Sr NO	Date	Topic	Sign
06	04/08/25	Missionaries and Cannibals Number Puzzle	

Q.1) Missionaries and Cannibals

→
```
moves = [(2,0), (0,2), (1,1), (1,0), (0,1)]
def is_valid (m_left, c_left, m_right, c_right):
    if m_left <0 or c_left <0 or m_right <0 or c_right <0:
        return False
    if (m_left >0 and m_left < c_left) or (m_right >0 and m_right < c_right):
        return False
    return True
def solve ():
    from collections import deque
    start=(3, 3, 1)
    goal = (0, 0, 0)
    queue = deque()
    queue.append ((start, [start]))
    visited = set()
    while queue:
        (m_left, c_left, boat), path= queue.popleft ()
        if (m_left, c_left, boat) in visited:
            continue
        visited.add ((m_left, c_left, boat))
        if (m_left , c_left , boat) == goal:
            return path
        for m , c in moves:
            if boat == 1:
                new_m_left = m_left - m
                new_c_left = c_left - c
                new_boat = 0
            else:
                new_m_left = m_left + m
                new_c_left = c_left + c
```

```
            new_boat = 1
        new_m_right = 3 - new_m_left
        new_c_right = 3 - new_m_left
        if is_valid(new_m_left, new_c_left, new_m_right, new_c_right):
            new_state not in visited:
                queue.append((new_state, path + [new_state]))
    return None
steps = solve()
if steps:
    for i, (m, c, b) in enumerate(steps):
        side = "Left" if b == 1 else "Right"
        print(f"Step {i}: Missionaries left: {m}, Cannibals left: {c}, Boat on: {b}")
else:
    print("No solution found.");
```

O/P:-

```
Step 0: Missionaries left: 3, Cannibals left: 3, Boat on: 1
Step 1: Missionaries left: 3, Cannibals left: 1, Boat on: 0
Step 2: Missionaries left: 3, Cannibals left: 2, Boat on: 1
Step 3: Missionaries left: 3, Cannibals left: 0, Boat on: 0
Step 4: Missionaries left: 3, Cannibals left: 1, Boat on: 1
Step 5: Missionaries left: 1, Cannibals left: 1, Boat on: 0
Step 6: Missionaries left: 2, Cannibals left: 1, Boat on: 1
Step 7: Missionaries left: 0, Cannibals left: 1, Boat on: 0
Step 8: Missionaries left: 1, Cannibals left: 1, Boat on: 1
Step 9: Missionaries left: 0, Cannibals left: 0, Boat on: 0
```


Q.2) Number Puzzl
→ From collections
goal = '12345
moves = {
0: [1,3]
1: [0,2,
2: [1,5]
3: [0,
4: [1,3
5: [2,
6: [3
7: [4
8: [5
}
def bfs(sta
visited =
queue =
while q
st
if
i
[move],

Q.2) Number Puzzle

→
```
from collections import deque
goal = '123456780'
moves = {
    0: [1,3],
    1: [0,2,4],
    2: [1,5],
    3: [0,4,6],
    4: [1,3,5,7],
    5: [2,4,8],
    6: [3,7],
    7: [4,6,8],
    8: [5,7]
}
def bfs(start):
    visited = set()
    queue = deque([(start, [])])
    while queue:
        state, path = queue.popleft()
        if state == goal:
            return path + [state]
        if state in visited:
            continue
        visited.add(state)
        zero = state.index('0')
        for move in moves[zero]:
            new_state = list(state)
            new_state[zero], new_state[move] = new_state
[move], new_state[zero]
        while queue:
```

Sr NO	Date	Topic	Sign
01	30/06/25	Search Algorithm	DATE:

a) WAP to implement BFS Algorithm

Code:-

```
from queue import Queue
adj_list = {
    'A' : ['B', 'D'],
    'B' : ['C', 'F'],
    'C' : ['E', 'G', 'H'],
    'G' : ['E', 'H'],
    'E' : ['B', 'F'],
    'F' : ['A'],
    'D' : ['F'],
    'H' : ['A'],
}
visited = {}
level = {}
parent = {}
bfs_traversal = []
queue = Queue()
for node in adj_list.keys():
    visited[node] = False
    parent[node] = None
    level[node] = -1
source = "A"
visited[source] = True
level[source] = 0
queue.put(source)
while not queue.empty():
    u = queue.get()
    bfs_traversal.append(u)
```

Q.2)

→ from coll

```
    state, path = queue.
     queue.append((' '.join(new_state), path + [state]))
  return None
start = '12345678'
solution = bfs(start)
if solution:
    print("Steps to solve:")
    for s in solution:
        print(s[0:3])
        print(s[3:6])
        print(s[6:9])
        print("------")
else:
    print("No solution found.")
```

o/p:-

Steps to solve

```
1 2 3      1 2 3      1 2 3      1 2 3
4 0 5      0 5 8      5 6 0      4 5 6
6 7 8      4 6 7      4 7 8      7 8 0
-----      -----      -----
1 2 3      1 2 3      1 2 3
4 5 0      5 0 8      5 0 6
6 7 8      4 6 7      4 7 8
-----      -----      -----
1 2 3      1 2 3      1 2 3
4 5 8      5 6 8      0 5 6
6 7 0      4 0 7      4 7 8
-----      -----      -----
1 2 3      1 2 3      1 2 3
4 5 8      5 6 8      4 5 6
0 6 7      4 7 0      7 8 0
-----      -----      -----
```


Q.1) Shu
imp
suits
ranks
'Ace'
deck
rando
print
for ca

o/p:-
Shuff
4 of
2 of
8 of
9 of
6 of
Ace o
3 o
6 o
4 o
6 o
King
Ace o
10 o
King
9 o
9 o
10

Sr No	Date	Topic	Sign
07	11/08/25	Shuffle Deck of cards Tic-tac-toe game	

Q.1) Shuffle Deck of cards

```
import random
suits = ["Hearts", 'Diamonds', 'Clubs', 'spades']
ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'king',
'Ace']
deck = [rank + " of " + suit for suit in suits for rank in ranks]
random.shuffle (deck)
print ("shuffled Deck of Cards: ")
for card in deck:
    print (card)
```

O/P:-

Shuffled deck of cards:
4 of Spades
2 of Clubs
8 of Clubs
9 of Hearts
6 of Clubs
Ace of Hearts
3 of Diamonds
6 of Diamonds
4 of Diamonds
6 of Spades
King of Spades
Ace of Spades Diamonds
10 of Spades
King of Diamonds
9 of Diamonds
9 of Spades
10 of Diamonds

DATE:

```
for v in adj_list[u]:
  if not visited[v]:
     visited[v] = True
     parent[v] = u
     level[v] = level[u]+1
     queue.put(v)
print("Bfs Traversal:", bfs_traversal)
node = "E"
path = []
while node is not None:
  path.append(node)
  node = parent[node]
path.reverse()
print("Shortest path is:", path)
```

O/P

```
BFS Traversal: ['A', 'B', 'D',
'C', 'F', 'E', 'G', 'H']
Shortest Path: ['A', 'B', 'F']
```

b) DFS

Code:-

```
g = {
'A': ['B', 'D'], 'B': ['C', 'F'], 'C': ['E', 'G', 'H'], 'G': ['E', 'H'],
'E': ['B', 'F'], 'F': ['A'], 'D': ['F'], 'H': ['A'],
}
def dfs(g, n, seen, d):
  if n not in seen:
    for i in g[n]:
      if seen[-1] is d:
        break
      dfs(g, i, seen, d)
    return seen
print(dfs(g, 'A', [], 'D'))
```

O/P:-

```
['A', 'B', 'C', 'E', 'F', 'G', 'H']
```

FOR EDUCATIONAL USE

Sr No 02

A) Write a
We cre
Checks
• San
• San
• Up
• Up
Code:
Func
def pr
for
p
Ch
def
#

2 of Diamonds
8 of Spades
4 of Hearts

Q.2) Tic-tac-toe game

→
```
board = [' ' for _ in range(9)]
player = 'X'
def show_board():
    print(f" {board[0]} | {board[1]} | {board[2]} ")
    print("-----------")
    print(f" {board[3]} | {board[4]} | {board[5]} ")
    print("-----------")
    print(f" {board[6]} | {board[7]} | {board[8]} ")
    print("-----------")
def is_winner(p):
    return(
        (board[0] == p and board[1] == p and board[2] == p) or
        (board[3] == p and board[4] == p and board[5] == p) or
        (board[6] == p and board[7] == p and board[8] == p) or
        (board[0] == p and board[3] == p and board[6] == p) or
        (board[1] == p and board[4] == p and board[7] == p) or
        (board[2] == p and board[5] == p and board[8] == p) or
        (board[0] == p and board[4] == p and board[8] == p) or
        (board[2] == p and board[4] == p and board[6] == p)
    )
def is_tie():
    return ' ' not in board
def game():
    global player
    while True:
```

Sr No	Date	Topic	Sign
02	07/01/25	N-Queens and tower of Hanoi Problem	

A) Write a program to simulate N-Queen problem.
We create a 4 x 4 board filled with '.' which means empty.
Checks if placing a queen at (row, col) is safe by checking:

- Same column above
- Same row
- Upper left diagonal
- Upper right diagonal

Code:

```
# function to print the board
def print_board(board):
    for row in board:
        print(" ".join(row))
    print()

# Check if it's safe to place the queen
def is_safe(board, row, col, n):
    # check column
    for i in range(row):
        if board[i][col] == 'Q':
            return False

    # Check upper-left diagonal
    i, j = row, col
    while i >= 0 and j >= 0:  # within the top row i,
                              # within the leftmost column j
        if board[i][j] == 'Q':
            return False
        i -= 1  # more one row up
        j -= 1  # more one column left
```

```
show_board()
try:
    move = int(input(f"Player {player} enter 0-8: "))
    if 0 <= move <= 8 and board[move] == ' ':
        board[move] = player
        if is_winner(player):
            show_board()
            print(f"Player {player} wins!")
        elif istie():
            show_board()
            print("It's a Tie!")
            break
        if player == 'X':
            player = 'O'
        else:
            player = 'X'
    else:
        print("Invalid Move")
except(ValueError, IndexError):
    print("Invalid input. Enter a number 0-8")
```

Output :-

```
|   |   |   |
|   |   |   |
|   |   |   |
Player X enter 0-8 : 0
|X|   |   |
|   |   |   |
|   |   |   |
Player O enter 0-8: 2
```

```
# check upper-right diagonal
i, j = row, col
while i >= 0 and j < n:  # Keep moving upwards and to right
    if board[i][j] == 'Q':
        return False
    i -= 1
    j += 1
return True

# backtracking function
def solve_queens(board, row, n):
    if row == n:
        print_board(board)
        return True  # return true if one solution is enough

    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 'Q'  # place queen
            if solve_queens(board, row+1, n):
                return True  # return to next row
            board[row][col] = '.'  # backtrack (remove queen)

# Main function
def four_queens():
    n = 4
    board = [['.' for _ in range(n)] for _ in range(n)]
                                    # _ is throwback variable
    if not solve_queens(board, 0, n):
        print("No solution found.")
```

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

Player X enter 0-8: 3

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

Player O enter 0-8: 4

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

Player X enter 0-8: 5

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

Player O enter 0-8: 7

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

Player O wins!

```
# Run the program
four_queens()
```

Output:-

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

B) Write a program to solve tower of Hanoi problem
https://www.mathsisfun.com/games/towerofhanoi.html
Move n disks from source rod to destination rod using an auxiliary rod, obeying these rules:

1. Only one disk can be moved at a time.
2. A larger disk can be moved at a
3. Only the top disk can be moved.

Code:-

```
def tower_of_hanoi(n,a,b,c):
    if n==1:
        print(f"Move disk1 from {a} to {c}")
        return
    tower_of_hanoi(n-1, a,c,b)
    print(f"Move disk {n} from {a} to {c}")
    tower_of_hanoi(n-1,b,a,c)
num_disks = 3
tower_of_hanoi(num_disks, 'A', 'B', 'C')
```

S.No	Date	Topic	Sign
08	15/09/25	Constraint satisfaction problems.	

Q.8) Constraint Satisfaction Problem

→ Code:-

```
import itertools
variables = ["A", "B", "C"]
colors = ["Red", "Blue", "Yellow"]
all_assignments = itertools.product(colors, repeat = len(variables))
def valid(assignment):
    A, B, C = assignment
    return (A != B) and (B != C) and (A != C)
solutions = []
for assignment in all_assignments:
    if valid(assignment):
        solutions.append(dict(zip(variables, assignment)))
print("Valid colorings of the map :")
for sol in solutions:
    print(sol)
```

O/P:-

Valid colorings of the map:

{'A': 'Red', 'B': 'Blue', 'C': 'Yellow'}
{'A': 'Red', 'B': 'Yellow', 'C': 'Blue'}
{'A': 'Blue', 'B': 'Red', 'C': 'Yellow'}
{'A': 'Blue', 'B': 'Yellow', 'C': 'Red'}
{'A': 'Yellow', 'B': 'Red', 'C': 'Blue'}
{'A': 'Yellow', 'B': 'Blue', 'C': 'Red'}

O/P:-

Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C

Sr No	Date	Topic	Sign
09	05/09/25	Derive the expression based on a. Associative law b. Distributed law	

Q.1) →

```
Associative law
print ("Arithmetic Associative law")
a = int(input("Enter the value for a : "))
b = int(input("Enter the value for b : "))
c = int(input("Enter the value for c : "))
lhs_add = (a+b)+c
rhs_add = a+(b+c)
print (f"(a+b)+c = ({a}+{b})+{c} = {lhs_add}")
print (f"a+(b+c) = {a}+({b}+{c}) = {rhs_add}")
if lhs_add == rhs_add:
    print("Addition Associative law holds")
else:
    print("Addition Associative law does not holds")
lhs_mul = (a*b)*c
rhs_mul = a*(b*c)
print(f"(a*b)*c = ({a}*{b})*{c} = {lhs_mul}")
print(f"a*(b*c) = {a}*({b}*{c}) = {rhs_mul}")
if lhs_mul == rhs_mul:
    print("Multiplication Associative law holds")
else:
    print("Multiplication Associative law does not holds")
print("Boolean Associative law")
print("Enter boolean values as 1(True) and 0(False): ")
A = bool(int(input("Enter value for A: ")))
B = bool(int(input("Enter value for B: ")))
C = bool(int(input("Enter value for C: ")))
lhsAND = (A and B) and C
rhsAND = A and (B and C)
print(f"(A and B) and C = ({A} and {B}) and {C} = {lhsAND}")
print(f"A and (B and C) = {A} and ({B} and {C}) = {rhsAND}")
```

SrNo	Date	Topic	Sign
03	25/08/25	Alpha Beta Search Hill Climbing	

Q.1 Alpha Beta Search

→ Code:-

```
tree = {
    'A' = ['B', 'C'],
    'B' = [3, 5],
    'C' = [6, 9]
}
def minimax_alpha_beta(node, depth, alpha, beta, max_player):
    if depth == 0:
        if node in tree:
            return tree[node][0] if max_player else tree[node][0]
        else:
            return node
        if max_player:
            value = float('-inf')
            for child in tree[node]:
                value = max(value, minimax_alpha_beta(child, depth-1,
alpha, beta, false))
                alpha = max(alpha, value)
                if alpha >= beta:
                print(f"Pruning branch at node {node}")
                break
                alpha = max(alpha, value)
                if alpha >= beta:
                    print(f"Pruning branch at node {node}")
            return value
        else:
            value = float('inf')
            for child in tree[node]:
                value = min(value, minimax_alpha_beta(child, depth-1,
```

```
if lhs_AND == rhs_AND :
    print("AND Associative law holds")
else:
    print("AND Associative law does not holds")
lhs_OR = (A or B) or C
rhs_OR = A or(B or C)
print(f"(A or B) or C = ({A} or {B}) or {C} = {lhs_OR}")
print(f"{A or (B or C) = {A} and ({B} or {C}) = {rhs_OR}")
if lhs_OR == rhs_OR:
    print("OR Associative law holds")
else:
    print("OR Associative law doesnot holds")
```

O/P:-

```
Arithmetic Associative law
Enter the value for a : 10
Enter the value for b : 20
Enter the value for C : 30
(a+b) + c = (10+20) + 30 = 60
a+(b+c) = 10+(20+30) = 60
Addition Associatie law holds
(a * b) * c = (10 * 20) * 30 = 6000
a * (b * c) = 10 * (20 * 30) = 6000
Multiplication Associatie law holds
Boolean Associatie law
Enter boolean values as 1 (True) and 0 (False):
Enter value for A: 1
Enter value for B: 1
Enter value for C: 0
(A and B) and C = (True and True) and False = False
```

Sr No	Date
04	21/07/

```
alpha, beta, True))
        beta = min(beta, value)
        if beta <= alpha:
            print(f"Pruning branch at the node {node}")
            break
    return value
best_score = minimax_alpha_beta('A', 1, float('-inf'), float('inf'), True)
print(f"The best score is : {best_score}")
```

Output:-
The best score is :

e.2) Hill Climbing Algorithm

```
import random
def play():
    user = input("Enter your choice from rock, paper, scissors: ").lower()
    choices = ["rock", "paper", "scissors"]
    computer = random.choice : {computer}")
    if user == computer :
        return "It's a tie!"
    elif (user == "rock" and computer == "scissor") or (
        (user == "paper" and computer == "rock") or (
        (user == "scissors" and computer == "paper"):
            return "You Win!"
    elif user not in choices:
        return "Invalid choices!"
    else :
        return "Computer Wins!"
print(play())
```

o/p:-
Enter your choice from rock
paper, saissors:
Your choice : paper
Computer choice : rock
You Win!

e.1)

Code:-

```
graph = {
"Oradea"
"Zerind"
"Arad"
"Timisoa
"Lujog"
"Meha
"Sibiu"
80}, 184)
"Fag
"Ri
"Bu
"Pi
75}, 140
"Cr
104}, 22
"D
}
def gre
# n : (h
print (
for
```

A and (B and C) = True and (True and False) = False
AND Associatie Law holds
(A or B) or C = (True or True) or False = True
A or (B or C) = True and (True or False) = True
OR Associatie Law holds.

(a.2) Distributed law

```
print("Arithmetic Distributed Law")
a = int(input("Enter the value for a: "))
b = int(input("Enter the value for b: "))
c = int(input("Enter the value for c: "))
lhs_ari = a * (b+c)
rhs_ari = (a * b) + (b * c)
print(f"a * (b+c) = {a} * ({b} + {c}) = {lhs_ari}")
print(f"(a * b) + (b * c) = ({a} * {b}) + ({b} * {c}) = {rhs_ari}")
if lhs_ari == rhs_ari:
    print("Addition Distributed law holds")
else:
    print("Addition Distributed Law does not holds")
print("Boolean Distributed Law")
print("Enter boolean values as 1 (True) and 0 (False): ")
A = bool(int(input("Enter a value for A: ")))
B = bool(int(input("Enter a value for B: ")))
C = bool(int(input("Enter a value for C: ")))
lhs_bool = A and (B or C)
rhs_bool = (A and B) or (A and C)
print(f"A and (B or C) = {A} and ({B} or {C}) = {lhs_bool}")
print(f"(A and B) or (A and C) = ({A} and {B}) or ({A} and {C}) = {rhs_bool}")
```

Sr.No	Date	Topic	Sign
04	21/07/25	A* Algorithm Greedy Best First Search	

1)

Code :-

```
graph = {
"Oradea" : ({"Zerind" : 71, "Sibiu" : 151}, 380),
 "Zerind" : ({"Oradea": 71, "Arad" : 75, 374),
 "Arad" : ({"Zerind " : 75, "Sibiu" : 140, "Timisoara" : 118}, 366),
 "Timisoara" : ({"Arad": 118, "Lugoj" : 111}, 329),
 "Lugoj" : ({"Timisoara" : 111, "Mehadia" : 70}, 244),
 "Mehadia": ({"Lugoj" : 70, "Drobeta": 75}, 241),
  "Sibiu": ({"Oradea" : 151, "Fagaras" : 99, "Rimnicu Vilcea" :
80}, 184),
   "Fagaras" : ({"Sibiu" : 80, "Pitesti" : 97, "Craiova" : 146}, 210),
   "Rimnicu Vilcea" : ({"Sibiu" : 99, "Bucharest" : 211}, 176),
   "Bucharest" : ({"Fagaras" : 211, "Pitesti" : 101, "Urziceni" : 85}, 120),
   "Pitesti" : ({"Rimnicu Vilcea" : 97, "Craiova" : 138, "Bucharest" :
75}, 140),
   "Craiova" : ({"Rimnicu Vilcea" : 146, "Pitesti" : 138, "Drobeta" :
104}, 220),
   "Drobeta" : ({"Mehadia" : 75, "Craiova" : 120}, 242)
}
def greedy_search_rec(graph, prev, dist, path, q):
# n : (h(n))
print ("Connected nodes of current node", prev, "with h(n) values")
    for n in graph[prev][0]:  # neighbors list prev = arad -> z,s,T
      if n not in path:
        q[n] = graph[n][1]   # n = z[1] = 374
        print(n, "->", q[n])
    while q:
      mn = min(q, key=q.get)
      print("Taking minimum h(n) vertex: ", mn)
      # print(mn)
```


o/p :-

```
Enter your choice from rock
    paper, scissors:
Your choice : paper
 Computer choice : rock
 You Win!
```

```
if lhs_bool == rhs_bool:
    print("Boolean Distributive Law (AND over OR) holds")
else:
    print("Boolean Distributive Law (AND over OR) does not holds")
lhs_bool2 = A or (B and C)
rhs_bool2 = (A or B) and (A or C)
if lhs_bool == rhs_bool2:
    print("Boolean Distributive Law (AND over OR) holds")
else:
    print("Boolean Distributive Law (AND over OR) does not holds")
```

O/P:-

Arithmetic Distributed law
Enter the value for a : 10
Enter the value for b : 15
Enter the value for c : 20
a*(b+c) = 10*(15+20) = 350
(a*b)+(b*c) = (10*15)+(15*20) = 450
Addition Distributed law does not holds
Boolean Distributed law
Enter a value for A : 1
Enter a value for B : 0
Enter a value for C : 1
A and (B or C) = True and (False or True) = True
(A and B) or (A and C) = (True and False) or (True and True) = True
Boolean Distributive law (And over OR) holds
A or (B and C) = True or (False and True) = True
(A or B) and (A or C) = (True or False) and (True or True) = True
Boolean Distributive law (AND over OR) holds.

Sr NO
10

1) →

male(
male(
male(
female
femal
femal
parent
parent
parent
paren
paren
parent
Father
math
Biblio

Quer
?-
no
1?-
true
yes
Bibl
tru
yes
1?
n

```
    if dst == mn:
        return path + [dst]
    #del q[mn]
    new_path = greedy_search_rec(graph, mn, dst, path + [mn], {})
    if new_path:
        return new_path
    return []
source = input("Enter source vertext: ")
dst = input("Enter destination vertext: ")
path = greedy_search_rec(graph, source, dst, [source], {})
if path:
    print(path)
else:
    print("Path not found!")
print("Greedy Search Recursive: ", " -> ".join(greedy_search
rec(graph, "Pitesti", "Bucharest", ["Pitesti"], {})) or
"Path not found")
```

O/P:-

Enter source vertext: Pitesti

Enter destination vertext: Bucharest

Connected nodes of current node Pitesti with h(n) values

Rimnicu Vilcea → 210

Craiova → 220

Bucharest → 120

Taking minimum h(n) vertext : Bucharest

Greedy search Recursive : Pitesti → Bucharest.

Q.2)

Code:-

graph = {

"A": {

"B": {

"C": {

"F": {

"D": {

"G": {

"E": {

"M": {

"H": {

"L": {

}

def get_

mn = {

for i

if

ret

def as

pri

values:

fo

Q.2)

Code:-

```
graph = {
  "A": ({"B": 10, "C": 5, "F": 75}, 185),
  "B": ({"A": 10, "D": 60, "G": 30}, 175),
  "C": ({"A": 5, "F": 50, "D": 40, "F": 70}, 115),
  "F": ({"A": 75, "C": 50, "M": 20}, 65),
  "D": ({"B": 60, "H": 55}, 145),
  "G": ({"B": 30, "H": 55}, 190),
  "E": ({"C": 70, "D": 65, "H": 11, "M": 80}, 135),
  "M": ({"E": 80, "F": 20, "I": 45}, 45),
  "H": ({"G": 35, "E": 11, "I": 90}, 145),
  "I": ({"H": 90, "M": 45}, 0)
}
def get_min(q):
    mn = (0, (0, float("INF")))
    for i in q:
        if sum(q[i]) < sum(mn[1]):
            mn = (i, q[i])
    return mn[0]
def a_star(graph, prev, dst, path, pcost, q):
    print("Connected nodes of current node", prev, "with h(n) values:")
    for n in graph[prev][0]:
        if n not in path:
            q[n] = (graph[n][1], graph[prev][0][n])
            print(n, "->", q[n])
            add1 = sum(q[n])
            path_cost = pcost + add1
    print("h" value for "n" is :", path_cost)
```

Q2)

```
stud student(rahul, math, 85).
student(sita, math, 92).
student(amit, science, 78).
student(rina, science, 89).
student(karan, english, 70).
student(sita, english, 88).
high_scorer(Name):- student(Name, _, Marks), Marks > 80.
math_student(Name) :- student(Name, math, _).
topper(Subject, Name, Marks):
        student(Name, Subject, Marks).
        !, (student(_, Subject, Marks2), Marks2 > Marks).
```

Queries :-

```
?- high_scorer(Name).
Name = rahul ?;
Name = sita ?;
Name = rina ?;
Name = sita
yes
!?- math_student(Name)
Name = rahul ?;
Name = sita ?;
no

?- topper(Subject, Name, Marks).
Marks = 92
Name = sita
Subject = math ?;
Marks = 89
Name = rina.
```

```
while q:
    mn = get_min(q)
    print("Selecting the minimum vertext:", mn)
    print("_____")
    if dst == mn:
        return path + [dst]
    pc = pcost + q[mn][i]
    print("Previous path cost", pc)
    new_path = a_star(graph, mn, dst, path + [mn]
pc, q)
    if new_path:
        return new_path
    return []
source = input("Enter source vertext: ")
dst = input("Enter destination vertext: ")
heuristic = int(input("Enter given heuristic value for source: "))
if path:
    print(path)
else:
    print("path not found")
print("->".join(a_star(graph, "A", "L", [], 0, {"A": (185, 0)}))
```

O/P:-

Enter Source Vertext : A
Enter Destination vertext : L
Enter given heuristic value for sources : 185
Connected nodes of current node A with h(n) values:
B → (175, 10)
A* value for B is : 185
C → (115, 5)

Subject = science?;
Marks = 88
Name = sita
Subject = english
yes.

Marks > 80.
-).

rks 2 > Marks).

3) Facts :-

1. If x is batsman, then x is cricketer
2. If x is a cricketer, then x is a sportsman
3. If x is a sportsman, then x is famous.

Queries :-

check if sachin is a cricketer
check if virat is a sportsman
list all famous persons.

→ code

```
batsman(Sachin)
batsman(virat)
batsman(Rohit)
batsman(Dhoni)
Cricketer(x) :- batsman(x).
sportsman(x) :- cricket(x).
famous(x) :- sportsman(x).
```

O/P :-

```
? - batsman(sachin)
true

? batsman(virat)
true
```

A* value for C is :120
F → (65, 75)
A* value for F is : 140
Selecting the minimum vertext : C
previous path cost : 5
connected nodes of current node C :
A → (85, 5)
A* value for A is :195
F → (65, 50)
A* value for D is : 235
D → value for D is : (190, 40)
A* value for D is : 220
F → (125, 70)
A* value for F is : 200
Selecting the minimum vertex : F

previous path cost : 55
connected nodes for current node : F
A → (185, 75)
A* value for A is : 315
M → (45, 20)
A* value for M is : 120
Selecting the minimum vertex : M

[C, F, M,]

? batsman (Dhoni)
true

? - batsman (Rohit).
true

? - Cricketer (x)
x = Sachin;
x = Virat;
x = Rohit;
x = Dhoni;

4) Teacher → Employee → Human → Living Being
Write a program with the following
• Facts: Define atleast four teachers (eg: Anita, Raj, Meera, Rahul)
• Rules:-
If x is a teacher, then x is an Employee
If x is an employee, then x is a human
If x is a human, then x is a living being
• Queries:-
Prove that anita is a human
Prove that Rahul is a living being
list all humans.

Code:-
teacher (anita)
teacher (Raj).
teacher (meera).
teacher (rahul).
employee (x) :- teacher (x)
human (x) :- employee (x)

living

O/P:-
?- human
true

?- living
x = anit
x = raj
x = m
x = ra

5) Student
Write a
Facts:-
Defin
Rules:-
If
If
If
Queries
Code:-
Stude
stude
stude
Stud
lear

Sr.No	Date	Topic	Sign
05	14/07/25	a. Water Jug Problem b. Travelling Salesman Problem	

Q.1) Water Jug Problem

Code:-

```
def water_jug_manual_path():
    # Each step in the format (Jug1, Jug2)
    steps = []
    # Step 1: (0,0) -> Initial State
    a, b = 0, 0
    steps.append((a, b))
    # Step 2: Fill the Jug -> (5,0)
    a = 5
    steps.append((a, b))
    # Step 3: Pour Jug1 -> Jug2 -> (1,4)
    transfer = min(a, 4 - b)  # 5 can give 4
    a -= transfer  # a = a-3 = 5-3 = 2
    b += transfer  # b = b+3 = 1+3 = 4
    steps.append((a, b))
    # step 4: Empty jug2 -> (1,0)
    b = 0
    steps.append((a, b))
    # steps: Pour Jug1 -> Jug2 -> (0,1)
    transfer = min(a, 4 - b)
    a -= transfer
    b += transfer
    steps.append((a, b))
    # step 6: fill jug -> (5,1)
    a = 5
    steps.append((a, b))
    # Step 7: pour Jug1 -> Jug2 -> (2,4)
    transfer = min(a, 4 - b)
    a -= transfer
```

living being (x) :- human (x).

O/P :-

?- human (anita).

true

?- living being (x).

x = anita;

x = raj;

x = meera;

x = rahul.

5) Student → learner → knowledge seeker → future Professional.

Write a prolog program with the following

Raj, Meera, Rahul)

Facts :-

Define atleast four students (eg. - Riya, Amit, Sam, Neha).

Rules :-

If x is a student, then x is a learner

If x is a learner, then x is a knowledge seeker

If x is a knowledge seeker, then x is a future Professional

Queries :-

Check if Riya is a learner

Prove that Amit is a future professional.

List all knowledge seeker.

Code :-

Student (riya).

student (amit).

student (sam).

Student (neha)

learner (x) :- student (x).

```
b += transfer
steps.append((a,b))
# Step 8 : Empty Jug2 → (2,0)
b = 0
steps.append((a,b))
# Print the path
print("Steps to reach (2,0):")
for idx, state in enumerate(steps):
    print(f"Step {idx} : Jug1={state[0]}L, Jug2={state
[1]}L")
#Run the function
water_jug_manual_path()
```

Output:-

Steps to reach (2,0):
Step 0 : Jug1 = 0L, Jug2=0L
Step 1 : Jug1 = 5L, Jug2 = 0L
Step 2 : Jug1 = 1L, Jug2 = 4L
Step 3 : Jug1 = 1L, Jug2 = 0L
Step 4 : Jug1 = 0L, Jug2 = 1L
Step 5 : Jug1 = 5L, Jug2 = 1L
Step 6 : Jug1 = 2L, Jug2 = 4L
Step 7 : Jug1 = 2L, Jug2 = 0L

Q.2) Travelling Salesman Problem

Code :-

```
import random
#Distance for 4 cities TSP
distance={
[0,2,9,10],
```

knowledge_seeker(x) :- learner(x)
Future_professional(x) :- knowledge_seeker(x)
O/P :-
?-learner(riya)
true
?-future_professionals(amit)
true
?-knowledge_seeker(x)
x = riya;
x = amit;
x = sam;
x = neha;
false

6) Dog → Animal → Pet → living Being
Write a prolog program with the following
- Define atleast four dogs(Tommy, Bruno, Lucky, Rocky)
- Rules :- If x is a dog, then x is an animal
If x is an animal, then x can be pet
If x is a pet, then x is a living being.
- Queries
Prove that Tommy is a pet
Prove that Bruno is a living being
List all living beings.

Code :-
dog(tommy).
dog(bruno).
dog(lucky).
dog(rocky).

```
[2,0,6,4],
[9,6,0,3],
[10,4,3,0]
}
#total distance of tour
def get_cost(tour):
    cost = 0
    for i in range(len(tour)):
        costt = distance[tour[i-1]][tour[i]]
        #distance
        #print("cost of tour", cost)
    return cost
    #new tour by swapping two cities
def get_neighbour(tour):
    a,b = random.sample(range(len(tour)),2)
    tour[a], tour[b] = tour[b], tour[a]
    return tour
define hill_climb():
    current = [0,1,2,3]
    random.shuffle(current)
    current_cost = get_cost(current)
    print("Starting tour:", current, "Cost:", current_cost)
    for i in range(10):
        neighbour = current[:]
        #print("current neighbour:", neighbour)
        neighbour = get_neighbour(neighbour)
        #print("connected neighbour:", neighbour)
        neighbour_cost = get_cost(neighbour)
        if neighbour_cost < current_cost:
            current = neighbour
```

ug2={state

```
animal(x) :- dog(x).
pet(x) :- animal(x).
living_being(x) :- pet(x).
O/P:-
?-pet(tommy)
true
?-living_being(bruno)
true
?-living_being(x)
x=tommy;
x=bruno;
x=lucky;
x=rocky;
false
```

7) Book → knowledge source → educational material → valuable resources.

- Facts :- Define atleast four books (eg; physics, maths, history, computers).
- Rules
  - If x is a book, then x is a knowledge source
  - If x is a knowledge source, then x is an education material
  - If x is an educational material, then x is a valuable resource
- Queries:-
  - Check if math is an educational material
  - Prove that Physics is a valuable resource
  - List all valuable resource.

Code:-

```
book(physics).
book(math).
book(history).
book(computer).
```

knowledge_resource(x) :- book(x).
educational_material(x) :- knowledge_source(x).
valuable_resources(x) :- educational_materials(x).
O/P :-
?- educational_material(math).
true
?- valuable_resources(physics).
true
? valuable_resource(x)
x = physics ;
x = math ;
x = history ;
x = computer ;
false

```
#print("current neighbour", current)
current_cost = neighbour_cost
#print("current cost", current_cost)
print("Better tour found:", current, "(cost:", current_cost)
return current, current_cost
best_tour, best_cost = hill_climb()
print("\n Best tour:", best_tour)
print("Best cost:", best_cost)
```

Output :-

Starting tour : [3,2,0,1] cost : 18

Best tour : [3, 2, 0, 1]

Best cost : 18

Sr NO	Date	Topic	Sign
10	11/09/25	a) Derive the predicate b) Given predicates i) Draw family tree ii) Definition	

1)

→
```
male(john).
male(mike).
male(david).
female(lisa).
female(susan).
female(anna).
parent(john, mike).
parent(john, lisa).
parent(susan, mike).
parent(susan, lisa).
parent(mike, david).
parent(anna, david).
father(F, C) :- male(F), parent(F, C).
mother(M, C) :- female(M), parent(M, C).
siblings(X, Y) :- parent(P, X), parent(P, Y), X \= Y.
```

Queries

```
?- father(mike, anna).
no
1?- mother(susan, mike).
true?
yes
siblings(mike, lisa).
true?
yes
1?- grandfather(david, mike).
no
```