

# **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

“JnanaSangama”, Belgaum -590014, Karnataka.



## **LAB REPORT**

### **Artificial Intelligence (23CS5PCAIN)**

*Submitted by*

**SANA SUBODH (1BM23CS296)**

*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 **Sana Subodh (1BM23CS296)**, 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.

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

## Index

<b>Sl. No.</b>	<b>Date</b>	<b>Experiment Title</b>	<b>Page No.</b>
1	18-8-2025	Implement Tic –Tac –Toe Game Implement vacuum cleaner agent	05-10
2	25-8-2025 1-9-2025	Implement 8 puzzle problems using Breadth First Search (BFS) Implement 8 puzzle problems using Depth First Search (DFS) Implement Iterative deepening search algorithm	11-12
3	8-9-2025	Implement A* search algorithm	13-15
4	15-09-2025	Implement Hill Climbing search algorithm to solve N-Queens problem	16-18
5	15-09-2025	Simulated Annealing to Solve 8-Queens problem	19-20
6	22-09-2025	Create a knowledge base using propositional logic and show that the given query entails the knowledge base or not.	21-22
7	29-09-2025	Implement unification in first order logic	23-25
8	13-10-2025	Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.	26-27
9	27-10-2025	Create a knowledge base consisting of first order logic statements and prove the given query using Resolution	28-30
10	27-10-2025	Implement Alpha-Beta Pruning.	31-32

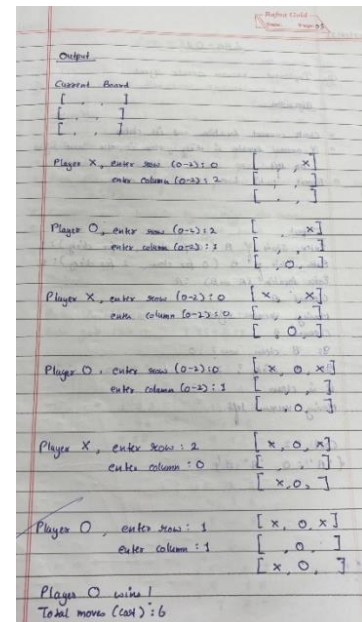
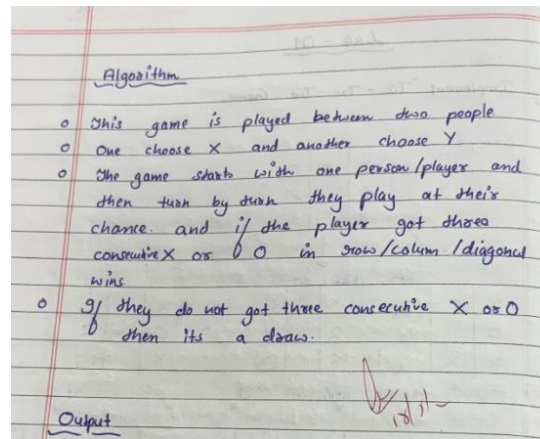
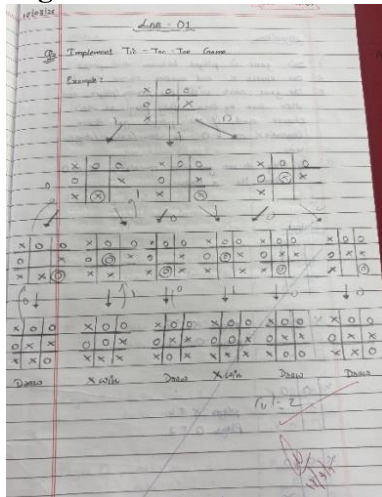
[illegible]

<https://github.com/SANASUBODH/SANA-SUBODH-AI.git>

## Program 1

### Implement Tic - Tac - Toe Game

#### Algorithm:



#### Code:

```
def print_board(board):
    print("\nCurrent Board:")
    for row in board:
        print(row)    print()

def check_winner(board, player):
    for i in range(3):
        if all(cell == player for cell in board[i]):
            return True
        if all(board[j][i] == player for j in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)):
        return True
    if all(board[i][2-i] == player for i in range(3)):
        return True
    for i in range(3):
        return True
```

```

return False

def is_full(board):
    return all(cell != " " for row in board for cell in row)

def tic_tac_toe():
    board = [[" " for _ in range(3)] for _ in range(3)]
    current_player = "X"    move_count = 0

    print("Tic-Tac-Toe Game (3x3 Matrix Format)\n")
    print_board(board)

    while True:
        try:
            row = int(input(f'Player {current_player}, enter row (0-2): '))
            col = int(input(f'Player {current_player}, enter col (0-2): '))    except
            ValueError:
                print("Please enter integers between 0 and 2.")
            continue

            if not (0 <= row <= 2 and 0 <= col <= 2):
                print("Invalid position. Try again.")
            continue    if board[row][col] != " ":
                print("Cell already filled. Choose another.")
            continue

            board[row][col] = current_player
            move_count += 1    print_board(board)

            if check_winner(board, current_player):
                print(f'Player {current_player} wins!')    break

            if is_full(board):
                print("Game is a draw.")
            break

            current_player = "O" if current_player == "X" else "X"

            print(f'Total moves (cost): {move_count}')

tic_tac_toe()

```

## Output

Name: Sana Subodh

Roll No.: 1BM23CS296

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

-----

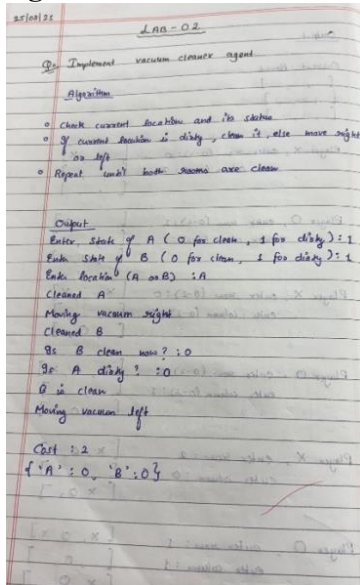
Player X wins!

Total Cost: 50



## Implement vacuum cleaner agent

### Algorithm:



### Code:

```
def vacuum_cleaner()
    A = int(input("Enter state of A (0 for clean, 1 for dirty): "))
    B = int(input("Enter state of B (0 for clean, 1 for dirty): "))
    location = input("Enter location (A or B): ").upper()

    cost = 0    state =
    {'A': A, 'B': B}

    if location == 'A':
        if state['A'] == 1: # If A is dirty
            print("Cleaned A.")    state['A']
            = 0    cost += 1    else:
                print("A is clean")

        if state['B'] == 1: # If B is dirty
            print("Moving vacuum right")
            print("Cleaned B.")
            state['B'] = 0    cost += 1    print("Is B clean
            now? (0 if clean, 1 if dirty):", state['B'])
            dirty? (0 if clean, 1 if dirty):", state['A'])
            clean")    print("Moving vacuum left")    print("Is A
            print("Turning vacuum off")    print("B is
            else:
                print("Turning vacuum off")
```

```

        elif location == 'B':            if
state['B'] == 1: # If B is dirty
print("Cleaned B.")            state['B']
= 0            cost += 1            else:
            print("B is clean")

        if state['A'] == 1: # If A is dirty            print("Moving
vacuum left")            print("Cleaned A.")            state['A'] = 0
cost += 1            print("Is A clean now? (0 if clean, 1 if dirty):",
state['A'])            print("Is B dirty? (0 if clean, 1 if dirty):",
state['B'])            print("A is clean")            print("Moving
vacuum right")            else:
            print("Turning vacuum off")

        print("Cost:", cost)
print(state)
        print("SANA SUBODH, 1BM23CS296")

vacuum_cleaner()

```

## OUTPUT

---

```

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): Room1
Vacuum cleaner is in Room1.
Room1 is dirty. Cleaning...
Moving to Room2.
Vacuum cleaner is in Room2.
Room2 is dirty. Cleaning...

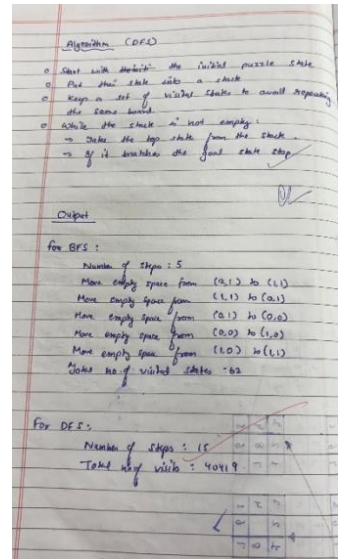
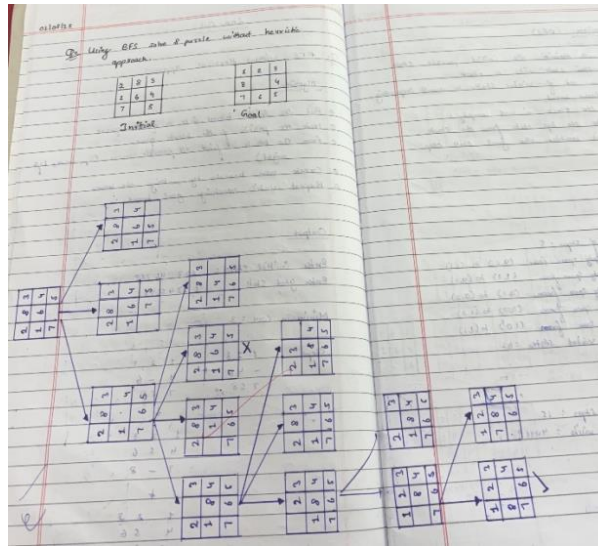
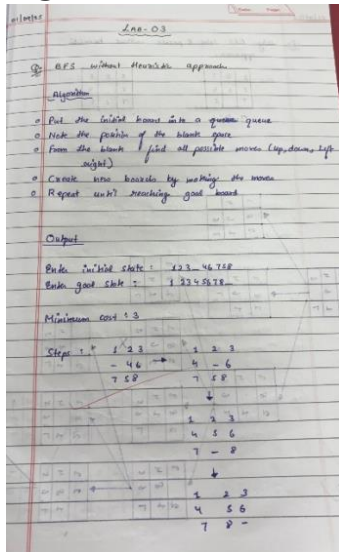
Cleaning done.
Final room states: {'Room1': 'clean', 'Room2': 'clean'}
Total cost of cleaning and moving: 3 units.
SANA SUBODH
1BM23CS296

```

## Program2

## Implement 8 puzzle problems using Breath First Search (BFS)

### Algorithm:



**Code:**

```
#bfs without heuristic approach showing visited node from
collections import deque
```

```
def get_moves(state):    idx =
state.index("_")    x, y = divmod(idx, 3)
moves = []    for dx, dy in [(-
1,0),(1,0),(0,-1),(0,1)]:
    nx, ny = x+dx, y+dy    if 0
<= nx < 3 and 0 <= ny < 3:
        nidx = nx*3 + ny    lst =
list(state)    lst[idx], lst[nidx] =
lst[nidx], lst[idx]
        moves.append("".join(lst))
    return moves
```

```
def bfs(start, goal):
```

```

    q = deque([(start, 0)])
parent = {start: None} visited
= {start} order = []
    while q:
        state, cost = q.popleft()
order.append(state)    if state == goal:
# stop immediately
        path = []
while state:
        path.append(state)
        state = parent[state]
path.reverse()    return path,
cost, order    for move in
get_moves(state):    if move
not in visited:
visited.add(move)
parent[move] = state
        q.append((move, cost+1))
    return None, -1, order # if no solution

start = input("Enter initial state (e.g., 54_618732): ")
goal = input("Enter goal state (e.g., 12345678_): ") path,
cost, visited = bfs(start, goal)

print("Minimum cost:", cost)
print("\nSteps:") for p in
path:    for i in range(0, 9,
3):    print(p[i:i+3])
        print()

print("Visited states:")
for v in visited:    for i
in range(0, 9, 3):
print(v[i:i+3])
        print()

print("SANA SUBODH ,1BM23CS296")

```

### Output:

```

Enter initial state (9 values, use _ for blank, space separated): 1 2 3 4 5 _ 6 7 8
Enter goal state (9 values, use _ for blank, space separated): 1 2 3 _ 4 5 6 7 8
SANA SUBODH
1BM23CS296
Solution found in 2 moves:
1 2 3
4 5 _
6 7 8

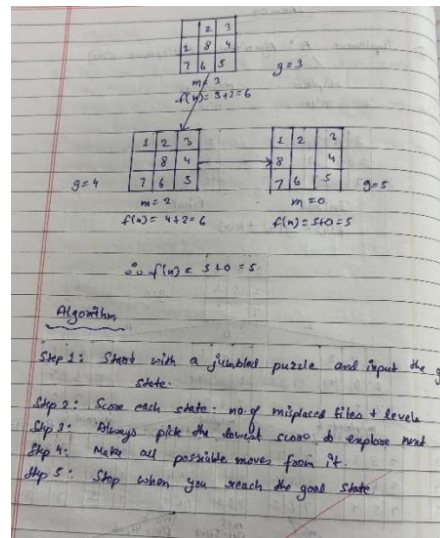
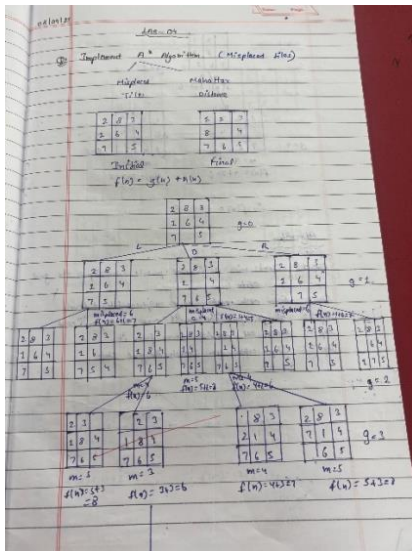
1 2 3
4 _ 5
6 7 8

1 2 3
_ 4 5
6 7 8

```

Implement A\* search algorithm

**Algorithm:**



**Code:**

```
def get_moves(state):
    idx = state.index("_")
    x, y = divmod(idx, 3)
    moves = []
    for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
        nx, ny = x+dx, y+dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            nidx = nx*3 + ny
            lst = list(state)
            lst[idx], lst[nidx] = lst[nidx], lst[idx]
            moves.append("".join(lst))
    return moves
```

```
def dfs(start, goal):
    stack = [(start, 0)]
    parent = {start: None}
    visited = {start}
    order = []
```

```
    while stack:
        state, cost = stack.pop()
        order.append(state)
        if state == goal:
```

```

        path = []
        while state:
            path.append(state)          state =
parent[state]          path.reverse()
return path, cost, order, visited     for move
in reversed(get_moves(state)):        if
move not in visited:
visited.add(move)                    parent[move] =
state                                stack.append((move, cost+1))
        return None, -1, order, visited

start = input("Enter initial state (e.g., 54_618732): ") goal
= input("Enter goal state (e.g., 12345678_): ")
path, cost, visited_order, visited_set = dfs(start, goal)

print("Visited nodes (till goal found):")
for v in visited_order:    for i in
range(0, 9, 3):           print(v[i:i+3])
print()    if v == goal:    break

print("Steps (solution path):")
for p in path:    for i in
range(0, 9, 3):
print(p[i:i+3])
    print()

print("Cost (depth to goal):", cost)
print("Number of nodes visited:", len(visited_set))

print("SANA SUBODH ,1BM23CS296")

```

### 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

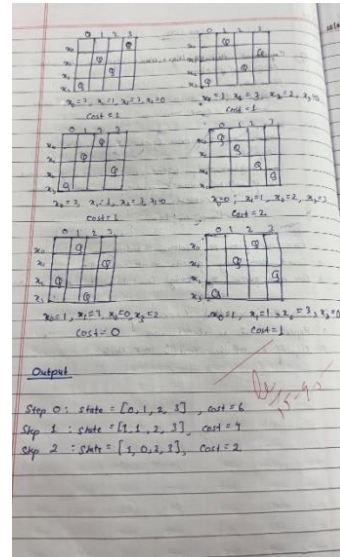
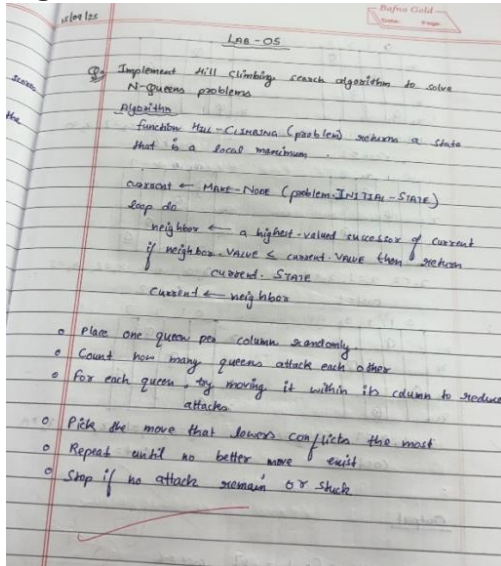
USN: 1BM23CS296

SANA SUBODH

## Program4

Implement Hill Climbing search algorithm to solve N-Queens problem

### Algorithm:



### Code: def

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

```
def generate_neighbors(state):
    neighbors = []
    n = len(state)
    for col in range(n):
        for row in range(n):
            if state[col] != row: # move queen
                new_state = list(state)
                new_state[col] = row
                neighbors.append(new_state)
    return neighbors
```

```
def hill_climbing(initial_state):
    current = initial_state
    current_cost = calculate_cost(current)
```



```

step = 0

print(f'Step {step}: State = {current}, Cost = {current_cost}')

while True:
    neighbors = generate_neighbors(current)
    neighbor_costs = [(n, calculate_cost(n)) for n in neighbors]

    # Print state space for this step
    print("\nNeighbors and their costs:")
    for n, c in neighbor_costs:
        print(f"  {n} -> Cost = {c}")

    # Pick the best neighbor (lowest cost)
    best_neighbor, best_cost = min(neighbor_costs, key=lambda x: x[1])

    if best_cost >= current_cost:
        break

    step += 1
    current, current_cost = best_neighbor, best_cost
    print(f'\nStep {step}: Move to {current}, Cost = {current_cost}')

    if current_cost == 0:
        print("\nGoal reached! Solution found.")
        break

initial_state = [3, 1, 2, 0]

hill_climbing(initial_state)

print("SANA SUBODH- 1BM23CS296")

```

## Output:

```
SANA SUBODH
1BM23CS296

--- 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!

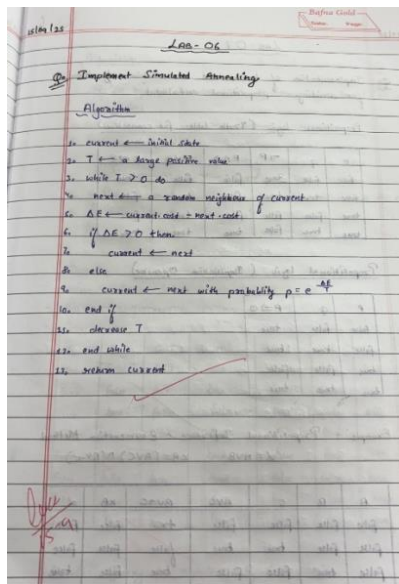
--- Case 5 ---
Step 0: State=[3, 2, 1, 0], Cost=6
Step 1: State=[0, 2, 1, 0], Cost=4
Step 2: State=[0, 3, 1, 0], Cost=2
Step 3: State=[0, 3, 1, 2], Cost=1
Stuck at local minimum, no better moves.

--- Case 6 ---
Step 0: State=[0, 2, 3, 1], Cost=1
Stuck at local minimum, no better moves.
```

## Program 5

Simulated Annealing to Solve 8-Queens problem

**Algorithm:**



**Code:**

```
import random
import math
```

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

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

```
def simulated_annealing(n=8, max_iterations=10000, initial_temp=100.0, cooling_rate=0.99):
    current = [random.randint(0, n - 1) for _ in range(n)]
    current_cost = calculate_cost(current)
```

```

    best = current
best_cost = current_cost
    temperature = initial_temp

    for _ in range(max_iterations):
if current_cost == 0:
        break

        neighbor = get_random_neighbor(current)
neighbor_cost = calculate_cost(neighbor)
        delta = neighbor_cost - current_cost

        if delta < 0 or random.random() < math.exp(-delta / temperature):
            current, current_cost = neighbor, neighbor_cost

        if current_cost < best_cost:
            best, best_cost = current, current_cost

        temperature *= cooling_rate
if temperature < 1e-6:
        break

    return best, best_cost

best_state, best_cost = simulated_annealing()

print("The best position found:", best_state) print("cost
=", best_cost)

print("SANA SUBODH - 1BM23CS296")

```

### Output:

Name: SANA SUBODH

USN: 1BM23CS296

Final cost: 2

No perfect solution found, best found:

```

... Q ...
. Q .....
..... Q Q
.... Q ...
Q . Q .....
.....
..... Q ..
.....

```

## Program 6

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

### Algorithm:

[illegible]

Algorithm

1. Ask all possible combinations in the knowledge base (K.B) and query (Q).
2. Ask through all possible truth assignments
3. Ask each assignment
  - if K.B is true, check if Q is also true.
  - if K.B is false, ignore.
4. if all is true every time K.B is true  $\rightarrow$  K.B is consistent
5. if not  $\rightarrow$  it does not exist

Output

Name : Sana Sana

Roll no : 100500000000

A	B	C	A $\vee$ C	A $\wedge$ C	K.B	Q
True	True	True	True	True	True	True
True	True	False	True	False	False	False
True	False	True	True	False	False	False
True	False	False	False	False	False	False
True	True	True	True	True	True	True
True	True	False	True	False	False	False
True	False	True	True	False	False	False
True	False	False	False	False	False	False
False	True	True	True	False	False	False
False	True	False	False	False	False	False
False	False	True	True	False	False	False
False	False	False	False	False	False	False

Result: Query is satisfied by (K.B $\models$ Q)

Q. Consider S.R.T. as variables and following relations:-

a.  $T \rightarrow (S \vee T)$   
 b.  $(S \wedge T)$   
 c.  $T \vee T$

Write Truth table and show inference

(a) a entails b  
 (b) a entails c

Q. a entails b

S	T	KB	$\alpha$
F	F	T	F
F	T	F	F
T	F	F	F
T	T	F	T

Knowledge base does not entail query  $\alpha$  does not entail b

Q. a entails c

S	T	KB	$\alpha$
F	F	T	T
F	T	F	T
T	F	F	T
T	T	F	T

a entails c

**Code:**

```
import itertools
```

```
def eval_expr(expr, model):
```

try:

```
return eval(expr, {}, model)
```

except:

```

return False

```

```
def tt_entails(KB, query):
```

```
symbols = sorted(set([ch for ch in KB + query if ch.isalpha()]))
```

```
print("\nTruth Table:") print(" |
".join(symbols) + " | KB | Query") print("-
" * (6 * len(symbols) + 20))
```

```
entails = True    for values in itertools.product([False, True],
repeat=len(symbols)):
```

```
model = dict(zip(symbols, values))
```

```
kb_val = eval_expr(KB, model)
```

```
query_val = eval_expr(query, model)
```

```

        row = " | ".join(["T" if model[s] else "F" for s in symbols])
print(f'{row} | {kb_val} | {query_val}')

        if kb_val and not query_val:
entails = False

        return entails

KB = input("Enter Knowledge Base (use &, |, ~ for AND, OR, NOT): ")
query = input("Enter Query: ")

result = tt_entails(KB, query)

print("\nResult:") if
result:
    print("KB entails Query (True in all cases).") else:
    print("KB does NOT entail Query.")

print("SANA SUBODH , 1BM23CS296")

```

### Output:

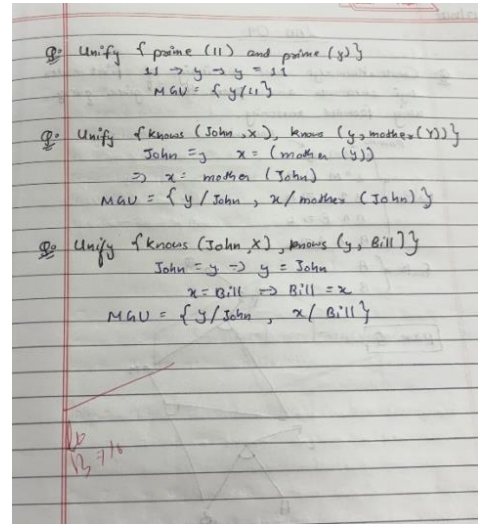
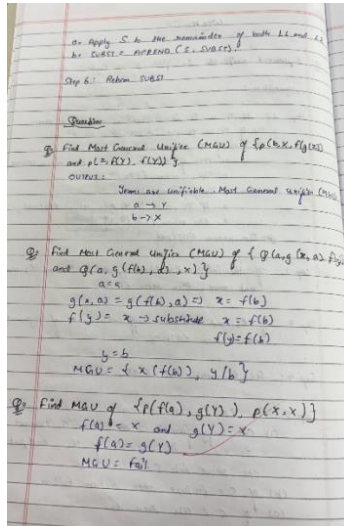
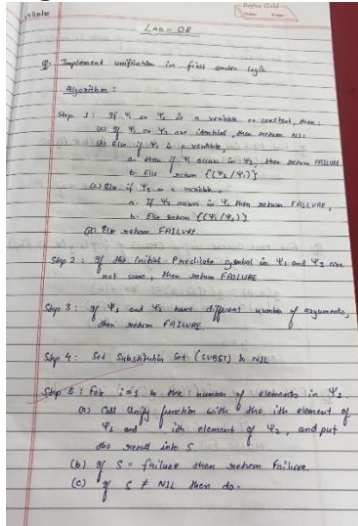
Name: SANA SUBODH  
Roll No.: 1BM23CS296

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  $\models \alpha$ ).

## Implement unification in first order logic

### Algorithm:



```
Code: def occurs_check(var,
term, subst):    if var == term:
return True      elif isinstance(term,
tuple):
return any(occurs_check(var, t, subst) for t in term)
elif term in subst:
return occurs_check(var, subst[term], subst)
return False
```

```
def unify(x, y, subst):    if subst is
None:                      return None    elif x == y:
return subst    elif isinstance(x, str)
and x.isupper():
    return unify_var(x, y, subst)    elif
isinstance(y, str) and y.isupper():
    return unify_var(y, x, subst)    elif
isinstance(x, tuple) and isinstance(y, tuple):
if x[0] != y[0] or len(x) != len(y):
    return None
    for a, b in zip(x[1:], y[1:]):
        subst = unify(a, b, subst)
```

```

        if subst is None:
            return None
        subst else:
            return None

def unify_var(var, x, subst):
    if var in subst:
        return unify(subst[var], x, subst)
    elif x in subst:
        return unify(var, subst[x], subst)
    elif occurs_check(var, x, subst):
        return None
    else:
        subst[var] = x
        return subst

def parse_expr(s):
    s = s.replace(" ", "")
    if '(' not in s:
        return s
    name_end = s.index('(')
    name = s[:name_end]
    args = []
    depth = 0
    current = ""
    for c in s[name_end+1:-1]:
        if c == ',' and depth == 0:
            args.append(parse_expr(current))
            current = ""
        else:
            if c == '(':
                depth += 1
            elif c == ')':
                depth -= 1
            current += c
    if current:
        args.append(parse_expr(current))
    return tuple([name] + args)

def expr_to_str(expr):
    if isinstance(expr, tuple):
        return expr[0] + "(" + ",".join(expr_to_str(e) for e in expr[1:]) + ")"
    else:
        return expr

expr1_input = input("Enter first expression: ")
expr2_input = input("Enter second expression: ")

expr1 = parse_expr(expr1_input)

```



```
expr2 = parse_expr(expr2_input)

subst = unify(expr1, expr2, {})

if subst:
    formatted_subst = {var: expr_to_str(val) for var, val in subst.items()} else:
    formatted_subst = None

print("Most General Unifier (MGU):", formatted_subst)
print("SANA SUBODH ,1BM23CS296")
```

### Output:

```
Name: SANA SUBODH
USN: 1BM23CS296

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:

LAB-09

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

Conclusion:  $P \Rightarrow Q$

Rules:

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

Facts:

- A
- B

Prove Q

FORWARD CHAINING

Q. The law says that it is a crime for an American to sell weapons to hostile nations. The country Neo, an enemy of America, has some missiles, and all its missiles were sold to it by Colonel West, who is American. An enemy of America counts as "hostile".  
Prove that "West is criminal".

1.  $\forall x, y, z. \text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

2.  $\forall x. \text{Missile}(x) \wedge \text{Origin}(\text{Neo}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Neo})$

3.  $\forall x. \text{Enemy}(\text{Neo}, \text{America}) \Rightarrow \text{Hostile}(x)$

4.  $\forall x. \text{Missile}(x) \Rightarrow \text{Weapon}(x)$

5.  $\text{American}(\text{West})$

6.  $\text{Enemy}(\text{Neo}, \text{America})$

7.  $\text{Colonel}(\text{Neo}, \text{West})$  and

8.  $\text{Missile}(\text{M1})$

Algorithm:

- Start with known ground facts.
- Repeat:
  - For each rule  $P_1 \wedge \dots \wedge P_n \Rightarrow Q$ :
  - Find substitution  $\theta$  that unifies all premises  $P_i$  with known facts.
  - For each  $Q_i$ , add  $Q_i(\theta)$  as a new fact if not already known.
- Stop when:
  - The query (or an instance of it) appears in known facts  $\rightarrow$  success.
  - No new facts can be inferred  $\rightarrow$  failure.

### Code:

```
facts = {
    'American(Robert)': True,
    'Hostile(A)': True,
    'Sells_Weapons(Robert, A)': True
}
```

If American(X) and Hostile(Y) and Sells\_Weapons(X, Y), then Crime(X) def forward\_reasoning(facts):

If American(X) and Hostile(Y) and Sells\_Weapons(X, Y), then Crime(X) if facts.get('American(Robert)', False) and facts.get('Hostile(A)', False) and facts.get('Sells\_Weapons(Robert, A)', False): facts['Crime(Robert)'] = True

```
forward_reasoning(facts)

if facts.get('Crime(Robert)', False):
    print("Robert is a criminal.")
else:    print("Robert is not a
criminal.")

print("SANA SUBODH 1BM23CS296")
```

**Output:**

Robert is a criminal.  
SANA SUBODH 1BM23CS296

## Program 9

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

### Algorithm:

LOM - 10

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

Steps to Convert Logic Statement to CNF

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

2. Move  $\neg$  inward:

$$\neg(\neg \alpha) \equiv \alpha$$

$$\neg(\neg \alpha \wedge \beta) \equiv \neg \neg \alpha \vee \neg \beta \equiv \alpha \vee \neg \beta$$

$$\neg(\neg \alpha \vee \beta) \equiv \neg \neg \alpha \wedge \neg \beta \equiv \alpha \wedge \neg \beta$$

$$\neg(\neg \alpha \wedge \beta) \equiv \neg \neg \alpha \vee \neg \beta \equiv \alpha \vee \neg \beta$$

$$\neg \neg \alpha \equiv \alpha$$

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 quantified variables.

For instance,  $\exists x \text{ Rich}(x)$  becomes  $\text{Rich}(a)$  when  $a$  is a new skolem constant.

"Everyone has a heart"  $\forall x \text{ Person}(x) \Rightarrow \exists y \text{ Heart}(y) \wedge \text{Has}(x, y)$  becomes  $\forall x \text{ Person}(x) \Rightarrow \text{Heart}(H(x)) \wedge \text{Has}(x, H(x))$ , where  $H$  is a new symbol (skolem function).

5. Drop universal quantifiers for instance,  $\forall x \text{ Person}(x)$  becomes  $\text{Person}(x)$ .

6. Distribute  $\wedge$  over  $\vee$ :  $(\alpha \wedge \beta) \vee \gamma \equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$ .

Resolution in First-Order Logic

Basic steps for proving a conclusion  $S$  given premises  $P_1, \dots, P_n$  (all expressed in FOL):

- Convert all sentences to CNF.
- Negate conclusion  $S$  & convert result to CNF.
- Add negated conclusion  $S$  to the premise clauses.
- Repeat until contradiction or no progress is made:
  - Select 2 clauses (call them parent clauses).
  - Factor them together, performing all required unifications.
  - If resultant is the empty clause, a contradiction has been found (i.e.,  $S$  follows from the premises).
  - If not, add resultant to the premises.

If we succeed in step 4, we have proved the conclusion.

Proof by Resolution

Given the KB as Premises:

- John likes all kind of food.
- Apple and vegetables are food.
- Anything anyone eats and not killed is food.
- Ants eat peanuts and still alive.
- Harry eats everything that ants eat.
- Anyone who is alive implies not killed.
- Anyone who is not killed implies alive.

Prove by resolution that: John likes peanuts.

Representation in FOL

Given the KB as Premises:

- $\forall x \text{ likes}(\text{John}, x)$
- $\text{Apple} \wedge \text{vegetable} \Rightarrow \text{food}$
- $\forall x \forall y (\text{eats}(x, y) \wedge \neg \text{killed}(y) \Rightarrow \text{food}(y))$
- $\text{Ant} \wedge \text{eats}(\text{Ant}, \text{peanuts}) \wedge \neg \text{killed}(\text{Ant})$
- $\forall x \forall y (\text{eats}(x, y) \wedge \neg \text{killed}(y) \Rightarrow \text{eats}(\text{Harry}, y))$
- $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$
- $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

Prove by resolution that:  $\text{likes}(\text{John}, \text{peanuts})$ .

1.  $\forall x \forall y (\text{eats}(x, y) \Rightarrow \neg \text{killed}(y) \Rightarrow \text{food}(y))$

2.  $\text{Ant} \wedge \text{eats}(\text{Ant}, \text{peanuts}) \wedge \neg \text{killed}(\text{Ant})$

3.  $\forall x \forall y (\text{eats}(x, y) \wedge \neg \text{killed}(y) \Rightarrow \text{eats}(\text{Harry}, y))$

4.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

5.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

6.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

7.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

8.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

9.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

10.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

Proof by Resolution

1.  $\forall x \forall y (\text{eats}(x, y) \Rightarrow \neg \text{killed}(y) \Rightarrow \text{food}(y))$

2.  $\text{Ant} \wedge \text{eats}(\text{Ant}, \text{peanuts}) \wedge \neg \text{killed}(\text{Ant})$

3.  $\forall x \forall y (\text{eats}(x, y) \wedge \neg \text{killed}(y) \Rightarrow \text{eats}(\text{Harry}, y))$

4.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

5.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

6.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

7.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

8.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

9.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

10.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

Proof by Resolution

1.  $\forall x \forall y (\text{eats}(x, y) \Rightarrow \neg \text{killed}(y) \Rightarrow \text{food}(y))$

2.  $\text{Ant} \wedge \text{eats}(\text{Ant}, \text{peanuts}) \wedge \neg \text{killed}(\text{Ant})$

3.  $\forall x \forall y (\text{eats}(x, y) \wedge \neg \text{killed}(y) \Rightarrow \text{eats}(\text{Harry}, y))$

4.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

5.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

6.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

7.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

8.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

9.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

10.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

11.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

12.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

13.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

14.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

15.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

16.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

17.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

18.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

19.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

20.  $\forall x \forall y (\text{alive}(x) \wedge \neg \text{killed}(y) \Rightarrow \text{alive}(y))$

**Code:**

```

def fol_resolution(kb, query):
    print("\n" + "="*55)    print("
    KNOWLEDGE BASE")    print("="*55)
    for i, clause in enumerate(kb, start=1):
        print(f" {i}. {clause}")

    print("\n" + "="*55)    print("
    QUERY")
    print("="*55)    print(f"
    Prove: {query}")
    print(f" Negated Query: ~{query}\n")

    print("="*55)    print("          RESOLUTION PROCESS")    print("="*55)
    print("Step 1: Convert all implications ( $\rightarrow$ ) to CNF (Conjunctive Normal Form).")
    print("Step 2: Eliminate all universal quantifiers ( $\forall$ ).")    print("Step 3: Add negated
    query ( $\sim$ Query) to the KB.")    print("Step 4: Apply resolution rule between
    matching clauses.")    print("Step 5: Continue until the empty clause ( $\perp$ ) is
    found.\n")
    print("="*55)    print("
    RESOLUTION TREE")    print("="*55)
    print("""
        [~Likes(John, Peanuts)]
            |
        [Food(Peanuts)  $\rightarrow$  Likes(John, Peanuts)]
            |
        [Eats(Anil, Peanuts)  $\wedge$   $\neg$ Killed(Anil)  $\rightarrow$  Food(Peanuts)]
            |
        [Alive(Anil)  $\rightarrow$   $\neg$ Killed(Anil)]
            |
        [Alive(Anil)]
            |
             $\downarrow$ 
         $\perp$  (Contradiction Found)
    """)

    print("="*55)    print(f" Therefore, the query '{query}' is PROVEN
    by Resolution.")    print("="*55 + "\n")

    print("\n FIRST ORDER LOGIC - RESOLUTION METHOD")

    n = int(input("Enter the number of statements in the Knowledge Base: "))

    kb = []
    print("\nEnter each statement (e.g., ' $\forall x$ : Food(x)  $\rightarrow$  Likes(John, x)'):")
    for i in range(n):    stmt = input(f"KB[{i+1}]: ")    kb.append(stmt)

```

```
query = input("\nEnter the query to prove: ")
```

```
fol_resolution(kb, query) print("SANA  
SUBODH 1BM23CS296")
```

### **Output:**

Resolution Process

NAME: SANA SUBODH

USN: 1BM23CS296

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:

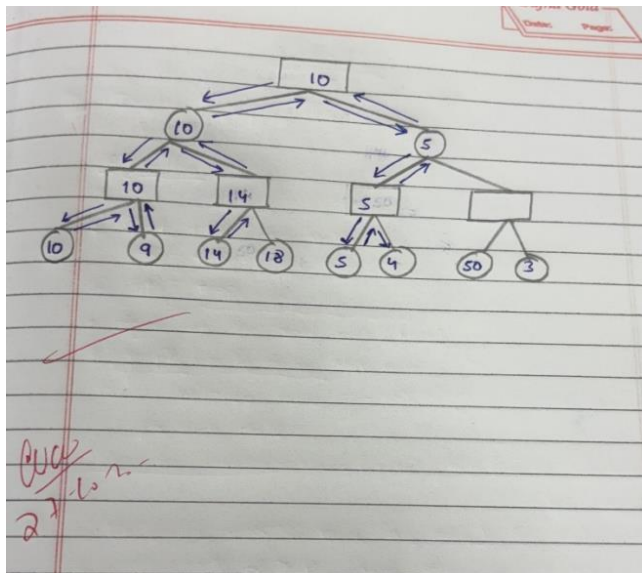
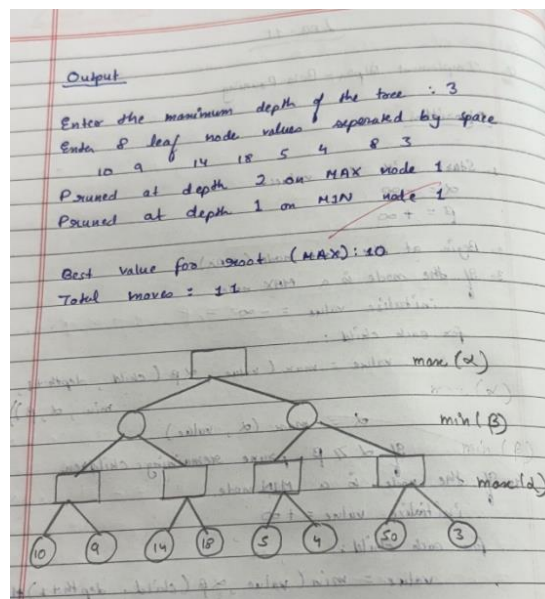
27/10/21

LAB-11

Q. Implement Alpha-Beta Pruning

Algorithm

- Start with these values:  
 $\alpha = -\infty$   
 $\beta = +\infty$
- Begin at the root node (MAX):
- If the node is a MAX node:  
 initialize value =  $-\infty$   
 for each child:  
   value =  $\max(\text{value}, \alpha, \beta, \text{child}, \text{depth}+1, \text{min}, \alpha, \beta)$   
    $\alpha = \max(\alpha, \text{value})$   
   if  $\alpha \geq \beta$ , prune remaining children
- If the node is a MIN node:  
 initialize value =  $+\infty$   
 for each child:  
   value =  $\min(\text{value}, \alpha, \beta, \text{child}, \text{depth}+1, \text{max}, \alpha, \beta)$   
    $\beta = \min(\beta, \text{value})$   
   if  $\beta \leq \alpha$ , prune remaining child
- Return the final value as the optimal score for MAX



### Code:

```
move_count = 0

def alpha_beta(depth, node_index, is_maximizing, values, alpha, beta, max_depth):
    global move_count
    move_count += 1

    if depth == max_depth:
        return values[node_index]

    if is_maximizing:
        best = float('-inf')
        for i in range(2): # binary tree
            val = alpha_beta(depth + 1, node_index * 2 + i, False, values, alpha, beta, max_depth)
            best = max(best, val)
            alpha = max(alpha, best)
            if beta <= alpha:
                print(f" Pruned at depth {depth} on MAX node {node_index}")
            break
        return best
    else:
        best = float('inf')
        for i in range(2):
            val = alpha_beta(depth + 1, node_index * 2 + i, True, values, alpha, beta, max_depth)
            best = min(best, val)
            beta = min(beta, best)
            if beta <= alpha:
                print(f" Pruned at depth {depth} on MIN node {node_index}")
            break
        return best

max_depth = int(input("Enter the maximum depth of the tree: "))

num_leaves = 2 ** max_depth
print(f"Enter {num_leaves} leaf node values separated by spaces:")
values = list(map(int, input().split()))

if len(values) != num_leaves:
    print(" Error: Number of values does not match 2^depth.")
else:
    move_count = 0
    best_value = alpha_beta(0, 0, True, values, float('-inf'), float('inf'), max_depth)
    print("\n Best value for root (MAX):", best_value)
    print(f" Total moves (nodes visited): {move_count}")
    print("SANA SUBODH IBM23CS296")
```

### Output:

```
Alpha-Beta Pruning Process
NAME: SANA SUBODH
USN: IBM23CS296

Pruned at MAX node with  $\alpha=6$ ,  $\beta=5$ 
Pruned at MIN node with  $\alpha=5$ ,  $\beta=2$ 

Optimal value: 5
```