PRACTICAL FILE

ARTIFICIAL INTELLIGENCE

(CSE401)

Program Name: B. Tech

Semester: 6th

Batch: 2019-23

DEPARTMENT OF INFORMATION TECHNOLOGY

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY UTTAR PRADESH

By

Gaurav Singh,

A2305319064

B. Tech 6IT-2X

Faculty:

Dr Archana Singh

INDEX

Practical No.	Name of Experiment	Date of Allotment of experiment	Date of Evaluation	Max Marks	Marks Obtained	Faculty Signature
1.	Write a program in python for Tower of Hanoi.	06/1/22	06/1/22	**1**		
2.	Write a python program to implement Depth First Search (DFS) algorithm.	13/1/22	13/1/22	**1**		
3.	Write a python program to implement Breadth First Search (BFS) algorithm.	20/01/22	20/01/22	**1**		
4.	Write a program in python to implement water jug problem.	27/01/22	27/01/22	**1**		
5.	Write a program in python for Graph Colouring Problem.	06/02/22	06/02/22	**1**		
6.	Write a program in python to implement TRUTH TABLES.	13/02/22	13/02/22	**1**		
7.	Write a program in python to implement A* algorithm.	10/03/22	10/03/22	**1**		
8.	Write a program in python to implement Tic-Tac-Toe.	17/03/22	17/03/22	**1**		
9.	Write a program in python to implement Tokenization of word and Sentences with the help of NLTK package	24/03/22	24/03/22	**1**		
10.	Write a program in python to implement Brute force solution to the Knapsack problem in Python	03/04/22	03/04/22	**1**		

Experiment 1

Objective: Write a python program to implement Tower of Hanoi.

Software Used: Visual Studio Code

Program:

```
#Tower of Hanoi by Gaurav
def TOH(n , source, destin, aux_rod):
    if n == 0:
        return
    TOH(n-1, source, aux_rod, destin)
    print("Move disk",n,"from",source,"to",destin)
    TOH(n-1, aux_rod, destin, source)
n = 4
TOH(n, 'P', 'Q', 'R')
```

Output:

```
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence> cd .\Practice
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence\Practice> python .\toh.py
Move disk 1 from P to R
Move disk 2 from P to Q
Move disk 1 from R to Q
Move disk 3 from P to R
Move disk 1 from Q to P
Move disk 2 from Q to R
Move disk 1 from P to R
Move disk 4 from P to Q
Move disk 1 from R to Q
Move disk 2 from R to P
Move disk 1 from Q to P
Move disk 3 from R to Q
Move disk 1 from P to R
Move disk 2 from P to Q
Move disk 1 from R to Q
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence\Practice>
```

Conclusion:

Hence, we have successfully implemented Tower of Hanoi using Python.

Experiment 2

Objective: Write a python program to implement Depth First Search (DFS) algorithm.

Software Used: Visual Studio Code

Program:

```
class graph:
    vertices = {}
    goalFound = False

    def __init__(self):
        pass

    def addEdge(self, u, v):
        if u in self.vertices:
            self.vertices[u].append(v)
        else:
            l = list()
            l.append(v)
            self.vertices[u] = l

    def print_graph(self):
        for i in self.vertices:
            print(f"{i} -> {self.vertices[i]}")

    def dfs_helper(self, s, g, v):
        if(self.goalFound):
            return

        if(s == g):
            print(s)
            print("Goal found!")
            self.goalFound = True

        v.add(s)
        print(s, end=" ")

        try:
            for n in self.vertices[s]:
                if(n not in v):
                    self.dfs_helper(n, g, v)
        except:
            return
```

```
    def dfs(self, start, goal):
        visited = set()
        if(start == goal):
            print("Start == Goal")
            return

        self.dfs_helper(start, goal, visited)


g = graph()

e = (int)(input("Enter the number of edges: "))

print("Enter the edges: ")
for i in range(e):
    edge = input()
    edge.split()
    g.addEdge(int(edge[0]), int(edge[2]))

# g.addEdge(0, 1)
# g.addEdge(0, 2)
# g.addEdge(1, 3)
# g.addEdge(3, 4)
# g.addEdge(4, 0)
# g.addEdge(4, 1)
# g.addEdge(4, 5)

print("\nGraph is: ")
g.print_graph()

start = (int)(input("\nEnter the start node: "))
goal = (int)(input("Enter the goal node: "))

print("\nDFS Traversal: ", end="")
g.dfs(start, goal)
```

Output:

```
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence> python .\lab2.py
Enter the number of edges: 7
Enter the edges:
0 1
0 2
1 3
3 4
4 0
4 1
4 5

Graph is:
0 -> [1, 2]
1 -> [3]
3 -> [4]
4 -> [0, 1, 5]

Enter the start node: 0
Enter the goal node: 4

DFS Traversal: 0 1 3 4
Goal found!
```

Conclusion:

Hence, we have successfully implemented DFS algorithm using Python.

Experiment 3

Objective: Write a python program to implement Breadth First Search (BFS) algorithm.

Software Used: Visual Studio Code

Program:

```
class graph:
    vertices = {}
    def __init__(self):
        pass
    def addEdge(self, u, v):
        if u in self.vertices:
            self.vertices[u].append(v)
        else:
            l = list()
            l.append(v)
            self.vertices[u] = l
    def print_graph(self):
        for i in self.vertices:
            print(f"{i} -> {self.vertices[i]}")
    def bfs(self, start, goal):
        visited = set()
        queue = []
        if(start == goal):
            print('Goal == Start')
            return
        visited.add(start)
        queue.append(start)

        while(len(queue) != 0):
            front = queue.pop(0)
            print(f"{front}, ", end="")
            if(front == goal):
                print("\nGoal Found!")
                return
            try:
                for n in self.vertices[front]:
                    if(n not in visited):
                        queue.append(n)
                        visited.add(n)
            except:
                pass
```

```
        print("\nTraversal Completed, Goal not found!")


g = graph()
e = (int)(input("Enter the number of edges: "))
print("Enter the edges: ")
for i in range(e):
    edge = input()
    edge.split()
    g.addEdge(int(edge[0]), int(edge[2]))

print("\nGraph is: ")
g.print_graph()

start = (int)(input("\nEnter the start node: "))
goal = (int)(input("Enter the goal node: "))

print("\nBFS Traversal: ", end="")
g.bfs(start, goal)
```

Output:

```
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence> python .\lab3.py
Enter the number of edges: 7
Enter the edges:
0 1
0 2
1 3
3 4
4 0
4 1
4 5

Graph is:
0 -> [1, 2]
1 -> [3]
3 -> [4]
4 -> [0, 1, 5]

Enter the start node: 1
Enter the goal node: 4

BFS Traversal: 1, 3, 4,
Goal Found!
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence>
```

Conclusion:

Hence, we have successfully implemented BFS algorithm using Python.

Experiment 4

Objective: Write a python program to implement Water Jug Problem.

Software Used: Visual Studio Code

Program:

```
print("Water-Jug Problem by Gaurav")
max1 = int(input("Enter the amount of water you want in Jug-1: "))
max2 = int(input("Enter the amount of water you want in Jug-2: "))
goal = int(input("Enter the amount of water you want in Jug: "))
print("Maximum limit of Jug-1: ",max1)
print("Maximum limit of Jug-2: ",max2)
print("Goal is: ",goal)
print("Jug-1,Jug-2")
def pour(jug1,jug2):
    print(jug1,"\t",jug2)
    if jug2 is goal:
        return
    elif jug2 is max2:
        pour(0,jug1)
    elif jug1!=0 and jug2 == 0:
        pour(0,jug1)
    elif jug1 is goal:
        pour(jug1,0)
    elif jug1<max1:
        pour(max1,jug2)
    elif jug1<(max2-jug2):
        pour(0,(jug1+jug2))
    else:
        pour(jug1-(max2-jug2),(max2-jug2)+jug2)
```

print(pour(0,0))

Output:

```
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence\Practice> python .\waterJug.py
Water-Jug Problem by Gaurav
Enter the amount of water you want in Jug-1: 5
Enter the amount of water you want in Jug-2: 7
Enter the amount of water you want in Jug: 4
Maximum limit of Jug-1:  5
Maximum limit of Jug-2:  7
Goal is:  4
Jug-1,Jug-2
0        0
5        0
0        5
5        5
3        7
0        3
5        3
1        7
0        1
5        1
0        6
5        6
4        7
0        4
None
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence\Practice>
```

Conclusion:

Hence, we have successfully implemented Water Jug using Python.

Experiment 5

Objective: Write a python program to implement Graph Colouring Problem.

Software Used: Visual Studio Code

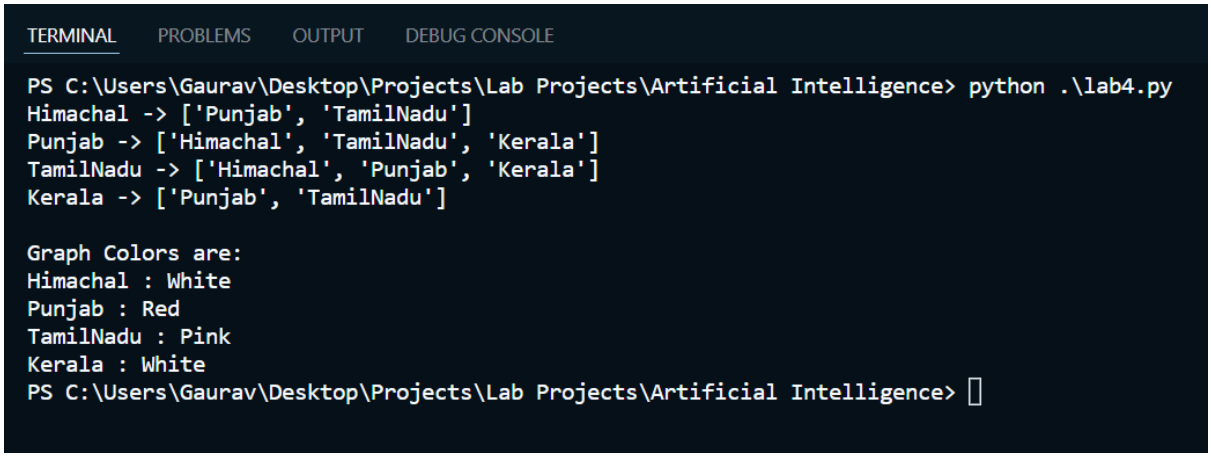
Program:

```
class graph:
    g = []
    vertices = {}
    def __init__(self):
        pass
    def addEdge(self, u, v):
        if u in self.vertices:
            self.vertices[u].append(v)
        else:
            l = list()
            l.append(v)
            self.vertices[u] = l

    def print_graph(self):
        for i in self.vertices:
            print(f"{i} -> {self.vertices[i]}")

    def make_adj(self):
        cities = self.vertices.keys()

        for i in cities:
            l = list()
            for j in cities:
                if(i == j):
```

```
                l.append(0)
            elif(j in self.vertices[i]):
                l.append(1)
            else:
                l.append(0)
        self.g.append(l)

def print_adj(self):
    for i in range(len(self.g)):
        for j in self.g[i]:
            print(f"{j} ", end="")
        print()

def isSafe(self, i, colors, c):
    for j in range(0, len(self.vertices.keys())):
        if self.g[i][j] == 1 and colors[j] == c:
            return False
    return True

def helper(self, colors, i):
    if(i == len(self.vertices.keys())):
        return True

    for c in range(5):
        if self.isSafe(i, colors, c):
            colors[i] = c
            if self.helper(colors, i+1):
                return True
            colors[i] = 0
```

```
    def graphColor(self, colors):
        if self.helper(colors, 0) == None:
            return False
        else:
            return True
g = graph()
g.addEdge("Himachal", "Punjab")
g.addEdge("Himachal", "TamilNadu")
g.addEdge("Punjab", "Himachal")
g.addEdge("Punjab", "TamilNadu")
g.addEdge("Punjab", "Kerala")
g.addEdge("TamilNadu", "Himachal")
g.addEdge("TamilNadu", "Punjab")
g.addEdge("TamilNadu", "Kerala")
g.addEdge("Kerala", "Punjab")
g.addEdge("Kerala", "TamilNadu")
g.print_graph()
g.make_adj()
pallet = ["White", "Red", "Pink", "Yellow", "Black"]
colors = [-1]*4
print("\nGraph Colors are: ")
if g.graphColor(colors):
    cities = ["Himachal", "Punjab", "TamilNadu", "Kerala"]
    for i in range(len(cities)):
        print(f"{cities[i]} : {pallet[colors[i]]}")
else:
    print("Not suffiecient colors!")
# print(colors)
```

Output:



Conclusion:

Hence, we have successfully implemented Graph Colouring Problem using Python.

Experiment 6

Objective: Write a python program to implement A* Algorithm.

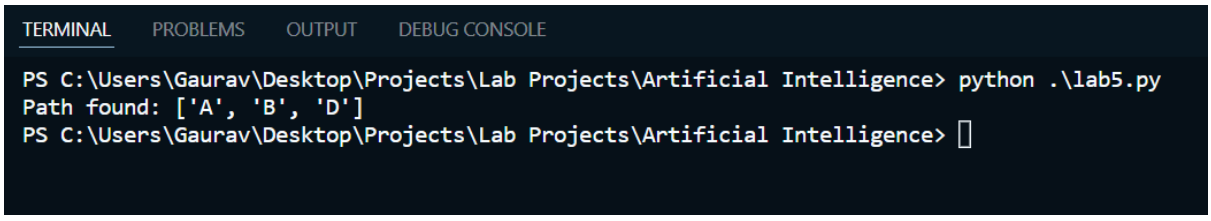
Software Used: Visual Studio Code

Program:

```
from collections import deque
class Graph:
    def __init__(self, adjacency_list):
        self.adjacency_list = adjacency_list
    def get_neighbors(self, v):
        return self.adjacency_list[v]
    def h(self, n):
        H = {
            'A': 1,
            'B': 1,
            'C': 1,
            'D': 1
        }
        return H[n]
    def a_star_algorithm(self, start_node, stop_node):
        open_list = set([start_node])
        closed_list = set([])
        g = {}

        g[start_node] = 0
        parents = {}
        parents[start_node] = start_node
        while len(open_list) > 0:
            n = None
            for v in open_list:
                if n == None or g[v] + self.h(v) < g[n] + self.h(n):
                    n = v;
            if n == None:
                print('Path does not exist!')
                return None
            if n == stop_node:
                reconst_path = []
                while parents[n] != n:
                    reconst_path.append(n)
                    n = parents[n]
                reconst_path.append(start_node)
                reconst_path.reverse()
```

```
            print('Path found: {}'.format(reconst_path))
            return reconst_path
        for (m, weight) in self.get_neighbors(n):

            if m not in open_list and m not in closed_list:
                open_list.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_list:
                        closed_list.remove(m)
                        open_list.add(m)

        open_list.remove(n)
        closed_list.add(n)

    print('Path does not exist!')
    return None
adjacency_list = {
   'A': [('B', 1), ('C', 3), ('D', 7)],
   'B': [('D', 5)],
   'C': [('D', 12)]
}
graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'D')
```

Output:



Conclusion:

Hence, we have successfully implemented A* Algorithm using Python.

Experiment 7

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

Software Used: Visual Studio Code

Program:

```
import random as ran
board = [[0,0,0],[0,0,0],[0,0,0]]
g = {
   1:[2,0],2:[2,1],3:[2,2],4:[1,0],5:[1,1],6:[1,2],7:[0,0],8:[0,1],9:[0,2]
}


def check(i,j):
   if board[i][j] == 0:
      return True
   else:
      return False


def checkleftdiagonal(x):
   if board[0][0] == board[1][1] == board[2][2] == x:
      return True
   else:
      return False


def checkrightdiagonal(x):
   if board[2][0] == board[1][1] == board[0][2] == x:
      return True
   else:
      return False
def checkRow(x):
```

```
    for i in range(3):
        if board[i][0] == board[i][1] == board[i][2] == x:
            return True
        else:
            return False

def checkCol(x):
    for i in range(3):
            if board[0][i] == board[1][i] == board[2][i] == x:
                return True
            else:
                return False

def game_choice(x):
    index = ran.randint(1,9)
    if (check(g[index][0],g[index][1])):
        print("Game choice is: ",index)
        board[g[index][0]][g[index][1]] = x
    else:
        game_choice(x)

def win(i):
    if(checkrightdiagonal(i)):
        return True
    elif(checkleftdiagonal(i)):
        return True
    elif(checkCol(i)):
        return True
    elif(checkRow(i)):
        return True
```

```
    return False

def player_choice(x):
    print("It's now",x,"turn")
    t = int(input("Enter the location Player: "))
    if(check(g[t][0],g[t][1])):
        print("Player Choice is: ",t)
        board[g[t][0]][g[t][1]] = x;

def displayboard():
    for i in range(3):
        print(" ")
        for j in range(3):
            print(board[i][j],"|",end=" ")
    print("\n")
def game():
    ch = int(input("Enter the choice 1,2: "))
    if(ch == 1):
        x = 2
    else:
        x = 1
    total_moves = 0
    while total_moves < 9:
        game_choice(x)
        total_moves += 1
        displayboard()
        if win(ch):
            print(ch,"Wins")
            break
        if win(x):
```

```
            print(x,"Wins")
            break
        print("\n")
        player_choice(ch)
        total_moves += 1
        displayboard()
        if win(ch):
            print(ch,"Wins")
            break
        if win(x):
            print(x,"Wins")
            break
        if(total_moves == 9):
            break
    print("||------Game Over------||")
    print("||------Game by Gaurav------||")
    print("Game is finished in",total_moves,"moves")
game()
```

Output:

```
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence> python .\lab6.py
Enter the choice 1,2: 2
Game choice is:  3

0 | 0 | 0 |
0 | 0 | 0 |
0 | 0 | 1 |



It's now 2 turn
Enter the location Player: 1
Player Choice is:  1

0 | 0 | 0 |
0 | 0 | 0 |
2 | 0 | 1 |

Game choice is:  6

0 | 0 | 0 |
0 | 0 | 1 |
2 | 0 | 1 |



It's now 2 turn
Enter the location Player: 3

0 | 0 | 0 |
0 | 0 | 1 |
2 | 0 | 1 |

Game choice is:  8
```

```
 It's now 2 turn
 Enter the location Player: 1

 0 | 1 | 0 |
 0 | 1 | 1 |
 2 | 0 | 1 |

 Game choice is:  2

 0 | 1 | 0 |
 0 | 1 | 1 |
 2 | 1 | 1 |



 It's now 2 turn
 Enter the location Player: 2

 0 | 1 | 0 |
 0 | 1 | 1 |
 2 | 1 | 1 |

 ||------Game Over------||
 ||------Game by Gaurav------||
 Game is finished in 10 moves
 PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence>
```

Conclusion:

Hence, we have successfully implemented Tic-Tac-Toe game using Python.

Experiment 8

Objective: Write a python program to develop Truth Tables for NAND gate, NOR gate, NOT gate and XOR gate.

Software Used: Visual Studio Code

Program:

```
print('WAP in python to develop for NAND gate, NOR gate, NOT gate and XOR gate\n')
def NOT(a):
    if a == 0:
        return 1
    elif a == 1:
        return 0
def NAND(a, b):
    if a == b == 1:
        return 0
    else:
        return 1
def NOR(a, b):
    if a == b == 0:
        return 1
    else:
        return 0
def XOR(a, b):
    if a != b:
        return 1
    else:
        return 0
print('Following are the Logic GATES for two input only( a,b )\n')
print('0 for False and 1 for True\n')
print("NAND Gate")
```

```
for a in range(2):
    for b in range(2):
        print('a = ', a, "b = ", b, "a NAND b = ", NAND(a, b))
print("NOR Gate")
for a in range(2):
    for b in range(2):
        print('a = ', a, "b = ", b, "a NOR b = ", NOR(a, b))
print("XOR Gate")
for a in range(2):
    for b in range(2):
        print('a = ', a, "b = ", b, "a XOR b = ", XOR(a, b))
print("NOT Gate")
for a in range(2):
    print('a = ', a, "a NOT = ", NOT(a))
```

Output:

```
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence> python .\lab7.p
WAP in python to develop for NAND gate, NOR gate, NOT gate and XOR gate

Following are the Logic GATES for two input only( a,b )

0 for False and 1 for True

NAND Gate
a =  0 b =  0 a NAND b =  1
a =  0 b =  1 a NAND b =  1
a =  1 b =  0 a NAND b =  1
a =  1 b =  1 a NAND b =  0
NOR Gate
a =  0 b =  0 a NOR b =  1
a =  0 b =  1 a NOR b =  0
a =  1 b =  0 a NOR b =  0
a =  1 b =  1 a NOR b =  0
XOR Gate
a =  0 b =  0 a XOR b =  0
a =  0 b =  1 a XOR b =  1
a =  1 b =  0 a XOR b =  1
a =  1 b =  1 a XOR b =  0
NOT Gate
a =  0 a NOT =  1
a =  1 a NOT =  0
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence>
```

Conclusion:

Hence, we have successfully implemented Truth Tables for various logic gates using Python.

Experiment 9

Objective: Write a python program to implement Tokenization of word and Sentences with the help of NLTK package.

Software Used: Visual Studio Code

Program:

```
from nltk.tokenize import sent_tokenize, word_tokenize
text = "Natural language processing (NLP) is a field " + \
    "of computer science, artificial intelligence " + \
    "and computational linguistics concerned with " + \
    "the interactions between computers and human " + \
    "(natural) languages, and, in particular, " + \
    "concerned with programming computers to " + \
    "fruitfully process large natural language " + \
    "corpora. Challenges in natural language " + \
    "processing frequently involve natural " + \
    "language understanding, natural language" + \
    "generation frequently from formal, machine" + \
    "-readable logical forms), connecting language " + \
    "and machine perception, managing human-" + \
    "computer dialog systems, or some combination " + \
    "thereof."

print(sent_tokenize(text))
print("  ")
print("  ")
```

Output:

```
['Natural language processing (NLP) is a field of computer science, artificial intelligence an
d computational linguistics concerned with the interactions between computers and human (natur
al) languages, and, in particular, concerned with programming computers to fruitfully process
large natural language corpora.', 'Challenges in natural language processing frequently involv
e natural language understanding, natural languagegeneration frequently from formal, machine-r
eadable logical forms), connecting language and machine perception, managing human-computer di
alog systems, or some combination thereof.']


['Natural', 'language', 'processing', '(', 'NLP', ')', 'is', 'a', 'field', 'of', 'computer', '
science', ',', 'artificial', 'intelligence', 'and', 'computational', 'linguistics', 'concerned
', 'with', 'the', 'interactions', 'between', 'computers', 'and', 'human', '(', 'natural', ')',
 'languages', ',', 'and', ',', 'in', 'particular', ',', 'concerned', 'with', 'programming', 'c
omputers', 'to', 'fruitfully', 'process', 'large', 'natural', 'language', 'corpora', '.', 'Cha
llenges', 'in', 'natural', 'language', 'processing', 'frequently', 'involve', 'natural', 'lang
uage', 'understanding', ',', 'natural', 'languagegeneration', 'frequently', 'from', 'formal',
',', 'machine-readable', 'logical', 'forms', ')', ',', 'connecting', 'language', 'and', 'machi
ne', 'perception', ',', 'managing', 'human-computer', 'dialog', 'systems', ',', 'or', 'some',
```

Conclusion:

Hence, we have successfully implemented Tokenization of word and Sentences with the help of NLTK package.

Experiment 10

Objective: Write a python program to implement Brute force solution to the Knapsack problem in Python.

Software Used: Visual Studio Code

Program:

```
def knapSack(W, wt, val, n):
    if n == 0 or W == 0:
        return 0
    if (wt[n-1] > W):
        return knapSack(W, wt, val, n-1)
    else:
        return max(val[n-1] + knapSack(W-wt[n-1], wt, val, n-1),
                   knapSack(W, wt, val, n-1))


val = [50, 100, 150, 200]
wt = [8, 16, 32, 40]
W = 64
n = len(val)
print("Knapsack by Gaurav")
print("Largest possible value:")
print(knapSack(W, wt, val, n))
```

Output:

```
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence\Practice> cd..
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence> python .\lab8.py
Knapsack by Gaurav
Largest possible value:
350
PS C:\Users\Gaurav\Desktop\Projects\Lab Projects\Artificial Intelligence>
```

Conclusion:

Hence, we have successfully implemented Brute force solution to the Knapsack problem.