1. Write a Python code to demonstrate the Two Water Jug Puzzle.

AIM:

To develop a Python code to demonstrate the Two Water Jug Puzzle.

PROGRAM:

```
# This function is used to initialize the
# dictionary elements with a default value.
from collections import defaultdict

# jug1 and jug2 contain the value
# for max capacity in respective jugs
# and aim is the amount of water to be measured.
jug1, jug2, aim = 4, 3, 2

# Initialize dictionary with
# default value as false.
visited = defaultdict(lambda: False)

# Recursive function which prints the
# intermediate steps to reach the final
# solution and return boolean value
# (True if solution is possible, otherwise False).
# amt1 and amt2 are the amount of water present
# in both jugs at a certain point of time.
def waterJugSolver(amt1, amt2):

    # Checks for our goal and
    # returns true if achieved.
    if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):
            print(amt1, amt2)
            return True

    # Checks if we have already visited the
    # combination or not. If not, then it proceeds further.
    if visited[(amt1, amt2)] == False:
            print(amt1, amt2)
```

```
                # Changes the boolean value of
                # the combination as it is visited.
                visited[(amt1, amt2)] = True

                # Check for all the 6 possibilities and
                # see if a solution is found in any one of them.
                return (waterJugSolver(0, amt2) or
                                waterJugSolver(amt1, 0) or
                                waterJugSolver(jug1, amt2) or
                                waterJugSolver(amt1, jug2) or
                                waterJugSolver(amt1 + min(amt2, (jug1-amt1)),
                                amt2 - min(amt2, (jug1-amt1))) or
                                waterJugSolver(amt1 - min(amt1, (jug2-amt2)),
                                amt2 + min(amt1, (jug2-amt2))))

        # Return False if the combination is
        # already visited to avoid repetition otherwise
        # recursion will enter an infinite loop.
        else:
                return False

print("Steps: ")

# Call the function and pass the
# initial amount of water present in both jugs.
waterJugSolver(0, 0)
```

OUTPUT:

```
Steps:
0 0
4 0
4 3
0 3
3 0
3 3
4 2
0 2
```

RESULT:

Thus the program has been verified and completed successfully.

2. Solving Water Jug Problem using BFS

AIM:

To develop a python program to solve a water jug problem using Breadth first search.

PROGRAM:

```
from collections import deque

def BFS(a, b, target):
    # Map is used to store the states, every
    # state is hashed to binary value to
    # indicate either that state is visited
    # before or not
    m = {}
    isSolvable = False
    path = []

    # Queue to maintain states
    q = deque()

    # Initializing with initial state
    q.append((0, 0))

    while (len(q) > 0):

        # Current state
        u = q.popleft()

        # q.pop() #pop off used state

        # If this state is already visited
        if ((u[0], u[1]) in m):
            continue

        # Doesn't met jug constraints
```

```
if ((u[0] > a or u[1] > b or
        u[0] < 0 or u[1] < 0)):
        continue

# Filling the vector for constructing
# the solution path
path.append([u[0], u[1]])

# Marking current state as visited
m[(u[0], u[1])] = 1

# If we reach solution state, put ans=1
if (u[0] == target or u[1] == target):
        isSolvable = True

        if (u[0] == target):
                if (u[1] != 0):

                        # Fill final state
                        path.append([u[0], 0])
        else:
                if (u[0] != 0):

                        # Fill final state
                        path.append([0, u[1]])

        # Print the solution path
        sz = len(path)
        for i in range(sz):
                print("(", path[i][0], ",",
                        path[i][1], ")")
        break

# If we have not reached final state
# then, start developing intermediate
# states to reach solution state
q.append([u[0], b]) # Fill Jug2
```

```
                q.append([a, u[1]]) # Fill Jug1

                for ap in range(max(a, b) + 1):

                        # Pour amount ap from Jug2 to Jug1
                        c = u[0] + ap
                        d = u[1] - ap

                        # Check if this state is possible or not
                        if (c == a or (d == 0 and d >= 0)):
                                q.append([c, d])

                        # Pour amount ap from Jug 1 to Jug2
                        c = u[0] - ap
                        d = u[1] + ap

                        # Check if this state is possible or not
                        if ((c == 0 and c >= 0) or d == b):
                                q.append([c, d])

                # Empty Jug2
                q.append([a, 0])

                # Empty Jug1
                q.append([0, b])

        # No, solution exists if ans=0
        if (not isSolvable):
                print("No solution")


# Driver code
if __name__ == '__main__':

        Jug1, Jug2, target = 4, 3, 2
        print("Path from initial state "
                "to solution state ::")
```

BFS(Jug1, Jug2, target)

OUTPUT:

Path from initial state to solution state ::

(0 , 0)

(0 , 3)

(4 , 0)

(4 , 3)

(3 , 0)

(1 , 3)

(3 , 3)

(4 , 2)

(0 , 2)

RESULT:

Thus the program has been verified and completed successfully.

3. Program for solving a water jug problem using Depth first search

AIM:

To a python program to solve water jug problem using Depth first search.

PROGRAM:

```
# 3 water jugs capacity -> (x,y,z) where x>y>z
# initial state (12,0,0)
# final state (6,6,0)

capacity = (12,8,5)
# Maximum capacities of 3 jugs -> x,y,z
x = capacity[0]
y = capacity[1]
z = capacity[2]

# to mark visited states
memory = {}

# store solution path
ans = []

def get_all_states(state):
    # Let the 3 jugs be called a,b,c
    a = state[0]
    b = state[1]
    c = state[2]

    if(a==6 and b==6):
        ans.append(state)
        return True

    # if current state is already visited earlier
    if((a,b,c) in memory):
        return False
```

```
memory[(a,b,c)] = 1

#empty jug a
if(a>0):
        #empty a into b
        if(a+b<=y):
                if( get_all_states((0,a+b,c)) ):
                        ans.append(state)
                        return True
        else:
                if( get_all_states((a-(y-b), y, c)) ):
                        ans.append(state)
                        return True
        #empty a into c
        if(a+c<=z):
                if( get_all_states((0,b,a+c)) ):
                        ans.append(state)
                        return True
        else:
                if( get_all_states((a-(z-c), b, z)) ):
                        ans.append(state)
                        return True

#empty jug b
if(b>0):
        #empty b into a
        if(a+b<=x):
                if( get_all_states((a+b, 0, c)) ):
                        ans.append(state)
                        return True
        else:
                if( get_all_states((x, b-(x-a), c)) ):
                        ans.append(state)
                        return True
        #empty b into c
        if(b+c<=z):
                if( get_all_states((a, 0, b+c)) ):
                        ans.append(state)
```

```
                        return True
                else:
                        if( get_all_states((a, b-(z-c), z)) ):
                                ans.append(state)
                                return True

        #empty jug c
        if(c>0):
                #empty c into a
                if(a+c<=x):
                        if( get_all_states((a+c, b, 0)) ):
                                ans.append(state)
                                return True
                else:
                        if( get_all_states((x, b, c-(x-a))) ):
                                ans.append(state)
                                return True
                #empty c into b
                if(b+c<=y):
                        if( get_all_states((a, b+c, 0)) ):
                                ans.append(state)
                                return True
                else:
                        if( get_all_states((a, y, c-(y-b))) ):
                                ans.append(state)
                                return True

        return False

initial_state = (12,0,0)
print("Starting work...\n")
get_all_states(initial_state)
ans.reverse()
for i in ans:
        print(i)
```

OUTPUT:

Starting work...

(12, 0, 0)
(4, 8, 0)
(0, 8, 4)
(8, 0, 4)
(8, 4, 0)
(3, 4, 5)
(3, 8, 1)
(11, 0, 1)
(11, 1, 0)
(6, 1, 5)
(6, 6, 0)

RESULT:

Thus the program has been verified and completed successfully.

4. Program to find out route distance between two cities.

AIM:

To develop a program to find out route distance between two cities.

PROGRAM:

```
def solve(n, edges, s):
    graph = [set() for i in range(n)]
    for (x, y) in edges:
        x -= 1
        y -= 1
        graph[x].add(y)
        graph[y].add(x)
    temp_arr = [-1] * n
    b_set = {s - 1}
    f = set(range(n)).difference(b_set)
    index = 0
    while len(b_set) > 0:
        for a in b_set:
            temp_arr[a] = index
        nxt = {f for f in f if not b_set.issubset(graph[f])}
        f = f.difference(nxt)
        b_set = nxt
        index += 1
    return (' '.join(str(t) for t in temp_arr if t > 0))
print(solve(4, [(1, 2), (2, 3), (1, 4)], 1))
```

OUTPUT:

3 1 2

RESULT:

Thus the program has been verified and completed successfully.

5. Program for Tic Tac Toe game Single players

AIM:

To develop a python program for Tic Tac Toe game single players.

PROGRAM:

```
import random

#to ask user to choose a letter
def select_letter():
    let=""
    auto_let=""
    #ask user to select a letter (X or O)
    while(let != "x" and let != "o"):
        let=input("Select X or O: ").replace(" ","").strip().lower()
        if let == "x":
            auto_let="o"
        else:
            auto_let="x"
    return let, auto_let

#to prepare a clean board for the game
def clean_board():
    #an empty board for X and O values
    brd=[" " for x in range(10)]
    return brd

#to check if board is full
def is_board_full(board):
    return board.count(" ")==0

#to insert a letter (X or O) in a specific position
def insert_letter(board,letter,pos):
    board[pos]=letter

#to take computer moves
def computer_move(board,letter):
    computer_letter=letter
    possible_moves=[]
```

```
available_corners=[]
available_edges=[]
available_center=[]
position=-1

#all possible moves
for i in range(1,len(board)):
    if board[i] ==" ":
        possible_moves.append(i)

#if the position can make X or O wins!
#the computer will choose it to win or ruin a winning of the user
for let in ["x","o"]:
    for i in possible_moves:
        board_copy=board[:]
        board_copy[i] = let
        if is_winner(board_copy,let):
            position=i

#if computer cannot win or ruin a winning, then it will choose a random position starting
#with the corners, the center then the edges
if position == -1:
    for i in range(len(board)):
        #an empty index on the board
        if board[i]==" ":
            if i in [1,3,7,9]:
                available_corners.append(i)
            if i is 5:
                available_center.append(i)
            if i in [2,4,6,8]:
                available_edges.append(i)
    #check corners first
    if len(available_corners)>0:
        print("it comes here")
        #select a random position in the corners
        position=random.choice(available_corners)
    #then check the availability of the center
    elif len(available_center)>0:
        #select the center as the position
        position=available_center[0]
```

```
        #lastly, check the availability of the edges
        elif len(available_edges)>0:
            #select a random position in the edges
            position=random.choice(available_edges)
    #fill the position with the letter
    board[position]=computer_letter

#to draw the board
def draw_board(board):
    board[0]=-1
    #draw first row
    print("   |   |   ")
    print(" "+board[1]+" | "+board[2]+" | "+board[3]+" ")
    print("   |   |   ")
    print("-"*11)
    #draw second row
    print("   |   |   ")
    print(" "+board[4]+" | "+board[5]+" | "+board[6]+" ")
    print("   |   |   ")
    print("-"*11)
    #draw third row
    print("   |   |   ")
    print(" "+board[7]+" | "+board[8]+" | "+board[9]+" ")
    print("   |   |   ")
    print("-"*11)
    return board

#to check if a specific player is the winner
def is_winner(board,letter):
    return (board[1] == letter and board[2] == letter and board[3] == letter) or \
    (board[4] == letter and board[5] == letter and board[6] == letter) or \
    (board[7] == letter and board[8] == letter and board[9] == letter) or \
    (board[1] == letter and board[4] == letter and board[7] == letter) or \
    (board[2] == letter and board[5] == letter and board[8] == letter) or \
    (board[3] == letter and board[6] == letter and board[9] == letter) or \
    (board[1] == letter and board[5] == letter and board[9] == letter) or \
    (board[3] == letter and board[5] == letter and board[7] == letter)

#to repeat the game
def repeat_game():
```

```
    repeat=input("Play again? Press y for yes and n for no: ")
    while repeat != "n" and repeat != "y":
        repeat=input("PLEASE, press y for yes and n for no: ")
    return repeat

#to play the game
def play_game():

    letter, auto_letter= select_letter()
    #clean the board
    board=clean_board()
    board=draw_board(board)
    #check if there are empty positions on the board
    while is_board_full(board) == False:
        try:
            position=int(input("Select a position (1-9) to place an "+letter+" : " ))

        except:
            position=int(input("PLEASE enter position using only NUMBERS from 1-9: "))

        #check if user selects out of range position
        if position not in range(1,10):
            position=int(input("Please, choose another position to place an "+letter+" from 1 to 9 :"))

        #check if user selects an occupied position by X or O
        if board[position] != " ":
            position=int(input("Please, choose an empty position to place an "+letter+" from 1 to 9: "))

        #put the letter in the selected position & computer plays then draw the board
        insert_letter(board,letter,position)
        #computer move
        computer_move(board,auto_letter)
        #draw the board
        board=draw_board(board)

        if is_winner(board,letter):
            print("Congratulations! You Won.")
            return repeat_game()
        elif is_winner(board,auto_letter):
```

```
            print("Hard Luck! Computer won")
            return repeat_game()

    #if " " not in board:
    if is_board_full(board):
        print("Tie Game :)")
        return repeat_game()

#Start the game
print("Welcome to Tic Tac Toe.")
repeat="y"
while(repeat=="y"):
    repeat=play_game()
```

OUTPUT:

```
Welcome to Tic Tac Toe.
Select X or O: o
  |  |
  |  |
  |  |
-----------
  |  |
  |  |
  |  |
-----------
  |  |
  |  |
  |  |
-----------
Select a position (1-9) to place an o : 4
it comes here
  |  |
  |  |
  |  |
-----------
  |  |
 o|  |
  |  |
-----------
  |  |
 x|  |
  |  |
-----------
Select a position (1-9) to place an o : 2
it comes here
  |  |
  | o| x
  |  |
-----------
  |  |
 o|  |
```

```
  |  |
-----------
  |  |
 x|  |
  |  |
-----------
Select a position (1-9) to place an o : 5
  |  |
  | o | x
  |  |
-----------
  |  |
 o | o |
  |  |
-----------
  |  |
 x | x |
  |  |
-----------
Select a position (1-9) to place an o : 6
  |  |
  | o | x
  |  |
-----------
  |  |
 o | o | o
  |  |
-----------
  |  |
 x | x | x
  |  |
-----------
Congratulations! You Won.
Play again? Press y for yes and n for no:
```

RESULT:

Thus the program has been verified and completed successfully.

6. Program for Tic Tac Toe game played by two different human players

AIM:

To develop a python program for Tic Tac Toe game played by two different human players.

PROGRAM:

```
#board and empty positions
board=['0','1','2','3','4','5','6','7','8']
empty = [0,1,2,3,4,5,6,7,8]

#function to display board
def display_board():
  print('  |  |  ')
  print(board[0]+' | '+board[1]+' | '+board[2])
  print('  |  |  ')
  print('---------')
  print('  |  |  ')
  print(board[3]+' | '+board[4]+' | '+board[5])
  print('  |  |  ')
  print('---------')
  print('  |  |  ')
  print(board[6]+' | '+board[7]+' | '+board[8])
  print('  |  |  ')

#function to take inputs from player-I and II
def player_input(player):
  player_symbol = ['X','O']
  correct_input = True

  position = int(input('player {playerNo} chance! Choose field to fill {symbol} '.format(playerNo = player
+1, symbol = player_symbol[player])))

  if board[position] == 'X' or board[position] == 'O':
    correct_input = False

  if not correct_input:
    print("Position already equipped")
    player_input(player)
  else:
```

```
    empty.remove(position)
    board[position] = player_symbol[player]
    return 1

#function checks if any player won
def check_win():
  #define players symbols and winning positions
  player_symbol = ['X','O']
  winning_positions =[[0,1,2],[3,4,5],[6,7,8],[0,3,6],[1,4,7],[2,5,8],[0,4,8],[2,4,6]]

  #check all winning positions for matching placements
  for check in winning_positions:
    first_symbol = board[check[0]]
    if first_symbol != ' ':
      won = True
      for point in check:
        if board[point] !=  first_symbol:
          won = False
          break
      if won:
        if first_symbol == player_symbol[0]:
          print('player 1 won')
        else:
          print('player 2 won')
        break
    else:
      won = False

  if won:
    return 0
  else:
    return 1

#function to invoke functions to play
def play():
  player = 0
  while empty and check_win():
    display_board()
    player_input(player)
    player = int(not player)
```

```
    if not empty:
      print("NO WINNER!")

  #driver code
  if __name__ == '__main__':
    play()
```

OUTPUT:

```
  |   |
0 | 1 | 2
  |   |
---------
  |   |
3 | 4 | 5
  |   |
---------
  |   |
6 | 7 | 8
  |   |
player 1 chance! Choose field to fill X 1
  |   |
0 | X | 2
  |   |
---------
  |   |
3 | 4 | 5
  |   |
---------
  |   |
6 | 7 | 8
  |   |
player 2 chance! Choose field to fill O 2
  |   |
0 | X | O
  |   |
---------
  |   |
3 | 4 | 5
  |   |
---------
  |   |
6 | 7 | 8
  |   |
player 1 chance! Choose field to fill X 4
  |   |
0 | X | O
  |   |
---------
  |   |
3 | X | 5
  |   |
---------
```

```
  |  |
6 | 7 | 8
  |  |
player 2 chance! Choose field to fill O 5
  |  |
0 | X | O
  |  |
---------
  |  |
3 | X | O
  |  |
---------
  |  |
6 | 7 | 8
  |  |
player 1 chance! Choose field to fill X 7
player 1 won
```

RESULT:

Thus the program has been verified and completed successfully.

7. Program to implement Tower of Hanoi

AIM:

To develop a Python Program to implement Tower of Hanoi.

PROGRAM:

```
# Recursive Python function to solve the tower of hanoi

def TowerOfHanoi(n , source, destination, auxiliary):
        if n==1:
                print ("Move disk 1 from source",source,"to destination",destination)
                return
        TowerOfHanoi(n-1, source, auxiliary, destination)
        print ("Move disk",n,"from source",source,"to destination",destination)
        TowerOfHanoi(n-1, auxiliary, destination, source)
# Driver code

n = 4

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

# A, C, B are the name of rods
```

OUTPUT:

```
Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B
Move disk 3 from source A to destination C
Move disk 1 from source B to destination A
Move disk 2 from source B to destination C
Move disk 1 from source A to destination C
Move disk 4 from source A to destination B
Move disk 1 from source C to destination B
Move disk 2 from source C to destination A
Move disk 1 from source B to destination A
Move disk 3 from source C to destination B
Move disk 1 from source A to destination C
```

Move disk 2 from source A to destination B

Move disk 1 from source C to destination B

RESULT:

Thus the program has been verified and completed successfully.

8. Program for building a magic square of Odd number of Rows and columns

AIM:

To develop a Python Program for building a magic square of Odd number of Rows and columns

PROGRAM:

```
# Python program to generate

# odd sized magic squares

# A function to generate odd

# sized magic squares

def generateSquare(n):

    # 2-D array with all

    # slots set to 0

    magicSquare = [[0 for x in range(n)]

                           for y in range(n)]

    # initialize position of 1

    i = n // 2

    j = n - 1


    # Fill the magic square

    # by placing values

    num = 1

    while num <= (n * n):

        if i == -1 and j == n: # 3rd condition

                j = n - 2

                i = 0

        else:


                # next number goes out of

                # right side of square
```

```
            if j == n:
                j = 0
            # next number goes
            # out of upper side
            if i < 0:
                i = n - 1
    if magicSquare[int(i)][int(j)]: # 2nd condition
            j = j - 2
            i = i + 1
            continue
    else:
            magicSquare[int(i)][int(j)] = num
            num = num + 1
    j = j + 1
    i = i - 1 # 1st condition
# Printing magic square
print("Magic Square for n =", n)
print("Sum of each row or column",
    n * (n * n + 1) // 2, "\n")

for i in range(0, n):
    for j in range(0, n):
            print('%2d ' % (magicSquare[i][j]),
                end='')
            # To display output
            # in matrix form
            if j == n - 1:
                print()
```

```
# Driver Code

# Works only when n is odd

n = 7

generateSquare(n)

def create(): f=open("fun.txt","x") f.close()
def write1():
    s="This is python program" f=open("fun.txt","w") f.write(s)
    f.close() def read1():
    f=open("fun.txt","r")
    x=f.read()
    print(x)
create()
write1()
read1()
```

OUTPUT

```
Magic Square for n = 7
Sum of each row or column 175

20 12  4 45 37 29 28
11  3 44 36 35 27 19
 2 43 42 34 26 18 10
49 41 33 25 17  9  1
40 32 24 16  8  7 48
31 23 15 14  6 47 39
22 21 13  5 46 38 30
```

RESULT:

Thus the program has been verified and completed successfully.

9. Program for building a magic square of Even number of Rows and columns

AIM:

To develop a Python Program for building a magic square of Even number of Rows and columns

PROGRAM:

```
# Python program to print magic square of double order

def DoublyEven(n):

        # 2-D matrix with all entries as 0
        arr = [[(n*y)+x+1 for x in range(n)]for y in range(n)]

        # Change value of array elements at fix location
        # as per the rule (n*n+1)-arr[i][[j]

        # Corners of order (n/4)*(n/4)
        # Top left corner
        for i in range(0,n//4):
                for j in range(0,n//4):
                        arr[i][j] = (n*n + 1) - arr[i][j];

        # Top right corner
        for i in range(0,n//4):
                for j in range(3 * (n//4),n):
                        arr[i][j] = (n*n + 1) - arr[i][j];

        # Bottom Left corner
        for i in range(3 * (n//4),n):
                for j in range(0,n//4):
                        arr[i][j] = (n*n + 1) - arr[i][j];

        # Bottom Right corner
        for i in range(3 * (n//4),n):
                for j in range(3 * (n//4),n):
                        arr[i][j] = (n*n + 1) - arr[i][j];

        # Centre of matrix,order (n/2)*(n/2)
        for i in range(n//4,3 * (n//4)):
                for j in range(n//4,3 * (n//4)):
                        arr[i][j] = (n*n + 1) - arr[i][j];

        # Printing the square
        for i in range(n):
                for j in range(n):
                        print ('%2d ' %(arr[i][j]),end=" ")
                print()

# Driver Program
n = 4
```

DoublyEven(n)

OUTPUT

16 2 3 13

5 11 10 8

9 7 6 12

4 14 15 1

RESULT:

Thus the program has been verified and completed successfully.

10. Program to implement five House logic puzzle problem

AIM:

To develop a Python Program to implement five House logic puzzle problem

PROGRAM:

```
# Houses
# 1 2 3 4 5

from constraint import *
problem = Problem()

nationality = ["English", "Spanish", "Ukrainian", "Norwegian", "Japanese"]
pet = ["dog", "snails", "fox", "horse", "zebra"]
cigarette = ["Old Gold", "Kools",
"Chesterfields", "Lucky Strike", "Parliaments"]
colour = ["red", "green", "yellow", "blue", "ivory"]
beverage = ["coffee", "milk", "orange juice", "water", "tea"]

criteria = nationality + pet + cigarette + colour + beverage
problem.addVariables(criteria,[1,2,3,4,5])

problem.addConstraint(AllDifferentConstraint(), nationality)
problem.addConstraint(AllDifferentConstraint(), pet)
problem.addConstraint(AllDifferentConstraint(), cigarette)
problem.addConstraint(AllDifferentConstraint(), colour)
problem.addConstraint(AllDifferentConstraint(), beverage)

problem.addConstraint(lambda e, r: e == r, ["English","red"])
problem.addConstraint(lambda s, d: s == d, ("Spanish","dog"))
problem.addConstraint(lambda c, g: c == g, ("coffee","green"))
problem.addConstraint(lambda u, t: u == t, ("Ukrainian","tea"))
problem.addConstraint(lambda g, i: g-i == 1, ("green","ivory"))
problem.addConstraint(lambda o, s: o == s, ("Old Gold","snails"))
problem.addConstraint(lambda k, y: k == y, ("Kools","yellow"))
problem.addConstraint(InSetConstraint([3]), ["milk"])
problem.addConstraint(InSetConstraint([1]), ["Norwegian"])
problem.addConstraint(lambda c, f: abs(c-f) == 1, ("Chesterfields","fox"))
problem.addConstraint(lambda k, h: abs(k-h) == 1, ("Kools","horse"))
problem.addConstraint(lambda l, o: l == o, ["Lucky Strike","orange juice"])
problem.addConstraint(lambda j, p: j == p, ["Japanese","Parliaments"])
problem.addConstraint(lambda k, h: abs(k-h) == 1, ("Norwegian","blue"))

solution = problem.getSolutions()[0]

for i in range(1,6):
    for x in solution:
        if solution[x] == i:
            print str(i), x
```

OUTPUT

1 yellow

1 water

1 Kools

1 fox

1 Norwegian

2 tea

2 blue

2 horse

2 Ukrainian

2 Chesterfields

3 Old Gold

3 English

3 milk

3 snails

3 red

4 ivory

4 dog

4 Lucky Strike

4 orange juice

4 Spanish

5 Parliaments

5 coffee

5 zebra

5 Japanese

5 green

RESULT:

Thus the program has been verified and completed successfully.

11. Program for solving A* shortest path algorithm

AIM:

To develop a Python Program for solving A* shortest path algorithm

PROGRAM:

```
# Python program for Dijkstra's single
# source shortest path algorithm. The program is
# for adjacency matrix representation of the graph
class Graph():

	def __init__(self, vertices):
		self.V = vertices
		self.graph = [[0 for column in range(vertices)]
					for row in range(vertices)]

	def printSolution(self, dist):
		print("Vertex \t Distance from Source")
		for node in range(self.V):
			print(node, "\t\t", dist[node])

	# A utility function to find the vertex with
	# minimum distance value, from the set of vertices
	# not yet included in shortest path tree
	def minDistance(self, dist, sptSet):

		# Initialize minimum distance for next node
		min = 1e7

		# Search not nearest vertex not in the
		# shortest path tree
		for v in range(self.V):
			if dist[v] < min and sptSet[v] == False:
				min = dist[v]
				min_index = v

		return min_index

	# Function that implements Dijkstra's single source
	# shortest path algorithm for a graph represented
	# using adjacency matrix representation
	def dijkstra(self, src):

		dist = [1e7] * self.V
		dist[src] = 0
		sptSet = [False] * self.V

		for cout in range(self.V):

			# Pick the minimum distance vertex from
```

```
                # the set of vertices not yet processed.
                # u is always equal to src in first iteration
                u = self.minDistance(dist, sptSet)

                # Put the minimum distance vertex in the
                # shortest path tree
                sptSet[u] = True

                # Update dist value of the adjacent vertices
                # of the picked vertex only if the current
                # distance is greater than new distance and
                # the vertex in not in the shortest path tree
                for v in range(self.V):
                        if (self.graph[u][v] > 0 and
                        sptSet[v] == False and
                        dist[v] > dist[u] + self.graph[u][v]):
                                dist[v] = dist[u] + self.graph[u][v]

        self.printSolution(dist)

# Driver program
g = Graph(9)
g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
                [4, 0, 8, 0, 0, 0, 0, 11, 0],
                [0, 8, 0, 7, 0, 4, 0, 0, 2],
                [0, 0, 7, 0, 9, 14, 0, 0, 0],
                [0, 0, 0, 9, 0, 10, 0, 0, 0],
                [0, 0, 4, 14, 10, 0, 2, 0, 0],
                [0, 0, 0, 0, 0, 2, 0, 1, 6],
                [8, 11, 0, 0, 0, 0, 1, 0, 7],
                [0, 0, 2, 0, 0, 0, 6, 7, 0]
                ]

g.dijkstra(0)
```

OUTPUT

```
Vertex   Distance from Source
0                0
1                4
2                12
3                19
4                21
5                11
6                9
7                8
8                14
```

RESULT:

Thus the program has been verified and completed successfully.

12. Program which demonstrates Best First Search

AIM:

To develop a Python Program which demonstrates Best First Search

PROGRAM:

```
# Python3 implementation to build a
# graph using Dictionaries

from collections import defaultdict

# Function to build the graph
def build_graph():
        edges = [
                ["A", "B"], ["A", "E"],
                ["A", "C"], ["B", "D"],
                ["B", "E"], ["C", "F"],
                ["C", "G"], ["D", "E"]
        ]
        graph = defaultdict(list)

        # Loop to iterate over every
        # edge of the graph
        for edge in edges:
                a, b = edge[0], edge[1]

                # Creating the graph
                # as adjacency list
                graph[a].append(b)
                graph[b].append(a)
        return graph

if __name__ == "__main__":
        graph = build_graph()

        print(graph)
```

OUTPUT

```
defaultdict(<class 'list'>, {'A': ['B', 'E', 'C'], 'B': ['A', 'D', 'E'], 'E': ['A', 'B', 'D'], 'C': ['A', 'F', 'G'], 'D': ['B', 'E'], 'F': ['C'], 'G': ['C']})
```

RESULT:

Thus the program has been verified and completed successfully.

13. Program to solve 8-Queens problem

AIM:

To develop a Python Program to solve 8-Queens problem.

PROGRAM:

```
# Taking number of queens as input from user
print ("Enter the number of queens")
N = int(input())

# here we create a chessboard
# NxN matrix with all elements set to 0
board = [[0]*N for _ in range(N)]

def attack(i, j):
    #checking vertically and horizontally
    for k in range(0,N):
        if board[i][k]==1 or board[k][j]==1:
            return True
    #checking diagonally
    for k in range(0,N):
        for l in range(0,N):
            if (k+l==i+j) or (k-l==i-j):
                if board[k][l]==1:
                    return True
    return False

def N_queens(n):
    if n==0:
        return True
    for i in range(0,N):
        for j in range(0,N):
            if (not(attack(i,j))) and (board[i][j]!=1):
                board[i][j] = 1
                if N_queens(n-1)==True:
                    return True
                board[i][j] = 0

    return False

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

OUTPUT

```
Enter the number of queens
5
[1, 0, 0, 0, 0]
[0, 0, 1, 0, 0]
[0, 0, 0, 0, 1]
[0, 1, 0, 0, 0]
[0, 0, 0, 1, 0]
```

RESULT:

Thus the program has been verified and completed successfully.

14. Program which demonstrate the precedence properties of operators

AIM:

To develop a Python Program to demonstrate the precedence properties of operators.

PROGRAM:

```
print(10 / 5 * 5) # Output: 10.0
print(10 - 10 + 10 / 10 * 10) # Output: 10.0
print(((6 - 3) + 2 * 4) * 8 / 4) # Output: 22.0
print(2 ** 3 + 4 // 7 - 6 * 9) # Output: -46
print(not True or True) # Output: True

p = 1
q = 2
if(p > 0 and q > 0):
        print('p and q are positive integer numbers.')
```

#Output:
p and q are positive integer numbers.

Associativity in Python

```
print(10 * 20 / 10)
print(20 / 10 * 10)
```

#Output:
20.0
20.0

Associativity of exponent operator from right to left

```
print(2 ** 2 ** 3)
print((2 ** 2) ** 3)
```

#Output:
256
64

RESULT:

Thus the program has been verified and completed successfully.

15. Program to calculate factorial of a number

AIM:

To develop a Python Program to calculate factorial of a number.

PROGRAM:

```
# Python program to find the factorial of a number provided by the user.

# change the value for a different result
num = 7

# To take input from the user
#num = int(input("Enter a number: "))

factorial = 1

# check if the number is negative, positive or zero
if num< 0:
   print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
   print("The factorial of 0 is 1")
else:
   for i in range(1,num + 1):
       factorial = factorial*i
   print("The factorial of",num,"is",factorial)
```

OUTPUT

The factorial of 7 is 5040

RESULT:

Thus the program has been verified and completed successfully.