NANDHA ENGINEERING COLLEGE

(Autonomous Institution)

Erode-638 052

RECORD NOTE BOOK

22AIP03 – ARTIFICIAL INTELLIGENCE LABORATORY

III – Semester

B.Tech - Artificial Intelligence and Data Science

Department of Artificial Intelligence and Data Science

NANDHA ENGINEERING COLLEGE, ERODE-52

Name : ……

Reg.No : …… Year : …….

Branch : …… Semester : …….

NANDHA ENGINEERING COLLEGE

(Autonomous Institution)

Erode-638 052

BONAFIDE CERTIFICATE

REGISTER NUMBER:

Certified that this is the Bonafide Record of work done by…….. of the **Third Semester** B.E./B.Tech. **Artificial Intelligence and Data Science** branch during the Academic Year **2023-2024** in the **22AIP03 – ARTIFICIAL INTELLIGENCE LABORATORY.**

…….. **……**

Staff-in-charge **Head of the Department**

Submitted for the End Semester Practical Examination

Held on…….

…….. **……..**

Internal Examiner **External Examiner**

INSTITUTE VISION AND MISSION	
VISION	• To be an Institute of excellence providing quality Engineering, Technology and Management education to meet the ever changing needs of the society.
MISSION	• To provide quality education to produce ethical and competent professionals with social Responsibility • To excel in the thrust areas of Engineering, Technology and Entrepreneurship by solving real- world problems. • To create a learner centric environment and improve continually to meet the changing global needs.

DEPARTMENT VISION AND MISSION	
VISION	• To emerge as a renowned department in providing quality Artificial Intelligence and Data Science education to meet the ever growing needs of the society.
MISSION	Artificial Intelligence and Data Science department is committed • To provide quality and value based education to produce Artificial Intelligence professionals with ethical and social responsibility. • To excel in the thrust areas of Artificial Intelligence, Machine Learning and Data Science by imparting programming knowledge and Mathematical skill set to solve real world problems. • To create a learner centric environment that motivates the students in adopting emerging technologies of the rapidly changing artificial intelligence and data science society.
PROGRAMME EDUCATIONAL OBJECTIVES (PEO)	The graduates of Artificial intelligence and data science will be able: **PEO1:** Core Competency: To apply mathematical, scientific and engineering concepts for an artificial intelligence and data scientist to remit the various challenges using emerging AI technologies. **PEO2:** Research, Innovation and Entrepreneurship: To work productively in multidisciplinary teams and provide innovative ideas for real time problems through research. **PEO3:** Ethics, Human values and Life-long learning: To embrace lifelong learning with higher ethical standards and be the source for socio economic growth.

PROGRAMME SPECIFIC OUTCOMES (PSO)	The students of Artificial intelligence and data science will be able to **PSO1:** Analytical Skill: Ability to Design and develop innovative automated systems applying mathematical, analytical, programming and operational skills to meet society needs. **PSO2:** Knowledge Proficiency: Provide a tangible foundation and enhance the abilities to qualify for employment, higher studies and research in artificial intelligence and data science with ethical values.

PROGRAM OUTCOMES:

At the end of this programme the students will be able to

a-l	GRADUATE ATTRIBUTES	PO No.	PROGRAMME OUTCOMES
a	Engineering Knowledge	**PO1**	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
b	Problem Analysis	**PO2**	Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
c	Design and Development of Solutions	**PO3**	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
d	Investigation of Complex Problems	**PO4**	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
e	Modern Tool Usage	**PO5**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

f	The Engineer and Society	**PO6**	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
g	Environment and Sustainability	**PO7**	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
h	Ethics	**PO8**	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
i	Individual and Team Work.	**PO9**	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
j	Communication	**PO10**	Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
k	Project Management and Finance	**PO11**	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one‘s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
l	Lifelong Learning	**PO12**	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

COURSE OBJECTIVES:

1. To design and implement search strategies.
2. To apply appropriate algorithms for solving given AI problems.
3. To Design and implement CSP Techniques.
4. To Design and implement logical reasoning agents.
5. To develop systems with probabilistic reasoning.

COURSE OUTCOMES:

1. The student will be able to Design and implement search strategies.
2. The students will be able to Develop programs to solve the given AI problems.
3. The student will be able to Implement game playing and CSP techniques.
4. The student will be able to Develop logical reasoning systems.
5. The student will be able to Develop probabilistic reasoning systems.

SYLLABUS:

1. Implement basic search strategies – 8-Puzzle, 8 - Queens problem.
2. Implement A* and memory bounded A* algorithms
3. Implement Minimax algorithm for game playing (Alpha-Beta pruning)
4. Implement simulated annealing algorithms for AI tasks
5. Implement backtracking algorithms for CSP
6. Implement local search algorithms for CSP
7. Build naïve Bayes models
8. Implement Bayesian networks and perform inferences
9. Mini-Project

Mapping of COs with POs / PSOs														
COs	**POs**												**PSOs**	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**1**	**2**
1	2	1	3	3	-	-	-	-	1	1	2	1	2	3
2	1	2	3	3	2	-	-	-	3	2	3	3	3	2
3	3	1	3	3	1	-	-	-	1	3	1	2	2	3
4	2	1	1	1	1	-	-	-	2	3	1	2	2	3
5	3	1	1	1	1	-	-	-	1	3	3	3	2	3
CO (W.A)	**2**	**1**	**2**	**2**	**1**	**-**	**-**	**-**	**2**	**2**	**2**	**2**	**2**	**3**

INDEX

Ex. No.	DATE	NAME OF THE EXPERIMENTS	PAGE No.	MARKS	SIGN
1.		Implement basic search strategies – 8-Puzzle, 8 - Queens problem.			
2.		Implement A* and memory bounded A* algorithms			
3.		Implement Minimax algorithm for game playing (Alpha-Beta pruning)			
4.		Implement simulated annealing algorithms for AI tasks			
5.		Implement backtracking algorithms for CSP			
6.		Implement local search algorithms for CSP			
7.		Build naïve Bayes models			
8.		Implement Bayesian networks and perform inferences			
9.		Mini-Project			
AVERAGE MARKS AWARDED					
	Content Beyond the Syllabus				

SEARCH STRATEGIES

EX. NO: 1

DATE:

a.8-PUZZLE

AIM:

To write a Python Program to solve an 8-Puzzle Problem.

ALGORITHM:

1. Import the 'copy' for deepcopy method
2. Import the heap methods from the python library for the Priority Queue
3. Initialize the values for rows and columns.
4. Create a class for the priority queue and for structure of a node.
5. In the class for priority queue, define the functions for the following:
   - For initializing the Priority Queue
   - For inserting a new key
   - For removing the minimum element from the Priority Queue
   - To Check if the Queue is empty or not
6. In the class for structure of a node, define the functions for the following:
   - Initialize the parent node, current node, matrix, empty space tile, order the nodes based on costs
   - Create new nodes
   - To print N by N Matrix
   - To know if (x,y) is a valid or invalid
   - To print the path from the root node to the final node
   - Solving the Puzzle using Branch and Bound Method
7. In the Main Code , Initialize the Matrix with value 0 assigned as an empty space.
8. Invoke the function solve() to solve the puzzle
9. Stop the Program

SOURCE CODE:

```
import copy

from heapq import heappush, heappop

n = 3
```

```
# bottom, left, top, right
rows = [ 1, 0, -1, 0 ]
cols = [ 0, -1, 0, 1 ]

# creating a class for the Priority Queue
class priorityQueue:

    # Constructor for initializing a
    # Priority Queue
    def __init__(self):
        self.heap = []

    # Inserting a new key 'key'
    def push(self, key):
        heappush(self.heap, key)

    # funct to remove the element that is minimum,
    # from the Priority Queue
    def pop(self):
        return heappop(self.heap)

    # funct to check if the Queue is empty or not
    def empty(self):
        if not self.heap:
            return True
        else:
            return False

# structure of the node
class nodes:

    def __init__(self, parent, mats, empty_tile_posi,
                 costs, levels):

        # This will store the parent node to the
        # current node And helps in tracing the
        # path when the solution is visible
        self.parent = parent

        # Useful for Storing the matrix
        self.mats = mats

        # useful for Storing the position where the
```

```
        # empty space tile is already existing in the matrix
        self.empty_tile_posi = empty_tile_posi

        # Store no. of misplaced tiles
        self.costs = costs

        # Store no. of moves so far
        self.levels = levels

    # This func is used in order to form the
    # priority queue based on
    # the costs var of objects
    def __lt__(self, nxt):
        return self.costs < nxt.costs

# method to calc. the no. of
# misplaced tiles, that is the no. of non-blank
# tiles not in their final posi
def calculateCosts(mats, final) -> int:

    count = 0
    for i in range(n):
        for j in range(n):
            if ((mats[i][j]) and
                (mats[i][j] != final[i][j])):
                count += 1

    return count

def newNodes(mats, empty_tile_posi, new_empty_tile_posi,
             levels, parent, final) -> nodes:

    # Copying data from the parent matrixes to the present matrixes
    new_mats = copy.deepcopy(mats)

    # Moving the tile by 1 position
    x1 = empty_tile_posi[0]
    y1 = empty_tile_posi[1]
    x2 = new_empty_tile_posi[0]
    y2 = new_empty_tile_posi[1]
    new_mats[x1][y1], new_mats[x2][y2] = new_mats[x2][y2], new_mats[x1][y1]

    # Setting the no. of misplaced tiles
    costs = calculateCosts(new_mats, final)
```

```
    new_nodes = nodes(parent, new_mats, new_empty_tile_posi,
                      costs, levels)
    return new_nodes

# func to print the N by N matrix
def printMatsrix(mats):

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

        print()

# func to know if (x, y) is a valid or invalid
# matrix coordinates
def isSafe(x, y):

    return x >= 0 and x < n and y >= 0 and y < n

# Printing the path from the root node to the final node
def printPath(root):

    if root == None:
        return

    printPath(root.parent)
    printMatsrix(root.mats)
    print()

# method for solving N*N - 1 puzzle algo
# by utilizing the Branch and Bound technique. empty_tile_posi is
# the blank tile position initially.
def solve(initial, empty_tile_posi, final):

    # Creating a priority queue for storing the live
    # nodes of the search tree
    pq = priorityQueue()

    # Creating the root node
    costs = calculateCosts(initial, final)
    root = nodes(None, initial,
                 empty_tile_posi, costs, 0)
```

```
    # Adding root to the list of live nodes
    pq.push(root)

    # Discovering a live node with min. costs,
    # and adding its children to the list of live
    # nodes and finally deleting it from
    # the list.
    while not pq.empty():

        # Finding a live node with min. estimatsed
        # costs and deleting it form the list of the
        # live nodes
        minimum = pq.pop()

        # If the min. is ans node
        if minimum.costs == 0:

            # Printing the path from the root to
            # destination;
            printPath(minimum)
            return

        # Generating all feasible children
        for i in range(n):
            new_tile_posi = [
                minimum.empty_tile_posi[0] + rows[i],
                minimum.empty_tile_posi[1] + cols[i], ]

            if isSafe(new_tile_posi[0], new_tile_posi[1]):

                # Creating a child node
                child = newNodes(minimum.mats,
                                 minimum.empty_tile_posi,
                                 new_tile_posi,
                                 minimum.levels + 1,
                                 minimum, final,)

                # Adding the child to the list of live nodes
                pq.push(child)

# Main Code

# Initial configuration
# Value 0 is taken here as an empty space
```

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

# Final configuration that can be solved
# Value 0 is taken as an empty space
final = [ [ 1, 2, 3 ],
        [ 5, 8, 6 ],
        [ 0, 7, 4 ] ]

# Blank tile coordinates in the
# initial configuration
empty_tile_posi = [ 1, 2 ]

# Method call for solving the puzzle
solve(initial, empty_tile_posi, final)
```

OUTPUT:

```
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
```

B.8-QUEENS

AIM:

To write a Python Program to solve an 8-Puzzle Problem.

ALGORITHM:

1. Get the number of queens as an input from the user
2. Create a Chessboard NxN matrix with all elements set to 0
3. Define the function attack() to check the arrangement of queens vertically, horizontally or diagonally.
4. Define the function N_queens() to place the n queens in their appropriate position in the chess board
5. Invoke the function N_queens(N) and print the result.
6. Stop the program

SOURCE CODE:

```
print ("Enter the number of queens")
N = int(input())
# here we create a chessboard
# NxN matrix with all elements set to 0
board = [[0]*N for _ in range(N)]
def attack(i, j):
    #checking vertically and horizontally
    for k in range(0,N):
        if board[i][k]==1 or board[k][j]==1:
            return True
    #checking diagonally
    for k in range(0,N):
        for l in range(0,N):
            if (k+l==i+j) or (k-l==i-j):
                if board[k][l]==1:
                    return True
    return False
def N_queens(n):
    if n==0:
        return True
    for i in range(0,N):
        for j in range(0,N):
            if (not(attack(i,j))) and (board[i][j]!=1):
                board[i][j] = 1
                if N_queens(n-1)==True:
                    return True
                board[i][j] = 0
    return False
N_queens(N)
```

```
for i in board:
    print (i)
```

OUTPUT

```
Enter the number of queens
8
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
```

VIVA QUESTIONS:

1. What are the applications of AI?
2. Define AI.
3. Which search method takes less memory?
4. Why do we need Artificial Intelligence?

MARK ALLOCATION		
Preparation & conduct of experiments	(50)	
Observation & result	(30)	
Record	(10)	
Viva –voce	(10)	
Total	(100)	

RESULT:

Thus search strategies of 8-Puzzle, 8 - Queens problem has been implemented using Python successfully.

A* AND MEMORY BOUNDED A* ALGORITHMS

EX NO: 2

DATE:

Aim:

To implement A* algorithm for shortest path problem using Python.

Algorithm:

Step 1: Define a Class Node and initialize the nodes

Step 2: In the class Node, define the methods for the following:

- initialization of nodes, assigning names to the nodes
- find its neighbours and create a link to the existing nodes.
- to create a graph.

Step 3: Define the method astar_search() to perform the following:

- Create an empty lists for open and closed
- Add all the nodes to the open list.
- Assign a start node and a goal node
- Add the start node to the closed list
- Loop until the open list is empty:

- >Get the neighbouring nodes, determine the lowest costnode by applying heuristics
- >Check if the neighbour to be added, if so append it the closed list
- >Check if we have reached the goal, return the path (From Current Node to Start Node By Node.parent)

Step 4: In the main Program, Assign the values for the nodes and heuristics.

Step 5: Invoke the method astar_search() and run the A* algorithm

Step 6: Display the shortest path from start node to the goal node.

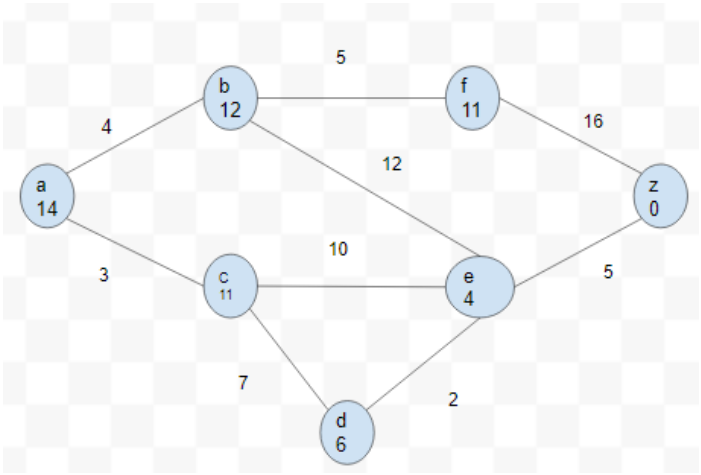
Step 7: Stop the program.

Program:

```
def get(graph, a, b=None):
   links = graph.setdefault(a, {})
   if b is None:
      return links
   else:
      return get(links,b)
class Node:
   def __init__(self, name:str, parent:str):
```

```
        self.name = name
        self.parent = parent
        self.g = 0
        self.h = 0
        self.f = 0
    def __eq__(self, other):
        return self.name == other.name
    def __lt__(self, other):
         return self.f < other.f
    def __repr__(self):
        return ('({0},{1})'.format(self.name, self.f))
def add_to_open(open, neighbor):
    for node in open:
        if (neighbor == node and neighbor.f > node.f):
            return False
    return True
def astar_search(graph, heuristics, start, end):
    open = []
    closed = []
    start_node = Node(start, None)
    goal_node = Node(end, None)
    open.append(start_node)
    while len(open) > 0:
        open.sort()
        current_node = open.pop(0)
        closed.append(current_node)
        if current_node == goal_node:
            path = []
            while current_node != start_node:
                path.append(current_node.name + ': ' + str(current_node.g))
                current_node = current_node.parent
            path.append(start_node.name + ': ' + str(start_node.g))
            return path[::-1]
        neighbors = graph.get(current_node.name)
        for key, value in neighbors.items():
            neighbor = Node(key, current_node)
            if(neighbor in closed):
                continue
            neighbor.g = current_node.g + get(graph,current_node.name, neighbor.name)
            neighbor.h = heuristics.get(neighbor.name)
            neighbor.f = neighbor.g + neighbor.h
            if(add_to_open(open, neighbor) == True):
                open.append(neighbor)
    return None
g={'a':{'b':4,'c':3},'b':{'f':5,'e':12},'c':{'e':10,'d':7},'d':{'e':2},'e':{'z':5},'f':{'z':16},'z':{}}
h={'a':14,'b':12,'c':11,'d':6,'e':4,'f':11,'z':0}
path = astar_search(g, h, 'a', 'z')
print(path)
```

Output:



['a: 0', 'c: 3', 'd: 10', 'e: 12', 'z: 17']

VIVA QUESTIONS:

1. A* algorithm is based on which search method?
2. What is memory bounded A* algorithm?
3. What are the applications of A* algorithm?

MARK ALLOCATION		
Preparation & conduct of experiments	(50)	
Observation & result	(30)	
Record	(10)	
Viva –voce	(10)	
Total	(100)	

RESULT:

Thus A* algorithm has been implemented using Python successfully.

ALPHA-BETA PRUNING

EX NO: 3

DATE:

Aim :

To write a Python program to solve tic tac toe gaming using MINIMAX with ALPHA-BETA Pruning

Algorithm:

1. First, generate the entire game tree starting with the current position of the game all the way up to the terminal states.
2. Apply the utility function to get the utility values for all the terminal states.
3. Determine the utilities of the higher nodes with the help of the utilities of the terminal nodes. For instance, in the diagram below, we have the utilities for the terminal states written in the squares.
4. Calculate the utility values with the help of leaves considering one layer at a time until the root of the tree.
5. Eventually, all the backed-up values reach to the root of the tree, i.e., the topmost point. At that point, MAX has to choose the highest value.

Alpha Beta Pruning:

The two-parameter can be defined as:

6. Alpha: The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞.
7. Beta: The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞. The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Program:

```
#tic tac toe with alpha beta pruning
from random import choice
```

```
from math import inf
board = [[0, 0, 0],  [0, 0, 0],  [0, 0, 0]]
def Gameboard(board):
    chars = {1: 'X', -1: 'O', 0: ' '}
    for x in board:
        for y in x:
            ch = chars[y]
            print(f'| {ch} |', end='')
        print('\n' + '---------------')
    print('===============')

def Clearboard(board):
    for x, row in enumerate(board):
        for y, col in enumerate(row):
            board[x][y] = 0
def winningPlayer(board, player):
    conditions = [[board[0][0], board[0][1], board[0][2]],
                  [board[1][0], board[1][1], board[1][2]],
                  [board[2][0], board[2][1], board[2][2]],
                  [board[0][0], board[1][0], board[2][0]],
                  [board[0][1], board[1][1], board[2][1]],
                  [board[0][2], board[1][2], board[2][2]],
                  [board[0][0], board[1][1], board[2][2]],
                  [board[0][2], board[1][1], board[2][0]]]

    if [player, player, player] in conditions:
        return True
    return False

def gameWon(board):
    return winningPlayer(board, 1) or winningPlayer(board, -1)

def printResult(board):
    if winningPlayer(board, 1):
        print('X has won! ' + '\n')
    elif winningPlayer(board, -1):
        print('O\'s have won! ' + '\n')
    else:
        print('Draw' + '\n')

def blanks(board):
    blank = []
    for x, row in enumerate(board):
        for y, col in enumerate(row):
```

```
            if board[x][y] == 0:
                blank.append([x, y])
    return blank

def boardFull(board):
    if len(blanks(board)) == 0:
        return True
    return False
def setMove(board, x, y, player):
    board[x][y] = player

def playerMove(board):
    e = True
    moves = {1: [0, 0], 2: [0, 1], 3: [0, 2],
             4: [1, 0], 5: [1, 1], 6: [1, 2],
             7: [2, 0], 8: [2, 1], 9: [2, 2]}
    while e:
        try:
            move = int(input('Enter a number between 1-9: '))
            if move < 1 or move > 9:
                print('Invalid Move! Try again!')
            elif not (moves[move] in blanks(board)):
                print('Invalid Move! Try again!')
            else:
                setMove(board, moves[move][0], moves[move][1], 1)
                Gameboard(board)
                e = False
        except(KeyError, ValueError):
            print('Enter a number!')

def getScore(board):
    if winningPlayer(board, 1):
        return 10
    elif winningPlayer(board, -1):
        return -10

    else:
        return 0

def abminimax(board, depth, alpha, beta, player):
    row = -1
    col = -1
    if depth == 0 or gameWon(board):
        return [row, col, getScore(board)]
```

```
    else:
        for cell in blanks(board):
            setMove(board, cell[0], cell[1], player)
            score = abminimax(board, depth - 1, alpha, beta, -player)
            if player == 1:
                # X is always the max player
                if score[2] > alpha:
                    alpha = score[2]
                    row = cell[0]
                    col = cell[1]
            else:
                if score[2] < beta:
                    beta = score[2]
                    row = cell[0]
                    col = cell[1]
            setMove(board, cell[0], cell[1], 0)
            if alpha >= beta:
                break

        if player == 1:
            return [row, col, alpha]
        else:
            return [row, col, beta]

def o_comp(board):
    if len(blanks(board)) == 9:
        x = choice([0, 1, 2])
        y = choice([0, 1, 2])
        setMove(board, x, y, -1)
        Gameboard(board)
    else:
        result = abminimax(board, len(blanks(board)), -inf, inf, -1)
        setMove(board, result[0], result[1], -1)
        Gameboard(board)
def x_comp(board):
    if len(blanks(board)) == 9:
        x = choice([0, 1, 2])
        y = choice([0, 1, 2])
        setMove(board, x, y, 1)
        Gameboard(board)
    else:
        result = abminimax(board, len(blanks(board)), -inf, inf, 1)
        setMove(board, result[0], result[1], 1)
        Gameboard(board)
```

```
def makeMove(board, player, mode):
    if mode == 1:
        if player == 1:
            playerMove(board)
        else:
            o_comp(board)
    else:
        if player == 1:
            o_comp(board)
        else:
            x_comp(board)

def pvc():
    while True:
        try:
            order = int(input('Enter to play 1st or 2nd: '))
            if not (order == 1 or order == 2):
                print('Please pick 1 or 2')
            else:
                break
        except(KeyError, ValueError):
            print('Enter a number')

    Clearboard(board)
    if order == 2:
        currentPlayer = -1
    else:
        currentPlayer = 1
    while not (boardFull(board) or gameWon(board)):
        makeMove(board, currentPlayer, 1)
        currentPlayer *= -1
    printResult(board)

# Driver Code
print("================================================")
print("TIC-TAC-TOE using MINIMAX with ALPHA-BETA Pruning")
print("================================================")
pvc()
```

Output:

```
================================================
TIC-TAC-TOE using MINIMAX with ALPHA-BETA Pruning
================================================
```

```
Enter to play 1st or 2nd: 2
|   ||   ||   |
---------------
|   ||   || O |
---------------
|   ||   ||   |
---------------
================
Enter a number between 1-9: 5
|   ||   ||   |
---------------
|   || X || O |
---------------
|   ||   ||   |
---------------
================
| O ||   ||   |
---------------
|   || X || O |
---------------
|   ||   ||   |
---------------
================
Enter a number between 1-9: 2
| O || X ||   |
---------------
|   || X || O |
---------------
|   ||   ||   |
---------------
================
| O || X ||   |
---------------
|   || X || O |
---------------
|   || O ||   |
---------------
================
Enter a number between 1-9: 8
Invalid Move! Try again!
Enter a number between 1-9:
```

VIVA QUESTIONS:

1. Define Alpha Beta Pruning.
2. What are the applications of Alpha Beta Pruning?
3. How does Alpha-Beta Pruning work?
4. What are the advantages of using Alpha-Beta Pruning?
5. What is the role of the maximizing and minimizing players in Alpha-Beta Pruning?

MARK ALLOCATION		
Preparation & conduct of experiments	(50)	
Observation & result	(30)	
Record	(10)	
Viva –voce	(10)	
Total	(100)	

RESULT:

Thus Python program to solve Tic Tac Toe problem using Minmax algorithm with Alpha Beta Pruning is implemented successfully.

SIMULATED ANNEALING ALGORITHM

EX NO: 4

DATE:

AIM:

To implement a simulated annealing algorithms using Python.

ALGORITHM:

1. Set an initial temperature (T) and a cooling rate (alpha) between 0 and 1. These parameters control the exploration-exploitation trade-off.
2. Calculate the cost (or objective function value) of the current state. This cost function should quantify how good or bad the solution is for the given optimization problem.
3. Repeat the following steps until a stopping criterion is met
4. Create a neighboring solution by making a small perturbation to the current solution.
5. Calculate the cost of the newly generated solution.
6. Compare the cost of the new solution with the cost of the current solution.
7. Reduce the temperature according to the cooling rate (alpha).
8. Determine when to stop the algorithm. This can be based on reaching a maximum number of iterations.
9. Keep track of the best solution found during the entire search process.

SOURCE CODE:

```
from numpy import asarray, exp
from numpy.random import randn, rand, seed
from matplotlib import pyplot

# Define objective function
def objective(step):
   return step[0] ** 2.0

# Define simulated annealing algorithm
def sa(objective, area, iterations, step_size, temperature):

  # create initial point
   start_point = area[:, 0] + rand( len( area ) ) * ( area[:, 1] - area[:, 0] )

 # evaluate initial point
```

```
    start_point_eval = objective(start_point)
    # Assign previous and new solution to previous and new_point_eval variable
    mia_start_point, mia_start_eval = start_point, start_point_eval
    outputs = []
    for i in range(iterations):
        # First step by mia
        mia_step = mia_start_point + randn( len( area ) ) * step_size
        mia_step_eval = objective(mia_step)
        if mia_step_eval < start_point_eval:
            start_point, start_point_eval = mia_step, mia_step_eval
      #Append the new values into the output list
            outputs.append(start_point_eval)
            print('Acceptance Criteria = %.5f' % mac," ",'iteration Number = ',i," ", 'best_so_far =
',start_point," " ,'new_best = %.5f' % start_point_eval)
        difference = mia_step_eval - mia_start_eval
        t = temperature / float(i + 1)
        # calculate Metropolis Acceptance Criterion / Acceptance Probability
        mac = exp(-difference / t)
        # check whether the new point is acceptable
        if difference < 0 or rand() < mac:
            mia_start_point, mia_start_eval = mia_step, mia_step_eval
    return [start_point, start_point_eval, outputs]

seed(1)
# define the area of the search space
area = asarray([[-6.0, 6.0]])
# initial temperature
temperature = 12
# define the total no. of iterations
iterations = 1200

# define maximum step_size
step_size = 0.1

# perform the simulated annealing search
start_point, output, outputs = sa(objective, area, iterations, step_size, temperature)

#plotting the values
pyplot.plot(outputs, 'ro-')
pyplot.xlabel('Improvement Value')
pyplot.ylabel('Evaluation of Objective Function')
pyplot.show()
```

OUTPUT

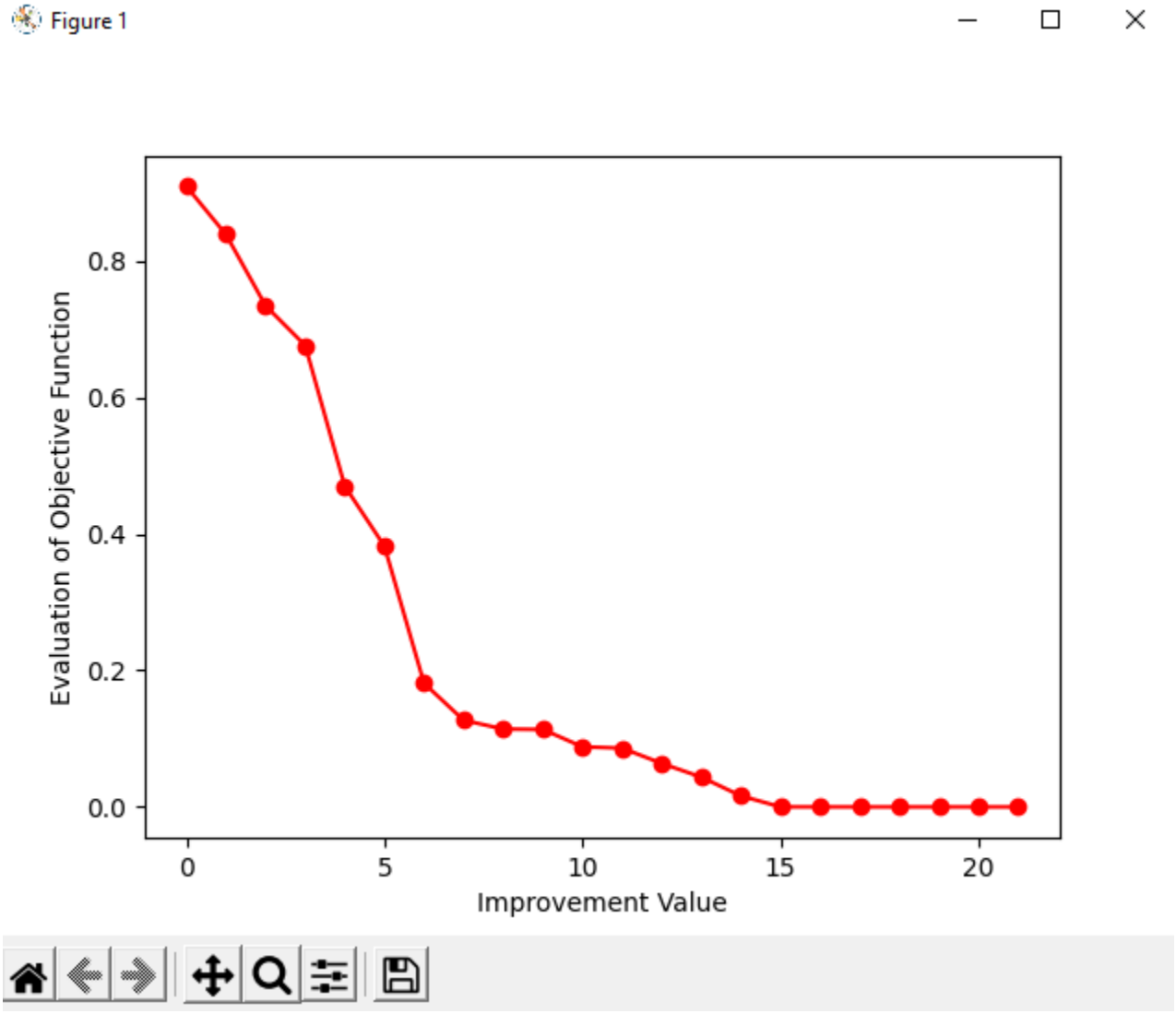


Figure 1
Evaluation of Objective Function
0.0
0.2
0.4
0.6
0.8
0
5
10
15
20
Improvement Value

VIVA QUESTIONS:

1. What is Simulated Annealing, and what is its main purpose in optimization?
2. What are the key components of the Simulated Annealing algorithm?
3. What is the cooling schedule in Simulated Annealing, and why is it important?
4. What is the purpose of the neighborhood generation step in Simulated Annealing?
5. How would you fine-tune the parameters of the Simulated Annealing algorithm for a specific problem?

MARK ALLOCATION		
Preparation & conduct of experiments	(50)	
Observation & result	(30)	
Record	(10)	
Viva –voce	(10)	
Total	(100)	

RESULT:

Thus Simulated Annealing algorithm was implemented successfully.

BACKTRACKING ALGORITHMS FOR CSP

EX NO: 5

DATE:

AIM:

To Implement backtracking algorithms for CSP using python.

ALGORITHM:

1. Create a function calcSubset for generating subsets recursively.
2. Parameters: A (original array), res (result vector), subset (current subset), index (current index).
3. Inside the function:
4. Add subset to res.
5. Iterate from index to the end of A.
6. Include the current element in subset.
7. Recursively call calcSubset with the updated subset and the next index.
8. Remove the current element from subset (backtrack).

SOURCE CODE:

```
def calcSubset(A, res, subset, index):
        # Add the current subset to the result list
        res.append(subset[:])

        # Generate subsets by recursively including and excluding elements
        for i in range(index, len(A)):
                # Include the current element in the subset
                subset.append(A[i])

                # Recursively generate subsets with the current element included
                calcSubset(A, res, subset, i + 1)

                # Exclude the current element from the subset (backtracking)
                subset.pop()


def subsets(A):
        subset = []
        res = []
        index = 0
        calcSubset(A, res, subset, index)
        return res


# Driver code
if __name__ == "__main__":
        array = [1, 2, 3]
```

```
res = subsets(array)

# Print the generated subsets
for subset in res:
        print(*subset)
```

OUTPUT

```
1
1 2
1 2 3
1 3
2
2 3
3
```

VIVA QUESTIONS:

1. What is a Constraint Satisfaction Problem (CSP)?
2. What is the backtracking algorithm, and how does it work in the context of CSPs?
3. What are constraint propagation techniques, and how do they enhance the backtracking algorithm for CSPs?
4. What is the difference between backtracking and depth-first search (DFS)?
5. What are some common challenges or issues that can arise when implementing a backtracking algorithm for CSPs?

MARK ALLOCATION		
Preparation & conduct of experiments	(50)	
Observation & result	(30)	
Record	(10)	
Viva –voce	(10)	
Total	(100)	

RESULT:

Thus the backtracking algorithms for CSP was executed successfully.

LOCAL SEARCH ALGORITHM FOR CSP

EX NO: 6

DATE:

AIM:

To implement local search algorithm for CSP using python.

ALGORITHM:

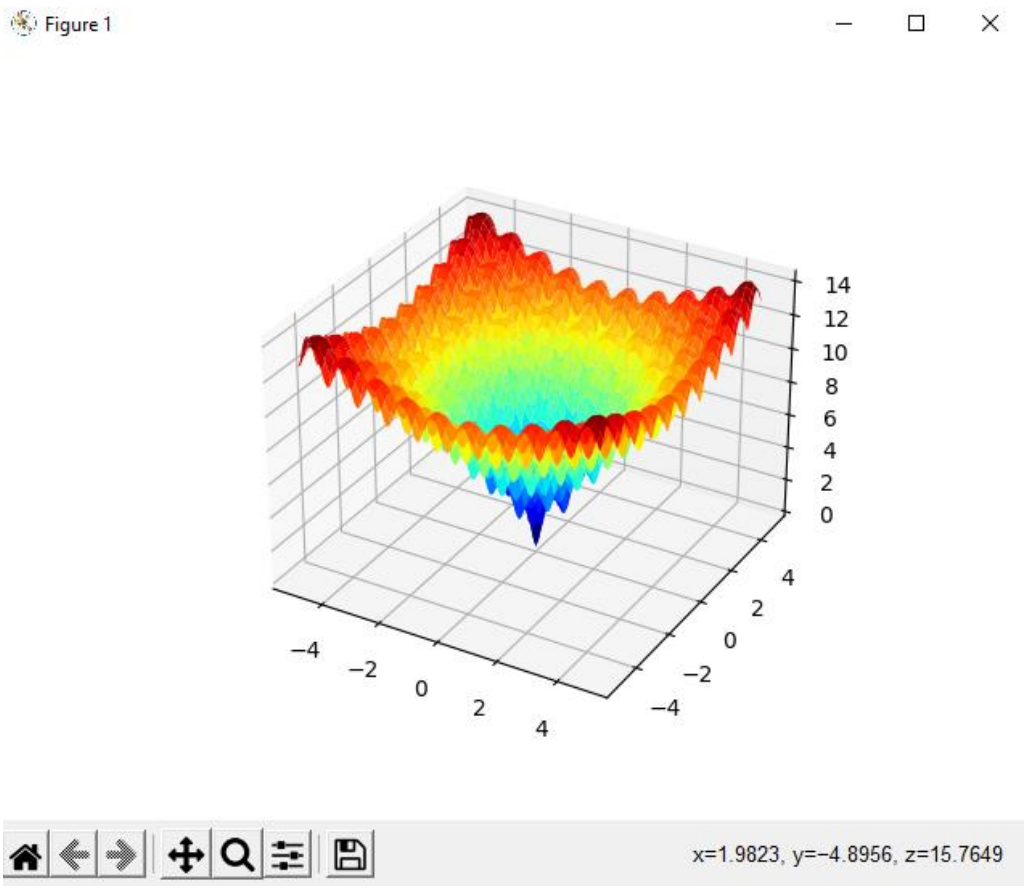
1. Start with a CSP defined by three components like variable, domain, components.
2. Create a fitness function that evaluates how well a particular assignment satisfies the constraints.
3. Start an iterative loop that continues until a termination condition is met.
4. In each iteration, generate neighboring solutions by making small modifications to the current solution.
5. Evaluate the fitness of each neighboring solution using the fitness function.
6. Choose the neighbor with the highest fitness as the new current solution.
7. Check whether the termination condition is met. Once the termination condition is met, return the best solution found during the search.
8. Experiment with different parameter settings, heuristics, and search strategies to optimize the algorithm's performance for your specific CSP.

SOURCE CODE:

```
# ackley multimodal function
from numpy import arange
from numpy import exp
from numpy import sqrt
from numpy import cos
from numpy import e
from numpy import pi
from numpy import meshgrid
from matplotlib import pyplot
from mpl_toolkits.mplot3d import Axes3D

# objective function
def objective(x, y):
	return -20.0 * exp(-0.2 * sqrt(0.5 * (x**2 + y**2))) - exp(0.5 * (cos(2 * pi * x) + cos(2 * pi * y))) + e + 20

# define range for input
r_min, r_max = -5.0, 5.0
# sample input range uniformly at 0.1 increments
xaxis = arange(r_min, r_max, 0.1)
yaxis = arange(r_min, r_max, 0.1)
# create a mesh from the axis
x, y = meshgrid(xaxis, yaxis)
```

```
# compute targets
results = objective(x, y)

# create a surface plot with the jet color scheme
figure = pyplot.figure()
axis = figure.gca(projection='3d')
axis.plot_surface(x, y, results, cmap='jet')

# show the plot
pyplot.show()
```

OUTPUT



VIVA QUESTIONS:

1. Explain the difference between global search and local search algorithms.
2. How does a local search algorithm for CSPs start its search?
3. How does the fitness function help in local search algorithms for CSPs?
4. How can you fine-tune a local search algorithm for a specific CSP problem?

MARK ALLOCATION		
Preparation & conduct of experiments	(50)	
Observation & result	(30)	
Record	(10)	
Viva –voce	(10)	
Total	(100)	

RESULT:

Thus the local search algorithm for CSP was executed successfully.

NAÏVE BAYES MODELS

EX NO: 7

DATE:

Aim:

To build a Naïve Bayes model for Weather Prediction.

Algorithm:

Step 1: Import the necessary modules

Step 2: Define the class Classifier and define the following functions:

- Define the constructor function to calculate probability
- Define calculate_prior() to calculate the individual probabilities
- Define get_cp() to calculate Likelihood of Evidence
- Define calculate_conditional_probabilities() to calculate the conditional probabilities
- Define classify() to print the result

Step 3: Invoke the function calculate_conditional_probabilities() and classify()

Step 4: Stop the program

Program:

```
from functools import reduce
import pandas as pd
import pprint
class Classifier():
    data = None
    class_attr = None
    priori = {}
    cp = {}
    hypothesis = None

  def __init__(self,filename=None, class_attr=None ):
        self.data = pd.read_csv(filename, sep=',', header =(0))
        self.class_attr = class_attr

    '''
        probability(class) =    How many  times it appears in column
                                ____________________________________
                                count of all class attribute
    '''
    def calculate_priori(self):
```

```
        class_values = list(set(self.data[self.class_attr]))
        class_data =  list(self.data[self.class_attr])
        for i in class_values:
            self.priori[i]  = class_data.count(i)/float(len(class_data))
        print ("Priori Values: ", self.priori)

    '''
Here we calculate the individual probabilities
        P(outcome|evidence) =   P(Likelihood of Evidence) x Prior prob of outcome
                                ___________________________________________
                                            P(Evidence)
    '''
    def get_cp(self, attr, attr_type, class_value):
        data_attr = list(self.data[attr])
        class_data = list(self.data[self.class_attr])
        total =1
        for i in range(0, len(data_attr)):
            if class_data[i] == class_value and data_attr[i] == attr_type:
                total+=1
        return total/float(class_data.count(class_value))

    '''
Here we calculate Likelihood of Evidence and multiple all individual probabilities with priori
        (Outcome|Multiple Evidence) = P(Evidence1|Outcome) x P(Evidence2|outcome) x ... x
P(EvidenceN|outcome) x P(Outcome)
        scaled by P(Multiple Evidence)
    '''
    def calculate_conditional_probabilities(self, hypothesis):
        for i in self.priori:
            self.cp[i] = {}
            for j in hypothesis:
                self.cp[i].update({ hypothesis[j]: self.get_cp(j, hypothesis[j], i)})
        print ("\nCalculated Conditional Probabilities: \n")
        pprint.pprint(self.cp)

    def classify(self):
        print ("Result: ")
        for i in self.cp:
            print (i, " ==> ", reduce(lambda x, y: x*y, self.cp[i].values())*self.priori[i])

if __name__ == "__main__":
    c = Classifier(filename="F:\AI LAB EXPTS\weather_data.csv", class_attr="Play" )
    c.calculate_priori()
    c.hypothesis = {"Outlook":'Rainy', "Temp":"Mild", "Humidity":'Normal' , "Windy":'t'}
```

```
c.calculate_conditional_probabilities(c.hypothesis)
c.classify()
```

Output:

```
Priori Values:  {'no': 0.35714285714285715, 'yes': 0.6428571428571429}

Calculated Conditional Probabilities:

{'no': {'Mild': 0.6, 'Normal': 0.4, 'Rainy': 0.8, 't': 0.8},
 'yes': {'Mild': 0.5555555555555556,
         'Normal': 0.7777777777777778,
         'Rainy': 0.3333333333333333,
         't': 0.4444444444444444}}
Result:
no  ==>  0.05485714285714286
yes  ==>  0.04115226337448559
```

VIVA QUESTIONS:

1. Explain the "Naïve" assumption in Naïve Bayes. Why is it called "Naïve"?
2. What types of Naïve Bayes classifiers are there, and how do they differ?
3. What are some advantages and limitations of the Naïve Bayes model?
4. Can Naïve Bayes handle multi-class classification problems? If so, how?

MARK ALLOCATION		
Preparation & conduct of experiments	(50)	
Observation & result	(30)	
Record	(10)	
Viva –voce	(10)	
Total	(100)	

Result:

Thus the Naïve Bayes model was implemented successfully.

BAYESIAN NETWORKS

EX NO: 8

DATE:

Aim:

To implement a Bayesian Network with inference for a heart disease data set using Python.

Algorithm:

Step 1: Import necessary packages.

Step 2: Import a Heart disease dataset

Step 3: Invoke MaximumLikelihoodEstimator and inference the Bayesian network using VariableElimination()

Step 4: Calculate the Probability of Heart disease with given evidence.

Step 5: Stop the Program

Program:

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination

heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())

print('\n Attributes and datatypes')
print(heartDisease.dtypes)

model=BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','he
artdisease'),('heartdisease','restecg'),('heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
```

```
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

Output:

```
Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+
 2. Probability of HeartDisease given evidence= cp
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

VIVA QUESTIONS:

1. Explain the key components of a Bayesian Model.
2. What are the advantages of using Bayesian Models in machine learning and data analysis?
3. How do you handle missing data in Bayesian Models?
4. What challenges or limitations are associated with Bayesian Models?

MARK ALLOCATION		
Preparation & conduct of experiments	(50)	
Observation & result	(30)	
Record	(10)	
Viva –voce	(10)	
Total	(100)	

RESULT:

Thus a Bayesian Network with an inference was built and implemented successfully.

CONTENT BEYOND THE SYLLABUS

PATH PLANNING PROBLEM USING BREADTH FIRST SEARCH

DATE:

Aim:

To solve the path planning problem using Breadth First Search

Algorithm:

Step 1: Import the necessary modules

Step 2: Define the function addEdge to add the edges to the graph.

Step 3: Define the function bfs to perform Breadth First Search

Step 4: Invoke the function bfs() and print the result

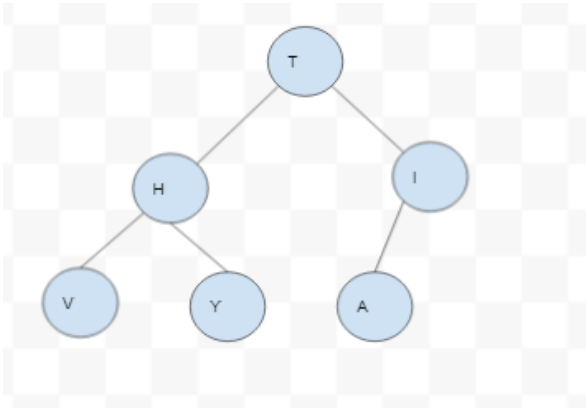
Step 5: Stop the program.

Program:

```
from collections import defaultdict
        graph=defaultdict(list)
def addEdge(u,v):
graph[u].append(v)
addEdge('T','H')
addEdge('T','I')
addEdge('H','V')
addEdge('H','Y')
addEdge('H','T')
addEdge('I','A')
addEdge('I','T')
addEdge('V','H')
addEdge('Y','H')
addEdge('A','I')

def bfs(initial_node):
```

```
    q=list()
    visited=list()
    visited.append(initial_node)
    q.append(initial_node)

    while q:
        temp=q.pop(0)
        print(temp, end=" ")
        for i in graph[temp]:
            if i not in visited:
                q.append(i)
                visited.append(i)
bfs('T')
```

Output:



T H I V Y A

RESULT:

Thus the path planning problem using Breadth First Search was implemented successfully.

MINI PROJECT

EX NO: 9

DATE:

MARK ALLOCATION		
Preparation & conduct of experiments	(50)	
Observation & result	(30)	
Record	(10)	
Viva –voce	(10)	
Total	(100)	

RESULT: