VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Artificial Intelligence (23CS5PCAIN)

Submitted by

SAMIR CHAUDHARY (1BM23CS294)

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 **SAMIR CHAUDHARY (1BM23CS294),** 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.

Lab faculty Incharge Name Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

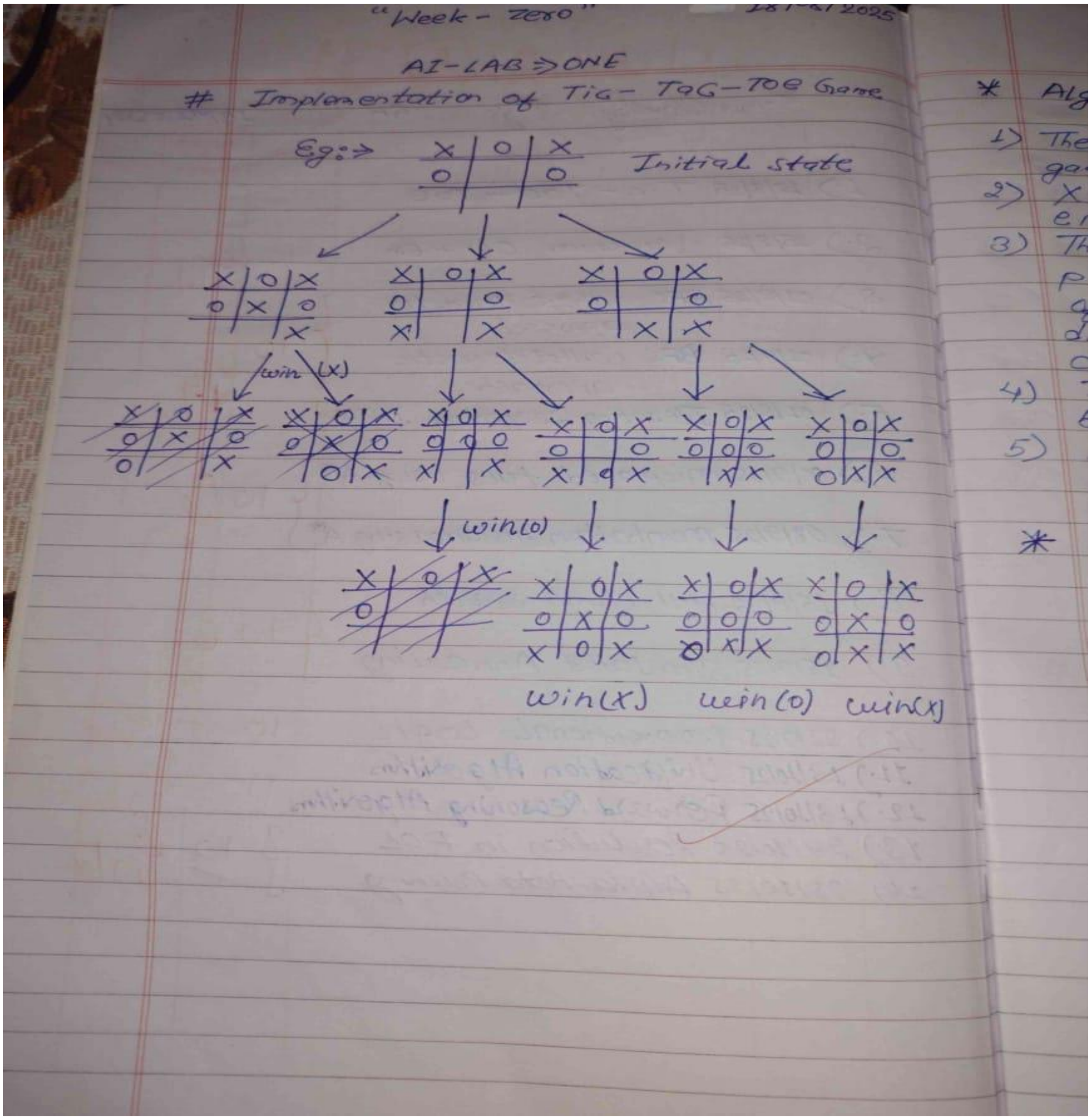
Index

Github Link: https://github.com/1BM23CS294/AILAB-1BM23CS294

Program 1

Implement Tic –Tac –Toe Game

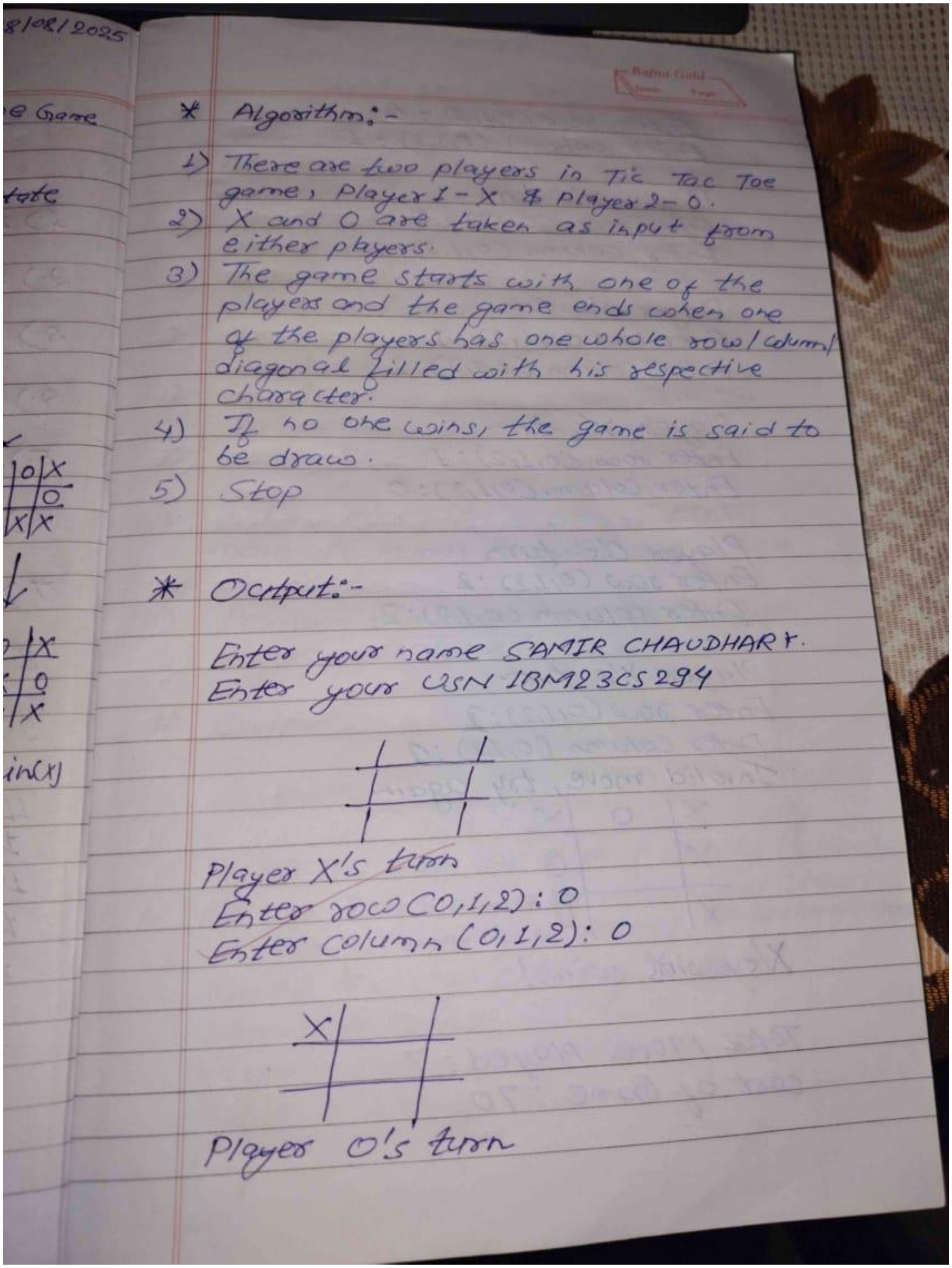
Algorithm:



"Week - zero" 28-01-2025

AI-LAB ⇒ ONE

Implementation of TIC-TAC-TOE Game



8/08/2025

e Game

tate

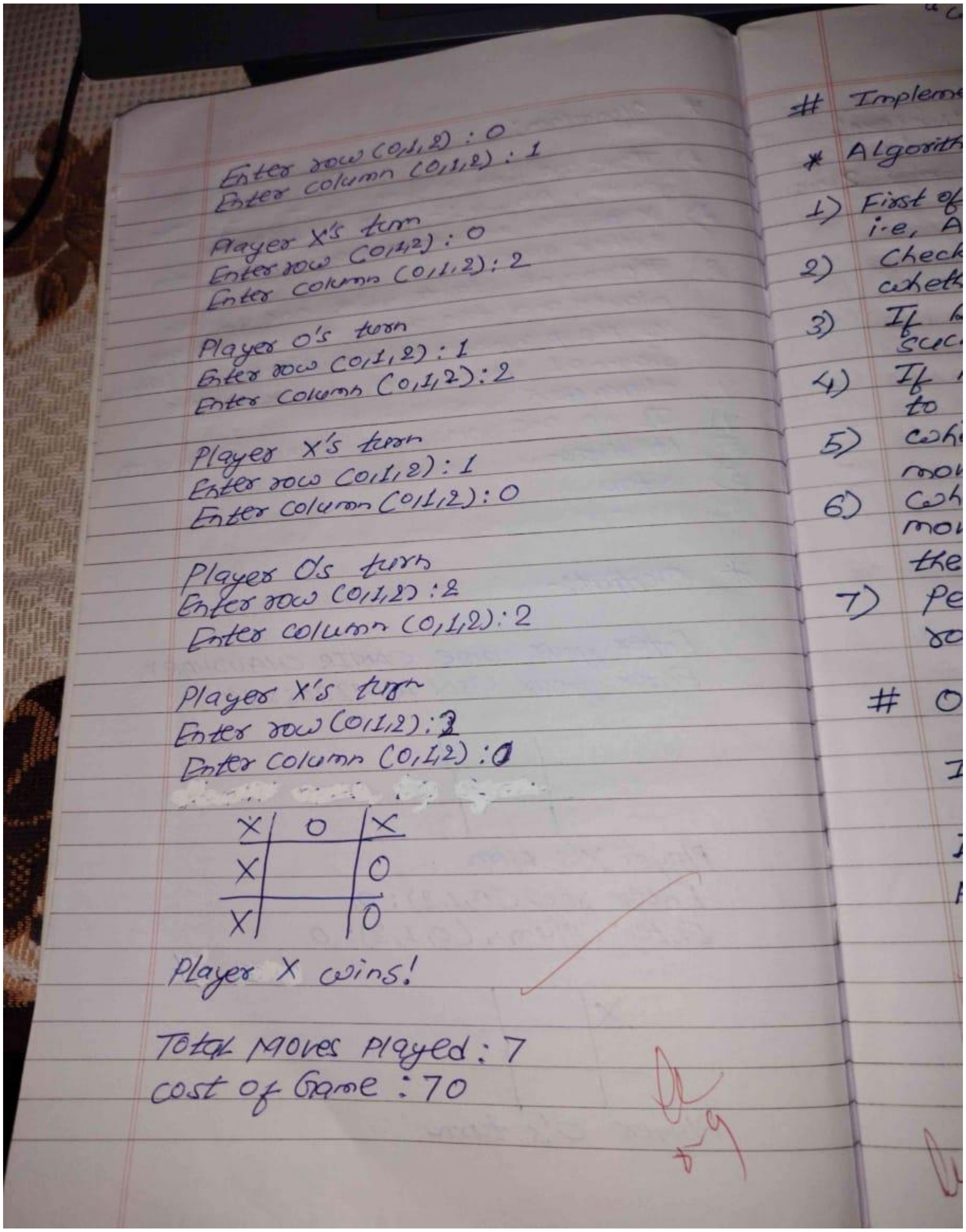
in(x)

* Algorithm:-

1) There are two players in Tic Tac Toe game, Player 1 - X & player 2 - 0.
2) X and O are taken as input from either players.
3) The game starts with one of the players and the game ends when one of the players has one whole row/column/diagonal filled with his respective character.
4) If no one wins, the game is said to be draw.
5) Stop

* Output:-

```
Enter your name SAMIR CHAUDHARY.
Enter your USN 1BM23CS294

  | |
 -+-+-
  | |
 -+-+-
  | |

Player X's turn
Enter row (0,1,2): 0
Enter column (0,1,2): 0

 X| |
 -+-+-
  | |
 -+-+-
  | |

Player O's turn
```



```
Enter row (0,1,2) : 0
Enter column (0,1,2) : 1

Player X's turn
Enter row (0,1,2) : 0
Enter column (0,1,2) : 2

Player O's turn
Enter row (0,1,2) : 1
Enter column (0,1,2) : 2

Player X's turn
Enter row (0,1,2) : 1
Enter column (0,1,2) : 0

Player O's turn
Enter row (0,1,2) : 2
Enter column (0,1,2) : 2

Player X's turn
Enter row (0,1,2) : 2
Enter column (0,1,2) : 0

 X | O | X
 X |   | O
 X |   | O

Player X wins!

Total Moves Played : 7
Cost of Game : 70
```

Code:

```
def print_board(board): print("\n") for row in board: print(" | ".join(row)) print("-" * 9)

def check_winner(board, player): for row in board: if all(s == player for s in row): return True for col in range(3): if all(board[row][col] == player for row in range(3)): return True if all(board[i][i] == player for i in range(3)) or all(board[i][2-i] == player for i in range(3)): return True return False

def tic_tac_toe_simulation(moves): print("Name: SAMIR CHAUDHARY") print("Roll No.: 1BM23CS294\n")

board = [[" " for _ in range(3)] for _ in range(3)]
players = ["X", "O"]
print("Tic-Tac-Toe Simulation:")
print_board(board)

cost = 0
for i, move in enumerate(moves):
    player = players[i % 2]
    row, col = move
    if board[row][col] == " ":
        board[row][col] = player
        cost += 10  # cost per move
        print(f"\nMove {i+1}: Player {player} -> ({row}, {col})")
        print_board(board)
        if check_winner(board, player):
            print(f"Player {player} wins!")
            print(f"Total Cost: {cost}")
            return
    else:
        print(f"Move {i+1}: Cell ({row},{col}) already occupied.")

print("It's a tie!")
print(f"Total Cost: {cost}")
#Pre-defined moves [(row, col), ...]

moves = [ (0,0), (0,1), (1,1), (0,2), (2,2) ]

tic_tac_toe_simulation(moves)
```

Output:

```
Name: SAMIR CHAUDHARY

Roll No.: 1BM23CS294
```

```
Tic-Tac-Toe Simulation:


  |  |
---------
  |  |
---------
  |  |
---------

Move 1: Player X -> (0, 0)


X |  |
---------
  |  |
---------
  |  |
---------

Move 2: Player O -> (0, 1)
```

```
X | O |
---------
  |   |
---------
  |   |
---------
```

```
Move 3: Player X -> (1, 1)
```

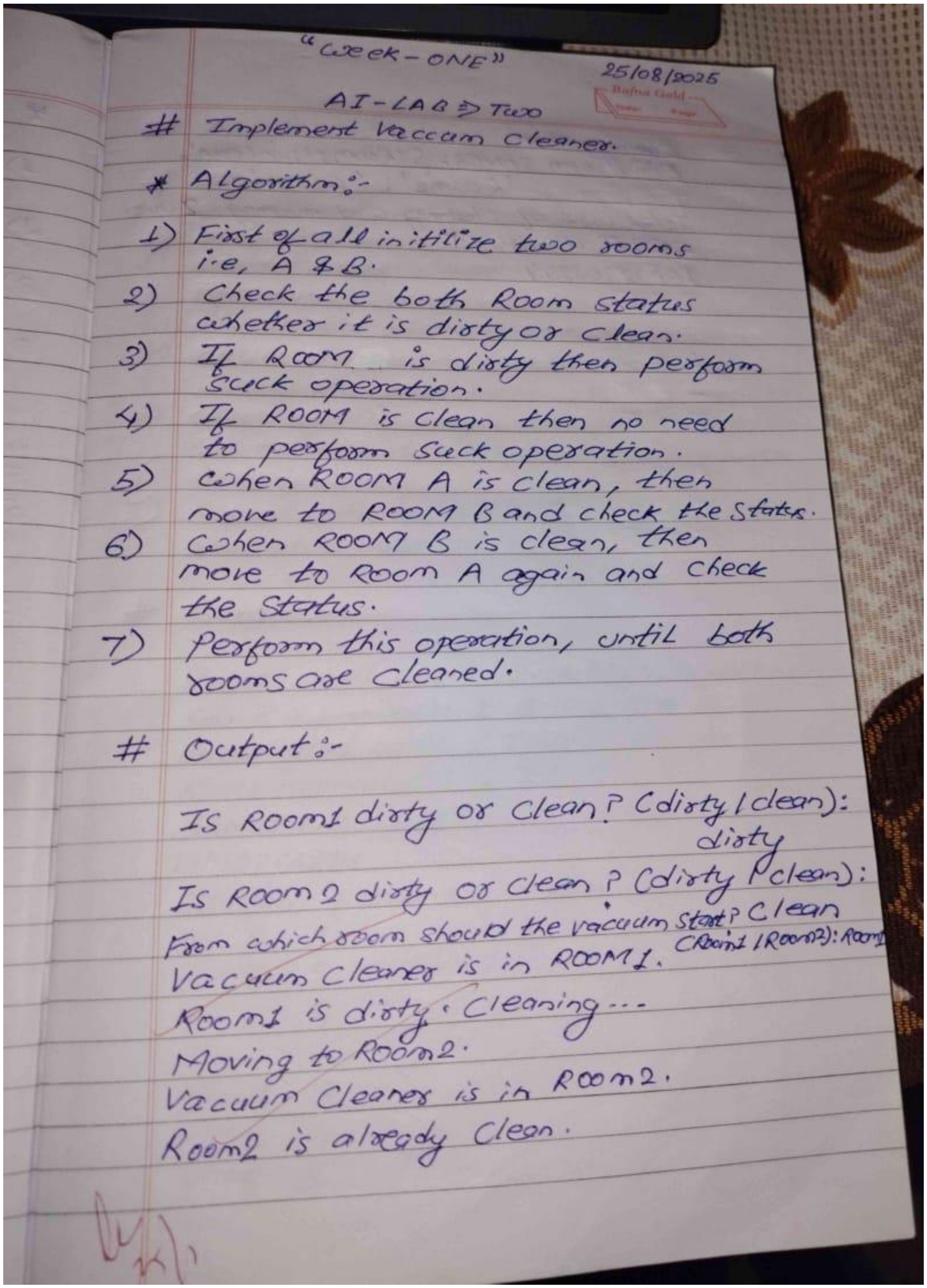
```
X | O |
---------
  | X |
---------
  |   |
---------
```

```
Move 4: Player O -> (0, 2)
```

```
X | O | O
---------
  | X |
---------
  |   |
---------

Move 5: Player X -> (2, 2)


X | O | O
---------
  | X |
---------
  |   | X
---------
Player X wins!
Total Cost: 50
```

Implement vacuum cleaner agent

Algorithm:



"Week-ONE"

25/08/2025

AI-LAB ⇒ Two

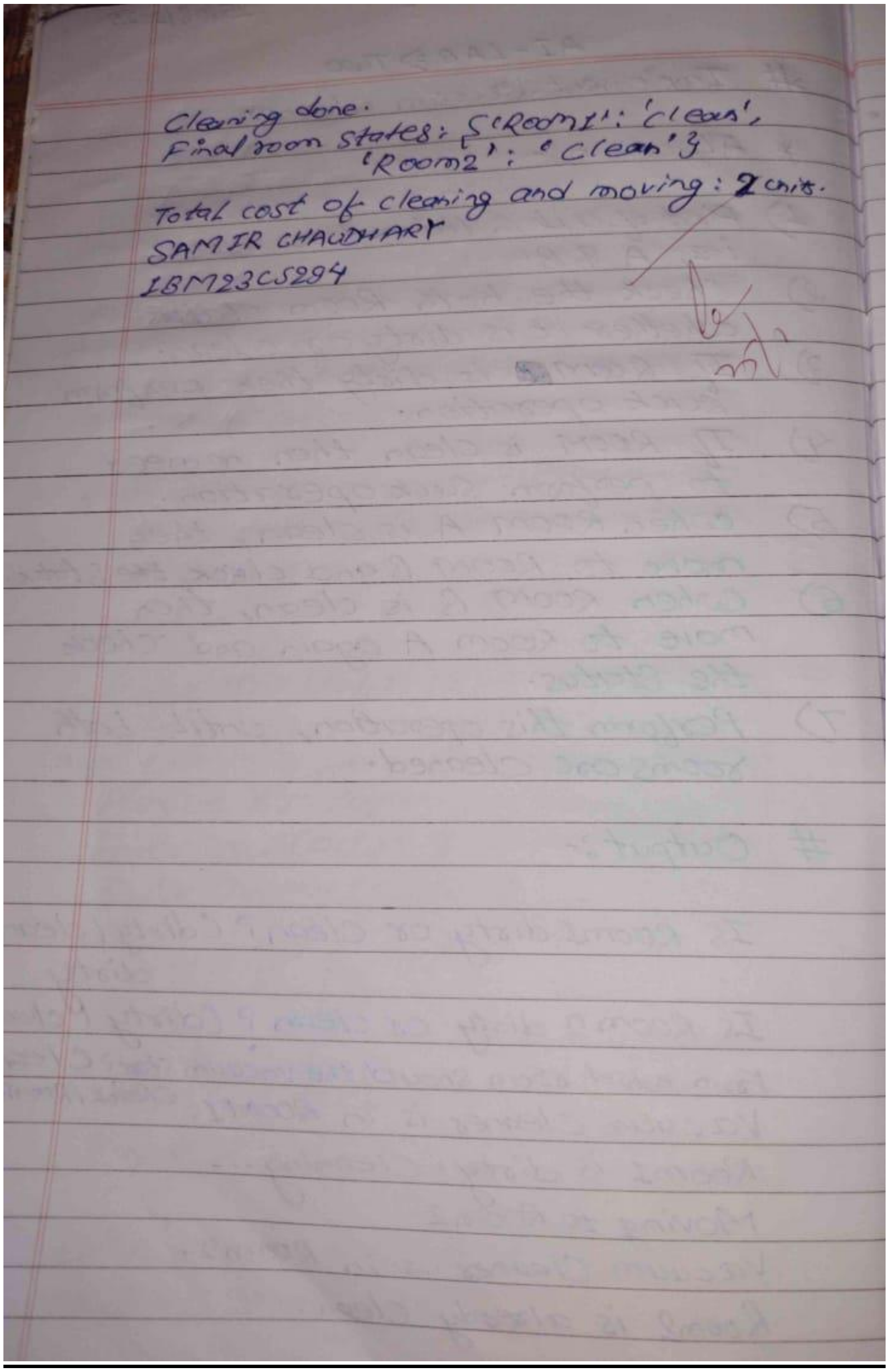
Implement Vaccum Cleaner.

* Algorithm:-

1) First of all initilize two rooms i.e, A & B.
2) Check the both Room status whether it is dirty or Clean.
3) If Room is dirty then perform suck operation.
4) If ROOM is Clean then no need to perform suck operation.
5) When ROOM A is clean, then move to ROOM B and check the Status.
6) When ROOM B is clean, then move to Room A again and check the Status.
7) Perform this operation, until both rooms are Cleaned.

Output:-

```
Is Room1 dirty or Clean? (dirty/clean): dirty
Is Room2 dirty or clean? (dirty/clean): Clean
From which room should the vacuum start? (Room1/Room2): Room1
Vacuum Cleaner is in ROOM1.
Room1 is dirty. Cleaning...
Moving to Room2.
Vacuum Cleaner is in Room2.
Room2 is already Clean.
```



```
Cleaning done.
Final room states: {'Room1': 'clean',
                    'Room2': 'clean'}
Total cost of cleaning and moving: 2 units.
```

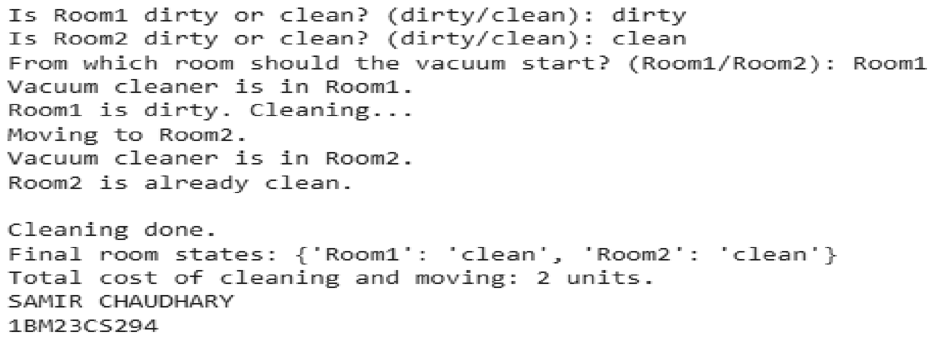
SAMIR CHAUDHARY

1BM23CS294

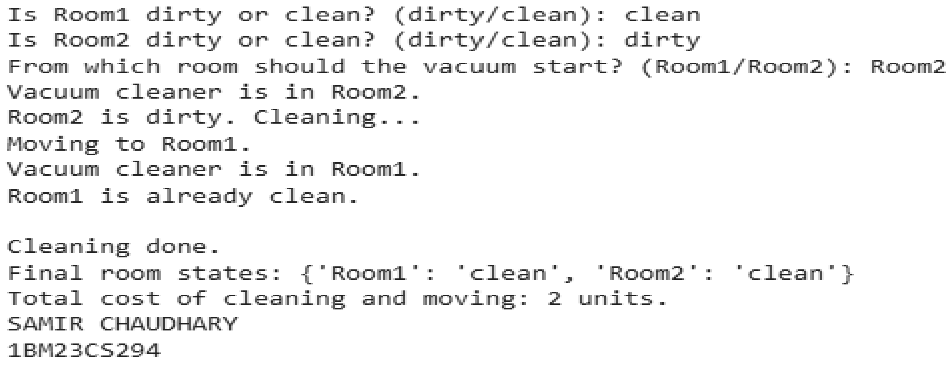
Code:

```
class VacuumCleaner: def init(self): self.rooms = { 'Room1': self.get_room_status('Room1'), 'Room2':
self.get_room_status('Room2') } self.current_room = self.get_starting_room() self.cost = 0

def get_room_status(self, room):
   while True:
      status = input(f"Is {room} dirty or clean? (dirty/clean): ").strip().lower()
      if status in ['dirty', 'clean']:
         return status
      print("Invalid input. Please enter 'dirty' or 'clean'.")

def get_starting_room(self):
   while True:
      room = input("From which room should the vacuum start? (Room1/Room2): ").strip()
      if room in ['Room1', 'Room2']:
         return room
      print("Invalid input. Please enter 'Room1' or 'Room2'.")

def clean_room(self):
   print(f"Vacuum cleaner is in {self.current_room}.")
   if self.rooms[self.current_room] == 'dirty':
      print(f"{self.current_room} is dirty. Cleaning...")
      self.rooms[self.current_room] = 'clean'
      self.cost += 1  # Cleaning cost is 1 unit
   else:
      print(f"{self.current_room} is already clean.")

def move_to_other_room(self):
   self.current_room = 'Room2' if self.current_room == 'Room1' else 'Room1'
   print(f"Moving to {self.current_room}.")
   self.cost += 1  # Moving cost is 1 unit

def run(self):
   self.clean_room()
   self.move_to_other_room()
   self.clean_room()
   print("\nCleaning done.")
   print(f"Final room states: {self.rooms}")
   print(f"Total cost of cleaning and moving: {self.cost} units.")


vacuum = VacuumCleaner() vacuum.run() print("SAMIR CHAUDHARY") print("1BM23CS294 ")
```

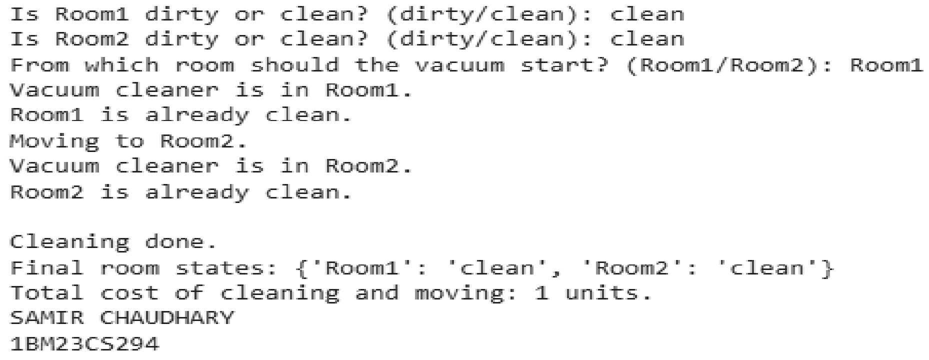
Output:



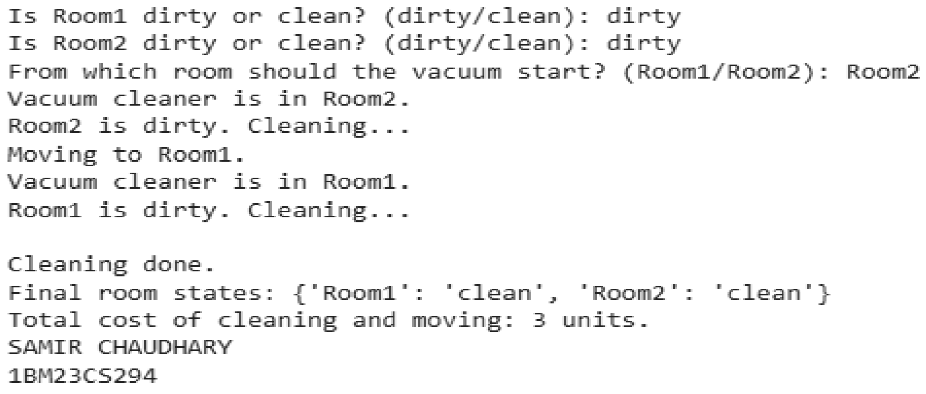
```
Is Room1 dirty or clean? (dirty/clean): dirty
Is Room2 dirty or clean? (dirty/clean): clean
From which room should the vacuum start? (Room1/Room2): Room1
Vacuum cleaner is in Room1.
Room1 is dirty. Cleaning...
Moving to Room2.
Vacuum cleaner is in Room2.
Room2 is already clean.

Cleaning done.
Final room states: {'Room1': 'clean', 'Room2': 'clean'}
Total cost of cleaning and moving: 2 units.
SAMIR CHAUDHARY
1BM23CS294
```



```
Is Room1 dirty or clean? (dirty/clean): clean
Is Room2 dirty or clean? (dirty/clean): dirty
From which room should the vacuum start? (Room1/Room2): Room2
Vacuum cleaner is in Room2.
Room2 is dirty. Cleaning...
Moving to Room1.
Vacuum cleaner is in Room1.
Room1 is already clean.

Cleaning done.
Final room states: {'Room1': 'clean', 'Room2': 'clean'}
Total cost of cleaning and moving: 2 units.
SAMIR CHAUDHARY
1BM23CS294
```



```
Is Room1 dirty or clean? (dirty/clean): clean
Is Room2 dirty or clean? (dirty/clean): clean
From which room should the vacuum start? (Room1/Room2): Room1
Vacuum cleaner is in Room1.
Room1 is already clean.
Moving to Room2.
Vacuum cleaner is in Room2.
Room2 is already clean.

Cleaning done.
Final room states: {'Room1': 'clean', 'Room2': 'clean'}
Total cost of cleaning and moving: 1 units.
SAMIR CHAUDHARY
1BM23CS294
```

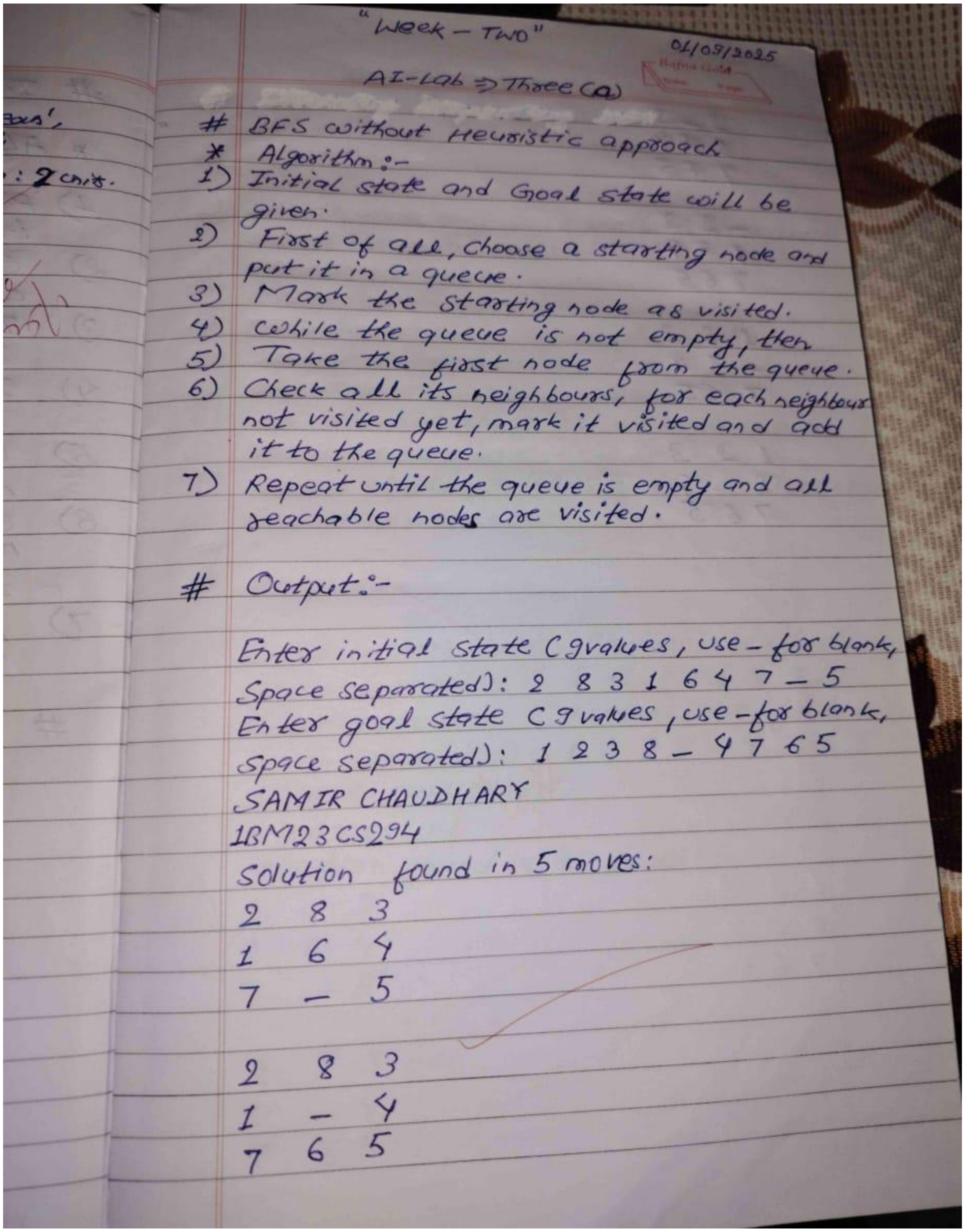


```
Is Room1 dirty or clean? (dirty/clean): dirty
Is Room2 dirty or clean? (dirty/clean): dirty
From which room should the vacuum start? (Room1/Room2): Room2
Vacuum cleaner is in Room2.
Room2 is dirty. Cleaning...
Moving to Room1.
Vacuum cleaner is in Room1.
Room1 is dirty. Cleaning...

Cleaning done.
Final room states: {'Room1': 'clean', 'Room2': 'clean'}
Total cost of cleaning and moving: 3 units.
SAMIR CHAUDHARY
1BM23CS294
```

Program-2

Implement 8 puzzle problems using Breadth First Search (DFS)

Algorithm:



"Week – TWO"

01/09/2025

AI-Lab ⇒ Three (a)

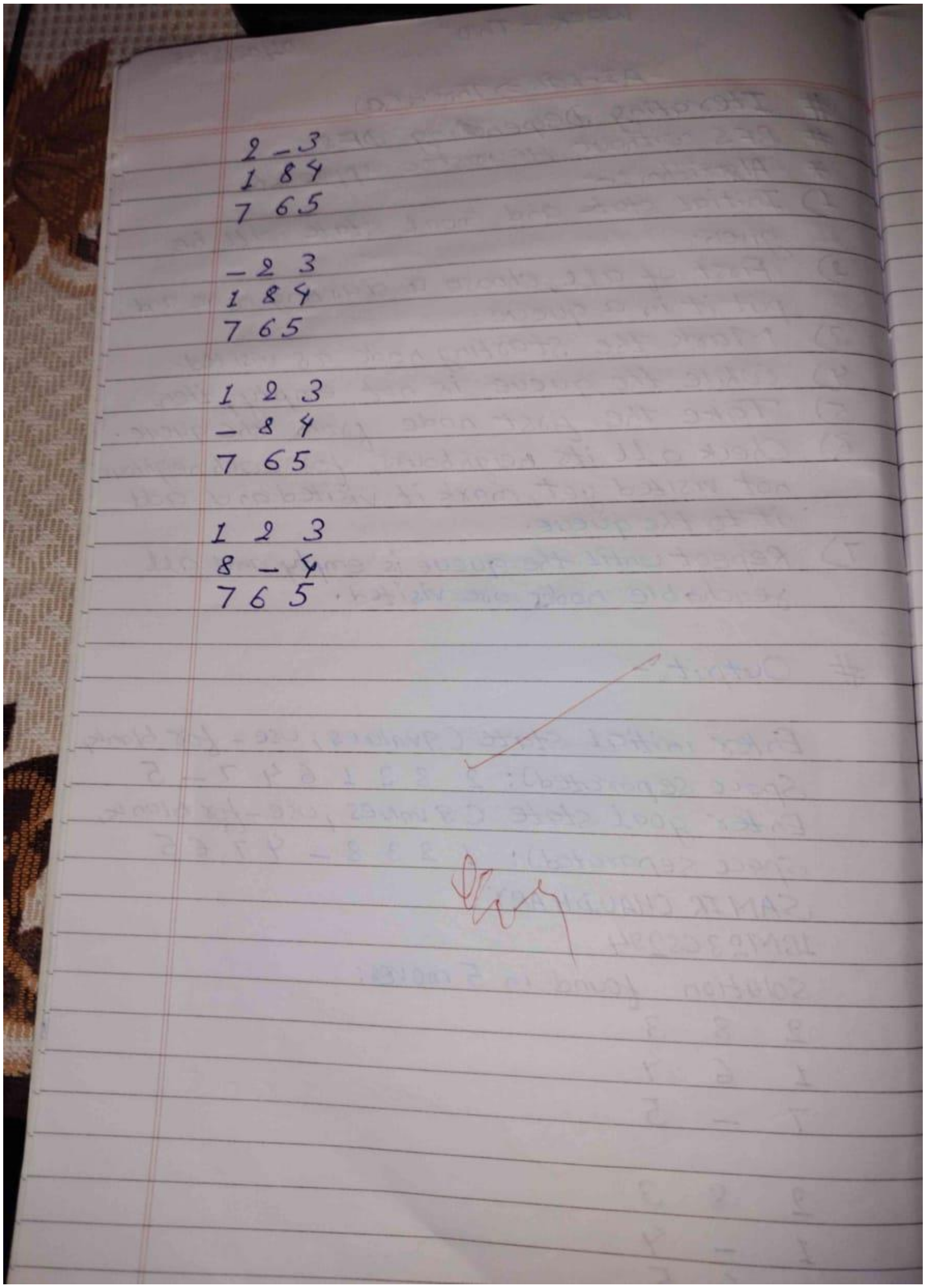
BFS without Heuristic approach

* Algorithm :-

1) Initial state and Goal state will be given.
2) First of all, choose a starting node and put it in a queue.
3) Mark the starting node as visited.
4) While the queue is not empty, then
5) Take the first node from the queue.
6) Check all its neighbours, for each neighbour not visited yet, mark it visited and add it to the queue.
7) Repeat until the queue is empty and all reachable nodes are visited.

Output :-

```
Enter initial state (9values, use - for blank,
Space separated): 2 8 3 1 6 4 7 - 5
Enter goal state (9 values, use - for blank,
space separated): 1 2 3 8 - 4 7 6 5
SAMIR CHAUDHARY
1BM23CS294
Solution found in 5 moves:
2 8 3
1 6 4
7 - 5

2 8 3
1 - 4
7 6 5
```



```
2 - 3
1 8 4
7 6 5

- 2 3
1 8 4
7 6 5

1 2 3
- 8 4
7 6 5

1 2 3
8 - 4
7 6 5
```

Code:

```
from collections import deque
```

```
def list_to_tuple(state): return tuple(state)
```

```
def get_neighbors(state): neighbors = [] state_list = list(state) blank_idx = state_list.index('_')
```

```
moves = {
    'up': -3,
    'down': 3,
    'left': -1,
    'right': 1
}

for move, pos_shift in moves.items():
    new_idx = blank_idx + pos_shift
    if move == 'up' and blank_idx < 3:
        continue
    if move == 'down' and blank_idx > 5:
        continue
    if move == 'left' and blank_idx % 3 == 0:
        continue
    if move == 'right' and blank_idx % 3 == 2:
        continue
    new_state = state_list[:]
    new_state[blank_idx], new_state[new_idx] = new_state[new_idx],
new_state[blank_idx]
    neighbors.append(tuple(new_state))
return neighbors
```

```
def bfs(start, goal): start = tuple(start) goal = tuple(goal) queue = deque([(start, [start])]) visited = set([start]) while queue: current, path = queue.popleft() if current == goal: return path for neighbor in get_neighbors(current): if neighbor not in visited: visited.add(neighbor) queue.append((neighbor, path + [neighbor])) return None
```

```
def print_state(state): for i in range(0, 9, 3): print(' '.join(str(x) for x in state[i:i+3])) print()
```

```
initial_state_input = input("Enter initial state (9 values, use _ for blank, space separated): ").split() goal_state_input = input("Enter goal state (9 values, use _ for blank, space separated): ").split()
```

```
path = bfs(initial_state_input, goal_state_input) print("SAMIR CHAUDHARY") print("1BM23CS294") if path: print(f"Solution found in {len(path)-1} moves:") for step in path: print_state(step) else: print("No solution found.")
```

Output:

```
Enter initial state (9 values, use _ for blank, space separated): 2 8 3 1 6 4 7 _ 5

Enter goal state (9 values, use _ for blank, space separated): 1 2 3 8 _ 4 7 6 5

SAMIR CHAUDHARY

1BM23CS294
```

```
Solution found in 5 moves: 2 8 3 1 6 4 7 _ 5

2 8 3 1 _ 4 7 6 5

2 _ 3 1 8 4 7 6 5

_ 2 3 1 8 4 7 6 5

1 2 3 _ 8 4 7 6 5

1 2 3 8 _ 4 7 6 5
```

Implement 8 puzzle problems using Depth First Search (DFS)

Algorithm:

```
def get_neighbors(state): neighbors = [] state_list = list(state) blank_idx = state_list.index('_')
moves = {
    'up': -3,
    'down': 3,
    'left': -1,
    'right': 1
}

for move, pos_shift in moves.items():
    new_idx = blank_idx + pos_shift

    if move == 'up' and blank_idx < 3:
        continue
    if move == 'down' and blank_idx > 5:
        continue
    if move == 'left' and blank_idx % 3 == 0:
        continue
    if move == 'right' and blank_idx % 3 == 2:
        continue

    new_state = state_list[:]
    new_state[blank_idx], new_state[new_idx] = new_state[new_idx],
new_state[blank_idx]
    neighbors.append(tuple(new_state))

return neighbors
```

```
def dfs(start, goal, max_depth=50): start = tuple(start) goal = tuple(goal)

stack = [(start, [start])]
visited = set()

while stack:
    current, path = stack.pop()

    if current == goal:
        return path

    if current not in visited and len(path) <= max_depth:
        visited.add(current)
        for neighbor in reversed(get_neighbors(current)):
            if neighbor not in visited:
                stack.append((neighbor, path + [neighbor]))

return None
```

```
def print_state(state): for i in range(0, 9, 3): print(' '.join(str(x) for x in state[i:i+3])) print()

initial_state = list("2831647_5") goal_state = list("1238_4765")

path = dfs(initial_state, goal_state) print("SAMIR CHAUDHARY"); print("1BM23CS294");

if path: print(f"Solution found in {len(path)-1} moves:") for step in path: print_state(step)

else: print("No solution found (or depth limit reached).")
```

Output:

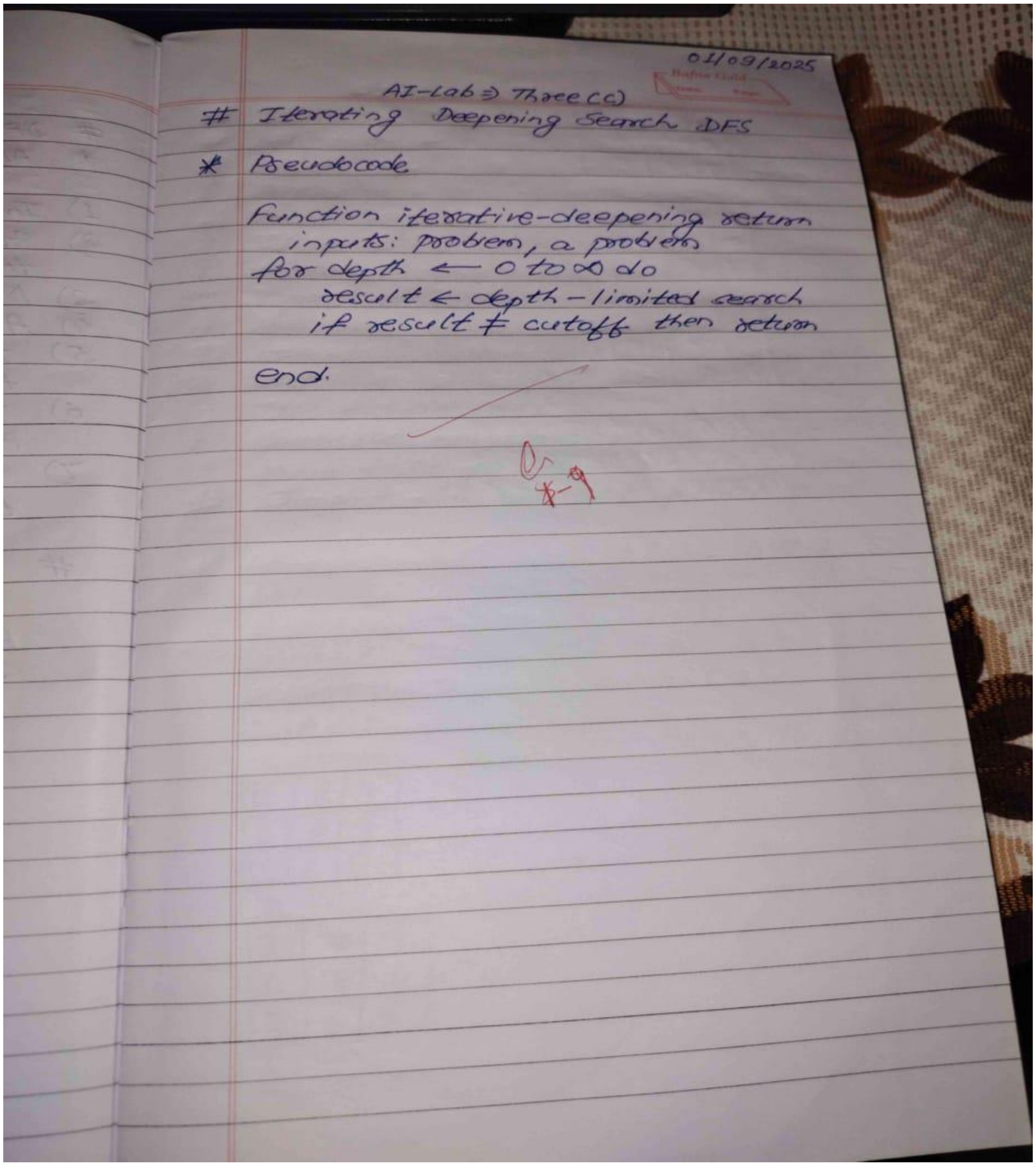
SAMIR CHAUDHARY

1BM23CS294

```
Solution found in 9 moves:

2 8 3

1 6 4

7 _ 5


2 8 3

1 _ 4

7 6 5


2 _ 3

1 8 4

7 6 5


_ 2 3

1 8 4

7 6 5


1 2 3

_ 8 4

7 6 5


1 2 3

8 _ 4

7 6 5
```

Implement Iterative deepening search algorithm

Algorithm:



01/09/2025

AI-Lab ⇒ Three (c)

Iterating Deepening Search DFS

* Pseudocode

```
function iterative-deepening return
  inputs: probrem, a problem
for depth ← 0 to ∞ do
    result ← depth-limited search
    if result ≠ cutoff then return
end.
```

Code:

```
def get_neighbors(state): neighbors = [] blank = state.index(0) x, y = divmod(blank, 3) moves = [(-1,0), (1,0), (0,-1), (0,1)] for dx, dy in moves: nx, ny = x + dx, y + dy if 0 <= nx < 3 and 0 <= ny < 3: new_blank = nx*3 + ny new_state = list(state) new_state[blank], new_state[new_blank] = new_state[new_blank], new_state[blank] neighbors.append(tuple(new_state)) return neighbors
```

def depth_limited_search(state, goal, limit, path, visited): if state == goal: return path if limit == 0: return None visited.add(state) for neighbor in get_neighbors(state): if neighbor not in visited: result = depth_limited_search(neighbor, goal, limit - 1, path + [neighbor], visited) if result is not None: return result return None

def iterative_deepening_search(initial, goal): depth = 0 while True: visited = set() result = depth_limited_search(initial, goal, depth, [initial], visited) if result is not None: return result, depth depth += 1

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

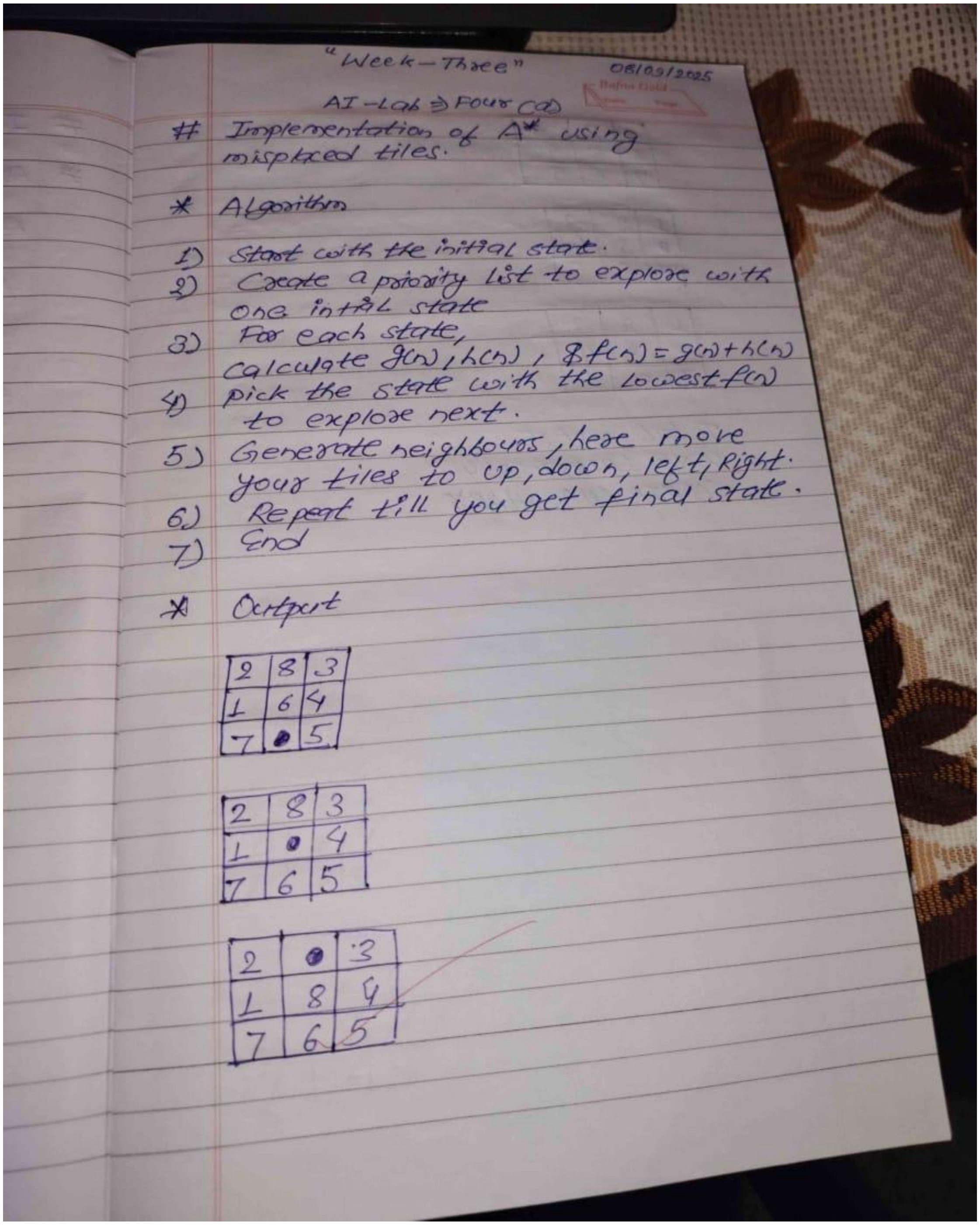
if **name** == "**main**": print("Enter initial state (9 numbers, use 0 for blank):") initial = tuple(map(int, input().split())) print("Enter goal state (9 numbers, use 0 for blank):") goal = tuple(map(int, input().split())) path, depth = iterative_deepening_search(initial, goal) print("\nSolution found at depth:", depth) print("Number of moves:", len(path)-1) print("\nSteps:") for step in path: print_state(step)

print("SAMIR CHAUDHARY\n1BM23CS294")

Program-3

Implement A* search algorithm

Algorithm: Using misplaced tiles



"Week-Three" 08/09/2025

AI-Lab ⇒ Four (a)

Implementation of A* using misplaced tiles.

* Algorithm

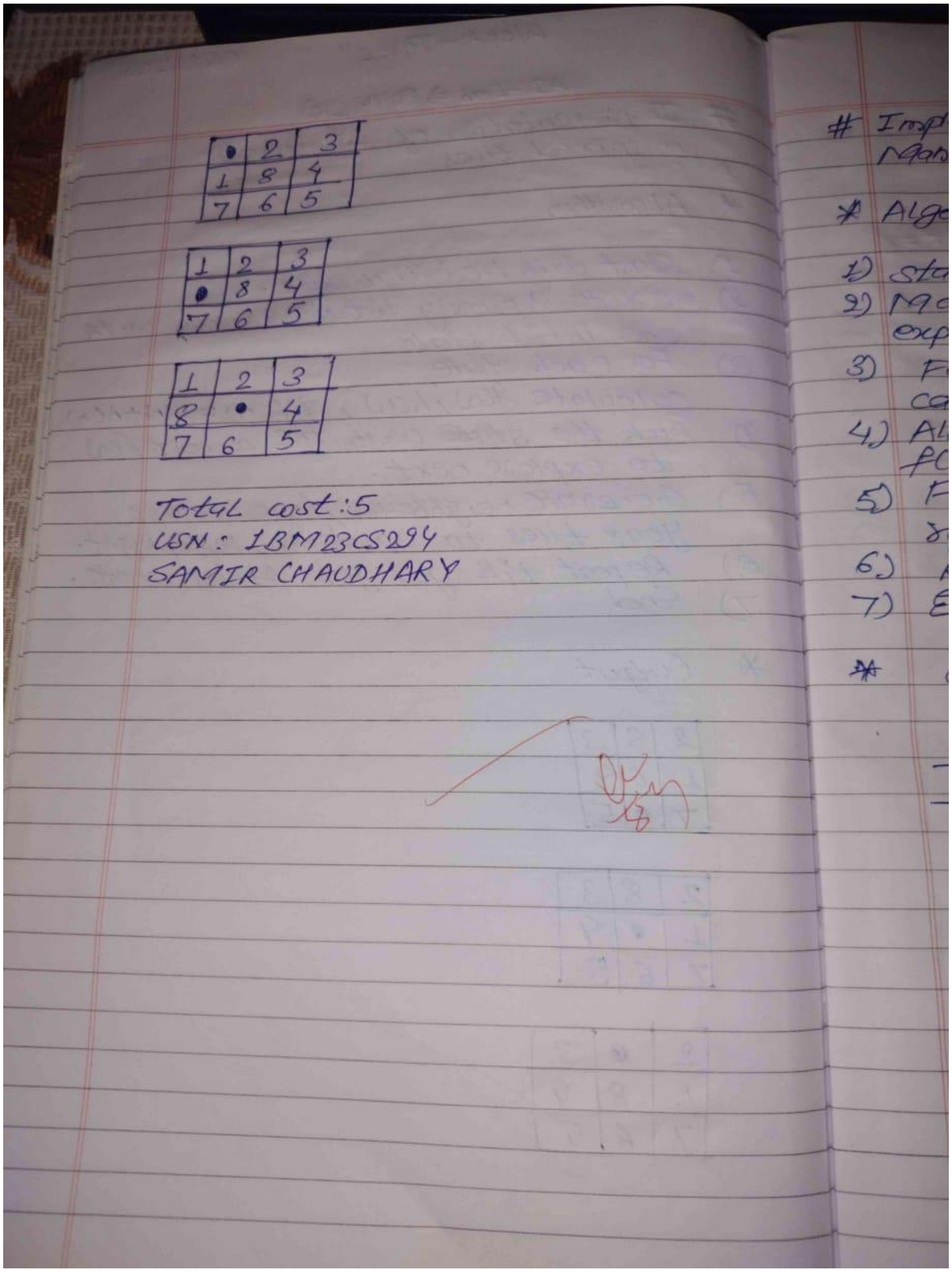
1) Start with the initial state.
2) Create a priority list to explore with one intial state
3) For each state, calculate $g(n)$, $h(n)$, & $f(n)=g(n)+h(n)$
4) pick the state with the lowest $f(n)$ to explore next.
5) Generate neighbours, here move your tiles to up, down, left, Right.
6) Repeat till you get final state.
7) End

* Output

2	8	3
1	6	4
7	●	5

2	8	3
1	●	4
7	6	5

2	●	3
1	8	4
7	6	5



●	2	3
1	8	4
7	6	5

1	2	3
●	8	4
7	6	5

1	2	3
8	●	4
7	6	5

Total cost:5
USN: 1BM23CS294
SAMIR CHAUDHARY

Imp
Nan

* Alg

1) Sta
2) Mo
exp
3) F
ca
4) Al
fc
5) F
6)
7) E

*

Code:

```
import heapq
```

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

class Node: def **init**(self, state, parent=None, g=0): self.state = state self.parent = parent self.g = g self.h = self.misplaced_tiles() self.f = self.g + self.h

```
def misplaced_tiles(self):
    return sum(
        1
        for i in range(3)
        for j in range(3)
        if self.state[i][j] != 0 and self.state[i][j] != goal_state[i][j]
    )

def __lt__(self, other):
    return self.f < other.f
```

def get_neighbors(state): neighbors = [] x, y = next((i, j) for i in range(3) for j in range(3) if state[i][j] == 0) moves = [(-1, 0), (1, 0), (0, -1), (0, 1)] for dx, dy in moves: nx, ny = x + dx, y + dy if 0 <= nx < 3 and 0 <= ny < 3: new_state = [row[:] for row in state] new_state[x][y], new_state[nx][ny] = new_state[nx][ny], new_state[x][y] neighbors.append(new_state) return neighbors

def state_to_tuple(state): return tuple(tuple(row) for row in state)

def reconstruct_path(node): path = [] while node: path.append(node.state) node = node.parent return path[::-1]

def a_star(start_state): start_node = Node(start_state) open_list = [] heapq.heappush(open_list, start_node) closed_set = set() while open_list: current = heapq.heappop(open_list) if current.state == goal_state: return reconstruct_path(current), current.g closed_set.add(state_to_tuple(current.state)) for neighbor in get_neighbors(current.state): if state_to_tuple(neighbor) in closed_set: continue neighbor_node = Node(neighbor, current, current.g + 1) heapq.heappush(open_list, neighbor_node) return None, -1

start_state = [[2, 8, 3], [1, 6, 4], [7, 0, 5]] solution, cost = a_star(start_state)

if solution: print("Solution Path:") for step in solution: for row in step: print(row) print() print("Total Cost:", cost) else: print("No solution found.")

print("USN: 1BM23CS294") print("SAMIR CHAUDHARY")

Output:

Solution Path:

[2, 8, 3]

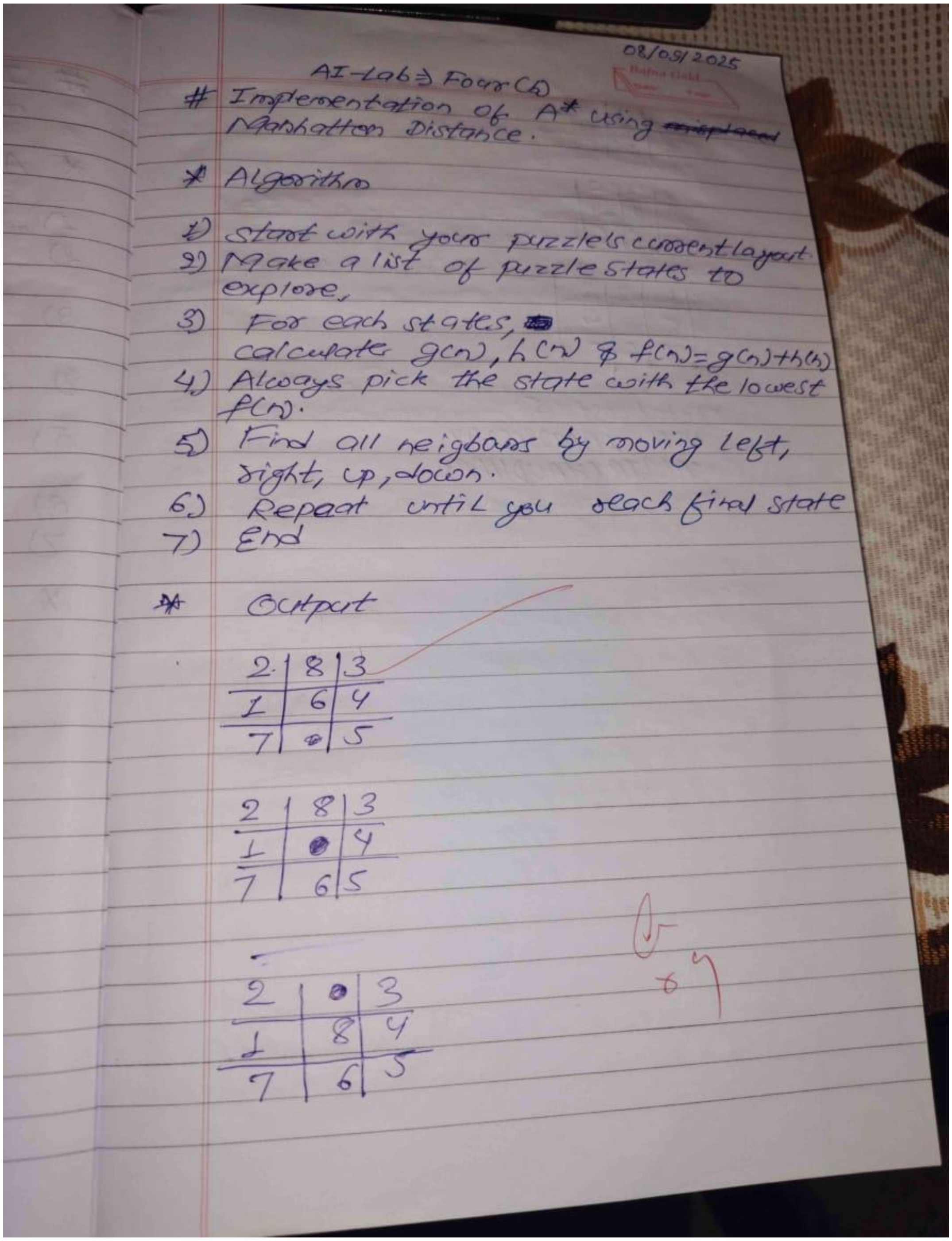
[1, 6, 4]

[7, 0, 5]

[2, 8, 3]

```
[1, 0, 4]
[7, 6, 5]

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

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

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

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

Total Cost: 5
USN: 1BM23CS294
SAMIR CHAUDHARY
```

Algorithm: Using Manhatten Distance



08/09/2025

AI-Lab ⇒ Four (b)

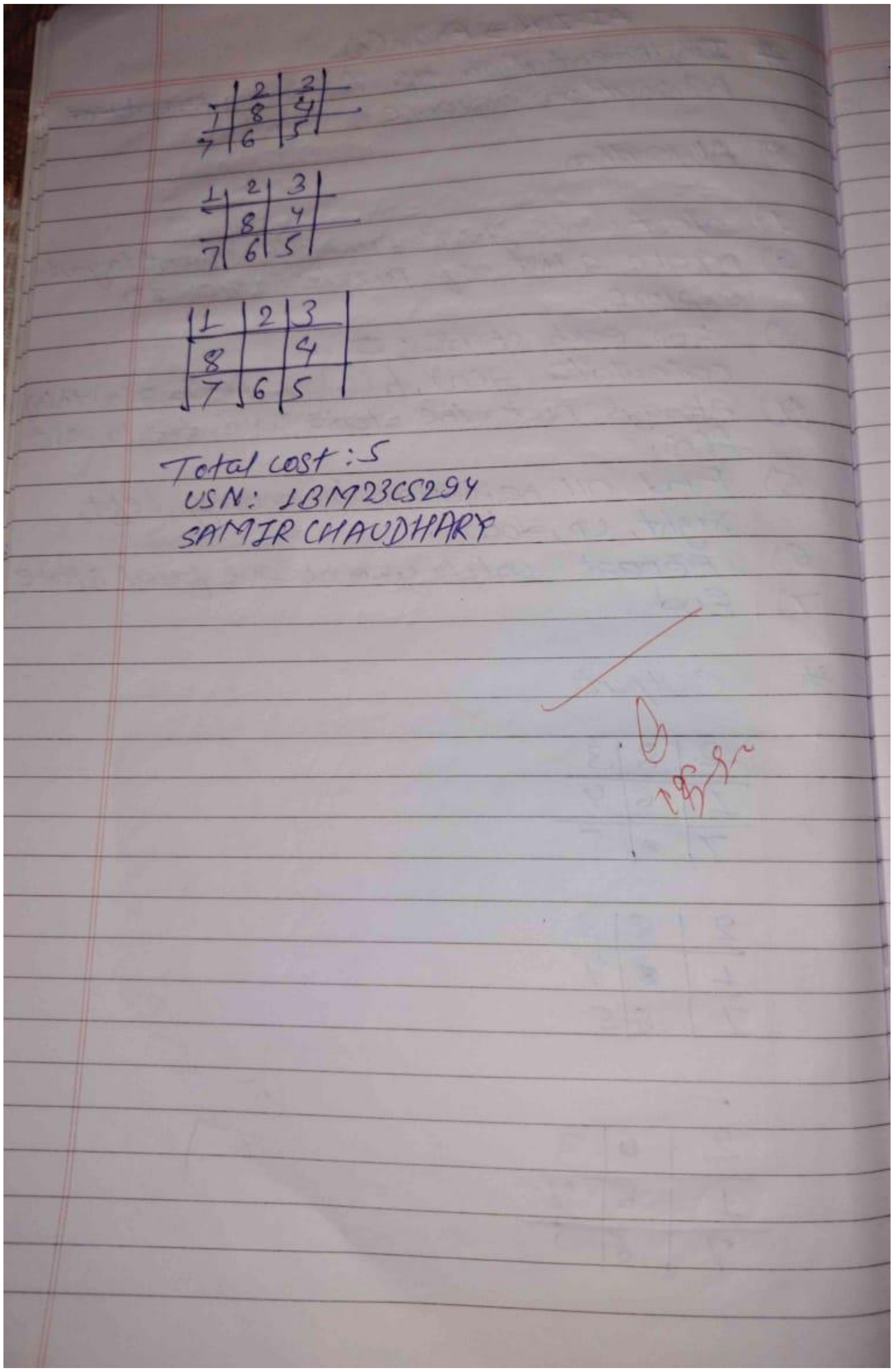
Implementation of A* using ~~misplaced~~ Manhatten Distance.

* Algorithm

1) Start with your puzzle's current layout.
2) Make a list of puzzle states to explore,
3) For each states, calculate g(n), h(n) & f(n)=g(n)+h(n)
4) Always pick the state with the lowest f(n).
5) Find all neigbours by moving left, right, up, down.
6) Repeat until you reach final state
7) End

* Output

```
2 | 8 | 3
1 | 6 | 4
7 | ● | 5

2 | 8 | 3
1 | ● | 4
7 | 6 | 5

2 | ● | 3
1 | 8 | 4
7 | 6 | 5
```



	2	3
1	8	4
7	6	5

1	2	3
	8	4
7	6	5

1	2	3
8		4
7	6	5

Total cost : 5
USN: 1BM23CS294
SAMIR CHAUDHARY

Code:

```
import heapq
```

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

class Node: def **init**(self, state, parent=None, g=0): self.state = state self.parent = parent self.g = g self.h = self.manhattan_distance() self.f = self.g + self.h

```
def manhattan_distance(self):
    dist = 0
    for i in range(3):
        for j in range(3):
            val = self.state[i][j]
            if val != 0:
                for x in range(3):
                    for y in range(3):
                        if goal_state[x][y] == val:
                            dist += abs(x - i) + abs(y - j)
                            break
    return dist

def __lt__(self, other):
    return self.f < other.f
```

def get_neighbors(state): neighbors = [] x, y = next((i, j) for i in range(3) for j in range(3) if state[i][j] == 0) moves = [(-1, 0), (1, 0), (0, -1), (0, 1)] for dx, dy in moves: nx, ny = x + dx, y + dy if 0 <= nx < 3 and 0 <= ny < 3: new_state = [row[:] for row in state] new_state[x][y], new_state[nx][ny] = new_state[nx][ny], new_state[x][y] neighbors.append(new_state) return neighbors

def state_to_tuple(state): return tuple(tuple(row) for row in state)

def reconstruct_path(node): path = [] while node: path.append(node.state) node = node.parent return path[::-1]

def a_star(start_state): start_node = Node(start_state) open_list = [] heapq.heappush(open_list, start_node) closed_set = set() while open_list: current = heapq.heappop(open_list) if current.state == goal_state: return reconstruct_path(current), current.g closed_set.add(state_to_tuple(current.state)) for neighbor in get_neighbors(current.state): if state_to_tuple(neighbor) in closed_set: continue neighbor_node = Node(neighbor, current, current.g + 1) heapq.heappush(open_list, neighbor_node) return None, -1start_state = [[2, 8, 3], [1, 6, 4], [7, 0, 5]] solution, cost = a_star(start_state)

if solution: print("Solution Path:") for step in solution: for row in step: print(row) print() print("Total Cost:", cost) else: print("No solution found.")

print("USN: 1BM23CS294") print("SAMIR CHAUDHARY")

Output:

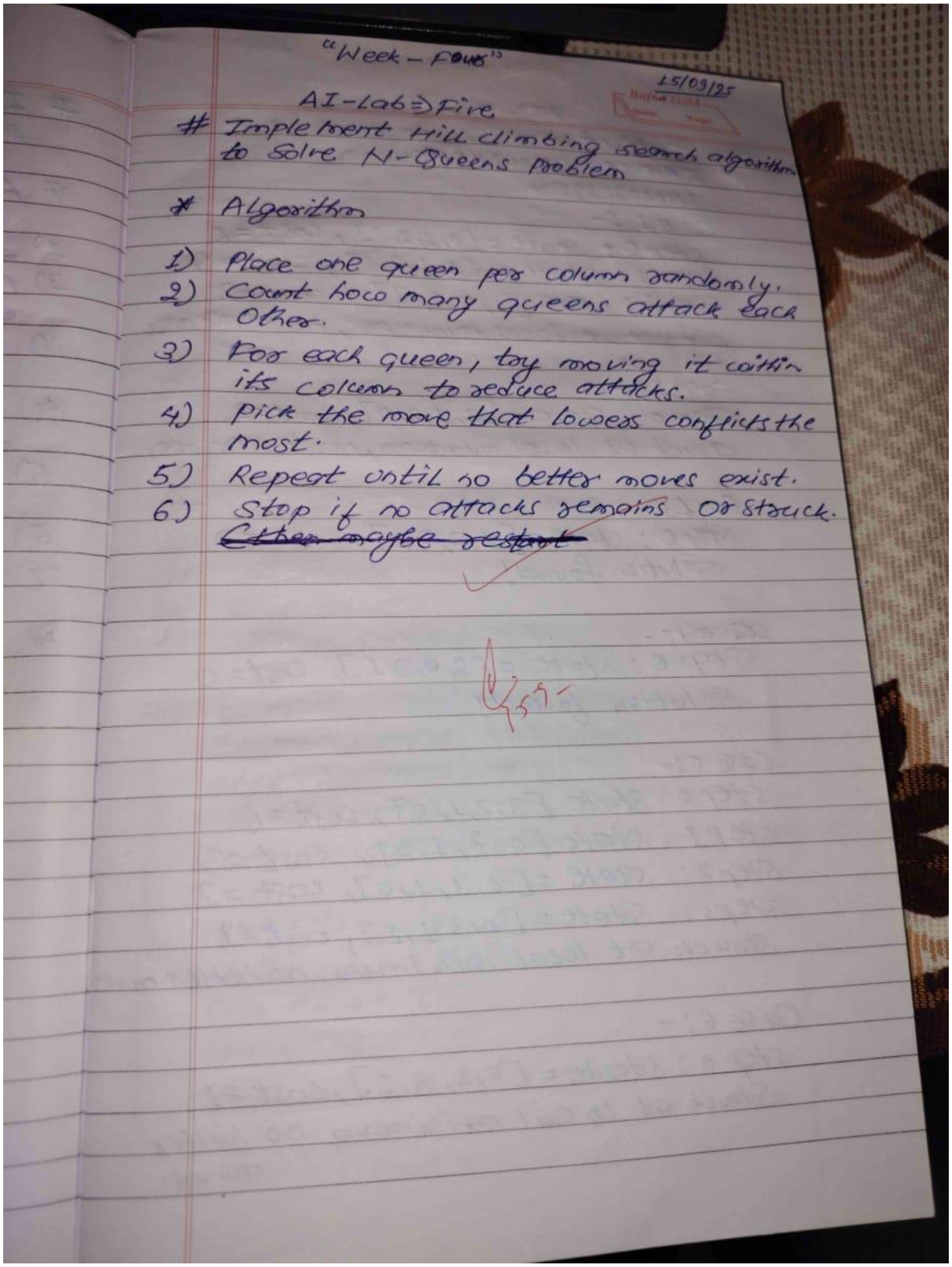
```
Solution Path:

[2, 8, 3]

[1, 6, 4]

[7, 0, 5]
```

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

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

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

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

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

Total Cost: 5
USN: 1BM23CS294
SAMIR CHAUDHARY
```

Program-4

Implement Hill Climbing search algorithm to solve N-Queens Problem

Algorithm:



"Week - Four"

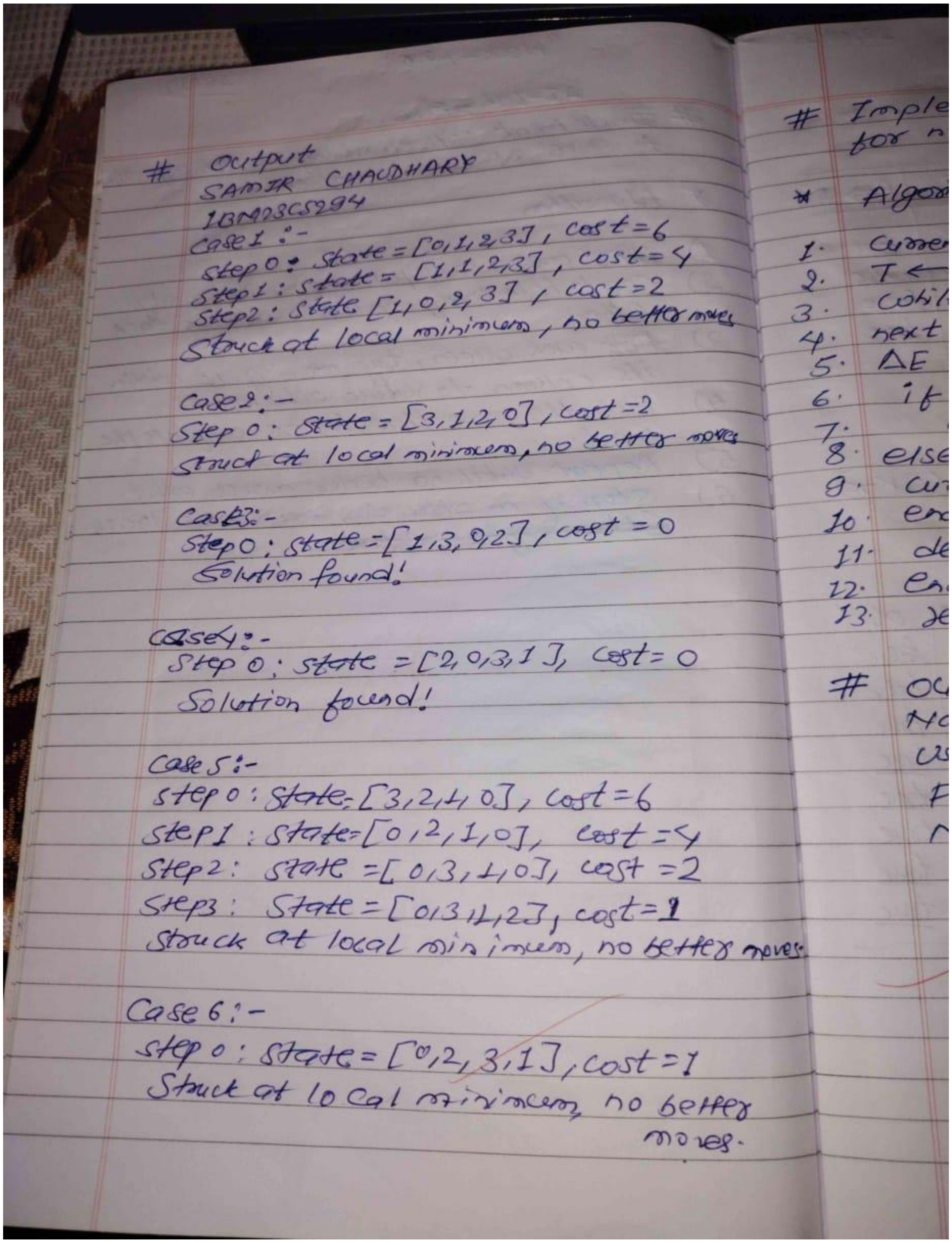
15/09/25

AI-Lab ⇒ Five

Implement Hill climbing search algorithm to Solve N-Queens Problem

Algorithm

1) Place one queen per column randomly.
2) Count how many queens attack each Other.
3) For each queen, try moving it within its column to reduce attacks.
4) Pick the move that lowers conflicts the most.
5) Repeat until no better moves exist.
6) Stop if no attacks remains Or Stuck.
~~(then maybe restart~~



Output
SAMIR CHAUDHARY
1BM23CS294
case1 :-
step 0: state = [0,1,2,3], cost = 6
step1 : state = [1,1,2,3], cost = 4
step2 : state [1,0,2,3], cost = 2
Struck at local minimum, no better moves

case2 :-
Step 0: state = [3,1,2,0], cost = 2
struck at local minimum, no better moves

case3:-
Step0 : state = [1,3,0,2], cost = 0
Solution found!

case4:-
Step 0 : state = [2,0,3,1], cost = 0
Solution found!

case5:-
step0 : state = [3,2,1,0], cost = 6
step1 : state = [0,2,1,0], cost = 4
step2 : state = [0,3,1,0], cost = 2
step3 : State = [0,3,1,2], cost = 1
struck at local minimum, no better moves.

Case 6 :-
step 0 : state = [0,2,3,1], cost = 1
Struck at local minimum, no better
moves.

# Imple
for n

* Algor

1. Curren
2. T ←
3. whil
4. next
5. ΔE
6. if
7.
8. else
9. cu
10. en
11. de
12. en
13. re

Ou
No
Us
F
N

Code:

```
def calculate_cost(board): conflicts = 0 n = len(board) for i in range(n): for j in range(i + 1, n): if board[i] == board[j]
or abs(board[i] - board[j]) == abs(i - j): conflicts += 1 return conflicts
```

```
def generate_neighbors(board): neighbors = [] n = len(board) for col in range(n): for row in range(n): if board[col] != row: new_board = board[:] new_board[col] = row neighbors.append(new_board) return neighbors

def hill_climbing_from_start(start_board): current = start_board[:] step = 0 while True: cost = calculate_cost(current) print(f"Step {step}: State={current}, Cost={cost}") if cost == 0: print("Solution found!\n") break neighbors = generate_neighbors(current) neighbor_costs = [(neighbor, calculate_cost(neighbor)) for neighbor in neighbors] best_neighbor, best_cost = min(neighbor_costs, key=lambda x: x[1]) if best_cost >= cost: print("Stuck at local minimum, no better moves.\n") break current = best_neighbor step += 1

print("SAMIR CHAUDHARY") print("1BM23CS294\n")

starting_boards = [ [0, 1, 2, 3], [3, 1, 2, 0], [1, 3, 0, 2], [2, 0, 3, 1], [3, 2, 1, 0], [0, 2, 3, 1] ]

for i, board in enumerate(starting_boards, 1): print(f"--- Case {i} ---") hill_climbing_from_start(board)
```

Output:

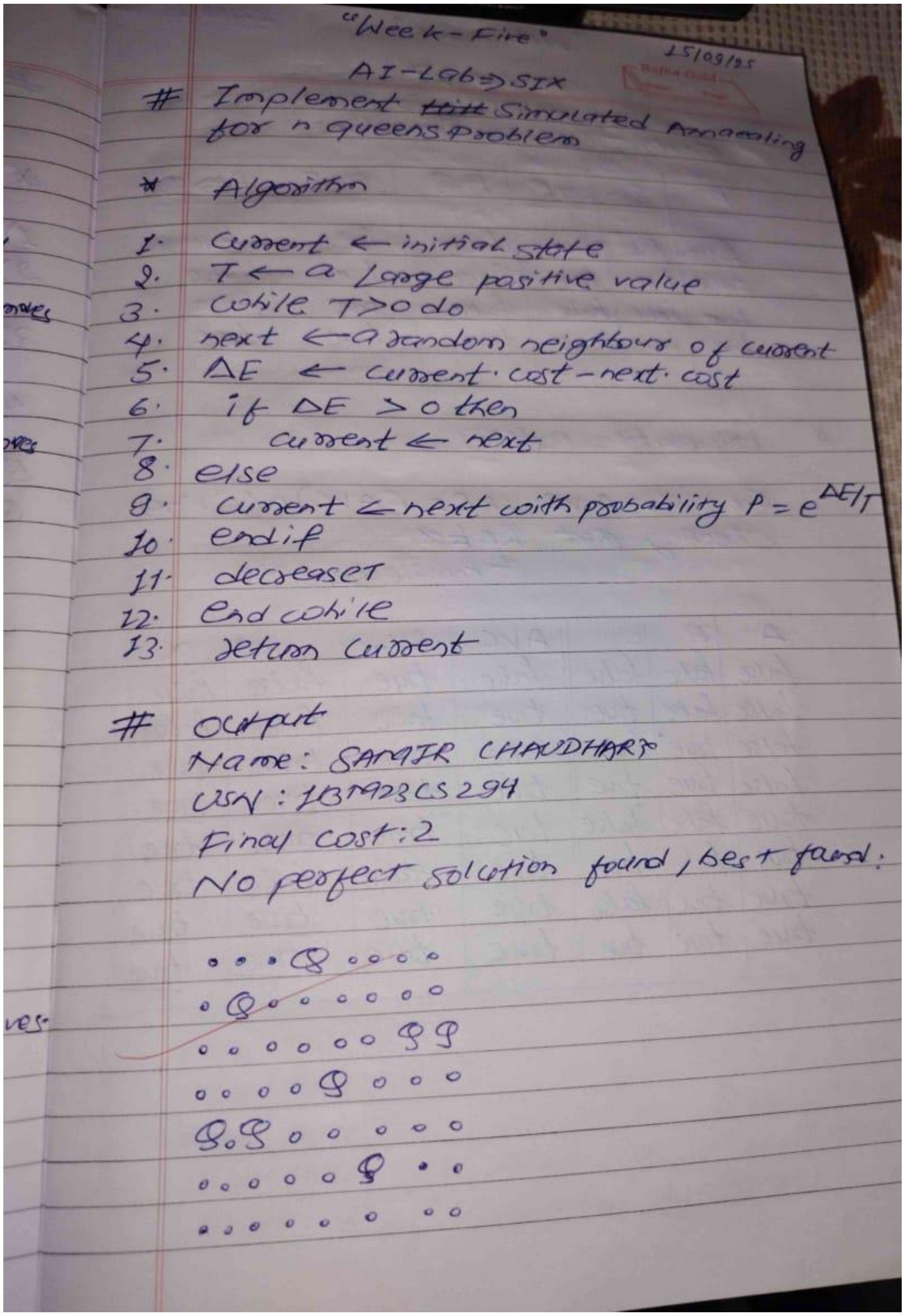
```
SAMIR CHAUDHARY

1BM23CS294



--- Case 1 ---

Step 0: State=[0, 1, 2, 3], Cost=6

Step 1: State=[1, 1, 2, 3], Cost=4

Step 2: State=[1, 0, 2, 3], Cost=2

Stuck at local minimum, no better moves.
```

```
--- Case 2 ---

Step 0: State=[3, 1, 2, 0], Cost=2

Stuck at local minimum, no better moves.


--- Case 3 ---

Step 0: State=[1, 3, 0, 2], Cost=0

Solution found!


--- Case 4 ---

Step 0: State=[2, 0, 3, 1], Cost=0

Solution found!


--- Case 5 ---

Step 0: State=[3, 2, 1, 0], Cost=6

Step 1: State=[0, 2, 1, 0], Cost=4

Step 2: State=[0, 3, 1, 0], Cost=2

Step 3: State=[0, 3, 1, 2], Cost=1

Stuck at local minimum, no better moves.


--- Case 6 ---

Step 0: State=[0, 2, 3, 1], Cost=1

Stuck at local minimum, no better moves.
```

Program-5

Simulated Annealing to Solve N-Queens problem

Algorithm:



"Week-Five"

15/09/25

AI-Lab ⇒ SIX

Implement ~~tiit~~ Simulated Annealing for n queens problem

* Algorithm

```
current ← initial state
T ← a Large positive value
while T>0 do
next ← a random neighbour of current
ΔE ← current·cost − next·cost
 if ΔE > 0 then
    current ← next
else
 current ← next with probability $P = e^{ΔE/T}$
endif
 decreaseT
End while
 return current
```

output

```
Name: SAMIR CHAUDHARY
USN : 1BM23CS294
Final cost:2
No perfect solution found, best found:

. . . Q . . . .
. Q . . . . . .
. . . . . . Q Q
. . . . Q . . .
Q . Q . . . . .
. . . . . Q . .
. . . . . . . .
```

Code:

Name: SAMIR CHAUDHARY

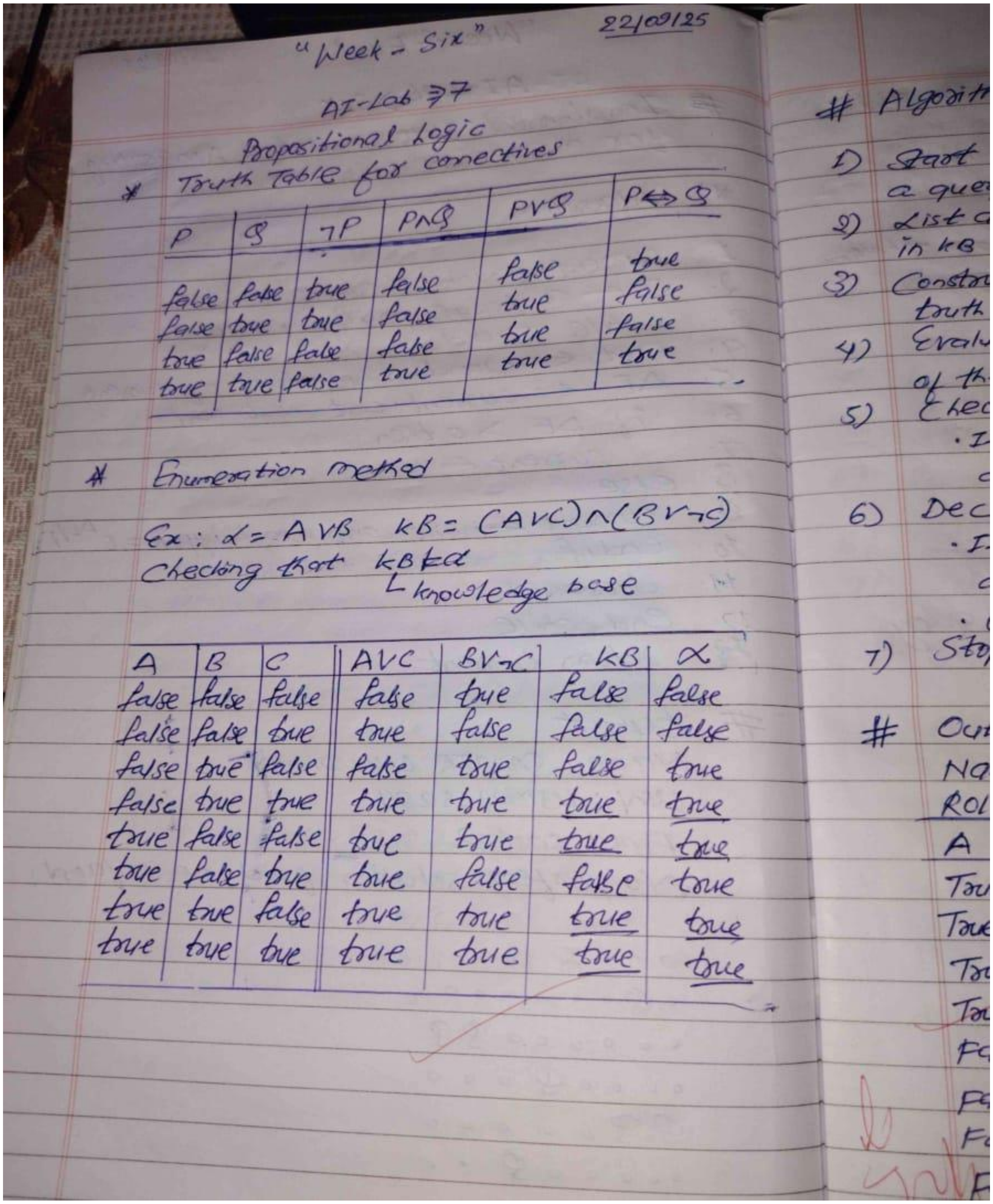
USN: 1BM23CS294

```
Final cost: 2

No perfect solution found, best found:

. . . Q . . . .

. Q . . . . . .

. . . . . . Q Q

. . . . Q . . .

Q . Q . . . . .

. . . . . . . .

. . . . . Q . .

. . . . . . . .
```

Program-6

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

Algorithm



"Week - Six" 22/09/25

AI-Lab #7

Propositional Logic

* Truth Table for conectives

P	Q	$\neg P$	$P \land Q$	$P \lor Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true
false	true	true	false	true	false
true	false	false	false	true	false
true	true	false	true	true	true

* Enumeration method

Ex: $\alpha = A \lor B$ $KB = (A \lor C) \land (B \lor \neg C)$

Checking that $KB \models \alpha$

L knowledge base

A	B	C	$A \lor C$	$B \lor \neg C$	KB	α
false	false	false	false	true	false	false
false	false	true	true	false	false	false
false	true	false	false	true	false	true
false	true	true	true	true	true	true
true	false	false	true	true	true	true
true	false	true	true	false	false	true
true	true	false	true	true	true	true
true	true	true	true	true	true	true

# Algorit

1) Start

a que

2) List c

in KB

3) Constr

truth

4) Evalu

of th

5) Chec

· I

c

6) Dec

· I

c

7) Sto

# Out

Na

ROl

A

Tru

True

Tru

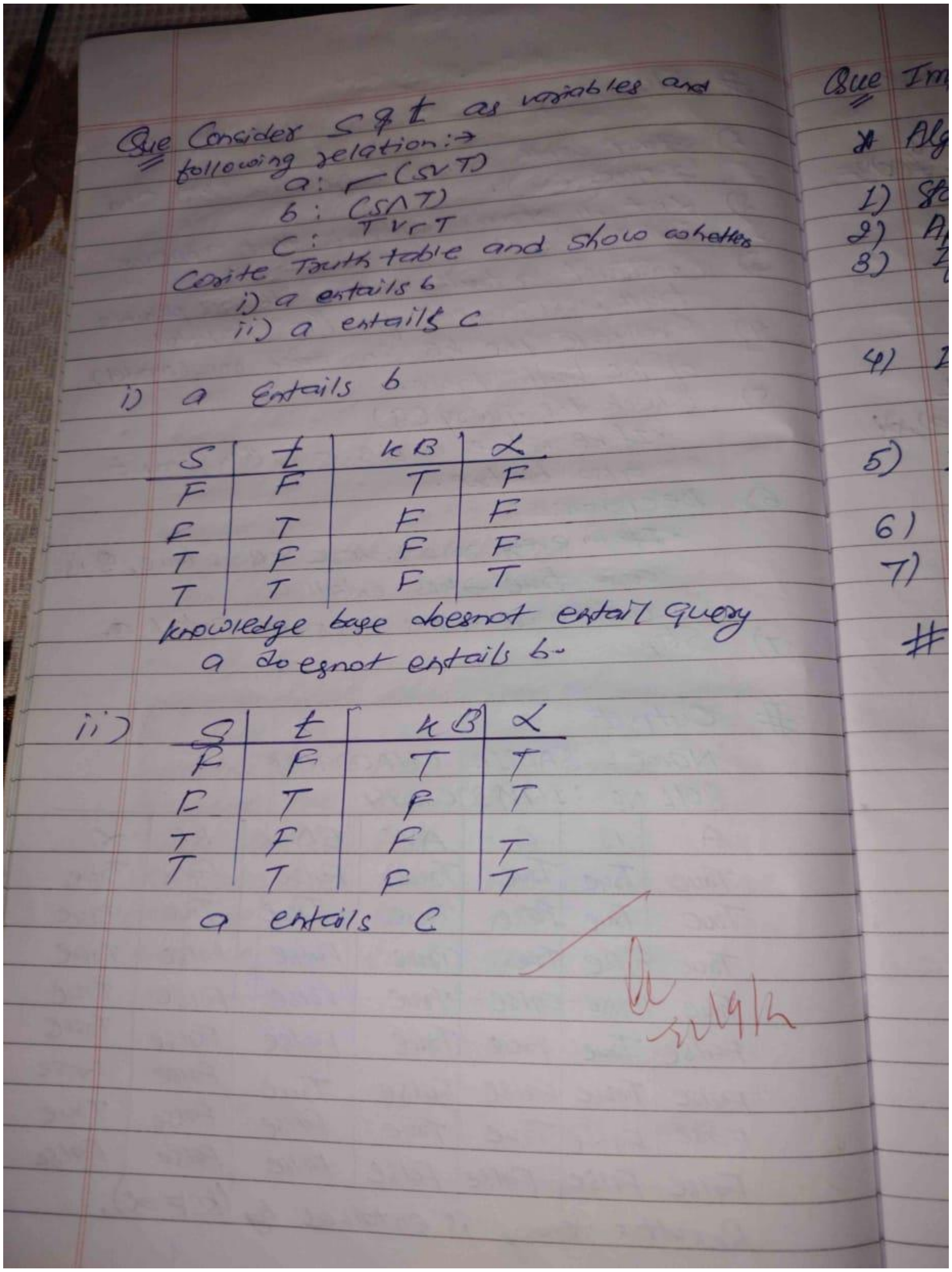
Tru

Fa

Fa

Fa

F



Que Consider S & t as variables and following relation:→

a: $\neg(S \vee T)$

b: $(S \wedge T)$

c: $T \vee \neg T$

Write Truth table and show whether

i) a entails b

ii) a entails c

i) a Entails b

S	t	KB	α
F	F	T	F
F	T	F	F
T	F	F	F
T	T	F	T

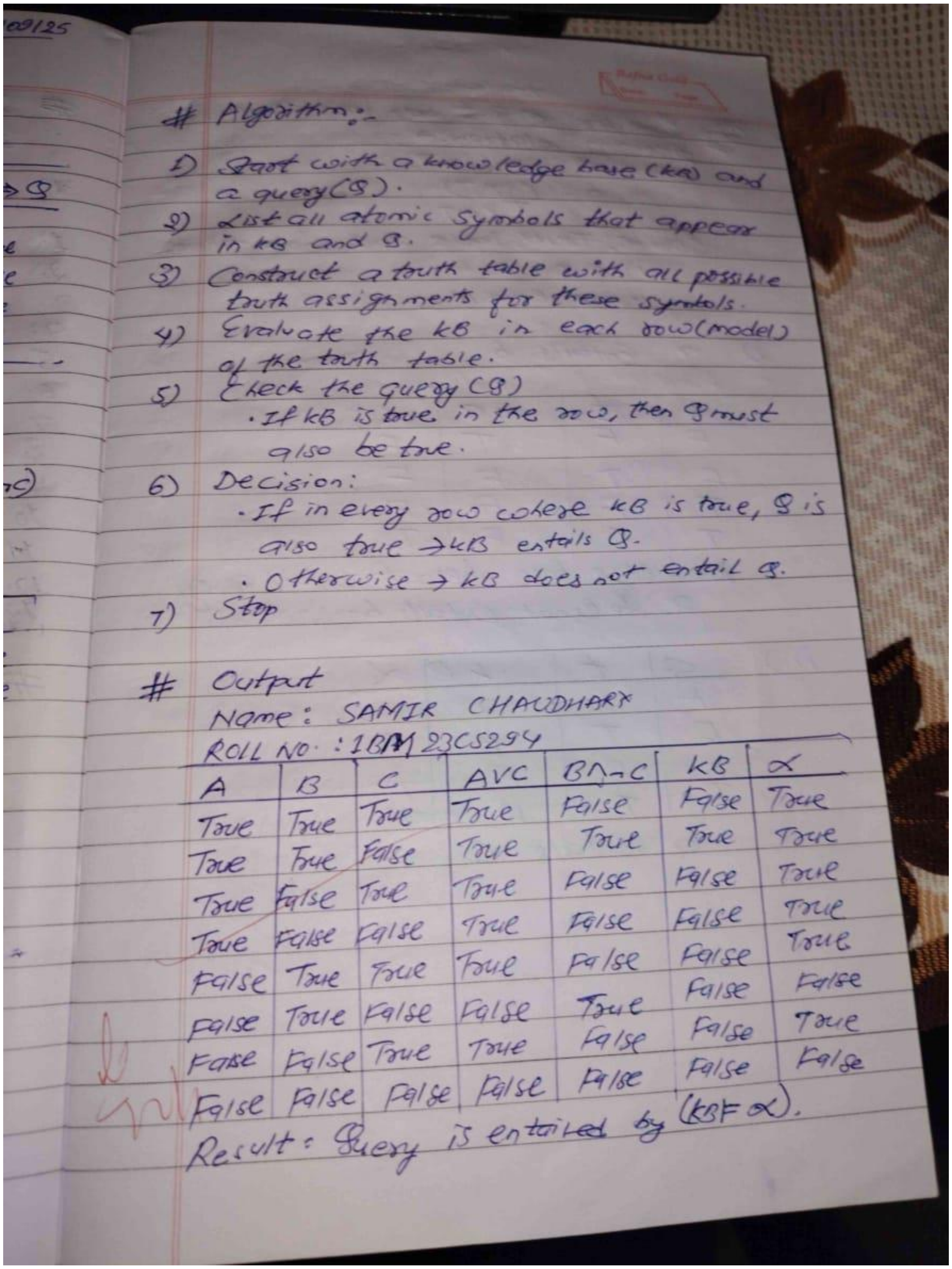
knowledge base doesnot entail query

a doesnot entails b.

ii)

S	t	KB	α
F	F	T	T
F	T	F	T
T	F	F	T
T	T	F	T

a entails c



Algorithm:-

1) Start with a knowledge base (KB) and a query (Q).
2) List all atomic symbols that appear in KB and Q.
3) Construct a truth table with all possible truth assignments for these symbols.
4) Evaluate the KB in each row (model) of the truth table.
5) Check the query (Q)
   - If KB is true in the row, then Q must also be true.
6) Decision:
   - If in every row where KB is true, Q is also true → KB entails Q.
   - Otherwise → KB does not entail Q.
7) Stop

Output

Name: SAMIR CHAUDHARY

ROLL NO.: 1BM23CS294

A	B	C	$A \vee C$	$B \wedge \neg C$	KB	α
True	True	True	True	False	False	True
True	True	False	True	True	True	True
True	False	True	True	False	False	True
True	False	False	True	False	False	True
False	True	True	True	False	False	False
False	True	False	False	True	False	True
Fase	False	True	True	False	False	False
False	False	False	False	False	False	

Result: Query is entailed by ($KB \models \alpha$).

Code:

from itertools import product

```
def evaluate(expr, model): if expr in model: return model[expr] elif "→" in expr: p, q = expr.split("→") return (not evaluate(p.strip(), model)) or evaluate(q.strip(), model) elif "&" in expr: p, q = expr.split("&") return evaluate(p.strip(), model) and evaluate(q.strip(), model) elif "|" in expr: p, q = expr.split("|") return evaluate(p.strip(), model) or evaluate(q.strip(), model) elif "~" in expr: return not evaluate(expr[1:].strip(), model) else: return False

def print_custom_truth_table(): print("Name: SAMIR CHAUDHARY") print("Roll No.: 1BM23CS294\n")

symbols = ['A', 'B', 'C']
KB_expr = "A∨C & B∧~C"
query = "A∨C"  # α
header = symbols + ["A∨C", "B∧¬C", "KB", "α"]

print(" | ".join(f"{h:>5}" for h in header))
print("-" * (7 * len(header)))

entails = True
for values in product([True, False], repeat=len(symbols)):
    model = dict(zip(symbols, values))
    avc = evaluate("A|C", model)
    bnc = evaluate("B&~C", model)
    kb_val = avc and bnc
    alpha_val = evaluate(query.replace("∨", "|").replace("∧", "&").replace("¬", "~"),
model)
    row = [model[s] for s in symbols] + [avc, bnc, kb_val, alpha_val]
    print(" | ".join(f"{str(v):>5}" for v in row))
    if kb_val and not alpha_val:
        entails = False

print("\nResult:")
if entails:
    print("Query is entailed by KB (KB ⊨ α).")
else:
    print("Query is NOT entailed by KB.")


print_custom_truth_table()
```

Output:

```
Name: SAMIR CHAUDHARY

Roll No.: 1BM23CS294


    A |    B |    C |  A∨C | B∧¬C |   KB |    α

-------------------------------------------------
```

```
 True |  True |  True |  True | False | False |  True
 True |  True | False |  True |  True |  True |  True
 True | False |  True |  True | False | False |  True
 True | False | False |  True | False | False |  True
False |  True |  True |  True | False | False |  True
False |  True | False | False |  True | False | False
False | False |  True |  True | False | False |  True
False | False | False | False | False | False | False
```

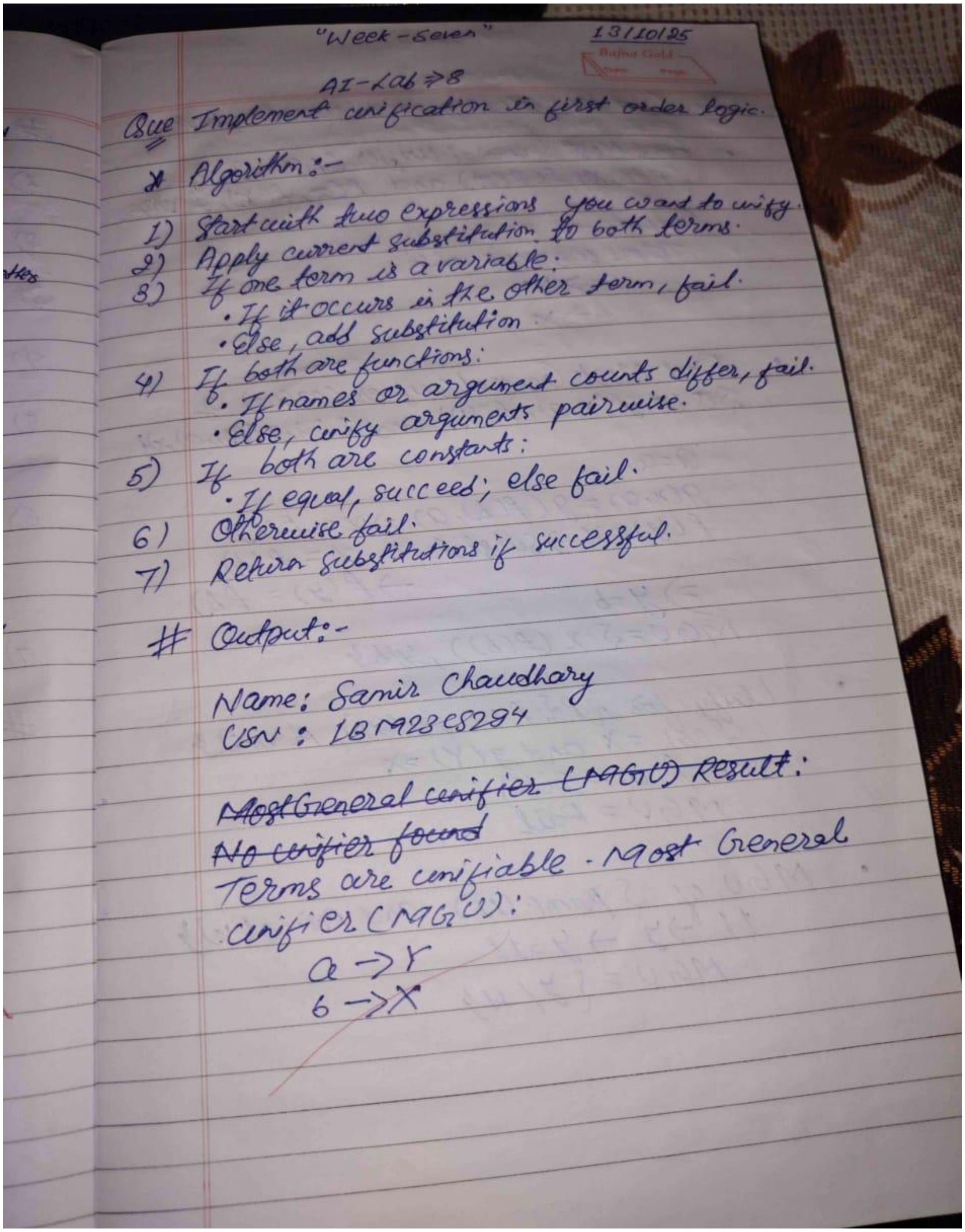
Result:

Query is entailed by KB (KB ⊨ α).

Program-7

Implement unification in first order logic

Algorithm



"Week-Seven" 13/10/25

AI-Lab ⇒ 8

Que Implement unification in first order logic.

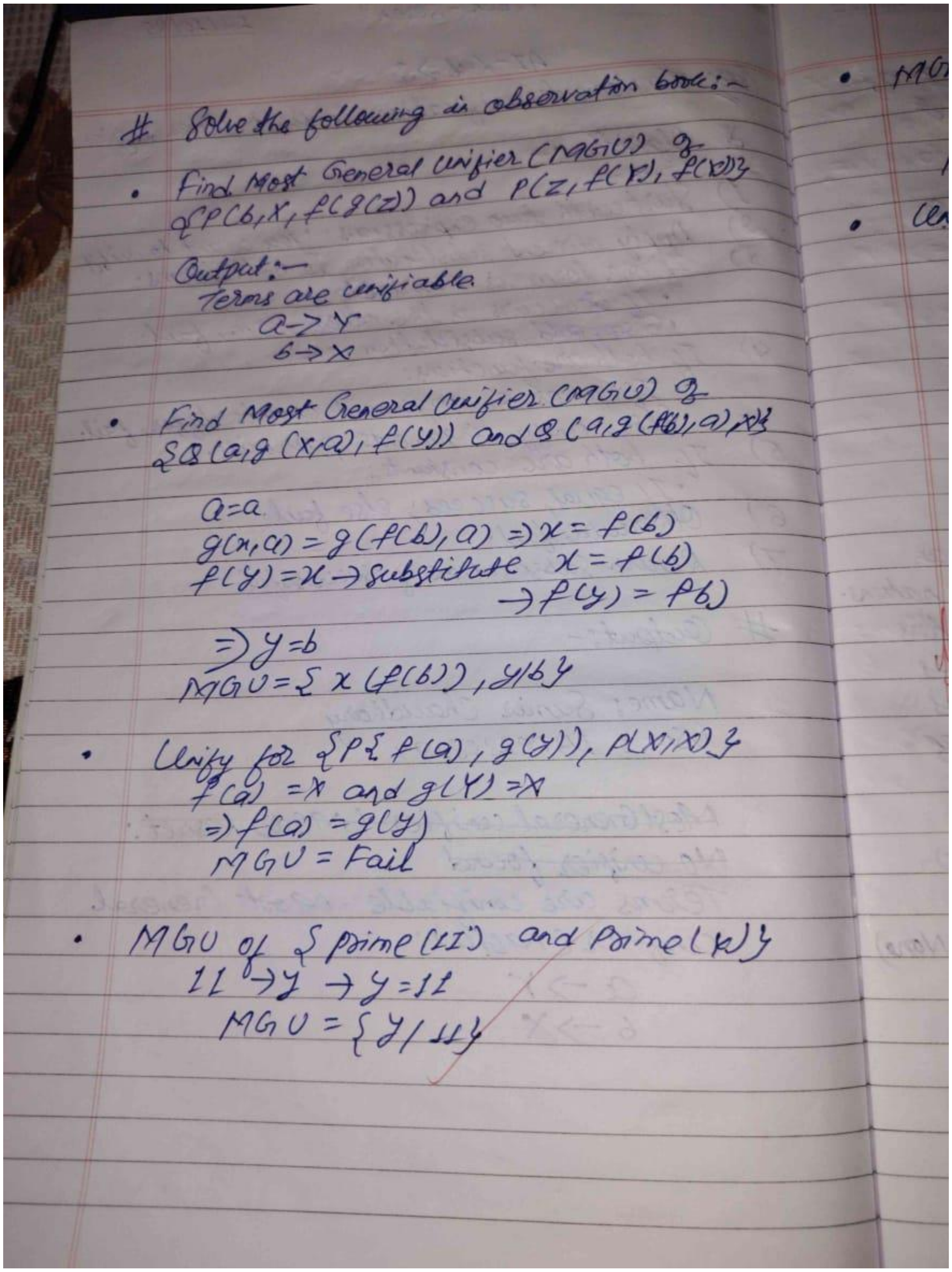
* Algorithm:-

1) Start with two expressions you want to unify.
2) Apply current substitution to both terms.
3) If one term is a variable:
   - If it occurs in the other term, fail.
   - Else, add substitution
4) If both are functions:
   - If names or argument counts differ, fail.
   - Else, unify arguments pairwise.
5) If both are constants:
   - If equal, succeed; else fail.
6) Otherwise fail.
7) Return substitutions if successful.

Output:-

Name: Samir Chaudhary
USN : 1BM23CS294

~~Most General unifier (MGU) Result~~:
~~No unifier found~~
Terms are unifiable. Most General unifier (MGU):
a -> Y
b -> X



Solve the following in observation book:-

- Find Most General Unifier (MGU) of $\{P(b, X, f(g(Z))$ and $P(Z, f(Y), f(Y))\}$

Output:-
Terms are unifiable
$a \rightarrow Y$
$b \rightarrow X$

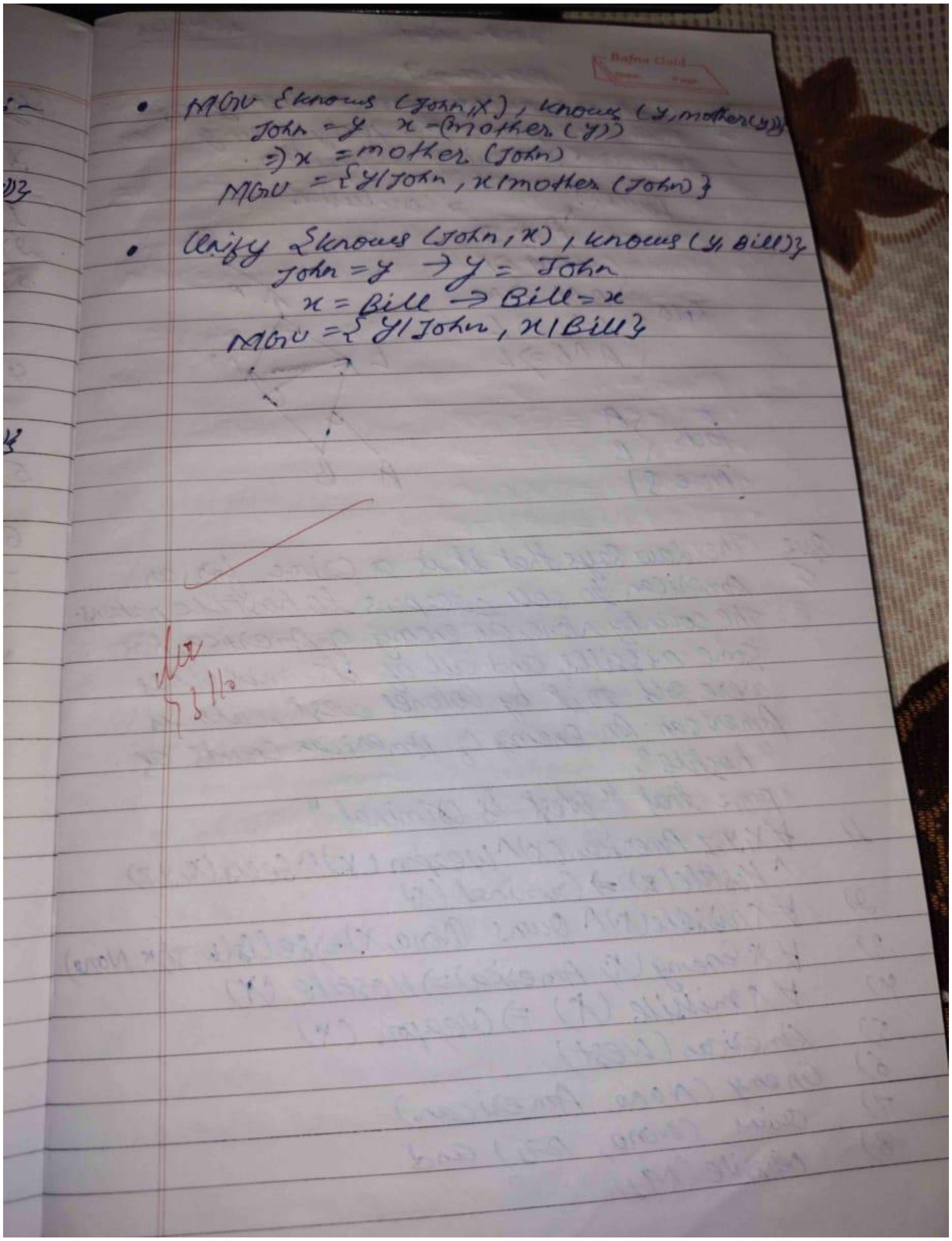
- Find Most General Unifier (MGU) of $\{Q(a, g(X, a), f(Y))$ and $Q(a, g(f(b), a), X)\}$

$a = a$
$g(x, a) = g(f(b), a) \Rightarrow x = f(b)$
$f(y) = x \rightarrow$ substitute $x = f(b)$
$\rightarrow f(y) = f(b)$

$\Rightarrow y = b$
$MGU = \{x(f(b)), y/b\}$

- Unify for $\{P\{f(a), g(y)), P(X, X)\}$
$f(a) = X$ and $g(Y) = X$
$\Rightarrow f(a) = g(y)$
MGU = Fail

- MGU of $\{prime(11)$ and $prime(y)\}$
$11 \rightarrow y \rightarrow y = 11$
$MGU = \{y/11\}$



- MGU {knows (John, X), knows (y, mother(y))}
  John = y x = (mother (y))
  ⇒ x = mother (John)
  MGU = {y/John, x/mother (John)}

- Unify {knows (John, x), knows (y, Bill)}
  John = y → y = John
  x = Bill → Bill = x
  MGU = {y/John, x/Bill}

Code:

```
from collections import deque
```

class Term:

```
def __init__(self, name, args=None):
    self.name = name
    self.args = args or []

def is_variable(self):
    # Variables: lowercase single letters or strings starting with lowercase letters
without args
    return self.args == [] and self.name.islower()

def is_constant(self):
    # Constants: lowercase letters that are NOT variables (e.g., single lowercase
letters but treated as constants)
    # For simplicity, consider constants as names that are not variables or functions
    return self.args == [] and not self.is_variable()

def __eq__(self, other):
    if not isinstance(other, Term):
        return False
    return self.name == other.name and self.args == other.args

def __hash__(self):
    return hash((self.name, tuple(self.args)))

def __repr__(self):
    if self.args:
        return f"{self.name}({', '.join(map(str, self.args))})"
    else:
        return self.name
```

def occurs_check(var, term, subst): if var == term: return True if term.is_variable() and term.name in subst: return occurs_check(var, subst[term.name], subst) if term.args: return any(occurs_check(var, arg, subst) for arg in term.args) return False

def unify(t1, t2, subst=None): if subst is None: subst = {}

```
stack = deque([(t1, t2)])

while stack:
    term1, term2 = stack.pop()

    # Apply substitutions
    while term1.is_variable() and term1.name in subst:
        term1 = subst[term1.name]
    while term2.is_variable() and term2.name in subst:
        term2 = subst[term2.name]

    if term1 == term2:
        continue
    if term1.is_variable():
        if occurs_check(term1, term2, subst):
            return None
        subst[term1.name] = term2

    elif term2.is_variable():
        if occurs_check(term2, term1, subst):
            return None
        subst[term2.name] = term1

    elif term1.name == term2.name and len(term1.args) == len(term2.args):
        for a1, a2 in zip(term1.args, term2.args):
            stack.append((a1, a2))
    else:
        # function names differ or different arity => cannot unify
        return None

return subst
```

```
def make_term(s): s = s.strip() if '(' not in s: return Term(s) i = s.index('(') name = s[:i] args_str = s[i+1:-1] args = []
balance = 0 current = " for ch in args_str: if ch == ',' and balance == 0: args.append(make_term(current)) current = "
else: if ch == '(': balance += 1 elif ch == ')': balance -= 1 current += ch if current: args.append(make_term(current))
return Term(name, args)
```

```
t1 = make_term("p(X, a)") t2 = make_term("p(b, Y)") print("Name: Samir Chaudhary") print("USN:
1BM23CS294\n")
```

```
result = unify(t1, t2)
```

```
if result is None: print("No unifier found for the given terms.") else: print("Terms are unifiable. Most General Unifier
(MGU):") for var, val in result.items(): print(f"{var} -> {val}")
```

Output:

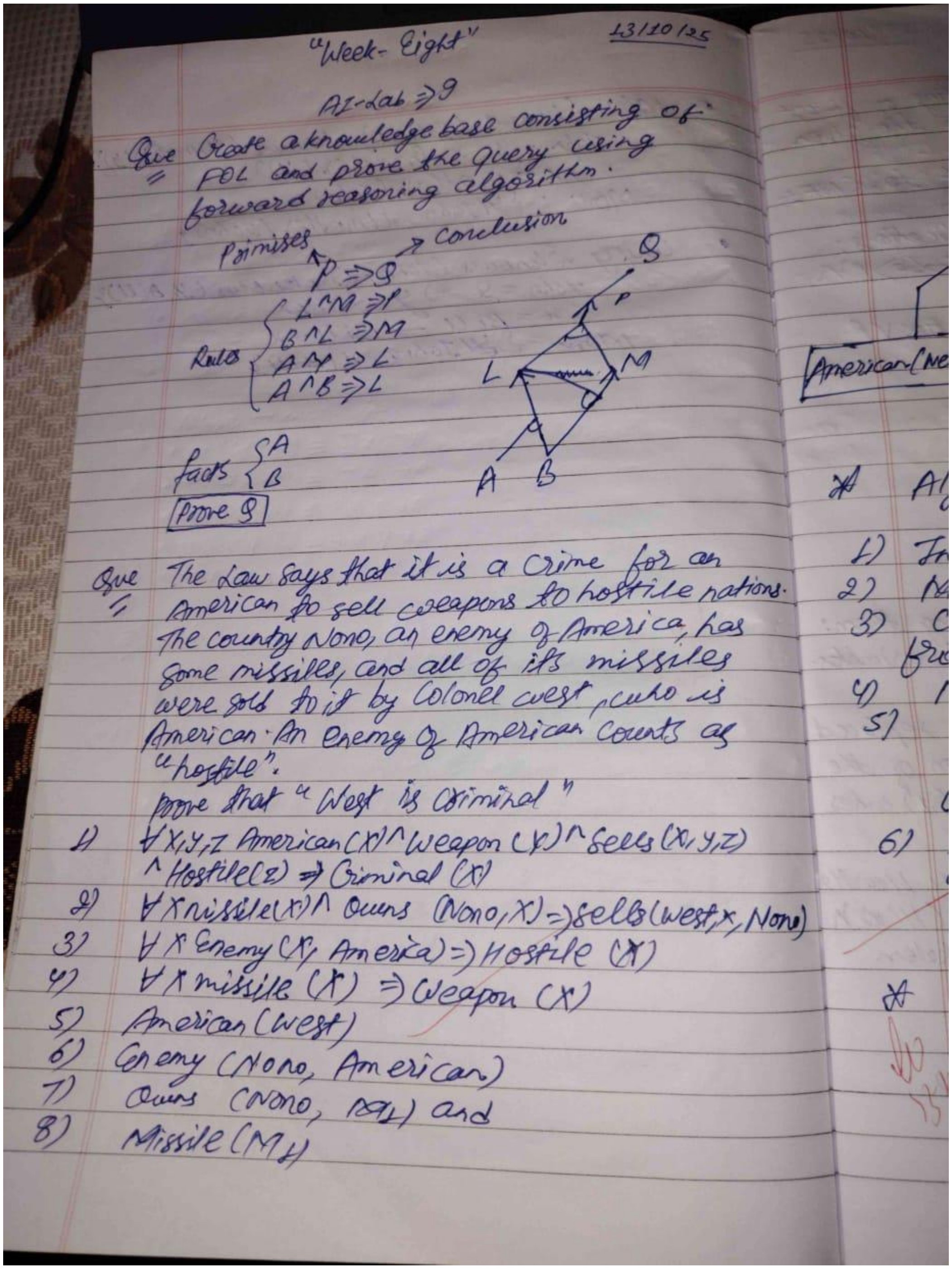
```
Name: Samir Chaudhary

USN: 1BM23CS294



Terms are unifiable. Most General Unifier (MGU):

a -> Y
```

```
b -> X
```

Program-8

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

Algorithm



"Week- Eight" 13/10/25

AI-Lab ⇒ 9

Que Create a knowledge base consisting of FOL and prove the query using forward reasoning algorithm.

Primises ← $P \Rightarrow Q$ → Conclusion

Rules
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge M \Rightarrow L$
$A \wedge B \Rightarrow L$

facts
A
B

Prove Q

Que The Law says that it is a Crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel west, who is American. An Enemy of American counts as "hostile".

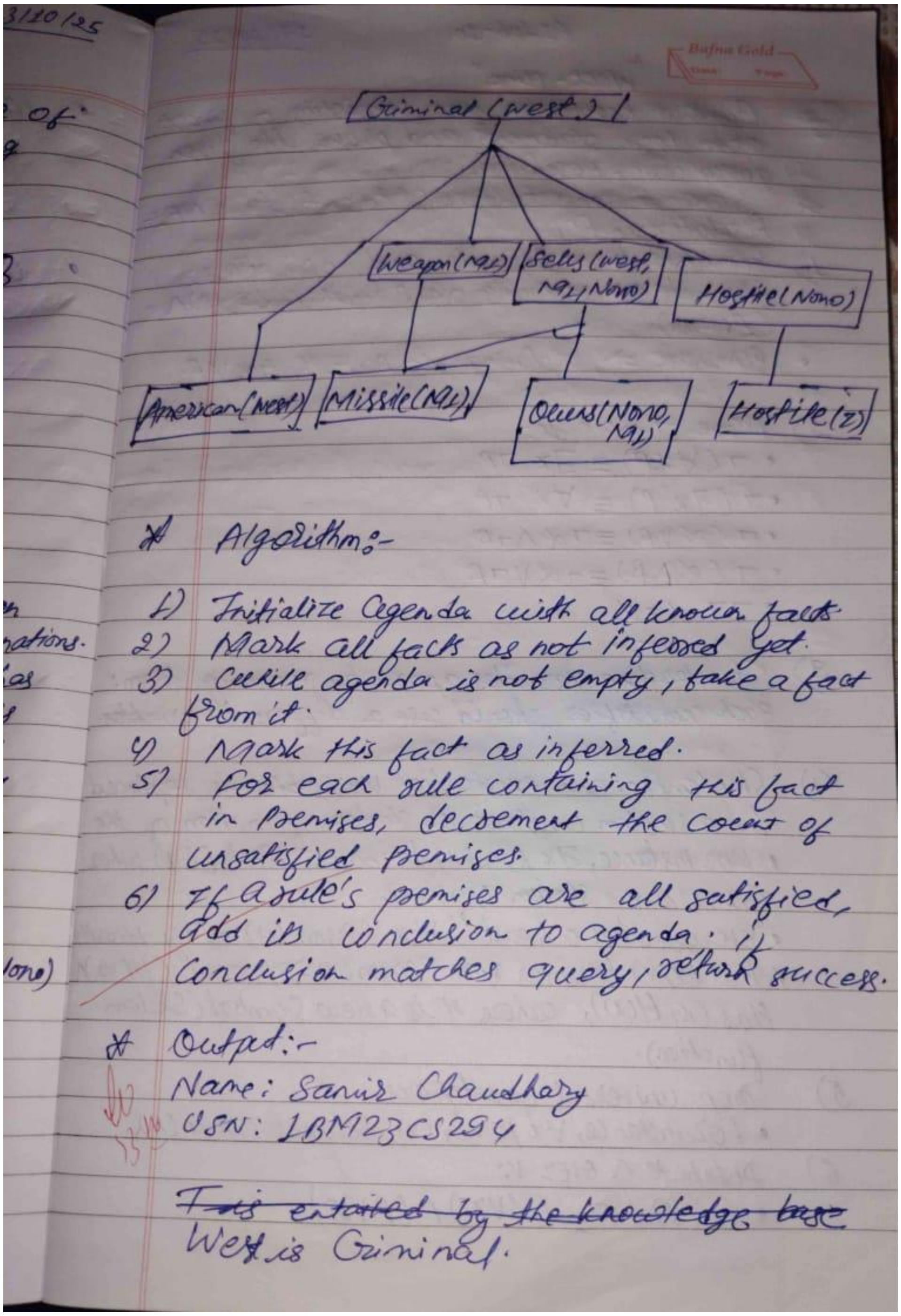
prove that "West is Criminal"

1) $\forall x,y,z$ American(x) ∧ Weapon (y) ∧ Sells (x, y, z) ∧ Hostile(z) ⇒ Criminal (x)
2) $\forall x$ missile(x) ∧ Owns (Nono, x) ⇒ sells(west, x, Nono)
3) $\forall x$ Enemy (x, America) ⇒ Hostile (x)
4) $\forall x$ missile (x) ⇒ Weapon (x)
5) American (West)
6) Enemy (Nono, American)
7) Owns (Nono, M1) and
8) Missile (M1)

American(Ne

Al

1) In
2) N
3) C
fro
4) I
5)
6)


Criminal(west)
Weapon(M1)
Sells(west, M1, Nono)
Hostile(Nono)
American(West)
Missile(M1)
Owns(Nono, M1)
Hostile(z)

3/10/25

# Algorithm:-

1) Initialize agenda with all known facts.
2) Mark all facts as not inferred yet.
3) While agenda is not empty, take a fact from it.
4) Mark this fact as inferred.
5) For each rule containing this fact in premises, decrement the count of unsatisfied premises.
6) If a rule's premises are all satisfied, add its conclusion to agenda. if conclusion matches query, return success.

# Output:-

Name: Samir Chaudhary
USN: 1BM23CS294

~~Is entailed by the knowledge base~~
West is Criminal.

Code:

```
print("Name: Samir Chaudhary")
print("USN: 1BM23CS294\n")


def forward_chaining(KB, query):
    inferred = {symbol: False for symbol in KB['symbols']}
    agenda = list(KB['facts'])
    count = {rule: len(KB['rules'][rule]['premises']) for rule in KB['rules']}

    while agenda:
        p = agenda.pop(0)
        if p == query:
            return True
        if not inferred[p]:
            inferred[p] = True
            for rule in KB['rules']:
                if p in KB['rules'][rule]['premises']:
                    count[rule] -= 1
                    if count[rule] == 0:
                        agenda.append(KB['rules'][rule]['conclusion'])
    return False

# Knowledge base facts and rules based on the problem

KB = {
    'symbols': ['American(West)', 'Enemy(Nono)', 'Hostile(Nono)', 'Missiles(Nono)',
                'Sells(West, Missiles, Nono)', 'Crime(West)'],
    'facts': [
        'American(West)',
        'Enemy(Nono)',

        'Missiles(Nono)',

        'Sells(West, Missiles, Nono)'

    ],

    'rules': {

        # Enemy is hostile

        'rule1': {'premises': ['Enemy(Nono)'], 'conclusion': 'Hostile(Nono)'},


        # Law: If American sells weapons to hostile nation, then criminal

        'rule2': {'premises': ['American(West)', 'Hostile(Nono)', 'Sells(West, Missiles, Nono)'], 'conclusion': 'Crime(West)'}

    }

}


query = 'Crime(West)'


if forward_chaining(KB, query):

    print("West is criminal.")

else:

    print("West is NOT criminal.")
```

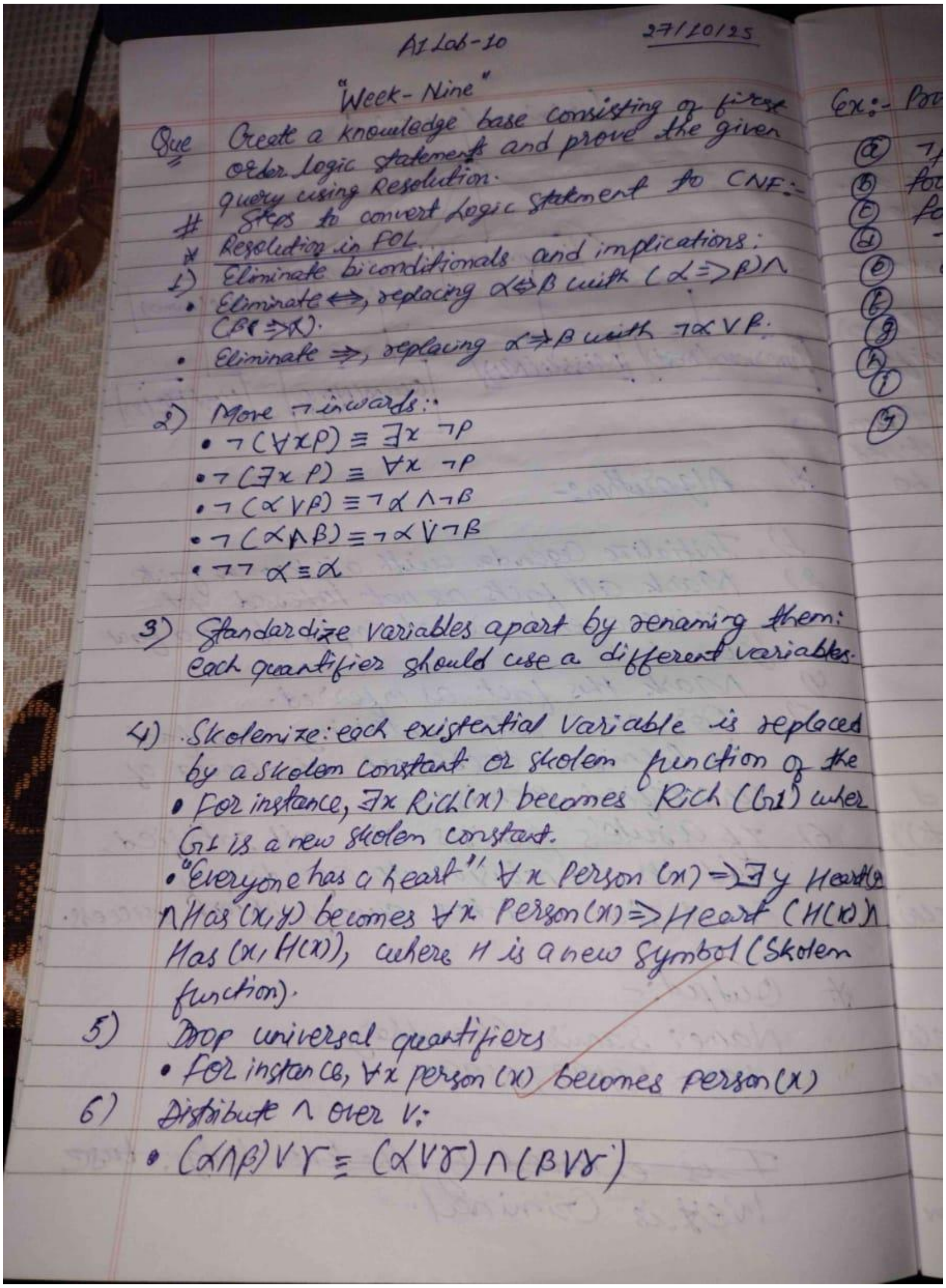
Output:

```
Name: Samir Chaudhary

USN: 1BM23CS294


West is criminal.
```

Program-9

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

Algorithm:



AI Lab-10 27/10/25

"Week-Nine"

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

Steps to convert Logic statment to CNF:-

* Resolution in FOL

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) More $\neg$ 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 variables.

4) Skolemize: each existential variable is replaced by a Skolem constant or skolem function of the

- For instance, $\exists x$ Rich(x) becomes Rich$(G1)$ wher G1 is a new skolem constant.
- "Everyone has a heart" $\forall x$ Person$(x) \Rightarrow \exists y$ Heart$(y) \wedge$ Has(x,y) becomes $\forall x$ Person$(x) \Rightarrow$ Heart$(H(x)) \wedge$ Has$(x, H(x))$, where H is a new symbol (Skolem function).

5) Drop universal quantifiers

- For instance, $\forall x$ person(x) becomes person(x)

6) Distribute $\wedge$ over $\vee$:

- $(\alpha \wedge \beta) \vee \gamma \equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$

Ex:- Pro

ⓐ ¬

ⓑ fo

ⓒ fo

ⓓ

ⓔ

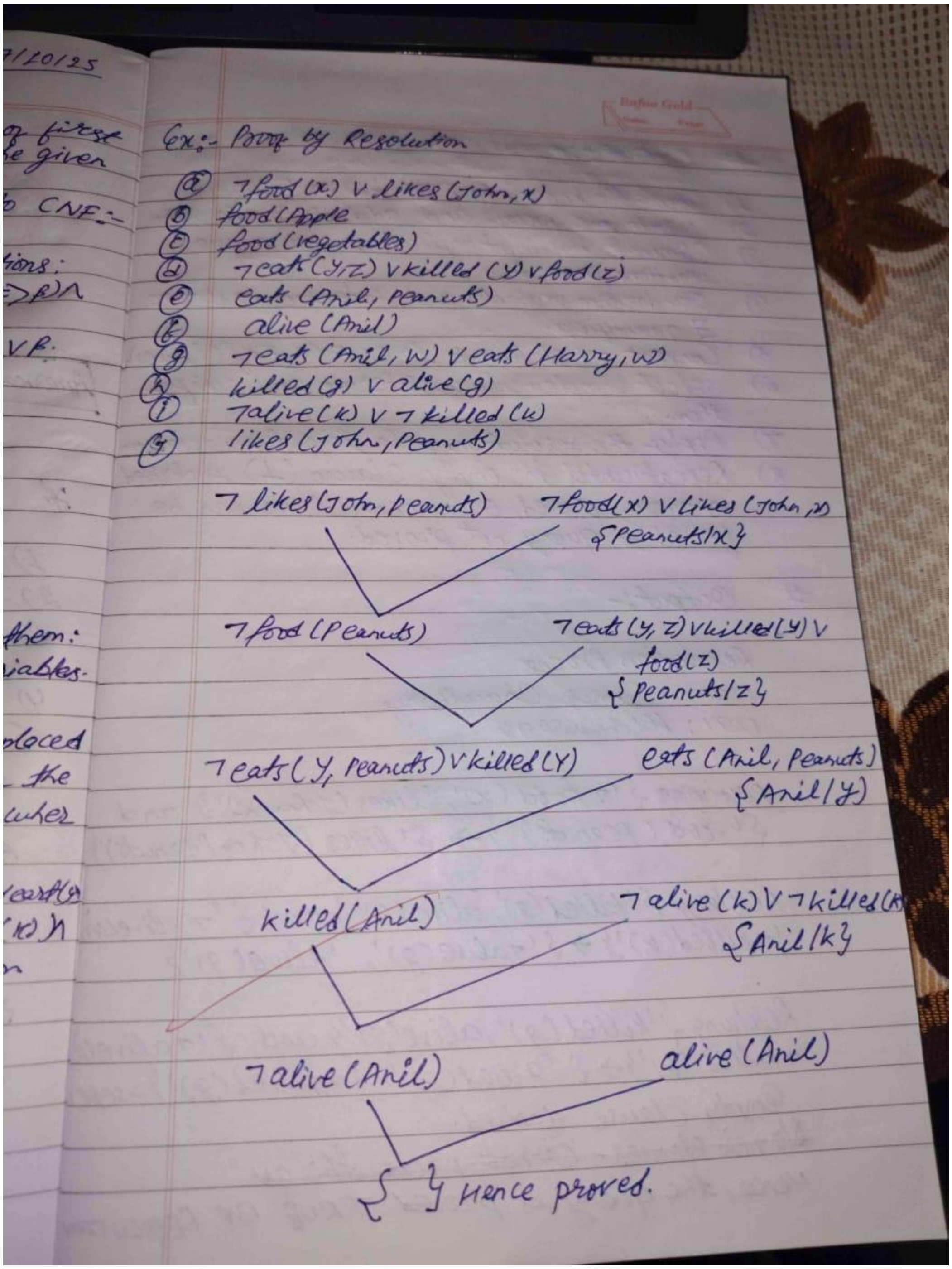
ⓕ

ⓖ

ⓗ

ⓘ

ⓙ



/1/2025

of first
e given

o CNF:-

ions:
=>β)∧

∨β.

them:
iables.

placed
the
wher

earth(
(k))∧
n

Ex:- Proof by Resolution

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

¬likes(John, peanuts)   ¬food(x) ∨ likes(John, x)
{Peanuts/x}

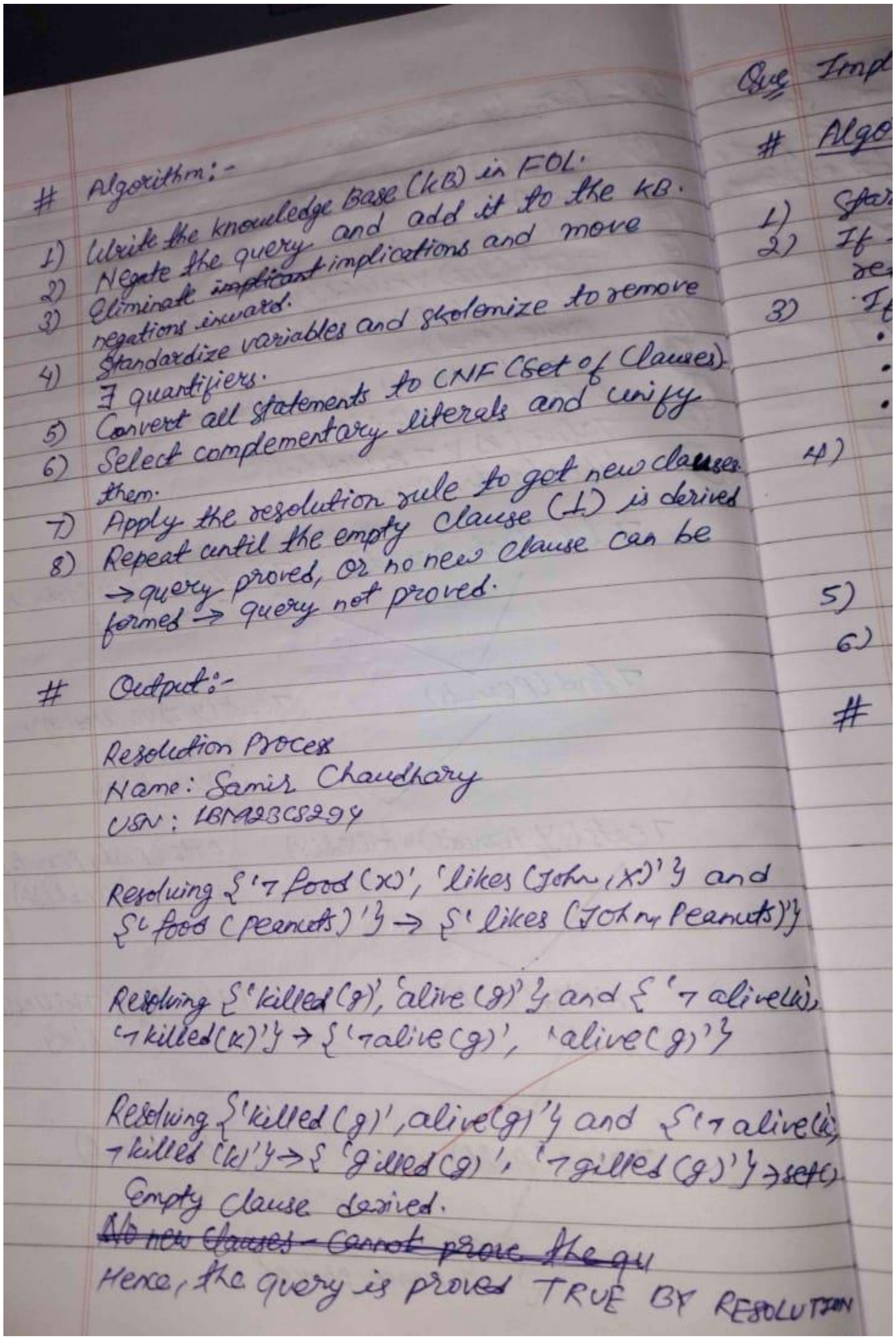
¬food(Peanuts)   ¬eats(y, z) ∨ killed(y) ∨ food(z)
{Peanuts/z}

¬eats(y, Peanuts) ∨ killed(Y)   eats(Anil, Peanuts)
{Anil/y}

killed(Anil)   ¬alive(k) ∨ ¬killed(k)
{Anil/k}

¬alive(Anil)   alive(Anil)

{ } Hence proved.



Algorithm:-

1) Write the knowledge Base (KB) in FOL.
2) Negate the query and add it to the KB.
3) Eliminate ~~implicant~~ implications and move negations inward.
4) Standardize variables and skolemize to remove ∃ quantifiers.
5) Convert all statements to CNF (Set of Clauses)
6) Select complementary literals and unify them.
7) Apply the resolution rule to get new clauses.
8) Repeat until the empty clause (⊥) is derived → query proved, or no new clause can be formed → query not proved.

Output:-

Resolution Process
Name: Samir Chaudhary
USN: 1BM23CS294

Resolving {'¬food(x)', 'likes(John,(x))'} and {'food(peanuts)'} → {'likes(John, Peanuts)'}

Resolving {'killed(g)', 'alive(g)'} and {'¬alive(k)', '¬killed(k)'} → {'¬alive(g)', 'alive(g)'}

Resolving {'killed(g)', alive(g)'} and {'¬alive(k), ¬killed (k)'} → {'gilled(g)', '¬gilled(g)'} → set()
Empty clause derived.
~~No new Clauses - Cannot prove the qu~~
Hence, the query is proved TRUE BY RESOLUTION

Que Impl

Algo

1) Star
2) If
re
3) If
•
•
•
4)
5)
6)
#

Code:

def unify(a, b): if a == b: return {} if '(' in a and '(' in b: if a.split('(')[0] == b.split('(')[0]: a_args = a[a.find('(')+1:-1].split(',') b_args = b[b.find('(')+1:-1].split(',') return {a_args[0]: b_args[0]} if a_args[0].islower() else {} return {}

```
def substitute(clause, subs): new_clause = set() for lit in clause: for var, val in subs.items(): lit = lit.replace(var, val) new_clause.add(lit) return new_clause

def resolve(ci, cj): resolvents = [] for di in ci: for dj in cj: if di.startswith("¬") and not dj.startswith("¬") and di[1:].split("(")[0] == dj.split("(")[0]: subs = unify(di[1:], dj) elif dj.startswith("¬") and not di.startswith("¬") and dj[1:].split("(")[0] == di.split("(")[0]: subs = unify(dj[1:], di) else: continue new_clause = substitute((ci - {di}) | (cj - {dj}), subs) resolvents.append(new_clause) return resolvents

KB = [ {"¬food(x)", "likes(John,x)"}, {"food(Peanuts)"}, {"killed(g)", "alive(g)"}, {"¬alive(k)", "¬killed(k)"} ]

clauses = [set(c) for c in KB] new = set()

print("Resolution Process") print("NAME: Samir Chaudhary") print("USN: 1BM23CS294\n")

found = False while True: for i in range(len(clauses)): for j in range(i + 1, len(clauses)): resolvents = resolve(clauses[i], clauses[j]) for res in resolvents: print(f"Resolving {clauses[i]} and {clauses[j]} → {res}") if not res: print("\n Empty clause derived.") print("Hence, the query is PROVED TRUE by Resolution.") found = True break if found: break if found: break if found: break if new.issubset(set(map(frozenset, clauses))): print("\nNo new clauses — cannot prove the query.") break for c in new: if set(c) not in clauses: clauses.append(set(c))
```

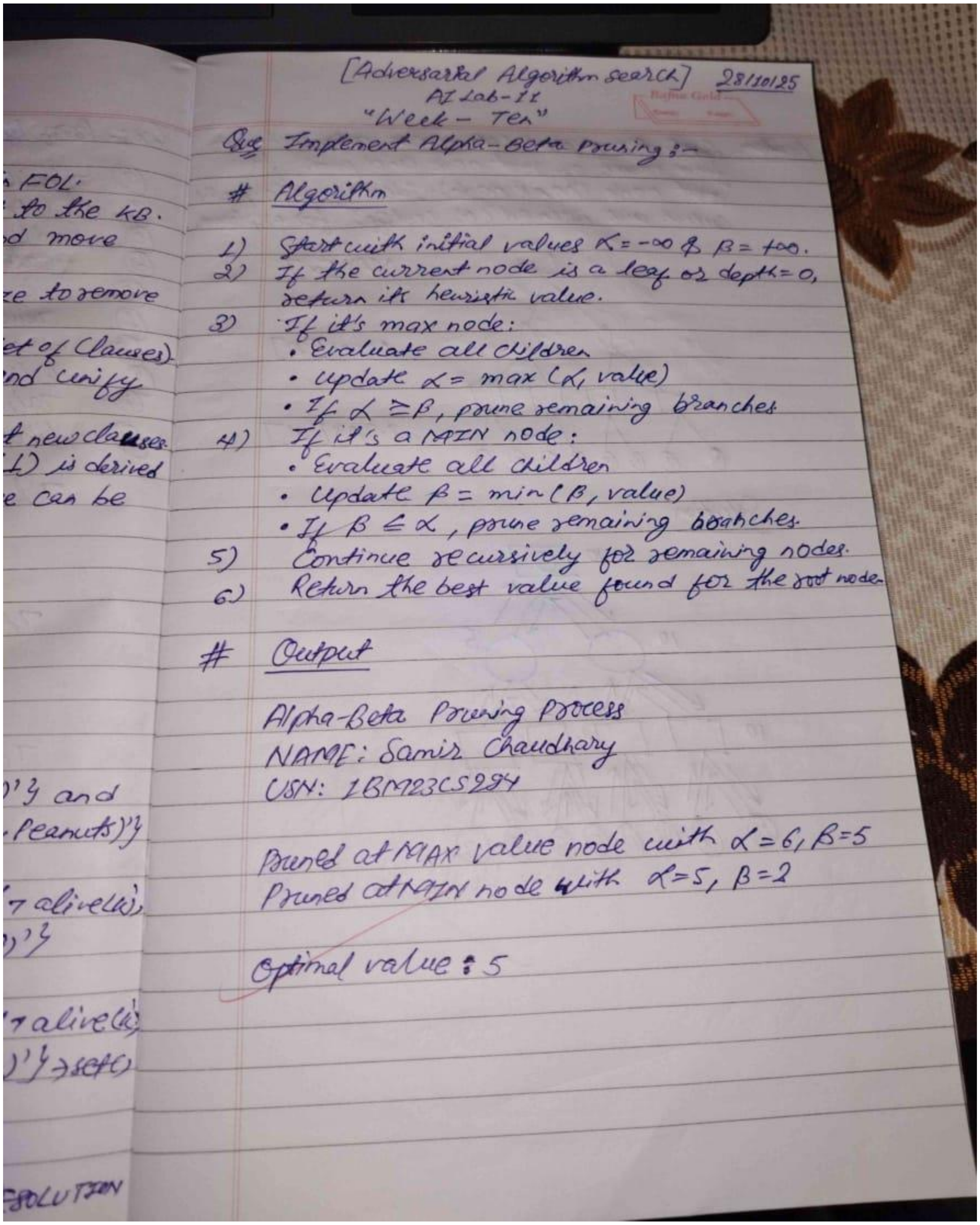
Output:

```
Resolution Process

NAME: Samir Chaudhary

USN: 1BM23CS294


Resolving {'¬food(x)', 'likes(John,x)'} and {'food(Peanuts)'} → {'likes(John,Peanuts)'}

Resolving {'killed(g)', 'alive(g)'} and {'¬alive(k)', '¬killed(k)'} → set()


Empty clause derived.

Hence, the query is PROVED TRUE by Resolution.
```

Program-10

Implement Alpha-Beta Pruning.

Algorithm:



FOL.
to the KB.
d move

ze to remove

et of Clauses)
nd unify

t new clauses
1) is derived
e can be

)'} and
Peanuts)'}

(¬alive(k),
)'}

(¬alive(k)
)'} → set()

SOLUTION

[Adversarial Algorithm search] 28/10/25
AI Lab-II
"Week - Ten"

Qs Implement Alpha-Beta Pruning :-

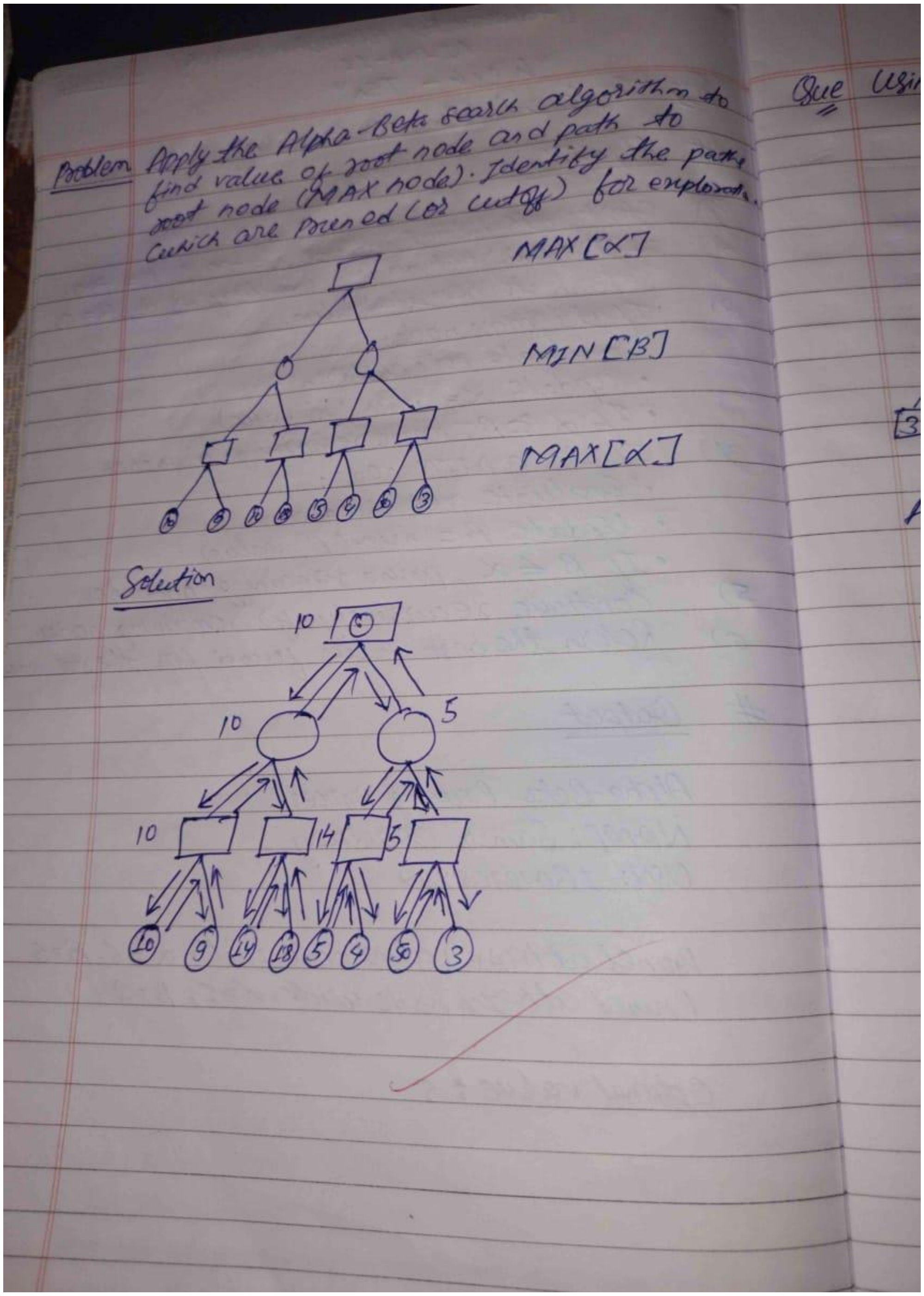
Algorithm

```
1) Start with initial values $\alpha = -\infty$ & $\beta = +\infty$.
2) If the current node is a leaf or depth = 0,
   return its heuristic value.
3) If it's max node:
   • Evaluate all children
   • update $\alpha = max(\alpha, value)$
   • If $\alpha \geq \beta$, prune remaining branches
4) If it's a MIN node:
   • Evaluate all children
   • Update $\beta = min(\beta, value)$
   • If $\beta \leq \alpha$, prune remaining branches.
5) Continue recursively for remaining nodes.
6) Return the best value found for the root node.
```

Output

```
Alpha-Beta Pruning Process
NAME: Samir Chaudhary
USN: 1BM23CS284

Pruned at MAX value node with α=6, β=5
Pruned at MIN node with α=5, β=2

Optimal value : 5
```

58



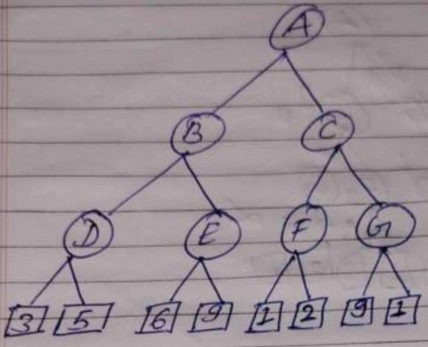
Problem Apply the Alpha-Beta search algorithm to find value of root node and path to root node (MAX node). Identify the paths which are pruned (or cutoff) for exploration.

Solution

Que Usi

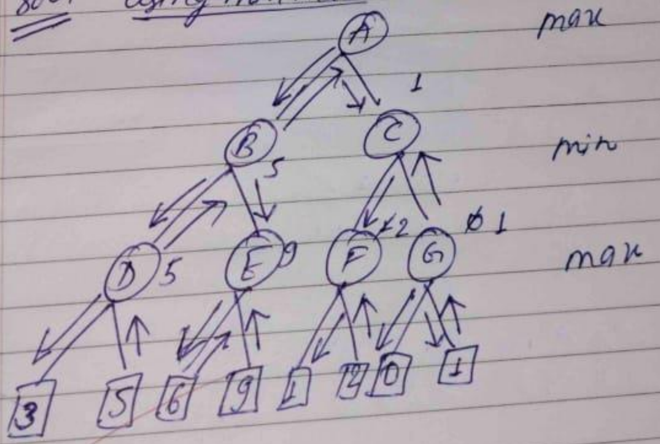
ithm to
to
the paths
explorat

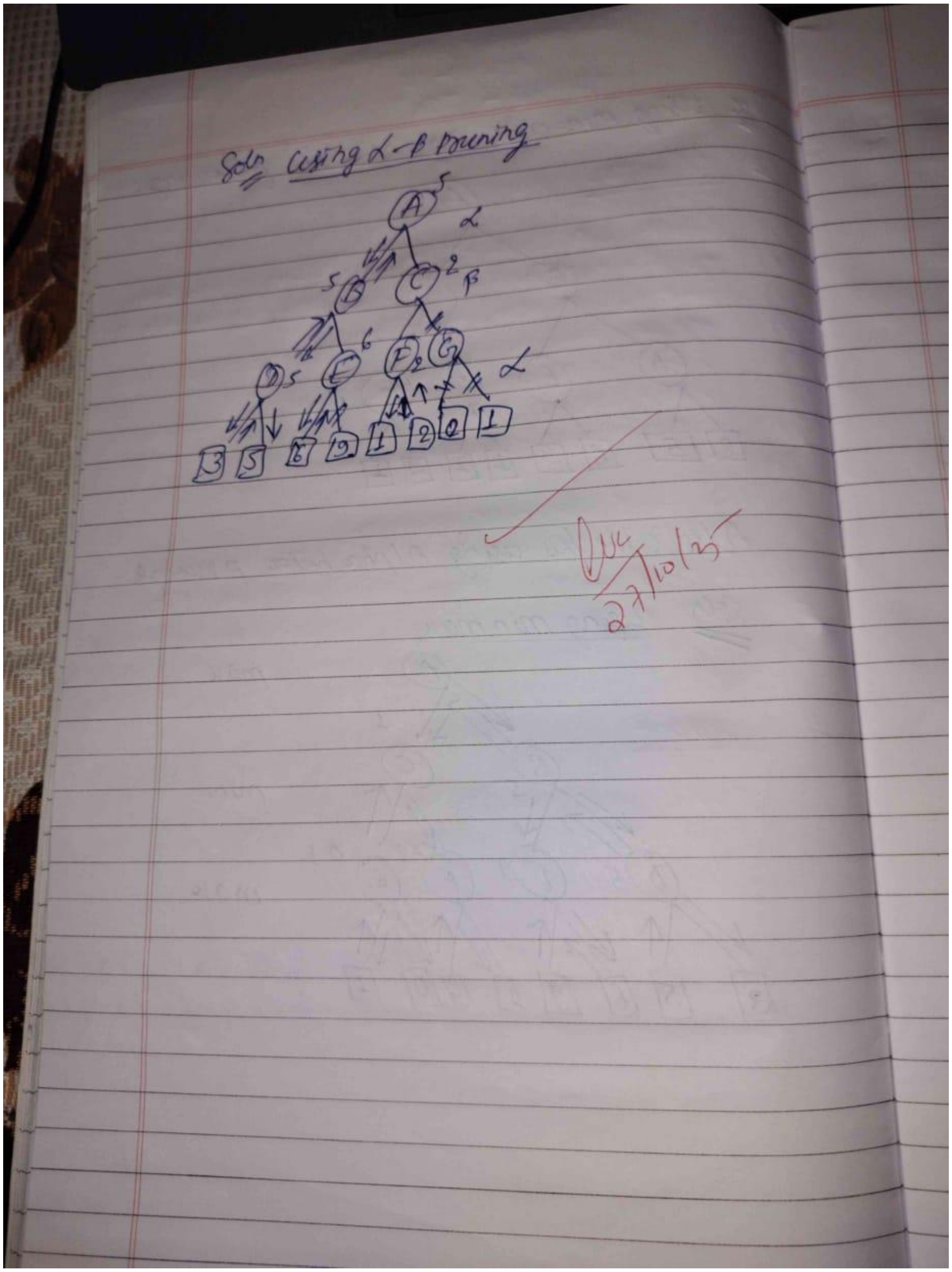
Que Using min-max Alg solve the following :-



Also solve this using Alpha beta pruning:-

Soln using min max





Soln using α-β Pruning

27/10/25

Code:

import math

def alpha_beta(node, depth, alpha, beta, maximizingPlayer, values, index=0): # Terminal condition: leaf node or depth reached if depth == 0 or index >= len(values): return values[index]

```
if maximizingPlayer:
    maxEval = -math.inf
    for i in range(2):  # two children per node
        val = alpha_beta(node*2 + i + 1, depth - 1, alpha, beta, False, values, 
index*2 + i)
        maxEval = max(maxEval, val)
        alpha = max(alpha, val)
        if beta <= alpha:
            print(f"Pruned at MAX node with α={alpha}, β={beta}")
            break
    return maxEval
else:
    minEval = math.inf
    for i in range(2):
        val = alpha_beta(node*2 + i + 1, depth - 1, alpha, beta, True, values, index*2 
+ i)
        minEval = min(minEval, val)
        beta = min(beta, val)
        if beta <= alpha:
            print(f"Pruned at MIN node with α={alpha}, β={beta}")
            break
    return minEval
```

values = [3, 5, 6, 9, 1, 2, 0, -1] # leaf node heuristic values

print("Alpha–Beta Pruning Process") print("NAME: Samir Chaudhary") print("USN: 1BM23CS294\n")

depth = 3 result = alpha_beta(0, depth, -math.inf, math.inf, True, values)

print("\nOptimal value:", result)

Output:

```
Alpha–Beta Pruning Process

NAME: Samir Chaudhary

USN: 1BM23CS294


Pruned at MAX node with α=6, β=5

Pruned at MIN node with α=5, β=2
```

Optimal value: 5