# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.



## LAB REPORT on

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

Submitted by

Samriddhi Singh (1BM23CS295)

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 **Samriddhi Singh (1BM23CS295)**, 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	Dr. Kavitha Sooda
Assistant Professor	Professor & HOD
Department of CSE, BMSCE	Department of CSE, BMSCE

## Index

Sl. No.	Date	Experiment Title	Page No.
1	18-8-2025	Implement Tic –Tac –Toe Game Implement vacuum cleaner agent	5-14
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	15-21
3	8-9-2025	Implement A* search algorithm	22-31
4	15-09-2025	Implement Hill Climbing search algorithm to solve N-Queens problem	32-33
5	15-09-2025	Simulated Annealing to Solve 8-Queens problem	34-35
6	22-09-2025	Create a knowledge base using propositional logic and show that the given query entails the knowledge base or not.	36-37
7	29-09-2025	Implement unification in first order logic	38-40
8	13-10-2025	Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.	41-42
9	27-10-2025	Create a knowledge base consisting of first order logic statements and prove the given query using Resolution	43-46
10	27-10-2025	Implement Alpha-Beta Pruning.	47-48

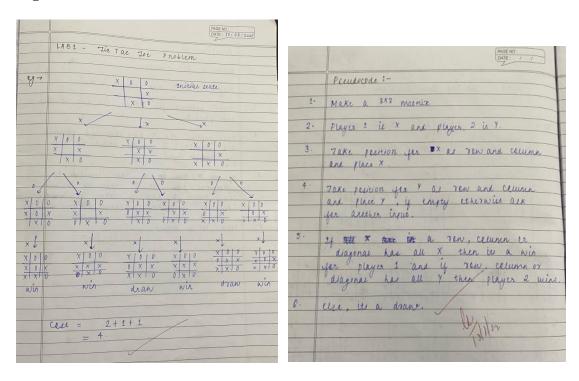
 $Github\ Link:\ \underline{https://github.com/samriddhisingh05/SamriddhiSingh}\ 1BM23CS295-AILab.git$ 

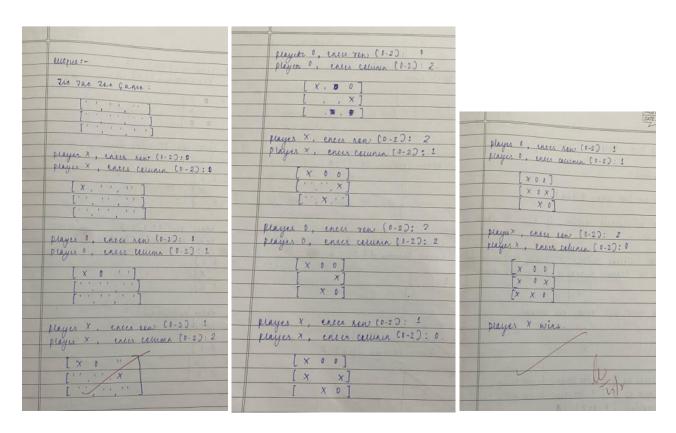
	Samsiddhi Singh	Sub:		
Std:	Div:	Roll No :		
elephon	C-Mail ID :			
Blood Gr	oup : Birth Day :	-1.to	1	
Sr. No	Title	Daly's	Sign/Remarks	dat
1.	Tic-Tac-Tee	A	10	18/8
2 .	Vacuum Cleaner	Wast.	10	25/8
3.	a) BFS ( Without Henristic approach)	- 2	10	1/9/
	b) BFS ( with Heurictic approach)		60	-1 11
	c) Iterative deepening (DFS)	1	(10	
4 -	A* algorithm - muspinced title	1-9	,	81912
	- Manhattan Diseance	W	410	
	Manufacture in	8-9	J	
5.	Hill clinibing algorithm.	1		15/9
1 4 1	Stimulated Arreating	1	10	4
		inte	J	
6.	Propositional legie	lk	10	2219
		22/1		
7.	unification algorithm	14/12	10	29/9
8.	First order Logic	13/26	_	13/1
9.	FOL ( With KB)	Me	lo	27/10
10.	Adversarial Search	27-1-		21110
				100
/	(enpuls)			
			The P	
-				

## Program 1

Implement Tic –Tac –Toe Game

## Algorithm:





```
def print_board(board):
    for row in board:
        print(row)

def check_winner(board, player):

    for i in range(3):
        if all(board[i][j] == player for j in range(3)):
            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

return True
```

```
def tic_tac_toe():
  board = [["" for _ in range(3)] for _ in range(3)]
  players = ["X", "O"]
  turn = 0
  path\_cost = 0
  for move in range(9):
     print("\nCurrent Board:")
     print_board(board)
     player = players[turn % 2]
     print(f"\nPlayer {player}'s turn:")
     row = int(input("Enter row (0-2): "))
    col = int(input("Enter col (0-2): "))
     if board[row][col] == " ":
       board[row][col] = player
       path_cost += 1
     else:
       print("Spot taken! Try again.")
       continue
     if check_winner(board, player):
       print("\nFinal Board:")
       print_board(board)
       print(f"\nPlayer {player} wins!")
       print(f"Path Cost = {path_cost}")
       return
    turn += 1
  print("\nFinal Board:")
  print_board(board)
  print("\nIt's a draw!")
  print(f"Path Cost = {path_cost}")
```

```
tic_tac_toe()
```

## print("Samriddhi Singh 1BM23CS295")

#### **OPUTPUT-Tic-tac-toe**

#### Case 1-

Current Board:

['','','']

['','','']

['','','']

Player X's turn:

Enter row (0-2): 0

Enter col (0-2): 0

Current Board:

['X', '', '']

['','','']

['','','']

Player O's turn:

Enter row (0-2): 0

Enter col (0-2): 1

Current Board:

['X', 'O', ' ']

['','','']

['','','']

Player X's turn:

Enter row (0-2): 1

Enter col (0-2): 1

Current Board:

['X', 'O', ' ']

['', 'X', '']

['','','']

Player O's turn:

Enter row (0-2): 1

Enter col (0-2): 2

Current Board:

['X', 'O', ' ']

[' ', 'X', 'O']

['','','']

Player X's turn:

Enter row (0-2): 2

Enter col (0-2): 2

Final Board:

['X', 'O', ' ']

[' ', 'X', 'O']

['','','X']

Player X wins!

Path Cost = 5

Samriddhi Singh 1BM23CS295

Case 2-

Current Board:

['','',''] ['','','']

['','','']

Player X's turn:

Enter row (0-2): 0

Enter col (0-2): 0

Current Board:

['X', '', '']

['','','']

['','','']

Player O's turn:

Enter row (0-2): 0

Enter col (0-2): 1

Current Board:

['X', 'O', ' ']

['','','']

['','','']

Player X's turn:

Enter row (0-2): 1

Enter col (0-2): 0

Current Board:

['X', 'O', ' ']

['X', ' ', ' ']

['','','']

Player O's turn:

Enter row (0-2): 1

Enter col (0-2): 1

Current Board:

['X', 'O', ' ']

['X', 'O', ' ']

['','','']

Player X's turn:

Enter row (0-2): 0

Enter col (0-2): 2

Current Board:

['X', 'O', 'X']

['X', 'O', ' ']

['','','']

Player O's turn:

Enter row (0-2): 2

Enter col (0-2): 2

Current Board:

['X', 'O', 'X']['X', 'O', ' ']

['','','O']

Player X's turn:

Enter row (0-2): 1

Enter col (0-2): 2

Current Board:

['X', 'O', 'X']

['X', 'O', 'X']

['','','O']

Player O's turn:

Enter row (0-2): 2

Enter col (0-2): 1

Final Board:

['X', 'O', 'X']

['X', 'O', 'X']

[' ', 'O', 'O']

Player O wins!

Path Cost = 8

Samriddhi Singh 1BM23CS295

#### Case 3-

Current Board:

['','','']

['','','']

['','','']

Player X's turn:

Enter row (0-2): 0

Enter col (0-2): 0

Current Board:

['X', '', '']

['','','']

['','','']

Player O's turn:

Enter row (0-2): 1

Enter col (0-2): 0

Current Board:

['X', '', '']

['O', '', '']

['','','']

Player X's turn:

Enter row (0-2): 0

Enter col (0-2): 2

Current Board:

['X', ' ', 'X']

 $['O',\,'\,',\,'\,']['\,',\,'\,',\,'\,']$ 

Player O's turn:

Enter row (0-2): 0

Enter col (0-2): 1

Current Board:

['X', 'O', 'X']

['O', '', '']

['','','']

Player X's turn:

Enter row (0-2): 1

Enter col (0-2): 2

Current Board:

['X', 'O', 'X']

['O', ' ', 'X']

['','','']

Player O's turn:

Enter row (0-2): 2

Enter col (0-2): 2

Current Board:

['X', 'O', 'X']

['O', ' ', 'X']

['','','O']

Player X's turn:

Enter row (0-2): 1

Enter col (0-2): 1

Current Board:

['X', 'O', 'X']

['O', 'X', 'X']

['','','O']

Player O's turn:

Enter row (0-2): 2

Enter col (0-2): 0

Current Board:

['X', 'O', 'X']

['O', 'X', 'X']

['O', ' ', 'O']

Player X's turn:

Enter row (0-2): 2

Enter col (0-2): 1

Final Board:

['X', 'O', 'X']

['O', 'X', 'X']

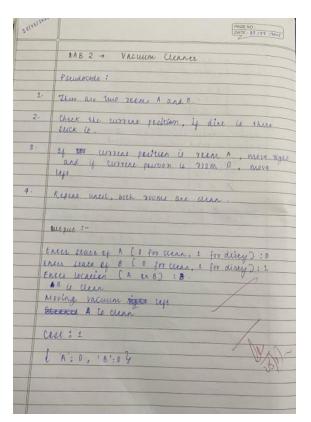
['O', 'X', 'O']

It's a draw!

Path Cost = 9

#### Implement vacuum cleaner agent

## **Algorithm:**



```
def vacuum world():
  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): "))
  loc = input("Enter location (A or B): ").upper()
  cost, s = 0, {'A': A, 'B': B}
  if loc == 'A':
     print("Cleaned A." if s['A'] else "A is clean"); cost += s['A']; s['A'] = 0
     print("Is A clean now?:", s['A']); print("Is B dirty?:", s['B'])
     print("Moving vacuum right")
     print("Cleaned B." if s[B'] else "B is clean"); cost += s[B']; s[B'] = 0
  elif loc == 'B':
     print("Cleaned B." if s[B'] else "B is clean"); cost += s[B']; s[B'] = 0
    print("Is B clean now?:", s['B']); print("Is A dirty?:", s['A'])
     print("Moving vacuum left")
     print("Cleaned A." if s['A'] else "A is clean"); cost += s['A']; s['A'] = 0
  else:
     print("Turning vacuum off")
  print("Cost:", cost, "\n", s)
```

## print("Samriddhi Singh 1BM23CS295")

vacuum\_world()

#### **OUTPUT Case1:**

```
Enter state of A (0 for clean, 1 for dirty): 1
Enter state of B (0 for clean, 1 for dirty): 0
Enter location (A or B): A
Cleaned A.
Is A clean now?: 0
Is B dirty?: 0
Moving vacuum right
B is clean
Cost: 1
{'A': 0, 'B': 0}
Samriddhi Singh 1BM23CS295
```

#### **OUTPUT Case2:**

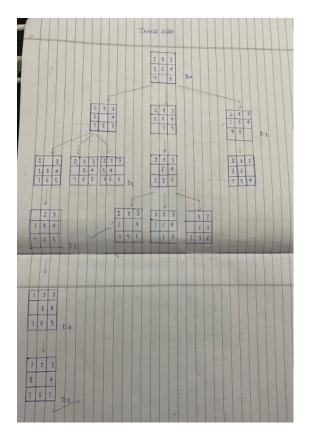
```
Enter state of A (0 for clean, 1 for dirty): 0
Enter state of B (0 for clean, 1 for dirty): 0
Enter location (A or B): a
A is clean
Is A clean now?: 0
Is B dirty?: 0
Moving vacuum right
B is clean
Cost: 0
{'A': 0, 'B': 0}
Samriddhi Singh 1BM23CS295
```

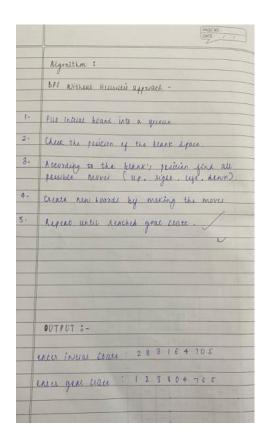
#### **OUTPUT Case3:**

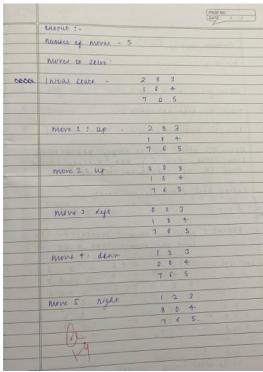
```
Enter state of A (0 for clean, 1 for dirty): 0
Enter state of B (0 for clean, 1 for dirty): 1
Enter location (A or B): B
Cleaned B.
Is B clean now?: 0
Is A dirty?: 0
Moving vacuum left
A is clean
Cost: 1
{'A': 0, 'B': 0}
Samriddhi Singh 1BM23CS295
```

**Program2**Implement 8 puzzle problems using Breath First Search (BFS)

## Algorithm:







#### Code:

from collections import deque

```
moves = {
  'Up': -3,
  'Down': 3,
  'Left': -1,
  'Right': 1
def is valid move(pos, move):
  if move == 'Left' and pos % 3 == 0:
    return False
  if move == 'Right' and pos % 3 == 2:
    return False
  if move == 'Up' and pos < 3:
    return False
  if move == 'Down' and pos > 5:
    return False
  return True
def get_neighbors(state):
  neighbors = []
  zero pos = state.index(0)
  for move, pos change in moves.items():
    if is_valid_move(zero_pos, move):
       new_zero_pos = zero_pos + pos_change
       new_state = list(state)
       new_state[zero_pos],
                                  new_state[new_zero_pos]
                                                                        new_state[new_zero_pos],
new_state[zero_pos]
       neighbors.append((tuple(new_state), move))
  return neighbors
def bfs(start_state, goal_state):
  queue = deque()
  queue.append((start_state, []))
  visited = set()
  visited.add(start_state)
  explored_states = []
  while queue:
    current_state, path = queue.popleft()
    explored_states.append(current_state)
    if current state == goal state:
       return path, explored_states
    for neighbor, move in get_neighbors(current_state):
       if neighbor not in visited:
         visited.add(neighbor)
         queue.append((neighbor, path + [move]))
  return None, explored states
```

```
def input_state(prompt):
  while True:
    raw = input(prompt).strip().split()
     if len(raw) != 9:
       print("Please enter exactly 9 numbers separated by spaces.")
       continue
    try:
       numbers = [int(x) for x in raw]
     except ValueError:
       print("Please enter valid integers only.")
       continue
     if set(numbers) != set(range(9)):
       print("Numbers must be from 0 to 8 without repetition.")
       continue
    return tuple(numbers)
def print_state(state):
  for i in range(3):
     print(state[3*i:3*i+3])
  print()
def main():
  start state = input state("Enter the initial state (9 numbers from 0 to 8, space separated): ")
  goal_state = input_state("Enter the goal state (9 numbers from 0 to 8, space separated): ")
  print("\nStarting BFS...\n")
  path, explored_states = bfs(start_state, goal_state)
  print(f"Total states explored: {len(explored_states)}\n")
  print("States explored in order:")
  for idx, state in enumerate(explored states, 1):
     print(f"State {idx}:")
     print_state(state)
  if path is None:
     print("No solution found.")
  else:
     print(f"Number of moves to solve: {len(path)}")
     print("Moves to solve:")
     current = start_state
     print("Initial state:")
     print_state(current)
     for i, move in enumerate(path, 1):
       print(f"Move {i}: {move}")
       zero_pos = current.index(0)
       pos change = moves[move]
       new_zero_pos = zero_pos + pos_change
       new_state = list(current)
       new_state[zero_pos],
                                  new_state[new_zero_pos]
                                                                         new_state[new_zero_pos],
                                                                  =
new_state[zero_pos]
       current = tuple(new_state)
```

```
print_state(current)
if __name__ == "__main__":
  main()
Output:
Number of moves to solve: 5
Moves to solve:
Initial state:(2, 8, 3)
(1, 6, 4)
(7, 0, 5)
Move 1: Up
(2, 8, 3)
(1, 0, 4)
(7, 6, 5)
Move 2: Up
(2, 0, 3)
(1, 8, 4)
(7, 6, 5)
Move 3: Left
(0, 2, 3)
(1, 8, 4)
(7, 6, 5)
Move 4: Down
(1, 2, 3)
(0, 8, 4)
```

Samriddhi Singh-1BM23CS295

(7, 6, 5) Move 5: Right (1, 2, 3) (8, 0, 4) (7, 6, 5) Implement 8 puzzle problems using Depth First Search (DFS)

## Algorithm:

	PAGE NO.
	Streeties despring (DIS)
	Algorithm:
1-	Stare from initial state.
2 -	Mark the node as vitited
3.	Push the node into a stack.
4.	When the stack is has empty paper node from stack visit ( if not already visited) Push all unvisited hodes into seach.
5.	Repeat until seach become empty.
	ENTPUT: ENELS INHAL State : 2331647-5 Que
	$\begin{array}{cccccccccccccccccccccccccccccccccccc$
	123 113 23
	1 2 3 1 1 3 2 3 8 4 ← 8 4 ← 1 8 4
	765 765 765
	Cest: 5
	no- ey nodes visited: 181++0.

```
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 \le nx < 3 and 0 \le 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("Samriddhi Singh,1BM23CS295")
```

## **Output:**

```
Steps (solution path):
283
164
7_5
283
1_4
765
2_3
184
765
2_3
184
765
123
_84
765
123
_84
765
123
_84
765
123
8_4
765
123
8_4
765
123
8_4
765
123
8_4
765
123
8_4
765
123
8_4
765
123
8_4
765
```

Implement Iterative deepening search algorithm

## Algorithm:

	FRAGENO:
	DES with Stevative despring search -
	Algerium!
L.	Let depth Unit d=0
2 -	Perform DFI upto depth d
3 -	If good speed, acture securior
4.	of nec. Increment d and repeat DPS
£-	steep when goat is found.
-	entiput: enter initial trate - 123 + 0 5 7 1
-	senoching with depth limit : 5
_	with diffe court
	STATCHING WITH depth limit &
	JEATTERING WITH SLEET LAND
	Starching 1215 diffit water 123 -> 123  William park: 405 450
	STATISTICAL RESERVE CONTENTS TO THE STATE OF
	Justing with auft that  1 2 3 -> 1 2 3  11111101 path: 4 0 5 4 5 0
	######################################
	Justing with auft that  1 2 3 -> 1 2 3  11111101 path: 4 0 5 4 5 0

```
def get_neighbors(state):
    neighbors = []
    idx = state.index("0")
    x, y = divmod(idx, 3)
```

```
moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]
  for dx, dy in moves:
     nx, ny = x + dx, y + dy
     if 0 \le nx \le 3 and 0 \le ny \le 3:
       new_idx = nx * 3 + ny
       state list = list(state)
       state_list[idx], state_list[new_idx] = state_list[new_idx], state_list[idx]
       neighbors.append("".join(state list))
  return neighbors
def dfs_limit(start_state, goal_state, limit):
  stack = [(start state, 0)]
  visited = set([start state])
  parent = {start_state: None}
  while stack:
     current_state, depth = stack.pop()
     if current_state == goal_state:
       path = \prod
       while current_state:
          path.append(current_state)
          current state = parent[current state]
       return path[::-1]
     if depth < limit:
       for neighbor in get_neighbors(current_state):
          if neighbor not in visited:
            visited.add(neighbor)
            parent[neighbor] = current_state
            stack.append((neighbor, depth + 1))
  return None
def iddfs(start_state, goal_state, max_depth):
  for limit in range(max_depth + 1):
     print(f"Searching with depth limit: {limit}")
     solution = dfs limit(start state, goal state, limit)
     if solution:
       return solution
  return None
print("Samriddhi Singh 1BM23CS295")
print("Enter the initial state (3x3, 0 for empty space):")
initial_state = "".join(input().split())
print("Enter the goal state (3x3, 0 for empty space):")
goal_state = "".join(input().split())
max_depth = 50 # You can adjust this as needed
solution = iddfs(initial_state, goal_state, max_depth)
```

if solution:

```
print("\nSolution path:")
for state in solution:

for i in range(0, 9, 3):
    print(" ".join(state[i:i+3]))
    print()
else:
    print("\nNo solution found.")
```

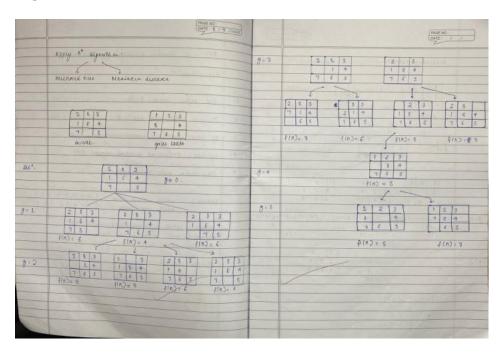
#### **Output:**

```
Samriddhi Singh 1BM23CS295
Enter the initial state (3x3, 0 for empty space):
1 2 3 4 0 5 7 8 6
Enter the goal state (3x3, 0 for empty space):
1 2 3 4 5 6 7 8 0
Searching with depth limit: 0
Searching with depth limit: 1
Searching with depth limit: 2
Solution path:
1 2 3
4 0 5
7 8 6
1 2 3
4 5 0
7 8 6
1 2 3
4 5 6
7 8 0
```

## Program3

Implement A\* search algorithm

## Algorithm:



A* algorithm ( misplaced titles)	
	0 - 1 L = 3, f = 4
	Sty : g = 1, L = 3, f = +
Britistize policythy and and and	2 8 3
entialize printly queue with start state	
	104
While queue is not empty do,	7 6 5
	The state of the same of the s
pep the sease with leavest find = gind + hind	suf 2: 9 = 2, h = 1, f = 5
24 lli goal clate. Setuen sources posts	2 0 3
thermise, generate all raid moves and past there	
to the quice	1 8 4 7 6 s
Non salar all	
Step when the good is reached on no collection is	Step 2: g = 3, h = 2, f = 5
found.	
	18+
	J 6 2.
A STATE OF THE PARTY OF THE PAR	
Sutput 3	Step 4: 9=4. h=1, f=5
entes puzza ciza 3	127
enter initial state . 282	0 8 4
164	7 6 5
715	
	Ittens: 0-5 1 0 0
	Step 5: 9 = 5, h = 0, f = 5
enter goal state: 123	8 0 4
804	765
7 6 5	
	Tatal
sellution path: Step 0: g=0, h=4, f=4	Tetal case : 5
2 8 3	nodes enjarded: 7
164	
7 0 5	

```
A* search algo. [Manhatan Mitane]

Algorithm;

1. Initialize agen list with stars teate, and set its cest fight where give and he Manhattan distance to grad.

2. While agen list is not empty.

3. Fee state with lenset first lie goal, recurrence path generale all varied moves and calls gich values.

4. Generale all varied moves and calls gich values.

5. Add children to egen list y net visited.

6. Culput: - ensue Initial state: 283164705

ensur goal state: 123:04765

6. Selection feurs in 5 moves.

283 283 -> 203

164 -> 104 184

105 765 165

123 123 6 023

304 6 084 184

765 765
```

```
#misplaced tiles
import heapq
from itertools import count
def misplaced_heuristic(board, goal):
  """h(n): number of tiles not in their goal position (excluding blank 0)."""
  n = len(board)
  misplaced = 0
  for i in range(n):
     for j in range(n):
       if board[i][j] != 0 and board[i][j] != goal[i][j]:
          misplaced += 1
  return misplaced
def find_blank(board):
  n = len(board)
  for i in range(n):
     for j in range(n):
       if board[i][j] == 0:
          return i, j
  raise ValueError("Board does not contain a blank tile (0)")
```

```
def neighbors(board):
  n = len(board)
  x, y = find\_blank(board)
  dirs = [(0,1),(0,-1),(1,0),(-1,0)]
  res = []
  for dx, dy in dirs:
     nx, ny = x + dx, y + dy
     if 0 \le nx \le n and 0 \le ny \le n:
        b = [list(row) for row in board]
       b[x][y], b[nx][ny] = b[nx][ny], b[x][y]
       res.append(tuple(tuple(row) for row in b))
  return res
def flatten(board):
  return [x for row in board for x in row]
def inversion_count(seq):
  arr = [x \text{ for } x \text{ in seq if } x != 0]
  inv = 0
  for i in range(len(arr)):
     for i in range(i+1, len(arr)):
        if arr[i] > arr[j]:
          inv += 1
  return inv
def blank_row_from_bottom(board):
  n = len(board)
  for i in range(n):
     for j in range(n):
       if board[i][j] == 0:
          return n - i # 1-indexed from bottom
  raise ValueError("Board does not contain a blank tile (0)")
def is solvable(start, goal):
  """General n-puzzle solvability test (odd/even width)."""
  n = len(start)
  start_flat = flatten(start)
  goal_flat = flatten(goal)
  # Map values to goal indices to compute relative order
  pos = {val: idx for idx, val in enumerate(goal flat)}
  start perm = [pos[val] for val in start flat]
  inv = inversion_count(start_perm)
  if n % 2 == 1:
     # odd grid: inversions parity must be even
     return inv \% 2 == 0
```

```
else:
     # even grid: blank row from bottom parity matters
     blank_row = blank_row_from_bottom(start)
     goal_blank_row = blank_row_from_bottom(goal)
     # When using relative permutation to goal, parity of blank rows must match
     return (inv + blank_row) % 2 == (0 + goal_blank_row) % 2
def reconstruct path(came from, current):
  path = [current]
  while current in came_from:
     current = came_from[current]
     path.append(current)
  path.reverse()
  return path
def a_star_misplaced(start, goal):
  start = tuple(tuple(row) for row in start)
  goal = tuple(tuple(row) for row in goal)
  if len(start) != len(start[0]) or len(goal) != len(goal[0]) or len(start) != len(goal):
    raise ValueError("Initial and goal must be square boards of the same size.")
  # Validate same tile multiset
  start vals = sorted(flatten(start))
  goal_vals = sorted(flatten(goal))
  if start_vals != goal_vals:
    raise ValueError("Initial and goal must contain the same set of tiles.")
  if not is_solvable(start, goal):
    return None, None, 0, 0 # unsolvable
  counter = count() # tie-breaker
  h0 = misplaced_heuristic(start, goal)
  g\_score = \{start: 0\}
  f0 = h0
  open_heap = [(f0, next(counter), start)]
  open\_set = \{start: f0\}
  closed = set()
  came_from = {}
  expansions = 0
  while open heap:
     _, _, current = heapq.heappop(open_heap)
    if current in closed:
       continue
    closed.add(current)
     if current == goal:
```

```
path = reconstruct_path(came_from, current)
       return path, g_score[current], expansions, len(closed)
     expansions += 1
     for nb in neighbors(current):
       tentative_g = g_score[current] + 1
       if nb in closed:
          continue
       if nb not in g_score or tentative_g < g_score[nb]:
          came_from[nb] = current
          g score[nb] = tentative g
         h = misplaced_heuristic(nb, goal)
         f = tentative_g + h
         if nb not in open set or f < open set[nb]:
            heapq.heappush(open_heap, (f, next(counter), nb))
            open\_set[nb] = f
  return None, None, expansions, len(closed)
def read_board(n, prompt):
  print(prompt)
  board = []
  for i in range(n):
    row = list(map(int, input().split()))
     if len(row) != n:
       raise ValueError(f"Row {i+1} must contain exactly {n} integers.")
     board.append(row)
  return board
def print board(board):
  for row in board:
     print(" ".join(f"{x}" for x in row))
def main():
  try:
     n = int(input("Enter puzzle size n (e.g., 3 for 3x3): ").strip())
     initial = read_board(n, "Enter initial state row by row (use 0 for blank):")
     goal = read_board(n, "Enter goal state row by row (use 0 for blank):")
    result = a_star_misplaced(initial, goal)
     path, cost, expansions, explored = result
     if path is None:
       print("No solution (unsolvable with given start/goal).")
       return
     print("\nSolution path (each state shows g, h, f):\n")
     for idx, state in enumerate(path):
       g = idx \# each step costs 1
```

```
h = misplaced_heuristic(state, tuple(tuple(r) for r in goal))
f = g + h
print(f"Step {idx}: g={g}, h={h}, f={f}")
print_board(state)
print()

print(f"Total cost (number of moves): {cost}")
print(f"Nodes expanded: {expansions}")
print(f"Nodes explored (unique): {explored}")
print("Samriddhi Singh,1BM23CS29517")

except Exception as e:
    print("Error:", e)

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

### **Output:**

```
Enter puzzle size n (e.g., 3 for 3x3): 3
Enter initial state row by row (use 0 for blank):
2 8 3
1 6 4
7 0 5
Enter goal state row by row (use 0 for blank):
1 2 3
8 0 4
7 6 5

Solution path (each state shows g, h, f):

Step 0: g=0, h=4, f=4
2 8 3
1 6 4
7 0 5

Step 1: g=1, h=3, f=4
2 8 3
1 0 4
7 6 5

Step 2: g=2, h=3, f=5
2 0 3
1 8 4
7 6 5

Step 3: g=3, h=2, f=5
0 2 3
1 8 4
7 6 5

Step 4: g=4, h=1, f=5
1 2 3
0 8 4
7 6 5

Step 5: g=5, h=0, f=5
1 2 3
8 0 4
7 6 5

Total cost (number of moves): 5
Nodes expanded: 6
Nodes expanded: 6
Nodes expanded: 10
Nodes expanded (unique): 7
Samriddhi Singh, 18M23CS29517
```

#### **Code:**

#MANHATTAN DISTANCE

import heapq

def manhattan(state, goal):

```
dist = 0
  for i in range(9):
     if state[i] != 0:
       x1, y1 = divmod(i, 3)
       j = goal.index(state[i])
       x2, y2 = divmod(j, 3)
       dist += abs(x1 - x2) + abs(y1 - y2)
  return dist
def get_neighbors(state):
  neighbors = []
  i = state.index(0)
  x, y = div mod(i, 3)
  moves = [(-1,0),(1,0),(0,-1),(0,1)]
  for dx, dy in moves:
     nx, ny = x + dx, y + dy
     if 0 \le nx \le 3 and 0 \le ny \le 3:
       j = nx*3 + ny
       new_state = list(state)
       new_state[i], new_state[j] = new_state[j], new_state[i]
       neighbors.append(tuple(new_state))
  return neighbors
def a_star(start, goal):
  open_heap = [(manhattan(start, goal), 0, start, [])]
  visited = set()
  while open heap:
     f, g, state, path = heapq.heappop(open_heap)
     if state == goal:
       return path + [state], g
     if state in visited: continue
     visited.add(state)
     for nb in get_neighbors(state):
       if nb not in visited:
          new_g = g + 1
          new_f = new_g + manhattan(nb, goal)
          heapq.heappush(open heap, (new f, new g, nb, path + [state]))
  return None, -1
print("Enter initial state (9 numbers, 0 for blank):")
start = tuple(map(int, input().split()))
print("Enter goal state (9 numbers, 0 for blank):")
goal = tuple(map(int, input().split()))
path, cost = a\_star(start, goal)
if path:
  print("\nSolution found in", cost, "moves\n")
  for step, p in enumerate(path):
     print("Step", step, "g=", step, "h=", manhattan(p, goal), "f=", step + manhattan(p, goal))
```

```
for i in range(0, 9, 3):
    print(p[i:i+3])
    print()
else:
    print("No solution found")
print("Samriddhi Singh,1BM23CS295")
```

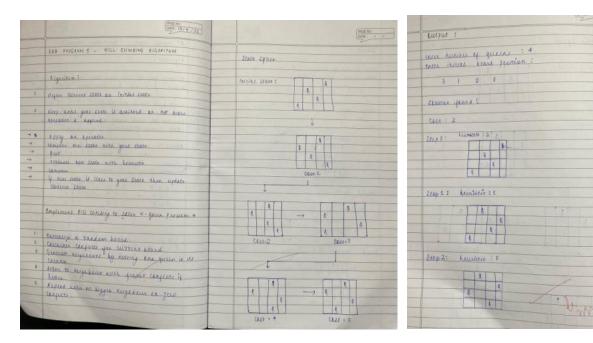
### **Output**:

```
Enter initial state (9 numbers, 0 for blank): 2 8 3 1 6 4 7 0 5 \,
Enter goal state (9 numbers, 0 for blank):
1 2 3 8 0 4 7 6 5
Solution found in 5 moves
Step 0 g= 0 h= 5 f= 5
(2, 8, 3)
(1, 6, 4)
(7, 0, 5)
Step 1 g= 1 h= 4 f= 5
(2, 8, 3)
(1, 0, 4)
(7, 6, 5)
Step 2 g= 2 h= 3 f= 5 (2, 0, 3) (1, 8, 4) (7, 6, 5)
Step 3 g= 3 h= 2 f= 5 (0, 2, 3)
(1, 8, 4)
(7, 6, 5)
Step 4 g= 4 h= 1 f= 5
(1, 2, 3)
(0, 8, 4)
(7, 6, 5)
Step 5 g= 5 h= 0 f= 5
(1, 2, 3)
(8, 0, 4)
(7, 6, 5)
Samriddhi Singh,1BM23CS295
```

### Program4

Implement Hill Climbing search algorithm to solve N-Queens problem

## Algorithm:



#### **Code:**

import random

for j in range(i + 1, n):

new\_board = list(board)

neighbors.append(new\_board)

new\_board[i], new\_board[j] = new\_board[j], new\_board[i]

```
return neighbors
```

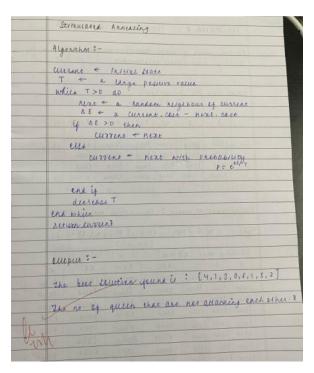
```
def hill_climbing(board, max_sideways=0): # sideways moves disabled
  steps = 0
  sideways\_moves = 0
  current_heur = heuristic(board)
  path = [board[:]]
  while True:
    if current_heur == 0:
       return board, steps, path
     neighbors = get_neighbors(board)
    neighbor_heuristics = [(neighbor, heuristic(neighbor)) for neighbor in neighbors]
    best_heur = min(h for _, h in neighbor_heuristics)
    best_neighbors = [nb for nb, h in neighbor_heuristics if h == best_heur]
    if best_heur > current_heur:
       return None, steps, path
    next_board = random.choice(best_neighbors)
    if best_heur < current_heur:
       sideways\_moves = 0
     elif best_heur == current_heur:
       sideways_moves += 1
       if sideways_moves > max_sideways:
         return None, steps, path
     board = next_board
     current heur = best heur
     path.append(board[:])
    steps += 1
def print_board(board):
  n = len(board)
  for row in range(n):
    line = ""
    for col in range(n):
       line += "Q " if board[col] == row else ". "
    print(line)
  print()
def main():
  n = int(input("Enter the number of queens (N): "))
```

```
print(f'Enter the initial board positions (row for each queen in column 0 to \{n-1\}):")
   print(f"Rows should be between 0 and {n-1}, space separated.")
   board_input = input()
   try:
      board = list(map(int, board_input.strip().split()))
   except ValueError:
      print("Invalid input format.")
      return
   if len(board) != n or any(r < 0 \text{ or } r >= n \text{ for } r \text{ in board}):
      print("Invalid board input.")
      return
   solution, cost, path = hill_climbing(board)
   if solution:
      print("\nSolution found!\n")
   else:
      print("\nNo solution found (stuck in local minimum).\n")
   print(f"Cost (steps taken): {cost}\n")
   print("Steps to reach solution:")
   for step_num, state in enumerate(path):
      print(f"Step {step_num}: heuristic = {heuristic(state)}")
      print_board(state)
   print("Samriddhi Singh,1BM23CS295")
if __name__ == "__main__":
   main()
Output:
Enter the number of queens (N): 4 Enter the initial board positions (row for each queen in column 0 to 3): Rows should be between 0 and 3, space separated. 3 2 0 1
Solution found!
Cost (steps taken): 2
Steps to reach solution:
Step 0: heuristic = 2
. . Q .
. . . Q .
Q . . . .
Step 1: heuristic = 1
Step 2: heuristic = 0
. Q . .
Samriddhi Singh,1BM23CS295
```

## Program 5

Simulated Annealing to Solve 8-Queens problem

## **Algorithm:**



```
import random, math
def cost(state):
  attacks = 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):
          attacks += 1
  return attacks
def neighbor(state):
  n = len(state)
  new_state = 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, T=1000, cooling=0.99):
```

```
current = [random.randint(0, n-1) for _ in range(n)]
  while T > 1e-6:
    next_state = neighbor(current)
    deltaE = cost(current) - cost(next_state)
    if deltaE > 0:
       current = next_state
     else:
       if random.random() < math.exp(deltaE / T):
         current = next_state
    T *= cooling
    if cost(current) == 0:
       break
  return current
solution = simulated_annealing(8)
print("The best position found is:", solution)
print("The number of queens that are not attacking each other is:", 8 if cost(solution) == 0 else 8 -
cost(solution))
print("Samriddhi Singh,1BM23CS295")
Output:
The best position found is: [4, 7, 3, 0, 6, 1, 5, 2]
The number of queens that are not attacking each other is: 8
Samriddhi Singh,1BM23CS295
```

# Program 6

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

# Algorithm:

THE							The state of	NE NO		PAGE NO.	
										CATE: / /	
	LAB MOGRAM & - PROPERTIONAL LOGIC										
										Algorithm t-	
	Sustemments of truck take commission algorithm for								4	celled all presentered separately from the recentage bale (165) and the going (16)	
									1-		
									1		
	Truth were for connectives -									Generale till prairies truck seripament for three symbols	
	1						-		3	Fire cash numer, there is to the come.	
	1	2		¬P	PAR	1AR PUB 1458					
			1	300	F.AUE.	7.8.0	1658	+	4.	of 68 is one, then sheet potenties & it true.	
		au	face	SALLA	false	Pale	eque.	-	4		
		False.	tene	tanz	falce	time	tauce		3.	24 any moder more to true the 5 years -	
	3	thic	tous	fein	Pares	take	false		-	Active tails (no entatiment)	
		true	LAWE	false	this	+9.00	true		1.		
									-	ey in every model where PE to true, & is also expenses as Autore True ( ** entries **)	
										Auture Thus ( ts entails *)	
	Inequationes inquience: Enumeration records									Que pice :	
										LANGE WHOMPHAGE HOLE (OAK (O) AC) (O' B (MACC))	
	Y. K. ANS KB. (NC)A(NAC)										
										Intel every (CELAR)	
	Chierry that KEFK								_	Express ( CA & C )	
	fater 1			AVE		BV-10	5.8	×			
				true			false	false	-		
	TRAM	time	The Wood				face	12224	-		
	TRIAL	tius	true	truck	-		fasse	63.00			
	take -	falls but			-		Esua	type	-		
	17,44	falor	1 his	the	- 2		rue	trac	-		
	CEANA	true	6024	+540	- 10		alce	true			
	File	THE	18166	±500		1000	rile	true	-		
					-	1	rue	true			

	PAGE NO.		FANCE HID (DATE)
A C 6 KE	K	-	0.12011 222
TARE THE THE THE	Thur	1	service 167 as variable and fellowing THE
There Pales Thus Pales	Thus		A. + (177)
Thise Table Falce Thise	True		b: (cht)
TAME FALSE FALSE TAME	Thue	-	6. TV.FT
raise True True Thue	TRUE	-	
Falls Pales Take False	False	-	white south table I than whiches
fairs Thee fairs Pales	True	0	
Fales face face face	False	1	A execute 6
		11	AC SWERCELD
Ores to caran X ? ! Tour		10	T T FSUT CAT
		1	1 1 TATE CALL
			8 8 1 8
			0 1 0 0 0 0
	1		1 0 0 n descrit
	m		1 1 1 Entail
			- Creation
		-	
SECTION AND VALUE OF THE PARTY		313	J T F(IVI) TVFT
			1 1 1
		/	0 1 9
			1 0 0
		1	Part of the second seco
		W	
			L1= 0
			a engle s

```
def parse_expr(tokens):
   token = tokens.pop(0)
   if token == '(':
        op = tokens.pop(0)
```

```
args = []
     while tokens[0] != ')':
       args.append(parse_expr(tokens))
     tokens.pop(0) # Remove ')'
     return (op, *args)
  else:
     return token
def tokenize(s):
  return s.replace('(', ' ( ').replace(')', ' ) ').split()
def tt entails(kb, alpha):
  symbols = list(get_symbols(kb) | get_symbols(alpha))
  print("Symbols:", symbols)
  header = symbols + ['KB', '\alpha']
  print("\t".join(header))
  # Start recursive check
  result = tt_check_all(kb, alpha, symbols, {})
  return result
def tt_check_all(kb, alpha, symbols, model):
  if not symbols:
     kb_val = pl_true(kb, model)
     alpha_val = pl_true(alpha, model)
     # Print current model and values
     row = [str(model.get(s, False)) for s in sorted(model.keys())] + [str(kb_val), str(alpha_val)]
     print("\t".join(row))
     if kb val:
       return alpha_val
     else:
       return True
  else:
     rest = symbols[1:]
     symbol = symbols[0]
     model_true = model.copy()
     model_true[symbol] = True
     model_false = model.copy()
     model false[symbol] = False
     return (tt_check_all(kb, alpha, rest, model_true) and
          tt_check_all(kb, alpha, rest, model_false))
def get_symbols(expr):
  if isinstance(expr, str):
     return {expr}
  elif isinstance(expr, tuple):
     symbols = set()
     for part in expr[1:] if expr[0] != 'not' else [expr[1]]:
       symbols |= get_symbols(part)
```

```
return symbols
  else:
    return set()
def pl_true(expr, model):
  if isinstance(expr, str):
    return model.get(expr, False)
  op = expr[0]
  if op == 'and':
    return all(pl_true(arg, model) for arg in expr[1:])
  elif op == 'or':
    return any(pl true(arg, model) for arg in expr[1:])
  elif op == 'not':
    return not pl_true(expr[1], model)
  elif op == 'implies':
    return (not pl_true(expr[1], model)) or pl_true(expr[2], model)
  else:
    raise ValueError(f"Unknown operator: {op}")
kb_input = input("Enter knowledge base (e.g. (and A (or B C))): ")
alpha_input = input("Enter query (e.g. A): ")
kb = parse_expr(tokenize(kb_input))
alpha = parse_expr(tokenize(alpha_input))
result = tt_entails(kb, alpha)
print(f"\nDoes KB entail \alpha? : {result}")
Output:
Enter knowledge base (e.g. (and A (or B C))): (and(or A C) (or B(not C)))
Enter query (e.g. A): (or A B)
Symbols: ['C', 'A', 'B']
                        KΒ
        Α
                В
True
        True
                True
                        True
                                True
True
        False True
                       False True
False True
                True
                        True
                                True
False False True
                        False False
True
        True
                False
                       True
                                True
True
        False False True
                                True
False True
                False False True
False False False False
```

Does KB entail α? : True Samriddhi Singh,1BM23CS295

## Program 7

Implement unification in first order logic

## Algorithm:

	LAB PROGRAM 7: Unification Algorithm		PAGE NO COST 1 1
	Unification > 11 is		1 = 2/4
	Uniquetien > 20 is a process to find Estatumente that make difference FOI ( Extra Order light)		Execut P (6,11/2,11/3)
	A STATE OF THE STA		1 = ×/ 11/2
1	Unity (Knows (John , 2), Knows (John , Jane)		1 = 1/3(2)
	unify ( Knew ( Jens, Jane), Knew (Jens,		11( x, Ryy), z: 6, x: +(x), y: g(z))}
2	unify ( Knows ( Jehn, 2), Knows (y, 514)	5.4	\$ (a, f(+(+), a), 2) \$
	0 = 4/ Jehn	ð s	FCX.XX.1. ((4)), 1(4)), 1(X.X.)
	Knowe ( Tehn, 2), Knows ( John, 516)		-> faita
	0 = %/ siu		
	Knows & C Tehn, Bill J, Knows (Tohn, BILL)	1414-	0 = 416
Ŋ 3	Find mess genual unifier ex		%: ((b) y:b
	( p ( b, x , f ( g (z) ) ) } ( p ( z , + (y) , + ( Y) ) }		

	Algorithm:	(MEC AD)
1	NIL C no change )	turpat :-
2:	If one is a variable, replace it with the ten	enses ifeses enquesion: p(b, x, f(g(=)))
3.	If the Main Marries to make the time	ence seeme exercise : 1(x, f(x), f(x))
	es es main question ex predicate name appe	men general unifies: ['2': 16', 'x': 'f(x)',
4	of they have a different no of arguments a stude	V
5-	ethernice, unity each corresponding argument era up	
€-	Keep applying and updating substitution as yough	- He
7.	The final set of subscriptions chained to new General unifer.	

```
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
     return 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()}
  formatted\_subst = None
print("Most General Unifier (MGU):", formatted_subst)
print("Samriddhi Singh,1BM23CS295")
```

### **Output:**

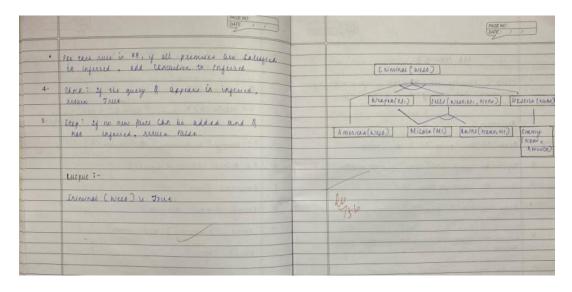
```
Enter first expression: p(b,X,f(g(Z)))
Enter second expression: p(Z,f(Y),f(Y))
Most General Unifier (MGU): {'Z': 'b', 'X': 'f(Y)', 'Y': 'g(Z)'}
Samriddhi Singh,1BM23CS295
```

## Program 8

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

## Algorithm:

	2 11 11 201		
			PAGE NO:
	LAB PROGRAM 8 : First Ender Logic		
		1	V2, y-1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
	Ceres a removed base crowsing of fire cran		HERE ( 13)
	forward reservey.		- Chinanal (n)
		2	** MARIA (*) A more ( News, 2) & tells ( mus. 2., News)
	f => A		New)
bullet:-		3	
	Ant a L	4	VX Nicele (Y) & Magazin (Y)
	AARD L	5	American Court
ALU:-	A L JM	6.	Eurny ( Mene, America)
	8 9	- 4	
	Face 1 A B	- 1	
4.			No. 1
2	The say says there is in a crime for an impurer		Algorithms !
	Mane on the country	1-	sages : Kennietza Base (KB) with FOI Watermers and a group of
	America ceres as "kaseres he enemy of	2	tamage to approx . It to see crack ex
3	Prove that west it comment		Lyu:



```
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("Samriddhi Singh,1BM23CS295")

Output:
```

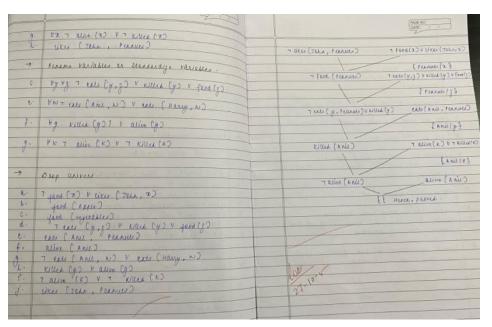
Robert is a criminal. Samriddhi Singh 1BM23CS295

## Program 9

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

### **Algorithm:**

LAB PROGRAM 9:		
		(MATE / /
crace a knowing ball century of commence and prome going guilty		
ceauments and prone given of	free eras	Honey care concepting that half are "Ya care (ARR. R) - Aryten while is not proper one stifter "Ya while (2) - alone configuration has been been as a single configuration of the
Accounties query	using light .	hayen who is allo inster one sites we knee (2) - ales
		Rayene who is not killed topics deine to acretis > 7
Algorithm:		X.III.ed. (X
		PAGE LUCY CONTRACTOR C
tel 1. Inque KB and query (8) to be prese		Press by reselvered that :
to be priese	the h	Tele like permer Wer (Tela, Permer)
2. Negate the account (at)		Tehr like france Uker (Taha, France)
2. Negate the query (NS) and add it	ta x8.	
3: Censeux all statements into CNF.	-	Eliminate Implication -
STREET WES CNF		
4- 1		K⇒B NIEL TKVB
4. Apply unification to your complementary between clouses.	Tierrati	
between Clauses.	R.	YX , FROM (X) V DRES ( TELL , X)
5.	6.	FROM (ALPRE) A LAND C
5. Reselve the success clauses to produce	0.	
daws.	e hen d.	RACE RACE PERSONS A MILES DATE OF
	6	CARL AND TO V SEE HAVE A
6. 34 an emen done 612 to	- F.	Va 7 [7 Killed (2)] V allive (2)
The street (2) is desired,	reacted.	42 1 alive (2) V 7 Killed (2)
presen true; exhausin nex es	neader. h	like ( Tohn , persue)
	7	Men regetien (7) inwards and sensite
Example !-		
	2	Y27 feat(2) V UKE (JELA, 2)
Teka likes all kind of feel yx: 10	Lod (2) - Liker (Tohnis) 6-	
appe and regulables are good from ( ap)	ree) a feed (regressivi) to	ford frages & yeard (vegenasies)
Abouting Delivery	cuc(243) - d.	Cate ( Anie , France ) ^ aux (Anie)
	COCO LUGUA	this thirt, takes y was three
	CALLS A LIMITAL	1/2 - Killed (2) 1 v aure (2)
and the peaner and will alive the ("A	NIL, PLANIEU JO	A. K. KILLER ( A.) 1 A MORE ( A.)
aliv	e (Carico	



```
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)
```

```
QUERY")
  print("
  print("="*55)
  print(f" Prove: {query}")
  print(f" Negated Query: \sim{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 (∀).")
  print("Step 3: Add negated query (~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")
  # Simulated resolution steps for John likes peanuts problem
  print("="*55)
  print("
                   RESOLUTION TREE")
  print("="*55)
  print("""
                [~Likes(John, Peanuts)]
                 [Food(Peanuts) \rightarrow Likes(John, Peanuts)]
                 [Eats(Anil, Peanuts) \land \neg Killed(Anil) \rightarrow Food(Peanuts)]
                [Alive(Anil) \rightarrow \neg Killed(Anil)]
                 [Alive(Anil)]
                  ⊥ (Contradiction Found)
  ("""
  print("="*55)
  print(f"Therefore, the query '{query}' is PROVEN by Resolution.")
  print("="*55 + "\n")
print("\n FIRST ORDER LOGIC - RESOLUTION METHOD")
print("----")
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("Samriddhi Singh 1BM23CS295")
```

### **Output:**

```
Enter the number of statements in the Knowledge Base: 9
Enter each statement (e.g., 'vx: Food(x) → Likes(John, x)'):

KB[1]: ∀x: Food(x) → Likes(John, x)

KB[2]: Food(Apple)

KB[3]: Food(Vegetables)
                                                                                                                                         Prove: Likes(John, Peanuts)
Negated Query: ~Likes(John, Peanuts)
KB[3]: ∀xy: (Eats(x, y) ∧ ¬Killed(x)) → Food(y

KB[5]: Eats(Anil, Peanuts)

KB[6]: Alive(Anil)

KB[7]: ∀xy: Eats(Harry, y) ← Eats(Anil, y)

KB[8]: ∀x: Alive(x) → ¬Killed(x)

KB[9]: ∀x: ¬Killed(x) → Alive(x)
                                                                                                                                                                       RESOLUTION PROCESS
                                                                                                                                            Step 1: Convert all implications (-) to CNF (Conjunctive Normal Form).
Step 2: Eliminate all universal quantifiers (\nabla).
Step 3: Add negated query (~Query) to the KB.
Step 4: Apply resolution rule between matching clauses.
Step 5: Continue until the empty clause (1) is found.
 Enter the query to prove: Likes(John, Peanuts)
                                                                                                                                             RESOLUTION TREE
                            KNOWLEDGE BASE
                                                                                                                                                                                          [~Likes(John, Peanuts)]
   1. \forall x: Food(x) \rightarrow Likes(John, x)
2. Food(Apple)
                                                                                                                                                                                          [Food(Peanuts) → Likes(John, Peanuts)]

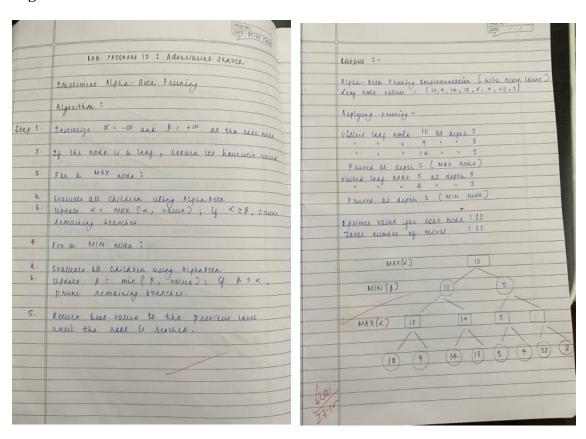
    Food(Vegetables)

   3. Food(Vegetables)
4. ∀x, y: (Eats(x, y) ∧ -Killed(x)) → Food(y
5. Eats(Anil, Peanuts)
6. Alive(Anil)
7. ∀x, y: Eats(Harry, y) ← Eats(Anil, y)
8. ∀x: Alive(x) → -Killed(x)
9. ∀x: -Killed(x) → Alive(x)
                                                                                                                                                                                          [Eats(Anil, Peanuts) \land \neg Killed(Anil) \rightarrow Food(Peanuts)]
                                                                                                                                                                                          [Alive(Anil) → ¬Killed(Anil)]
                                                                                                                                                                                          [Alive(Anil)]
                                                                                                                                                                                            ⊥ (Contradiction Found)
        QUERY
                                                                                                                                              Therefore, the query 'Likes(John, Peanuts)' is PROVEN by Resolution.
   Prove: Likes(John, Peanuts)
Negated Query: ~Likes(John, Peanuts)
                                                                                                                                             Samriddhi Singh 1BM23CS295
```

#### Program 10

Implement Alpha-Beta Pruning.

### Algorithm:



```
moves = 0
def alphabeta(depth, nodeIndex, maximizingPlayer, values, alpha, beta):
  global moves
  moves += 1
  if depth == 3:
    print(f"Visited leaf node {values[nodeIndex]} at depth {depth}")
    return values[nodeIndex]
  if maximizingPlayer:
    best = float('-inf')
    for i in range(2):
       val = alphabeta(depth + 1, nodeIndex * 2 + i, False, values, alpha, beta)
       best = max(best, val)
       alpha = max(alpha, best)
       if beta <= alpha:
         print(f" Pruned at depth {depth} (MAX node)")
         break
    return best
  else:
    best = float('inf')
    for i in range(2):
       val = alphabeta(depth + 1, nodeIndex * 2 + i, True, values, alpha, beta)
       best = min(best, val)
       beta = min(beta, best)
       if beta <= alpha:
         print(f" Pruned at depth {depth} (MIN node)")
         break
    return best
print("Alpha–Beta Pruning Implementation (With Move Count)")
values = [10, 9, 14, 18, 5, 4, 50, 3]
print("Leaf node values:", values)
alpha = float('-inf')
beta = float('inf')
print("\nApplying Alpha–Beta Pruning...\n")
optimal_value = alphabeta(0, 0, True, values, alpha, beta)
print("\n----")
```

```
print(f" Optimal value for the root node: {optimal_value}")
print(f" Total number of moves (nodes evaluated): {moves}")
print("-----")
print("Samriddhi Singh 1BM23CS295")
```

### **Output:**

```
Alpha-Beta Pruning Implementation (With Move Count)
Leaf node values: [10, 9, 14, 18, 5, 4, 50, 3]

Applying Alpha-Beta Pruning...

Visited leaf node 10 at depth 3
Visited leaf node 9 at depth 3
Visited leaf node 14 at depth 3
Pruned at depth 2 (MAX node)
Visited leaf node 5 at depth 3
Visited leaf node 4 at depth 3
Pruned at depth 1 (MIN node)

Optimal value for the root node: 10
Total number of moves (nodes evaluated): 11

Samriddhi Singh 1BM23CS295
```