SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be **University** *u/s 3 of UGC Act, 1956)*

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

FACULTY OF ENGINEERING & TECHNOLOGY

(Formerly SRM University, Under section 3 of UGC Act, 1956)

S.R.M. NAGAR, KATTANKULATHUR –603 203, KANCHEEPURAM DISTRICT

SCHOOL OF COMPUTING

DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS

Course Code: 18CSE305J

Course Name: Artificial Intelligence

LAB REPORT

NAME: YUVRAJ SINGH CHAUHAN

REG. NO.:
RA1911027010058

SECTION: N1
CSE – Big Data Analytics

TABLE OF CONTENT

Date: Ex No: 1	**Title of the Lab** Implementation of Toy Program using Python	**Name:** Yuvraj Singh Chauhan **Registration Number:** RA1911027010058 **Section:** N1 **Lab Batch:** 1 **Day Order:** 3

(a) Vacuum Cleaner Problem

AIM: To implement a vacuum cleaner world problem which returns a sequence of actions that leads to the goal state, along with the path cost.

Description of the Concept or Problem given:

Developed a simple reflex agent program in Python for the vacuum-cleaner world problem. This program defines the States, Goal State, Goal Test, Actions, Transition Model, and Path Cost. For each possible initial state, the program returns a sequence of actions that leads to the goal state, along with the path cost.

Manual Solution:

Location need to be entered as A/B (in capitals) where A and B are adjacent rooms respectively.

Status need to be entered as 0/1 where 0 means clean while 1 means dirty.

The location first mentioned will be the place where the vacuum cleaner will be deployed. For every cleaning and moving cost will be incremented.

Program Implementation [Coding]

```
def vacuum_cleaner():
    goal = {'A': '0', 'B': '0'}

    cost = 0
```

```
location_input = input("Enter Location of Vacuum ")
status_input = input("Enter status of " + location_input )
status_input_complement = input("Enter status of other room ")
print("Initial Location Condition " + str(goal))

if location_input == 'A':
    print("Vacuum is placed in Location A ")
    if status_input == '1':
        print("Location A is Dirty.")
        goal['A'] = '0'
        cost += 1
        print("Cost for cleaning A " + str(cost))
        print("Location A has been Cleaned.")

        if status_input_complement == '1':
            print("Location B is Dirty.")
            print("Moving right to the Location B. ")
            cost += 1
            print("Cost for moving right " + str(cost))
            goal['B'] = '0'
            cost += 1
            print("Cost for sucking B " + str(cost))
            print("Location B has been Cleaned. ")
        else:
            print("No action needed for B ")
            print("Location B is already clean.")

    if status_input == '0':
        print("Location A is already clean ")
```

```
        if status_input_complement == '1':
            print("Location B is Dirty.")
            print("Moving right to the Location B. ")
            cost += 1
            print("Cost for moving right " + str(cost))
            goal['B'] = '0'
            cost += 1
            print("Cost for sucking B " + str(cost))
            print("Location B has been Cleaned. ")
        else:
            print("No action needed for B")
            print(cost)
            print("Location B is already clean.")

else:
    print("Vacuum is placed in location B")
    if status_input == '1':
        print("Location B is Dirty.")
        goal['B'] = '0'
        cost += 1
        print("Cost for cleaning " + str(cost))
        print("Location B has been Cleaned.")

        if status_input_complement == '1':
            print("Location A is Dirty.")
            print("Moving left to the Location A. ")
            cost += 1
            print("Cost for moving left " + str(cost))
            goal['A'] = '0'
```

```
            cost += 1
            print("Cost for sucking A " + str(cost))
            print("Location A has been Cleaned.")

        else:
            print(cost)
            print("Location B is already clean.")

            if status_input_complement == '1':
                print("Location A is Dirty.")
                print("Moving left to the Location A. ")
                cost += 1
                print("Cost for moving left " + str(cost))
                goal_state['A'] = '0'
                cost += 1
                print("Cost for sucking A " + str(cost))
                print("Location A has been Cleaned. ")
            else:
                print("No action needed for A")
                print("Location A is already clean.")

    print("Goal State: ")
    print(goal)
    print("Performance Measurement: " + str(cost))

vacuum_cleaner()
```

Screenshots of the Outputs:

```
Enter Location of Vacuum A
Enter status of A1
Enter status of other room 0
Initial Location Condition {'A': '0', 'B': '0'}
Vacuum is placed in Location A
Location A is Dirty.
Cost for cleaning A 1
Location A has been Cleaned.
No action needed for B
Location B is already clean.
Goal State:
{'A': '0', 'B': '0'}
Performance Measurement: 1
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

(b) Framer-Goat-Wolf-Grass Problem

AIM: To implement the Farmer-Goat-Wolf-Grass problem in python.

Description of the Concept or Problem given:

A farmer wants to cross a river but he is not alone. He also has a goat, a wolf, and a grass along with him. There is only one boat available which can support the farmer and either of the goat, wolf or the grass. So at a time, the boat can have only two objects (farmer and one other).

But the problem is, if the goat and wolf are left alone (either in the boat or onshore), the wolf will eat the goat. Similarly, if the Goat and grass are left alone, then goat will eat the grass. The farmer wants to cross the river with all three of his belongings: goat, wolf, and grass.

Manual Solution:

- Taking wolf on other side will leave goat and cabbage together. Also taking away cabbage will make wolf and goat be alone. Hence, the farmer will first take goat on the other side and return back alone. We have farmer, wolf, and cabbage at one side and goat on the other side.
- Now, he will take the wolf along, drop the wolf on the other side and return with the goat. So now on one side, we have farmer, cabbage, and goat and on the other side, we have a wolf.
- Now, he takes the cabbage along and returns alone. So now the scenario is: farmer, goat on one side and wolf, cabbage on the other side.
- Now, finally, he crosses the river with the goat and hence succeeds in taking all his belongings with him.

Program Implementation [Coding]

```
import os
import time

names = {"F": "Farmer",
        "W": "Wolf",
        "G": "Goat",
        "GR": "Grass"}


forbidden_states = [{"W", "G"}, {"G", "GR"}, {"G", "GR", "W"}]

def print_story():
    print("""
#### WOLF, GOAT and CABBAGE PROBLEM ####

Once upon a time a farmer went to a market and purchased a wolf, a goat, and a cabbage. On his way home, the farmer came

to the bank of a river and rented a boat. But crossing the river by boat, the farmer could carry only himself and a single

one of his purchases: the wolf, the goat, or the cabbage.

If left unattended together, the wolf would eat the goat, or the goat would eat the cabbage.

The farmer's challenge was to carry himself and his purchases to the far bank of the river, leaving each purchase intact.

How did he do it?

""")
    input("Press enter to continue.")
```

```
def clear():
    print("*" * 60, "\n")

def print_state(state):
    left_bank, right_bank = state
    print("#### CURRENT STATE OF PUZZLE ####")
    print()
    left_bank_display = [names[item] for item in left_bank]
    right_bank_display = [names[item] for item in right_bank]
    print()

def get_move():
    print("Which item do you wish to take across the river?")
    answer = ""
    while answer.upper() not in ["F", "W", "G", "GR"]:
        answer = input("Just Farmer (f), Wolf (w), Goat (g) or Grass (gr)? ")
    return answer.upper()

def process_move(move, state):
    temp_state = [state[0].copy(), state[1].copy()]
    containing_set = 0 if move in state[0] else 1
    if "F" not in state[containing_set]:
        print("Illegal move.")
        print()
        time.sleep(1)
        return state
    if containing_set == 0:
        temp_state[0].difference_update({move, "F"})
```

```
            temp_state[1].update([move, "F"])
        elif containing_set == 1:
            temp_state[1].difference_update({move, "F"})
            temp_state[0].update([move, "F"])
        if temp_state[0] not in forbidden_states and temp_state[1] not in forbidden_states:
            state = [temp_state[0].copy(), temp_state[1].copy()]
        else:
            print("Illegal move.")
            print()
            time.sleep(1)
        print()
        return state

def is_win(state):
    return state[1] == {"F", "W", "G", "GR"}

def main():
    left_bank = {"F", "W", "G", "GR"}
    right_bank = set()
    state = [left_bank, right_bank]
    print_story()
    while not is_win(state):
        clear()
        print_state(state)
        move = get_move()
        state = process_move(move, state)
    print("Well done - you solved the puzzle!")

main()
```

Screenshots of the Outputs:

```
#### WOLF, GOAT and CABBAGE PROBLEM ####

Once upon a time a farmer went to a market and purchased a wolf, a goat, and a cabbage. On his way home, the farmer came
to the bank of a river and rented a boat. But crossing the river by boat, the farmer could carry only himself and a single
one of his purchases: the wolf, the goat, or the cabbage.

If left unattended together, the wolf would eat the goat, or the goat would eat the cabbage.

The farmer's challenge was to carry himself and his purchases to the far bank of the river, leaving each purchase intact.
How did he do it?

Press enter to continue.
**********************************************************

#### CURRENT STATE OF PUZZLE ####


Which item do you wish to take across the river?
Just Farmer (f), Wolf (w), Goat (g) or Grass (gr)? g

**********************************************************

#### CURRENT STATE OF PUZZLE ####


Which item do you wish to take across the river?
Just Farmer (f), Wolf (w), Goat (g) or Grass (gr)? f

**********************************************************

#### CURRENT STATE OF PUZZLE ####


Which item do you wish to take across the river?
Just Farmer (f), Wolf (w), Goat (g) or Grass (gr)? gr
```

```
Which item do you wish to take across the river?
Just Farmer (f), Wolf (w), Goat (g) or Grass (gr)? gr

************************************************************

#### CURRENT STATE OF PUZZLE ####


Which item do you wish to take across the river?
Just Farmer (f), Wolf (w), Goat (g) or Grass (gr)? g

************************************************************

#### CURRENT STATE OF PUZZLE ####


Which item do you wish to take across the river?
Just Farmer (f), Wolf (w), Goat (g) or Grass (gr)? w

************************************************************

#### CURRENT STATE OF PUZZLE ####


Which item do you wish to take across the river?
Just Farmer (f), Wolf (w), Goat (g) or Grass (gr)? f

************************************************************

#### CURRENT STATE OF PUZZLE ####


Which item do you wish to take across the river?
Just Farmer (f), Wolf (w), Goat (g) or Grass (gr)? g

Well done - you solved the puzzle!
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

Date: Ex No: 2	Title of the Lab Agents and Real world problems	Name: Yuvraj Singh Chauhan Registration Number: RA1911027010058 Section: N1 Lab Batch: 1 Day Order: 3

(a) Wumpus World

AIM: To implement the wumpus world problem in python.

Description of the Concept or Problem given:

The Wumpus world is a cave with 16 rooms (4×4). Each room is connected to others through walkways (no rooms are connected diagonally). The knowledge-based agent starts from Room[1, 1]. The cave has – some pits, a treasure and a beast named Wumpus. The Wumpus can not move but eats the one who enters its room. If the agent enters the pit, it gets stuck there. The goal of the agent is to take the treasure and come out of the cave. The agent is rewarded, when the goal conditions are met. The agent is penalized, when it falls into a pit or being eaten by the Wumpus.

Heuristics used : Early termination, Pure symbols, Unit clauses

Manual Solution:

i. Start from 1,1 and maintain a visited and to_explore sets and a list to mark safe tiles.

ii. Check for stench and breeze and add it to the knowledge base.

iii. Check if there is no wumpus / pit in the adjacent rooms by using dpll algorithm and add corresponding sentence to kb if both are not present and update to_explore set and also update the list with safe tiles.

iv. If we can't say for sure that the adjacent room is safe from both wumpus and pit, check for the possibility that there is wumpus and pit.

v. Explore new tiles from to_explore set.

vi. Keep exploring the to_explore tiles using bfs algorithm to go from one safe tile to another.

vii. If the tile is 4,4 exit.

viii. If the to_explore set is empty, backtrack to 1,1 and start exploring again.

['','P',''], # Rooms [1,1] to [4,1]

['','',''], # Rooms [1,2] to [4,2]

['W','',''], # Rooms [1,3] to [4,3]

['','',''], # Rooms [1,4] to [4,4]

This shows how effective unit clause is in case of our problem which has a lot of unit clauses as we progress deep into the level. This skewed result might also be due to the face that unit clause heuristic in this case also detects failure (p,~p) in this case.

Program Implementation [Coding]

"""

Agent

"""

```
class Agent:
    def __init_(self):
        self._wumpusWorld = [
                ['','P',''], # Rooms [1,1] to [4,1]
                ['','',''], # Rooms [1,2] to [4,2]
                ['W','',''], # Rooms [1,3] to [4,3]
                ['','',''], # Rooms [1,4] to [4,4]
               ] # This is the wumpus world shown in the assignment question.
                 # A different instance of the wumpus world will be used for evaluation.
        self._curLoc = [1,1]
        self._isAlive = True
        self._hasExited = False

    def __FindIndicesForLocation(self,loc):
        x,y = loc
        i,j = y-1, x-1
        return i,j
```

```
    def __CheckForPitWumpus(self):
        ww = self._wumpusWorld
        i,j = self._FindIndicesForLocation(self._curLoc)
        if 'P' in ww[i][j] or 'W' in ww[i][j]:
            print(ww[i][j])
            self._isAlive = False
            print('Agent is DEAD.')
        return self._isAlive


    def TakeAction(self,action): # The function takes an action and returns whether the Agent is alive
                                 # after taking the action.
        validActions = ['Up','Down','Left','Right']
        assert action in validActions, 'Invalid Action.'
        if self._isAlive == False:
            print('Action cannot be performed. Agent is DEAD. Location:{0}'.format(self._curLoc))
            return False
        if self._hasExited == True:
            print('Action cannot be performed. Agent has exited the Wumpus world.'.format(self._curLoc))
            return False


        index = validActions.index(action)
        validMoves = [[0,1],[0,-1],[-1,0],[1,0]]
        move = validMoves[index]
        newLoc = []
        for v, inc in zip(self._curLoc,move):
            z = v + inc #increment location index
```

```
            z = 4 if z>4 else 1 if z<1 else z #Ensure that index is between 1 and 4
            newLoc.append(z)
        self._curLoc = newLoc
        print('Action Taken: {0}, Current Location {1}'.format(action,self._curLoc))
        if self._curLoc[0]==4 and self._curLoc[1]==4:
            self._hasExited=True
        return self._CheckForPitWumpus()

    def __FindAdjacentRooms(self):
        cLoc = self._curLoc
        validMoves = [[0,1],[0,-1],[-1,0],[1,0]]
        adjRooms = []
        for vM in validMoves:
            room = []
            valid = True
            for v, inc in zip(cLoc,vM):
                z = v + inc
                if z<1 or z>4:
                    valid = False
                    break
                else:
                    room.append(z)
            if valid==True:
                adjRooms.append(room)
        return adjRooms


    def PerceiveCurrentLocation(self): #This function perceives the current location.
                                       #It tells whether breeze and stench are present in the current
location.
```

```
        breeze, stench = False, False
        ww = self.__wumpusWorld
        if self._isAlive == False:
            print('Agent cannot perceive. Agent is DEAD. Location:{0}'.format(self._curLoc))
            return [None,None]
        if self._hasExited == True:
            print('Agent cannot perceive. Agent has exited the Wumpus
World.'.format(self._curLoc))
            return [None,None]

        adjRooms = self._FindAdjacentRooms()
        for room in adjRooms:
            i,j = self._FindIndicesForLocation(room)
            if 'P' in ww[i][j]:
                breeze = True
            if 'W' in ww[i][j]:
                stench = True
        return [breeze,stench]

    def FindCurrentLocation(self):
        return self._curLoc

def main():
    ag = Agent()
    print('curLoc',ag.FindCurrentLocation())
    print('Percept [breeze, stench] :',ag.PerceiveCurrentLocation())
    ag.TakeAction('Right')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Right')
    print('Percept',ag.PerceiveCurrentLocation())
```

```
    ag.TakeAction('Right')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Up')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Up')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Up')
    print('Percept',ag.PerceiveCurrentLocation())

if __name_=='_main_':
    main()

"""
Main Program
"""
from Agent import *
import copy
numberOfCalls=0

class KnowledgeBase:
    #clause[i]=-1 represents the presence of negative literal represented by i
    #clause[i]=1 represents the presence of positive literal represented by i
    #values 0 to 15 represents W(1,1) to W(4,4)
    #values 16 to 31 represents S(1,1) to S(4,4)
    #values 33 to 47 represents P(1,1) to P(4,4)
    #values 48 to 63 represents B(1,1) to B(4,4)
    def init (self):
```

```
self.clauses= []

#clauses for atleast 1 Wumpus and 1 Pit
atleast1Wumpus= {}
atleast1Pit = {}
for i in range (16):
    atleast1Wumpus[i]=1
    atleast1Pit[i+32]=1
self.clauses.append(atleast1Wumpus)
self.clauses.append(atleast1Pit)

#clauses for atmost 1 Wumpus and 1 Pit
for i in range(16):
    for j in range(i+1, 16):
        atmost1Wumpus={}
        atmost1Pit={}
        atmost1Wumpus[i]=-1
        atmost1Wumpus[j]=-1
        atmost1Pit[i+32]=-1
        atmost1Pit[j+32]=-1
        self.clauses.append(atmost1Wumpus)
        self.clauses.append(atmost1Pit)

#Stench-Wumpus bijection clauses
for i in range(16):
    stenchWumpusClause={}
    stenchWumpusClause[i+16]=-1
    if (i+4)//4 < 4:
        stenchWumpusClause[i+4]=1
```

```
            stenchClause={}
            stenchClause[i+16]=1
            stenchClause[i+4]=-1
            self.clauses.append(stenchClause)
        if(i-4)//4 >= 0:
            stenchWumpusClause[i-4]=1
            stenchClause={}
            stenchClause[i+16]=1
            stenchClause[i-4]=-1
            self.clauses.append(stenchClause)
        if i//4 == (i+1)//4:
            stenchWumpusClause[i+1]=1
            stenchClause={}
            stenchClause[i+16]=1
            stenchClause[i+1]=-1
            self.clauses.append(stenchClause)
        if i//4 == (i-1)//4:
            stenchWumpusClause[i-1]=1
            stenchClause={}
            stenchClause[i+16]=1
            stenchClause[i-1]=-1
            self.clauses.append(stenchClause)
        self.clauses.append(stenchWumpusClause)

    #Breeze-Pit Bijection Clauses
    for i in range(16):
        breezePitClause={}
        breezePitClause[i+48]=-1
        if(i+4)//4 < 4:
```

```
            breezePitClause[i+4+32]=1
            pitClause={}
            pitClause[i+48]=1
            pitClause[i+4+32]=-1
            self.clauses.append(pitClause)
        if(i-4)//4 >= 0:
            breezePitClause[i-4+32]=1
            pitClause={}
            pitClause[i+48]=1
            pitClause[i-4+32]=-1
            self.clauses.append(pitClause)
        if i//4 == (i+1)//4:
            breezePitClause[i+1+32]=1
            pitClause={}
            pitClause[i+48]=1
            pitClause[i+1+32]=-1
            self.clauses.append(pitClause)
        if i//4 == (i-1)//4:
            breezePitClause[i-1+32]=1
            pitClause={}
            pitClause[i+48]=1
            pitClause[i-1+32]=-1
            self.clauses.append(pitClause)
        self.clauses.append(breezePitClause)

    #No wumpus and pit at [1, 1]
    noWumpusStart={0:-1}
    noPitStart={32:-1}
    self.clauses.append(noWumpusStart)
```

```
        self.clauses.append(noPitStart)

    def AddClause(self, clause): #adding a clause to knowledge base
        self.clauses.append(clause)

    def getclauses(self): #return Wumpus clauses
        return copy.deepcopy(self.clauses)

def FindPureSymbol(clauses, symbols):
    for symbol in symbols:
        positive=0
        negative=0
        for clause in clauses:
            if symbol in clause:
                if clause[symbol]==1:
                    positive= positive+1
                else:
                    negative= negative+1
        if negative==0:
            return symbol, 1
        elif positive==0:
            return symbol, -1
    return -1, 0

def FindUnitClause(clauses):
    for clause in clauses:
        if len(clause)==1:
            for symbol in clause:
```

```
            return symbol, clause[symbol]
    return -1, 0

def selectSymbol(clauses, symbols):
    count={}
    positive={}
    negative={}
    for clause in clauses:
        for literal in clause:
            if literal not in count:
                count[literal]=0
                positive[literal]=0
                negative[literal]=0

            count[literal]= count[literal]+1
            if clause[literal]==1:
                positive[literal]=positive[literal]+1
            else:
                negative[literal]=negative[literal]+1

    maxLiteral= list(symbols.keys())[0]
    maxCount=0
    for literal in count:
        if count[literal]>maxCount:
            maxLiteral= literal
            maxCount= count[literal]

    if positive[maxLiteral]>negative[maxLiteral]:
        return maxLiteral, 1
```

```
    return maxLiteral, -1

def DPLL(clauses, symbols, model):
    global numberOfCalls
    numberOfCalls= numberOfCalls+1
    removeClauses=[]
    for clause in clauses:
        valueUnknown=True
        deleteLiterals=[]
        for literal in clause.keys():
            if literal in model.keys():
                if model[literal]==clause[literal]: #clause is true
                    removeClauses.append(clause)
                    valueUnknown=False
                    break
                else:
                    deleteLiterals.append(literal)

        for literal in deleteLiterals:
            del clause[literal]
        if valueUnknown==True and not bool(clause): #clause is false
            return False

    clauses= [ x for x in clauses if x not in removeClauses]

    if len(clauses)==0: #all clauses are true
        return True

    pureSymbol, value = FindPureSymbol(clauses, symbols)
```

```
    if value!=0:
        del symbols[pureSymbol]
        model[pureSymbol]=value
        return DPLL(clauses, symbols, model)

    unitSymbol, value = FindUnitClause(clauses)
    if value!=0:
        del symbols[unitSymbol]
        model[unitSymbol]=value
        return DPLL(clauses, symbols, model)

    symbol, value= selectSymbol(clauses, symbols)
    del symbols[symbol]
    model[symbol]= value

    if DPLL(copy.deepcopy(clauses), copy.deepcopy(symbols), copy.deepcopy(model)):
        return True

    model[symbol]= -value
    return DPLL(clauses, symbols, model)


def DPLLSatisfiable(clauses):
    symbols={}
    for clause in clauses:
        for literal in clause:
            symbols[literal]=True

    model={}
```

```
    return DPLL(clauses, symbols, model)


def MoveToUnvisited(ag, visited, goalLoc, dfsVisited): #dfs to new safe room
    curPos=ag.FindCurrentLocation()
    curLoc= 4*(curPos[0]-1)+curPos[1]-1
    if(curLoc==goalLoc):
        return True
    dfsVisited[curLoc]=True


    if curPos[1]+1 <=4 and (visited[curLoc+1]==True or (curLoc+1)==goalLoc) and
dfsVisited[curLoc+1]==False:
        ag.TakeAction('Up')
        roomReachable= MoveToUnvisited(ag, visited, goalLoc, dfsVisited)
        if roomReachable:
            return True
        ag.TakeAction('Down')


    if curPos[0]+1 <=4 and (visited[curLoc+4]==True or (curLoc+4)==goalLoc) and
dfsVisited[curLoc+4]==False:
        ag.TakeAction('Right')
        roomReachable= MoveToUnvisited(ag, visited, goalLoc, dfsVisited)
        if roomReachable:
            return True
        ag.TakeAction('Left')


    if curPos[0]-1 >0 and (visited[curLoc-4]==True or (curLoc-4)==goalLoc) and
dfsVisited[curLoc-4]==False:
        ag.TakeAction('Left')
        roomReachable= MoveToUnvisited(ag, visited, goalLoc, dfsVisited)
        if roomReachable:
```

```
            return True
        ag.TakeAction('Right')

    if curPos[1]-1 >0 and (visited[curLoc-1]==True or (curLoc-1)==goalLoc) and
dfsVisited[curLoc-1]==False:
        ag.TakeAction('Down')
        roomReachable= MoveToUnvisited(ag, visited, goalLoc, dfsVisited)
        if roomReachable:
            return True
        ag.TakeAction('Up')

    return False

def ExitWumpusWorld(ag, kb):
    visited = [False for i in range(16)] #Rooms Visited till now
    while(ag.FindCurrentLocation()!=[4, 4]):
        percept= ag.PerceiveCurrentLocation()
        curPos = ag.FindCurrentLocation()
        curLocIndex= 4*(curPos[0]-1)+ curPos[1]-1
        visited[curLocIndex]=True

        breezeClause={}
        stenchClause={}

        if percept[0]==True: #breeze
            breezeClause[curLocIndex+48]=1
        else:
            breezeClause[curLocIndex+48]=-1
        kb.AddClause(breezeClause) #presence/absence of breeze
```

```
        if percept[1]==True: #stench
            stenchClause[curLocIndex+16]=1
        else:
            stenchClause[curLocIndex+16]=-1
        kb.AddClause(stenchClause) #presence/absence of stench

        for newLoc in range(16):
            if visited[newLoc]==False:
                tempclauses= kb.getclauses()
                checkClause={newLoc:1, newLoc+32:1}
                tempclauses.append(checkClause)
                if DPLLSatisfiable(tempclauses)==False:
                    #Room is safe
                    noWumpus={newLoc:-1}
                    noPit={newLoc+32:-1}
                    kb.AddClause(noWumpus)
                    kb.AddClause(noPit)
                    dfsVisited = [False for i in range(16)]
                    roomReachable=MoveToUnvisited(ag, visited, newLoc, dfsVisited) #dfs to new
safe Room
                    if roomReachable:
                        break

def main():
    ag = Agent()
    kb= KnowledgeBase()
    print('Start Location: {0}'.format(ag.FindCurrentLocation()))
    ExitWumpusWorld(ag, kb)
    print('{0} reached. Exiting the Wumpus World.'.format(ag.FindCurrentLocation()))
    print('Total number of times DPLL function is called: {0}'.format(numberOfCalls))
```

```
if __name__=='__main__':
    main()
```

Screenshots of the Outputs:

```
Start Location: [1, 1]
Action Taken: Up, Current Location [1, 2]
Action Taken: Down, Current Location [1, 1]
Action Taken: Right, Current Location [2, 1]
Action Taken: Left, Current Location [1, 1]
Action Taken: Up, Current Location [1, 2]
Action Taken: Down, Current Location [1, 1]
Action Taken: Right, Current Location [2, 1]
Action Taken: Up, Current Location [2, 2]
Action Taken: Left, Current Location [1, 2]
Action Taken: Down, Current Location [1, 1]
Action Taken: Right, Current Location [2, 1]
Action Taken: Left, Current Location [1, 1]
Action Taken: Up, Current Location [1, 2]
Action Taken: Right, Current Location [2, 2]
Action Taken: Up, Current Location [2, 3]
Action Taken: Down, Current Location [2, 2]
Action Taken: Left, Current Location [1, 2]
Action Taken: Down, Current Location [1, 1]
Action Taken: Right, Current Location [2, 1]
Action Taken: Left, Current Location [1, 1]
Action Taken: Up, Current Location [1, 2]
Action Taken: Right, Current Location [2, 2]
Action Taken: Up, Current Location [2, 3]
Action Taken: Up, Current Location [2, 4]
Action Taken: Left, Current Location [1, 4]
Action Taken: Right, Current Location [2, 4]
Action Taken: Down, Current Location [2, 3]
Action Taken: Down, Current Location [2, 2]
Action Taken: Right, Current Location [3, 2]
Action Taken: Up, Current Location [3, 3]
Action Taken: Up, Current Location [3, 4]
Action Taken: Left, Current Location [2, 4]
Action Taken: Left, Current Location [1, 4]

Action Taken: Left, Current Location [2, 4]
Action Taken: Left, Current Location [1, 4]
Action Taken: Right, Current Location [2, 4]
Action Taken: Down, Current Location [2, 3]
Action Taken: Right, Current Location [3, 3]
Action Taken: Down, Current Location [3, 2]
Action Taken: Left, Current Location [2, 2]
Action Taken: Left, Current Location [1, 2]
Action Taken: Down, Current Location [1, 1]
Action Taken: Right, Current Location [2, 1]
Action Taken: Left, Current Location [1, 1]
Action Taken: Up, Current Location [1, 2]
Action Taken: Right, Current Location [2, 2]
Action Taken: Right, Current Location [3, 2]
Action Taken: Up, Current Location [3, 3]
Action Taken: Left, Current Location [2, 3]
Action Taken: Up, Current Location [2, 4]
Action Taken: Right, Current Location [3, 4]
Action Taken: Left, Current Location [2, 4]
Action Taken: Left, Current Location [1, 4]
Action Taken: Right, Current Location [2, 4]
Action Taken: Down, Current Location [2, 3]
Action Taken: Right, Current Location [3, 3]
Action Taken: Down, Current Location [3, 2]
Action Taken: Right, Current Location [4, 2]
Action Taken: Left, Current Location [3, 2]
Action Taken: Up, Current Location [3, 3]
Action Taken: Up, Current Location [3, 4]
Action Taken: Left, Current Location [2, 4]
Action Taken: Left, Current Location [1, 4]
Action Taken: Right, Current Location [2, 4]
Action Taken: Down, Current Location [2, 3]
Action Taken: Down, Current Location [2, 2]
Action Taken: Left, Current Location [1, 2]
Action Taken: Down, Current Location [1, 1]
Action Taken: Right, Current Location [2, 1]
Action Taken: Left, Current Location [1, 1]
Action Taken: Up, Current Location [1, 2]
Action Taken: Right, Current Location [2, 2]

Action Taken: Up, Current Location [2, 3]
Action Taken: Up, Current Location [2, 4]
Action Taken: Right, Current Location [3, 4]
Action Taken: Down, Current Location [3, 3]
Action Taken: Down, Current Location [3, 2]
Action Taken: Right, Current Location [4, 2]
Action Taken: Down, Current Location [4, 1]
Action Taken: Up, Current Location [4, 2]
Action Taken: Up, Current Location [4, 3]
Action Taken: Up, Current Location [4, 4]
[4, 4] reached. Exiting the Wumpus World.
Total number of times DPLL function is called: 2070
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

(b) Tourism Recommendation Problem

AIM: To implement the Tourism Recommendation Problem in python.

Description of the Concept or Problem given:

In the tourism recommendation system, we create an AI program which recommends various tourism related assistance like recommending some famous tourist spots in the city, famous restaurants and hotels, or suggesting some packages to the user according to its constraints.

The program recommends Hotels and Restaurants with respect to price constraints given by the user, and suggests some beautiful tourist spots according to user's wish (historical places, water bodies, hills, seasonal places, famous temples etc).

Manual Solution:

Firstly, suggest what best you have, then ask user what specifically he is looking for

If the user's preference has price constraints, inform about all available price ranges and ask their preferred range

Find products in that price range and recommend the best of them, covering variety of facilities

If the user is interested in the 5 day trip, suggest him/her some exclusive packages (with in detail information of hotel, price offered VS actual price, Tourist places covered). If asked in particular price range, show all the packages available in that price range.

Program Implementation [Coding]

```
import random

Lakes = ["Fateh Sagar Lake", "Lake Pichola", "Udaisagar Lake", "Jaisamand Lake", "Rajsamand Lake", "Doodh Talai", "Govardhan Sagar Lake"]

Historical = ["UDAIPUR CITY PALACE", "LAKE PALACE", "JAG MANDIR", "Sajjangarh Palace", "MONSOON PALACE", "AHAR MUSEUM", "JAGDISH TEMPLE", "SAHELIYON KI BARI", "BAGORE KI HAVELI", "HALDIGHATI", "NAVALAKHA MAHAL", "THE CRYSTAL GALLERY", "NAGDA", "SAHASTRA BAHU TEMPLE"]

Seasonal = ["BHARATIYA LOK KALA MANDAL(August)", "SHILPGRAM(December)" ]

Sites = ["Bahubali Hills", "Neemach Mata Temple", "Sajjangarh Fort"]

Rest = [["Tribute","Ambrai-Amet" ,"HaveliKabab", "MistriSteam"], ["Restaurant1559", "ADCafe RaJheel's" ,"Rooftop Restaurant", "Rainbow Restaurant"] ,["Rainbow Restaurant", "Nataraj", "Sukhidya Chawpati"]]

#4day pakage 2 adults

Hotel = [[["The Leela Palace", 42430,"55,067 Rs", ["AC King Sized Bedroom", "Pool", "Free Parking", "Hot Tub", "Free Wifi", "Breakfast Included", "Lake View", "Restaurant"]],["Taj Lake Palace", 49000 ,"57,200 Rs", ["AC King Sized Bedroom", "Pool", "Free Parking", "Hot Tub", "Free Wifi", "Breakfast Included","Lake View", "Full service Spa"]]],[["Hotel Lakend", 12400, "17,700 Rs", ["AC King Sized Bedroom", "Pool", "Spa", "BathTub", "Free Wifi", "Breakfast Included", "Gym"]], ["Shiv Nivas Palace", 15340, "20,400 Rs", ["AC King Sized Bedroom", "Pool", "Game Zone", "Lake View", "Free Wifi", "Breakfast Included"]]], [["Hotel Royal Palm", 2300, "7,500 Rs", ["AC King Sized Bedroom", "Free Parking", "BathTub", "Free Wifi", "Breakfast Included"]], ["Lake Pichola Hotel", 5551, "9,500 Rs", ["AC King Sized Bedroom", "Pool", "Lake View", "Free Wifi", "Free Parking","Rooftop Restaurant"]]]]

brag = [10000, 15000, 17000, 7000,20000]

def packages(z):

    print("HOTEL DETAILS:\n",z[0]," :  Our Price - ",z[2], "(4Night)"," \t Market Price - ",z[1]+ random.choice(brag),"Rs (4Night)")

    print("\tAnemities :\n", end="\t\t")

    for i in range(len(z[3])):

        print("->",z[3][i], end='\t')

        if (i+1)%4 ==0:

            print(end="\n\t\t")

    print("\n\tTourism Spots covered:\n\tLakes :", end="\n\t\t")
```

```
    x=0
    for i in random.sample(Lakes, 5):
        print("->",i, end= '\t')
        x+=1
        if x%3==0 and x!=0:
            print(end="\n\t\t")
    print("\n\tHistorical Places :",end="\n\t\t")
    x = 0
    for i in random.sample(Historical, 10):
        print("->",i, end='\t\t')
        x += 1
        if x % 3 == 0 and x!=0:
            print(end="\n\t\t")
    print("\n\tHills Covered :",end="\n\t\t")
    x = 0
    for i in random.sample(Sites, 2):
        print("->",i, end='\t')
        x += 1
        if x % 3 == 0 and x!=0:
            print(end="\n\t\t")
    print("\n\tSeasonal Fests (if available) :", end="\n\t\t")
    for i in Seasonal:
        print("->",i, end='\t')
    print('\n')

def hoteldeets(z):
    print(z[0]," :\t",z[1],"Rs (Per Night)")
    print("\tAnemities :", end="\t\t")
```

```
    for i in range(len(z[3])):
        print(z[3][i], end=',\t')
    print("\n")
def Whatuwannado():
    print("\nTell me What can I do for you?\n1. Recommend a Tourist Spot\n2. Recommend some good Restaurants\n3. Tell some Good Hotels\n4. Checkout Our Exclusive Packages\n5. Exit")
    u = int(input("Your Choice: "))
    return u
print("\Welcome! I am your ChatBot, l am here to help you have a wonderful experience in my Favourite City Udaipur!")
#print(len(Historical))
while True:
    u= Whatuwannado()
    if u==1:
        print("\nHere are some Top Rated Places to visit: ")
        for i in random.sample(Lakes, 2):
            print("->", i, end='\t\t')
        for i in random.sample(Sites, 1):
            print("->", i, end='\t\t')
        print(" ")
        for i in random.sample(Historical, 3):
            print("->", i, end='\t\t')
        print('\n->', Seasonal[1])
        print("\nIf you are looking for something specific, Please select:\n1. Lakes\n2. Site Seeing\n3. Historical Places\n4. Seasonal Spots\n5. Main menu\n6. Say Goodbye\n")
        x=int(input("Your Choice: "))
        if(x==1):
            print("\nHere's the list of Top 7 Lakes in the City", end='\n\t')
            for i in range(len(Lakes)):
                print("->", Lakes[i], end='\t\t')
```

```
            if i%3==0 and i!=0:
                print(" ",end="\n\t")
        elif x==2:
            print("\nHere's the list of Top 7 Lakes in the City", end='\n\t')
            for i in range(len(Sites)):
                print("->", Sites[i], end='\t\t')
                if i%3==0 and i!=0:
                    print(" ")
        elif x==3:
            print("\nHere's the list of Top 14 Historical Places of the City", end='\n\t')
            for i in range(len(Historical)):
                print("->", Historical[i], end='\t\t')
                if i%3==0 and i!=0:
                    print(" ",end="\n\t")
        elif x == 4:
            print("\nHere's the list of some Famous Seasonal Fests in the City", end='\n\t')
            print("->", Seasonal[0], '\t\t', Seasonal[1])
        elif x == 5:
            continue
        elif x==6:
            print("\nTime to say Goodbye YoY ....Hope to see u soon\n B-byee", end='\n\t')
            quit()
    elif u==2:
        print("\nHere are some Top Recommended Restaurants: ", end='\n\t')
        z=random.sample(Rest[0], 2)
        print("->", z[0],'\t\t->', z[1], end='\n\t')
        z = random.sample(Rest[1], 1)
        print("->", z[0], end='\t\t')
        z = random.sample(Rest[2], 1)
```

```
    print("->", z[0])
    print(("\nPlease Select your price range for better recommendation:\n1. Royal (Cost for two: 3,500 Rs)\n2. Premium(Cost for two: 1,500 Rs)\n3. Classic (Cost for two: 700 Rs)\n4. Return to Main menu\n5. Say Goodbye\n"))
    x=int(input("Your Choice: "))
    if(x==1):
        print("Here's a list of Royal Restaurants of the City",end="\n\t")
        for i in range(len(Rest[0])):
            print("->", Rest[0][i], end='\t\t')
            if (i+1)%2==0:
                print(" ",end="\n\t")
    elif x==2:
        print("Here's a list of Best Premium Restaurants of the City", end="\n\t")
        for i in range(len(Rest[1])):
            print("->", Rest[1][i], end="\t\t")
            if (i + 1) % 2 == 0:
                print(" ", end="\n\t")
    elif x==3:
        print("Here's a list of Top Classic Restaurants and Foodspots of the City", end="\n\t")
        for i in range(len(Rest[2])):
            print("->", Rest[2][i], end="\t\t")
            if (i + 1) % 2 == 0:
                print(" ", end="\n\t")
    elif x == 4:
        continue
    elif x==5:
        print("\nTime to say Goodbye YoY ....Hope to see u soon\n B-byee")
        quit()
    #     print("Here's the list of some Famous Seasonal Fests in the City")
    #     print(Seasonal[0], '\t', Seasonal[1])
```

```
    elif u==3:
        print("\nHere are some Handpicked Hotels you may like: ")
        z=random.sample(Hotel[0], 1)
        hoteldeets(z[0])
        z = random.sample(Hotel[1], 1)
        hoteldeets(z[0])
        z = random.sample(Hotel[2], 1)
        hoteldeets(z[0])
        print("\nPlease Select your hotel price range for better recommendation:\n1. Royal (Cost Per Night: 45,500 Rs)\n2. Premium(Cost Per Night: 15,500 Rs)\n3. Classic (Cost Per Night: 5,000 Rs)\n4. Return to Main menu\n6. Say Goodbye\n")
        x=int(input("Your Choice: "))
        if(x==1):
            print("\nHere's a list of Handpicked Royal Hotels for you :")
            for i in range(len(Hotel[0])):
                hoteldeets(Hotel[0][i])
        elif x==2:
            print("\nHere's a list of Best Premium Restaurants Hotels for you :")
            for i in range(len(Hotel[1])):
                hoteldeets(Hotel[1][i])
        elif x==3:
            print("\nHere's a list of Top Classic Hotels for you :")
            for i in range(len(Hotel[2])):
                hoteldeets(Hotel[1][i])
        elif x==4:
            continue
        elif x==5:
            print("\nTime to say Goodbye YoY ....Hope to see u soon\n B-byee")
            quit()
    elif u == 4:
```

```
print("\nOur Best 3 Handpicked Packages are: ")
z=random.sample(Hotel[0], 1)
print("Package I - ")
packages(z[0])
z = random.sample(Hotel[1], 1)
print("Package II - ")
packages(z[0])
z = random.sample(Hotel[2], 1)
print("Package III - ")
packages(z[0])


print("\nPlease Select your Trip price range for better recommendation:\n1. Royal Pack (Cost 4Night Trip: 56,000 Rs)\n2. Premium(Cost 4Night Trip: 18,000 Rs)\n3. Classic (Cost 4Night Trip: 8,000 Rs)\n4. Return to Main menu\n5. Nvm I just go to sleep(exit)\n")
x = int(input("Your Choice: "))
if (x == 1):
    print("\nHere's a list of Handpicked Royal Hotels for you :",end=' ')
    for i in range(len(Hotel[0])):
        print("\n\nPackage ",i+1,end=". - \n")
        packages(Hotel[0][i])
elif x == 2:
    print("\nHere's a list of Best Premium Restaurants Hotels for you :")
    for i in range(len(Hotel[1])):
        print("\n\nPackage ",i+1,end=". - \n")
        packages(Hotel[1][i])
elif x == 3:
    print("\nHere's a list of Top Classic Hotels for you :")
    for i in range(len(Hotel[2])):
        print("\n\nPackage ",i+1,end=". -\n")
        packages(Hotel[1][i])
```

```
    elif x==4:
        continue
    elif x==5:
        print("\nTime to say Goodbye YoY ....Hope to see u soon\n B-byee")
        quit()
    elif u==5:
        quit()
```

Screenshots of the Outputs:

```
\Welcome! I am your ChatBot, I am here to help you have a wonderful experience in my Favourite City Udaipur!

Tell me What can I do for you?
1. Recommend a Tourist Spot
2. Recommend some good Restaurants
3. Tell some Good Hotels
4. Checkout Our Exclusive Packages
5. Exit
Your Choice: 2

Here are some Top Recommended Restaurants:
        -> Tribute              -> HaveliKabab
        -> ADCafe RaJheel's             -> Rainbow Restaurant

Please Select your price range for better recommendation:
1. Royal (Cost for two: 3,500 Rs)
2. Premium(Cost for two: 1,500 Rs)
3. Classic (Cost for two: 700 Rs)
4. Return to Main menu
5. Say Goodbye

Your Choice: 2
Here's a list of Best Premium Restaurants of the City
        -> Restaurant1559               -> ADCafe RaJheel's
        -> Rooftop Restaurant           -> Rainbow Restaurant

Tell me What can I do for you?
1. Recommend a Tourist Spot
2. Recommend some good Restaurants
3. Tell some Good Hotels
4. Checkout Our Exclusive Packages
5. Exit
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

Date: **Ex No:** 3	**Title of the Lab** Constraint Satisfaction Problem	**Name:** Yuvraj Singh Chauhan **Registration Number:** RA1911027010058 **Section:** N1 **Lab Batch:** 1 **Day Order:** 3

(a) Cryptarithmetic

AIM: To implement the Cryptarithmetic Problem (CROSS + ROADS = DANGER) problem in python.

Description of the Concept or Problem given:

Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacementeither with alphabets or other symbols. In cryptarithmetic problem, the digits (0-9) get substituted by some possible alphabets or symbols.

The task in cryptarithmetic problem is to substitute each digit withan alphabet to get the result arithmetically correct.

Manual Solution:

We can perform all the arithmetic operations on a givencryptarithmetic problem.

The rules or constraints on a cryptarithmetic problem are asfollows:

- There should be a unique digit to be replaced with a uniquealphabet.
- The result should satisfy the predefined arithmetic rules, i.e.,2+2 =4, nothing else.
- Digits should be from 0-9 only.
- There should be only one carry forward, while performing theaddition operation on a problem.
- The problem can be solved from both sides, i.e., lefthand side(L.H.S), or righthand side (R.H.S).

Program Implementation [Coding]

```
import itertools

def get_value(word, substitution):
```

```
    s = 0
    factor = 1
    for letter in reversed(word):
        s += factor * substitution[letter]
        factor *= 10
    return s

def solve2(equation):
    left, right = equation.lower().replace(' ', '').split('=')
    left = left.split('+')
    letters = set(right)
    for word in left:
        for letter in word:
            letters.add(letter)
    letters = list(letters)

    digits = range(10)
    for perm in itertools.permutations(digits, len(letters)):
        sol = dict(zip(letters, perm))

        if sum(get_value(word, sol) for word in left) == get_value(right, sol):
            print(' + '.join(str(get_value(word, sol)) for word in left) + " = {} (mapping:
{})".format(get_value(right, sol), sol))

a=input("Enter the Problem: ")
print(a)
solve2(a)
```

Screenshots of the Outputs:

```
Enter the Problem: CROSS + ROADS = DANGER
CROSS + ROADS = DANGER
96233 + 62513 = 158746 (mapping: {'r': 6, 's': 3, 'o': 2, 'e': 4, 'd': 1, 'c': 9, 'g': 7, 'a': 5, 'n': 8})
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

(b) Graph Coloring

AIM: To implement the graph colouring problem in python.

Description of the Concept or Problem given:

Graph colouring is the procedure of assignment of colours to each vertex of a graph G such that no adjacent vertices get same colour.The objective is to minimise the number of colours while colouring agraph. The smallest number of colours required to colour a graph Gis called its chromatic number of that graph.

Manual Solution:

The steps required to colour a graph G with n number of verticesare as follows −

Arrange the vertices of the graph in some order.

Choose the first vertex and colour it with the first colour. Choose the next vertex and colour it with the lowest numbered colour that has not been coloured on any vertices adjacent to it.

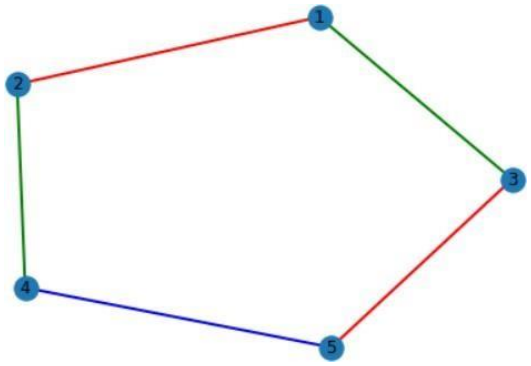
If all the adjacent vertices are coloured with this colour, assign a new colour to it. Repeat this step until all the vertices are coloured.

Program Implementation [Coding]

```
import matplotlib.pyplot as plt
import networkx as nx
from matplotlib.patches import Polygon
import numpy as np
G = nx.Graph()

colors = {0:"red", 1:"green", 2:"blue", 3:"yellow"}
G.add_nodes_from([1,2,3,4,5])
G.add_edges_from([(1,2), (1,3), (2,4), (3,5), (4,5)])
nodes = list(G.nodes)
edges = list(G.edges)
color_lists = []
color_of_edge = []
some_colors = ['red','green','blue','yellow']
for i in range(len(nodes) + 1):
    color_lists.append([])
    color_of_edge.append(-1)
```

```
def getSmallestColor(ls1,ls2):
    i = 1
    while(i in ls1 or i in ls2):
        i = i + 1
    return i

#iterate over edges
i = 0
for ed in edges:
    newColor = getSmallestColor(color_lists[ed[0]],color_lists[ed[1]])
    color_lists[ed[0]].append(newColor)
    color_lists[ed[1]].append(newColor)
    color_of_edge[i] = newColor
    i = i + 1

# Makin graph again G = nx.Graph()
for i in range(len(edges)):
    G.add_edge(edges[i][0],edges[i][1],color=some_colors[color_of_edge[i]-1])
colors = nx.get_edge_attributes(G,'color').values()
nx.draw(G, edge_color=colors, with_labels=True, width=2)
plt.show()
```

Screenshots of the Outputs:



Signature of the Student

[YUVRAJ SINGH CHAUHAN]

Date: Ex No: 4	Title of the Lab Implementation of BFS and DFS in real world applications	Name: Yuvraj Singh Chauhan Registration Number: RA1911027010058 Section: N1 Lab Batch: 1 Day Order: 3

(a) Nth-Level Friends using BFS

AIM: To find nth-Level friends for a random person in the friend network using BFS.

Description of the Concept or Problem given:

Breadth First Search, being a level order traversal, would be quite efficient over Depth First Search for many business based insights, like the following: 1. Finding all the friends of all the people in the network 2. Finding all the mutual friends for a node in the network 3. Finding the shortest path between two people in the network 4. Finding the nth level friends for a person in the network.

Manual Solution:

We can find the path between two people by running a BFS algorithm, starting the traversal from one person in level order until we reach the other person or in a much optimised way we can run a bi-directional BFS from both the nodes until our search meet at some point and hence we conclude the path , whereas DFS being a depth wise traversal may run through many unnecessary sub-trees, unknowing of the fact that the friend could be on the first level itself. Moreover, in order to find friends at nth level, using BFS this could be done in much less time, as this traversal keeps account of all the nodes in each level.

Program Implementation [Coding]:

```
from collections import deque

graph = {}

queries = []

def accept_values():

    vertex_edge = [int(i) for i in input().strip().split(" ")]

    for i in range(vertex_edge[1]):
```

```
        edge = [int(i) for i in input().strip().split(" ")]
        try:
            graph[edge[0]].append(edge[1])
        except:
            graph[edge[0]] = [edge[1]]
        try:
            graph[edge[1]].append(edge[0])
        except:
            graph[edge[1]] = [edge[0]]
    number_of_queries = int(input())
    for i in range(number_of_queries):
        queries.append([int(i) for i in input().strip().split(" ")])
    for query in queries:
        bfs(query)

def bfs(query):
    counter = 0
    q = deque()
    q.append(query[0])
    visited = {query[0] : True}
    distance = {query[0] : 0}
    while q:
        popped = q.popleft()
        for neighbour in graph[popped]:
            if neighbour not in visited:
                visited[neighbour] = True

                if distance[popped] + 1 > query[1]:
                    for key in distance:
```

```
                if distance[key] == query[1]:
                    counter +=1
            print(counter)
            return
        else:
            distance[neighbour] = distance[popped] + 1
            q.append(neighbour)
    print(counter)

accept_values()
```

Screenshots of the Outputs:

```
0 61 21 32 43 64 5 -1 -11 2
4 6
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

(b) Topological Sort using DFS

AIM: To find the Topological Sort of a graph using DFS.

Description of the Concept or Problem given:

Topological sort is an algorithm that takes a directed acyclic graph and returns the sequence of nodes where every node will appear before other nodes that it points to. Topological sorting for graphs is not applicable if the graph is not a Directed Acyclic Graph. In it, directed acyclic graph is the operation of arranging the nodes in the order in such a way that if there exists an edge (i,j), i precedes j in the lists. A topological sort basically gives a sequence in which we should perform the job and helps us to check whether the graph consists of the cycle or not. Every graph can have more than one topological sorting possible.

Manual Solution:

The algorithm of the topological sort goes like this:

1. Identify the node that has no in-degree(no incoming edges) and select that node as the source node of the graph.
2. Delete the source node with zero in-degree and also delete all its outgoing edges from the graph. Insert the deleted vertex in the result array.
3. Update the in-degree of the adjacent nodes after deleting the outgoing edges.
4. Repeat step 1 to step 3 until the graph is empty.

The resulting array at the end of the process is called the topological ordering of the directed acyclic graph. If due to some reason, there are some nodes left but they have the incoming edges, that means that the graph is not an acyclic graph and topological ordering does not exist.

Program Implementation [Coding]:

```
from collections import defaultdict

class Graph:
    def __init__(self,n):
        self.graph = defaultdict(list)
        self.N = n

    def addEdge(self,m,n):
        self.graph[m].append(n)

    def sortUtil(self,n,visited,stack):
        visited[n] = True
        for element in self.graph[n]:
            if visited[element] == False:
                self.sortUtil(element,visited,stack)
        stack.insert(0,n)

    def topologicalSort(self):
        visited = [False]*self.N
        stack =[]
        for element in range(self.N):
            if visited[element] == False:
                self.sortUtil(element,visited,stack)
        print(stack)

graph = Graph(5)
graph.addEdge(0,1);
graph.addEdge(0,3);
```

```
graph.addEdge(1,2);

graph.addEdge(2,3);

graph.addEdge(2,4);

graph.addEdge(3,4);

print("The Topological Sort Of The Graph Is:  ")
graph.topologicalSort()
```

Screenshots of the Outputs:

```
The Topological Sort Of The Graph Is:
[0, 1, 2, 3, 4]
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

Date: Ex No: 5	Title of the Lab Implementation of Informed Search Algorithms - Best First and A*	Name: Yuvraj Singh Chauhan Registration Number: RA1911027010058 Section: N1 Lab Batch: 1 Day Order: 3

(a) Best First Search

AIM: To implement the Best First Algorithm.

Description of the Concept or Problem given:

In BFS and DFS, when we are at a node, we can consider any of the adjacent as the next node. So both BFS and DFS blindly explore paths without considering any cost function. The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore.

Manual Solution:

1. Define a list, OPEN, consisting solely of a single node, the start node, s.
2. IF the list is empty, return failure.
3. Remove from the list the node n with the best score (the node where f is the minimum), and move it to a list, CLOSED.
4. Expand node n.
5. IF any successor to n is the goal node, return success and the solution (by tracing the path from the goal node to s).
6. FOR each successor node: 1.apply the evaluation function, f, to the node. 2. IF the node has not been in either list, add it to OPEN.
7. Looping structure by sending the algorithm back to the second step.

Program Implementation [Coding]:

```
from queue import PriorityQueue

v = 14

graph = [[] for i in range(v)]
```

```
def best_first_search(source, target, n):
    visited = [0] * n
    visited[0] = True
    pq = PriorityQueue()
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ")
        if u == target:
            break
        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()

def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))

addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
```

addedge(8, 10, 6)

addedge(9, 11, 1)

addedge(9, 12, 10)

addedge(9, 13, 2)

source = 0

target = 9

best_first_search(source, target, v)

Screenshots of the Outputs:

```
0 1 3 2 8 9
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

(b) A*

AIM: To implement the A* Algorithm.

Description of the Concept or Problem given:

A* is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost (least distance travelled, shortest time, etc.). It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

Manual Solution:

1. Define a list, OPEN, consisting solely of a single node, the start node, s.
2. IF the list is empty, return failure.
3. Remove from the list the node n with the best score (the node where f is the minimum), and move it to a list, CLOSED.
4. Expand node n.
5. IF any successor to n is the goal node, return success and the solution (by tracing the path from the goal node to s).
6. FOR each successor node: 1.apply the evaluation function, f, to the node. 2. IF the node has not been in either list, add it to OPEN.
7. Looping structure by sending the algorithm back to the second step.

Program Implementation [Coding]:

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}
```

```
parents = {}
g[start_node] = 0
parents[start_node] = start_node

while len(open_set) > 0:
    n = None
    for v in open_set:
        if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
            n = v

    if n == stop_node or Graph_nodes[n] == None:
        pass
    else:
        for (m, weight) in get_neighbors(n):
            if m not in open_set and m not in closed_set:
                open_set.add(m)
                parents[m] = n
                g[m] = g[n] + weight
            else:
                if g[m] > g[n] + weight:
                    g[m] = g[n] + weight
                    parents[m] = n

                    if m in closed_set:
                        closed_set.remove(m)
                        open_set.add(m)
    if n == None:
        print('Path does not exist!')
        return None
```

```
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path

        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None

def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 99,
        'D': 1,
        'E': 7,
```

```
        'G': 0,
    }
    return H_dist[n]

Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1),('G', 9)],
    'C': None,
    'E': [('D', 6)],
    'D': [('G', 1)],
}
aStarAlgo('A', 'G')
```

Screenshots of the Outputs:

```
Path found: ['A', 'E', 'D', 'G']
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

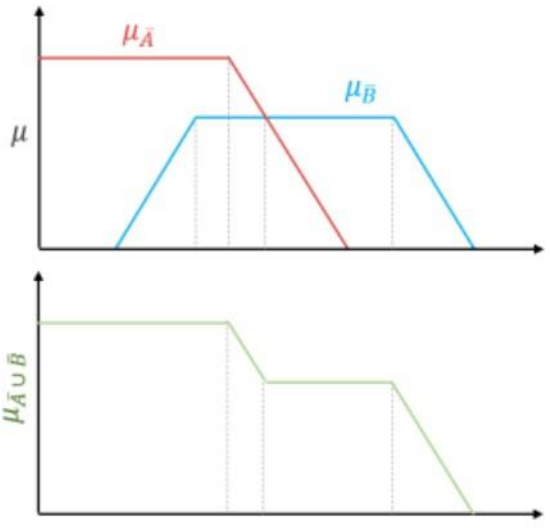
Date: Ex No: 6	Title of the Lab Implementation of Fuzzy logic and Dempster Shafer theory to handle uncertainty in Knowledge	Name: Yuvraj Singh Chauhan Registration Number: RA1911027010058 Section: N1 Lab Batch: 1 Day Order: 3

(a) Fuzzy Logic

AIM: To implement Fuzzy Logic.

Description of the Concept or Problem given:

In case of union of crisp sets, we simply have to select repeated elements only once. In case of fuzzy sets, when there are common elements in both the fuzzy sets, we should select the element with **maximum membership value**.
The **union** of two fuzzy sets A and B is a fuzzy set C , written as C = A U B
Graphically we can represent union operation as follow. Red and Blue membership functions represents the fuzzy value for elements in set A and B, respectively. Wherever these fuzzy functions overlaps, we have to consider the point with maximum membership value.



Fuzzy union

Manual Solution:

1. Import matplotlib from the python library.
2. Initialize an array of numbers in two 2D arrays a and b which would act as sets of numbers.
3. Define two functions checkset() and set_and_mf_of_set() which will check whether the elements in the array contains more than one membership value or not, if not than return the set.
4. Initialize another set of numbers in two 2D array x and y with slightly different values and store them in a different variable and check whether they contain they contain membership value or not.

5. Find the union of arrays x and y using the fuzzy logic code of union for finding the union between the arrays and print them.
6. Plot the corresponding graphs of set_and_mf_of_set(x) and set_and_mf_of_set(y).
7. The middle graph represents the union between the two sets of arrays.

Program Implementation [Coding]:

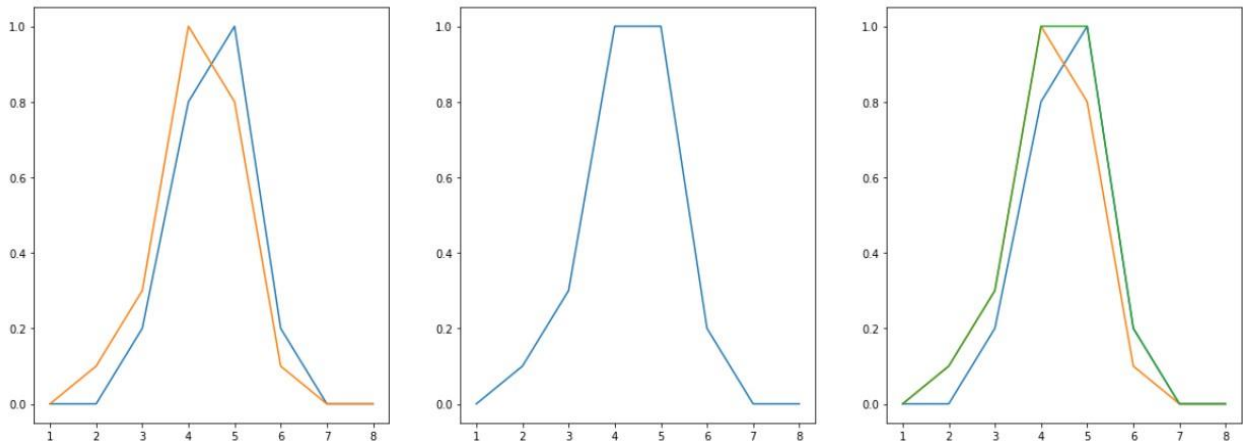
```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (20,7)
a =[[1,0.0],[2,0.0],[3,0.2],[4,0.8],[5,1],[6,0.2],[7,0.0],[8,0.0]]
b  =[[1,0.0],[2,0.1],[3,0.3],[4,1],[5,0.5],[6,0.1],[7,0.0],[8,0.0]]

print()
def checkset(a):
    r=0
    for i in range(len(a)):
        for j in range(i+1,len(a)):
            if a[i][0]==a[j][0]:
                r+=1
                break
    return r

def set_and_mf_of_set(a):
    p = checkset(a)
    if p == 0:
        set1 =[]
        mfset=[]
        for i in range(len(a)):
            set1.append(a[i][0])
            mfset.append(a[i][1])
        return set1,mfset
    else:
        print("In Set at one element more than one MemberShip Value")

a = set_and_mf_of_set(a)
x =[[1,0.0],[2,0.0],[3,0.2],[4,0.8],[5,1],[6,0.2],[7,0.0],[8,0.0]]
y =[[1,0.0],[2,0.1],[3,0.3],[4,1],[5,0.8],[6,0.1],[7,0.0],[8,0]]
p,mfp = set_and_mf_of_set(x)
q,mfq = set_and_mf_of_set(y)

def Union(a,b):
    p = checkset(a)
    q = checkset(b)
    if p == 0 & q==0:
        union=[]
        if len(a) < len(b) :
            temp = a
            a = b
            b = temp
        for i in range (len(a)):
            for j in range(0,len(b)):
                if a[i][0] == b[j][0]:
                    if a[i][1] > b[j][1]:
                        union.append(a[i])
```

```
                else:
                    union.append(b[j])
            else:
                if len(union)==0:
                    union.append(a[i])
                else:
                    p=0
                    for k in range(0,len(union)):
                        if union[k][0]==a[i][0]:
                            p+=1
                            if p==0:
                                union.append(a[i])



        return union
    else:
        print("In Set at one element more than one MemberShip Value")


z=Union(y,x)
r,mfr = set_and_mf_of_set(z)

print(x)
print(y)
print("Union is",z)

plt.subplot(131)
plt.plot(p,mfp)
plt.plot(q,mfq)

plt.subplot(132)
plt.plot(r,mfr)

plt.subplot(133)

plt.plot(p,mfp)
plt.plot(q,mfq)
plt.plot(r,mfr)
```

Screenshots of the Outputs:

```
[[1, 0.0], [2, 0.0], [3, 0.2], [4, 0.8], [5, 1], [6, 0.2], [7, 0.0], [8, 0.0]]
[[1, 0.0], [2, 0.1], [3, 0.3], [4, 1], [5, 0.8], [6, 0.1], [7, 0.0], [8, 0]]
Union is [[1, 0.0], [2, 0.1], [3, 0.3], [4, 1], [5, 1], [6, 0.2], [7, 0.0], [8, 0.0]]
```

Out[1]: [<matplotlib.lines.Line2D at 0x187c7729910>]



Signature of the Student

[YUVRAJ SINGH CHAUHAN]

(b) Dempster Shafer Theory

AIM: To implement Dempster Shafer Theory.

Description of the Concept or Problem given:

Dempster-Shafer Theory is a mathematical theory of evidence, offers an alternative to traditional probabilistic theory for the mathematical representation of uncertainty. The significant innovation of this framework is that it allows for the allocation of a probability mass to sets or intervals as opposed to mutually exclusive singletons. D-S is a potentially valuable tool for the evaluation of risk and reliability in engineering applications when it is not possible to obtain a precise measurement from experiments, or when knowledge is obtained from expert elicitation. An important aspect of D-S theory is the combination of evidence obtained from multiple sources and the modelling of conflict between them.

Manual Solution:

1. **Mass functions** denoted by m: A mass function is in many respects the most fundamental belief representation and all other representations can be easily obtained from a mass function. Formally, a mass function is a mapping assigning a mass value to each hypothesis belongs to of the frame of discernment is the amount of belief strictly committed to hypothesis.
2. **Belief functions** denoted by bel: The total amount of belief committed to a hypothesis, including all subsets is denoted by bel(A). The function is called a belief function. It can be directly computed from a mass function. A belief function is sometimes interpreted as de

fining a “lower bound” for an unknown probability function.

3. **Plausibility functions** denoted by pl: The plausibility is the amount of belief not strictly committed to the complement. It therefore expresses how plausible a hypothesis is, i.e., how much belief mass potentially supports. Whereas can be viewed as a lower bound for an unknown probability function. Under a lower and upper probability interpretation, the plausibility can be viewed as an “upper bound”.
4. **Commonality functions** denoted by q: The commonality states how much mass in total is committed to and all of the supersets with as its subset. The commonality therefore expresses how much mass potentially supports the entire set.
5. Test the various belief functions from the pyds MassFunction.
6. Print the MassFunction frame of discernment and the powerset.

Program Implementation [Coding]:

```
from Pyds import *
m1 = MassFunction({'a':0.4, 'b':0.2, 'ab':0.1, 'abc':0.3})
m2 = MassFunction({'b':0.5, 'c':0.2, 'ac':0.3, 'a':0.0})
print("m1:",m1)
print("m1: bpa of {'a','b'}=", m1['ab'])
print("m1: belief of {'a','b'}=", m1.bel('ab'))
print("m1: plausibility of {'a','b'}=", m1.pl('ab'))
print("m1: commonality of {'a','b'}=", m1.q('ab'))
print("m2:",m2)
print("m2: bpa of {'b'}=", m2['b'])
print("m2: belief of {'b'}=", m2.bel('b'))
print("m2: plausibility of {'b'}=", m2.pl('b'))
print("m2: commonality of {'b'}=", m2.q('b'))
m1_1 = MassFunction([(('a',), 0.4), (('b',), 0.2), (('a', 'b'), 0.1), (('a', 'b', 'c'), 0.3)])
if (m1_1==m1):
   print("m1_1 Equal to m1")
m1_2 = MassFunction([('a', 0.4), ('b', 0.2), ('ab', 0.1), ('abc', 0.3)])
if (m1_2==m1):
   print("m1_2 Equal to m1")
print("frame of discernment", m1.frame())
print("powerset of frame", list(m1.all()))
```

Screenshots of the Outputs:

```
m1: {{'a'}:0.4; {'a', 'b', 'c'}:0.3; {'b'}:0.2; {'a', 'b'}:0.1}
m1: bpa of {'a','b'}= 0.1
m1: belief of {'a','b'}= 0.7000000000000001
m1: plausibility of {'a','b'}= 1.0
m1: commonality of {'a','b'}= 0.4
m2: {{'b'}:0.5; {'a', 'c'}:0.3; {'c'}:0.2; {'a'}:0.0}
m2: bpa of {'b'}= 0.5
m2: belief of {'b'}= 0.5
m2: plausibility of {'b'}= 0.5
m2: commonality of {'b'}= 0.5
m1_1 Equal to m1
m1_2 Equal to m1
frame of discernment frozenset({'a', 'c', 'b'})
powerset of frame [frozenset(), frozenset({'a'}), frozenset({'c'}), frozenset({'b'}), frozenset({'a', 'c'}), frozenset({'a',
'b'}), frozenset({'b', 'c'}), frozenset({'a', 'b', 'c'})]
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

Date: Ex No: 7	Title of the Lab Implementation of Unification and Resolution in SWI Prolog	Name: Yuvraj Singh Chauhan Registration Number: RA1911027010058 Section: N1 Lab Batch: 1 Day Order: 3

(a) Unification

AIM: To implement Unification in SWI Prolog.

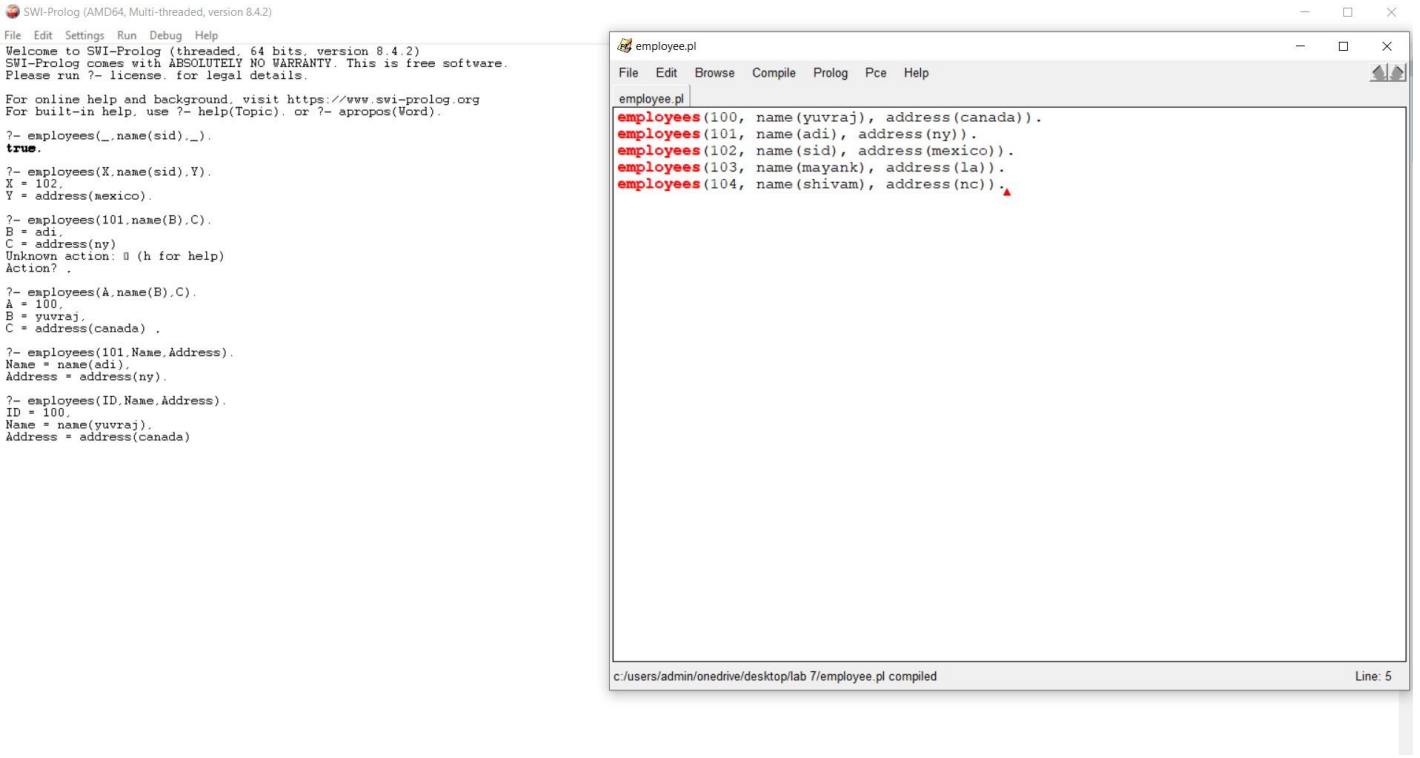
Description of the Concept or Problem given:

Prolog uses the unification technique, and it is a very general form of matching technique. In unification, one or more variables being given value to make the two call terms identical. This process is called binding the variables to values. For example, Prolog can unify the terms cat(A), and cat(mary) by binding variable A to atom mary that means we are giving the value mary to variable A.

Manual Solution:

1. If Y1 or Y2 is a variable or constant, then:
a) If Y1 , or Y2 are identical, then return NIL.
b) Else if Y1 is a variable,
    a. then if Y1, occurs in Y2, then return FAILURE
    b. Else return {({Y2,/Y1)).
c) Else if Y2 is a variable,
    a. If Y2 occurs in Y1, then return FAILURE,
    b. Else return {(Y1/Y2)}.
d) Else return FAILURE.
2. If the initial Predicate symbol in Y1, and Y2 are not same, then return FAILURE.
3. If Y1 and Y2 have a different number of arguments, then return FAILURE.
4. Set Substitution set(SUBST) to NIL.
5. For i=1 to the number of elements in Y1.
a) Call Unify function with the ith element of Y1, and ith element of Y2, and put the result into S.
b) If S=failure then returns Failure
c) If S =/= NIL then do,
    a. Apply S to the remainder of both L1 and L2.
    b. SUBST = APPEND(S, SUBST).
6. Return SUBST.

Screenshots of the Outputs:



Signature of the Student

[YUVRAJ SINGH CHAUHAN]

(b) Resolution

AIM: To implement Resolution in SWI Prolog.

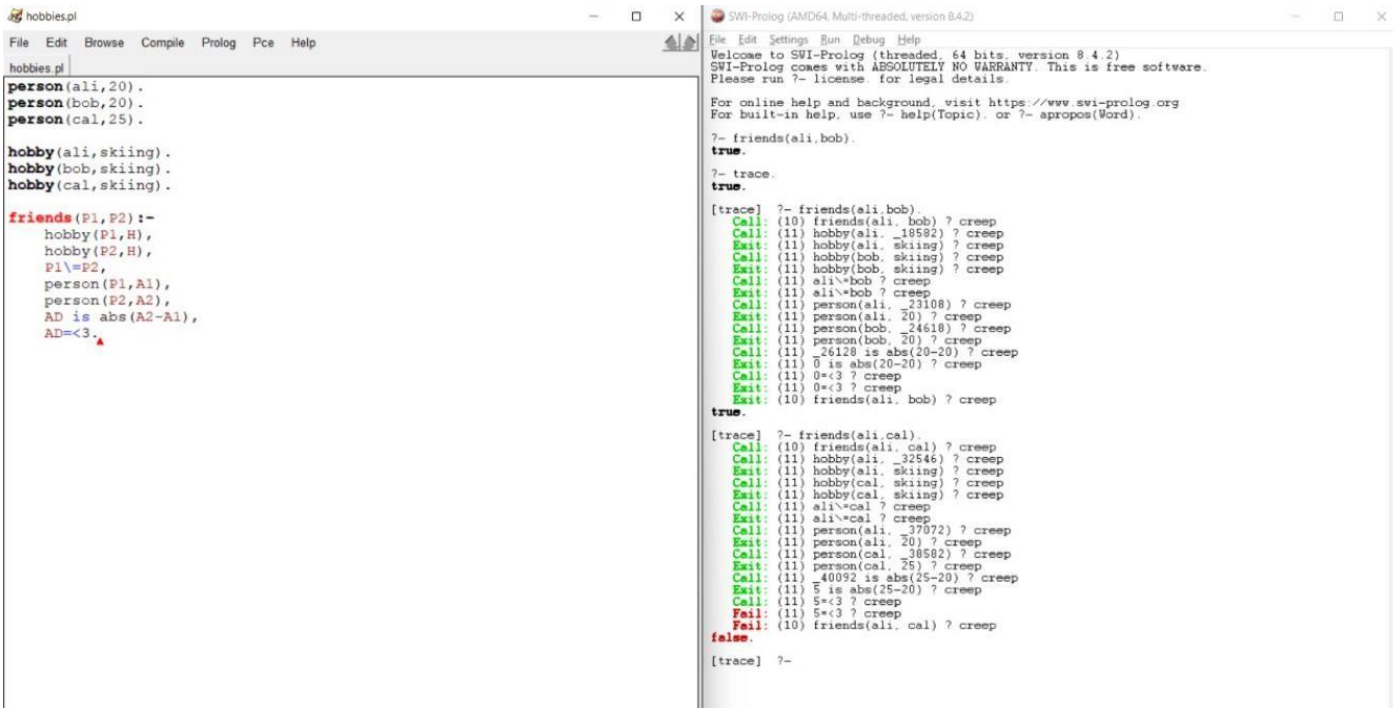
Description of the Concept or Problem given:

In simple words resolution is inference mechanism. Let's say we have clauses m :- b. and t :- p, m, z. So from that we can infer t :- p, b, z. - that is called resolution. Means, when you resolve two clauses you get one new clause. Another easy example, we have two sentences (1) All women like shopping. (2) Olivia is a woman. Now we ask query 'Who likes shopping'. So, by resolving above sentences we can have one new sentence Olivia likes shopping.

Manual Solution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

Screenshots of the Outputs:



```
person(ali,20).
person(bob,20).
person(cal,25).

hobby(ali,skiing).
hobby(bob,skiing).
hobby(cal,skiing).

friends(P1,P2):-
    hobby(P1,H),
    hobby(P2,H),
    P1\=P2,
    person(P1,A1),
    person(P2,A2),
    AD is abs(A2-A1),
    AD=<3.
```

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- friends(ali,bob).
true.

?- trace.
true.

[trace]  ?- friends(ali,bob).
   Call: (10) friends(ali, bob) ? creep
   Call: (11) hobby(ali, _18582) ? creep
   Exit: (11) hobby(ali, skiing) ? creep
   Call: (11) hobby(bob, skiing) ? creep
   Exit: (11) hobby(bob, skiing) ? creep
   Call: (11) ali\=bob ? creep
   Exit: (11) ali\=bob ? creep
   Call: (11) person(ali, _23108) ? creep
   Exit: (11) person(ali, 20) ? creep
   Call: (11) person(bob, _24618) ? creep
   Exit: (11) person(bob, 20) ? creep
   Call: (11) _26128 is abs(20-20) ? creep
   Exit: (11) 0 is abs(20-20) ? creep
   Call: (11) 0=<3 ? creep
   Exit: (11) 0=<3 ? creep
   Exit: (10) friends(ali, bob) ? creep
true.

[trace]  ?- friends(ali,cal).
   Call: (10) friends(ali, cal) ? creep
   Call: (11) hobby(ali, _32546) ? creep
   Exit: (11) hobby(ali, skiing) ? creep
   Call: (11) hobby(cal, skiing) ? creep
   Exit: (11) hobby(cal, skiing) ? creep
   Call: (11) ali\=cal ? creep
   Exit: (11) ali\=cal ? creep
   Call: (11) person(ali, _37072) ? creep
   Exit: (11) person(ali, 20) ? creep
   Call: (11) person(cal, _38582) ? creep
   Exit: (11) person(cal, 25) ? creep
   Call: (11) _40092 is abs(25-20) ? creep
   Exit: (11) 5 is abs(25-20) ? creep
   Call: (11) 5=<3 ? creep
   Fail: (11) 5=<3 ? creep
   Fail: (10) friends(ali, cal) ? creep
false.

[trace]  ?-
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

Date: **Ex No:** 8	**Title of the Lab** Implementation of a Supervised Machine Learning algorithms for Telco Churn Analysis Dataset	**Name:** Yuvraj Singh Chauhan **Registration Number:** RA1911027010058 **Section:** N1 **Lab Batch:** 1 **Day Order:** 3

AIM: To find the best fitting algorithm for Telco Churn Analysis dataset.

Description of Telco Churn Analysis:

For Telco companies it is key to attract new customers and at the same time avoid contract terminations (=churn) to grow their revenue generating base. Looking at churn, different reasons trigger customers to terminate their contracts, for example better price offers, more interesting packages, bad service experiences or change of customers' personal situations.

Churn analytics provides valuable capabilities to predict customer churn and also define the underlying reasons that drive it. The churn metric is mostly shown as the percentage of customers that cancel a product or service within a given period (mostly months). If a Telco company had 10 Mio. customers on the 1st of January and received 500K contract terminations until the 31st of January the monthly churn for January would be 5%.

Telcos apply machine learning models to predict churn on an individual customer basis and take counter measures such as discounts, special offers or other gratifications to keep their customers. A customer churn analysis is a typical classification problem within the domain of supervised learning.

Dataset:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	Str
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	No	Yes	Yes	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	No	No	No	

Importing Files for ML:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
```

```
url = 'https://raw.githubusercontent.com/yuvrajsinghchauhan/Telco-Churn-Analysis-ML-/main/Telco%20Customer%20Churn.csv'
```

KNN Classifier:

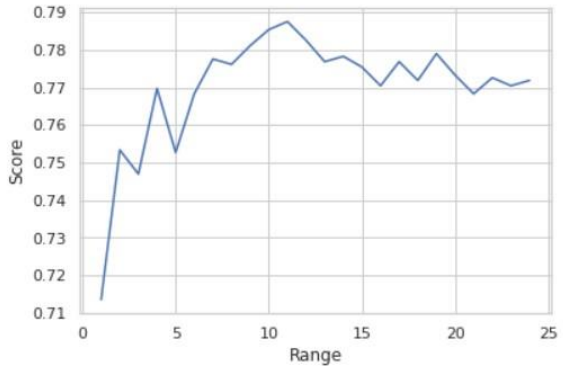
```
knn_model = KNeighborsClassifier(n_neighbors = 11) #set K neighbor as 11
knn_model.fit(x_train,y_train)
predicted_y = knn_model.predict(x_test)
accuracy_knn = knn_model.score(x_test,y_test)
print("KNN accuracy according to K=11 is :",accuracy_knn)
```

```
KNN accuracy according to K=11 is : 0.7874911158493249
```

```
# %%KNN Classification
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3) #set K neighbor as 3
knn.fit(x_train,y_train)
predicted_y = knn.predict(x_test)
print("KNN accuracy according to K=3 is :",knn.score(x_test,y_test))
```

```
KNN accuracy according to K=3 is : 0.7469793887704336
```

```
score_array = []
for each in range(1,25):
    knn_loop = KNeighborsClassifier(n_neighbors = each) #set K neighbor as 3
    knn_loop.fit(x_train,y_train)
    score_array.append(knn_loop.score(x_test,y_test))

plt.plot(range(1,25),score_array)
plt.xlabel("Range")
plt.ylabel("Score")
plt.show()
```



Support Vector Machine:

```
# %%SVM Classification
from sklearn.svm import SVC
svc_model = SVC(random_state = 1)
svc_model.fit(x_train,y_train)
accuracy_svc = svc_model.score(x_test,y_test)
print("SVM accuracy is :",accuracy_svc)
```

```
SVM accuracy is : 0.7967306325515281
```

Naïve Bayes Classification:

```
# %%Naive Bayes Classification
from sklearn.naive_bayes import GaussianNB
nb_model = GaussianNB()
nb_model.fit(x_train,y_train)
accuracy_nb = nb_model.score(x_test,y_test)
print("Naive Bayes accuracy is :",accuracy_nb)
```

```
Naive Bayes accuracy is : 0.7213930348258707
```

Decision Tree Classification:

```
# %%Decision Tree Classification
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()
dt_model.fit(x_train,y_train)
accuracy_dt = dt_model.score(x_test,y_test)
print("Decision Tree accuracy is :",accuracy_dt)
```

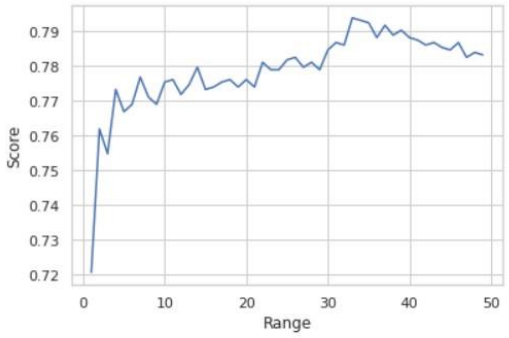
```
Decision Tree accuracy is : 0.7171286425017769
```

Random Forest Classification:

```
# %%Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
rf_model_initial = RandomForestClassifier(n_estimators = 5, random_state = 1)
rf_model_initial.fit(x_train,y_train)
print("Random Forest accuracy for 5 trees is :",rf_model_initial.score(x_test,y_test))
```

```
Random Forest accuracy for 5 trees is : 0.7668798862828714
```

```
score_array = []
for each in range(1,50):
    rf_loop = RandomForestClassifier(n_estimators = each, random_state = 1) #set K neighbor as 3
    rf_loop.fit(x_train,y_train)
    score_array.append(rf_loop.score(x_test,y_test))

plt.plot(range(1,50),score_array)
plt.xlabel("Range")
plt.ylabel("Score")
plt.show()
```




```
rf_model = RandomForestClassifier(n_estimators = 33, random_state = 1) #set tree number as 33
rf_model.fit(x_train,y_train)
accuracy_rf = rf_model.score(x_test,y_test)
print("Random Forest accuracy for 33 trees is :",accuracy_rf)
```

```
Random Forest accuracy for 33 trees is : 0.7938877043354655
```

Conclusion:

So, from the above comparison it can be observed that the model accuracy for the SVM and Random Forest Classifier are the highest (i.e., 79%), however higher accuracy doesn't mean that they will be the best always as depending on the model and data it can either overfit or take very long time to give the output and it could be more prone to outliers as well.

Therefore, taking all those things into consideration the best model for our dataset would be "Support Vector Machine".

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

Date: Ex No: 9	Title of the Lab Implementation of Natural language Processing for Amazon Customer Review Dataset	Name: Yuvraj Singh Chauhan Registration Number: RA1911027010058 Section: N1 Lab Batch: 1 Day Order: 3

AIM: To find the Positive and Negative Reviews for Amazon Customer Review dataset.

Description of Amazon Customer Review:
The Internet has revolutionized the way we buy products. In the retail e-commerce world of online marketplace, where experiencing products are not feasible. Also, in today's retail marketing world, there are so many new products are emerging every day. Therefore, customers need to rely largely on product reviews to make up their minds for better decision making on purchase. However, searching and comparing text reviews can be frustrating for users. Hence we need better numerical ratings system based on the reviews which will make customers purchase decision with ease.

During their decision making process, consumers want to find useful reviews as quickly as possible using rating system. Therefore, models able to predict the user rating from the text review are critically important. Getting an overall sense of a textual review could in turn improve consumer experience. Also, it can help businesses to increase sales, and improve the product by understanding customer's needs.

Sentiment Analysis, also known as Opinion Mining, is one of the common research areas which performs Natural Language Processing (NLP) tasks for the purpose to extract subjective information by analyzing text data written by users. In the case of sentiment analysis of review data, the main goal is to identify the user's subjectivity and classify the statements into different groups of sentiments.

The amazon review dataset for electronics products were considered. The reviews given by the user to different products as well as reviews about user's experience with the product(s) were also considered.
In this program, we aim to perform a sentiment analysis of product reviews written by online users from Amazon. This problem will be viewed as a multi-classification process and we seek to predict the sentiment scale of the user reviews based on machine learning classifiers.

Dataset:

	label	review
0	pos	Stuning even for the non-gamer: This sound tra...
1	pos	The best soundtrack ever to anything.: I'm rea...
2	pos	Amazing!: This soundtrack is my favorite music...
3	pos	Excellent Soundtrack: I truly like this soundt...
4	pos	Remember, Pull Your Jaw Off The Floor After He...
5	pos	an absolute masterpiece: I am quite sure any o...
6	neg	Buyer beware: This is a self-published book, a...
7	pos	Glorious story: I loved Whisper of the wicked ...
8	pos	A FIVE STAR BOOK: I just finished reading Whis...
9	pos	Whispers of the Wicked Saints: This was a easy...

Importing Files for NLP:

```
import os
import re
import string
import spacy
import nltk
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from spacy import displacy
from wordcloud import WordCloud
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix,classification_report,plot_confusion_matrix,accuracy_score
from sklearn.metrics import plot_precision_recall_curve,plot_roc_curve
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.linear_model import LogisticRegression
from collections import Counter
import warnings
nltk.download("vader_lexicon")
warnings.filterwarnings('ignore')
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
url = 'https://raw.githubusercontent.com/yuvrajsinghchauhan/Amazon-Customer-Review-NLP-/main/amazon_reviews.tsv'
```

Sentiment Analyzer:

```
analysis_sa = SentimentIntensityAnalyzer()
```

```
random_text = 'the most exciting book i have ever read'
print(analysis_sa.polarity_scores(random_text))
```

```
{'neg': 0.0, 'neu': 0.632, 'pos': 0.368, 'compound': 0.5413}
```

```
random_text = 'This was the worst film to ever disgrace the screen.'
print(analysis_sa.polarity_scores(random_text))
```

```
{'neg': 0.477, 'neu': 0.523, 'pos': 0.0, 'compound': -0.8074}
```

```
df['score']= df['review'].apply(lambda rew: analysis_sa.polarity_scores(rew))
df['compound']= df['score'].apply(lambda score: score['compound'])
df['predict']= df['compound'].apply(lambda value: 'pos' if value >=0 else 'neg')
```

```
df.head(10)
```

	label	review	score	compound	predict
0	pos	Stuning even for the non-gamer: This sound tra...	{'neg': 0.088, 'neu': 0.669, 'pos': 0.243, 'co...	0.9454	pos
1	pos	The best soundtrack ever to anything.: I'm rea...	{'neg': 0.018, 'neu': 0.837, 'pos': 0.145, 'co...	0.8957	pos
2	pos	Amazing!: This soundtrack is my favorite music...	{'neg': 0.04, 'neu': 0.692, 'pos': 0.268, 'com...	0.9858	pos
3	pos	Excellent Soundtrack: I truly like this soundt...	{'neg': 0.09, 'neu': 0.615, 'pos': 0.295, 'com...	0.9814	pos
4	pos	Remember, Pull Your Jaw Off The Floor After He...	{'neg': 0.0, 'neu': 0.746, 'pos': 0.254, 'comp...	0.9781	pos
5	pos	an absolute masterpiece: I am quite sure any o...	{'neg': 0.014, 'neu': 0.737, 'pos': 0.249, 'co...	0.9900	pos
6	neg	Buyer beware: This is a self-published book, a...	{'neg': 0.124, 'neu': 0.806, 'pos': 0.069, 'co...	-0.8744	neg
7	pos	Glorious story: I loved Whisper of the wicked ...	{'neg': 0.064, 'neu': 0.588, 'pos': 0.349, 'co...	0.9908	pos
8	pos	A FIVE STAR BOOK: I just finished reading Whis...	{'neg': 0.113, 'neu': 0.712, 'pos': 0.174, 'co...	0.8353	pos
9	pos	Whispers of the Wicked Saints: This was a easy...	{'neg': 0.033, 'neu': 0.777, 'pos': 0.19, 'com...	0.8196	pos

Predict Review:

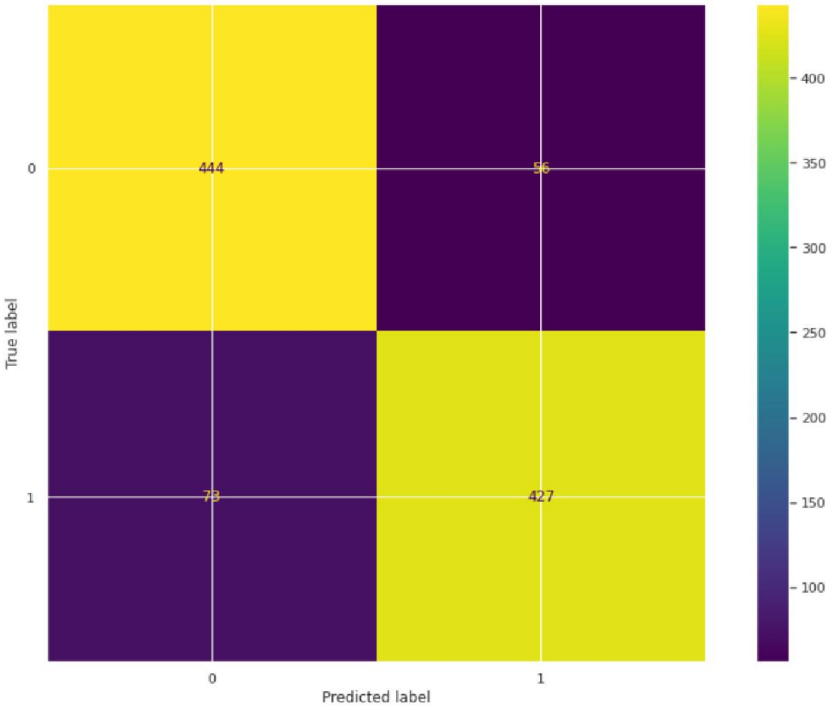
```
review_predictor = PredictReview()
```

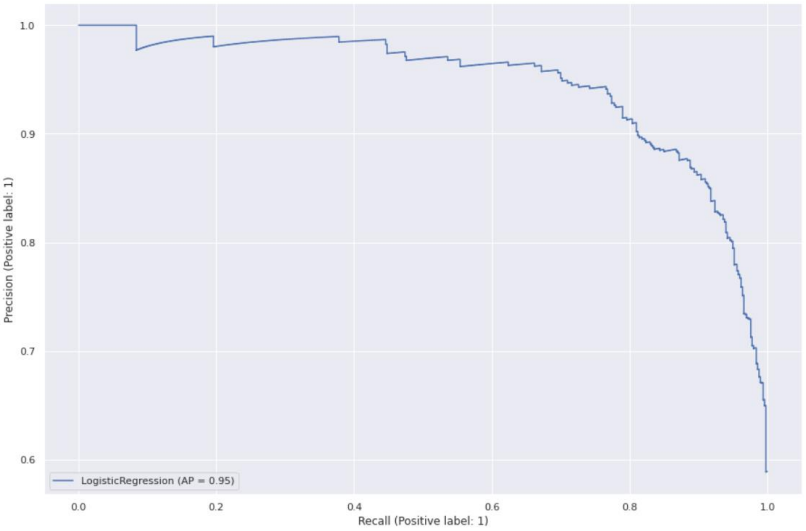
```
import nltk
nltk.download('stopwords')
```

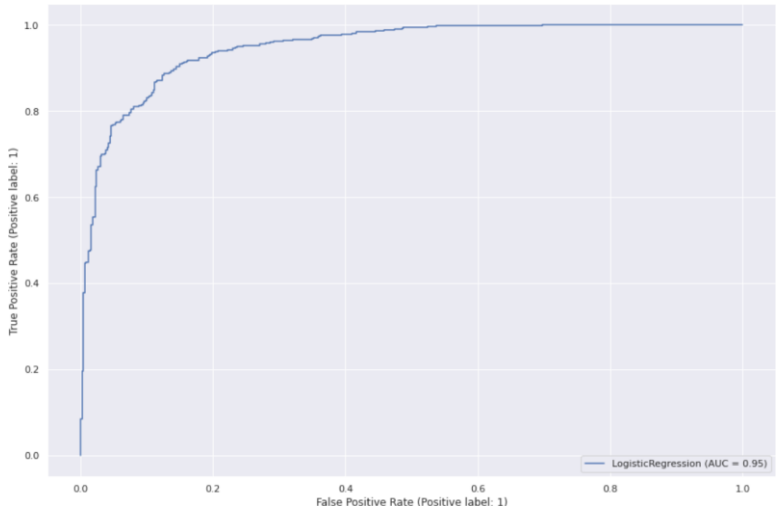
```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
plt.rcParams["figure.figsize"] = (15,10)
model,coverter = review_predictor.base(df)
plt.show()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f3a9cc73ad0>
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay object at 0x7f3a9cc52150>
<sklearn.metrics._plot.roc_curve.RocCurveDisplay object at 0x7f3a9bc1fb90>
              precision    recall  f1-score   support

           0       0.89      0.86      0.87       517
           1       0.85      0.88      0.87       483

    accuracy                           0.87      1000
   macro avg       0.87      0.87      0.87      1000
weighted avg       0.87      0.87      0.87      1000

Overall accuracy is 87% 💣

<Figure size 864x864 with 0 Axes>
```


444
56
73
427
0
1
True label
0
1
Predicted label
400
350
300
250
200
150
100


1.0
0.9
0.8
0.7
0.6
Precision (Positive label: 1)
LogisticRegression (AP = 0.95)
0.0
0.2
0.4
0.6
0.8
1.0
Recall (Positive label: 1)



Quick Predictor:

```
def quick_preictor(text):
    answer = review_predictor.test_sample(text,coverter,model)
    if answer == 'negative':
        print('\033[1m'+'\033[91m'+'Prediction is : '+answer+"\n")
        print(text)
    else:
        print('\033[1m'+'\033[92m'+'Prediction is : '+answer+"\n")
        print(text)
```

```
text = 'Purchased this keyboard in September 2018 and it has already stopped working. Some keys do not work while others simply input a gibberish com
quick_preictor(text)
```

```
Prediction is : negative

Purchased this keyboard in September 2018 and it has already stopped working. Some keys do not work while others simply input a gibberish combination
of characters. I also purchased the 4 year squaretrade warranty but since it is under the manufacturer warranty they wont allow me to make a claim.
```

```
text = 'Love it just have to sync it up with all the other RGB but love it. Big Razor fan'
quick_preictor(text)
```

```
Prediction is : positive

Love it just have to sync it up with all the other RGB but love it. Big Razor fan
```

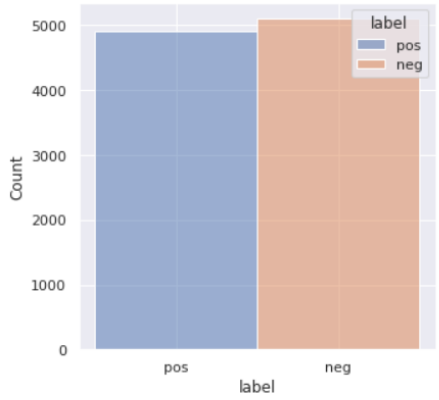
```
text = "I get cold really easy and this jacket is great if you want a warm one. I'm 5'10 and weigh 135lbs and the small fits me well. It has fleece l
quick_preictor(text)
```

```
Prediction is : positive

I get cold really easy and this jacket is great if you want a warm one. I'm 5'10 and weigh 135lbs and the small fits me well. It has fleece lined poc
kets and a fleece lined hood. It has a good quality YKK zipper. The coat snaps shut in addition to zippering shut. It has elastic around the wrists f
or a nice tight fit that won't allow breezes to flow up your arm. There a fleece-line inside pocket about chest high big enough for a cell phone or w
allet. It has a velcro enclosure.
```

```
text = "Dont do it ! It will tell u steps and heartbeat .. But dont bother connecting it to your phone 👎 Not android anyways."
quick_preictor(text)
```

```
Prediction is : negative

Dont do it ! It will tell u steps and heartbeat .. But dont bother connecting it to your phone 👎 Not android anyways.
```

```
sns.set(rc={'figure.figsize':(5,5)})
sns.histplot(df, x="label", hue="label")
plt.show()
```



Conclusion:
Hence, Sentiment Analysis can successfully performed for Amazon Customer Review using NLP with an accuracy of 87%.

Signature of the Student

[YUVRAJ SINGH CHAUHAN]

Date: **Ex No:** 10	**Title of the Lab** Training of ANN using Deep Learning for Human Attack Prediction Dataset	**Name:** Yuvraj Singh Chauhan **Registration Number:** RA1911027010058 **Section:** N1 **Lab Batch:** 1 **Day Order:** 3

AIM: To train an ANN for Human Attack Prediction dataset.

Description of Human Attack Prediction:

Nowadays, health diseases are increasing day by day due to life style, hereditary. Especially, heart disease has become more common these days. Each individual has different values for Blood pressure, cholesterol and pulse rate. In health care system there is large amount of data but poor in knowledge. The reason behind it is lack of effective analysis system to discover hidden relationship and data. Because of these data mining is used for the extracting and analyzing the knowledge. Numbers of data mining techniques are used like decision tree, Naïve Bayes, KNN, K-means, BP algorithm. But neural network is one of the most important techniques in data mining. We use the Intelligent Algorithm i.e., ANN (Artificial Neural Network). By using data mining techniques, it takes less time for the prediction of the disease with more accuracy.

Dataset:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4	1
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6	1
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7	1
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	7	1
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	8	1

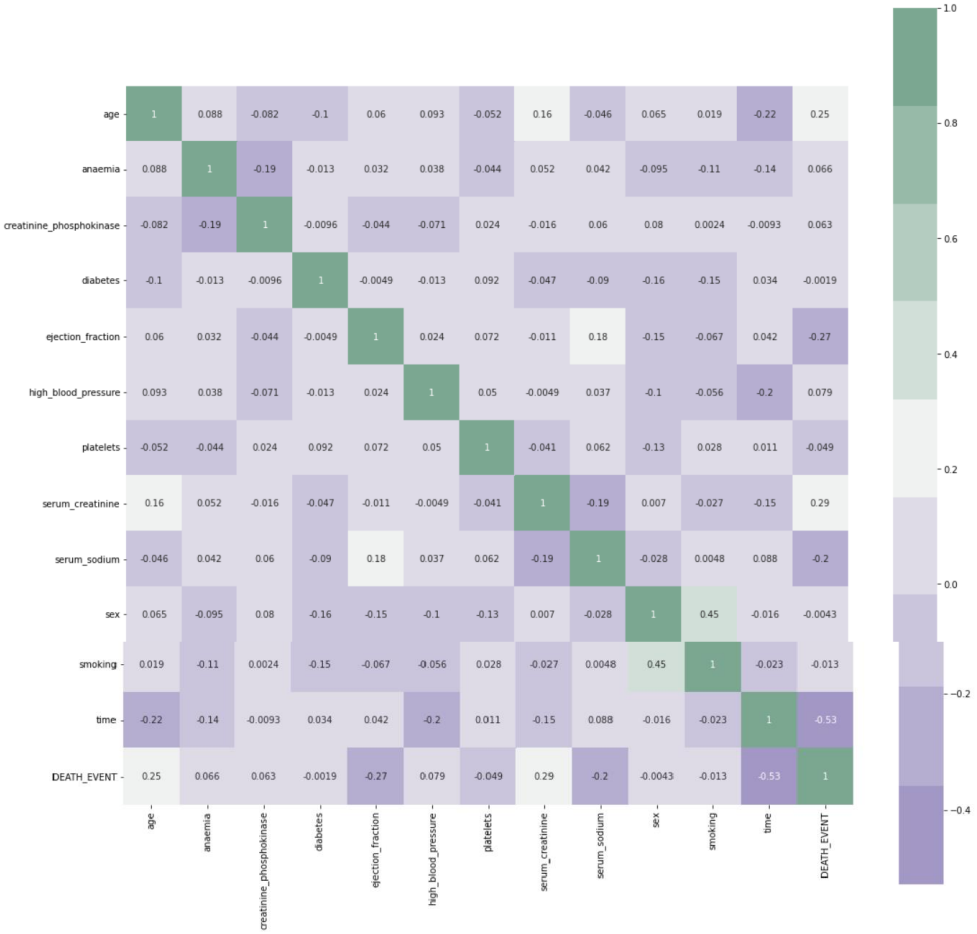
Importing Files for DL:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import seaborn as sns
from keras.layers import Dense, BatchNormalization, Dropout, LSTM
from keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from keras import callbacks
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_score
```

```
#loading data
data = pd.read_csv("https://raw.githubusercontent.com/yuvrajsinghchauhan/Human-Attack-Prediction-DL-/main/heart_failure_clinical_records_dataset.csv",e
```

Implementing Corelation Matrix:

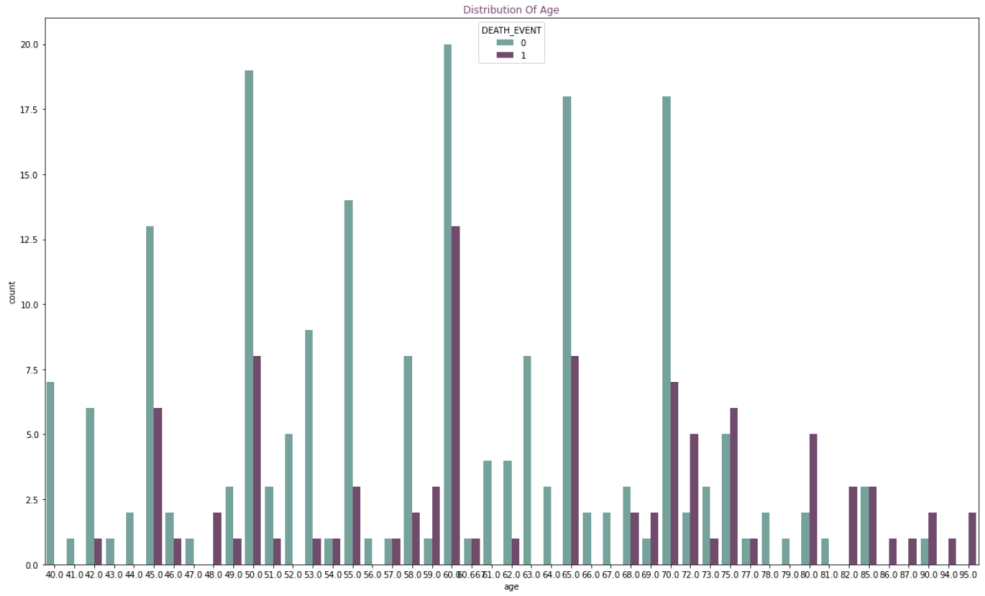
```
#Examaning a corelation matrix of all the features
cmap = sns.diverging_palette(275,150,  s=40, l=65, n=9)
corrmat = data.corr()
plt.subplots(figsize=(18,18))
sns.heatmap(corrmat,cmap= cmap,annot=True, square=True);
```



Evaluating Age Distribution:

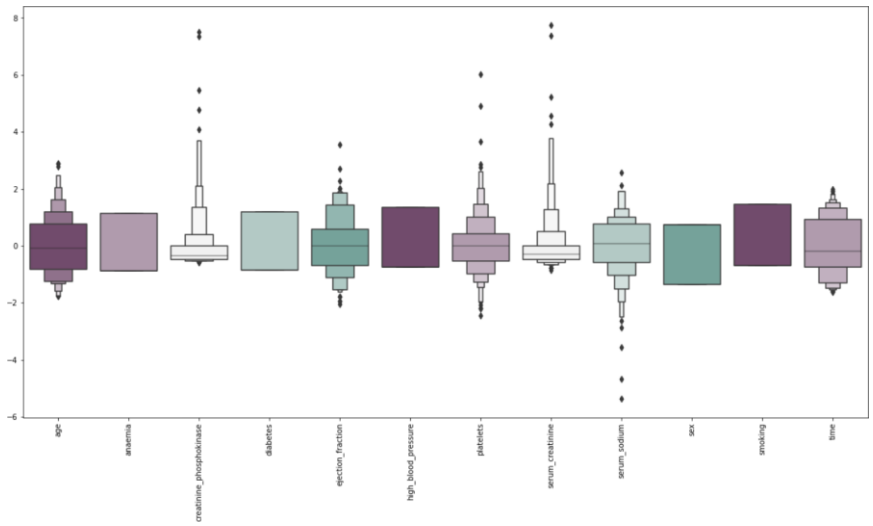
```
#Evauating age distrivution
plt.figure(figsize=(20,12))
#colours =["#774571","#b398af","#f1f1f1" ,"#afcdc7", "#6daa9f"]
Days_of_week=sns.countplot(x=data['age'],data=data, hue ="DEATH_EVENT",palette = cols)
Days_of_week.set_title("Distribution Of Age", color="#774571")
```

```
Text(0.5, 1.0, 'Distribution Of Age')
```



Scaled Features:

```
#looking at the scaled features
colours =["#774571","#b398af","#f1f1f1" ,"#afcdc7", "#6daa9f"]
plt.figure(figsize=(20,10))
sns.boxenplot(data = X_df,palette = colours)
plt.xticks(rotation=90)
plt.show()
```



Developing The ANN:

```
#spliting test and training sets
X_train, X_test, y_train,y_test = train_test_split(X_df,y,test_size=0.25,random_state=7)
```

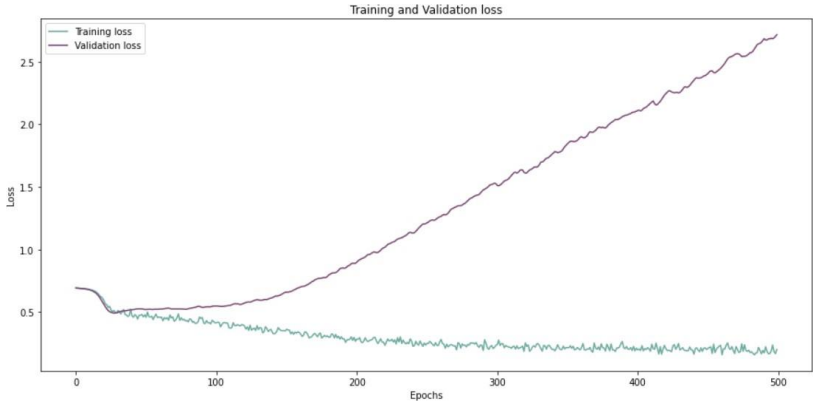
```
early_stopping = callbacks.EarlyStopping(
    min_delta=0.001, # minimium amount of change to count as an improvement
    patience=20, # how many epochs to wait before stopping
    restore_best_weights=True)

# Initialising the NN
model = Sequential()

# layers
model.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu', input_dim = 12))
model.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dropout(0.25))
model.add(Dense(units = 4, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
from tensorflow.keras.optimizers import SGD
# Compiling the ANN
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Train the ANN
history = model.fit(X_train, y_train, batch_size = 32, epochs = 500, validation_split=0.2)
```

```
Epoch 1/500
6/6 [==============================] - 1s 40ms/step - loss: 0.6928 - accuracy: 0.6480 - val_loss: 0.6922 - val_accuracy: 0.6667
Epoch 2/500
6/6 [==============================] - 0s 6ms/step - loss: 0.6919 - accuracy: 0.6480 - val_loss: 0.6911 - val_accuracy: 0.6667
Epoch 3/500
6/6 [==============================] - 0s 6ms/step - loss: 0.6909 - accuracy: 0.6480 - val_loss: 0.6901 - val_accuracy: 0.6667
Epoch 4/500
```

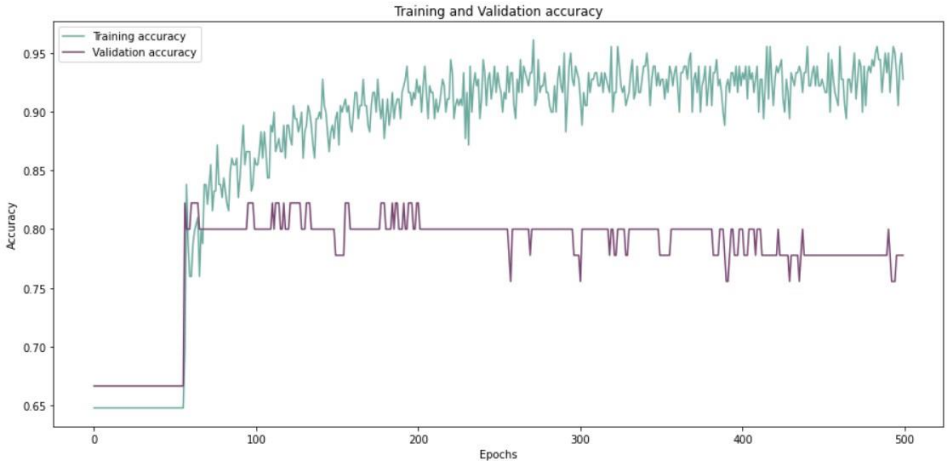
```
val_accuracy = np.mean(history.history['val_accuracy'])
print("\n%s: %.2f%%" % ('val_accuracy', val_accuracy*100))
```

```
val_accuracy: 78.08%
```

```
history_df = pd.DataFrame(history.history)
plt.figure(figsize=(15,7))
plt.plot(history_df.loc[:, ['loss']], "#6daa9f", label='Training loss')
plt.plot(history_df.loc[:, ['val_loss']],"#774571", label='Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc="best")

plt.show()
```



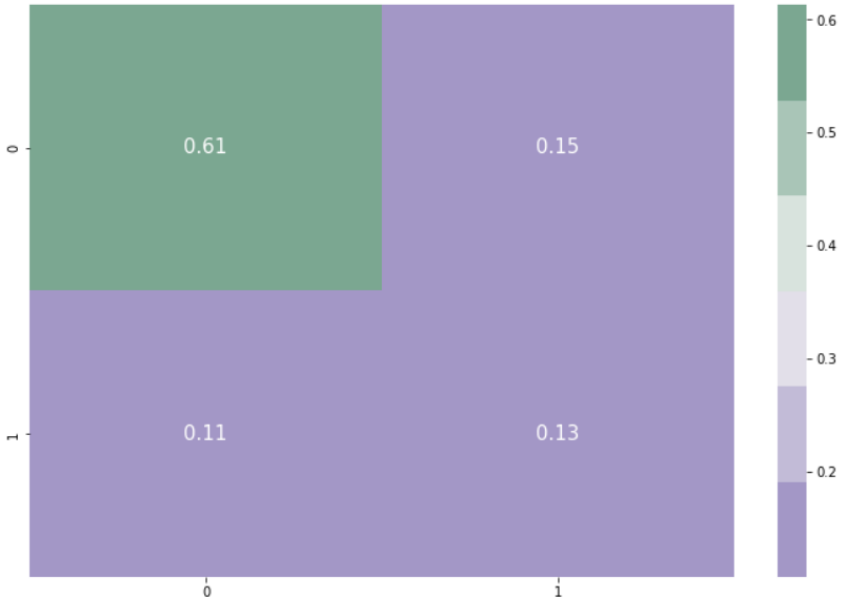
```
history_df = pd.DataFrame(history.history)
plt.figure(figsize=(15,7))
plt.plot(history_df.loc[:, ['accuracy']], "#6daa9f", label='Training accuracy')
plt.plot(history_df.loc[:, ['val_accuracy']], "#774571", label='Validation accuracy')

plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Predicting the Test Set Result:

```
# Predicting the test set results
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)
np.set_printoptions()
# confusion matrix
cmap1 = sns.diverging_palette(275,150,  s=40, l=65, n=6)
plt.subplots(figsize=(12,8))
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix/np.sum(cf_matrix), cmap = cmap1, annot = True, annot_kws = {'size':15})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0da8804390>
```



Conclusion:

From the analysis it is concluded that artificial neural network algorithm is best for classification of knowledge data from large amount of medical data. It has good performance with increase in efficiency with an accuracy of 78% when provided with normalized data. The data is normalized using a classifier. The Artificial neural network is one of the best for heart disease prediction.

Signature of the Student

[YUVRAJ SINGH CHAUHAN]