SRM MADURAI

COLLEGE FOR ENGINEERING AND TECHNOLOGY

Approved by AICTE, New Delhi | Affiliated to Anna University, Chennai

DEPARTMENT OF

ELECTRONICS AND COMMUNICATION ENGINEERING

LABORATORY MANUAL

Sub.Code	:	AD3311
Sub.Name	:	ARTIFICIAL INTELLIGENCE LABORATORY
Branch	:	AI & DS
Year / Semester	:	II / III
Regulation	:	R-2021

Prepared By,
Ms. R.Sangeetha AP/ECE

Verified By,
Dr.P.Tharcis HoD/ECE

Approved By,

INSTITUTE VISION
To become a centre of excellence in preparing engineering with excellent technical, scientific research and entrepreneurial abilities to contribute to the society.

INSTITUTE MISSION	
1	Providing comprehensive learning environment
2	Imparting state-of-the-art technology to fulfil the needs of the students and Industry
3	Establishing Industry-Institute alliance for bilateral benefits
4	Promoting Research and Development activities
5	Offering student lead activities to inculcate ethics, social responsibilities, entrepreneurial, and leadership skills

DEPARTMENT VISION
To become a centre of excellence in technical education and scientific research in the field of Computer Science and Engineering for the wellbeing of the society.

DEPARTMENT MISSION	
1	Producing graduates with a strong theoretical and practical in computer technology to meet the Industry expectation.
2	Offering holistic learning ambience for faculty and students to investigate, apply and transfer knowledge.
3	Inculcating interpersonal traits among the students leading to employability and entrepreneurship.
4	Establishing effective linkage with the Industries for the mutual benefits
5	Strengthening Research activities to solve the problems related to industry and society.

SYLLABUS

COURSE CODE	COURSE NAME	L	T	P	C
AD3311	ARTIFICIAL INTELLIGENCE LABORATORY	0	0	3	1.5

COURSE OBJECTIVES :

- To design and implement search strategies
- To implement game playing techniques
- To implement CSP techniques
- To develop systems with logical reasoning
- To develop systems with probabilistic reasoning

EXPERIMENTS

1. Implement basic search strategies – 8-Puzzle, 8 - Queens problem, Cryptarithmetic.
2. Implement A* and memory bounded A* algorithms
3. Implement Minimax algorithm for game playing (Alpha-Beta pruning)
4. Solve constraint satisfaction problems
5. Implement propositional model checking algorithms
6. Implement forward chaining, backward chaining, and resolution strategies
7. Build naïve Bayes models
8. Implement Bayesian networks and perform inferences
9. Mini-Project

TOTAL: 45 Periods

CONTENT BEYOND SYLLABI: TensorFlow or PyTorch

COURSE OUTCOMES:
On completion of the course, students will be able to:

CO1: Design and implement search strategies
CO2: Implement game playing and CSP techniques
CO3: Develop logical reasoning systems
CO4: Develop probabilistic reasoning systems

EQUIPMENT / SOFTWARE AND HARDWARE REQUIREMENT

- INTEL based desktop PC with min. 8GB RAM and 500 GB HDD, 17” or higher TFT Monitor, Keyboard and mouse
- Windows 10 or higher operating system / Linux Ubuntu 20 or higher
- Python3.9 and above, Python, Numpy, Scipy, Matplotlib, Pandas, seaborn, Pycharm

List of Experiments

EX.NO:1A	IMPLEMENT BASIC SEARCH STRATEGIES – 8-PUZZLE

AIM:
To implement basic startegies using 8-puzzle

AlGORITHM:

1) Import necessary modules and functions.
2) Set the value of n to 3and define movement lists row and col.
3) Define the priority Queue class with methods for a priority queue.
4) Define the node class to represent nodes in the search tree.
5) Define calculate Cost to compute the number of misplaced tiles.
6) Define the new Node function to create new nodes in the search tree.
7) Define the solve function to solve the puzzle using a priority queue, iterating until a solution is found.

PROGRAM:
```
import copy
from heapq import heappush, heappop

n = 3
row = [1, 0, -1, 0]
col = [0, -1, 0, 1]

class PriorityQueue:
    def __init__(self):
        self.heap = []

    def push(self, k):
        heappush(self.heap, k)

    def pop(self):
        return heappop(self.heap)

    def empty(self):
        return not self.heap

class Node:
    def __init__(self, parent, mat, empty_tile_pos, cost, level):
        self.parent = parent
        self.mat = mat
```

```
        self.empty_tile_pos = empty_tile_pos
        self.cost = cost
        self.level = level

    def __lt__(self, nxt):
        return self.cost < nxt.cost

def calculateCost(mat, final) -> int:
    count = 0
    for i in range(n):
        for j in range(n):
            if mat[i][j] and (mat[i][j] != final[i][j]):
                count += 1
    return count

def newNode(mat, empty_tile_pos, new_empty_tile_pos, level, parent, final) -> Node:
    new_mat = copy.deepcopy(mat)
    x1, y1 = empty_tile_pos
    x2, y2 = new_empty_tile_pos
    new_mat[x1][y1], new_mat[x2][y2] = new_mat[x2][y2], new_mat[x1][y1]
    cost = calculateCost(new_mat, final)
    return Node(parent, new_mat, new_empty_tile_pos, cost, level)

def printMatrix(mat):
    for i in range(n):
        for j in range(n):
            print("%d " % (mat[i][j]), end=" ")
        print()

def isSafe(x, y):
    return 0 <= x < n and 0 <= y < n

def printPath(root):
    if root is None:
        return
    printPath(root.parent)
    printMatrix(root.mat)
    print()

def solve(initial, empty_tile_pos, final):
    cost = calculateCost(initial, final)
    root = Node(None, initial, empty_tile_pos, cost, 0)
    pq = PriorityQueue()
    pq.push(root)

    while not pq.empty():
        minimum = pq.pop()
        if minimum.cost == 0:
            printPath(minimum)
            return
```

```
        for i in range(4):
            new_tile_pos = [minimum.empty_tile_pos[0] + row[i], minimum.empty_tile_pos[1] + col[i]]
            if isSafe(*new_tile_pos):
                child = newNode(minimum.mat, minimum.empty_tile_pos, new_tile_pos, minimum.level + 1,
minimum, final)
                pq.push(child)

initial = [[1, 2, 3], [5, 6, 0], [7, 8, 4]]
final = [[1, 2, 3], [5, 8, 6], [0, 7, 4]]
empty_tile_pos = [1, 2]

solve(initial, empty_tile_pos, final)
```

```
In [1]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
1  2  3
5  6  0
7  8  4

1  2  3
5  0  6
7  8  4

1  2  3
5  8  6
7  0  4

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

Description	Marks Allotted	Marks Obtained
Performance	**25**	
Record	**15**	
Viva- voce	**10**	
Total	**50**	

RESULT:
Thus the program is executed and implemented successfully.

EX.NO:1B	8 QUEEN PROGRAM

AIM:

To implement the search strategy using 8-queesns problem

ALGORITHM:

1. Define a function `is_safe` to check if a queen can be placed at a given position on the board.
2. Define a function `solveNQueens` to recursively solve the problem by trying all possible placements of queens on the board.
3. If all queens are placed, return `True`.
4. Try placing a queen in each row of the current column and recursively check if it's safe to do so.
5. If placing the queen leads to a solution, return `True`.
6. Backtrack and try placing the queen in a different row if no solution is found.
7. Define a function `print_board` to print the solution board.
8. Define a function `solveQueens` to initialize the board and solve the 8-Queens problem.
9. Call the `solveQueens` function to find and print a solution.

.

PROGRAM:

```
def isSafe(board, row, col):
    # Check this row on the left side
    for i in range(col):
        if board[row][i] == 1:
            return False

    # Check upper diagonal on the left side
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check lower diagonal on the left side
    for i, j in zip(range(row, len(board), 1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True

def solveNQueens(board, col):
    if col >= len(board):
        return True

    for i in range(len(board)):
        if isSafe(board, i, col):
            board[i][col] = 1
            if solveNQueens(board, col + 1):
                return True
            board[i][col] = 0
```

```
        return False



def printBoard(board):
    for i in range(len(board)):
        for j in range(len(board)):
            print(board[i][j], end="")
        print()

def solveQueens():
    board = [[0]*N for _ in range(N)]
    if not solveNQueens(board, 0):
        print("Solution does not exist")
        return False

    printBoard(board)
    return True

N = 8
solveQueens()
```

OUTPUT:

```
In [2]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
10000000
00000010
00001000
00000001
01000000
00010000
00000100
00100000
```

Description	Marks Allotted	Marks Obtained
Performance	**25**	
Record	**15**	
Viva- voce	**10**	
Total	**50**	

RESULT:
Thus the program is executed and implemented successfully.

EX.NO:1C	CRYPT ARITHMETIC PROGRAM

AIM:
To implement the serach strategy using cryptarithmetic

ALGORITHM:
1. Define a function solve_cryptarithmetic which takes the equation as input.
2. Extract the unique characters (operands and result) from the equation.
3. Generate all possible permutations of digits from 0 to 9 for the unique characters.
4. Iterate through each permutation.
5. Replace the characters in the equation with their corresponding digits according to the current permutation.
6. Evaluate the equation with the current permutation of digits.
7. If the equation holds true, print the solution.
8. If no solution is found, print a message indicating that no solution exists.

PROGRAM:
```
from itertools import permutations

def solve_cryptarithmetic(puzzle):
    # Extracting terms and result from the puzzle
    term1, term2, result = puzzle

    # Extracting unique characters
    unique_chars = set(char for word in puzzle for char in word if char.isalnum())
    unique_chars = sorted(unique_chars)

    # Generate permutations of digits for the unique characters
    for digits in permutations(range(10), len(unique_chars)):
        char_to_digit = {char: digit for char, digit in zip(unique_chars, digits)}

        # Convert terms and result to numeric values
        numeric_term1 = int("".join(str(char_to_digit[char]) for char in term1))
        numeric_term2 = int("".join(str(char_to_digit[char]) for char in term2))
        numeric_result = int("".join(str(char_to_digit[char]) for char in result))

        # Check if leading zeros are present
        if any(word[0] == '0' and len(word) > 1 for word in [term1, term2, result]):
            continue

        # Evaluate the puzzle
        if numeric_term1 + numeric_term2 == numeric_result:
            return char_to_digit
    return None
```

```
# Example usage
if __name__ == "__main__":
    puzzle = ('CAR', 'BUS', 'TRUCK')
    solution = solve_cryptarithmetic(puzzle)

    if solution:
        print("Solution found:")
        for word in puzzle:
            print("".join(str(solution[char]) if char.isalnum() else char for char in word), end=" ")
    else:
        print("No solution found.")
```

OUTPUT:

Solution found:

521 + 836 = 01357

```
In [30]: runfile('C:/Users/SRM/.spyder-py3/temp.py',
wdir='C:/Users/SRM/.spyder-py3')
Solution found:
521 836 01357
```

AUGMENTED QUESTIONS :

1. Implement a Python program to solve a cryptarithmetic puzzle such as `"CAT + DOG == ANIMAL"`. The program should assign unique digits to letters and verify the equation's validity.

2. Write a Python program to solve the 8-Puzzle problem with a custom initial configuration (e.g., `[[1, 3, 4], [8, 6, 2], [7, 0, 5]]`). Use Breadth-First Search (BFS) to find the sequence of moves to reach the goal state.

3. Create a Python program to solve a Sudoku puzzle using constraint satisfaction techniques. The program should input an unsolved Sudoku grid and output the solved grid, ensuring all rows, columns, and 3x3 subgrids adhere to Sudoku rules.

VIVA QUESTIONS:

1. How does Breadth-First Search (BFS) ensure completeness in finding a solution to the 8-Puzzle problem?
2. Explain a key difference between Depth-First Search (DFS) and Breadth-First Search (BFS) when applied to solving
the 8-Queens problem.
3. What role do heuristic functions play in the A* algorithm when solving the 8-Puzzle problem?
4. How does constraint propagation contribute to solving cryptarithmetic puzzles like `"SEND + MORE == MONEY"`?
5. Compare the approach of backtracking in solving the 8-Queens problem and Cryptarithmetic puzzles.

Description	Marks Allotted	Marks Obtained
Performance	25	
Record	15	
Viva- voce	10	
Total	50	

RESULT:
Thus the program is executed and implemented successfully.

EX.NO:2A	A*ALGORITHM PROGRAM

AIM:

To implement A* algorithm for searching using python program

ALGORITHM:

1. Import heapq for priority queue implementation.
2. Define the heuristic function to calculate the heuristic distance between a node and the goal.
3. Implement the A* algorithm, utilizing heapq for the priority queue, tracking the path, and g-score for each node.
4. Iterate until the open set is empty:
5. Pop the node with the lowest f-score from the open set.
6. If the current node is the goal, reconstruct and return the path.
7. Otherwise, update the g-score and add neighbors to the open set if necessary.
8. Define the neighbors function to obtain the neighbors of a given node.
9. Utilize the A* algorithm by passing the start, goal, and neighbors, then print the result.

PROGRAM:

```
import heapq

def heuristic(node, goal):
    return abs(node[0] - goal[0]) + abs(node[1] - goal[1])

def astar(start, goal, neighbors):
    open_set = [(0, start)]
    came_from = {}
    g_score = {start: 0}

    while open_set:
        _, current = heapq.heappop(open_set)

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]

        for neighbor in neighbors(current):
            tentative_g_score = g_score[current] + 1
```

```
            if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score = tentative_g_score + heuristic(neighbor, goal)
                heapq.heappush(open_set, (f_score, neighbor))

    return None

# Example usage
def neighbors(node):
    x, y = node
    return [(x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)]

start = (0, 0)
goal = (5, 5)
path = astar(start, goal, neighbors)

if path is not None:
    print(f"Path from {start} to {goal}:\n{path}")
else:
    print(f"No path found from {start} to {goal}")
```

OUTPUT:

Pathfrom (0,0)to(5,5):[(0,0),(0,1),(0,2),(0,3),(0,4),(0,5),(1,5),(2,5),(3,5),(4,5),(5,5)]

```
In [1]: runfile('C:/Users/SRM/.spyder-py3/temp.py', wdir='C:/
Users/SRM/.spyder-py3')
Path from (0, 0) to (5, 5):
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (1, 5), (2,
5), (3, 5), (4, 5), (5, 5)]

In [2]:
```

Description	Marks Allotted	Marks Obtained
Performance	25	
Record	15	
Viva- voce	10	
Total	**50**	

RESULT:
Thus the program is executed and implemented successfully.

EX.NO:2B	MEMORY BOUNDED A*

AIM:

To implement memory bound program using python.

ALGORITHM:

1. Initialize the priority queue open_set with the starting node (0, start) where start is the starting position.
2. Initialize dictionaries came_from to track predecessors and g_score to track the cost from the start to each node with {start: 0}.
3. Begin the main loop that continues while open_set is not empty:
4. Pop the node with the lowest f_score from open_set using heapq.heappop().
5. If the popped node current is the goal node goal, reconstruct the path from goal to start using came_from, appending nodes to path.
6. For each neighbor of current obtained from the neighbors() function, calculate tentative_g_score as the sum of g_score[current] and 1. Update came_from, g_score, and f_score if tentative_g_score is lower than the current g_score[neighbor]. Push (f_score, neighbor) onto open_set and update memory_used as the maximum of its current value and the sum of len(open_set) and len(came_from).

PROGRAM:

```
import heapq

def heuristic(node, goal):
    # Simple Manhattan distance heuristic
    return abs(node[0] - goal[0]) + abs(node[1] - goal[1])

def ma_star(start, goal, neighbors, memory_limit):
    open_set = [(0, start)]  # Priority queue (f_score, node)
    came_from = {}
    g_score = {start: 0}
    memory_used = 0

    while open_set:
        _, current = heapq.heappop(open_set)

        if current == goal:
            # Reconstruct the path from goal to start
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)  # Add the start node to the path
            return path[::-1], memory_used

        for neighbor in neighbors(current):
            tentative_g_score = g_score[current] + 1
```

```
            if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
            came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score = tentative_g_score + heuristic(neighbor, goal)
                heapq.heappush(open_set, (f_score, neighbor))
                memory_used = max(memory_used, len(open_set) + len(came_from))

        if memory_used > memory_limit:
            print(f"Exceeded memory limit ({memory_limit} units)")
            return None, memory_used

        # Debugging prints
        print(f"Current node: {current}, Open set size: {len(open_set)}, Memory used: {memory_used}")

    return None, memory_used

# Example usage
def neighbors(node):
    x, y = node
    return [(x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)]

start = (0, 0)
goal = (5, 5)
memory_limit = 20  # Adjust memory limit as needed

path, memory_used = ma_star(start, goal, neighbors, memory_limit)
if path is not None:
    print(f"Path from {start} to {goal}:")
    for step in path:
        print(step)
    print(f"Memory used: {memory_used} units of memory")
else:
    print(f"No path found from {start} to {goal} within memory limit of {memory_limit} units of memory")
```

OUTPUT:

No path found from(0,0)to (5,5)within memory limit of 10

```
In [15]: runfile('C:/Users/SRM/.spyder-py3/temp.py',
wdir='C:/Users/SRM/.spyder-py3')
Current node: (0, 0), Open set size: 4, Memory used: 8
Current node: (0, 1), Open set size: 6, Memory used: 13
Current node: (0, 2), Open set size: 8, Memory used: 18
Exceeded memory limit (20 units)
No path found from (0, 0) to (5, 5) within memory limit of 20
units of memory
```

AUGMENTED QUESTIONS :

1. Write a Python program to implement the A* algorithm for finding the shortest path on a grid from a start point to a goal point. The grid will be represented as a 2D list where each element is either 0 (passable) or 1 (impassable).
2. Write a Python program that visualizes and analyzes the performance of the A* algorithm on different grid configurations. The program should:Generate random grid configurations (e.g., varying densities of obstacles) and run the A* algorithm on each configuration.

VIVA QUESTIONS :

1. Explain the A* algorithm and its key components.
2. What are the necessary components for implementing the A* algorithm?
3. How does the A* algorithm ensure optimality and completeness in pathfinding?
4. Describe the concept of memory-bounded A* (MA*). How does it differ from the standard A* algorithm?
5. What are the trade-offs involved in using MA* compared to A*?

Description	**Marks Allotted**	**Marks Obtained**
Performance	**25**	
Record	**15**	
Viva- voce	**10**	
Total	**50**	

RESULT:
Thus the program is executed and implemented successfully.

EX.NO:3	MINIMAX ALGORITHM FOR GAME PLAYING

AIM:

To implement min max algorithm for game playing using alpha beta pruning

ALGORITHM:

1. Define the constants MAX and MIN to represent the maximum and minimum values.
2. Implement the minimax function, which takes parameters depth, nodeIndex, maximizingPlayer, values, alpha, and beta.
3. Check if the depth equals 3. If so, return the value of the current node.
4. If maximizingPlayer is True, initialize best to MIN and iterate over the possible child nodes:
5. Recursively call minimax with maximizingPlayer set to False.
6. Update best and alpha and perform alpha-beta pruning.
7. If maximizingPlayer is False, initialize best to MAX and iterate over the possible child nodes:
8. Recursively call minimax with maximizingPlayer set to True.
9. Update best and beta and perform alpha-beta pruning.
10. In the main block, initialize the list of values.
11. Call the minimax function with the initial parameters and print the optimal value returned.

PROGRAM:

```
MAX, MIN = 1000, -1000

def minimax(depth, nodeIndex, maximizingPlayer, values, alpha, beta):
    if depth == 3:
        return values[nodeIndex]
    if maximizingPlayer:
        best = MIN
        for i in range(2):
            val = minimax(depth + 1, nodeIndex * 2 + i, False, values, alpha, beta)
            best = max(best, val)
            alpha = max(alpha, best)
            if beta <= alpha:
                break
        return best

    else:
        best = MAX
        for i in range(2):
            val = minimax(depth + 1, nodeIndex * 2 + i, True, values, alpha, beta)
            best = min(best, val)
            beta = min(beta, best)
            if beta <= alpha:
                break
        return best

if __name___== "_main_":
    values = [3, 5, 6, 9, 1, 2, 0, -1]
```

```
print("The optimal value is:", minimax(0, 0, True, values, MIN, MAX))
```

OUTPUT:

The optimal value is: 5

```
In [16]: runfile('C:/Users/SRM/.spyder-py3/temp.py',
wdir='C:/Users/SRM/.spyder-py3')
The optimal value is: 5
```

Augmented questions :

1. Write a python program to implement Alpha-Beta Pruning with Minimax for Optimal Search
2. Write a python program to create a Connect Four Game with Minimax:
3. Write a python program using Minimax for Decision Making in Economic Simulations:

Viva Questions:

1. Can you explain the basic principle of the Minimax algorithm?
2. How does the Minimax algorithm handle the evaluation of game states?
3. What is the role of the evaluation function in the Minimax algorithm?
4. How does the Minimax algorithm differ from the Alpha-Beta pruning technique?
5. Can you describe a scenario where the Minimax algorithm might not be the best choice?

Description	**Marks Allotted**	**Marks Obtained**
Performance	**25**	
Record	**15**	
Viva- voce	**10**	
Total	**50**	

RESULT:

Thus the program is executed and implemented successfully.

EX.NO:4	CONSTRAINT SATISFACTION PROBLEMS

AIM:

To implement the constraint satisfaction problems using python programming.

ALGORITHM:

1. Define the `is_valid` function to check if the current assignment satisfies all constraints.
2. Implement the `backtrack` function to recursively explore possible assignments for each variable until a valid solution is found.
3. The `Variable` class represents a variable with its name and domain of possible values.
4. Implement the `solve_csp` function, which initializes an empty assignment and starts the backtracking algorithm with the given variables

PROGRAM:

```
class Variable:
    def __init__(self, name, domain):
        self.name = name
        self.domain = domain

def is_valid(assignment):
    # Check if the assignment satisfies all constraints
    # Implement your constraint checking logic here
    return True  # Placeholder; adjust as per actual constraints

def backtrack(assignment, variables):
    if len(assignment) == len(variables):
        return assignment

    var = variables[len(assignment)]
    for val in var.domain:
        assignment[var.name] = val
        if is_valid(assignment):
            result = backtrack(assignment, variables)
            if result is not None:
                return result
        assignment.pop(var.name)

    return None
```

```
def solve_csp(variables):
    assignment = {}
    return backtrack(assignment, variables)

# Example usage:
variables = [
    Variable('A', [1, 2, 3]),
    Variable('B', [4, 5, 6]),
    Variable('C', [7, 8, 9])
]

solution = solve_csp(variables)
if solution:
    print("Solution found:", solution)
else:
    print("No solution exists.")
```

OUTPUT:

Solution found: {'A':1,'B':4, 'C':7}

```
In [17]: runfile('C:/Users/SRM/.spyder-py3/temp.py',
wdir='C:/Users/SRM/.spyder-py3')
Solution found: {'A': 1, 'B': 4, 'C': 7}
```

Augmented Questions:

1. Write a program to Solve a Sudoku Variant (e.g., Killer Sudoku) Using CSP Techniques:
2. Class Timetabling Problem Using CSP:
3. Cryptarithm Solver Using CSP Techniques:

Viva Questions:

1. What is a Constraint Satisfaction Problem (CSP) and what are its key components?
2. Can you describe the difference between a constraint graph and a constraint hypergraph in CSPs?
3. What are some common techniques used to solve CSPs?
4. How does the backtracking algorithm work in the context of CSPs?
5. What is arc consistency and how does it improve the efficiency of solving CSPs?

Description	**Marks Allotted**	**Marks Obtained**
Performance	**25**	
Record	**15**	
Viva- voce	**10**	
Total	**50**	

RESULT:
Thus the program is executed and implemented successfully.

EX.NO:5	PROPOSITIONAL MODEL CHECKING ALGORITHMS

AIM:

To implement propositional model checking algorithm using python program.

ALGORITHM:

1. Define the `evaluateFunction` to evaluate a formula based on the given model.
2. For an 'and' operation, recursively evaluate both sub-formulas and return their logical AND.
3. For an 'or' operation, recursively evaluate both sub-formulas and return their logical OR.
4. For a 'not' operation, recursively evaluate the sub-formula and return its logical NOT.
5. Look up variable values in the model and return the corresponding value.
6. Define the `model_check` function to check if the formula holds in at least one model by evaluating it against all models.
7. Print the result of the model check..

PROGRAM:

```
def evaluate(formula, model):
    if formula[0] == 'and':
        return evaluate(formula[1], model) and evaluate(formula[2], model)
    elif formula[0] == 'or':
        return evaluate(formula[1], model) or evaluate(formula[2], model)
    elif formula[0] == 'not':
        return not evaluate(formula[1], model)
    else:
        return model.get(formula, False)

def model_check(formula, models):
    for model in models:
        if evaluate(formula, model):
            return True
    return False

formula = ('not', ('and', 'p', ('or', 'q', 'r')))
models = [
    {'p': True, 'q': False, 'r': True},
    {'p': False, 'q': True, 'r': False},
    {'p': True, 'q': True, 'r': False}
]

result = model_check(formula, models)
print(f"The formula holds in at least one model: {result}")
```

OUTPUT:

The formula holds in atleast one model:True

```
In [18]: runfile('C:/Users/SRM/.spyder-py3/temp.py',
wdir='C:/Users/SRM/.spyder-py3')
The formula holds in at least one model: True
```

Augmented Questions :

1. Probabilistic Model Checking Using Markov Chains
2. Temporal Logic Model Checking for Real-Time Systems
3. Hybrid Systems Model Checking

Viva Questions :

1. What is propositional model checking and why is it important?
2. Can you explain the difference between explicit state model checking and symbolic model checking?
3. How does the DPLL algorithm work in the context of propositional model checking?
4. What is the role of Binary Decision Diagrams (BDDs) in symbolic model checking?
5. Can you describe the concept of bounded model checking (BMC) and how it differs from traditional model checking?

Description	**Marks Allotted**	**Marks Obtained**
Performance	**25**	
Record	**15**	
Viva- voce	**10**	
Total	**50**	

RESULT:

Thus the program is executed and implemented successfully.

EX.NO:6A	FORWARD CHAINING PROGRAM

AIM:

ALGORITHM:

1. Start with known facts.
2. Iterate through rules in the knowledge base.
3. If all conditions of a rule are met, add the conclusion to new facts.
4. If the goal is in new facts, return True.
5. Add new facts to known facts and repeat if there are new facts.
6. If no new facts can be derived, return False.

PROGRAM:

```
def forward_chaining(kb, facts, goal):
    while True:
        new_facts = set()
        for rule in kb:
            if all(f in facts for f in rule['if']):
                new_facts.add(rule['then'])

        if not new_facts:
            break

        if goal in new_facts:
            return True

        facts.update(new_facts)

    return False

# Example usage
knowledge_base = [
    {'if': ['A', 'B'], 'then': 'C'},
    {'if': ['C', 'D'], 'then': 'E'}
]

initial_facts = {'A', 'B', 'D'}
goal_fact = 'E'

result = forward_chaining(knowledge_base, initial_facts, goal_fact)
print(f"Forward Chaining result: {result}")
```

OUTPUT:

Forward Chaining result:True

```
In [19]: runfile('C:/Users/SRM/.spyder-py3/temp.py',
wdir='C:/Users/SRM/.spyder-py3')
Forward Chaining result: True
```

Description	**Marks Allotted**	**Marks Obtained**
Performance	**25**	
Record	**15**	
Viva- voce	**10**	
Total	**50**	

RESULT:
Thus the program is executed and implemented successfully.

EX.NO:6B	BACKWARD CHAINING PROGRAM

AIM:

To implement backward chaining program using python programming.

ALGORITHM:

1. Check if the goal is already a known fact.
2. Iterate through the rules in the knowledge base.
3. If the rule's conclusion matches the goal, recursively check if all conditions (subgoals) of the rule are satisfied.
4. If all subgoals are satisfied, return True.
5. If no rule can satisfy the goal, return False.
6. Return the result of the recursive checks.

PROGRAM:

```
def backward_chaining(kb, facts, goal):
    if goal in facts:
        return True

    for rule in kb:
        if rule['then'] == goal:
            if all(backward_chaining(kb, facts, cond) for cond in rule['if']):
                return True

    return False

# Example usage
knowledge_base = [
    {'if': ['A', 'B'], 'then': 'C'},
    {'if': ['C', 'D'], 'then': 'E'}
]

initial_facts = {'A', 'B', 'D'}
goal_fact = 'E'

result = backward_chaining(knowledge_base, initial_facts, goal_fact)
print(f"Backward Chaining result: {result}")
```

OUTPUT:

Backward Chaining result:True

```
In [20]: runfile('C:/Users/SRM/.spyder-py3/temp.py',
wdir='C:/Users/SRM/.spyder-py3')
Backward Chaining result: True
```

Description	Marks Allotted	Marks Obtained
Performance	**25**	
Record	**15**	
Viva- voce	**10**	
Total	**50**	

RESULT:

Thus the program is executed and implemented successfully.

EX.NO: 6C	RESOLUTION STRATEGIES PROGRAM

AIM:

To implement resolution strategies using python programming**.**

ALGORITHM:

1. Initialize the list of clauses from the knowledge base and add the negation of the query.
2. Loop until no new clauses can be added:
3. Iterate over all pairs of clauses.
4. Resolve each pair of clauses.
5. If an empty clause is derived, return True (indicating the query is proven).

PROGRAM:

```
def resolve(c1, c2):
    resolved = False
    for literal in c1:
        neg_literal = ('not', literal) if literal[0] != 'not' else literal[1]
        if neg_literal in c2:
            resolved = True
            new_clause = tuple(set(c1 + c2) - {literal, neg_literal})
            if len(new_clause) == 0:
                return ()
            return new_clause
    return resolved
def resolution(knowledge_base, query):
    clauses = [tuple(knowledge_base[key]) for key in knowledge_base]
    clauses.append(('not', query))

    while True:
        new_clauses = list(clauses)
        for i in range(len(clauses)):
            for j in range(i + 1, len(clauses)):
                c1, c2 = clauses[i], clauses[j]
                resolvent = resolve(c1, c2)
                if resolvent == ():
                    return True
                if resolvent and resolvent not in new_clauses:
                    new_clauses.append(resolvent)

        if new_clauses == clauses:
```

```
            return False
        else:
            clauses = new_clauses

# Example usage
knowledge_base = {
    'p': ['q', 'r'],
    'q': ['s'],
    'r': [],
    's': ['p']
}
query = 'r'

result = resolution(knowledge_base, query)
print(f"Resolution: Query '{query}' is {'True' if result else 'False'}")
```

OUTPUT:

Resolution: Query 'not r' is False

```
In [21]: runfile('C:/Users/SRM/.spyder-py3/temp.py',
wdir='C:/Users/SRM/.spyder-py3')
Resolution: Query 'r' is False
```

Augmented Questions :

1. Implementing a Forward Chaining Rule-Based System for Financial Fraud Detection:
2. Backward Chaining for Natural Language Understanding in Chatbots
3. Resolution-Based Theorem Prover for Propositional Logic

Viva Questions :

1. Can you explain the forward chaining method in logic programming? How does it work?
2. What are the main differences between forward chaining and backward chaining?
3. How does backward chaining handle recursive rules and infinite loops?
4. What is the resolution strategy in propositional logic and how does it differ from forward and backward chaining?
5. Can you describe a real-world application where forward chaining, backward chaining, and resolution strategies might be used?

Description	**Marks Allotted**	**Marks Obtained**
Performance	**25**	
Record	**15**	
Viva- voce	**10**	
Total	**50**	

RESULT:
Thus the program is executed and implemented successfully.

EX.NO: 7	BUILD NAÏVE BAYES MODELS PROGRAM

AIM:

To implement build naïve bayes models using python programming.

ALGORITHM:

1. Initialize the classifier and dictionaries for class probabilities, mean, and variance.
2. For each class in the training data, calculate and store prior probabilities, mean, and variance.
3. For each test sample, compute posterior probabilities for each class.
4. Assign the class with the highest posterior probability to each test sample.
5. Use Gaussian distribution to calculate likelihoods based on class mean and variance.
6. Train the model using training data and predict the classes for the test data.

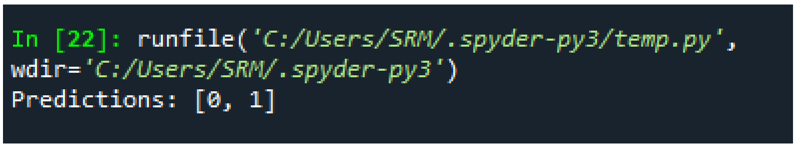
PROGRAM:

```
import numpy as np

class NaiveBayesClassifier:

    def __init__(self):
        self.class_prob = {}
        self.mean = {}
        self.variance = {}

    def fit(self, X, y):
        self.classes = np.unique(y)
        for c in self.classes:
            X_c = X[y == c]
            self.class_prob[c] = len(X_c) / len(X)
            self.mean[c] = X_c.mean()
            self.variance[c] = X_c.var()

    def predict(self, X):
        predictions = []
        for x in X:
            posteriors = []
            for c in self.classes:
                prob = self.class_prob[c]
```

```
            prob *= self.calculate_likelihood(self.mean[c], self.variance[c], x)
            posteriors.append(prob)
        predictions.append(self.classes[np.argmax(posteriors)])
    return predictions

@staticmethod
def calculate_likelihood(mean, var, x):
    exponent = np.exp(-((x - mean) ** 2) / (2 * var))
    return (1 / (np.sqrt(2 * np.pi * var))) * exponent

# Example usage
X_train = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)
y_train = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
X_test = np.array([3.5, 7]).reshape(-1, 1)

model = NaiveBayesClassifier()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
print("Predictions:", predictions)
```

OUTPUT:

Predictions: [0, 1]



Augmented Questions:

1. Sentiment Analysis on Social Media Posts Using Naïve Bayes
2. Spam Detection in Email Using Naïve Bayes
3. Naïve Bayes Classifier for Predicting Disease Outbreaks

Viva Questions :

1. What is the Naïve Bayes algorithm, and how does it work?
2. What are the types of Naïve Bayes models commonly used, and how do they differ?
3. How do you handle missing data or sparse features in Naïve Bayes modeling?
4. What are the advantages and limitations of Naïve Bayes models?
5. Can you describe a scenario where a Naïve Bayes model would be appropriate, and how would you evaluate its performance?

Description	Marks Allotted	Marks Obtained
Performance	**25**	
Record	**15**	
Viva- voce	**10**	
Total	**50**	

RESULT:
Thus the program is executed and implemented successfully.

EX.NO: 8	BAYESIAN NETWORK AND PERFORM INFERENCES PROGRAM

AIM:

To implement Bayesian network and perform inferences program using python.

ALGORITHM:

1. Initialize the Bayesian Network with an empty dictionary of nodes.
2. Add nodes to the network along with their parents and probabilities.
3. Calculate the probability of a node given evidence.
4. Query the probability of a specific node given evidence.
5. Print the probabilities for each node in the network.

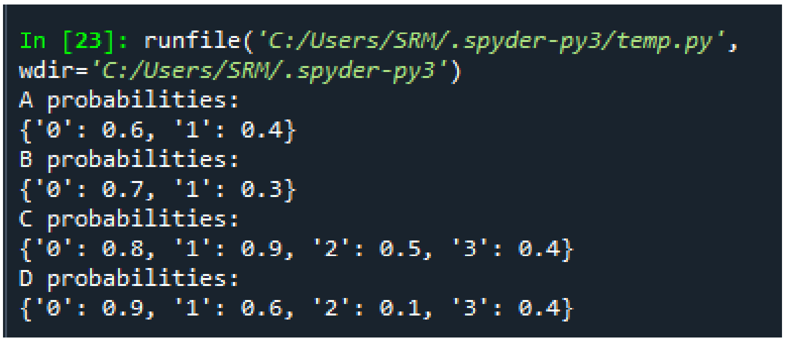
PROGRAM:

```
class BayesianNetwork:
    def _init_(self):
     self.nodes = {}

  def add_node(self, node_name, parents, probabilities):
      self.nodes[node_name] = {'parents': parents, 'probabilities': probabilities}

  def get_probability(self, node_name, evidence):
      parents = self.nodes[node_name]['parents']
      probabilities = self.nodes[node_name]['probabilities']

      if len(parents) == 0:
          return probabilities[0] if evidence[node_name] == 1 else probabilities[1]

      parent_values = [evidence[parent] for parent in parents]
      index = sum(val * 2 ** i for i, val in enumerate(reversed(parent_values)))
              return probabilities[index] if evidence[node_name] == 1 else probabilities[1 - index]

  def query(self, query_node, evidence):
      result_prob = 0.0
      for val in [0, 1]:
          evidence[query_node] = val
          prob = 1.0
          for node in self.nodes:
              if node != query_node:
```

```
                prob *= self.get_probability(node, evidence)
            result_prob += prob

        query_probabilities = [self.get_probability(query_node, evidence) for val in [0, 1]]
        return result_prob / sum(query_probabilities)

# Example usage
network = BayesianNetwork()
# Adding nodes and their conditional probabilities
network.add_node('A', [], [0.6, 0.4])
network.add_node('B', [], [0.7, 0.3])
network.add_node('C', ['A', 'B'], [0.8, 0.9, 0.5, 0.4])
network.add_node('D', ['C'], [0.9, 0.6, 0.1, 0.4])

# Displaying probabilities for each node
for node in network.nodes:
    node_probabilities = {str(i): prob for i, prob in enumerate(network.nodes[node]['probabilities'])}
    print(f"{node} probabilities:\n{node_probabilities}")
```

OUTPUT:

A probabilities: {'0': 0.6, '1': 0.4}
B probabilities: {'0': 0.7, '1': 0.3}
C probabilities: {'0': 0.8, '1': 0.9, '2': 0.5, '3': 0.4}
D probabilities: {'0': 0.9, '1': 0.6, '2': 0.1, '3': 0.4}



```
In [23]: runfile('C:/Users/SRM/.spyder-py3/temp.py',
wdir='C:/Users/SRM/.spyder-py3')
A probabilities:
{'0': 0.6, '1': 0.4}
B probabilities:
{'0': 0.7, '1': 0.3}
C probabilities:
{'0': 0.8, '1': 0.9, '2': 0.5, '3': 0.4}
D probabilities:
{'0': 0.9, '1': 0.6, '2': 0.1, '3': 0.4}
```

Augmented Questions:

1. Medical Diagnosis Using Bayesian Networks:
2. Predictive Maintenance Using Bayesian Networks:
3. Fraud Detection in Financial Transactions Using Bayesian Networks:

Viva Questions :

1. What is a Bayesian network, and how does it represent probabilistic relationships?
2. Explain the process of performing probabilistic inference in Bayesian networks.
3. How do you handle missing data or incomplete evidence when performing inference in Bayesian networks?
4. What are the advantages of using Bayesian networks over other probabilistic modeling techniques?
5. Can you describe a real-world application where Bayesian networks are used, and how would you evaluate the performance of the model?

Description	**Marks Allotted**	**Marks Obtained**
Performance	**25**	
Record	**15**	
Viva- voce	**10**	
Total	**50**	

RESULT:
Thus the program is executed and implemented successfully.

EX.NO: 9	Mini Project Sample Ideas

1. Sentiment Analysis on Social Media Data:
2. Handwritten Digit Recognition:
3. Chatbot Development:
4. Predictive Maintenance for Equipment:
5. Image Classification Using Transfer Learning:
6. Recommendation System for Movies or Products:
7. Autonomous Agent for Simple Games:
8. Speech Recognition System:
9. Fraud Detection in Financial Transactions:
10. Traffic Sign Recognition Using Computer Vision: