INDEX

Experiment No.	Category of Assignment	Code	Name of Experiment	Date of Allotment of experiment	Date of Evaluation	Max. Marks	Marks obtained	Sign. of Faculty
1.	**Mandatory Experiment**	**LR (0)**	Write a program to implement A* algorithm in python			**1**		
2.	**Mandatory Experiment**		Write a program to implement Single Player Game			**1**		
3.	**Mandatory Experiment**		Write a program to implement Tic-Tac-Toe game problem			**1**		
4.	**Mandatory Experiment**		Implement Brute force solution to the Knapsack problem in Python			**1**		
5.	**Mandatory Experiment**		Implement Graph coloring problem using python			**1**		
6.	**Mandatory Experiment**		Write a program to implement BFS for water jug problem using Python			**1**		
7.	**Mandatory Experiment**		Write a program to implement DFS using Python			**1**		
8.	**Mandatory Experiment**		Tokenization of word and Sentences with the help of NLTK package			**1**		
9.	**Mandatory Experiment**		Design an XOR truth table using Python			**1**		
10.	**Mandatory Experiment**		Study of SCIKIT fuzzy			**1**		

Experiment 1

Date:

Objective: Write a program to implement A* algorithm in python

Software Used: Python

Theory: A* Search algorithm is one of the best and popular techniques used in path-finding and graph traversals. Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A* Search Algorithms come to the rescue. What A* Search Algorithm does is that at each step it picks the node according to a value '**f**' which is a parameter equal to the sum of two other parameters – '**g**' and '**h**'. At each step it picks the node/cell having the lowest 'f', and processes that node/cell. We define '**g**' and '**h**' as simply as possible below:

g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.

h = the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don't know the actual distance until we find the path, because all sorts of things can be in the way (walls, water, etc.). There can be many ways to calculate this '**h**'.

Code:

```
# Describe your graph here
Graph_nodes = {
   'A': [('B', 4), ('C', 3)],
   'B': [('E', 12), ('F', 5)],
   'C': [('E', 10), ('D', 7)],
   'D': [('E', 2)],
   'E': [('G', 5)],
   'F': [('G', 16)],

}
```

```
def aStarAlgo(start_node, stop_node):
    print('Starting Node = ' + start_node)
    print('Goal Node = ' + stop_node)
    print('The graph is as follows:')
    print('A-B = 4, A-C = 3')
    print('B-E = 12, B-F = 5')
    print('C-E = 10, C-D = 7')
    print('D-E = 2')
    print('E-G = 5')
    print('F-G = 16')

    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 is None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v

        if n == stop_node or Graph_nodes[n] is 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:
                        # update g(m)
                        g[m] = g[n] + weight
```

```
                # change parent of m to n
                parents[m] = n

                # if m in closed set,remove and add to open
                if m in closed_set:
                    closed_set.remove(m)
                    open_set.add(m)

        if n is 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': 14,
        'B': 12,
        'C': 11,
        'D': 6,
        'E': 4,
        'F': 11,
        'G': 0,

    }

    return H_dist[n]

start_node = input("Enter starting node: ")
goal_node = input("Enter goal node: ")
#aStarAlgo('A', 'G')
aStarAlgo(start_node, goal_node)
```

Output:

```
Enter starting node:
A
Enter goal node:
G
Starting Node = A
Goal Node = G
The graph is as follows:
A-B = 4, A-C = 3
B-E = 12, B-F = 5
C-E = 10, C-D = 7
D-E = 2
E-G = 5
F-G = 16
Path found: ['A', 'C', 'D', 'E', 'G']


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Experiment 2

Date:

Objective: Write a program to implement Single Player Game

Software Used: Python

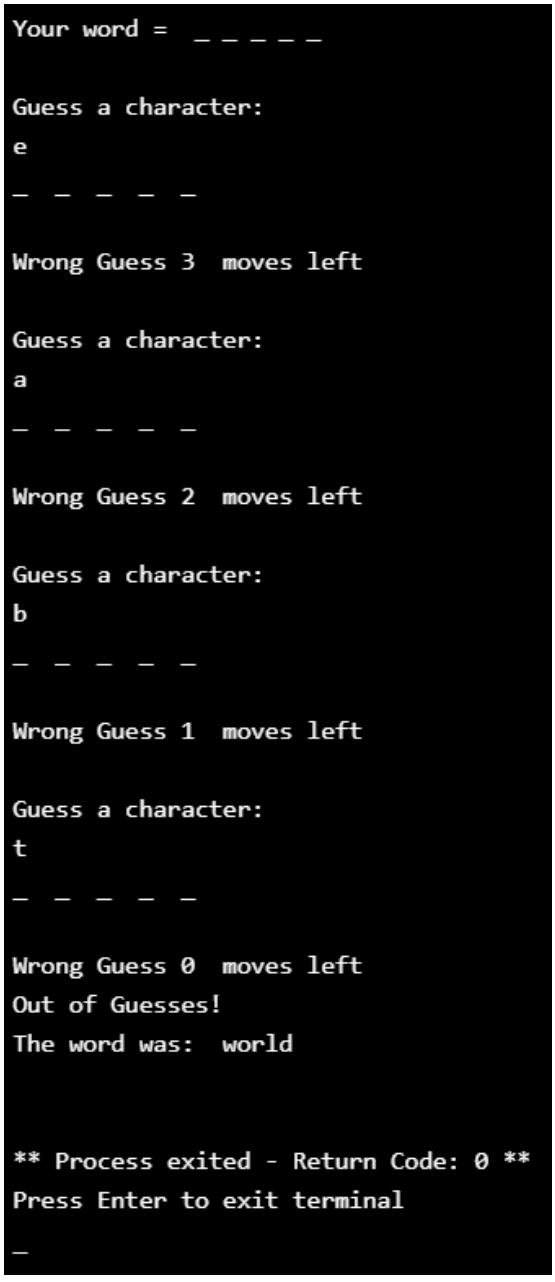
Theory: Artificial intelligence's game-playing area is crucial. Games do not need a great deal of information; all we need to know are the rules, allowed movements, and the circumstances for winning or losing the game. The majority of games need the participation of numerous participants. Single-player games, on the other hand, are distinct in terms of the challenges they present to the player. A one-player game is a struggle alone against an aspect of the environment (an artificial opponent), one's own talents, time, or chance, as opposed to a game with several players competing with or against each other to attain the game's goal.

Code:

```
import math
import random

r = ['tree','sun','ball','moon','earth','grass','world']
def guess_game():

    word = random.choice(r)
    len_of_word = len(word)
    moves = math.floor(0.8*len_of_word)
    hidden_word = []
    hidden_word = ["_ "]*len_of_word
    print("Your word = ", end=" ")
    print("_ "*len_of_word)
    guesses_made=[]
```

```
    while moves > 0:
        guess = input("\nGuess a character: ")
        contains= False
        if guess in guesses_made:
            print("You already guessed ", guess)
            continue
        else:
            guesses_made.append(guess)
        for idx in range(len_of_word):

            if word[idx] in guess:
                print(guess, end=" ")
                hidden_word[idx]=guess
                contains= True
            elif hidden_word[idx] != "_ ":
                print(hidden_word[idx], end=" ")
            else:
                print("_ ", end=" ")
        if contains == False:
            moves = moves-1
            print("\n\nWrong Guess", moves, " moves left ")
        else:
            print("\n\nCorrect Guess")
        if ("".join(hidden_word) == word):
            print("The word is: ", word)
            print(" You Won!")
            return
    print("Out of Guesses!")
    print("The word was: ", word)
    return
guess_game()
```

Output:


Your word = _ _ _ _ _
Guess a character:
e
_ _ _ _ _
Wrong Guess 3 moves left
Guess a character:
a
_ _ _ _ _
Wrong Guess 2 moves left
Guess a character:
b
_ _ _ _ _
Wrong Guess 1 moves left
Guess a character:
t
_ _ _ _ _
Wrong Guess 0 moves left
Out of Guesses!
The word was: world
** Process exited - Return Code: 0 **
Press Enter to exit terminal
_

Experiment 3

Date:

Objective: Write a program to implement Tic-Tac-Toe game problem

Software Used: Python

Theory: Tic Tac Toe is also known as Noughts and Crosses or Xs and Os, and it is a game in which players take turns marking spaces on a 3x3 grid with their own marks. If three consecutive marks (Horizontal, Vertical, and Diagonal) are made, the player who owns these movements wins. If neither player succeeds, the game finishes in a tie. If both players constantly use their best strategy, the game would always finish in a tie.

Code:

```
from copy import deepcopy

class Tic_Tac_Toe:
    def __init__(self, size):
        self.size = size

    def Display_Current_State(self, curr_state):
        print("\n")
        print('[' + curr_state[0][0] + ' ' + curr_state[0][1] + ' ' + curr_state[0][2] + "]")
        print('[' + curr_state[1][0] + ' ' + curr_state[1][1] + ' ' + curr_state[1][2] + "]")
        print("[" + curr_state[2][0] + ' ' + curr_state[2][1] + ' ' + curr_state[2][2] + "]")

    def Success(self, state):
        for i in range(self.size):
            if (state[i][0] == 'X' and state[i][1] == 'X' and state[i][2] == 'X'):
                return True
```

```
    for i in range(self.size):
        if (state[0][i] == 'X' and state[1][i] == 'X' and state[2][i] == 'X'):
            return True
    if (state[0][0] == 'X' and state[1][1] == 'X' and state[2][2] == 'X'):
        return True
    if (state[0][2] == 'X' and state[1][1] == 'X' and state[2][0] == 'X'):
        return True
    return False

def Lose(self, state):
    for i in range(self.size):
        if (state[i][0] == 'O' and state[i][1] == 'O' and state[i][2] == 'O'):
            return True
    for i in range(self.size):
        if (state[0][i] == 'O' and state[1][i] == 'O' and state[2][i] == 'O'):
            return True
    if (state[0][0] == 'O' and state[1][1] == 'O' and state[2][2] == 'O'):
        return True
    if (state[0][2] == 'O' and state[1][1] == 'O' and state[2][0] == 'O'):
        return True
    return False

def Draw(self, state):
    for i in range(self.size):
        for j in range(self.size):
            if(state[i][j] == '_'):
                return False
    return True


def Best_Move(self, state, player):
```

```
    if(self.Success(state)):
        return 1
    if(self.Lose(state)):
        return -1
    if(self.Draw(state)):
        return 0
    if(player == 'min'):
        maxv = -1
        for i in range(self.size):
            for j in range(self.size):
                if(state[i][j] == '_'):
                    temp_state = deepcopy(state)
                    temp_state[i][j] = 'X'
                    val = self.Best_Move(temp_state, 'max')
                    if(val > maxv):
                        maxv = val
        return maxv
    if(player == 'max'):
        minv = 1
        for i in range(self.size):
            for j in range(self.size):
                if (state[i][j] == '_'):
                    temp_state = deepcopy(state)
                    temp_state[i][j] = 'O'
                    val = self.Best_Move(temp_state, 'min')
                    if (val < minv):
                        minv = val
        return minv

def Computer_Move(self, state):
    minv = 1
```

```
    mini = 0
    index = 0
    for i in range(self.size):
        for j in range(self.size):
            index += 1
            if (state[i][j] == '_'):
                temp_state = deepcopy(state)
                temp_state[i][j] = 'O'
                val = self.Best_Move(temp_state, 'min')
                if (val < minv):
                    minv = val
                    mini = index
    return mini


def Start_Game(self):
    curr_state = [['_','_','_'] for i in range(self.size)]
    print("Enter position number to place your X: ")
    print("[1 2 3]")
    print("[4 5 6]")
    print("[7 8 9]")
    self.Display_Current_State(curr_state)
    for i in range(9):
        if(i%2 == 0):
            user_move = int(input("\n User's turn :"))
            while(curr_state[int((user_move-1)/self.size)][int((user_move-1)%self.size)] != '_'):
                user_move = int(input("\n Enter a valid move :"))
            x_cord = int((user_move-1)/self.size)
            y_cord = int((user_move-1)%self.size)
            curr_state[x_cord][y_cord] = 'X'
            self.Display_Current_State(curr_state)
```

```
        else:
            print("\n System's turn.....")
            system_move = self.Computer_Move(curr_state)
            x_cord = int((system_move-1)/self.size)
            y_cord = int((system_move-1)%self.size)
            curr_state[x_cord][y_cord] = 'O'
            self.Display_Current_State(curr_state)

        if (self.Success(curr_state)):
            print("\n You win the game!")
            return
        elif(self.Lose(curr_state)):
            print("\n You lose the game!")
            return
        elif(self.Draw(curr_state)):
            print("\n Match draw!")
            return

t = Tic_Tac_Toe(3)
t.Start_Game()
```

Output:

```
Enter position number to place your X:
[1 2 3]
[4 5 6]
[7 8 9]


[_ _ _]
[_ _ _]
[_ _ _]

User's turn :
5


[_ _ _]
[_ X _]
[_ _ _]

System's turn.....


[O _ _]
[_ X _]
[_ _ _]

User's turn :
3


[O _ X]
[_ X _]
[_ _ _]
```

```
System's turn.....


[O _ X]
[_ X _]
[O _ _]

User's turn :
4


[O _ X]
[X X _]
[O _ _]

System's turn.....


[O _ X]
[X X O]
[O _ _]

User's turn :
8


[O _ X]
[X X O]
[O X _]
```

```
System's turn.....


[O O X]
[X X O]
[O X _]

User's turn :
9


[O O X]
[X X O]
[O X X]

 Match draw!


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Experiment 4

Date:

Objective: Implement Brute force solution to the Knapsack problem in Python

Software Used: Python

Theory: A knapsack problem algorithm is a constructive approach to combinatorial optimization. The problem is basically about a given set of items, each with a specific weight and a value. Therefore the programmer needs to determine each item's number to include in a collection so that the total weight is less than or equal to a given limit. And also, the total value is maximum. It derives its name from the fixed-size knapsack that must be filled up to its limit with the most valuable items.

0/1 knapsack problem is a special case knapsack problem that does not fill the knapsack with fractional items.

Code:

```
import itertools
from xml.etree.ElementPath import find

def find_combination(nums):
    result: list = []
    for idx in range(len(nums)):
        for sub_set in itertools.combinations(nums, idx):
            result.append(list(sub_set))

    result.pop(0)
    result.append(nums)
    return result
```

```
wt = [2, 5, 10,5]
val = [20, 30, 50, 10]
W = 16
n = len(val)
print("Item\tWeight\tValue")

for i in range(n):
    print(i+1,"\t",wt[i],"\t",val[i])
print("Knapsack Capacity:",W)
result= find_combination([1,2,3,4])

tw=[]
tp=[]
w=0
p=0

for item in result:
    for i in item:
        w+=wt[i-1]
        p+=val[i-1]
    tw.append(w)
    tp.append(p)
    w=0
    p=0
max=tp[0]
maxindex=0

print("Item  Weight Profit \n")

for i in range(len(result)):
```

```
    if tw[i]>W:
        print(result[i],"-->",tw[i],"-->",tp[i],"--->","Not feasible")
    else:
        print(result[i],"-->",tw[i],"-->",tp[i])
        if max<tp[i]:
            max=tp[i]
            maxindex=i
print("\n\nThe solution is...")
print("Item: ",result[maxindex], "Max Profit", tp[maxindex])
```

Output:

```
Item    Weight  Value
1        2       20
2        5       30
3        10      50
4        5       10
Knapsack Capacity: 16
Item  Weight Profit

[1] --> 2 --> 20
[2] --> 5 --> 30
[3] --> 10 --> 50
[4] --> 5 --> 10
[1, 2] --> 7 --> 50
[1, 3] --> 12 --> 70
[1, 4] --> 7 --> 30
[2, 3] --> 15 --> 80
[2, 4] --> 10 --> 40
[3, 4] --> 15 --> 60
[1, 2, 3] --> 17 --> 100 ---> Not feasible
[1, 2, 4] --> 12 --> 60
[1, 3, 4] --> 17 --> 80 ---> Not feasible
[2, 3, 4] --> 20 --> 90 ---> Not feasible
[1, 2, 3, 4] --> 22 --> 110 ---> Not feasible


The solution is...
Item:  [2, 3] Max Profit 80
```

Experiment 5

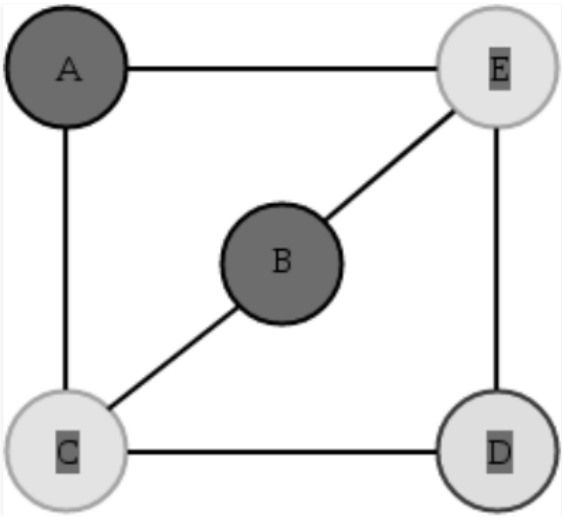
Date:

Objective: Implement Graph coloring problem using python

Software Used: Python

Theory: Graph coloring problem is to assign colors to certain elements of a graph subject to certain constraints.

Vertex coloring is the most common graph coloring problem. The problem is, given m colors, find a way of coloring the vertices of a graph such that no two adjacent vertices are colored using the same color. The other graph coloring problems like Edge Coloring (No vertex is incident to two edges of same color) and Face Coloring (Geographical Map Coloring) can be transformed into vertex coloring.



Chromatic Number: The smallest number of colors needed to color a graph G is called its chromatic number. For example, the following can be colored a minimum 2 colors.

vertex_coloring

The problem to find the chromatic number of a given graph is NP Complete.

Applications of Graph Coloring:

1. Making a Schedule or Time Table: Suppose we want to make an exam schedule for a university. We have listed different subjects and students enrolled in every subject. Many subjects would have common students (of the same batch, some backlog students, etc). How do we schedule the exam so that no two exams with a common student are scheduled at same time? How many minimum time slots are needed to schedule all exams? This problem can be represented as a graph where every vertex is a subject and an edge between two vertices means there is a common student. So this is a graph coloring problem where the minimum number of time slots is equal to the chromatic number of the graph.

Code:

```
colors = ['Red', 'Blue', 'Green', 'Yellow', 'Black','Pink','Grey','Violet']
states = ['a', 'b', 'c', 'd','e','f']
neighbors = {}
neighbors['a'] = ['b', 'c','d']
neighbors['b'] = ['a', 'c']
neighbors['c'] = ['a', 'd', 'e']
neighbors['d'] = ['c', 'e']
neighbors['e'] = ['d', 'c','f']
neighbors['f'] = ['c', 'e']
colors_of_states = {}

def promising(state, color):

    for neighbor in neighbors.get(state):
        color_of_neighbor = colors_of_states.get(neighbor)
        if color_of_neighbor == color:
            return False

    return True

def get_color_for_state(state):
    for color in colors:
        if promising(state, color):
```

```
            return color

def main():
    for state in states:
        colors_of_states[state] = get_color_for_state(state)

    print (colors_of_states)
    print("CHROMATIC NUMBER IS:",len(set(colors_of_states.values())))

main()
```

Output:

Experiment 6

Date:

Objective: Write a program to implement BFS for water jug problem using Python

Software Used: Python

Theory: You have a 4-gallon and a 3-gallon water jug. You have a pump with an unlimited amount of water. You need to get exactly 2 gallons in a 4-gallon jug.

State representation: (x, y)

–x: Contents of four gallon

–y: Contents of three gallon

•Start state: (0, 0)

•Goal state (2, n)

Operators

–Fill 3-gallon from pump, fill 4-gallon from pump

–Fill 3-gallon from 4-gallon, fill 4-gallon from 3-gallon

–Empty 3-gallon into 4-gallon, empty 4-gallon into 3-gallon

–Dump 3-gallon down drain, dump 4-gallon down drain

Code:

```
from __future__ import annotations
from collections import deque

def jug_constraints(tup: tuple[int, int], jug_one: int, jug_two: int) -> bool:
    if any([tup[0] > jug_one, tup[1] > jug_two, tup[0] < 0, tup[1] < 0]):
        return True

def water_jug(jug_one: int, jug_two: int, target_capacity: int) -> list[tuple[int, int]]:
    initial_state: tuple[int, int] = (0, 0)
    queue: deque[tuple[int, int]] = deque([initial_state])
    result: list[tuple[int, int]] = [initial_state]
    visited: dict[tuple[int, int], bool] = {
        initial_state: True
    }
```

```
    while queue:
        current_tup: tuple[int, int] = queue.popleft()
        if current_tup not in visited and not jug_constraints(current_tup, jug_one, jug_two):
            result.append(current_tup)
            visited[current_tup] = True
            if current_tup[0] == target_capacity or current_tup[1] == target_capacity:
                if current_tup[0] == target_capacity and current_tup[1] != 0:
                    result.append((current_tup[0], 0))
                elif current_tup[0] != 0:
                    result.append((0, current_tup[1]))
                return result
        queue.append((current_tup[0], jug_two))
        queue.append((jug_one, current_tup[1]))
        for pour_amt in range(max(jug_one, jug_two) + 1):
            c, d = current_tup[0] + pour_amt, current_tup[1] - pour_amt
            if (d >= 0 and d == 0) or c == jug_one:
                queue.append((c, d))
            if (d >= 0 and d == 0) or c == jug_two:
                queue.append((d, c))
        queue.append((jug_one, 0))
        queue.append((0, jug_two))

if __name__.__contains__('__main__'):
    data_map: dict[str, str] = {
        'Name' : 'Yash Walia',
        'Exp' : 'Water Jug Problem'
    }
    print(data_map)
    jug_one, jug_two, target = [int(x) for x in input().split()]

    for ans_tup in water_jug(jug_one, jug_two, target):
        print({'Water in Jug One': ans_tup[1], 'Water in Jug Two': ans_tup[0]})
```

Output:

```
{'Name': 'Yash Walia', 'Exp': 'Water Jug Problem'}
3 2 1
{'Water in Jug One': 0, 'Water in Jug Two': 0}
{'Water in Jug One': 2, 'Water in Jug Two': 0}
{'Water in Jug One': 0, 'Water in Jug Two': 3}
{'Water in Jug One': 2, 'Water in Jug Two': 3}
{'Water in Jug One': 0, 'Water in Jug Two': 2}
{'Water in Jug One': 2, 'Water in Jug Two': 2}
{'Water in Jug One': 2, 'Water in Jug Two': 1}
{'Water in Jug One': 0, 'Water in Jug Two': 1}


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Experiment 7

Date:

Objective: Write a program to implement DFS using Python

Software Used: Python

Theory: Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path.

Create a recursive function that takes the index of the node and a visited array.

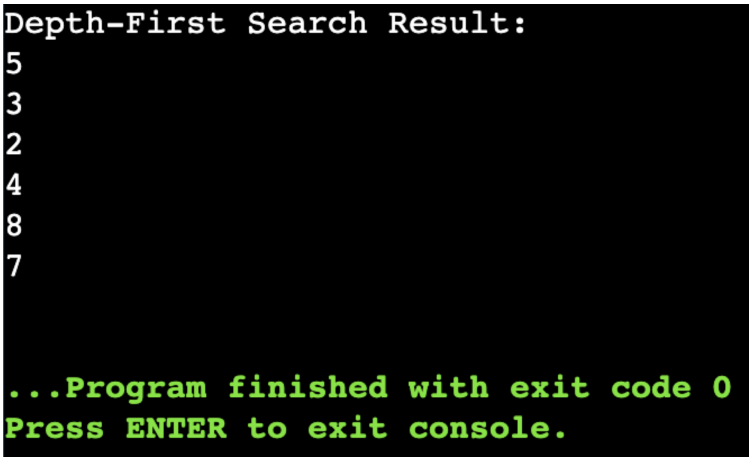
1. Mark the current node as visited and print the node.
2. Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

Code:

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = set()
```

```
def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)


print("Depth-First Search Result: ")
dfs(visited, graph, '5')
```

Output:



Experiment 8

Date:

Objective: Tokenization of word and Sentences with the help of NLTK package

Software Used: Python

Theory: NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

Code:

```
import nltk
from nltk.tokenize import word_tokenize

nltk.download('punkt')

text = "Dr. APJ Abdul Kalam is a famous name in the whole world. \
He is counted among the greatest scientists of the 21st century.\
Even more, he becomes the 11th president of India and served his country.\
He was the most valued person of the country as his contribution as a scientist and as a president is beyond compare.\
```

Apart from that, his contribution to the ISRO (Indian Space Research Organization) is remarkable.\
He headed many projects that contributed to the society also he was the one who helped in the development of Agni and Prithvi missiles.\
For his involvement in the Nuclear power in India, he was known as Missile Man of India. And due to his contribution to the country, the government awarded him with the highest civilian award."

```
print('\n\nOutput of word tokenizer: \n', word_tokenize(text))

from nltk.tokenize import sent_tokenize
print("\n\nOutput of sentence tokenizer: \n", sent_tokenize(text))

from nltk.tokenize import WordPunctTokenizer
wp = WordPunctTokenizer()
print('\n\nOutput of Punctuation Tokenizer: \n', wp.tokenize(text))
```

Output:

```
Output of word tokenizer:

['Dr.', 'APJ', 'Abdul', 'Kalam', 'is', 'a', 'famous', 'name', 'in', 'the', 'whole', 'world', '.', 'He',
'is', 'counted', 'among', 'the', 'greatest', 'scientists', 'of', 'the', '21st', 'century.Even', 'more',
',', 'he', 'becomes', 'the', '11th', 'president', 'of', 'India', 'and', 'served', 'his', 'country.He',
'was', 'the', 'most', 'valued', 'person', 'of', 'the', 'country', 'as', 'his', 'contribution', 'as', 'a',
'scie', 'ntist', 'and', 'as', 'a', 'president', 'is', 'beyond', 'compare.Apart', 'from', 'that', ',',
'his', 'contribution', 'to', 'the', 'ISRO', '(', 'Ind', 'ian', 'Space', 'Research', 'Organization', ')',
'is', 'remarkable.He', 'headed', 'many', 'projects', 'that', 'contributed', 'to', 'the', 'society', 'also',
'he', 'was', 'the', 'one', 'who', 'helped', 'in', 'the', 'development', 'of', 'Agni', 'and', 'Prithvi',
'missiles.For', 'his', 'involvement', 'in', 'the', 'Nuclear', 'power', 'in', 'India', ',', 'he', 'was',
'known', 'as', 'Missile', 'Man', 'of', 'India', '.', 'And', 'due', 'to', 'his', 'contribution', 'to',
'the', 'country', ',', 'the', 'government', 'awarded', 'him', 'with', 'the', 'highest', 'civilian',
'award', '.']
```

Output of sentence tokenizer:

```
[
 'Dr. APJ Abdul Kalam is a famous name in the whole world.',

 'He is counted among the greatest scientists of the 21st century.Even more, he becomes the 11th president
  of India and served his country. He was the most valued person of the country as his contribution as a
  scientist and as a president is beyond compare. Apart from that, his contribution to the ISRO (Indian
  Space Research Organization) is remarkable. He headed many projects that contributed to the society also
  he was the one who helped in the development of Agni and Prithvi missiles. For his involvement in the
  Nuclear power in India, he was known as Missile Man of India.',

 'And due to his contribution to the country, the government awarded him with the highest civilian award.'
]
```

Output of Punctuation Tokenizer:

```
[
  'Dr', '.', 'APJ', 'Abdul', 'Kalam', 'is', 'a', 'famous', 'name', 'in', 'the', 'whole', 'world', '.',
  'He', 'is', 'counted', 'among', 'the', 'greatest', 'scientists', 'of', 'the', '21st', 'century', '.',
  'Even', 'more', ',', 'he', 'becomes', 'the', '11th', 'president', 'of', 'India', 'and', 'served', 'his',
  'country', '.', 'He', 'was', 'the', 'most', 'valued', 'person', 'of', 'the', 'country', 'as', 'his',
  'contribution', 'as', 'a', 'scie', 'ntist', 'and', 'as', 'a', 'president', 'is', 'beyond', 'compare',
  '.', 'Apart', 'from', 'that', ',', 'his', 'contribution', 'to', 'the', 'ISRO', '(', 'Ind', 'ian',
  'Space', 'Research', 'Organization', ')', 'is', 'remarkable', '.', 'He', 'headed', 'many', 'projects',
  'that', 'contributed', 'to', 'the', 'society', 'also', 'he', 'was', 'the', 'one', 'who', 'helped', 'in',
  'the', 'development', 'of', 'Agni', 'and', 'Prithvi', 'missiles', '.', 'For', 'his', 'involvement', 'in',
  'the', 'Nuclear', 'power', 'in', 'India', ',', 'he', 'was', 'known', 'as', 'Missile', 'Man', 'of',
  'India', '.', 'And', 'due', 'to', 'his', 'contribution', 'to', 'the', 'country', ',', 'the',
  'government', 'awarded', 'him', 'with', 'the', 'highest', 'civilian', 'award', '.'
]
```

Experiment 9

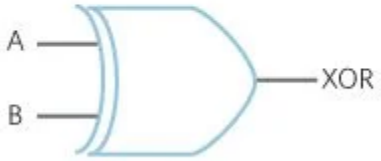
Date:

Objective: Design an XOR truth table using Python

Software Used: Python

Theory: The XOR gate stands for the Exclusive-OR gate. This gate is a special type of gate used in different types of computational circuits.
The XOR gate gives an output of 1 if either of the inputs is different, it gives 0 if they are the same.

$$X = A \oplus B$$



A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Code:

```
def decimalToBinary(n):
    return bin(n).replace("0b", "")

def xorTable(l):
    for i in l:
        for j in i:
            if i.count('1') % 2 == 0:
                output = '0'
            else:
```

```
            output = '1'
        print(j, end="  ")
    print("  ", output)


print("Enter no. of inputs: ")
n = int(input())

if n == 2:
    print("A  B C=A\'B + AB\'")
elif n == 3:
    print("A  B  C   D=ABC+A\'B\'C+AB\'C\'+A\'BC\'")

l = []
t = 2 ** n

for i in range(t):
    q = decimalToBinary(i)
    q = str(q)
    a = q.zfill(n)
    l.append(a)

if __name__ == '__main__':
    xorTable(l)
```

Output:

```
Enter no. of inputs:
3
A  B  C   D=ABC+A'B'C+AB'C'+A'BC'
0  0  0     0
0  0  1     1
0  1  0     1
0  1  1     0
1  0  0     1
1  0  1     0
1  1  0     0
1  1  1     1


...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment 10

Date:

Objective: To study documentation related to SciKit fuzzy used in Scipy

Software Used: Python

Theory:

Scikit-Fuzzy is a collection of fuzzy logic algorithms intended for use in the SciPy Stack, written in the Python computing language.

This SciKit is developed by the SciPy community.

Overview and Terminology:

Fuzzy Logic is a methodology predicated on the idea that the "truthiness" of something can be expressed over a continuum. This is to say that something isn't true or false but instead partially true or partially false.

A fuzzy variable has a crisp value which takes on some number over a predefined domain (in fuzzy logic terms, called a universe). The crisp value is how we think of the variable using normal mathematics. A fuzzy variable also has several terms that are used to describe the variable. The terms taken together are the fuzzy set which can be used to describe the "fuzzy value" of a fuzzy variable. These terms are usually adjectives like "poor," "mediocre," and "good." Each term has a membership function that defines how a crisp value maps to the term on a scale of 0 to 1. In essence, it describes "how good" something is.

A fuzzy control system links fuzzy variables using a set of rules. These rules are simply mappings that describe how one or more fuzzy variables relates to another. These are expressed in terms of an IF-THEN statement; the IF part is called the antecedent and the THEN part is the consequent. In the tipping example, one rule might be "IF the service was good THEN the tip will be good." The exact math related to how a rule is used to calculate the value of the consequent based on the value of the antecedent is outside the scope of this primer.

The Tipping Problem:

Taking the tipping example full circle, if we were to create a controller which estimates the tip we should give at a restaurant, we might structure it as such:

- Antecedents (Inputs)
    - service

        - Universe (ie, crisp value range): How good was the service of the waitress, on a scale of 1 to 10?
        - Fuzzy set (ie, fuzzy value range): poor, acceptable, amazing
    - food quality
        - Universe: How tasty was the food, on a scale of 1 to 10?
        - Fuzzy set: bad, decent, great
- Consequents (Outputs)
    - tip
        - Universe: How much should we tip, on a scale of 0% to 25%
        - Fuzzy set: low, medium, high
- Rules
    - IF the service was good or the food quality was good, THEN the tip will be high.
    - IF the service was average, THEN the tip will be medium.
    - IF the service was poor and the food quality was poor THEN the tip will be low.
- Usage
    - If I tell this controller that I rated:
        - the service as 9.8, and
        - the quality as 6.5,
    - it would recommend I leave:
        - a 20.2% tip.

Analysis:

scikit-fuzzy is a fuzzy logic Python package that works with numpy arrays. The package is imported as skfuzzy:
>>> import skfuzzy
though the recommended import statement uses an alias:
>>> import skfuzzy as fuzz
Most functions of skfuzzy are brought into the base package namespace. You can introspect the functions available in fuzz when using IPython by:
>>> import skfuzzy as fuzz
>>> fuzz
And then pressing the **Tab** key.

Within scikit-fuzzy, universe variables and fuzzy membership functions are represented by numpy arrays.

Generation of membership functions is as simple as:
>>> import numpy as np
>>> import skfuzzy as fuzz
>>> x = np.arange(11)

```
>>> mfx = fuzz.trimf(x, [0, 5, 10])
>>> x
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
>>> mfx
array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ,  0.8,  0.6,  0.4,  0.2,  0. ])
```

While most functions are available in the base namespace, the package is factored with a logical grouping of functions in submodules.

Learning Outcome:

The 'tipping problem' is commonly used to illustrate the power of fuzzy logic principles to generate complex behavior from a compact, intuitive set of expert rules.

Input variables

A number of variables play into the decision about how much to tip while dining. Consider two of them:

- quality : Quality of the food
- service : Quality of the service

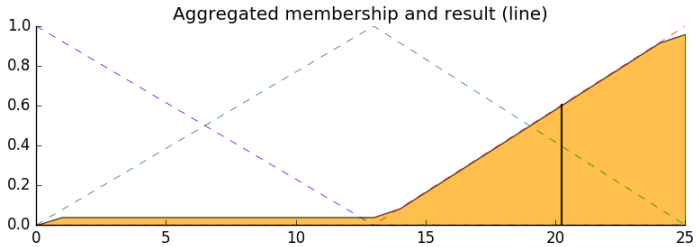
Output variable

The output variable is simply the tip amount, in percentage points:

- tip : Percent of bill to add as tip

Defuzzification

Finally, to get a real world answer, we return to crisp logic from the world of fuzzy membership functions. For the purposes of this example the centroid method will be used.
The result is a tip of 20.2%.



Final Thoughts

The power of fuzzy systems is allowing complicated, intuitive behavior based on a sparse system of rules with minimal overhead. Note our membership function universes were coarse, only defined at the integers, but fuzz.interp_membership allowed the effective resolution to increase on demand. This system can respond to arbitrarily small changes in inputs, and the processing burden is minimal.

Internal Assessment (Viva Component) Sheet for Lab Experiment
Department of Computer Science & Engineering
Amity University, Noida (UP)

Programme	B. Tech CSE	Course Name	Artificial Intelligence
Course Code	[CSE401]	Semester	6
Student Name	Yash Walia	Enrollment No.	A2305219036

Marking Criteria

Criteria	Total Marks	Marks Obtained	Comments
Clarity of the Subject (H)	4		
Quality of theoretical Discussion (I)	6		
Total	10		