SIMATS SCHOOL OF ENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES COMPUTER SCIENCE AND ENGINEERING

LIST OF EXPERIMENTS

**BRANCH :** Computer Science and Engineering

**YEAR/SEM :** II & III/ IV & VI

**SUB CODE & TITLE :** CSA17- Artificial Intelligence

|  |  |
| --- | --- |
| **PROGRAM NUMBER** | **LIST OF PROGRAMS** |
| 1 | Write a python program to implement Simple Calculator program?  **a=int(input('enter first number'))**  **b=int(input('enter the second number'))**  **print('select choise:')**  **print('1.add\n2.sub\n3.mul\ndiv\npower\n')**  **choise=input('enter the choice:')**  **if choise=='1':**  **print(a,'+',b,'=',a+b)**  **elif choise=='2':**  **print(a,'-',b,'=',a-b)**  **elif choise=='3':**  **print(a,'\*',b,'=',a\*b)**  **elif choise=='4':**  **print(a,'/',b,'=',a/b)**  **elif choise=='5':**  **print(a,'\*\*',b,'=',a\*\*b)**  **else:**  **print('INVALID')** |
| 2 | Write a python program to Add Two Matrices.  X = [[1,2,3],  [4 ,5,6],  [7 ,8,9]]    Y = [[9,8,7],  [6,5,4],  [3,2,1]]      result = [[0,0,0],  [0,0,0],  [0,0,0]]    # iterate through rows  for i in range(len(X)):  # iterate through columns  for j in range(len(X[0])):  result[i][j] = X[i][j] + Y[i][j]    for r in result:  print(r) |
| 3 | Write a python program to Transpose a Ma    X = [[12,7],  [4 ,5],  [3 ,8]]  result = [[0,0,0],  [0,0,0]]  # iterate through rows  for i in range(len(X)):  # iterate through columns  for j in range(len(X[0])):  result[j][i] = X[i][j]  for r in result:  print(r) |
| 4 | Write a python program to sort the sentence in alphabetical order?  a=input('Enter the statement')  l=list(a.split())  l.sort()  for i in l:  print(i) |
| 5 | Write a python program to implement List operations (Nested List, Length, **Concatenation**, **Membership**, **Iteration**, Indexing and **Slicing**)?  str=['kiran','t','e','j','a']  l=len(str)  print(l)  **Concatenation:**  odd=[1,3,5]  print(odd+[7,9])  **Membership:**  list=['k','i','r','a','n']  print('k' in list)  print('a' not in list)  **Iteration:**  for fri in ['yella','koti','naveen']:  print('I love ',fri)  **Slicing:**  mylist=['y','e','l','l','a']  print(mylist[2:5])  print(mylist[3:]) |
| 6 | Write a python program to implement List methods (Add, Append, Extend & Delete).  t=['k','i','r','a']  t.append('n')  print(t)  t.extend('k')  print(t)  t.insert(0,'t')  t.insert(1,'e')  print(t)  t.remove('t')  t.remove('e')  print(t) |
| 7 | Write a python program to Illustrate Different Set Operations?  print("Enter 7 distinct elements in set a: ")  a= set(input() for i in range(7))  print("Enter 7 distinct elements in set b: ")  b= set(input() for i in range(7))  print("A union B is: ", a|b)  print("A intersection b is: ", a&b)  print("Set difference between a and b is: ", a-b)  print("Symmetric Difference between a and b is: ", a^b) |
| 8 | Write a python program to generate Calendar for the given month and year?  import calendar  yy =int(input('yy'))  mm =int(input('mm'))  print(calendar.month(yy, mm)) |
| 9 | Write a python program to remove punctuations from the given string?  p='''!@#$%^&\*()\_~"":;/?<>.,[]{}='''  string='kiran!=is very @#$ ugly &p\*er\*son'  nopunch=''  for char in string:  if char not in p:  nopunch=nopunch+char  print(nopunch) |
| 10 | Write the python program to solve 8-Puzzle problem  import copy  from heapq import heappush, heappop  n = 3  rows = [ 1, 0, -1, 0 ]  cols = [ 0, -1, 0, 1 ]  class priorityQueue:  def \_\_init\_\_(self):  self.heap = []  def push(self, key):  heappush(self.heap, key)  def pop(self):  return heappop(self.heap)  def empty(self):  if not self.heap:  return True  else:  return False  class nodes:    def \_\_init\_\_(self, parent, mats, empty\_tile\_posi,  costs, levels):  self.parent = parent  self.mats = mats  self.empty\_tile\_posi = empty\_tile\_posi  self.costs = costs  self.levels = levels  def \_\_lt\_\_(self, nxt):  return self.costs < nxt.costs  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:  new\_mats = copy.deepcopy(mats)  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]  costs = calculateCosts(new\_mats, final)    new\_nodes = nodes(parent, new\_mats, new\_empty\_tile\_posi,  costs, levels)  return new\_nodes  def printMatsrix(mats):    for i in range(n):  for j in range(n):  print("%d " % (mats[i][j]), end = " ")    print()  def isSafe(x, y):    return x >= 0 and x < n and y >= 0 and y < n  def printPath(root):    if root == None:  return    printPath(root.parent)  printMatsrix(root.mats)  print()  def solve(initial, empty\_tile\_posi, final):  pq = priorityQueue()  costs = calculateCosts(initial, final)  root = nodes(None, initial,  empty\_tile\_posi, costs, 0)  pq.push(root)  while not pq.empty():  minimum = pq.pop()  if minimum.costs == 0:  printPath(minimum)  return  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]):  child = newNodes(minimum.mats,  minimum.empty\_tile\_posi,  new\_tile\_posi,  minimum.levels + 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\_posi = [ 1, 2 ]  solve(initial, empty\_tile\_posi, final) |
| 11 | Write the python program to solve 8-Queen problem  print ("enter the number")  N = int(input())  list= [[0]\*N for \_ in range(N)]  def attack(i, j):  for k in range(0,N):  if list[i][k]==1 or list[k][j]==1:  return True  for k in range(0,N):  for l in range(0,N):  if (k+l==i+j) or (k-l==i-j):  if list[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 (list[i][j]!=1):  list[i][j] = 1  if N\_queens(n-1)==True:  return True  list[i][j] = 0  return False  N\_queens(N)  for i in list:  print (i) |
| 12 | Write the python program for Water Jug Problem  a=int(input("Enter Jug A Capacity: "))  b=int(input("Enter Jug B Capacity: "))  ai=int(input("Initially Water in Jug A: "))  bi=int(input("Initially Water in Jug B: "))  af=int(input("Final State of Jug A: "))  bf=int(input("Final State of Jug B: "))  print("List Of Operations You can Do:\n")  print("1.Fill Jug A Completely\n")  print("2.Fill Jug B Completely\n")  print("3.Empty Jug A Completely\n")  print("4.Empty Jug B Completely\n")  print("5.Pour From Jug A till Jug B filled Completely or A becomes empty\n")  print("6.Pour From Jug B till Jug A filled Completely or B becomes empty\n")  print("7.Pour all From Jug B to Jug A\n")  print("8.Pour all From Jug A to Jug B\n")  while ((ai!=af or bi!=bf)):  op=int(input("Enter the Operation: "))  if(op==1):  ai=a  elif(op==2):  bi=b  elif(op==3):  ai=0  elif(op==4):  bi=0  elif(op==5):  if(b-bi>ai):  bi=ai+bi  ai=0  else:  ai=ai-(b-bi)  bi=b  elif(op==6):  if(a-ai>bi):  ai=ai+bi  bi=0  else:  bi=bi-(a-ai)  ai=a  elif(op==7):  ai=ai+bi  bi=0  elif(op==8):  bi=bi+ai  ai=0  print(ai,bi); |
| 13 | Write the python program for Cript-Arithmetic problem  def solutions():  # letters = ('s', 'e', 'n', 'd', 'm', 'o', 'r', 'y')  all\_solutions = list()  for s in range(9, -1, -1):  for e in range(9, -1, -1):  for n in range(9, -1, -1):  for d in range(9, -1, -1):  for m in range(9, 0, -1):  for o in range(9, -1, -1):  for r in range(9, -1, -1):  for y in range(9, -1, -1):  if len(set([s, e, n, d, m, o, r, y])) == 8:  send = 1000 \* s + 100 \* e + 10 \* n + d  more = 1000 \* m + 100 \* o + 10 \* r + e  money = 10000 \* m + 1000 \* o + 100 \* n + 10 \* e + y  if send + more == money:  all\_solutions.append((send, more, money))  return all\_solutions  print(solutions()) |
| 14 | Write the python program for Missionaries Cannibal problem  boat\_side = "Right"  missionary\_on\_right = 3  cannibals\_on\_right = 3  missionary\_on\_left = 0  cannibals\_on\_left = 0  user\_interface = ''  for i in range(0,missionary\_on\_left):  user\_interface += "\U0001f482"  for i in range(0,cannibals\_on\_left):  user\_interface += "\U0001f479"  user\_interface += '|'  for i in range(0,5):  user\_interface += "\U0001f30a"  user\_interface += "\U0001f6A2|"  for i in range(0,missionary\_on\_right):  user\_interface += "\U0001f482"  for i in range(0,cannibals\_on\_right):  user\_interface += "\U0001f479"  print(user\_interface)  while True:  missionary = int(input("Enter number of Missionary in boat on "+ boat\_side + ":"))  cannibals = int(input("Enter number of Cannibals in boat on " + boat\_side + ":"))    #Maximum number of people one the boat should be 1 or 2  if (missionary+cannibals) != 1 and (missionary+cannibals) != 2:  print("Invalid move")  continue    #Turn based decisions  if boat\_side == "Right":  if missionary > missionary\_on\_right or cannibals > cannibals\_on\_right:  print("Invalid move")  continue    missionary\_on\_right -= missionary  cannibals\_on\_right -= cannibals  missionary\_on\_left += missionary  cannibals\_on\_left += cannibals    boat\_side = "Left"    user\_interface = ''  for i in range(0,missionary\_on\_left):  user\_interface += "\U0001f482"  for i in range(0,cannibals\_on\_left):  user\_interface += "\U0001f479"  user\_interface += '|\U0001f6A2'  for i in range(0,5):  user\_interface += "\U0001f30a"  user\_interface += "|"  for i in range(0,missionary\_on\_right):  user\_interface += "\U0001f482"  for i in range(0,cannibals\_on\_right):  user\_interface += "\U0001f479"  print(user\_interface)    else:  if missionary > missionary\_on\_left or cannibals > cannibals\_on\_left:  print("Invalid move")  continue    missionary\_on\_right += missionary  cannibals\_on\_right += cannibals  missionary\_on\_left -= missionary  cannibals\_on\_left -= cannibals    boat\_side = "Right"    user\_interface = ''  for i in range(0,missionary\_on\_left):  user\_interface += "\U0001f482"  for i in range(0,cannibals\_on\_left):  user\_interface += "\U0001f479"  user\_interface += '|'  for i in range(0,5):  user\_interface += "\U0001f30a"  user\_interface += "\U0001f6A2|"  for i in range(0,missionary\_on\_right):  user\_interface += "\U0001f482"  for i in range(0,cannibals\_on\_right):  user\_interface += "\U0001f479"  print(user\_interface)    #condition for losing  if (missionary\_on\_right != 0 and missionary\_on\_right < cannibals\_on\_right) or (missionary\_on\_left != 0 and missionary\_on\_left < cannibals\_on\_left):  print("YOU LOSE")  break    #condition for winning  if missionary\_on\_left == 3 and cannibals\_on\_left == 3:  print("YOU WIN")  break  print("GAME OVER") |
| 15 | Write the python program for Vacuum Cleaner problem  def vacuum\_world():  goal\_state = {'A': '0', 'B': '0'}  cost = 0  location\_input = input("Enter Location of Vacuum") #user\_input of location vacuum is placed  status\_input = input("Enter status of " + location\_input) #user\_input if location is dirty or clean  status\_input\_complement = input("Enter status of other room")  print("Initial Location Condition" + str(goal\_state))  if location\_input == 'A':  # Location A is Dirty.  print("Vacuum is placed in Location A")  if status\_input == '1':  print("Location A is Dirty.")  # suck the dirt and mark it as clean  goal\_state['A'] = '0'  cost += 1 #cost for suck  print("Cost for CLEANING A " + str(cost))  print("Location A has been Cleaned.")  if status\_input\_complement == '1':  # if B is Dirty  print("Location B is Dirty.")  print("Moving right to the Location B. ")  cost += 1 #cost for moving right  print("COST for moving RIGHT" + str(cost))  # suck the dirt and mark it as clean  goal\_state['B'] = '0'  cost += 1 #cost for suck  print("COST for SUCK " + str(cost))  print("Location B has been Cleaned. ")  else:  print("No action" + str(cost))  # suck and mark clean  print("Location B is already clean.")  if status\_input == '0':  print("Location A is already clean ")  if status\_input\_complement == '1':# if B is Dirty  print("Location B is Dirty.")  print("Moving RIGHT to the Location B. ")  cost += 1 #cost for moving right  print("COST for moving RIGHT " + str(cost))  # suck the dirt and mark it as clean  goal\_state['B'] = '0'  cost += 1 #cost for suck  print("Cost for SUCK" + str(cost))  print("Location B has been Cleaned. ")  else:  print("No action " + str(cost))  print(cost)  # suck and mark clean  print("Location B is already clean.")  else:  print("Vacuum is placed in location B")  # Location B is Dirty.  if status\_input == '1':  print("Location B is Dirty.")  # suck the dirt and mark it as clean  goal\_state['B'] = '0'  cost += 1 # cost for suck  print("COST for CLEANING " + str(cost))  print("Location B has been Cleaned.")  if status\_input\_complement == '1':  # if A is Dirty  print("Location A is Dirty.")  print("Moving LEFT to the Location A. ")  cost += 1 # cost for moving right  print("COST for moving LEFT" + str(cost))  # suck the dirt and mark it as clean  goal\_state['A'] = '0'  cost += 1 # cost for suck  print("COST for SUCK " + str(cost))  print("Location A has been Cleaned.")  else:  print(cost)  # suck and mark clean  print("Location B is already clean.")  if status\_input\_complement == '1': # if A is Dirty  print("Location A is Dirty.")  print("Moving LEFT to the Location A. ")  cost += 1 # cost for moving right  print("COST for moving LEFT " + str(cost))  # suck the dirt and mark it as clean  goal\_state['A'] = '0'  cost += 1 # cost for suck  print("Cost for SUCK " + str(cost))  print("Location A has been Cleaned. ")  else:  print("No action " + str(cost))  # suck and mark clean  print("Location A is already clean.")  # done cleaning  print("GOAL STATE: ")  print(goal\_state)  print("Performance Measurement: " + str(cost))  vacuum\_world() |
| 16 | Write the python program to implement BFS.  graph = {  'A': ['B', 'C', "D"],  'B': ['E', "F"],  'C': ['G', "I"],  'D': ["I"],  'E': [],  "F": [],  'G': [],  "I": []  }  def bfs(visit\_complete, graph, current\_node):  visit\_complete.append(current\_node)  queue = []  queue.append(current\_node)  while queue:  s = queue.pop(0)  print(s)  for neighbour in graph[s]:  if neighbour not in visit\_complete:  visit\_complete.append(neighbour)  queue.append(neighbour)  bfs([], graph, 'A') |
| 17 | Write the python program to implement DFS.  graph = {  '5' : ['3','7'],  '3' : ['2', '4'],  '7' : ['8'],  '2' : [],  '4' : ['8'],  '8' : []  }  visited = set()  def dfs(visited, graph, node):  if node not in visited:  print (node)  visited.add(node)  for neighbour in graph[node]:  dfs(visited, graph, neighbour)  print("Following is the Depth-First Search")  dfs(visited, graph, '5') |
| 18 | Write the python to implement Travelling Salesman Problem  from sys import maxsize  from itertools import permutations  V = 4  def travellingSalesmanProblem(graph, s):  vertex = []  for i in range(V):  if i != s:  vertex.append(i)  min\_path = maxsize  next\_permutation=permutations(vertex)  for i in next\_permutation:  current\_pathweight = 0  k = s  for j in i:  current\_pathweight += graph[k][j]  k = j  current\_pathweight += graph[k][s]  min\_path = min(min\_path, current\_pathweight)  return min\_path  if \_\_name\_\_ == "\_\_main\_\_":  graph = [[0, 10, 15, 20], [10, 0, 35, 25],  [15, 35, 0, 30], [20, 25, 30, 0]]  s = 0  print(travellingSalesmanProblem(graph, s)) |
| 19 | Write the python program to implement A\* algorithm  class Node():  """A node class for A\* Pathfinding"""  def \_\_init\_\_(self, parent=None, position=None):  self.parent = parent  self.position = position  self.g = 0  self.h = 0  self.f = 0  def \_\_eq\_\_(self, other):  return self.position == other.position  def astar(maze, start, end):  """Returns a list of tuples as a path from the given start to the given end in the given maze"""  # Create start and end node  start\_node = Node(None, start)  start\_node.g = start\_node.h = start\_node.f = 0  end\_node = Node(None, end)  end\_node.g = end\_node.h = end\_node.f = 0  # Initialize both open and closed list  open\_list = []  closed\_list = []  # Add the start node  open\_list.append(start\_node)  # Loop until you find the end  while len(open\_list) > 0:  # Get the current node  current\_node = open\_list[0]  current\_index = 0  for index, item in enumerate(open\_list):  if item.f < current\_node.f:  current\_node = item  current\_index = index  # Pop current off open list, add to closed list  open\_list.pop(current\_index)  closed\_list.append(current\_node)  # Found the goal  if current\_node == end\_node:  path = []  current = current\_node  while current is not None:  path.append(current.position)  current = current.parent  return path[::-1] # Return reversed path  # Generate children  children = []  for new\_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]: # Adjacent squares  # Get node position  node\_position = (current\_node.position[0] + new\_position[0], current\_node.position[1] + new\_position[1])  # Make sure within range  if node\_position[0] > (len(maze) - 1) or node\_position[0] < 0 or node\_position[1] > (len(maze[len(maze)-1]) -1) or node\_position[1] < 0:  continue  # Make sure walkable terrain  if maze[node\_position[0]][node\_position[1]] != 0:  continue  # Create new node  new\_node = Node(current\_node, node\_position)  # Append  children.append(new\_node)  # Loop through children  for child in children:  # Child is on the closed list  for closed\_child in closed\_list:  if child == closed\_child:  continue  # Create the f, g, and h values  child.g = current\_node.g + 1  child.h = ((child.position[0] - end\_node.position[0]) \*\* 2) + ((child.position[1] - end\_node.position[1]) \*\* 2)  child.f = child.g + child.h  # Child is already in the open list  for open\_node in open\_list:  if child == open\_node and child.g > open\_node.g:  continue  # Add the child to the open list  open\_list.append(child)  def main():  maze = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]  start = (0, 0)  end = (7, 6)  path = astar(maze, start, end)  print(path)  if \_\_name\_\_ == '\_\_main\_\_':  main() |
| 20 | Write the python program for Map Coloring to implement CSP.  G = [[ 0, 1, 1, 0, 1, 0],  [ 1, 0, 1, 1, 0, 1],  [ 1, 1, 0, 1, 1, 0],  [ 0, 1, 1, 0, 0, 1],  [ 1, 0, 1, 0, 0, 1],  [ 0, 1, 0, 1, 1, 0]]  node = "abcdef"  t\_={}  for i in range(len(G)):  t\_[node[i]] = i  degree =[]  for i in range(len(G)):  degree.append(sum(G[i]))  colorDict = {}  for i in range(len(G)):  colorDict[node[i]]=["Blue","Red","Yellow","Green"]  sortedNode=[]  indeks = []  for i in range(len(degree)):  \_max = 0  j = 0  for j in range(len(degree)):  if j not in indeks:  if degree[j] > \_max:  \_max = degree[j]  idx = j  indeks.append(idx)  sortedNode.append(node[idx])  theSolution={}  for n in sortedNode:  setTheColor = colorDict[n]  theSolution[n] = setTheColor[0]  adjacentNode = G[t\_[n]]  for j in range(len(adjacentNode)):  if adjacentNode[j]==1 and (setTheColor[0] in colorDict[node[j]]):  colorDict[node[j]].remove(setTheColor[0])  for t,w in sorted(theSolution.items()):  print("Node",t," = ",w) |
| 21 | Write the python program for Tic Tac Toe game  import random  class TicTacToe:  def \_\_init\_\_(self):  self.board = []  def create\_board(self):  for i in range(3):  row = []  for j in range(3):  row.append('-')  self.board.append(row)  def get\_random\_first\_player(self):  return random.randint(0, 1)  def fix\_spot(self, row, col, player):  self.board[row][col] = player  def is\_player\_win(self, player):  win = None  n = len(self.board)  # checking rows  for i in range(n):  win = True  for j in range(n):  if self.board[i][j] != player:  win = False  break  if win:  return win  # checking columns  for i in range(n):  win = True  for j in range(n):  if self.board[j][i] != player:  win = False  break  if win:  return win  # checking diagonals  win = True  for i in range(n):  if self.board[i][i] != player:  win = False  break  if win:  return win  win = True  for i in range(n):  if self.board[i][n - 1 - i] != player:  win = False  break  if win:  return win  return False  for row in self.board:  for item in row:  if item == '-':  return False  return True  def is\_board\_filled(self):  for row in self.board:  for item in row:  if item == '-':  return False  return True  def swap\_player\_turn(self, player):  return 'X' if player == 'O' else 'O'  def show\_board(self):  for row in self.board:  for item in row:  print(item, end=" ")  print()  def start(self):  self.create\_board()  player = 'X' if self.get\_random\_first\_player() == 1 else 'O'  while True:  print(f"Player {player} turn")  self.show\_board()  # taking user input  row, col = list(  map(int, input("Enter row and column numbers to fix spot: ").split()))  print()  # fixing the spot  self.fix\_spot(row - 1, col - 1, player)  # checking whether current player is won or not  if self.is\_player\_win(player):  print(f"Player {player} wins the game!")  break  # checking whether the game is draw or not  if self.is\_board\_filled():  print("Match Draw!")  break  # swapping the turn  player = self.swap\_player\_turn(player)  # showing the final view of board  print()  self.show\_board()  # starting the game  tic\_tac\_toe = TicTacToe()  tic\_tac\_toe.start() |
| 22 | Write the python program to implement Minimax algorithm for gaming |
| 23 | Write the python program to implement Apha & Beta pruning algorithm for gaming |
| 24 | Write the python program to implement Decision Tree |
| 25 | Write the python program to implement Feed forward neural Network  import numpy as np  def relu(n):  if n<0:  return 0  else:  return n  inp=np.array([[-1,2],[2,2],[3,3]])  weights=[np.array([3,3]),np.array([1,5]),np.array([3,3]),np.array([1,5]),np.array([2,-1])]  for x in inp :  node0=relu((x\*weights[0]).sum())  node1=relu((x\*weights[1]).sum())  node2=relu(([node0,node1]\*weights[2]).sum())  node3=relu(([node0,node1]\*weights[3]).sum())  op=relu(([node2,node3]\*weights[4]).sum())  print(x,op) |
| 26 | Write a Prolog Program to Sum the Integers from 1 to n.  sum(x+y):-  S is x+y,  write(S). |
| 27 | Write a Prolog Program for A DB WITH NAME, DOB. |
| 28 | Write a Prolog Program for STUDENT-TEACHER-SUB-CODE. |
| 29 | Write a Prolog Program for PLANETS DB. |
| 30 | Write a Prolog Program to implement Towers of Hanoi. |
| 31 | Write a Prolog Program to print particular bird can fly or not. Incorporate required queries. |
| 32 | Write the prolog program to implement family tree  Pam, Liz, Ann and Pat are female, while Tom, Bob and Jim are male persons. Using this information, define the following relations:   * Define the “mother” relation: * Define the “father” relation: * Define the “grandfather” relation: * Define the “grandmother” relation: * Define the “sister” relation * Define the “brother” relation   female(pammi).  female(lizza).  female(patty).  female(anny).  male(jimmy).  male(bobby).  male(tomy).  male(pitter).  parent(pammi,bobby).  parent(tomy,bobby).  parent(tomy,lizza).  parent(bobby,anny).  parent(bobby,patty).  parent(patty,jimmy).  parent(bobby,pitter).  parent(pitter,jimmy).  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. |

|  |  |
| --- | --- |
| 33 | Write a Prolog Program to suggest Dieting System based on Disease.  food(idly).  food(dosa).  food(rice).  food(sambar).  food(suger).  quality(idly,1).  quality(dose,1).  quality(rice,1).  quality(sambar,1).  quality(suger,1).  calories(idly,1,100).  calories(dose,1,250).  calories(rice,1,200).  calories(sambar,1,150).  calories(suger,1,30). |
| 34 | Write a Prolog program to implement Monkey Banana Problem  move(state(middle,onbox,middle,hasnot),  grasp,  state(middle,onbox,middle,has)).  move(state(P,onfloor,P,H),  climb,  state(P,onbox,P,H)).  move(state(P1,onfloor,P1,H),  push(P1,P2),  state(P2,onfloor,P2,H)).  move(state(P1,onfloor,B,H),  walk(P1,P2),  state(P2,onfloor,B,H)).  canget(state(\_,\_,\_,has)).  canget(State1):-  move(State1,\_,State2),  canget(State2). |
| 35 | Write a Prolog Program for fruit and its color using Back Tracking.  colour(cherry,red).  colour(banana,yellow).  colour(apple,red).  colour(apple,green).  colour(orange,orange).  fruit(cherry,red).  fruit(banana,yellow).  fruit(apple,red).  fruit(apple,green).  fruit(orange,orange). |
| 36 | Write a Prolog Program to implement Best First Search algorithm  connected(1,7,1).  connected(1,8,1).  connected(1,3,1).  connected(7,4,1).  connected(7,20,1).  connected(7,17,1).  connected(8,6,1).  connected(3,9,1).  connected(3,12,1).  connected(9,19,1).  connected(4,42,1).  connected(20,28,1).  connected(17,10,1).  connected2(X,Y,D) :- connected(X,Y,D).  connected2(X,Y,D) :- connected(Y,X,D).  next\_node(Current, Next, Path) :-  connected2(Current, Next, \_),  not(member(Next, Path)).  breadth\_first(Goal, Goal, \_,[Goal]).  breadth\_first(Start, Goal, Visited, Path) :-  findall(X,  (connected2(X,Start,\_),not(member(X,Visited))),  [T|Extend]),  write(Visited), nl,  append(Visited, [T|Extend], Visited2),  append(Path, [T|Extend], [Next|Path2]),  breadth\_first(Next, Goal, Visited2, Path2). |
| 37 | Write the prolog program for Medical Diagnosis  symptom(amit,fever).  symptom(amit,rash).  symptom(amit,headache).  symptom(amit,runny\_nose).  symptom(kaushal,chills).  symptom(kaushal,fever).  symptom(kaushal,hedache).  symptom(dipen,runny\_nose).  symptom(dipen,rash).  symptom(dipen,flu).  hypothesis(Patient,measels):-  symptom(Patient,fever),  symptom(Patient,cough),  symptom(Patient,conjunctivitis),  symptom(Patient,rash).  hypothesis(Patient,german\_measles) :-  symptom(Patient,fever),  symptom(Patient,headache),  symptom(Patient,runny\_nose),  symptom(Patient,rash).  hypothesis(Patient,flu) :-  symptom(Patient,fever),  symptom(Patient,headache),  symptom(Patient,body\_ache),  symptom(Patient,chills).  hypothesis(Patient,common\_cold) :-  symptom(Patient,headache),  symptom(Patient,sneezing),  symptom(Patient,sore\_throat),  symptom(Patient,chills). |
| 38 | Write a Prolog Program for forward Chaining. Incorporate required queries.  rainy(chennai).  rainy(coimbatore).  rainy(ooty).  cold(ooty).  snowy(X):-  rainy(X),  cold(X). |
| 39 | Write a Prolog Program for backward Chaining. Incorporate required queries.  rainy(chennai).  rainy(coimbatore).  rainy(ooty).  cold(ooty).  snowy(X):-  rainy(X),  cold(X). |
| 40 | Create a Web Blog using Word press to demonstrate Anchor Tag, Title Tag, etc. |

**INTERNAL EXAMINER EXTERNAL EXAMINER**