**Artificial Intelligence**

**Manual**

**Students Name :- Anup Patil – A028**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Practical  No | Practical name | date | Page no | Sign |
|  | 1. Write a program to implement depth first search algorithm | 03/07/24 | 2 |  |
|  | 1. Write a program to implement breadth first search algorithm | 03/07/24 | 3 to 4 |  |
| 2 | 1. Write a program to simulate 4-Queen / N-Queen problem. | 12/07/24 | 5 to 6 |  |
|  | 1. Write a program to solve tower of Hanoi problem. | 12/07/24 | 6 |  |
| 3 | b) Write a program for Hill climbing problem | 12/07/24 | 7 to 9 |  |
| 5 | 1. Write a program to solve water jug problem | 24/07/24 | 10 to 12 |  |
|  | 1. Design the simulation of tic – tac – toe game using min-max algorithm. | 24/07/24 | 13 to 15 |  |
| 6 | 1. Write a program to solve Missionaries and Cannibals problem | 25/07/24 | 16 to 20 |  |
| 7 | 1. Write a program to shuffle Deck of cards | 25/07/24 | 21 to 23 |  |
|  | 1. Write a program to shuffle Deck of cards | 25/07/24 | 23 |  |
| 8 | 1. Solve the block of World problem | 25/07/24 | 24 to 25 |  |
| 10 | a)Write a program to derive the predicate.  (for e.g.: Sachin is batsman , batsman is cricketer) - > Sachin is Cricketer. | 26/07/24 | 26 |  |
|  | 1. Write a program which contains three   predicates: male, female, parent. Make  rules for following family relations: father,  mother, grandfather, grandmother,  brother, sister, uncle, aunt, nephew and niece,  cousin.  Question:  i. Draw FamilyTree. | 26/07/24 | 26 to 27 |  |

|  |  |  |
| --- | --- | --- |
| **Practical No 1 (a)** | | Date :- 3/07/2024 |
| Question :- | Write a program to implement depth first search algorithm | |
| Source Code :- | graph = {'A': set(['B', 'C']),  'B': set(['A', 'D', 'E']),  'C': set(['A', 'F']),  'D': set(['B']),  'E': set(['B', 'F']),  'F': set(['C', 'E'])}  def dfs(graph, start):  visited, stack = set(), [start]  while stack:  vertex = stack.pop()  if vertex not in visited:  visited.add(vertex)  stack.extend(graph[vertex] - visited)  return visited  print(dfs(graph, 'A')) # {'E', 'D', 'F', 'A', 'C', 'B'}  def dfs\_paths(graph, start, goal):  stack = [(start, [start])]  while stack:  (vertex, path) = stack.pop()  for next in graph[vertex] - set(path):  if next == goal:  yield path + [next]  else:  stack.append((next, path + [next]))  l1 = list(dfs\_paths(graph, 'A', 'E')) # [['A', 'C', 'F'], ['A', 'B', 'E', 'F']]  print(l1) | |
| Output :- |  | |

|  |  |  |
| --- | --- | --- |
| **Practical No 1 (b)** | | Date :-3/07/2024 |
| Question :- | Write a program to implement breadth first search algorithm | |
| Source Code :- | graph = {'A': set(['B', 'C']),  'B': set(['A', 'D', 'E']),  'C': set(['A', 'F']),  'D': set(['B']),  'E': set(['B', 'F']),  'F': set(['C', 'E'])}  def bfs(graph, start):  visited, queue = set(), [start]  while queue:  vertex = queue.pop(0)  if vertex not in visited:  visited.add(vertex)  queue.extend(graph[vertex] - visited)  return visited  print(bfs(graph, 'C ')) # {'B', 'C', 'A', 'F', 'D', 'E'}  def bfs\_paths(graph, start, goal):  queue = [(start, [start])]  while queue:  (vertex, path) = queue.pop(0)  for next in graph[vertex] - set(path):  if next == goal:  yield path + [next]  else:  queue.append((next, path + [next]))  print(list(bfs\_paths(graph, 'A', 'F'))) # [['A', 'C', 'F'], ['A', 'B', 'E', 'F']]  def shortest\_path(graph, start, goal):  try:  return next(bfs\_paths(graph, start, goal))  except StopIteration:  return None  print(shortest\_path(graph, 'A', 'F')) # ['A', 'C', 'F'] | |
| Output :- | Screenshot 2024-07-30 082531 | |

|  |  |  |
| --- | --- | --- |
| **Practical 2(a)** | | Date 12/07/2024 |
| Question | Write a program to simulate 4-Queen / N-Queen problem. | |
| Source Code | from math import \*  import sys  x={}  n=int(4)  def clear\_future\_blocks(k):  for i in range(k,n+1):  x[i]=None  def place(k,i):  if(i in x.values()):  return False  j=1  while(j<k):  if abs(x[j]-i)==abs(j-k):  return False  j+=1  return True  def NQueens(k):  for i in range(1,n+1):  clear\_future\_blocks(k)  if place(k,i):  x[k]=i  if(k==n):  for j in x:  print(x[j],end='')  for k in range(1,n+1):  if(k==x[j]):  print('Q',end='')  else:  print('.',end='')  print()  print()  print('\_\_\_\_\_')  else:  NQueens(k+1)  NQueens(1) | |
| Output : | Screenshot 2024-07-30 090213 | |

|  |  |  |
| --- | --- | --- |
| **Practical 2 (b)** | | Date 12/07/2024 |
| Question | Write a program to solve tower of Hanoi problem. | |
| Source Code | def moveTower(height,fromPole, toPole, withPole):  if height >= 1:  moveTower(height-1,fromPole,withPole,toPole)  moveDisk(fromPole,toPole)  moveTower(height-1,withPole,toPole,fromPole)  def moveDisk(fp,tp):  print("moving disk from",fp,"to",tp)  moveTower(3,"A","B","C") | |
| Output : | Screenshot 2024-07-30 091127 | |

|  |  |  |
| --- | --- | --- |
| **Practical 3(b)** | | Date :-12/07/2024 |
| Question | Write a program for Hill climbing problem | |
| Source Code | import math  increment = 0.1  startingPoint = [1, 1]  point1 = [1,5]  point2 = [6,4]  point3 = [5,2]  point4 = [2,1]  def distance(x1, y1, x2, y2):  dist = math.pow(x2-x1, 2) + math.pow(y2-y1, 2)  return dist  def sumOfDistances(x1, y1, px1, py1, px2, py2, px3, py3, px4, py4):  d1 = distance(x1, y1, px1, py1)  d2 = distance(x1, y1, px2, py2)  d3 = distance(x1, y1, px3, py3)  d4 = distance(x1, y1, px4, py4)  return d1 + d2 + d3 + d4  def newDistance(x1, y1, point1, point2, point3, point4):  d1 = [x1, y1]  d1temp = sumOfDistances(x1, y1, point1[0],point1[1], point2[0],point2[1],  point3[0],point3[1], point4[0],point4[1] )  d1.append(d1temp)  return d1  minDistance = sumOfDistances(startingPoint[0], startingPoint[1],  point1[0],point1[1], point2[0],point2[1],  point3[0],point3[1], point4[0],point4[1] )  flag = True  def newPoints(minimum, d1, d2, d3, d4):  if d1[2] == minimum:  return [d1[0], d1[1]]  elif d2[2] == minimum:  return [d2[0], d2[1]]  elif d3[2] == minimum:  return [d3[0], d3[1]]  elif d4[2] == minimum:  return [d4[0], d4[1]]  i = 1  while flag:  d1 = newDistance(startingPoint[0]+increment, startingPoint[1], point1, point2,  point3, point4)  d2 = newDistance(startingPoint[0]-increment, startingPoint[1], point1, point2,  point3, point4)  d3 = newDistance(startingPoint[0], startingPoint[1]+increment, point1, point2,  point3, point4)  d4 = newDistance(startingPoint[0], startingPoint[1]-increment, point1, point2,  point3, point4)  print (i,' ', round(startingPoint[0], 2), round(startingPoint[1], 2))  minimum = min(d1[2], d2[2], d3[2], d4[2])  if minimum < minDistance:  startingPoint = newPoints(minimum, d1, d2, d3, d4)  minDistance = minimum  #print i,' ', round(startingPoint[0], 2), round(startingPoint[1], 2)  i+=1  else:  flag = False | |
| Output : | 1 1 1  2 1.1 1  3 1.2 1  4 1.3 1  5 1.4 1  6 1.5 1  7 1.6 1  8 1.6 1.1  9 1.7 1.1  10 1.7 1.2  11 1.7 1.3  12 1.8 1.3  13 1.8 1.4  14 1.9 1.4  15 2.0 1.4  16 2.0 1.5  17 2.1 1.5  18 2.1 1.6  19 2.2 1.6  20 2.2 1.7  21 2.3 1.7  22 2.3 1.8  23 2.3 1.9  24 2.4 1.9  25 2.5 1.9  26 2.5 2.0  27 2.6 2.0  28 2.6 2.1  29 2.7 2.1  30 2.7 2.2  31 2.8 2.2  32 2.8 2.3  33 2.9 2.3  34 2.9 2.4  35 3.0 2.4  36 3.0 2.5  37 3.1 2.5  38 3.1 2.6  39 3.2 2.6  40 3.2 2.7  41 3.2 2.8  42 3.3 2.8  43 3.4 2.8  44 3.4 2.9  45 3.5 2.9  46 3.5 3.0 | |

|  |  |  |
| --- | --- | --- |
| **Practical 5 (a)** | | Date :- 24/07/2024 |
| Question:- | Write a program to solve water jug problem | |
| Code :- | # 3 water jugs capacity -> (x,y,z) where x>y>z  # initial state (12,0,0)  # final state (6,6,0)  capacity = (12,8,5)  # Maximum capacities of 3 jugs -> x,y,z  x = capacity[0]  y = capacity[1]  z = capacity[2]  # to mark visited states memory is a dictionarycontaining key value pair.  memory = {}  # store solution path  ans = []  def get\_all\_states(state):  # Let the 3 jugs be called a,b,c  a = state[0]  b = state[1]  c = state[2]  if(a==6 and b==6):  ans.append(state)  return True  # if current state is already visited earlier  if((a,b,c)in memory):  return False  memory[(a,b,c)] = 1  #empty jug a  if(a>0):  #empty a into b  if(a+b<=y):  if(get\_all\_states((0,a+b,c)) ):  ans.append(state)  return True  else:  if( get\_all\_states((a-(y-b), y, c)) ):  ans.append(state)  return True  #empty a into c  if(a+c<=z):  if(get\_all\_states((0,b,a+c))):  ans.append(state)  return True  else:  if( get\_all\_states((a-(z-c),b,z)) ):  ans.append(state)  return True  #empty jug b  if(b>0):  #empty b into a  if(a+b<=x):  if(get\_all\_states((a+b,0,c)) ):  ans.append(state)  return True  else:  if(get\_all\_states((x,b-(x-a),c)) ):  ans.append(state)  return True  #empty b into c  if(b+c<=z):  if(get\_all\_states((a, 0, b+c ,ans.append(state)))):  return True  else:  if(get\_all\_states((a,b-(z-c),z)) ):  ans.append(state)  return True  #empty jug c  if(c>0):  #empty c into a  if(a+c<=x):  if(get\_all\_states((a+c,b,0))):  ans.append(state)  return True  else:  if(get\_all\_states((x,b,c-(x-a)))):  ans.append(state)  return True  #empty c into b  if(b+c<=y):  if(get\_all\_states((a,b+c,0)) ):  ans.append(state)  return True  else:  if(get\_all\_states((a,y,c-(y-b))) ):  ans.append(state)  return True  return False  initial\_state = (12,0,0)  print("Starting work...\n")  get\_all\_states(initial\_state)  ans.reverse()  for i in ans:  print(i) | |
| Output :- | Screenshot 2024-08-09 092905.png | |

|  |  |  |
| --- | --- | --- |
| **Practical 5 (b)** | | Date:-24/07/2024 |
| Question | Design the simulation of tic – tac – toe game using min-max algorithm. | |
| Source code | import os  import time  board = [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ']  player = 1  ########win Flags##########  Win = 1  Draw = -1  Running = 0  Stop = 1  ###########################9  Game = Running  Mark = 'X'  #This Function Draws Game Board  def DrawBoard():  print(" %c | %c | %c " % (board[1],board[2],board[3]))  print("\_\_\_|\_\_\_|\_\_\_")  print(" %c | %c | %c " % (board[4],board[5],board[6]))  print("\_\_\_|\_\_\_|\_\_\_")  print(" %c | %c | %c " % (board[7],board[8],board[9]))  print(" | | ")  #This Function Checks position is empty or not  def CheckPosition(x):  if(board[x] == ' '):  return True  else:  return False  #This Function Checks player has won or not  def CheckWin():  global Game  #Horizontal winning condition  if(board[1] == board[2] and board[2] == board[3] and board[1] != ' '):  Game = Win  elif(board[4] == board[5] and board[5] == board[6] and board[4] != ' '):  Game = Win  elif(board[7] == board[8] and board[8] == board[9] and board[7] != ' '):  Game = Win  #Vertical Winning Condition  elif(board[1] == board[4] and board[4] == board[7] and board[1] != ' '):  Game = Win  elif(board[2] == board[5] and board[5] == board[8] and board[2] != ' '):  Game = Win  elif(board[3] == board[6] and board[6] == board[9] and board[3] != ' '):  Game=Win  #Diagonal Winning Condition  elif(board[1] == board[5] and board[5] == board[9] and board[5] != ' '):  Game = Win  elif(board[3] == board[5] and board[5] == board[7] and board[5] != ' '):  Game=Win  #Match Tie or Draw Condition  elif(board[1]!=' ' and board[2]!=' ' and board[3]!=' ' and board[4]!=' ' and board[5]!=' ' and board[6]!=' ' and board[7]!=' ' and board[8]!=' ' and board[9]!=' '):  Game=Draw  else:  Game=Running  print("Tic-Tac-Toe Game")  print("Anup [X] --- Bhagyashri [O]\n")  print()  print()  print("Please Wait...")  time.sleep(1)  while(Game == Running):  os.system('cls')  DrawBoard()  if(player % 2 != 0):  print("Anup's chance")  Mark = 'X'  else:  print("Bhagyashri's chance")  Mark = 'O'  choice = int(input("Enter the position between [1-9] where you want to mark :"))  if(CheckPosition(choice)):  board[choice] = Mark  player+=1  CheckWin()  os.system('cls')  DrawBoard()  if(Game==Draw):  print("Game Draw")  elif(Game==Win):  player-=1  if(player%2!=0):  print("Anup Won")  else:  print("Bhagyashri Won") | |
| output | Screenshot 2024-08-09 093802.png | |

|  |  |  |
| --- | --- | --- |
| **Practical 6(a)** | | Date:- 25/07/2024 |
| Question | Write a program to solve Missionaries and Cannibals problem. | |
| Source code | import math  # Missionaries and Cannibals Problem  class State():  def \_\_init\_\_(self, cannibalLeft, missionaryLeft, boat, cannibalRight,missionaryRight):  self.cannibalLeft = cannibalLeft  self.missionaryLeft = missionaryLeft  self.boat = boat  self.cannibalRight = cannibalRight  self.missionaryRight = missionaryRight  self.parent = None  def is\_goal(self):  if self.cannibalLeft == 0 and self.missionaryLeft == 0:  return True  else:  return False  def is\_valid(self):  if self.missionaryLeft >= 0 and self.missionaryRight >= 0 and self.cannibalLeft >= 0 and self.cannibalRight >= 0 and (self.missionaryLeft == 0 or self.missionaryLeft >=self.cannibalLeft) and (self.missionaryRight == 0 or self.missionaryRight >= self.cannibalRight):  return True  else:  return False  def \_\_eq\_\_(self, other):  return self.cannibalLeft == other.cannibalLeft and self.missionaryLeft == other.missionaryLeft and self.boat == other.boat and self.cannibalRight == other.cannibalRight and self.missionaryRight == other.missionaryRight  def \_\_hash\_\_(self):  return hash((self.cannibalLeft, self.missionaryLeft, self.boat, self.cannibalRight, self.missionaryRight))  def successors(cur\_state):  children = [];  if cur\_state.boat == 'left':  new\_state = State(cur\_state.cannibalLeft, cur\_state.missionaryLeft - 2, 'right', cur\_state.cannibalRight, cur\_state.missionaryRight + 2)  ## Two missionaries cross left to right.  if new\_state.is\_valid():  new\_state.parent = cur\_state  children.append(new\_state)  new\_state = State(cur\_state.cannibalLeft - 2, cur\_state.missionaryLeft, 'right', cur\_state.cannibalRight + 2, cur\_state.missionaryRight)  ## Two cannibals cross left to right.  if new\_state.is\_valid():  new\_state.parent = cur\_state  children.append(new\_state)  new\_state = State(cur\_state.cannibalLeft - 1, cur\_state.missionaryLeft - 1, 'right', cur\_state.cannibalRight + 1, cur\_state.missionaryRight + 1)  ## One missionary and one cannibal cross left to right.  if new\_state.is\_valid():  new\_state.parent = cur\_state  children.append(new\_state)  new\_state = State(cur\_state.cannibalLeft, cur\_state.missionaryLeft - 1, 'right',cur\_state.cannibalRight, cur\_state.missionaryRight + 1)  ## One missionary crosses left to right.  if new\_state.is\_valid():  new\_state.parent = cur\_state  children.append(new\_state)  new\_state = State(cur\_state.cannibalLeft - 1, cur\_state.missionaryLeft, 'right', cur\_state.cannibalRight + 1, cur\_state.missionaryRight)  ## One cannibal crosses left to right.  if new\_state.is\_valid():  new\_state.parent = cur\_state  children.append(new\_state)  else:  new\_state = State(cur\_state.cannibalLeft, cur\_state.missionaryLeft + 2, 'left', cur\_state.cannibalRight, cur\_state.missionaryRight - 2)  ## Two missionaries cross right to left.  if new\_state.is\_valid():  new\_state.parent = cur\_state  children.append(new\_state)  new\_state = State(cur\_state.cannibalLeft + 2, cur\_state.missionaryLeft, 'left', cur\_state.cannibalRight - 2, cur\_state.missionaryRight)  ## Two cannibals cross right to left.  if new\_state.is\_valid():  new\_state.parent = cur\_state  children.append(new\_state)  new\_state = State(cur\_state.cannibalLeft + 1, cur\_state.missionaryLeft + 1, 'left', cur\_state.cannibalRight - 1, cur\_state.missionaryRight - 1)  ## One missionary and one cannibal cross right to left.  if new\_state.is\_valid():  new\_state.parent = cur\_state  children.append(new\_state)  new\_state = State(cur\_state.cannibalLeft, cur\_state.missionaryLeft + 1, 'left', cur\_state.cannibalRight, cur\_state.missionaryRight - 1)  ## One missionary crosses right to left.  if new\_state.is\_valid():  new\_state.parent = cur\_state  children.append(new\_state)  new\_state = State(cur\_state.cannibalLeft + 1, cur\_state.missionaryLeft, 'left',cur\_state.cannibalRight - 1, cur\_state.missionaryRight)  ## One cannibal crosses right to left.  if new\_state.is\_valid():  new\_state.parent = cur\_state  children.append(new\_state)  return children  def breadth\_first\_search():  initial\_state = State(3,3,'left',0,0)  if initial\_state.is\_goal():  return initial\_state  frontier = list()  explored = set()  frontier.append(initial\_state)  while frontier:  state = frontier.pop(0)  if state.is\_goal():  return state  explored.add(state)  children = successors(state)  for child in children:  if (child not in explored) or (child not in frontier):  frontier.append(child)  return None  def print\_solution(solution):  path = []  path.append(solution)  parent = solution.parent  while parent:  path.append(parent)  parent = parent.parent  for t in range(len(path)):  state = path[len(path) - t - 1]  print ("(" + str(state.cannibalLeft) + "," + str(state.missionaryLeft) + "," + state.boat + "," + str(state.cannibalRight) + "," + str(state.missionaryRight) + ")")  def main():  solution = breadth\_first\_search()  print ("Missionaries and Cannibals solution:")  print ("(cannibalLeft,missionaryLeft,boat,cannibalRight,missionaryRight)")  print\_solution(solution)  # if called from the command line, call main()  if \_\_name\_\_ == "\_\_main\_\_":  main() | |
| Output | Screenshot 2024-08-09 094625.png | |

|  |  |  |
| --- | --- | --- |
| **Practical 7(a)** | | Date:-25/07/2024 |
| Question:- | Write a program to shuffle Deck of cards. | |
| Source code:- | #first let's import random procedures since we will be shuffling  import random  #next, let's start building list holders so we can place our cards in there:  suits = ["Hearts", "Diamonds", "Clubs", "Spades"]  royals = ["J", "Q", "K", "A"]  deck = []  cardfaces = []  #now, let's start using loops to add our content:  for i in range(2,11):  cardfaces.append(str(i)) #this adds numbers 2-10 and converts them to string data  for j in range(4):  cardfaces.append(royals[j]) #this will add the royal faces to the cardbase  for k in range(4):  for l in range(13):  card = (cardfaces[l] + " of " + suits[k])    #this makes each card, cycling through suits, but first through faces  deck.append(card)  #this adds the information to the "full deck" we want to make  #now let's shuffle our deck!  random.shuffle(deck)  #now let's see the cards!  for m in range(52):  print(deck[m]) | |
| Output :- | 6 of Clubs  K of Diamonds  10 of Diamonds  J of Spades  A of Spades  10 of Spades  A of Hearts  9 of Clubs  J of Clubs  7 of Spades  J of Diamonds  2 of Spades  5 of Spades  K of Clubs  5 of Clubs  K of Hearts  3 of Hearts  Q of Clubs  8 of Diamonds  3 of Spades  9 of Diamonds  10 of Hearts  4 of Spades  2 of Diamonds  8 of Hearts  8 of Spades  5 of Hearts  7 of Diamonds  2 of Hearts  4 of Diamonds  Q of Diamonds  A of Diamonds  3 of Clubs  K of Spades  4 of Hearts  6 of Diamonds  6 of Hearts  9 of Spades  7 of Clubs  Q of Spades  8 of Clubs  3 of Diamonds  10 of Clubs  Q of Hearts  A of Clubs  6 of Spades  4 of Clubs  J of Hearts  2 of Clubs | |

|  |  |  |
| --- | --- | --- |
| **Practical 7 (b)** | | Date :-25/07/2024 |
| Question :- | Write a program to shuffle Deck of cards. | |
| Source Code :- | # import modules  import itertools, random  # make a deck of cards  deck = list(itertools.product(range(1,14),['Spade','Heart','Diamond','Club']))  # shuffle the cards  random.shuffle(deck)  # draw five cards  print("You got:")  for i in range(3):  print(deck[i][0], "of", deck[i][1]) | |
| Output :- | Screenshot 2024-08-12 082007.png | |

|  |  |  |
| --- | --- | --- |
| **Practical 8(a)** | | Date :- 25/07/2024 |
| Question | Solve the block of World problem. | |
| Source Code | from \_\_future\_\_ import print\_function  from simpleai.search import CspProblem, backtrack, min\_conflicts,  MOST\_CONSTRAINED\_VARIABLE, HIGHEST\_DEGREE\_VARIABLE,  LEAST\_CONSTRAINING\_VALUE  variables = ('WA', 'NT', 'SA', 'Q', 'NSW', 'V', 'T')  domains = dict((v, ['red', 'green', 'blue']) for v in variables)  def const\_different(variables, values):  return values[0] != values[1] **# expect the value of the neighbors to be different**  constraints = [  (('WA', 'NT'), const\_different),  (('WA', 'SA'), const\_different),  (('SA', 'NT'), const\_different),  (('SA', 'Q'), const\_different),  (('NT', 'Q'), const\_different),  (('SA', 'NSW'), const\_different),  (('Q', 'NSW'), const\_different),  (('SA', 'V'), const\_different),  (('NSW', 'V'), const\_different),  ]  my\_problem = CspProblem(variables, domains, constraints)  print(backtrack(my\_problem))  print(backtrack(my\_problem,  variable\_heuristic=MOST\_CONSTRAINED\_VARIABLE))  print(backtrack(my\_problem,  variable\_heuristic=HIGHEST\_DEGREE\_VARIABLE))  print(backtrack(my\_problem,  value\_heuristic=LEAST\_CONSTRAINING\_VALUE))  print(backtrack(my\_problem,  variable\_heuristic=MOST\_CONSTRAINED\_VARIABLE,  value\_heuristic=LEAST\_CONSTRAINING\_VALUE))  print(backtrack(my\_problem,  variable\_heuristic=HIGHEST\_DEGREE\_VARIABLE,  value\_heuristic=LEAST\_CONSTRAINING\_VALUE))  print(min\_conflicts(my\_problem)) | |
| Output | Screenshot 2024-08-12 083816.png | |

|  |  |  |
| --- | --- | --- |
| **Practical 10(a)** | | Date :- 26/07/2024 |
| Question | Write a program to derive the predicate.  (for e.g.: Sachin is batsman , batsman is cricketer) - > Sachin is Cricketer. | |
| Source Code | batsman(sachin).  cricketer(a).  cricketer(x):-  batsman(x). | |
| Output | Screenshot 2024-08-12 084507.png | |

|  |  |  |
| --- | --- | --- |
| **Practical 10 (b)** | | Date :- 26/07/2024 |
| Question | Write a program which contains three predicates: male, female, parent. Make  rules for following family relations: father, mother, grandfather, grandmother,  brother, sister, uncle, aunt, nephew and niece, cousin.  Question:  i. Draw FamilyTree. | |
| Source Code | female(shrusti).  female(jidnyasa).  female(bhagyashri).  female(sarika).  male(hem).  male(ram).  male(ronak).  male(shiv).  parent(shrusti,ram).  parent(ronak,ram).  parent(ronak,jidnyasa).  parent(ram,sarika).  parent(ram,bhagyashri).  parent(bhagyashri,hem).  parent(ram,shiv).  parent(shiv,hem).  mother(X,Y):- parent(X,Y),female(X).  father(X,Y):- parent(X,Y),male(X).  haschild(X):- parent(X,\_).  sister(X,Y):- parent(Z,X),parent(Z,Y),female(X),X\==Y.  brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),X\==Y. | |
| Output | Screenshot 2024-08-12 090824.png | |