**SAVEETHA SCHOOL OF ENGINEERING SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**COMPUTER SCIENCE AND ENGINEERING**

**LIST OF EXPERIMENTS**

Subject Code **: MLA 01**

Subject **: ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS**

|  |  |  |
| --- | --- | --- |
| 1 |  | Python program to solve the Water Jug Problem |
| 2 |  | Python program for the [Breadth First Search](https://favtutor.com/blogs/breadth-first-search-python) |
| 3 |  | Python program for the [Depth First Search](https://favtutor.com/blogs/depth-first-search-python) |
| 4 |  | Python program to solve the Cryptarithmetic Puzzles |
| 5 |  | Python program to implement the Minimax Algorithm |
| 6 |  | Python program to solve the 8 queens problem in Python |
| 7 |  | Python program to implement the Tic-Tac-Toe Game |
| 8 |  | Python program to solve the Travelling Salesman Problem |
| 9 |  | Python program to solve the tower of hanoi problem in python |
| 10 |  | Python program to solve the 0-1 Knapsack Problem |
| 11 |  | Python program to implement Decision Tree |
| 12 |  | Python program to solve the 8-PUZZLE PROBLEM |
| 13 |  | Python program to implement Feed forward neural Network |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 14 |  | Prolog program to implement family tree |  |  |  |
| 15 |  | Prolog program to implement fibonacci series |  |  |  |
| 16 |  | Prolog program to find factorial numbers |  |  |  |
| 17 |  | Prolog program to find GCD number |  |  |  |
| 18 |  | Prolog Program To Printing All Elements of a List |  |  |  |
| 19 |  | Prolog program to append an integer into the list |  |  |  |
| 20 |  | Prolog program to list membership |  |  |  |
| 21 |  | ROLOG PROGRAM FOR Healthcare Data |  |  |  |
| 22 |  | Eliminate consecutive duplicates of list elements. |  |  |  |
| 23 |  | Run-length encoding of a list. |  |  |  |
| 24 |  | Truth tables for logical expressions. |  |  |  |
| 25 |  | Prolog - Monkey and Banana Problem |  |  |  |

**PYTHON PROGRAM TO SOLVE WATER JUG PROBLEM**

from collections import defaultdict

jug1, jug2, aim = 4, 3, 2

visited = defaultdict(lambda: False)

def waterJugSolver(amt1, amt2):

if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):

print(amt1, amt2)

return True

if visited[(amt1, amt2)] == False:

print(amt1, amt2)

visited[(amt1, amt2)] = True

return (waterJugSolver(0, amt2) or

waterJugSolver(amt1, 0) or

waterJugSolver(jug1, amt2) or

waterJugSolver(amt1, jug2) or

waterJugSolver(amt1 + min(amt2, (jug1-amt1)),

amt2 - min(amt2, (jug1-amt1))) or

waterJugSolver(amt1 - min(amt1, (jug2-amt2)),

amt2 + min(amt1, (jug2-amt2))))

else:

return False

print("Steps: ")

waterJugSolver(0, 0)

OUTPUT:

**Steps:**

**0 0**

**4 0**

**4 3**

**0 3**

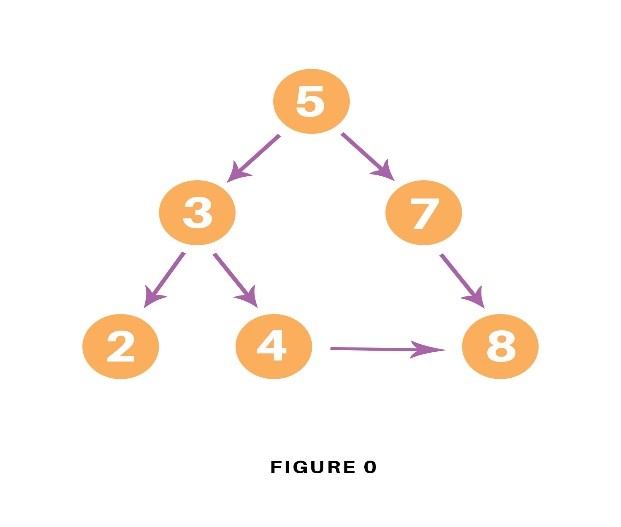
**3 0**

**3 3**

**4 2**

**0 2**

**PYTHON PROGRAM FOR [BREADTH FIRST SEARCH](https://favtutor.com/blogs/breadth-first-search-python)**



graph = {

'5' : ['3','7'],

'3' : ['2', '4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited = [] # List for visited nodes.

queue = [] #Initialize a queue

def bfs(visited, graph, node): #function for BFS

visited.append(node)

queue.append(node)

while queue: # Creating loop to visit each node

m = queue.pop(0)

print (m, end = " ")

for neighbour in graph[m]:

if neighbour not in visited:

visited.append(neighbour)

queue.append(neighbour)

# Driver Code

print("Following is the Breadth-First Search")

bfs(visited, graph, '5') # function calling

OUTPUT:

**Following is the Breadth-First Search**

**> 5 3 7 2 4 8 >**

**PYTHON PROGRAM FOR [DEPTH FIRST SEARCH](https://favtutor.com/blogs/breadth-first-search-python)**

graph = {

'5' : ['3','7'],

'3' : ['2', '4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node): #function for dfs

if node not in visited:

print (node)

visited.add(node)

for neighbour in graph[node]:

dfs(visited, graph, neighbour)

# Driver Code

print("Following is the Depth-First Search")

dfs(visited, graph, '5')

OUTPUT:

**Following is the Depth-First Search**

**5**

**3**

**2**

**4**

**8**

**7**

**>**

**PYTHON PROGRAM TO SOLVE CRYPTARITHMETIC PUZZLES**

def isCryptSolution(crypt, solution):

newsol = list(zip(\*reversed(solution)))

newstring1 = ''

total = 0

for word in range(len(crypt)-1):

subtotal, sol\_total = 0, 0

newstring = ''

for char in crypt[word]:

idx = newsol[0].index(char)

newstring = newstring + newsol[1][idx]

subtotal = int(newstring)

# if newstring[0] == '0':

# return False

total = total + subtotal

for char1 in crypt[-1]:

nidx = newsol[0].index(char1)

newstring1 = newstring1 + newsol[1][nidx]

sol\_total = int(newstring1)

if total == sol\_total and newstring[0] != '0':

return print('True')

elif total == 0 and newstring[0] == '0' and len(newstring) == 1:

return print('True')

else:

return print('False')

crypt = ["SEND", "MORE", "MONEY"]

solution = [['O', '0'],

['M', '1'],

['Y', '2'],

['E', '5'],

['N', '6'],

['D', '7'],

['R', '8'],

['S', '9']]

isCryptSolution(crypt, solution)

OR

def isCryptSolution(crypt, solution):

newsol = list(zip(\*reversed(solution)))

newstring1 = ''

total = 0

for word in range(len(crypt)-1):

subtotal, sol\_total = 0, 0

newstring = ''

for char in crypt[word]:

idx = newsol[0].index(char)

newstring = newstring + newsol[1][idx]

subtotal = int(newstring)

# if newstring[0] == '0':

# return False

total = total + subtotal

for char1 in crypt[-1]:

nidx = newsol[0].index(char1)

newstring1 = newstring1 + newsol[1][nidx]

sol\_total = int(newstring1)

if total == sol\_total and newstring[0] != '0':

return print('True')

elif total == 0 and newstring[0] == '0' and len(newstring) == 1:

return print('True')

else:

return print('False')

crypt = ["BASE", "BAL", "GAMES"]

solution = [['A', '4'],

['B', '2'],

['E', '1'],

['G', '0'],

['L', '5'],

['M', '9'],

['S', '6']]

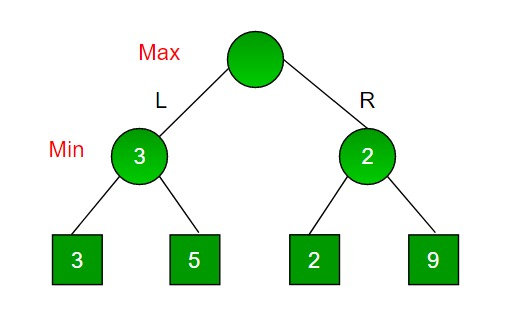
isCryptSolution(crypt, solution)

**OUTPUT:**

**True**

**>**

**PYTHON PROGRAM TO IMPLEMENT MINIMAX ALGORITHM.**



import math

def minimax (curDepth, nodeIndex,

maxTurn, scores,

targetDepth):

if (curDepth == targetDepth):

return scores[nodeIndex]

if (maxTurn):

return max(minimax(curDepth + 1, nodeIndex \* 2,

False, scores, targetDepth),

minimax(curDepth + 1, nodeIndex \* 2 + 1,

False, scores, targetDepth))

else:

return min(minimax(curDepth + 1, nodeIndex \* 2,

True, scores, targetDepth),

minimax(curDepth + 1, nodeIndex \* 2 + 1,

True, scores, targetDepth))

# Driver code

scores = [3, 5, 2, 9]

treeDepth = math.log(len(scores), 2)

print("The optimal value is : ", end = "")

print(minimax(0, 0, True, scores, treeDepth))

**OUTPUT:**

**The optimal value is : 3**

**8 queens problem in Python**

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)

**Tic-Tac-Toe Game in Python**

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()

**TRAVELLING SALESMAN PROBLEM**

# Python3 program to implement traveling salesman

# problem using naive approach.

from sys import maxsize

from itertools import permutations

V = 4

# implementation of traveling Salesman Problem

def travellingSalesmanProblem(graph, s):

# store all vertex apart from source vertex

vertex = []

for i in range(V):

if i != s:

vertex.append(i)

# store minimum weight Hamiltonian Cycle

min\_path = maxsize

next\_permutation=permutations(vertex)

for i in next\_permutation:

# store current Path weight(cost)

current\_pathweight = 0

# compute current path weight

k = s

for j in i:

current\_pathweight += graph[k][j]

k = j

current\_pathweight += graph[k][s]

# update minimum

min\_path = min(min\_path, current\_pathweight)

return min\_path

# Driver Code

if \_\_name\_\_ == "\_\_main\_\_":

# matrix representation of graph

graph = [[0, 10, 15, 20], [10, 0, 35, 25],

[15, 35, 0, 30], [20, 25, 30, 0]]

s = 0

print(travellingSalesmanProblem(graph, s))

Program 9: write a program to solve tower of hanoi problem in python

def TowerOfHanoi(n , source, destination, auxiliary):

if n==1:

print ("Move disk 1 from source",source,"to destination",destination)

return

TowerOfHanoi(n-1, source, auxiliary, destination)

print ("Move disk",n,"from source",source,"to destination",destination)

TowerOfHanoi(n-1, auxiliary, destination, source)

# Driver code

n = 4

TowerOfHanoi(n,'A','B','C')

# A, C, B are the name of rods

**10. Python Program for 0-1 Knapsack Problem**

# a dynamic approach

# Returns the maximum value that can be stored by the bag

def knapSack(W, wt, val, n):

K = [[0 for x in range(W + 1)] for x in range(n + 1)]

#Table in bottom up manner

for i in range(n + 1):

for w in range(W + 1):

if i == 0 or w == 0:

K[i][w] = 0

elif wt[i-1] <= w:

K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w])

else:

K[i][w] = K[i-1][w]

return K[n][W]

#Main

val = [50,100,150,200]

wt = [8,16,32,40]

W = 64

n = len(val)

print(knapSack(W, wt, val, n))

11. Python program to implement Decision Tree

12. Python program to solve the 8-Puzzle Problem

class Node:

def \_\_init\_\_(self,data,level,fval):

""" Initialize the node with the data, level of the node and the calculated fvalue """

self.data = data

self.level = level

self.fval = fval

def generate\_child(self):

""" Generate child nodes from the given node by moving the blank space

either in the four directions {up,down,left,right} """

x,y = self.find(self.data,'\_')

""" val\_list contains position values for moving the blank space in either of

the 4 directions [up,down,left,right] respectively. """

val\_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]

children = []

for i in val\_list:

child = self.shuffle(self.data,x,y,i[0],i[1])

if child is not None:

child\_node = Node(child,self.level+1,0)

children.append(child\_node)

return children

def shuffle(self,puz,x1,y1,x2,y2):

""" Move the blank space in the given direction and if the position value are out

of limits the return None """

if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):

temp\_puz = []

temp\_puz = self.copy(puz)

temp = temp\_puz[x2][y2]

temp\_puz[x2][y2] = temp\_puz[x1][y1]

temp\_puz[x1][y1] = temp

return temp\_puz

else:

return None

def copy(self,root):

""" Copy function to create a similar matrix of the given node"""

temp = []

for i in root:

t = []

for j in i:

t.append(j)

temp.append(t)

return temp

def find(self,puz,x):

""" Specifically used to find the position of the blank space """

for i in range(0,len(self.data)):

for j in range(0,len(self.data)):

if puz[i][j] == x:

return i,j

class Puzzle:

def \_\_init\_\_(self,size):

""" Initialize the puzzle size by the specified size,open and closed lists to empty """

self.n = size

self.open = []

self.closed = []

def accept(self):

""" Accepts the puzzle from the user """

puz = []

for i in range(0,self.n):

temp = input().split(" ")

puz.append(temp)

return puz

def f(self,start,goal):

""" Heuristic Function to calculate hueristic value f(x) = h(x) + g(x) """

return self.h(start.data,goal)+start.level

def h(self,start,goal):

""" Calculates the different between the given puzzles """

temp = 0

for i in range(0,self.n):

for j in range(0,self.n):

if start[i][j] != goal[i][j] and start[i][j] != '\_':

temp += 1

return temp

def process(self):

""" Accept Start and Goal Puzzle state"""

print("Enter the start state matrix \n")

start = self.accept()

print("Enter the goal state matrix \n")

goal = self.accept()

start = Node(start,0,0)

start.fval = self.f(start,goal)

""" Put the start node in the open list"""

self.open.append(start)

print("\n\n")

while True:

cur = self.open[0]

print("")

print(" | ")

print(" | ")

print(" \\\'/ \n")

for i in cur.data:

for j in i:

print(j,end=" ")

print("")

""" If the difference between current and goal node is 0 we have reached the goal node"""

if(self.h(cur.data,goal) == 0):

break

for i in cur.generate\_child():

i.fval = self.f(i,goal)

self.open.append(i)

self.closed.append(cur)

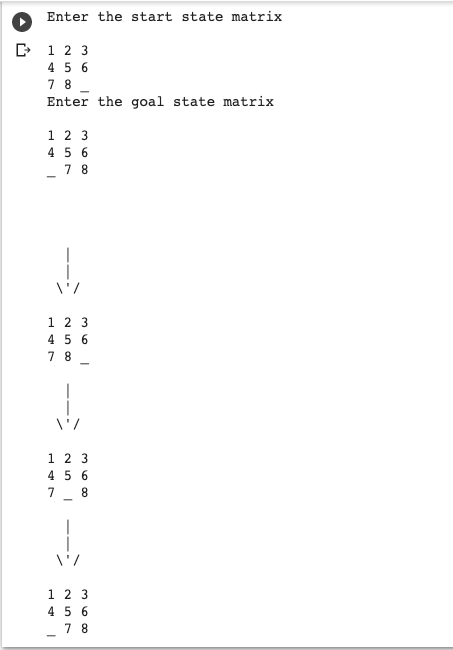
del self.open[0]

""" sort the opne list based on f value """

self.open.sort(key = lambda x:x.fval,reverse=False)

puz = Puzzle(3)

puz.process()



**13. Python program to implement Feed forward neural Network**

import numpy as np

import pandas as pd

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

# Function importing Dataset

def importdata():

balance\_data = pd.read\_csv(

'https://archive.ics.uci.edu/ml/machine-learning-'+

'databases/balance-scale/balance-scale.data',

sep= ',', header = None)

# Printing the dataswet shape

print ("Dataset Length: ", len(balance\_data))

print ("Dataset Shape: ", balance\_data.shape)

# Printing the dataset obseravtions

print ("Dataset: ",balance\_data.head())

return balance\_data

# Function to split the dataset

def splitdataset(balance\_data):

# Separating the target variable

X = balance\_data.values[:, 1:5]

Y = balance\_data.values[:, 0]

# Splitting the dataset into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size = 0.3, random\_state = 100)

return X, Y, X\_train, X\_test, y\_train, y\_test

# Function to perform training with giniIndex.

def train\_using\_gini(X\_train, X\_test, y\_train):

# Creating the classifier object

clf\_gini = DecisionTreeClassifier(criterion = "gini",

random\_state = 100,max\_depth=3, min\_samples\_leaf=5)

# Performing training

clf\_gini.fit(X\_train, y\_train)

return clf\_gini

# Function to perform training with entropy.

def tarin\_using\_entropy(X\_train, X\_test, y\_train):

# Decision tree with entropy

clf\_entropy = DecisionTreeClassifier(

criterion = "entropy", random\_state = 100,

max\_depth = 3, min\_samples\_leaf = 5)

# Performing training

clf\_entropy.fit(X\_train, y\_train)

return clf\_entropy

# Function to make predictions

def prediction(X\_test, clf\_object):

# Predicton on test with giniIndex

y\_pred = clf\_object.predict(X\_test)

print("Predicted values:")

print(y\_pred)

return y\_pred

# Function to calculate accuracy

def cal\_accuracy(y\_test, y\_pred):

print("Confusion Matrix: ",

confusion\_matrix(y\_test, y\_pred))

print ("Accuracy : ",

accuracy\_score(y\_test,y\_pred)\*100)

print("Report : ",

classification\_report(y\_test, y\_pred))

# Driver code

def main():

# Building Phase

data = importdata()

X, Y, X\_train, X\_test, y\_train, y\_test = splitdataset(data)

clf\_gini = train\_using\_gini(X\_train, X\_test, y\_train)

clf\_entropy = tarin\_using\_entropy(X\_train, X\_test, y\_train)

# Operational Phase

print("Results Using Gini Index:")

# Prediction using gini

y\_pred\_gini = prediction(X\_test, clf\_gini)

cal\_accuracy(y\_test, y\_pred\_gini)

print("Results Using Entropy:")

# Prediction using entropy

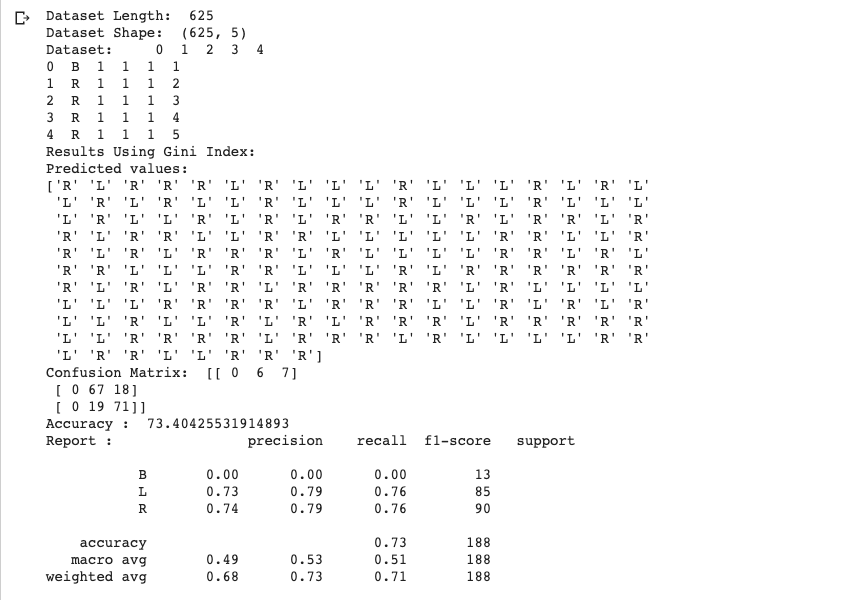
y\_pred\_entropy = prediction(X\_test, clf\_entropy)

cal\_accuracy(y\_test, y\_pred\_entropy)

# Calling main function

if \_\_name\_\_=="\_\_main\_\_":

main()

OUTPUT:

|  |
| --- |
|  |
|  |
|  |
| Write the prolog program for Working with Lists:   1. Printing all elements of a list. 2. To append an integer into the list. 3. List Membership. |
| Prolog program to find factorial number |
| Prolog program to find Armstrong number |
| Prolog program for Healthcare data |
| Create a Web Blog using WordPress |

**14. PROLOG PROGRAM TO IMPLEMENT FAMILY TREE.**

/\* Facts \*/

male(jack).

male(oliver).

male(ali).

male(james).

male(simon).

male(harry).

female(helen).

female(sophie).

female(jess).

female(lily).

parent\_of(jack,jess).

parent\_of(jack,lily).

parent\_of(helen, jess).

parent\_of(helen, lily).

parent\_of(oliver,james).

parent\_of(sophie, james).

parent\_of(jess, simon).

parent\_of(ali, simon).

parent\_of(lily, harry).

parent\_of(james, harry).

/\* Rules \*/

father\_of(X,Y):- male(X),

parent\_of(X,Y).

mother\_of(X,Y):- female(X),

parent\_of(X,Y).

grandfather\_of(X,Y):- male(X),

parent\_of(X,Z),

parent\_of(Z,Y).

grandmother\_of(X,Y):- female(X),

parent\_of(X,Z),

parent\_of(Z,Y).

sister\_of(X,Y):- %(X,Y or Y,X)%

female(X),

father\_of(F, Y), father\_of(F,X),X \= Y.

sister\_of(X,Y):- female(X),

mother\_of(M, Y), mother\_of(M,X),X \= Y.

aunt\_of(X,Y):- female(X),

parent\_of(Z,Y), sister\_of(Z,X),!.

brother\_of(X,Y):- %(X,Y or Y,X)%

male(X),

father\_of(F, Y), father\_of(F,X),X \= Y.

brother\_of(X,Y):- male(X),

mother\_of(M, Y), mother\_of(M,X),X \= Y.

uncle\_of(X,Y):-

parent\_of(Z,Y), brother\_of(Z,X).

ancestor\_of(X,Y):- parent\_of(X,Y).

ancestor\_of(X,Y):- parent\_of(X,Z),

ancestor\_of(Z,Y).

**OUTPUT :**

?-mother\_of(X,jess).

?-parent\_of(X,simon).

?-sister\_of(X,lily).

?-ancestor\_of(X,lily).

**15. PROLOG PROGRAM TO IMPLEMENT FIBONACCI SERIES**

fib(0, 1) :-

!.

fib(n, f) :-

fib(1, n, 1, 1, f).

fib(n, n, \_, f, f) :-

!.

fib(n0, n, f0, f1, f) :-

n1 is n0 + 1,

f2 is f0 + f1,

fib(n1, n, f1, f2, f).

**OUTPUT:**

**Fib** = 1346269

?- time(fib(30, Fib)).

**16. PROLOG PROGRAM TO FIND FACTORIAL NUMBER**

predicates

factorial(integer, real)

go

clauses

go if

write("Enter a positive integer number:"),

readint(N),

factorial(N,Result),

write("Factorial of", N, "is=", Result).

factorial(0, 1)

factorial(N, Result) if N>0,

N1=N-1,

factorial(N1, Res),

Result=N\*Res.

**OUTPUT:**

goals:

factorial(5, Answer)

Answer=120

17. GCD OF TWO NUMBERS

gcd(X,Y):-X=Y,write('GCD of two numbers is '),write(X);

X=0,write('GCD of two numbers is '),write(Y);

Y=0,write('GCD of two numbers is '),write(X);

Y>X,Y1 is Y-X,gcd(X,Y1);

X>Y,Y1 is X-Y,gcd(Y1,Y).



18. PROLOG PROGRAM to PRINTING ALL ELEMENTS OF A LIST

printlist([]).

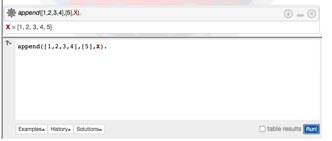
printlist([X|List]) :- write(X),nl, printlist(List).



19. PROLOG PROGRAM TO APPEND AN INTEGER INTO THE LIST

append([],L,L).

append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).



20. PROLOG PROGRAM TO LIST MEMBERSHIP

member(X,List):- delete(X,List,\_).

delete(X,[X|Tail],Tail). delete(X,[Y|Tail1],[Y|Tail2]):-

delete(X,Tail1,Tail2).



21. PROLOG PROGRAM FOR Healthcare Data

domains disease,indication,name=symbol predicates

hypothesis(name,disease) symptom(name,indication)

clauses

symptom(yamini,fever). symptom(yamini,rash) . symptom(yamini,headache). symptom(yamini,runn\_nose). symptom(hemanth,chills). symptom(hemanth,fever). symptom(hemnth,headache). symptom(radhika,runny\_nose). symptom(radhika,rash). symptom(radhika,flu). hypothesis(Patient,measels): symptom(Patient,fever), symptom(Patient,cough), symptom(Patient,conjunctivitis), symptom(Patient,r ash). hypothesis(Patient,german\_measl es): symptom(Patient,f ev er ), symptom(Patient,headache), symptom(Patient,runny\_nose), symptom(Patient,rash).



22. **Eliminate consecutive duplicates of list elements.**

If a list contains repeated elements they should be replaced with a single copy of the element. The order of the elements should not be changed.  
  
Example:  
?- compress([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).  
X = [a,b,c,a,d,e]

23. **Run-length encoding of a list.**

Use the result of problem P09 to implement the so-called run-length encoding data compression method. Consecutive duplicates of elements are encoded as terms [N,E] where N is the number of duplicates of the element E.  
  
Example:  
?- encode([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).  
X = [[4,a],[1,b],[2,c],[2,a],[1,d][4,e]]

24. **Truth tables for logical expressions.**

Define predicates and/2, or/2, nand/2, nor/2, xor/2, impl/2 and equ/2 (for logical equivalence) which succeed or fail according to the result of their respective operations; e.g. and(A,B) will succeed, if and only if both A and B succeed. Note that A and B can be Prolog goals (not only the constants true and fail).

A logical expression in two variables can then be written in prefix notation, as in the following example: and(or(A,B),nand(A,B)).

Now, write a predicate table/3 which prints the truth table of a given logical expression in two variables.

Example:  
?- table(A,B,and(A,or(A,B))).  
true true true  
true fail true  
fail true fail  
fail fail fail

25. **Prolog - Monkey and Banana Problem**

on(floor,monkey).

on(floor,chair).

in(room,monkey).

in(room,chair).

in(room,banana).

at(ceiling,banana).

strong(monkey).

grasp(monkey).

climb(monkey,chair).

push(monkey,chair):-

strong(monkey).

under(banana,chair):-

push(monkey,chair).

canreach(banana,monkey):-

at(floor,banana);

at(ceiling,banana),

under(banana,chair),

climb(monkey,chair).

canget(banana,monkey):-

canreach(banana,monkey),grasp(monkey).