attacks: 3
State: (1, 2, 0, 1), Attacks: 3
State: (1, 2, 0, 2), Attacks: 2
State: (1, 2, 0, 3), Attacks: 1
State: (1, 2, 1, 0), Attacks: 5
What can I help you build?
12:36 AM
Python 3
colab.research.google.com/drive/1BFoKGeMAxdSQldgxTqsTLAroQjkSVxv3#scrollTo=7s4uMFypxeR3
Finish update
Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM
Disk
[3]
print(f"Final State: {final_state}, Attacks: {final_attacks}")
State: (1, 2, 1, 0), Attacks: 5
State: (1, 2, 1, 1), Attacks: 5
State: (1, 2, 1, 2), Attacks: 5
State: (1, 2, 1, 3), Attacks: 3
State: (1, 2, 2, 0), Attacks: 3
State: (1, 2, 2, 1), Attacks: 4
State: (1, 2, 2, 2), Attacks: 4
State: (1, 2, 2, 3), Attacks: 3
State: (1, 2, 3, 0), Attacks: 4
State: (1, 2, 3, 1), Attacks: 4
State: (1, 2, 3, 2), Attacks: 5
State: (1, 2, 3, 3), Attacks: 4
State: (1, 3, 0, 0), Attacks: 1
State: (1, 3, 0, 1), Attacks: 3
State: (1, 3, 0, 2), Attacks: 0
State: (1, 3, 0, 3), Attacks: 1
State: (1, 3, 1, 0), Attacks: 2
State: (1, 3, 1, 1), Attacks: 4
State: (1, 3, 1, 2), Attacks: 2
State: (1, 3, 1, 3), Attacks: 2
State: (1, 3, 2, 0), Attacks: 1
State: (1, 3, 2, 1), Attacks: 4
State: (1, 3, 2, 2), Attacks: 2
State: (1, 3, 2, 3), Attacks: 3
State: (1, 3, 3, 0), Attacks: 2
State: (1, 3, 3, 1), Attacks: 4
State: (1, 3, 3, 2), Attacks: 3
State: (1, 3, 3, 3), Attacks: 4
State: (2, 0, 0, 0), Attacks: 4
State: (2, 0, 0, 1), Attacks: 3
What can I help you build?
Variables
Terminal
12:36 AM
Python 3
22°C
Mostly cloudy
Search
ENG
IN
12:58 AM
23-09-2025


colab.research.google.com/drive/1BFoKGeMAxdSQldgxTqsTLAroQjkSVxv3#scrollTo=7s4uMFypxeR3
Finish update
Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM Disk
[3]
print(f"Final State: {final_state}, Attacks: {final_attacks}")
State: (2, 0, 1, 2), Attacks: 4
State: (2, 0, 1, 3), Attacks: 1
State: (2, 0, 2, 0), Attacks: 2
State: (2, 0, 2, 1), Attacks: 2
State: (2, 0, 2, 2), Attacks: 4
State: (2, 0, 2, 3), Attacks: 2
State: (2, 0, 3, 0), Attacks: 1
State: (2, 0, 3, 1), Attacks: 0
State: (2, 0, 3, 2), Attacks: 3
State: (2, 0, 3, 3), Attacks: 1
State: (2, 1, 0, 0), Attacks: 4
State: (2, 1, 0, 1), Attacks: 5
State: (2, 1, 0, 2), Attacks: 4
State: (2, 1, 0, 3), Attacks: 4
State: (2, 1, 1, 0), Attacks: 3
State: (2, 1, 1, 1), Attacks: 4
State: (2, 1, 1, 2), Attacks: 4
State: (2, 1, 1, 3), Attacks: 3
State: (2, 1, 2, 0), Attacks: 3
State: (2, 1, 2, 1), Attacks: 5
State: (2, 1, 2, 2), Attacks: 5
State: (2, 1, 2, 3), Attacks: 5
State: (2, 1, 3, 0), Attacks: 1
State: (2, 1, 3, 1), Attacks: 2
State: (2, 1, 3, 2), Attacks: 3
State: (2, 1, 3, 3), Attacks: 3
State: (2, 2, 0, 0), Attacks: 4
State: (2, 2, 0, 1), Attacks: 3
State: (2, 2, 0, 2), Attacks: 4
State: (2, 2, 0, 3), Attacks: 2
State: (2, 2, 1, 0), Attacks: 4
What can I help you build?
Variables Terminal
12:36 AM Python 3
22°C Mostly cloudy
Search
ENG IN
12:58 AM 23-09-2025

colab.research.google.com/drive/1BFoKGeMAxdSQldgxTqsTLAroQjkSVxv3#scrollTo=7s4uMFypxeR3
Finish update
Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM Disk
[3]
print(f"Final State: {final_state}, Attacks: {final_attacks}")
State: (2, 2, 0, 3), Attacks: 2
State: (2, 2, 1, 0), Attacks: 4
State: (2, 2, 1, 1), Attacks: 3
State: (2, 2, 1, 2), Attacks: 5
State: (2, 2, 1, 3), Attacks: 2
State: (2, 2, 2, 0), Attacks: 4
State: (2, 2, 2, 1), Attacks: 4
State: (2, 2, 2, 2), Attacks: 6
State: (2, 2, 2, 3), Attacks: 4
State: (2, 2, 3, 0), Attacks: 3
State: (2, 2, 3, 1), Attacks: 2
State: (2, 2, 3, 2), Attacks: 5
State: (2, 2, 3, 3), Attacks: 3
State: (2, 3, 0, 0), Attacks: 3
State: (2, 3, 0, 1), Attacks: 4
State: (2, 3, 0, 2), Attacks: 3
State: (2, 3, 0, 3), Attacks: 3
State: (2, 3, 1, 0), Attacks: 2
State: (2, 3, 1, 1), Attacks: 3
State: (2, 3, 1, 2), Attacks: 3
State: (2, 3, 1, 3), Attacks: 2
State: (2, 3, 2, 0), Attacks: 3
State: (2, 3, 2, 1), Attacks: 5
State: (2, 3, 2, 2), Attacks: 5
State: (2, 3, 2, 3), Attacks: 5
State: (2, 3, 3, 0), Attacks: 2
State: (2, 3, 3, 1), Attacks: 3
State: (2, 3, 3, 2), Attacks: 4
State: (2, 3, 3, 3), Attacks: 4
State: (3, 0, 0, 0), Attacks: 4
State: (3, 0, 0, 1), Attacks: 2
What can I help you build?
Variables Terminal
12:36 AM Python 3
22°C Mostly cloudy
Search
ENG IN
12:59 AM 23-09-2025


colab.research.google.com/drive/1BFoKGeMAxdSQldgxTqsTLAroQjkSVxv3#scrollTo=7s4uMFypxeR3
Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
print(f"Final State: {final_state}, Attacks: {final_attacks}")
State: (3, 0, 0, 0), Attacks: 4
State: (3, 0, 0, 1), Attacks: 2
State: (3, 0, 0, 2), Attacks: 2
State: (3, 0, 0, 3), Attacks: 2
State: (3, 0, 1, 0), Attacks: 5
State: (3, 0, 1, 1), Attacks: 3
State: (3, 0, 1, 2), Attacks: 4
State: (3, 0, 1, 3), Attacks: 3
State: (3, 0, 2, 0), Attacks: 2
State: (3, 0, 2, 1), Attacks: 1
State: (3, 0, 2, 2), Attacks: 2
State: (3, 0, 2, 3), Attacks: 2
State: (3, 0, 3, 0), Attacks: 3
State: (3, 0, 3, 1), Attacks: 1
State: (3, 0, 3, 2), Attacks: 3
State: (3, 0, 3, 3), Attacks: 3
State: (3, 1, 0, 0), Attacks: 3
State: (3, 1, 0, 1), Attacks: 3
State: (3, 1, 0, 2), Attacks: 1
State: (3, 1, 0, 3), Attacks: 3
State: (3, 1, 1, 0), Attacks: 4
State: (3, 1, 1, 1), Attacks: 4
State: (3, 1, 1, 2), Attacks: 3
State: (3, 1, 1, 3), Attacks: 4
State: (3, 1, 2, 0), Attacks: 2
State: (3, 1, 2, 1), Attacks: 3
State: (3, 1, 2, 2), Attacks: 2
State: (3, 1, 2, 3), Attacks: 4
State: (3, 1, 3, 0), Attacks: 2
State: (3, 1, 3, 1), Attacks: 2
What can I help you build?
Variables Terminal
12:36 AM Python 3
22°C Mostly cloudy
Search
ENG IN
12:59 AM 23-09-2025
colab.research.google.com/drive/1BFoKGeMAxdSQldgxTqsTLAroQjkSVxv3#scrollTo=7s4uMFypxeR3
Untitled2.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
State: (3, 2, 1, 3), Attacks: 4
State: (3, 2, 2, 0), Attacks: 4
State: (3, 2, 2, 1), Attacks: 3
State: (3, 2, 2, 2), Attacks: 4
State: (3, 2, 2, 3), Attacks: 4
State: (3, 2, 3, 0), Attacks: 5
State: (3, 2, 3, 1), Attacks: 3
State: (3, 2, 3, 2), Attacks: 5
State: (3, 2, 3, 3), Attacks: 5
State: (3, 3, 0, 0), Attacks: 3
State: (3, 3, 0, 1), Attacks: 3
State: (3, 3, 0, 2), Attacks: 1
State: (3, 3, 0, 3), Attacks: 3
State: (3, 3, 1, 0), Attacks: 4
State: (3, 3, 1, 1), Attacks: 4
State: (3, 3, 1, 2), Attacks: 3
State: (3, 3, 1, 3), Attacks: 4
State: (3, 3, 2, 0), Attacks: 3
State: (3, 3, 2, 1), Attacks: 4
State: (3, 3, 2, 2), Attacks: 3
State: (3, 3, 2, 3), Attacks: 5
State: (3, 3, 3, 0), Attacks: 4
State: (3, 3, 3, 1), Attacks: 4
State: (3, 3, 3, 2), Attacks: 4
State: (3, 3, 3, 3), Attacks: 6
Enter initial state as 4 numbers separated by space (e.g. 0 1 2 3): 3 1 2 0
Starting Hill Climb from: (3, 1, 2, 0), Attacks: 2
=== Final Result ===
Final State: (3, 1, 2, 0), Attacks: 2
What can I help you build?
Variables Terminal
12:36 AM Python 3
22°C Mostly cloudy
Search
ENG IN
01:00 AM 23-09-2025

Date __/__/__
Page _____

8. Implement Hill climbing algorithm to solve N-Queens problem.

Ans:

Algorithm:-

function HILL-CLIMBING(problem) returns a state that is a local maximum current ← MAKE-NODE(problem, INITIAL-STATE)

loop do
neighbor ← a highest-valued successor of current
if neighbor.VALUE ≤ current.VALUE then return current.STATE
current ← neighbor.

- State: 4 queens on the board. No pair of queens are attacking each other.

~~Neighbor~~

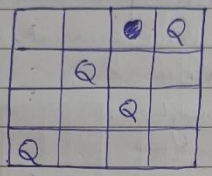
– Variables: x_0, x_1, x_2, x_3 where x_i is the row position of the queen in column i. Assume that there is one queen per column

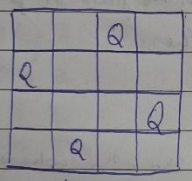
– Domain for each variable: $x_i \in \{0, 1, 2, 3\}$ $\forall i$.

- Initial state: a random state
- Goal state: 4 queens on the board. No pair of queens are attacking each other.
- Neighbor relation: Swap the row positions of two queens.
- Cost function: The number of pairs of queens attacking each other, directly or indirectly.

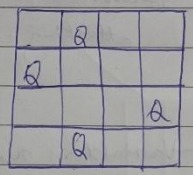
Transition diagram:-

Initial = [3, 1, 2, 0]



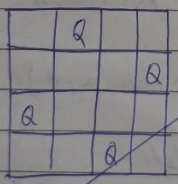


h = 0

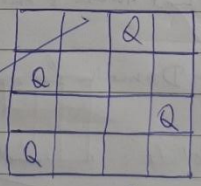


h = 3

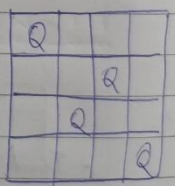
h = 4



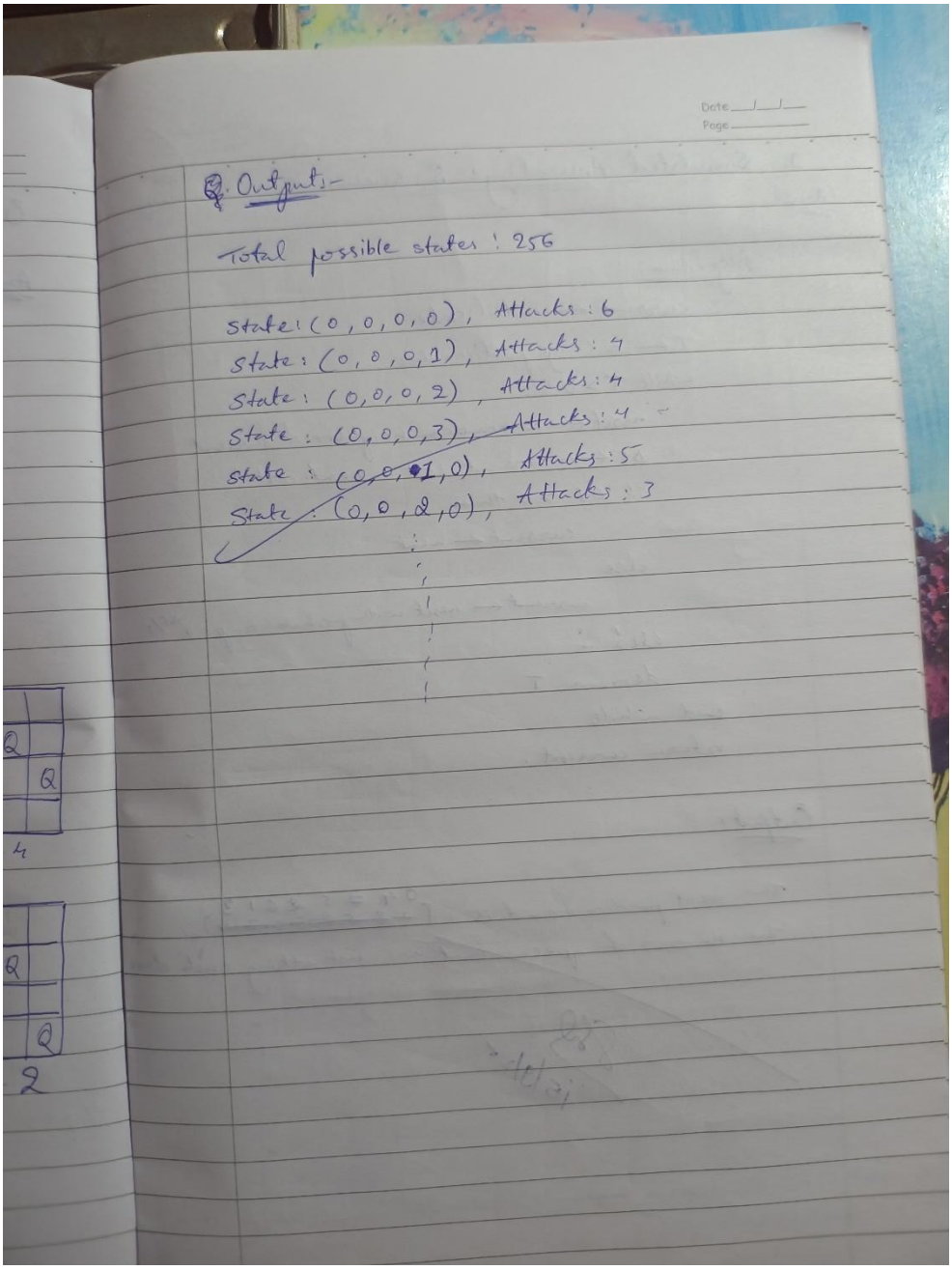
h = 0



h = 2



h = 2


Output:-
Total possible states : 256
State: (0,0,0,0), Attacks : 6
State: (0,0,0,1), Attacks : 4
State: (0,0,0,2), Attacks : 4
State: (0,0,0,3), Attacks : 4
State: (0,0,1,0), Attacks : 5
State: (0,0,2,0), Attacks : 3
Q
Q
4
Q
Q
2

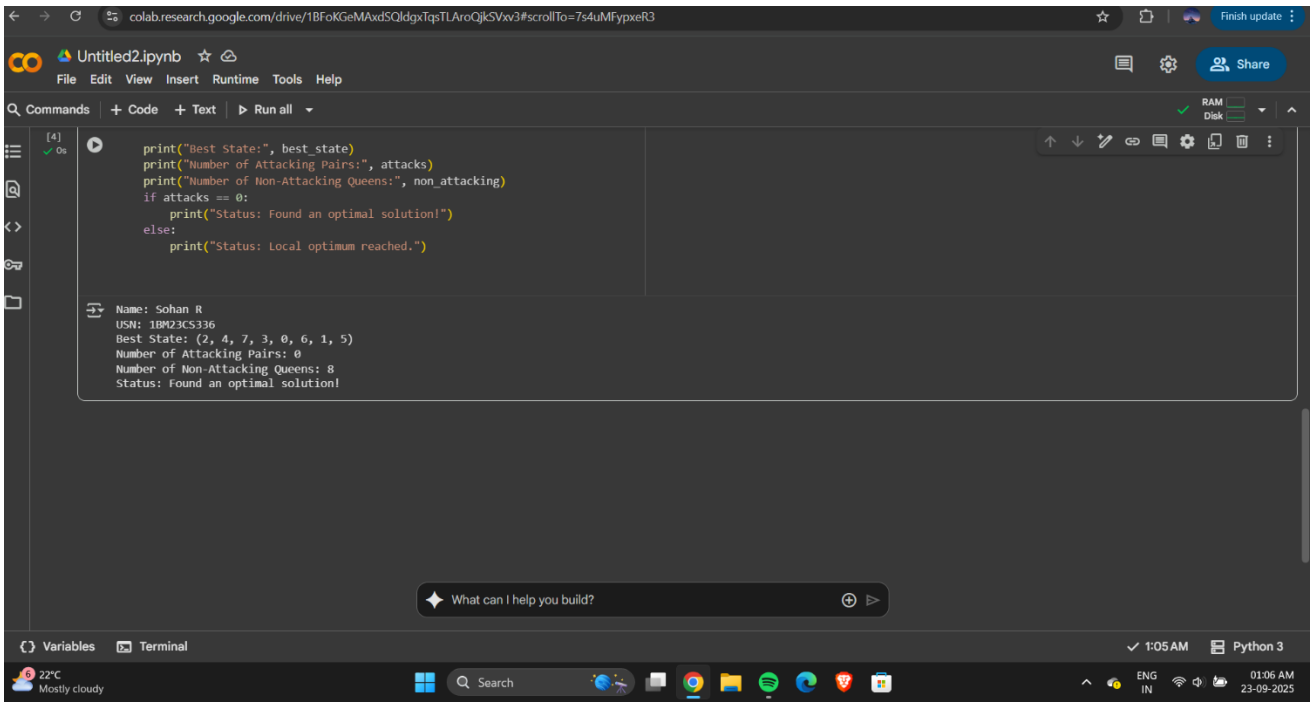
Program 5

Simulated Annealing to Solve 8-Queens problem

Code:

```
import random
import math

def calculate_attacks(state):
    attacks = 0
    n = len(state)
    for i in range(n):
        for j in range(i + 1, n):
            if state[i] == state[j]:
                attacks += 1
            if abs(state[i] - state[j]) == abs(i - j):
                attacks += 1
    return attacks

def get_random_neighbor(state):
    n = len(state)
    col = random.randint(0, n - 1)
    row = random.randint(0, n - 1)
    while row == state[col]:
        row = random.randint(0, n - 1)
    new_state = list(state)
    new_state[col] = row
    return tuple(new_state)

def simulated_annealing(n=8, max_iterations=10000, temperature=100.0, cooling_rate=0.99):
    current = tuple(random.randint(0, n - 1) for _ in range(n))
    current_attacks = calculate_attacks(current)
    best = current
    best_attacks = current_attacks

    for _ in range(max_iterations):
        if current_attacks == 0:
            break
        neighbor = get_random_neighbor(current)
        neighbor_attacks = calculate_attacks(neighbor)
        delta = current_attacks - neighbor_attacks
        if delta > 0 or random.random() < math.exp(delta / temperature):
            current, current_attacks = neighbor, neighbor_attacks
            if current_attacks < best_attacks:
                best, best_attacks = current, current_attacks
        temperature *= cooling_rate
```

```
    return best, best_attacks

if __name__ == "__main__":
    print("Name: Sohan R")
    print("USN: 1BM23CS336")

    best_state, attacks = simulated_annealing()
    non_attacking = 8 - attacks

    print("Best State:", best_state)
    print("Number of Attacking Pairs:", attacks)
    print("Number of Non-Attacking Queens:", non_attacking)
    if attacks == 0:
        print("Status: Found an optimal solution!")
    else:
        print("Status: Local optimum reached.")
```



9. Simulated Annealing: 8-Queens.

Ans.

Algorithm:

```
current ← initial state
T ← a large positive value
while T > 0 do
    next ← a random neighbor of current
    ΔE ← current.cost − next.cost
    if ΔE > 0 then
        current ← next
    else
        current ← next with probability p = e^(ΔE/T)
    end if
    decrease T
end while
return current.
```

Output:

The best position found is : [0 4 7 5 2 6 1 3]

The number of queens that are not atking each other : 8

15/9/24

10. Ans

Program 6

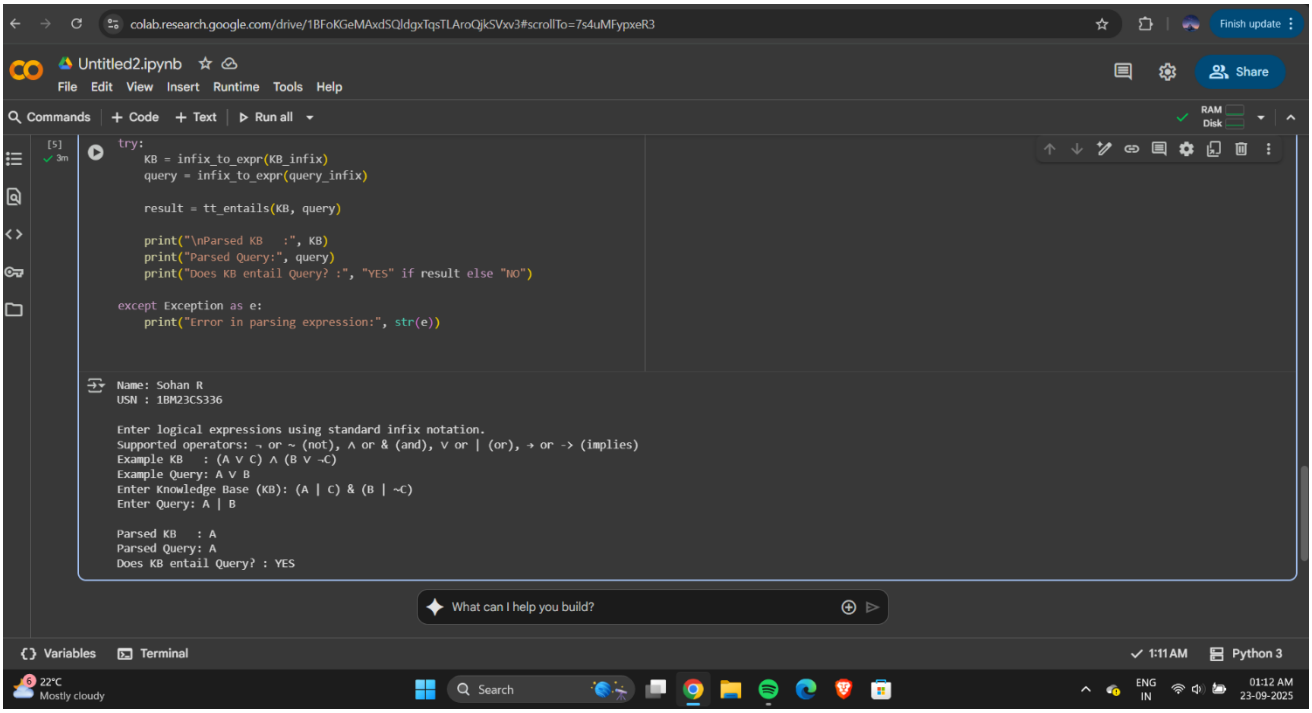
Create a knowledge base using propositional logic and show that the given query entails the knowledge base or not.

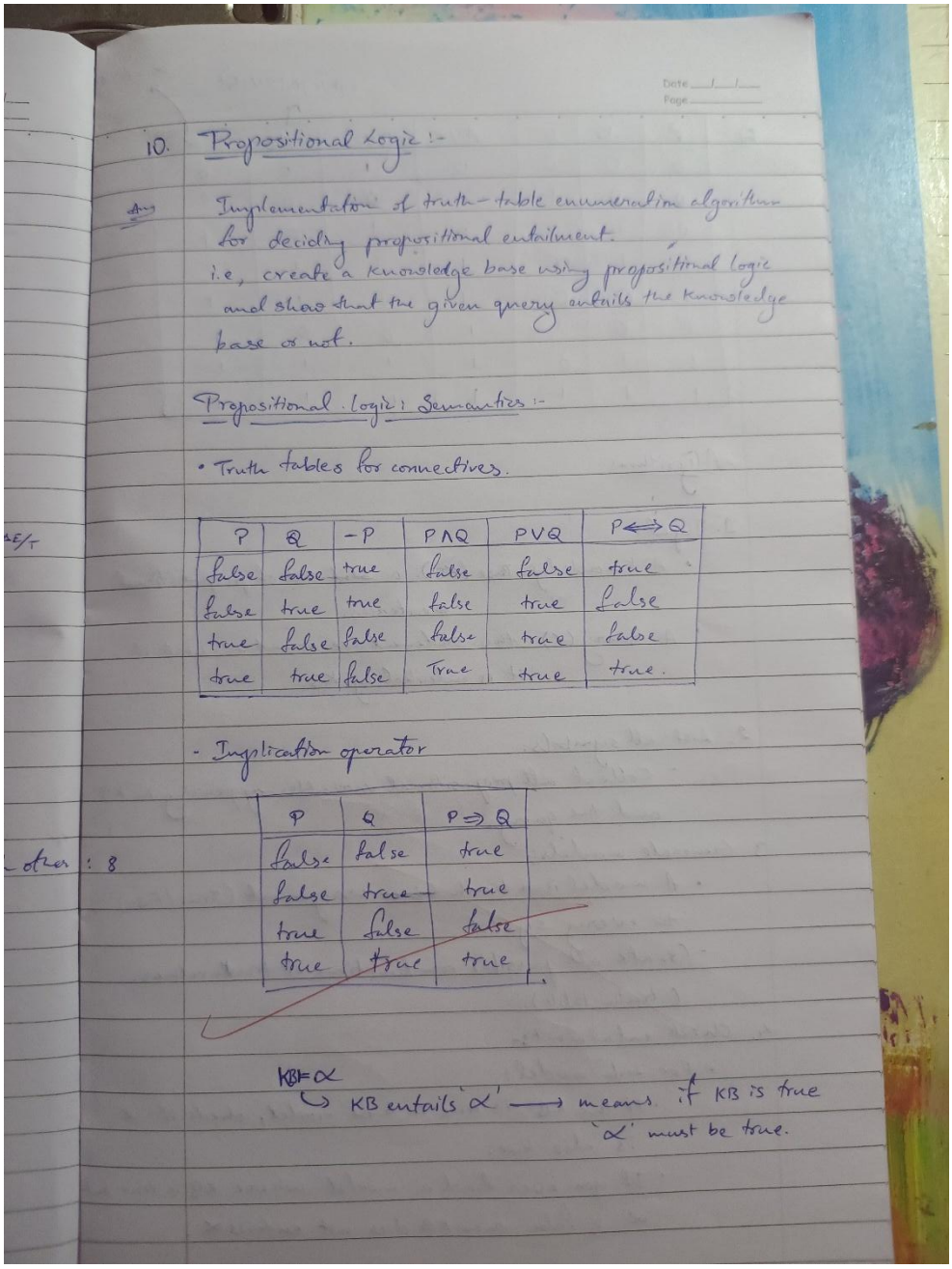
Code:

```
import re
def pl_true(expr, model):
    if isinstance(expr, str):
        return model[expr]
    elif isinstance(expr, tuple):
        op = expr[0]
        if op == "not":
            return not pl_true(expr[1], model)
        elif op == "and":
            return pl_true(expr[1], model) and pl_true(expr[2], model)
        elif op == "or":
            return pl_true(expr[1], model) or pl_true(expr[2], model)
        elif op == "implies":
            return (not pl_true(expr[1], model)) or pl_true(expr[2], model)
    return False

def tt_entails(KB, query):
    symbols = list(get_symbols(KB) | get_symbols(query))
    return check_all(KB, query, symbols, {})

def get_symbols(expr):
    if isinstance(expr, str):
        return {expr}
    elif isinstance(expr, tuple):
        return get_symbols(expr[1]) | (get_symbols(expr[2]) if len(expr) > 2 else set())
    return set()

def check_all(KB, query, symbols, model):
    if not symbols:
        if pl_true(KB, model):
            return pl_true(query, model)
        else:
            return True
    else:
        rest = symbols[1:]
        p = symbols[0]
        m1 = model.copy(); m1[p] = True
        m2 = model.copy(); m2[p] = False
        return check_all(KB, query, rest, m1) and check_all(KB, query, rest, m2)

precedence = {
```

```
    'not': 3,
    'and': 2,
    'or': 1,
    'implies': 0
}

def infix_to_expr(infix_str):
    # Replace common symbols with standard keywords
    infix_str = infix_str.replace("¬", "not").replace("~", "not")
    infix_str = infix_str.replace("∧", "and").replace("&", "and")
    infix_str = infix_str.replace("∨", "or").replace("|", "or")
    infix_str = infix_str.replace("→", "implies").replace("->", "implies")

    tokens = re.findall(r'[A-Za-z]+|not|and|or|implies|\(|\)', infix_str)

    output = []
    ops = []

    def pop_op():
        op = ops.pop()
        if op == "not":
            right = output.pop()
            output.append((op, right))
        else:
            right = output.pop()
            left = output.pop()
            output.append((op, left, right))

    for token in tokens:
        if re.fullmatch(r'[A-Za-z]+', token):  # symbol
            output.append(token)
        elif token == '(':
            ops.append(token)
        elif token == ')':
            while ops and ops[-1] != '(':
                pop_op()
            ops.pop()  # remove '('
        elif token in precedence:
            while (ops and ops[-1] in precedence and
                   precedence[ops[-1]] >= precedence[token]):
                pop_op()
            ops.append(token)

    while ops:
        pop_op()

    return output[0]
```

```
print("Name: Sohan R")
print("USN : 1BM23CS336")

print("\nEnter logical expressions using standard infix notation.")
print("Supported operators: ¬ or ~ (not), ∧ or & (and), ∨ or | (or), → or -> (implies)")
print("Example KB   : (A ∨ C) ∧ (B ∨ ¬C)")
print("Example Query: A ∨ B")

KB_infix = input("Enter Knowledge Base (KB): ")
query_infix = input("Enter Query: ")

try:
    KB = infix_to_expr(KB_infix)
    query = infix_to_expr(query_infix)

    result = tt_entails(KB, query)

    print("\nParsed KB   :", KB)
    print("Parsed Query:", query)
    print("Does KB entail Query? :", "YES" if result else "NO")

except Exception as e:
    print("Error in parsing expression:", str(e))
```





10. Propositional Logic :-

Aim Implementation of truth-table enumeration algorithm for deciding propositional entailment.
i.e, create a knowledge base using propositional logic and show that the given query entails the knowledge base or not.

Propositional Logic : Semantics :-

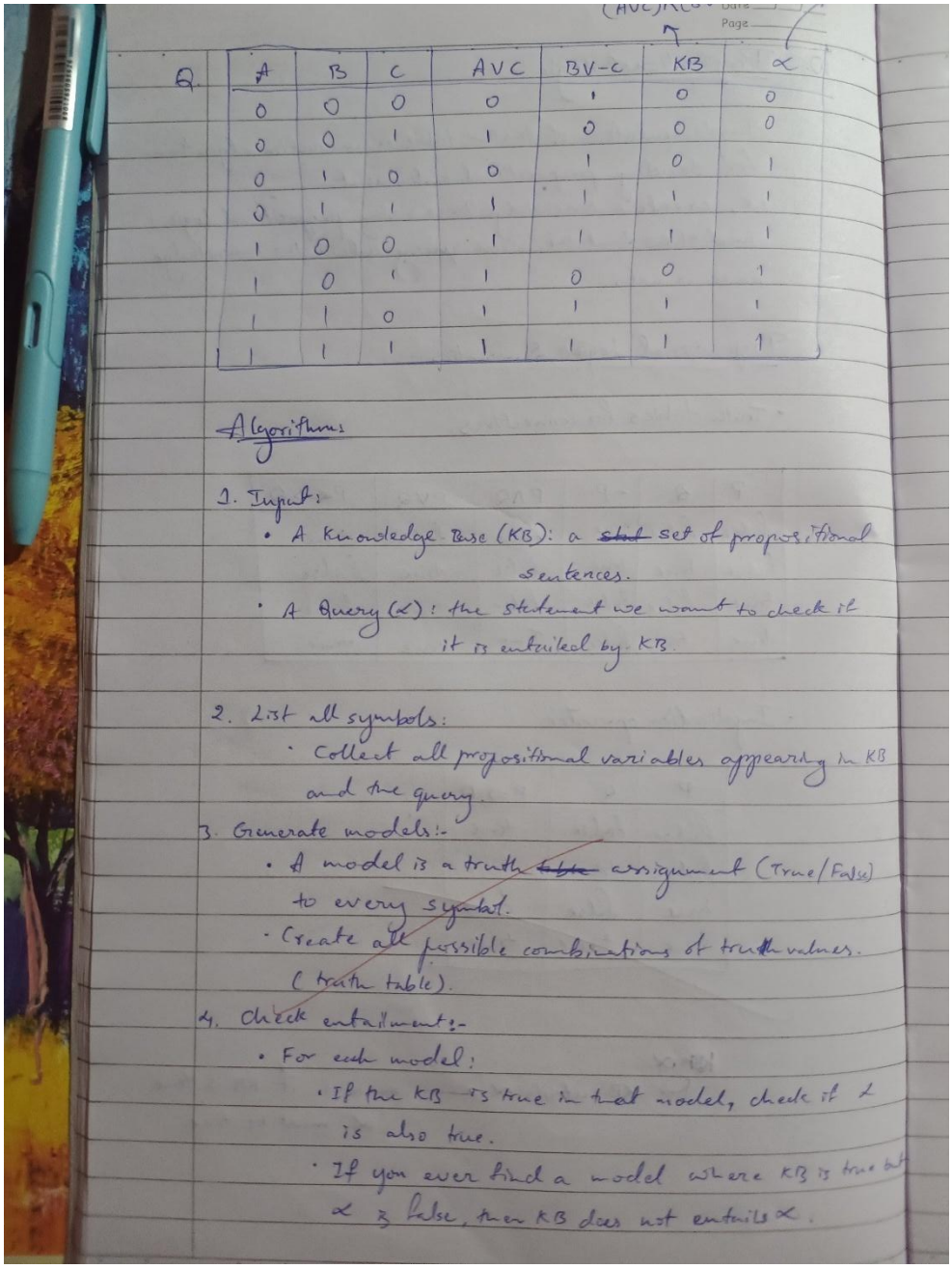
- Truth tables for connectives.

P	Q	¬P	P∧Q	P∨Q	P⟺Q
false	false	true	false	false	true
false	true	true	false	true	false
true	false	false	false	true	false
true	true	false	True	true	true.

- Implication operator

P	Q	P⇒Q
false	false	true
false	true	true
true	false	false
true	true	true

KB⊨α
↳ KB entails 'α' ⟶ means if KB is true 'α' must be true.

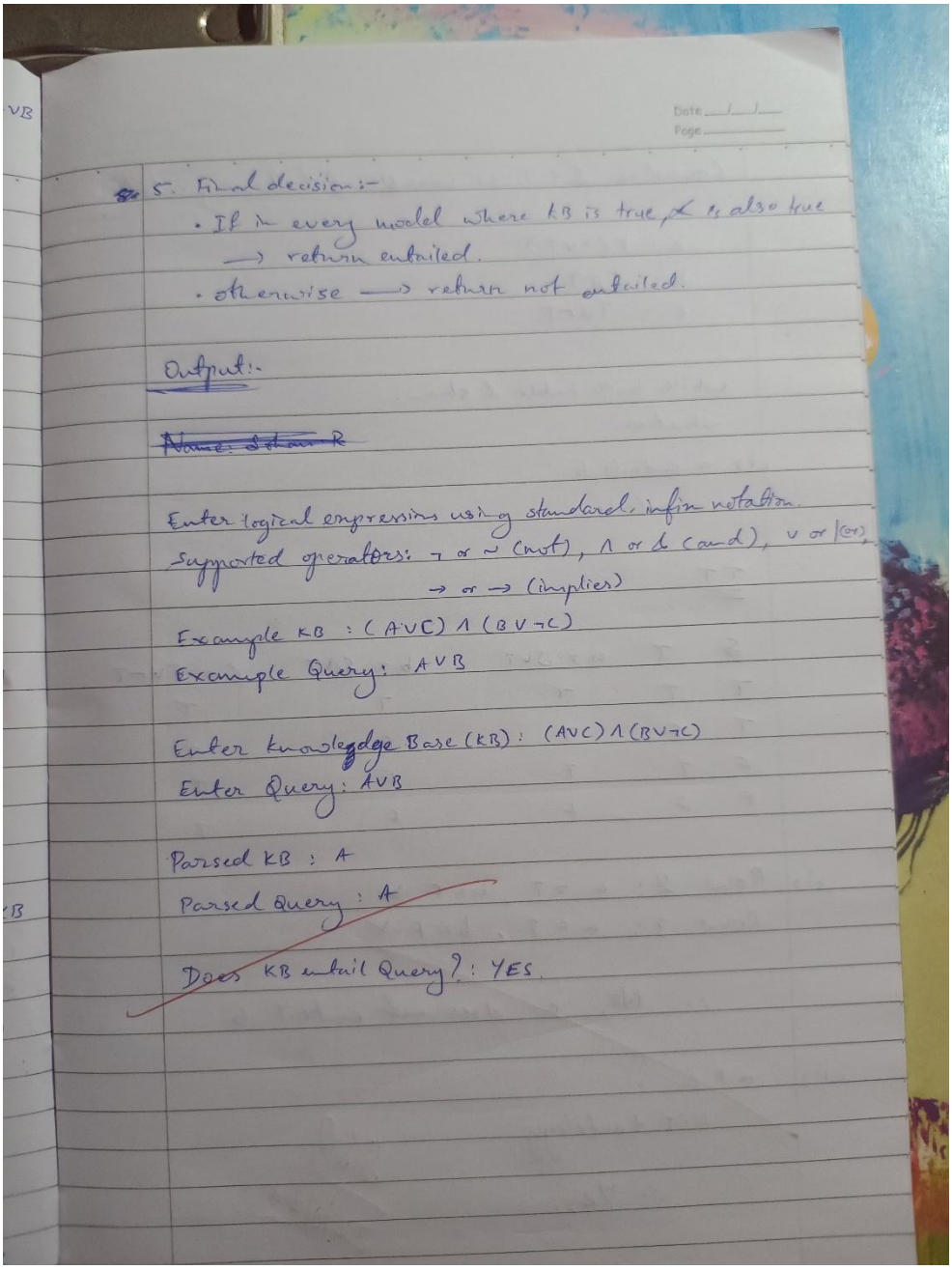


Q.

A	B	C	A ∨ C	B ∨ ¬C	KB	α
0	0	0	0	1	0	0
0	0	1	1	0	0	0
0	1	0	0	1	0	1
0	1	1	1	1	1	1
1	0	0	1	1	1	1
1	0	1	1	0	0	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Algorithm:

1. Input:
   - A Knowledge Base (KB): a ~~short~~ set of propositional sentences.
   - A Query (α): the statement we want to check if it is entailed by KB.
2. List all symbols:
   - Collect all propositional variables appearing in KB and the query.
3. Generate models:-
   - A model is a truth ~~table~~ assignment (True/False) to every symbol.
   - Create all possible combinations of truth values. (truth table).
4. Check entailment:-
   - For each model:
     - If the KB is true in that model, check if α is also true.
     - If you ever find a model where KB is true but α is false, then KB does not entails α.



5. Final decision:-
- If in every model where KB is true, α is also true
  → return entailed.
- otherwise → return not entailed.

Output:-

~~Name: Sharan R~~

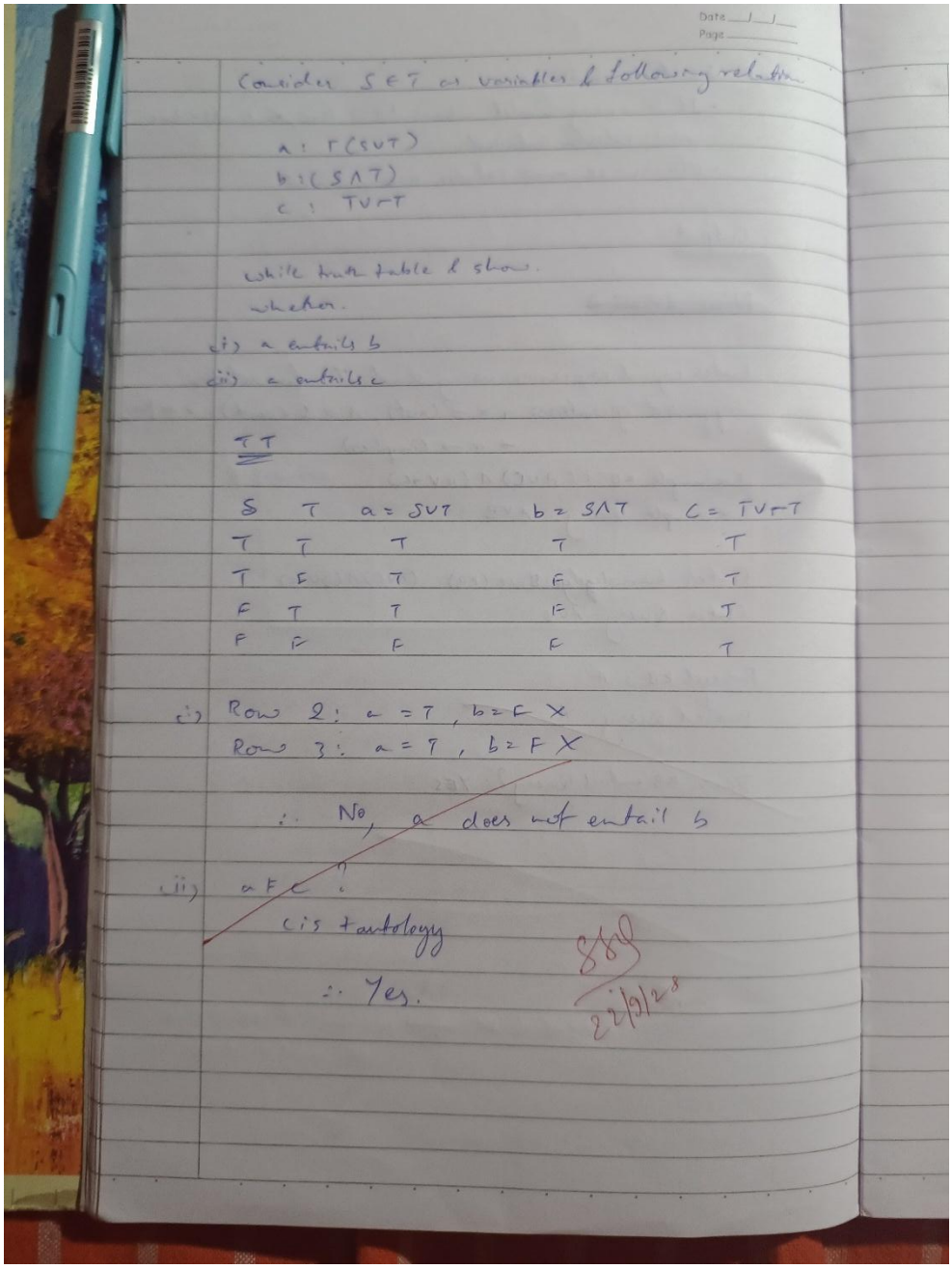
Enter logical expressions using standard infix notation.
Supported operators: ¬ or ~ (not), ∧ or & (and), ∨ or | (or),
→ or => (implies)

Example KB : (A∨C) ∧ (B∨¬C)
Example Query: A∨B

Enter knowledge Base (KB): (A∨C) ∧ (B∨¬C)
Enter Query: A∨B

Parsed KB : A
Parsed Query : A

Does KB entail Query? : YES



Consider S & T as variables & following relation

a : T(SVT)
b : (S∧T)
c : TV¬T

Write truth table & show whether

(i) a entails b
(ii) a entails c

TT

S	T	a = SVT	b = S∧T	C = TV¬T
T	T	T	T	T
T	F	T	F	T
F	T	T	F	T
F	F	F	F	T

(i) Row 2 : a = T, b = F ✗
Row 3 : a = T, b = F ✗

∴ No, a does not entail b

(ii) a ⊨ c :
c is tautology

∴ Yes.

8/8/23
22/9/23

Program 7

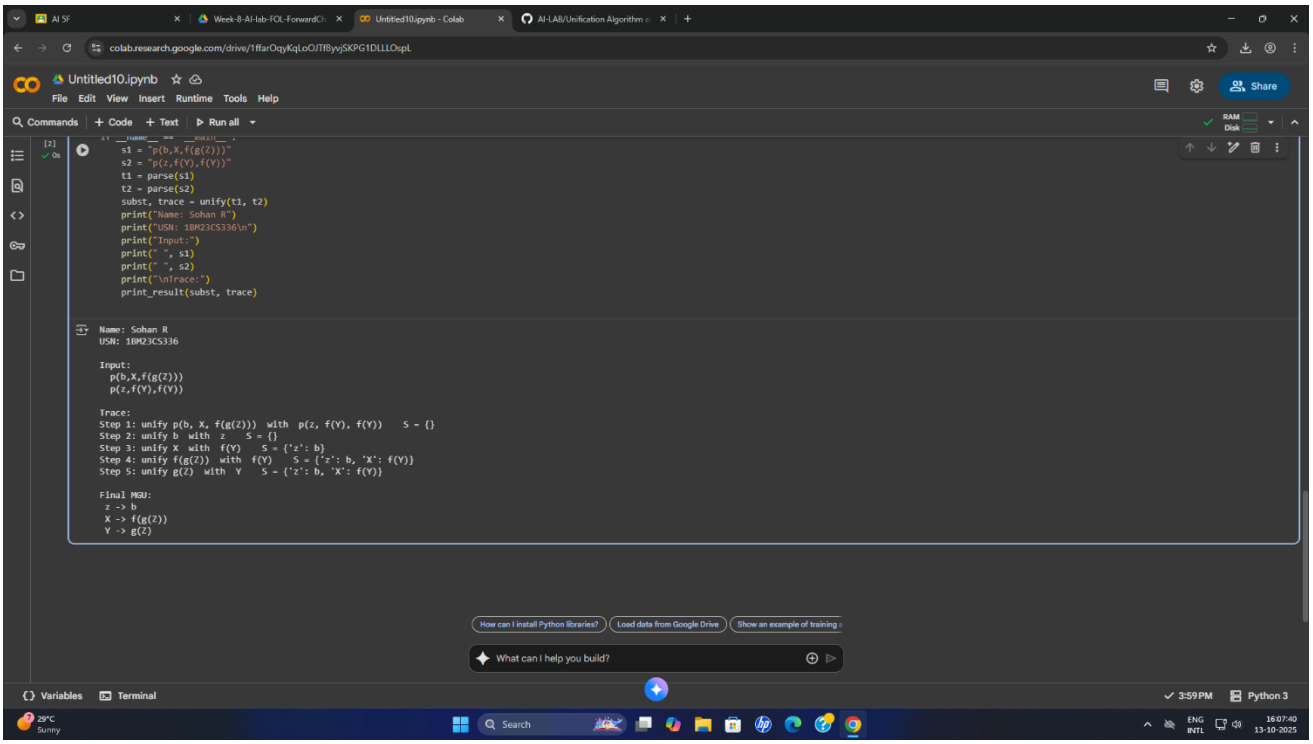
Implement unification in first order logic

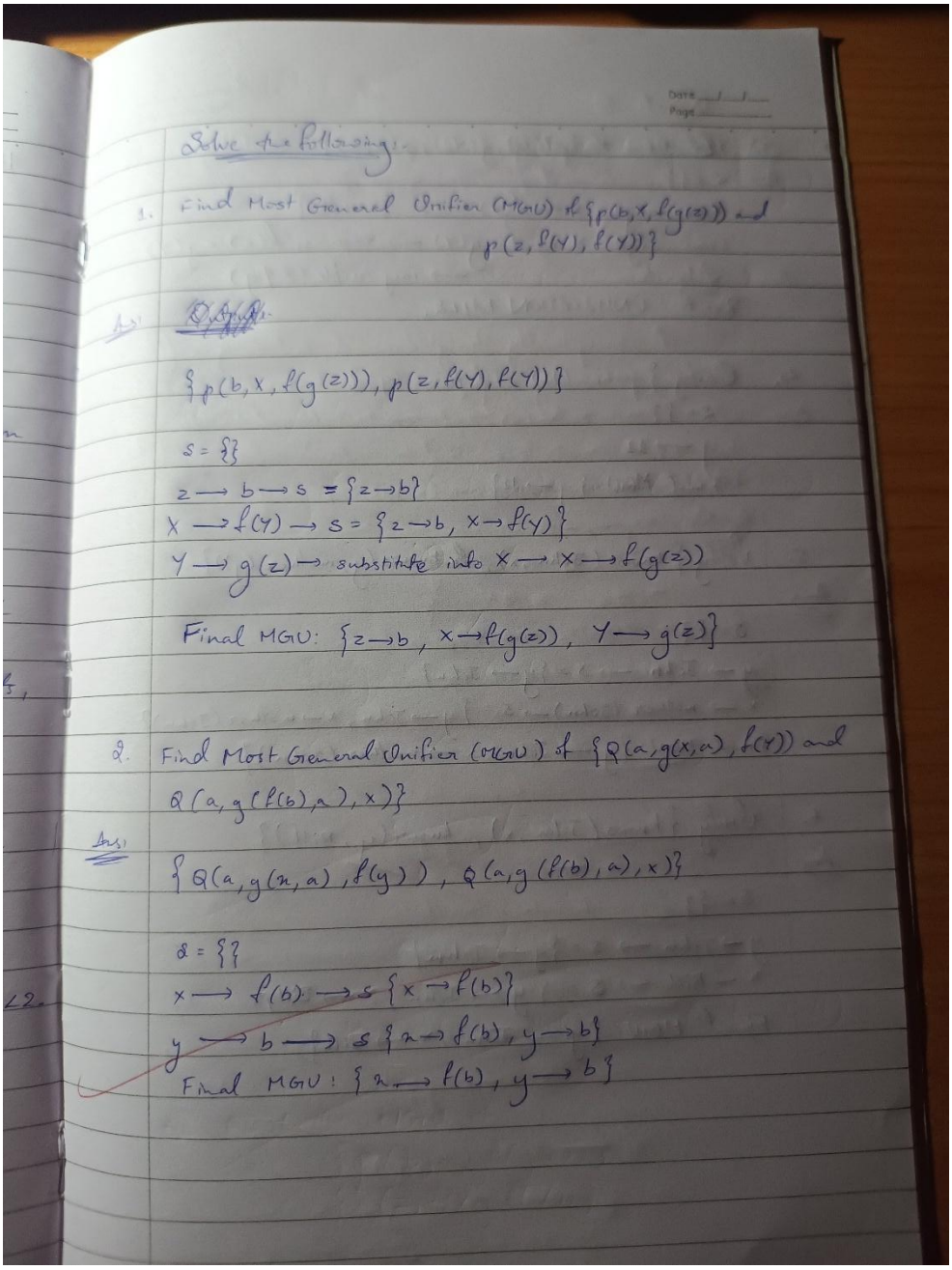
Code:

```
import re
from collections import deque
VARIABLES = {"X", "Y", "Z", "z"}

TOKEN_REGEX = r"\s*([A-Za-z0-9_]+|[,()])\s*"

def tokenize(s):
    return [t for t in re.findall(TOKEN_REGEX, s) if t.strip()]

class Term:
    def __init__(self, name, args=None):
        self.name = name
        self.args = args or []
    def is_variable(self):
        return (not self.args) and (self.name in VARIABLES)
    def is_constant(self):
        return (not self.args) and (self.name not in VARIABLES)
    def __repr__(self):
        if self.args:
            return f"{self.name}({', '.join(map(repr, self.args))})"
        return self.name

def parse_term(tokens):
    name = tokens.popleft()
    if tokens and tokens[0] == '(':
        tokens.popleft()
        args = []
        if tokens[0] != ')':
            while True:
                args.append(parse_term(tokens))
                if tokens[0] == ',':
                    tokens.popleft()
                    continue
                elif tokens[0] == ')':
                    break
        tokens.popleft()
        return Term(name, args)
    return Term(name, [])

def parse(s):
    tks = deque(tokenize(s))
    term = parse_term(tks)
```

```
    if tks:
        raise ValueError("Extra tokens after parse: " + " ".join(tks))
    return term

def apply_subst(t, subst):
    if t.is_variable():
        if t.name in subst:
            return apply_subst(subst[t.name], subst)
        return t
    if t.args:
        return Term(t.name, [apply_subst(a, subst) for a in t.args])
    return t

def occurs_check(var, term, subst):
    t = apply_subst(term, subst)
    if t.is_variable():
        return t.name == var
    if t.args:
        return any(occurs_check(var, a, subst) for a in t.args)
    return False

def unify(a, b):
    subst = {}
    eqs = deque([(a, b)])
    trace = []
    step = 0
    while eqs:
        step += 1
        s, t = eqs.popleft()
        s = apply_subst(s, subst)
        t = apply_subst(t, subst)
        trace.append((step, s, t, dict(subst)))
        if repr(s) == repr(t):
            continue
        if s.is_variable():
            if occurs_check(s.name, t, subst):
                return None, trace
            subst[s.name] = t
            continue
        if t.is_variable():
            if occurs_check(t.name, s, subst):
                return None, trace
            subst[t.name] = s
            continue

        if s.args and t.args:
            if s.name != t.name or len(s.args) != len(t.args):
```

```
            return None, trace
        for sa, ta in reversed(list(zip(s.args, t.args))):
            eqs.appendleft((sa, ta))
        continue

    return None, trace
  return subst, trace

def print_result(subst, trace):
  for item in trace:
    step, s, t, st = item
    print(f"Step {step}: unify {s}  with  {t}    S = {st}")
  if subst is None:
    print("\nResult: UNIFICATION FAILED")
  else:
    final = {k: apply_subst(v, subst) for k, v in subst.items()}
    print("\nFinal MGU:")
    for k, v in final.items():
      print(f"  {k} -> {v}")

if __name__ == "__main__":
  s1 = "p(b,X,f(g(Z)))"
  s2 = "p(z,f(Y),f(Y))"
  t1 = parse(s1)
  t2 = parse(s2)
  subst, trace = unify(t1, t2)
  print("Name: Sohan R")
  print("USN: 1BM23CS336\n")
  print("Input:")
  print(" ", s1)
  print(" ", s2)
  print("\nTrace:")
  print_result(subst, trace)
```


Untitled10.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
s1 = "p(b,X,f(g(Z)))"
s2 = "p(z,f(Y),f(Y))"
t1 = parse(s1)
t2 = parse(s2)
subst, trace = unify(t1, t2)
print("Name: Sohan R")
print("USN: 1BM23CS336\n")
print("Input:")
print(" ", s1)
print(" ", s2)
print("\nTrace:")
print_result(subst, trace)
Name: Sohan R
USN: 1BM23CS336
Input:
p(b,X,f(g(Z)))
p(z,f(Y),f(Y))
Trace:
Step 1: unify p(b, X, f(g(Z))) with p(z, f(Y), f(Y)) S = {}
Step 2: unify b with z S = {}
Step 3: unify X with f(Y) S = {'z': b}
Step 4: unify f(g(Z)) with f(Y) S = {'z': b, 'X': f(Y)}
Step 5: unify g(Z) with Y S = {'z': b, 'X': f(Y)}
Final MGU:
z -> b
X -> f(g(Z))
Y -> g(Z)
How can I install Python libraries?
Load data from Google Drive
What can I help you build?
Variables
Terminal
3:59PM
Python 3
Search



Solve the following:

1. Find Most General Unifier (MGU) of $\{p(b, x, f(g(z))) \text{ and } p(z, f(y), f(y))\}$

Ans:

$\{p(b, x, f(g(z))), p(z, f(y), f(y))\}$

$s = \{\}$

$z \longrightarrow b \longrightarrow s = \{z \to b\}$

$x \longrightarrow f(y) \longrightarrow s = \{z \to b, x \to f(y)\}$

$y \longrightarrow g(z) \longrightarrow$ substitute into $x \longrightarrow x \longrightarrow f(g(z))$

Final MGU: $\{z \to b, x \to f(g(z)), y \longrightarrow g(z)\}$

2. Find Most General Unifier (MGU) of $\{Q(a, g(x, a), f(y)) \text{ and } Q(a, g(f(b), a), x)\}$

Ans:

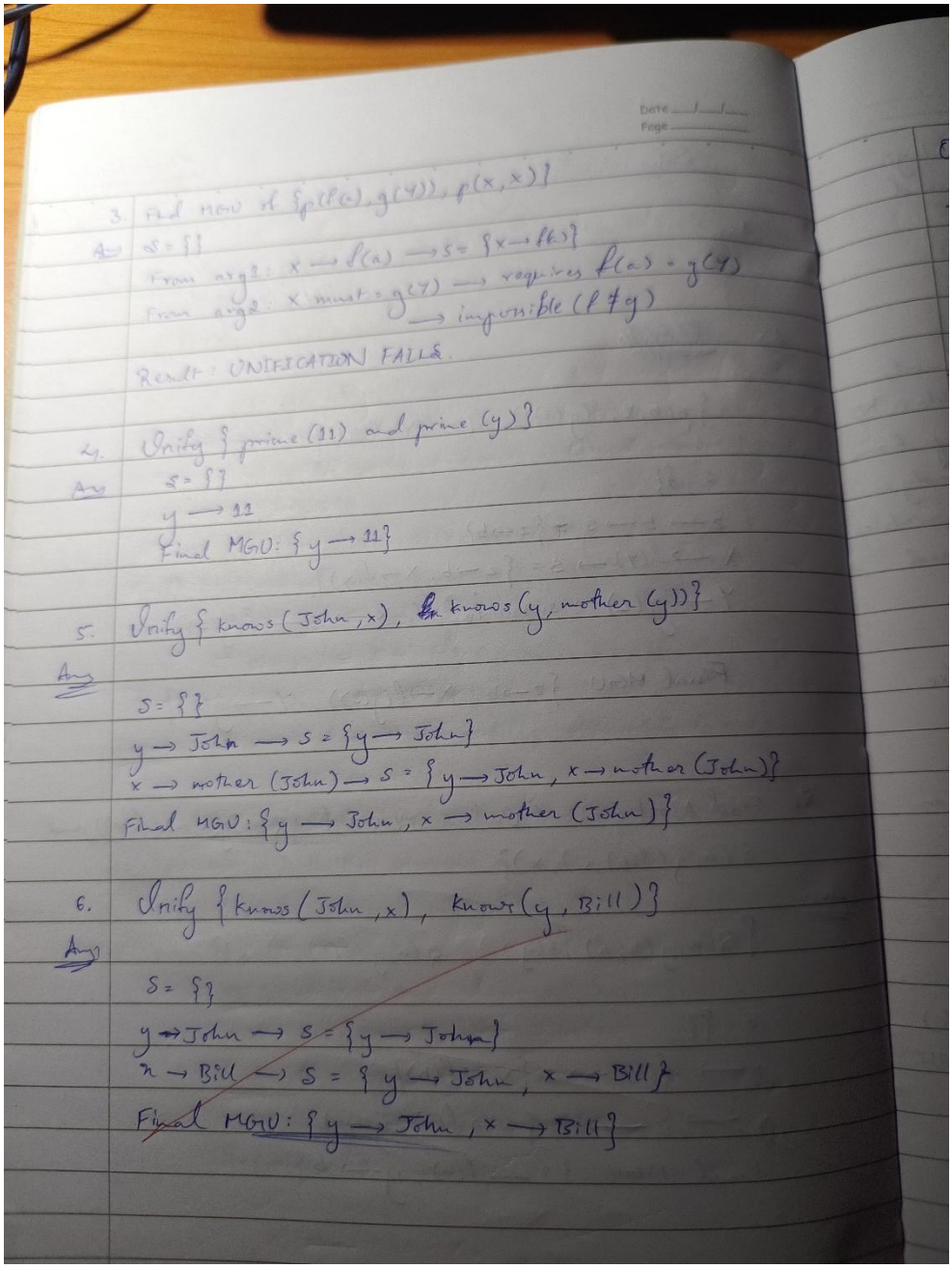
$\{Q(a, g(x, a), f(y)), Q(a, g(f(b), a), x)\}$

$s = \{\}$

$x \longrightarrow f(b) \longrightarrow s \{x \to f(b)\}$

$y \longrightarrow b \longrightarrow s \{x \to f(b), y \to b\}$

Final MGU: $\{x \longrightarrow f(b), y \longrightarrow b\}$



3. Find MGU of $\{p(f(a), g(y)), p(x, x)\}$

Ans $S = \{\}$

From arg 1: $x \rightarrow f(a) \rightarrow S = \{x \rightarrow f(a)\}$

From arg 2: x must = $g(y)$ → requires $f(a) = g(y)$

→ impossible ($f \neq g$)

Result: UNIFICATION FAILS.

4. Unify { prime (11) and prime (y) }

Ans $S = \{\}$

$y \rightarrow 11$

Final MGU: $\{y \rightarrow 11\}$

5. Unify { knows (John, x), knows (y, mother (y)) }

Ans $S = \{\}$

$y \rightarrow John \rightarrow S = \{y \rightarrow John\}$

$x \rightarrow mother(John) \rightarrow S = \{y \rightarrow John, x \rightarrow mother(John)\}$

Final MGU: $\{y \rightarrow John, x \rightarrow mother(John)\}$

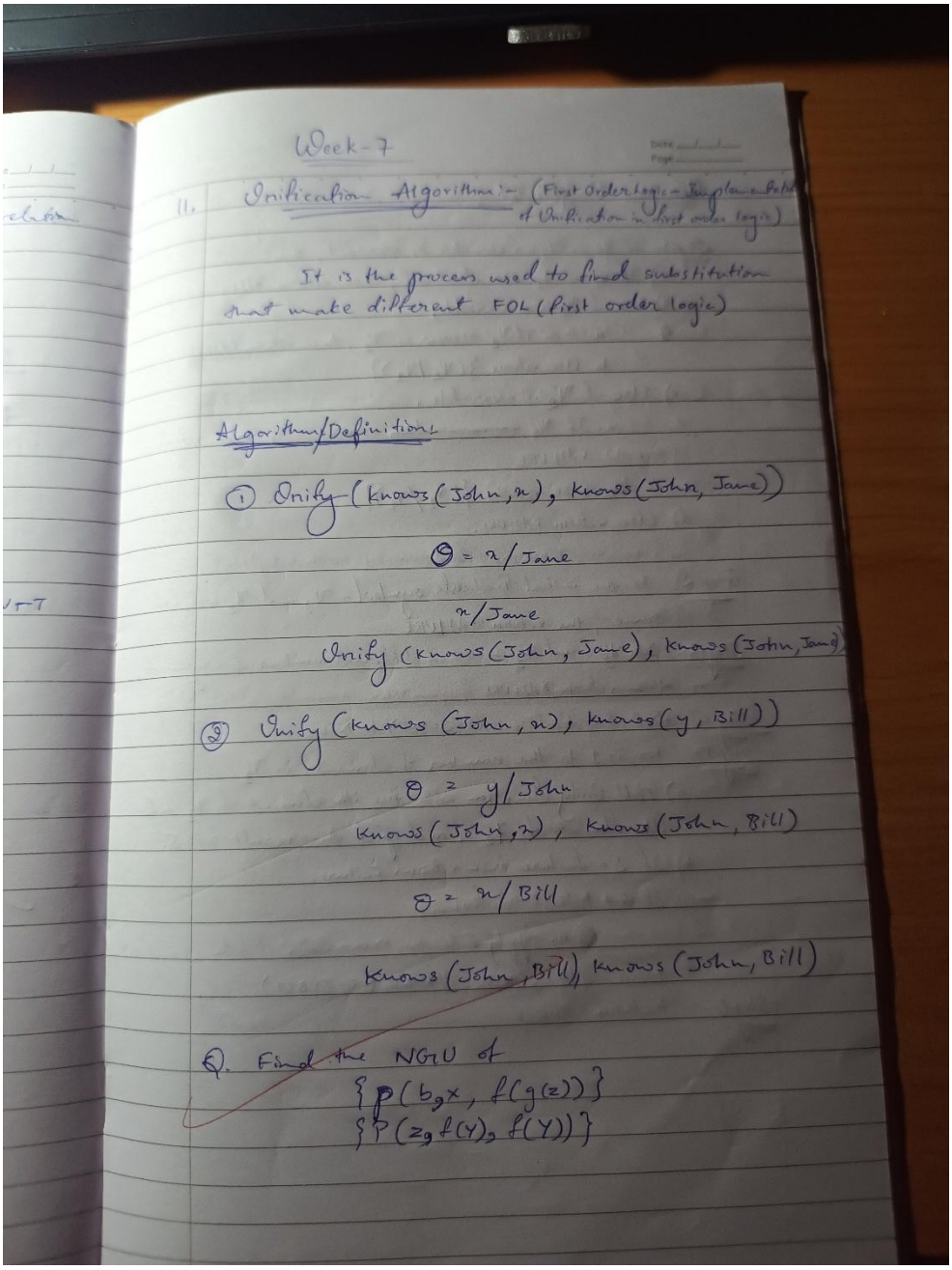
6. Unify { knows (John, x), knows (y, Bill) }

Ans $S = \{\}$

$y \rightarrow John \rightarrow S = \{y \rightarrow John\}$

$x \rightarrow Bill \rightarrow S = \{y \rightarrow John, x \rightarrow Bill\}$

Final MGU: $\{y \rightarrow John, x \rightarrow Bill\}$



Week-7

11. Unification Algorithm :- (First Order Logic - Important Points of Unification in first order logic)

It is the process used to find substitution that make different FOL (First order logic)

Algorithm/Definition:

① Unify (knows (John, x), knows (John, Jane))

$\theta = x/Jane$

$x/Jane$

Unify (knows (John, Jane), knows (John, Jane))

② Unify (knows (John, x), knows (y, Bill))

$\theta = y/John$

knows (John, x), knows (John, Bill)

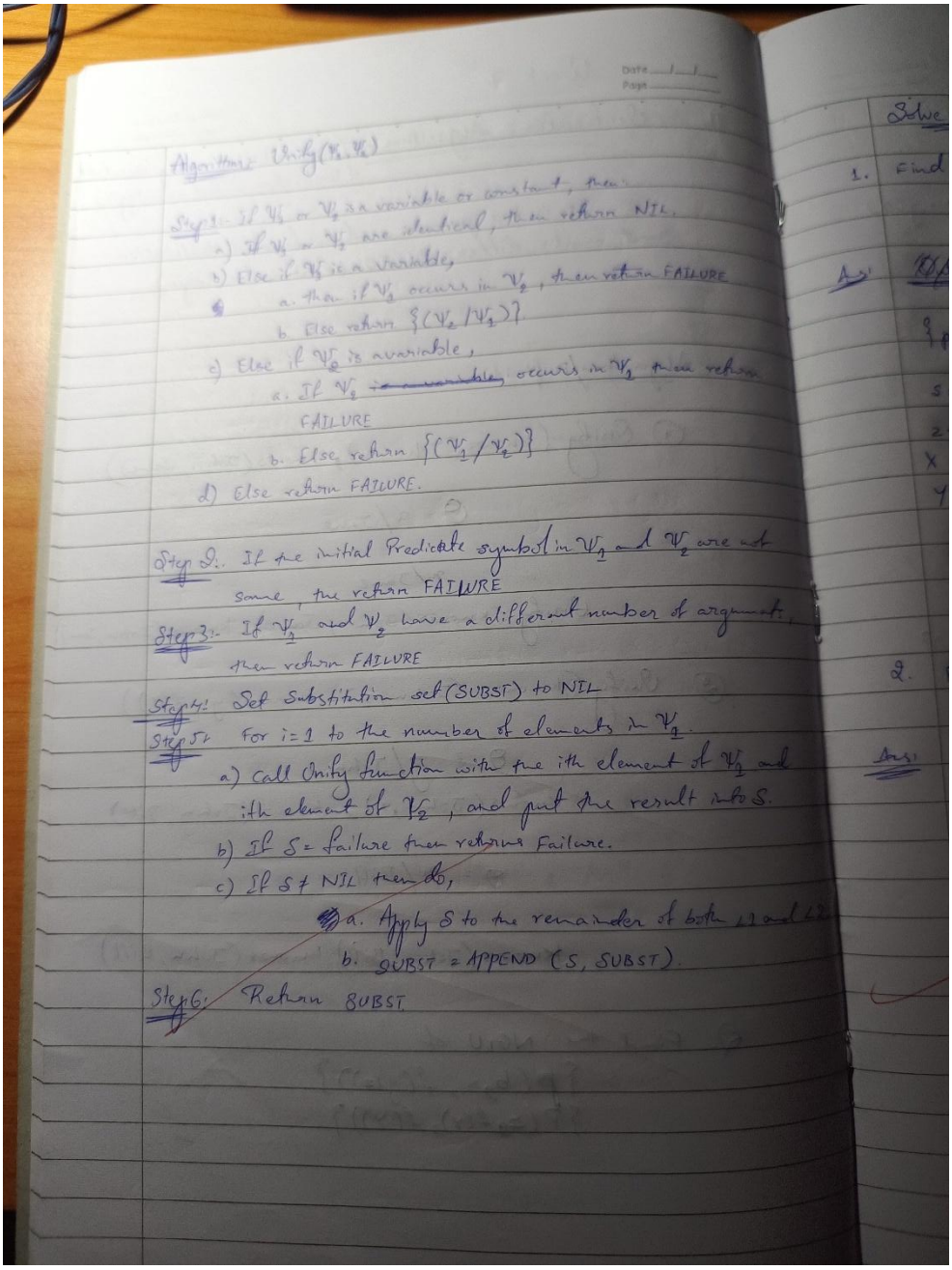
$\theta = x/Bill$

knows (John, Bill), knows (John, Bill)

Q. Find the MGU of

$\{p(b, x, f(g(z))\}$

$\{p(z, f(y), f(y))\}$



Algorithm: Unify(Ψ_1, Ψ_2)

Step 1:- If Ψ_1 or Ψ_2 is a variable or constant, then:
a) If Ψ_1 = Ψ_2 are identical, then return NIL.
b) Else if Ψ_1 is a variable,
a. then if Ψ_1 occurs in Ψ_2, then return FAILURE
b. Else return $\{(\Psi_2 / \Psi_1)\}$
c) Else if Ψ_2 is a variable,
a. If Ψ_2 occurs in Ψ_1 then return FAILURE
b. Else return $\{(\Psi_1 / \Psi_2)\}$
d) Else return FAILURE.

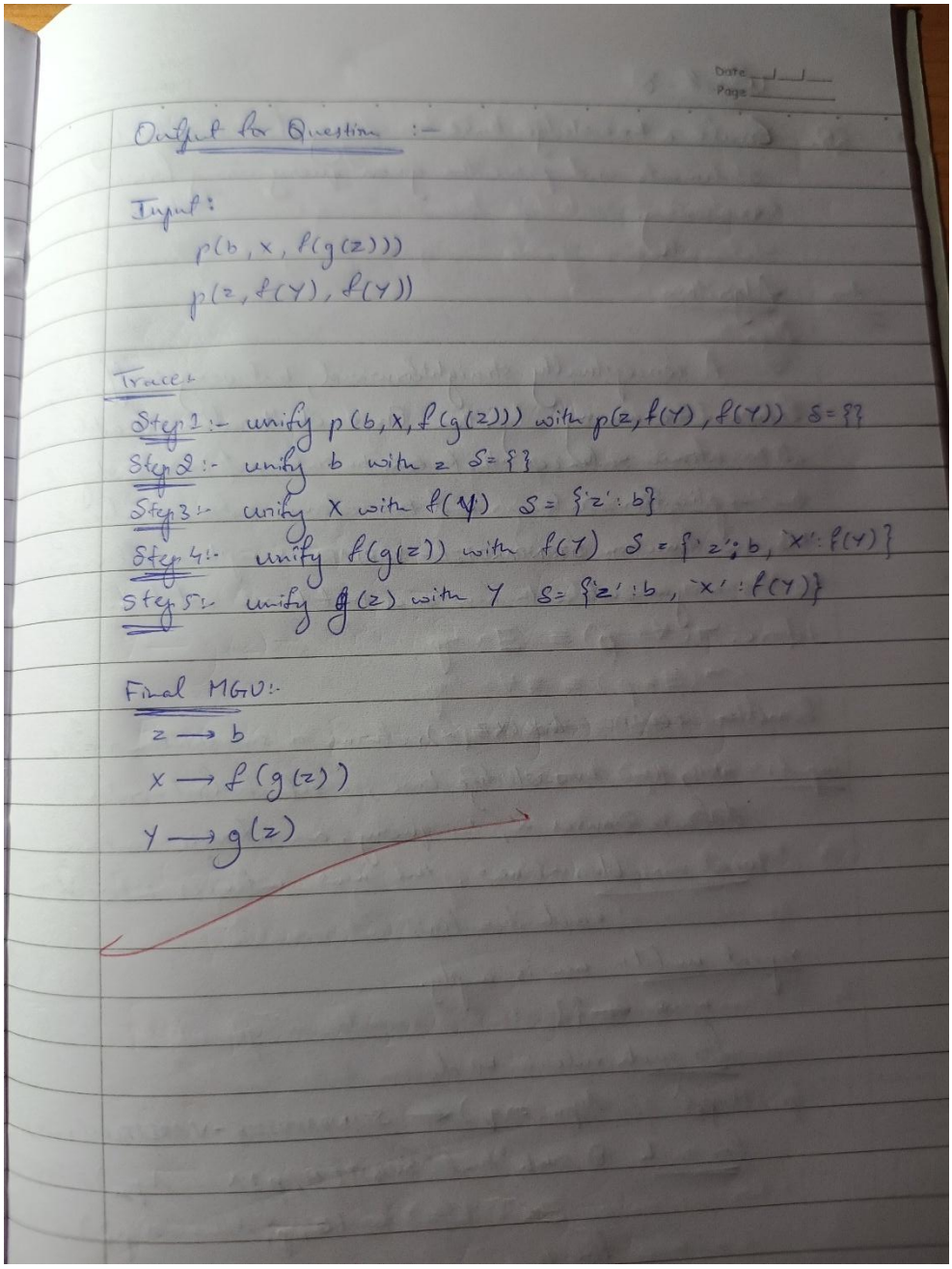
Step 2:- If the initial Predicate symbol in Ψ_1 and Ψ_2 are not same, the return FAILURE

Step 3:- If Ψ_1 and Ψ_2 have a different number of arguments, then return FAILURE

Step 4:- Set Substitution set (SUBST) to NIL

Step 5:- For i=1 to the number of elements in Ψ_1.
a) Call Unify function with the ith element of Ψ_1 and ith element of Ψ_2, and put the result into S.
b) If S= failure then returns Failure.
c) If S ≠ NIL then do,
a. Apply S to the remainder of both $L1$ and $L2$
b. SUBST = APPEND (S, SUBST).

Step 6:- Return SUBST



Output for Question :-

Input:

$p(b, x, f(g(z)))$

$p(z, f(y), f(y))$

Trace:

Step 1 :- unify $p(b, x, f(g(z)))$ with $p(z, f(y), f(y))$ $S = \{\}$

Step 2 :- unify b with z $S = \{\}$

Step 3 :- unify x with $f(y)$ $S = \{z : b\}$

Step 4 :- unify $f(g(z))$ with $f(y)$ $S = \{z : b, x : f(y)\}$

Step 5 :- unify $g(z)$ with y $S = \{z : b, x : f(y)\}$

Final MGU:-

$z \longrightarrow b$

$x \longrightarrow f(g(z))$

$y \longrightarrow g(z)$

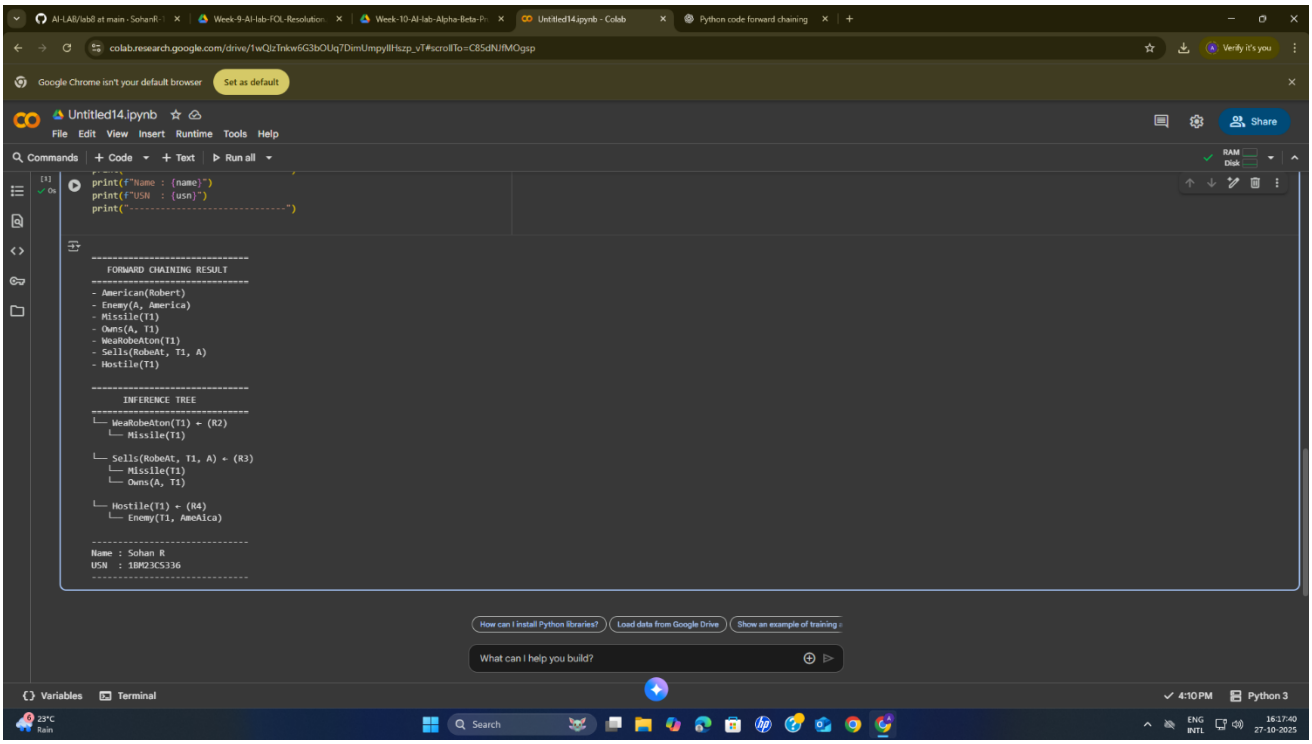
Program 8

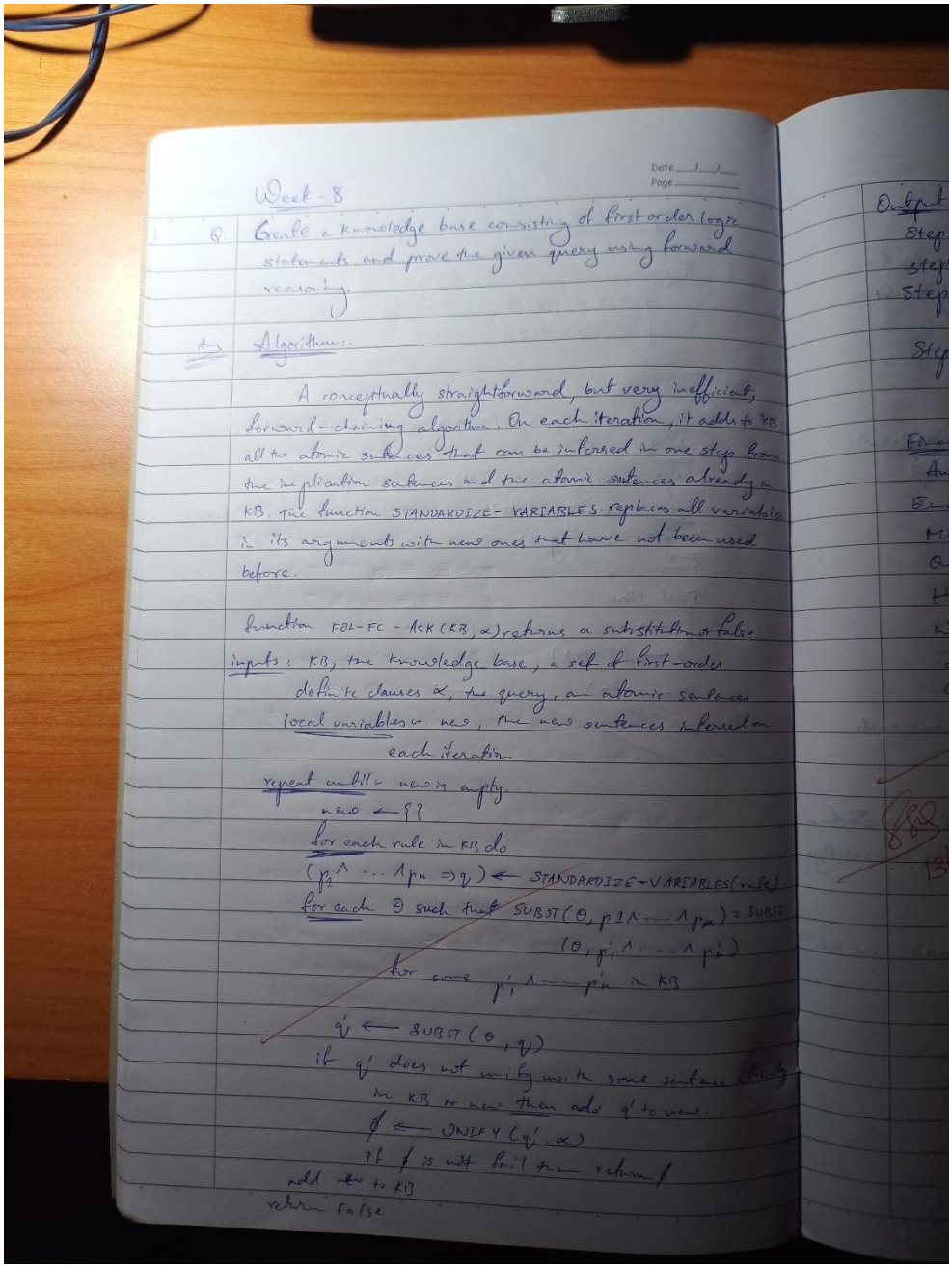
Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.

Code:

```
name = "Sohan R"
usn = "1BM23CS336"

KB = [
    "American(Robert)",
    "Enemy(A, America)",
    "Missile(T1)",
    "Owns(A, T1)"
]

rules = [
    {"name": "R1", "if": ["American(p)", "Weapon(q)", "Sells(p, q, r)", "Hostile(r)"], "then":
"Criminal(p)"},
    {"name": "R2", "if": ["Missile(x)"], "then": "Weapon(x)"},
    {"name": "R3", "if": ["Missile(x)", "Owns(A, x)"], "then": "Sells(Robert, x, A)"},
    {"name": "R4", "if": ["Enemy(x, America)"], "then": "Hostile(x)"}
]

def match(pattern, fact):
    return pattern.split("(")[0] == fact.split("(")[0]

def condition_satisfied(cond, KB):
    return any(match(cond, fact) for fact in KB)

def substitute(statement, mapping=None):
    if not mapping:
        mapping = {"p": "Robert", "r": "A", "q": "T1", "x": "T1"}
    for var, val in mapping.items():
        statement = statement.replace(var, val)
    return statement

def forward_chaining(KB, rules):
    inferred = set()
    derivations = {}
    added = True
    while added:
        added = False
        for rule in rules:
            if all(condition_satisfied(cond, KB) for cond in rule["if"]):
                conclusion = substitute(rule["then"])
                if conclusion not in KB:
```

```
                KB.append(conclusion)
                inferred.add(conclusion)
                derivations[conclusion] = {"rule": rule["name"], "premises": [substitute(p) for p in
rule["if"]]}
                added = True
    return KB, derivations

def print_tree(derivations, goal, level=0):
    if goal not in derivations:
        print("   " * level + f"└── {goal}")
        return
    rule_info = derivations[goal]
    print("   " * level + f"└── {goal} ← ({rule_info['rule']})")
    for premise in rule_info["premises"]:
        print_tree(derivations, premise, level + 1)

final_KB, derivations = forward_chaining(KB.copy(), rules)

print("\n==============================")
print("   FORWARD CHAINING RESULT")
print("==============================")
for fact in final_KB:
    print(f"- {fact}")

print("\n==============================")
print("      INFERENCE TREE")
print("==============================")
for conclusion in derivations:
    print_tree(derivations, conclusion)
    print()

print("------------------------------")
print(f"Name : {name}")
print(f"USN  : {usn}")
print("------------------------------")
```


Untitled14.ipynb
File Edit View Insert Runtime Tools Help
Share
print(f"Name : {name}")
print(f"USN : {usn}")
print("------------------------------")
FORWARD CHAINING RESULT
- American(Robert)
- Enemy(A, America)
- Missile(T1)
- Owns(A, T1)
- WeaRobeAton(T1)
- Sells(RobeAt, T1, A)
- Hostile(T1)
INFERENCE TREE
└─ WeaRobeAton(T1) ← (R2)
└─ Missile(T1)
└─ Sells(RobeAt, T1, A) ← (R3)
└─ Missile(T1)
└─ Owns(A, T1)
└─ Hostile(T1) ← (R4)
└─ Enemy(T1, AmeAlca)
Name : Sohan R
USN : 1BM23CS336
How can I install Python libraries?
Load data from Google Drive
What can I help you build?
Variables
Terminal
4:10 PM
Python 3



Week - 8

8. Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.

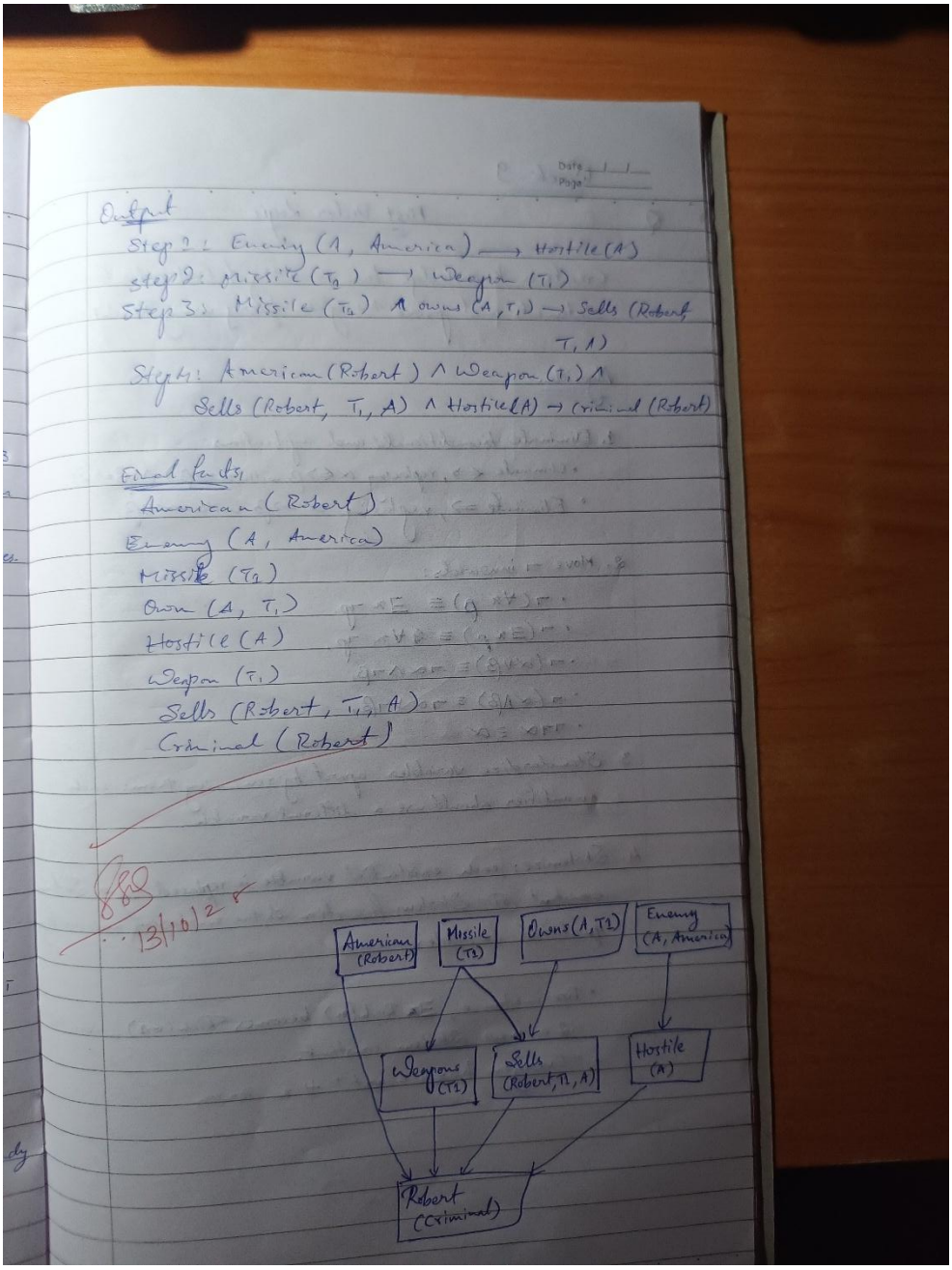
Algorithm:

A conceptually straightforward, but very inefficient, forward-chaining algorithm. On each iteration, it adds to KB all the atomic sentences that can be inferred in one step from the implication sentences and the atomic sentences already in KB. The function STANDARDIZE-VARIABLES replaces all variables in its arguments with new ones that have not been used before.

function FOL-FC-ASK(KB, α) returns a substitution or false

inputs: KB, the knowledge base, a set of first-order definite clauses
α, the query, an atomic sentence
local variables: new, the new sentences inferred on each iteration

repeat until new is empty
  new ← {}
  for each rule in KB do
    (p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-VARIABLES(rule)
    for each θ such that SUBST(θ, p1 ∧ ... ∧ pn) = SUBST(θ, p'1 ∧ ... ∧ p'n)
      for some p'1, ..., p'n in KB
      q' ← SUBST(θ, q)
      if q' does not unify with some sentence already in KB or new then add q' to new
      φ ← UNIFY(q', α)
      if φ is not fail then return φ
  add new to KB
return False


Output
Step 1 : Enemy (A, America) → Hostile (A)
Step 2 : Missile (T2) → Weapon (T1)
Step 3 : Missile (T2) ∧ Owns (A, T1) → Sells (Robert, T1, A)
Step 4 : American (Robert) ∧ Weapon (T1) ∧ Sells (Robert, T1, A) ∧ Hostile (A) → Criminal (Robert)
Final facts
American (Robert)
Enemy (A, America)
Missile (T2)
Own (A, T1)
Hostile (A)
Weapon (T1)
Sells (Robert, T1, A)
Criminal (Robert)
American (Robert)
Missile (T2)
Owns (A, T2)
Enemy (A, America)
Weapons (T2)
Sells (Robert, T1, A)
Hostile (A)
Robert (Criminal)

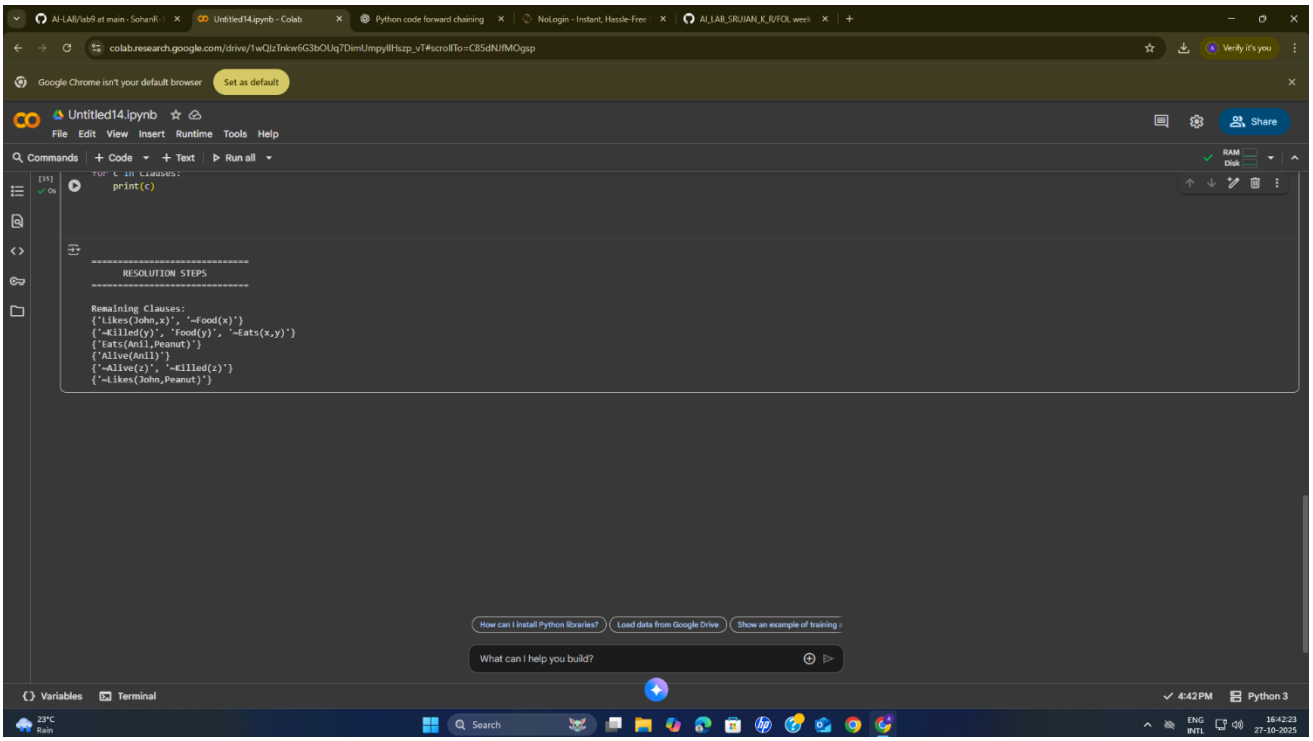
Program 9

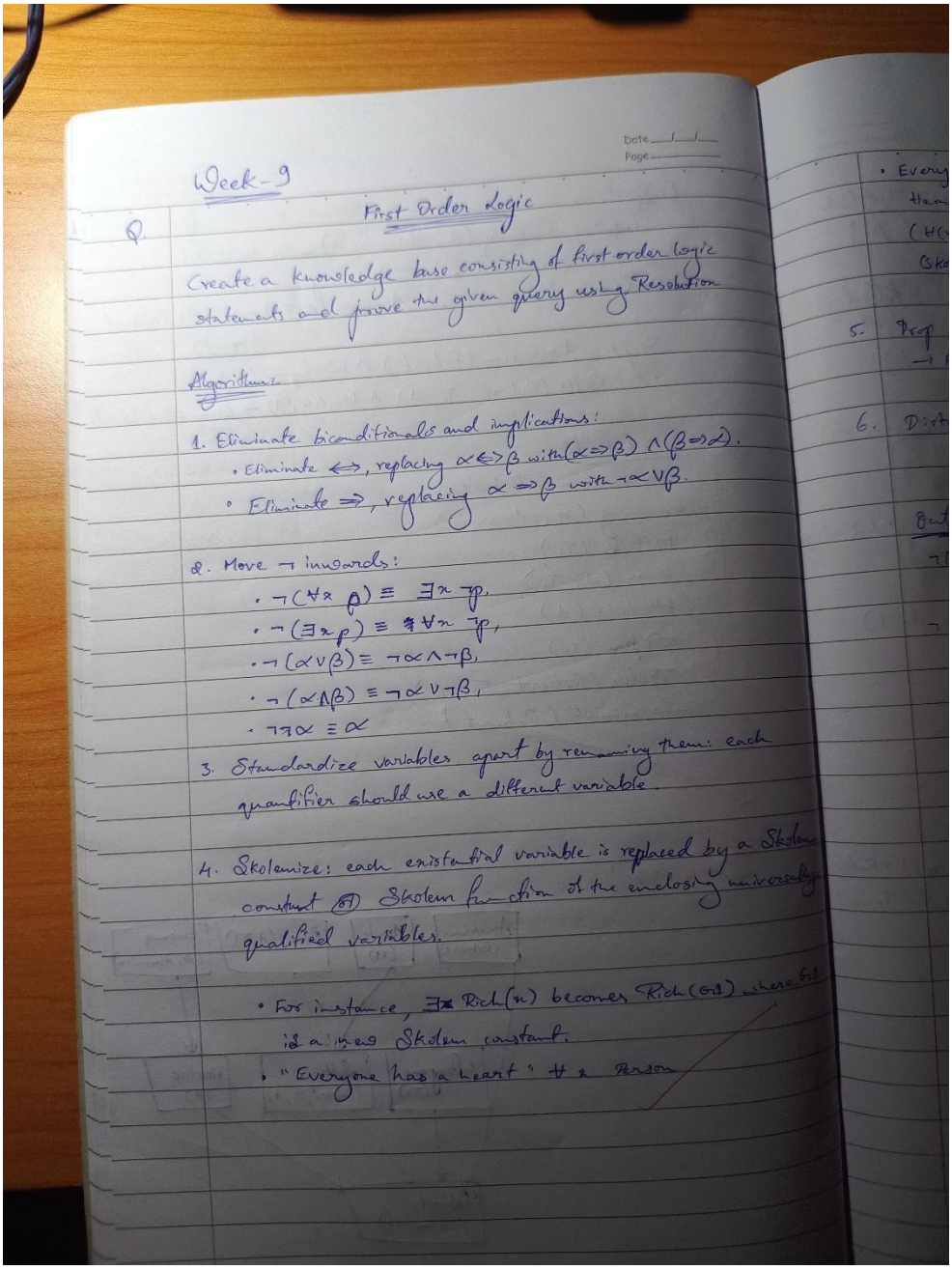
Create a knowledge base consisting of first order logic statements and prove the given query using Resolution.

Code:

```
def negate_literal(lit):
    return lit[1:] if lit.startswith("~") else "~" + lit

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

    for di in ci:
        ndi = negate_literal(di)
        if ndi in cj:
            new_clause = (ci - {di}) | (cj - {ndi})
            if len(new_clause) == 0:
                resolvents.append({"res": set(), "pair": (ci, cj)})
            else:
                resolvents.append({"res": new_clause, "pair": (ci, cj)})
    return resolvents


clauses = [
    {"~Food(x)", "Likes(John,x)"},
    {"~Eats(x,y)", "~Killed(y)", "Food(y)"},
    {"Eats(Anil,Peanut)"},
    {"Alive(Anil)"},
    {"~Alive(z)", "~Killed(z)"},
    {"~Likes(John,Peanut)"}
]

steps = []

changed = True
while changed:
    changed = False
    new = []

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

```
            for r in resolvents:
                res = r["res"]
                if res == set(): # NIL Found
                    steps.append((ci, cj, set()))
                    print("\n NIL (Contradiction Found)")
                    print("=> Query Proven TRUE")
                    changed = False
                    break

                if res not in clauses and res not in new:
                    new.append(res)
                    steps.append((ci, cj, res))
                    changed = True
        if changed is False and resolvents and res == set():
            break

    for c in new:
        clauses.append(c)

print("\n================================")
print("      RESOLUTION STEPS")
print("================================")
for (a, b, c) in steps:
    if c == set():
        print(f"{a}  +  {b}  =>  NIL")
    else:
        print(f"{a}  +  {b}  =>  {c}")

print("\nRemaining Clauses:")
for c in clauses:
    print(c)
```


Untitled14.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
print(c)
RESOLUTION STEPS
Remaining Clauses:
{'Likes(John,x)', '~Food(x)'}
{'~Killed(y)', 'Food(y)', '~Eats(x,y)'}
{'Eats(Anil,Peanut)'}
{'Alive(Anil)'}
{'~Alive(z)', '~Killed(z)'}
{'~Likes(John,Peanut)'}
How can I install Python libraries?
Load data from Google Drive
What can I help you build?
Variables
Terminal
4:42 PM
Python 3



Week-9

Q. First Order Logic

Create a knowledge base consisting of first order logic statements and prove the given query using Resolution

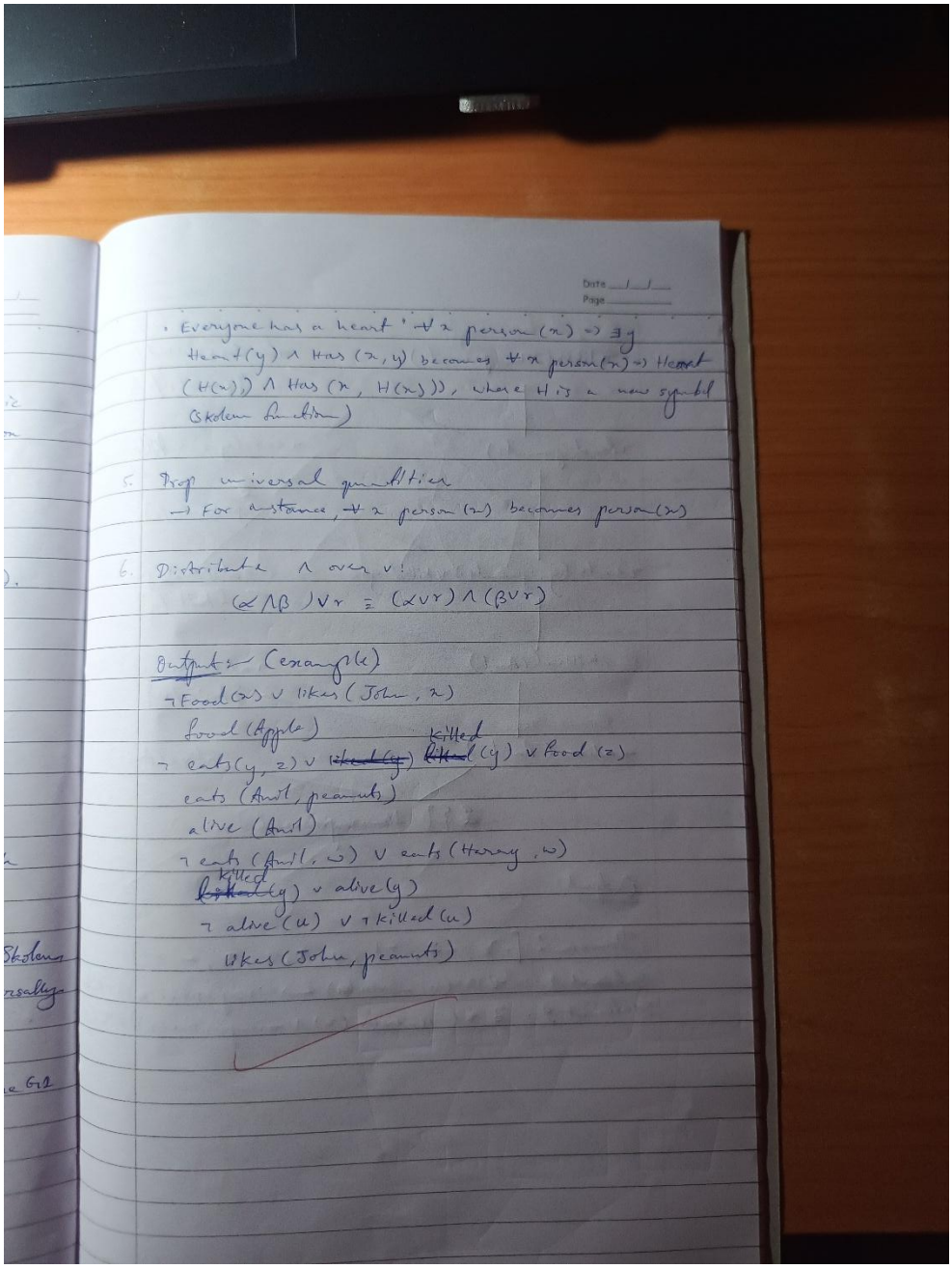
Algorithm:

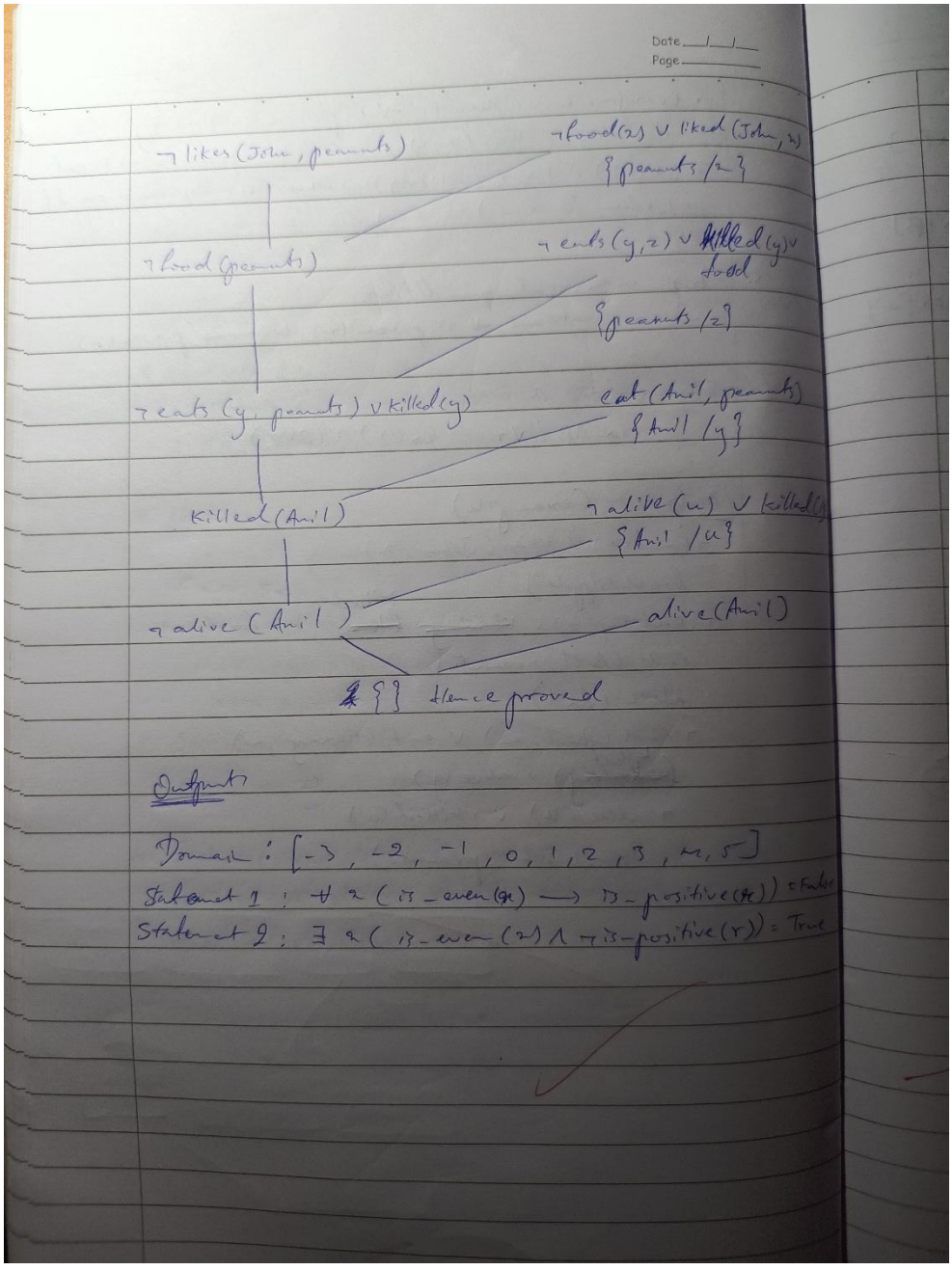
1. Eliminate biconditionals and implications:
   - Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
   - Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

2. Move $\neg$ inwards:
   - $\neg(\forall x\ p) \equiv \exists x \neg p$,
   - $\neg(\exists x\ p) \equiv \forall x \neg p$,
   - $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$,
   - $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$,
   - $\neg\neg\alpha \equiv \alpha$

3. Standardize variables apart by renaming them: each quantifier should use a different variable.

4. Skolemize: each existential variable is replaced by a Skolem constant or Skolem function of the enclosing universally qualified variables.
   - For instance, $\exists x$ Rich(x) becomes Rich(G1) where G1 is a new Skolem constant.
   - "Everyone has a heart" $\forall x$ Person


Date
Page
4. Everyone has a heart: ∀x person(x) ⇒ ∃y Heart(y) ∧ Has(x, y) becomes ∀x person(x) ⇒ Heart(H(x)) ∧ Has(x, H(x)), where H is a new symbol (Skolem function)
5. Drop universal quantifiers
→ For instance, ∀x person(x) becomes person(x)
6. Distribute ∧ over ∨:
(α ∧ β) ∨ γ ≡ (α ∨ γ) ∧ (β ∨ γ)
Output: (example)
¬Food(x) ∨ likes(John, x)
food(Apple)
¬eats(y, z) ∨ killed(y) ∨ Food(z)
eats(Ant, peanuts)
alive(Ant)
¬eats(Ant, w) ∨ eats(Harry, w)
killed(y) ∨ alive(y)
¬alive(u) ∨ ¬killed(u)
likes(John, peanuts)

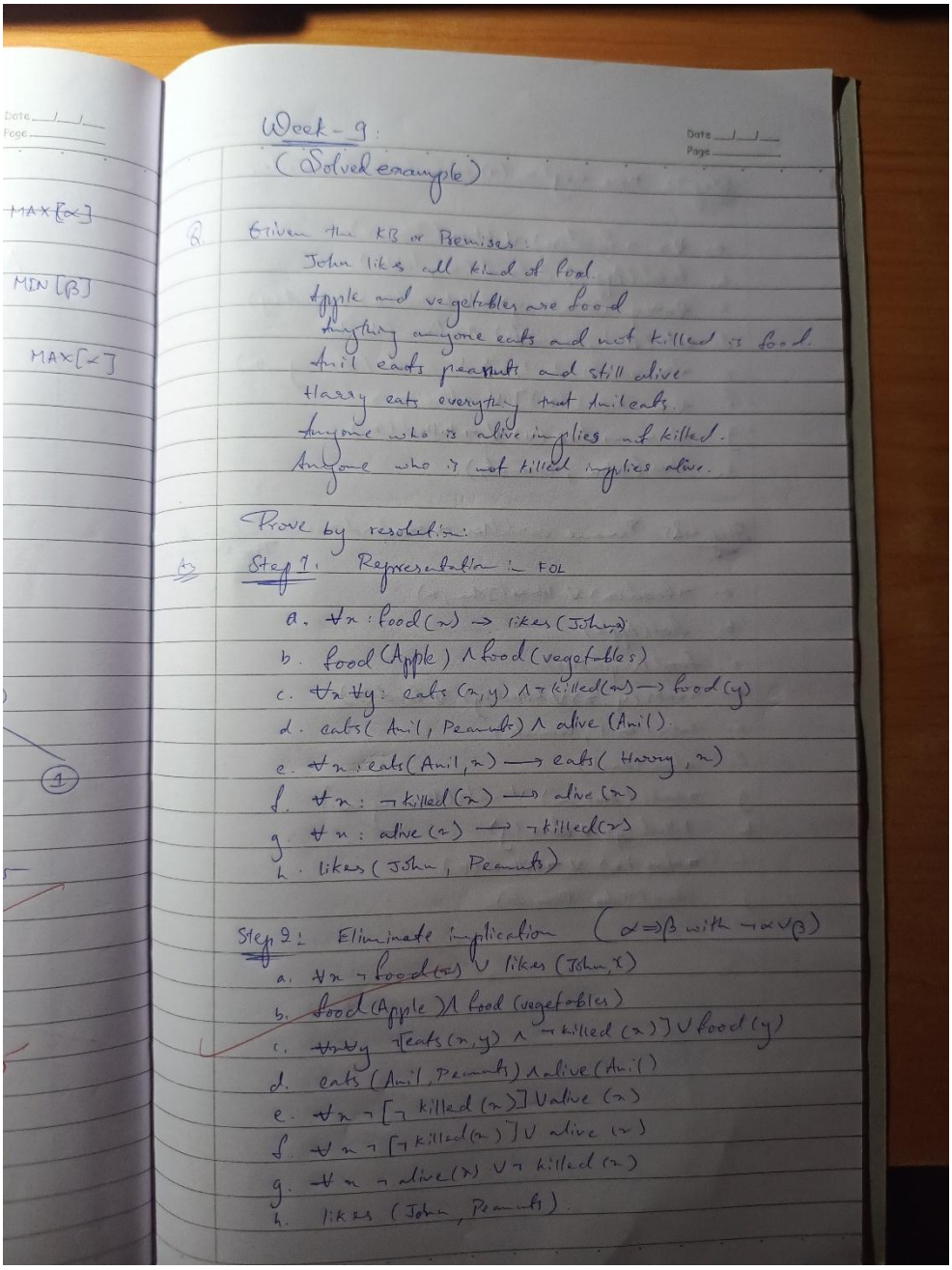


Output:

Domain : [-3, -2, -1, 0, 1, 2, 3, 4, 5]

Statement 1 : $\forall x$ (is_even(x) $\rightarrow$ is_positive(x)) = False

Statement 2 : $\exists x$ (is_even(x) $\wedge$ $\neg$is_positive(x)) = True



Week - 9:

(Solved example)

Q. Given the KB or Premises:

John likes all kind of food.

Apple and vegetables are food

Anything anyone eats and not killed is food.

Anil eats peanuts and still alive

Harry eats everything that Anil eats.

Anyone who is alive implies not killed.

Anyone who is not killed implies alive.

Prove by resolution:

Step 1: Representation in FOL

a. $\forall x : food(x) \rightarrow likes(John, x)$

b. $food(Apple) \wedge food(vegetables)$

c. $\forall x \forall y : eats(x, y) \wedge \neg killed(x) \rightarrow food(y)$

d. $eats(Anil, Peanuts) \wedge alive(Anil)$

e. $\forall x : eats(Anil, x) \rightarrow eats(Harry, x)$

f. $\forall x : \neg killed(x) \rightarrow alive(x)$

g. $\forall x : alive(x) \rightarrow \neg killed(x)$

h. $likes(John, Peanuts)$

Step 2: Eliminate implication ($\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$)

a. $\forall x \neg food(x) \vee likes(John, x)$

b. $food(Apple) \wedge food(vegetables)$

c. $\forall y \forall x [eats(x, y) \wedge \neg killed(x)] \vee food(y)$

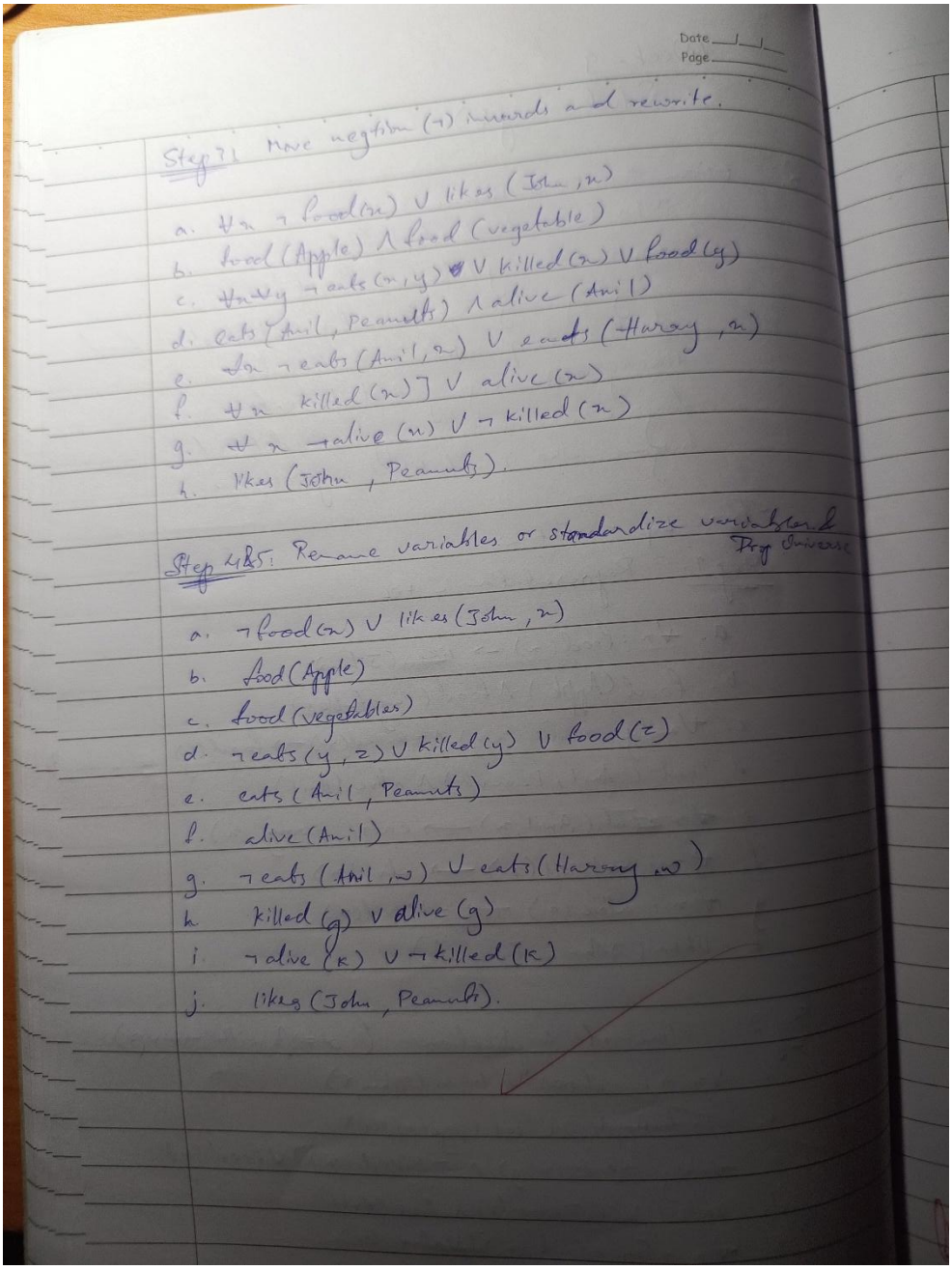
d. $eats(Anil, Peanuts) \wedge alive(Anil)$

e. $\forall x \neg [\neg killed(x)] \vee alive(x)$

f. $\forall x \neg [\neg killed(x)] \vee alive(x)$

g. $\forall x \neg alive(x) \vee \neg killed(x)$

h. $likes(John, Peanuts)$.



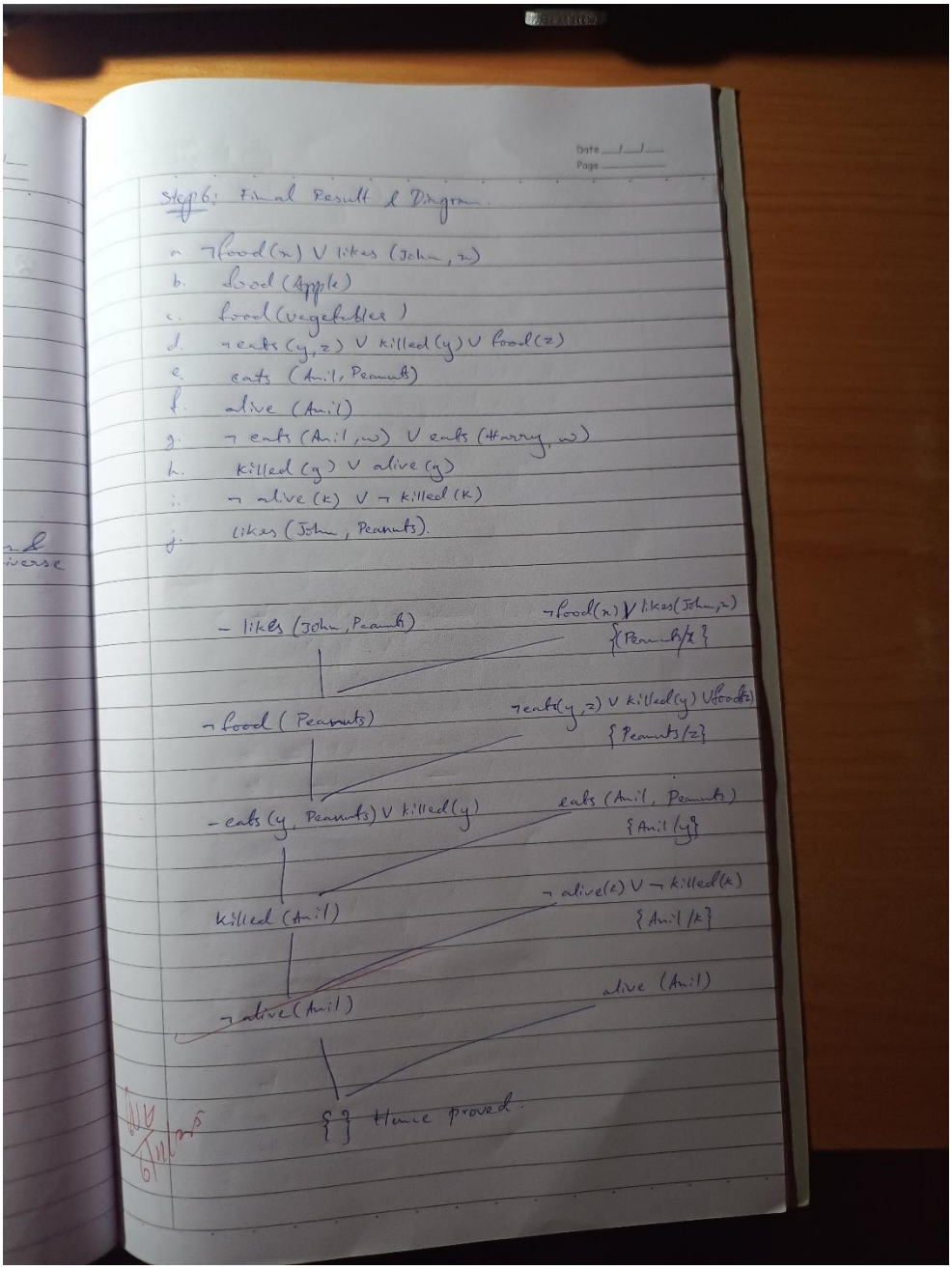
Date / Page

Step 1: Move negation (¬) inwards and rewrite.

a. $\forall x \neg food(x) \vee likes(John, x)$
b. $food(Apple) \wedge food(vegetable)$
c. $\forall x \forall y \neg eats(x, y) \vee killed(x) \vee food(y)$
d. $eats(Anil, Peanuts) \wedge alive(Anil)$
e. $\forall x \neg eats(Anil, x) \vee eats(Harry, x)$
f. $\forall x killed(x)] \vee alive(x)$
g. $\forall x \neg alive(x) \vee \neg killed(x)$
h. $likes(John, Peanuts)$.

Step 4&5: Rename variables or standardize variables and Drop Universal

a. $\neg food(x) \vee likes(John, x)$
b. $food(Apple)$
c. $food(vegetables)$
d. $\neg eats(y, z) \vee killed(y) \vee food(z)$
e. $eats(Anil, Peanuts)$
f. $alive(Anil)$
g. $\neg eats(Anil, w) \vee eats(Harry, w)$
h. $killed(g) \vee alive(g)$
i. $\neg alive(k) \vee \neg killed(k)$
j. $likes(John, Peanuts)$.


Step 6: Final Result & Diagram.
a. ¬food(x) V likes (John, x)
b. food (Apple)
c. food (vegetables)
d. ¬eats (y, z) V killed (y) V food(z)
e. eats (Anil, Peanuts)
f. alive (Anil)
g. ¬ eats (Anil, w) V eats (Harry, w)
h. killed (g) V alive (g)
i. ¬ alive (k) V ¬ killed (k)
j. likes (John, Peanuts).
- likes (John, Peanuts)
¬food(x) V likes(John, x)
{Peanuts/x}
¬food (Peanuts)
¬eats(y, z) V killed(y) V food(z)
{Peanuts/z}
- eats (y, Peanuts) V killed(y)
eats (Anil, Peanuts)
{Anil/y}
killed (Anil)
¬ alive(k) V ¬ killed(k)
{Anil/k}
¬alive(Anil)
alive (Anil)
{} Hence proved.

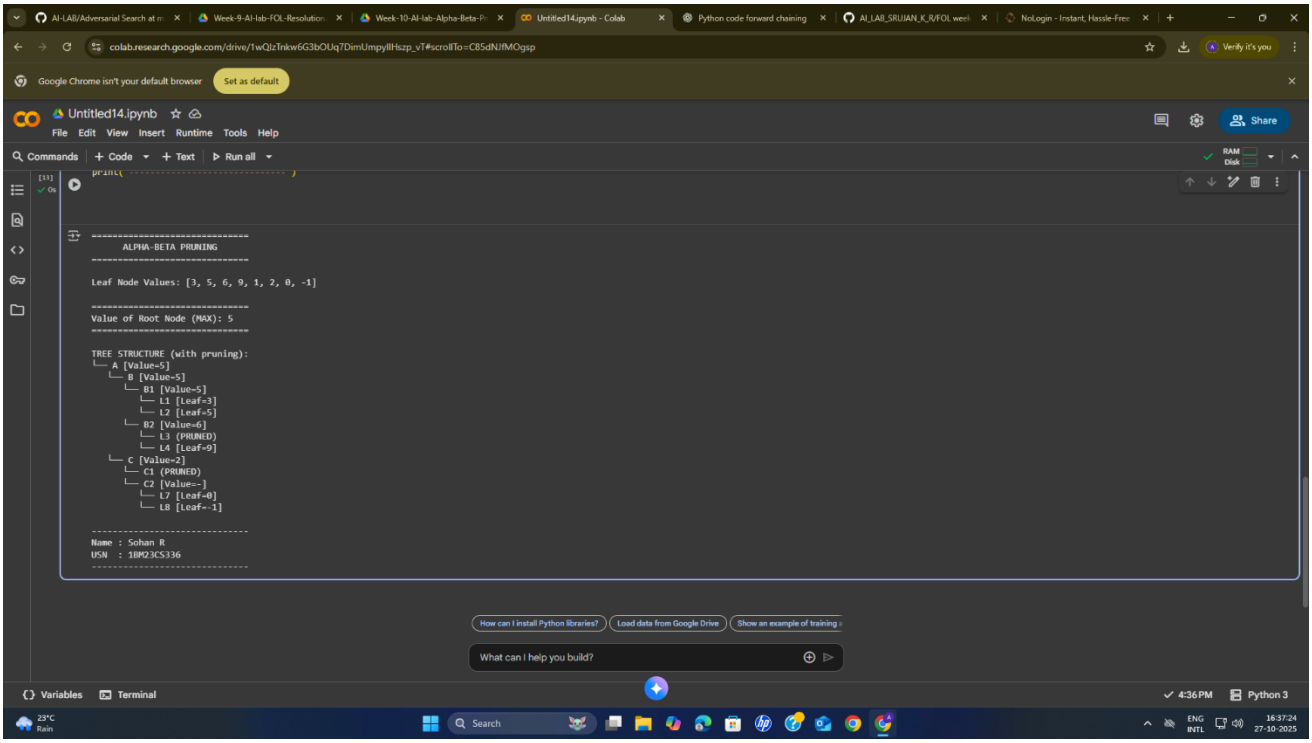
Program 10

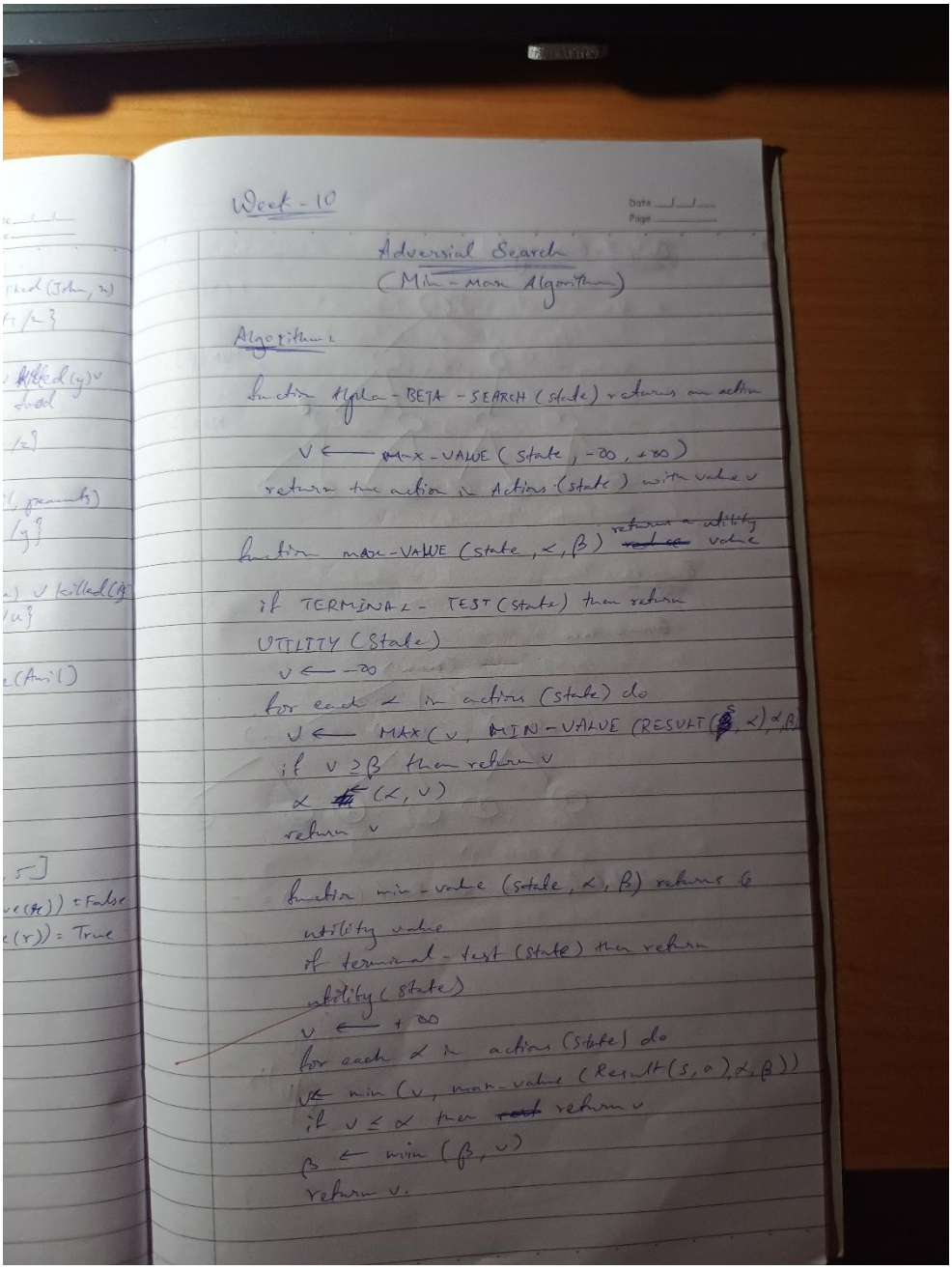
Adversial Search or Alpha- Beta pruning

Code:

```
name = "Sohan R"
usn = "1BM23CS336"

import math

class Node:
    def __init__(self, name, value=None):
        self.name = name
        self.value = value
        self.children = []
        self.pruned = False

def alpha_beta(node, depth, alpha, beta, maximizing_player):
    if not node.children:  # leaf node
        return node.value

    if maximizing_player:
        max_eval = -math.inf
        for child in node.children:
            eval = alpha_beta(child, depth + 1, alpha, beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                child.pruned = True
                break
        node.value = max_eval
        return max_eval
    else:
        min_eval = math.inf
        for child in node.children:
            eval = alpha_beta(child, depth + 1, alpha, beta, True)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                child.pruned = True
                break
        node.value = min_eval
        return min_eval

def print_tree(node, level=0):
    prefix = "    " * level + " └── "
    if node.pruned:
```

```
        print(f"{prefix}{node.name} (PRUNED)")
    else:
        if node.children:
            val = node.value if node.value is not None else "-"
            print(f"{prefix}{node.name} [Value={val}]")
            for child in node.children:
                print_tree(child, level + 1)
        else:
            print(f"{prefix}{node.name} [Leaf={node.value}]")

leaf_values = [3, 5, 6, 9, 1, 2, 0, -1]

root = Node("A")
b = Node("B")
c = Node("C")
root.children = [b, c]

b1 = Node("B1")
b2 = Node("B2")
c1 = Node("C1")
c2 = Node("C2")
b.children = [b1, b2]
c.children = [c1, c2]

b1.children = [Node("L1", leaf_values[0]), Node("L2", leaf_values[1])]
b2.children = [Node("L3", leaf_values[2]), Node("L4", leaf_values[3])]
c1.children = [Node("L5", leaf_values[4]), Node("L6", leaf_values[5])]
c2.children = [Node("L7", leaf_values[6]), Node("L8", leaf_values[7])]

print("================================")
print("      ALPHA-BETA PRUNING")
print("================================")
print(f"\nLeaf Node Values: {leaf_values}\n")

result = alpha_beta(root, 0, -math.inf, math.inf, True)

print("================================")
print(f"Value of Root Node (MAX): {result}")
print("================================")

print("\nTREE STRUCTURE (with pruning):")
print_tree(root)

print("\n------------------------------")
print(f"Name : {name}")
print(f"USN  : {usn}")
print("------------------------------")
```


Untitled14.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
ALPHA-BETA PRUNING
Leaf Node Values: [3, 5, 6, 9, 1, 2, 0, -1]
Value of Root Node (MAX): 5
TREE STRUCTURE (with pruning):
└── A [Value=5]
└── B [Value=5]
└── B1 [Value=5]
└── L1 [Leaf=3]
└── L2 [Leaf=5]
└── B2 [Value=6]
└── L3 (PRUNED)
└── L4 [Leaf=9]
└── C [Value=2]
└── C1 (PRUNED)
└── C2 [Value=-]
└── L7 [Leaf=0]
└── L8 [Leaf=-1]
Name : Sohan R
USN : 1BM23CS336
How can I install Python libraries?
Load data from Google Drive
What can I help you build?
Variables
Terminal
4:36 PM
Python 3



Week - 10

Date / /
Page

Adversial Search
(Min - Max Algorithm)

Algorithm :

function Alpha - BETA - SEARCH (state) returns an action

$V \leftarrow$ MAX - VALUE (state, $-\infty$, $+\infty$)

return the action in Actions (state) with value v

function MAX - VALUE (state, α, β) ~~return a utility value~~

if TERMINAL - TEST (state) then return UTILITY (State)

$v \leftarrow -\infty$

for each α in actions (state) do

$v \leftarrow$ MAX (v, MIN - VALUE (RESULT (s, α), α, β)

if $v \geq \beta$ then return v

$\alpha \leftarrow$ MAX (α, v)

return v

function min - value (state, α, β) returns a utility value

if terminal - test (state) then return utility (state)

$v \leftarrow +\infty$

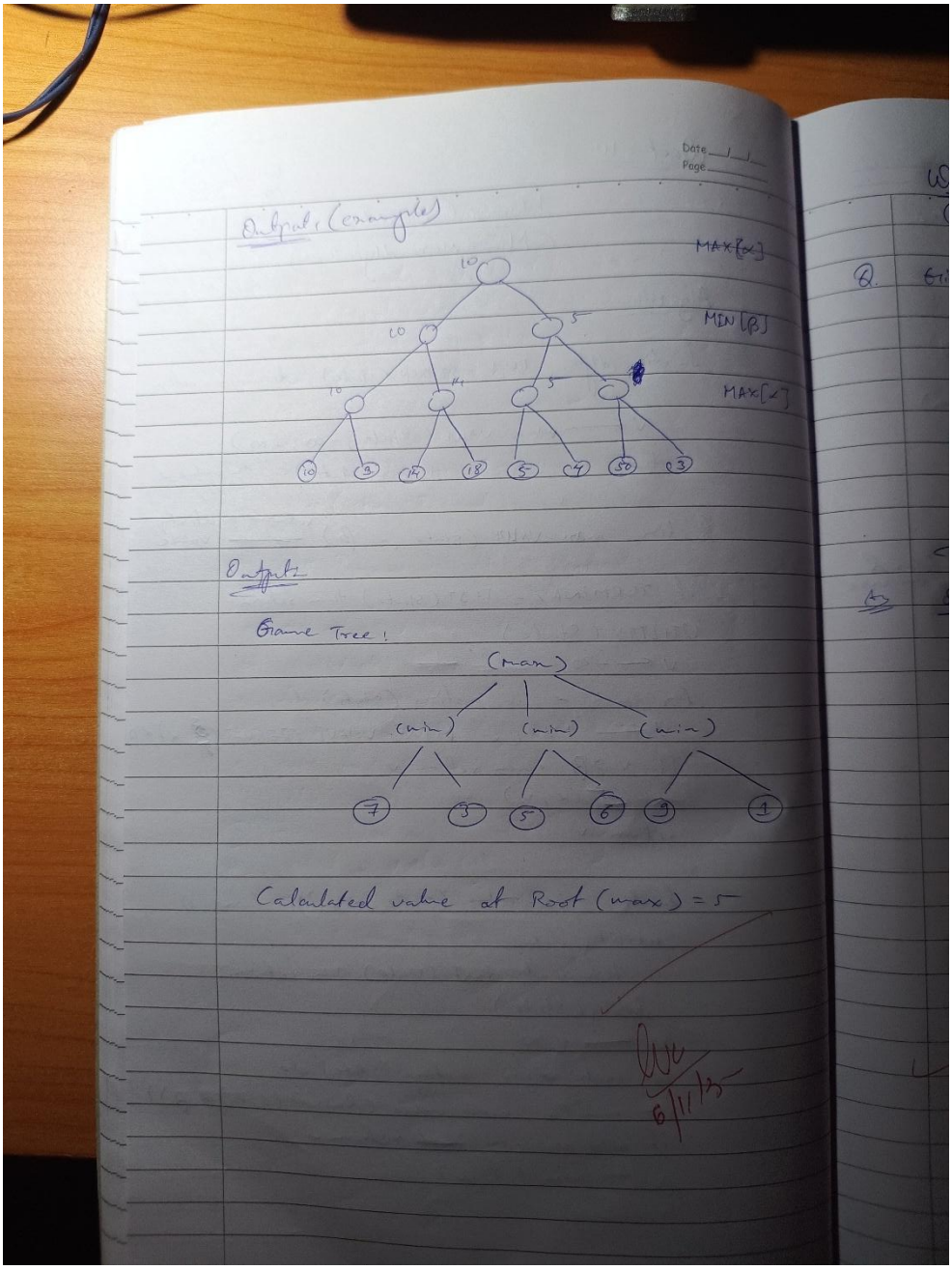
for each α in action (State) do

$v \leftarrow$ min (v, max - value (Result (s, a), α, β))

if $v \leq \alpha$ then return v

$\beta \leftarrow$ min (β, v)

return v.


Output: (example)
MAX [A]
MIN [B]
MAX [C]
10
10
5
10
14
5
10
3
14
18
5
7
50
3
Output:
Game Tree:
(max)
(min)
(min)
(min)
7
3
5
6
9
1
Calculated value at Root (max) = 5
6/11/25