PRACTICAL FILE

Course Title: Artificial Intelligence

Course Code: CSE401

Department of Computer Science and Engineering
Amity Institute of Engineering and Technology

Uttar Pradesh, Noida

Name: Bhavy Kumar Rajput

Class: 6CSE-Evng- X

Enrolment Number: A2345919033

Faculty Guide: Dr Gaurav Dubey sir

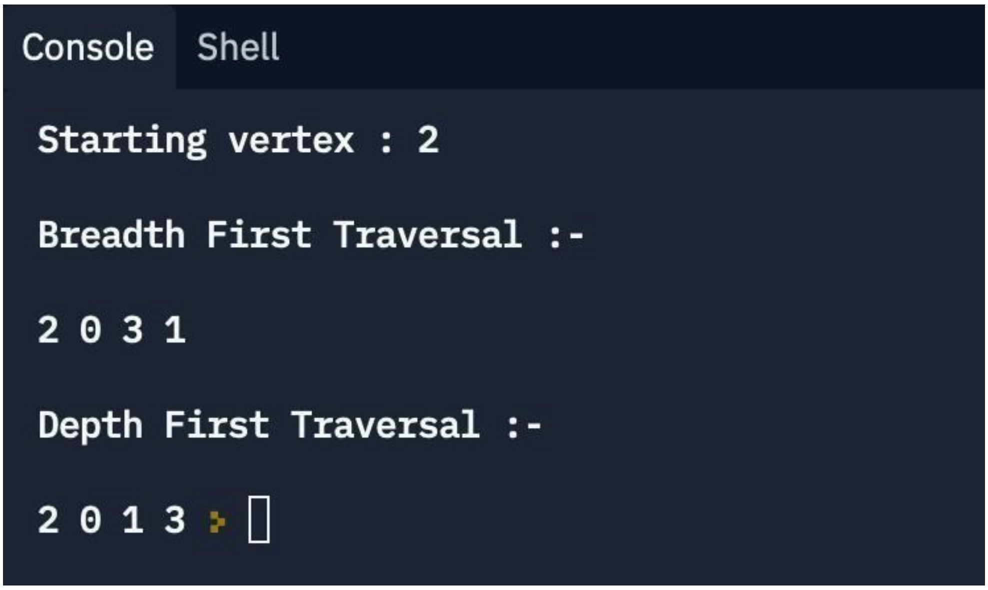
AI LAB WORK-1

Aim: Write a program in python for implementing BFS and DFS.

Code:

```
from collections import defaultdict
class Graph:
    def__init__(self):
        self.graph = defaultdict(list)
def addEdge(self,u,v):
        self.graph[u].append(v)
def BFS(self, s): visited = [False] * (max(self.graph)
    + 1) queue = [] queue.append(s) visited[s] =
    True
while queue:
            s = queue.pop(0) print
            (s, end = " ") for i
            in self.graph[s]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True
def DFSUtil(self, v, visited):
        visited.add(v)
        print(v, end=' ')
for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)
def DFS(self, v):
        visited = set() self.DFSUtil(v,
        visited)
g = Graph()
    g.addEdge(0, 1)
    g.addEdge(0, 2)
    g.addEdge(1, 2)
    g.addEdge(2, 0)
    g.addEdge(2, 3)
    g.addEdge(3, 3)
startNode = int(input("Starting vertex : ")) print
    ("\nBreadth First Traversal :- \n")
    g.BFS(startNode)
    print("\n\nDepth First Traversal :-\n")
    g.DFS(startNode)
```

Output:



AI LAB WORK-2

Aim: Write a program in python to develop tic tac toe game.

Code:

```
        import random
        class TicTacToe:
def init (self):
                self.board = []
            def create_board(self):
                for i in range(3): row =
                    [] for j in range(3):
                    row.append('-')
                    self.board.append(row)
def get_random_first_player(self): return
            random.randint(0, 1)
            def fix_spot(self, row, col, player):
                self.board[row][col] = player
            def is_player_win(self, player):
                win = None n =

                len(self.board)

                for i in range(n): win
                    = True for j in
                    range(n):
                        if self.board[i][j] != player:
                            win =
                    False break if
                    win:
                        return win
for i in range(n): win = True for j
                in range(n):
                        if self.board[j][i] != player:
                            win =
                    False break if
                    win:
```

```
                        return win
win = True for i in range(n):
                    if self.board[i][i] != player:
                        win = False
                break if win:
                    return win
win = True for i in range(n):
                    if self.board[i][n - 1 - i] != player:
                        win = False
                break if win:
                return win return
                False
for row in self.board:
                    for item in row:
                        if item == '-':
                            return False
                return True
def is_board_filled(self):
                for row in self.board:
                    for item in row:
                        if item == '-':
                            return False
                return True
            def swap_player_turn(self, player):
                return 'X' if player == 'O' else 'O'
            def show_board(self):
                for row in self.board:
                    for item in row:
                        print(item, end=" ")
                    print()
            def start(self):
                self.create_board()
player = 'X' if self.get_random_first_player() == 1 else 'O' while True:
                    print(f"Player {player} turn")

                    self.show_board()

                    row, col = list( map(int, input("Enter row and column
                        numbers to fix spot:
        ").split())) print() self.fix_spot(row - 1, col -
                    1, player)

                    if self.is_player_win(player):
                        print(f"Player {player} wins the game!")
                        break
if self.is_board_filled():
                        print("Match Draw!")
                        break
                    player = self.swap_player_turn(player)
                print()
        self.show_board() tic_tac_toe =
        TicTacToe() tic_tac_toe.start()
```

Output:

```
Player 0 turn
- - -
- - -
- - -
Enter row and column numbers to fix spot: 1 1

Player X turn
0 - -
- - -
- - -
Enter row and column numbers to fix spot: 2 2

Player 0 turn
0 - -
- X -
- - -
Enter row and column numbers to fix spot: 1 2

Player X turn
0 0 -
- X -
- - -
Enter row and column numbers to fix spot: 2 3

Player 0 turn
0 0 -
- X X
- - -
Enter row and column numbers to fix spot: 1 3

Player 0 wins the game!

0 0 0
- X X
- - -
>
```

AI LAB WORK-3

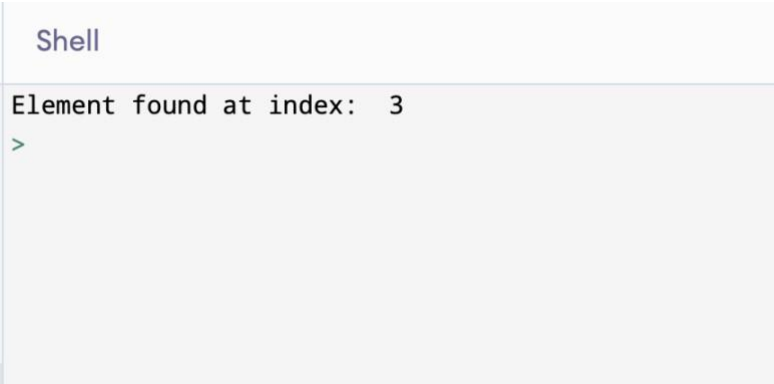
Aim: Write program from linear search and binary Search.

Code:

```
# Linear Search in Python
def linearSearch(array, n, x):

    # Going through array sequencially
    for i in range(0, n):
        if (array[i] == x):
            return i
    return -1


array = [2, 4, 0, 1, 9] x = 1 n =
len(array) result =
```

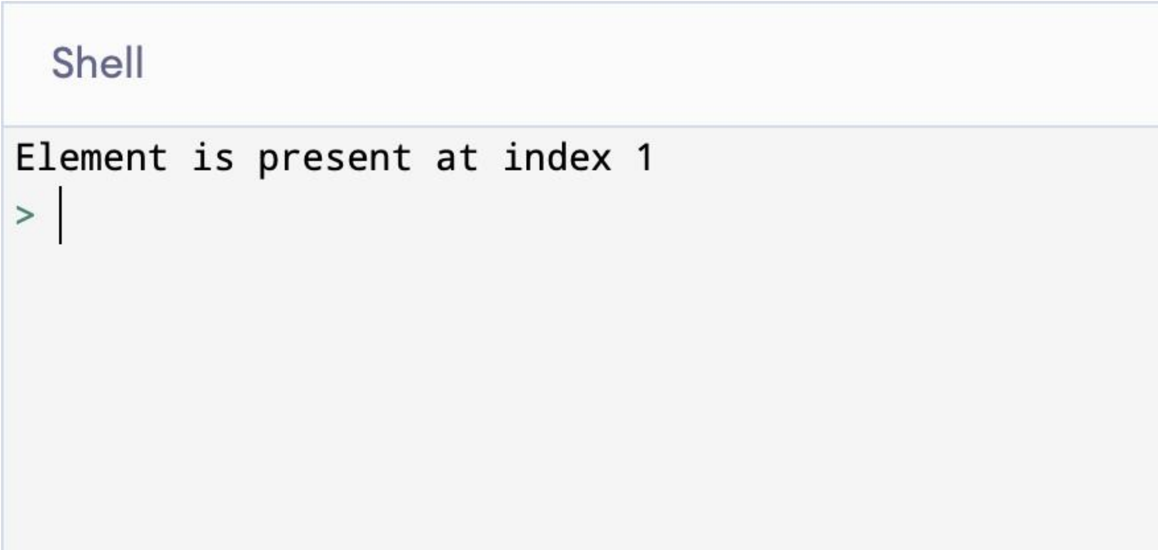
```
linearSearch(array, n, x)
if(result == -1):
    print("Element not found")
else:
    print("Element found at index: ", result)
```

Output:




```
# Binary Search in python
def binarySearch(array, x, low, high):

    # Repeat until the pointers low and high meet each
    other while low <= high: mid = low + (high - low)//2

        if array[mid] == x:
            return mid
elif array[mid] < x:
            low = mid + 1
else: high = mid - 1 return -1

array = [3, 4, 5, 6, 7, 8, 9] x = 4 result =
binarySearch(array, x, 0, len(array)-1)

if result != -1:
    print("Element is present at index " + str(result))
else:
    print("Not found”)
```

Output:



Aim: Write a model in python for implementing Disease detection by Logistic Regression.

Dataset used: Heart_Disease_Prediction.csv

Code:

+ Code + Text RAM Disk Editing

```
[1] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    from scipy import stats
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
    from sklearn import metrics
    from sklearn.linear_model import LogisticRegression
```

```
[2] df = pd.read_csv('./Heart_Disease_Prediction.csv')
    pd.set_option('display.max_rows', None)
    df.head()
```

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST	Number of vessels fluro	Thallium	Heart Disease
0	70	1	4	130	322	0	2	109	0	2.4	2	3	3	Presence
1	67	0	3	115	564	0	2	160	0	1.6	2	0	7	Absence
2	57	1	2	124	261	0	0	141	0	0.3	1	0	7	Presence
3	64	1	4	128	263	0	0	105	1	0.2	2	1	7	Absence
4	74	0	2	120	269	0	2	121	1	0.2	1	1	3	Absence

```
[3] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Age                      270 non-null    int64
 1   Sex                      270 non-null    int64
 2   Chest pain type          270 non-null    int64
 3   BP                       270 non-null    int64
 4   Cholesterol              270 non-null    int64
 5   FBS over 120             270 non-null    int64
 6   EKG results              270 non-null    int64
 7   Max HR                   270 non-null    int64
 8   Exercise angina          270 non-null    int64
 9   ST depression            270 non-null    float64
 10  Slope of ST              270 non-null    int64
 11  Number of vessels fluro  270 non-null    int64
 12  Thallium                 270 non-null    int64
 13  Heart Disease            270 non-null    object
dtypes: float64(1), int64(12), object(1)
memory usage: 29.7+ KB
```

0.000000 2.000000 153.500000 0.000000 0.80000 2.000

i 66.000000 1.000000 4 .60000 2.000

6.20000 3.000

7ballium

```
dt.describe(l
```

	Age	Sex	paic tYp°		Cholesterol	120	results				Slope
count	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.00000	270.000
mean	54.433333	0677Т78	3.17407^.	131.34444^.	249.659259	0.148148	1.022222	149.677778	0.329630	1.05000	1.585
std	9.109067	0.468195	0.950090	17.861608	51.686237	0.355906	0.997891	23.165717	0.470952	1.14521	0.614
min	29.000000	0000000	1.000000	94.000000	126.000000	0.000000	0.000000	74.000000	0.000000	0.00000	1.000
	48.000000	0.000000	3.000000	120.000000	213.000000	0.000000	0.000000	133.000000	0.000000	0.00000	1.000
	55.000000	1000000	3.000000	130.000000	245.000000						
	61.000000	i.000000	4.000000	140.000000	280.000000	0.000000	2.000000				
	77.000000	i 000000	4000000	200.000000	564.000000	4000000	2.000000	202.000000	4.000000		

```
[5J dt[ J.value_counts()
```

```
Absence
Presence    120
Name: Heart Disease, dtype: int64
```

```
Heart_Disease = T
df[                [Heart_Disease [x]   x ln dl [
df.head()
```

		Sex	Chest		CbolesCerol	over Z20			angioa	aepresslon	6lope	fIUFO	Heart
	70	1	4	130	322	0	2	109	0				
	67	0	3	115	561	0	2	160	0				7
	57	1		12^	261	0	0	141	0				
	64		4	i28	263	0		i05		0.2			
4		0	2	i20	269	0	2	i21		0.2			

```
[7] x = dt.drop(                    , axis=1l
    y = d1'[                    ]
    x_train, x_test, y_train, y_test =  tra1n_test_spAlt(x, y, test_size=0.3)
```

```
[8  lr = LogistlcRegress ton()
    lr.lit(x_train, y_train)

    y_pred = Tr.predict(x_test)
    print(                                        , metrics.accuracy_score(y_tesl, y_pred)
```

```
Accuracy of logisllc regression ctassilier:     0.8024691358024691
/usr/local/t1b/python3.7/d1st—packages/sktearn/linear_nodet/_tog is tic.py: 818 :
STOP: TOTAL N0. of ITERATIONS REACHED LIMIT.

Increase the number of ite rations (riax_1ter) or scale tne data as shown in:

Please also refer to lhe documentation tor aPternative soAver options

  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
     sklearn.metrics          classitication_report
heart_report_names = [            ,            ]
print(etassification_report(y_tesl, y_pred, target_names=heart_report_names))
```

	precls ion	rec alt	fl-score	support
Present	0.86	0.78	0.82	46
Absent	0.74	0.83	0.78	35
accuracy			0.80	81
macro avg	0.80	0.81		81
weighted avg	0.81	0.80	0.80	81

2.^ 2 1 1.6 2 0 0

0. 1 0 1

2 7 0

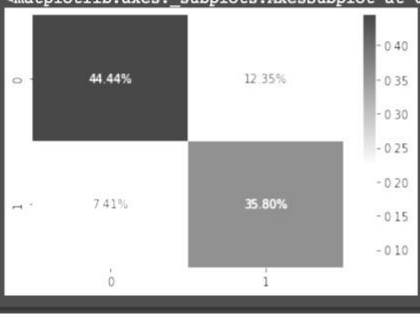
1 1 3 0

Conve rgenceNa rn ing : Lbfgs 1'aiTed t o conve rc

```
[10] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Age                      270 non-null    int64
 1   Sex                      270 non-null    int64
 2   Chest pain type          270 non-null    int64
 3   BP                       270 non-null    int64
 4   Cholesterol              270 non-null    int64
 5   FBS over 120             270 non-null    int64
 6   EKG results              270 non-null    int64
 7   Max HR                   270 non-null    int64
 8   Exercise angina          270 non-null    int64
 9   ST depression            270 non-null    float64
 10  Slope of ST              270 non-null    int64
 11  Number of vessels fluro  270 non-null    int64
 12  Thallium                 270 non-null    int64
 13  Heart Disease            270 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 29.7 KB
```

```
from sklearn.metrics import confusion_matrix
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2a45c42d90>
```



AI LAB WORK-5

Aim: Write a model in python for implementing Disease detection by classification.

Dataset used: Heart_Disease_Prediction.csv

Code:

```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    from scipy import stats
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
    from sklearn import metrics
    from sklearn.linear_model import LogisticRegression
    from sklearn.svm import SVC
```

```
df = pd.read_csv('/Users/mac/Downloads/Heart_Disease_Prediction.csv')
pd.set_option('display.max_rows', None)
df.head()
```

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST	Number of vessels fluro	Thallium	Heart Disease
0	70	1	4	130	322	0	2	109	0	2.4	2	3	3	Presence
1	67	0	3	115	564	0	2	160	0	1.6	2	0	7	Absence
2	57	1	2	124	261	0	0	141	0	0.3	1	0	7	Presence
3	64	1	4	128	263	0	0	105	1	0.2	2	1	7	Absence
4	74	0	2	120	269	0	2	121	1	0.2	1	1	3	Absence

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 14 columns):
Age                         270 non-null int64
Sex                         270 non-null int64
Chest pain type             270 non-null int64
BP                          270 non-null int64
Cholesterol                 270 non-null int64
FBS over 120                270 non-null int64
EKG results                 270 non-null int64
Max HR                      270 non-null int64
Exercise angina             270 non-null int64
ST depression               270 non-null float64
Slope of ST                 270 non-null int64
Number of vessels fluro     270 non-null int64
Thallium                    270 non-null int64
Heart Disease               270 non-null object
dtypes: float64(1), int64(12), object(1)
memory usage: 29.6+ KB
```

```
df [ ].value_counts()

Absence 150
Presence 120

Heart_Disease 1, 01
df [ Heart_Disease[x) 1n df l df . head (
)

x dl . d rop( , axzs=1)
dfl
x_traln,x_test,y_traln,y_test = train_test_split(x,y,test_size=0.3, random_state=42)

lr.predict(x_test)
l r = Log 1s t1cReg res s 1on ( )
l r. f1t (x_t raln , y_t raln )
y_p , met mcs . accu rac y_s
co re ( y_test , y_p red red ) )
3r int (

sklearn.metrics classification_report
heart_report_names
int(classification_report(y_test y_pred target_names=heart_report_names))

precision recall —sco re support

49
32

P resent 0.85 0.96
micro
Absent 8. 92 0.75 0.83

avg
macro avg
uelght ed avg

SVITl SVC ( ke rne1=

print( , metrics.accuracy_score)
```

```
Accuracy from sematic vector classfier: <function accuracy_score at 0x1a1b830c80>

heart_report_names = [          ,          )
print(classification_report(y_test, y_pred,  target_names=heart_report_names))

              precision    recall  f1-score   support

     Present       0.82      0.92      0.87        49
      Absent       0.85      0.69      0.76        32

   micro avg       0.83      0.83      0.83        81
   macro avg       0.83      0.80      0.81        81
weighted avg       0.83      0.83      0.82        81
```

AI LAB WORK-6

Aim: Write programs to implement A* and Fractional Knapsack algorithms in Python.

Code:

A* Algorithm

```
from collections import deque

class Graph:
    # example of adjacency list (or rather map)
    # adjacency_list = {
    # 'A': [('B', 1), ('C', 3), ('D', 7)],
    # 'B': [('D', 5)],
    # 'C': [('D', 12)]
    # }
def init (self, adjacency_list):
        self.adjacency_list = adjacency_list
def get_neighbors(self, v):
        return self.adjacency_list[v]

    # heuristic function with equal values for all
    nodes 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 is a list of nodes which have been visited,
but who's neighbors
        # haven't all been inspected, starts off with the start
node
        # closed_list is a list of nodes which have been visited
        # and who's neighbors have been inspected open_list
        = set([start_node])
        closed_list = set([])

        # g contains current distances from start_node to all
other nodes
        # the default value (if it's not found in the map) is
+infinity g =
        {}
g[start_node] = 0
```

```
                # parents contains an adjacency map of all nodes
                parents = {}
                parents[start_node] = start_node
 while len(open_list) > 0:
                    n = None

                    # find a node with the lowest value of f() -
        evaluation function 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 the current node is the stop_node
                    # then we begin reconstructin the path from it to the
        start_node       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 all neighbors of the current node do   for (m, weight)
in self.get_neighbors(n):
                          # if the current node isn't in both open_list and
        closed_list
                        # add it to open_list and note n as it's parent
                        if m not in open_list and m not in closed_list:
                            open_list.add(m)
                            parents[m] = n g[m]
                            = g[n] + weight


 # otherwise, check if it's quicker to first visit n, then m
                        # and if it is, update parent data and g data
                          # and if the node was in the closed_list, move it
        to open_list  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)
```

```
            # remove n from the open_list, and add it to
closed_list
            # because all of his neighbors were
            inspected open_list.remove(n)
            closed_list.add(n)


        print('Path does not exist!')
        return None
```

Fractional Knapsack Algorithm

```
class Solution:
     def solve(self, weights, values, capacity):
          res = 0
          for pair in sorted(zip(weights, values), key=lambda x: -
x[1]/x[0]): if not bool(capacity):
               break
               if pair[0] > capacity: res += int(pair[1] /
                    (pair[0] / capacity)) capacity = 0
               elif pair[0] <= capacity:
                    res += pair[1]
                    capacity -= pair[0]
          return int(res)

ob = Solution() weights = [6, 7, 3]
values = [110, 120, 2] capacity = 10
print(ob.solve(weights, values,
capacity))
```

Output:

Result

```
$python main.py
230
```