bfs

graph = {'A':['B','C'],

'B':['A','D','E'],

'C':['A','F','G'],

'D':['B'],

'E':['B'],

'F':['C'],

'G':['C']

}

visited = [] # List to keep track of visited nodes.

queue = [] #Initialize a queue

def bfs(visited, graph, node):

visited.append(node)

queue.append(node)

while queue:

s = queue.pop(0)

print (s)

for neighbour in graph[s]:

if neighbour not in visited:

visited.append(neighbour)

queue.append(neighbour)

# Driver Code

bfs(visited, graph, 'A')

#DFS

def dfs(graph,node,visited):

if node not in visited:

print(node)

visited.add(node)

for neighbour in graph[node]:

dfs(graph,neighbour,visited)

graph={

'A':['B','C'],

'B':['A','D','E'],

'C':['A','F','G'],

'D':['B'],

'E':['B'],

'F':['C'],

'G':['C']

}

dfs(graph,'A',set())

lab 2

player, opponent = 'x', 'o'

def isMovesLeft(board) :

for i in range(3) :

for j in range(3) :

if (board[i][j] == '\_') :

return True

return False

def evaluate(b) :

# Checking for Rows for X or O victory.

for row in range(3) :

if (b[row][0] == b[row][1] and b[row][1] == b[row][2]) :

if (b[row][0] == player) :

return 10

elif (b[row][0] == opponent) :

return -10

# Checking for Columns for X or O victory.

for col in range(3) :

if (b[0][col] == b[1][col] and b[1][col] == b[2][col]) :

if (b[0][col] == player) :

return 10

elif (b[0][col] == opponent) :

return -10

# Checking for Diagonals for X or O victory.

if (b[0][0] == b[1][1] and b[1][1] == b[2][2]) :

if (b[0][0] == player) :

return 10

elif (b[0][0] == opponent) :

return -10

if (b[0][2] == b[1][1] and b[1][1] == b[2][0]) :

if (b[0][2] == player) :

return 10

elif (b[0][2] == opponent) :

return -10

# Else if none of them have won then return 0

return 0

def minimax(board, depth, isMax) :

score = evaluate(board)

if (score == 10) :

return score

# If Minimizer has won the game return his/her

# evaluated score

if (score == -10) :

return score

# If there are no more moves and no winner then

# it is a tie

if (isMovesLeft(board) == False) :

return 0

# If this maximizer's move

if (isMax) :

best = -1000

# Traverse all cells

for i in range(3) :

for j in range(3) :

# Check if cell is empty

if (board[i][j]=='\_') :

# Make the move

board[i][j] = player

# Call minimax recursively and choose

# the maximum value

best = max( best, minimax(board,

depth + 1,

not isMax) )

# Undo the move

board[i][j] = '\_'

return best

# If this minimizer's move

else :

best = 1000

# Traverse all cells

for i in range(3) :

for j in range(3) :

# Check if cell is empty

if (board[i][j] == '\_') :

# Make the move

board[i][j] = opponent

# Call minimax recursively and choose

# the minimum value

best = min(best, minimax(board, depth + 1, not isMax))

# Undo the move

board[i][j] = '\_'

return best

def findBestMove(board) :

bestVal = -1000

bestMove = (-1, -1)

for i in range(3) :

for j in range(3) :

# Check if cell is empty

if (board[i][j] == '\_') :

# Make the move

board[i][j] = player

moveVal = minimax(board, 0, False)

# Undo the move

board[i][j] = '\_'

if (moveVal > bestVal) :

bestMove = (i, j)

bestVal = moveVal

print("The value of the best Move is :", bestVal)

print()

return bestMove

# Driver code

board = [

[ 'x', 'o', 'x' ],

[ 'o', 'o', 'x' ],

[ '\_', '\_', '\_' ]

]

bestMove = findBestMove(board)

print("The Optimal Move is :")

print("ROW:", bestMove[0], " COL:", bestMove[1])

lab 3

**def** selectionSort(array, size):

**for** step **in** range(size):

min\_idx = step

**for** i **in** range(step **+** 1, size):

**if** array[i] **<** array[min\_idx]:

min\_idx = i

(array[step], array[min\_idx]) = (array[min\_idx], array[step])

data = [2, 45, 0, 11, 9, 56, 3]

size = len(data)

selectionSort(data, size)

**print**('Sorted Array in Ascending Order:')

**print**(data)

*# Prim's Algorithm in Python*

INF = 9999999

V = 5

G = [[0, 9, 75, 0, 0],

[9, 0, 95, 19, 42],

[75, 95, 0, 51, 66],

[0, 19, 51, 0, 31],

[0, 42, 66, 31, 0]]

selected = [0, 0, 0, 0, 0]

no\_edge = 0

selected[0] = True

**print**("Edge : Weight\n")

**while** (no\_edge **<** V **-** 1):

minimum = INF

x = 0

y = 0

**for** i **in** range(V):

**if** selected[i]:

**for** j **in** range(V):

**if** ((**not** selected[j]) **and** G[i][j]):

**if** minimum **>** G[i][j]:

minimum = G[i][j]

x = i

y = j

**print**(str(x) **+** "-" **+** str(y) **+** ":" **+** str(G[x][y]))

selected[y] = True

no\_edge += 1

lab4

N=8

def printSolution(board):

for i in range(N):

for j in range(N):

print(board[i][j], end = " ")

print()

def isSafe(row, col, slashCode, backslashCode,

rowLookup, slashCodeLookup,

backslashCodeLookup):

if (slashCodeLookup[slashCode[row][col]] or

backslashCodeLookup[backslashCode[row][col]] or

rowLookup[row]):

return False

return True

def solveNQueensUtil(board, col, slashCode, backslashCode,

rowLookup, slashCodeLookup,

backslashCodeLookup):

if(col >= N):

return True

for i in range(N):

if(isSafe(i, col, slashCode, backslashCode,

rowLookup, slashCodeLookup,

backslashCodeLookup)):

board[i][col] = 1

rowLookup[i] = True

slashCodeLookup[slashCode[i][col]] = True

backslashCodeLookup[backslashCode[i][col]] = True

if(solveNQueensUtil(board, col + 1,

slashCode, backslashCode,

rowLookup, slashCodeLookup,

backslashCodeLookup)):

return True

board[i][col] = 0

rowLookup[i] = False

slashCodeLookup[slashCode[i][col]] = False

backslashCodeLookup[backslashCode[i][col]] = False

return False

def solveNQueens():

board = [[0 for i in range(N)]

for j in range(N)]

# helper matrices

slashCode = [[0 for i in range(N)]

for j in range(N)]

backslashCode = [[0 for i in range(N)]

for j in range(N)]

# arrays to tell us which rows are occupied

rowLookup = [False] \* N

# keep two arrays to tell us

# which diagonals are occupied

x = 2 \* N - 1

slashCodeLookup = [False] \* x

backslashCodeLookup = [False] \* x

# initialize helper matrices

for rr in range(N):

for cc in range(N):

slashCode[rr][cc] = rr + cc

backslashCode[rr][cc] = rr - cc + 7

if(solveNQueensUtil(board, 0, slashCode, backslashCode,

rowLookup, slashCodeLookup,

backslashCodeLookup) == False):

print("Solution does not exist")

return False

# solution found

printSolution(board)

return True

# Driver Code

solveNQueens()

lab 5

import random

R\_EATING = "I don't like eating anything because I'm a bot obviously!"

R\_ADVICE = "If I were you, I would go to the internet and type exactly what you wrote there!"

def unknown():

response = ["Could you please re-phrase that? ",

"...",

"Sounds about right.",

"What does that mean?"][

random.randrange(4)]

return response

import re

import long\_responses as long

def message\_probability(user\_message, recognised\_words, single\_response=False, required\_words=[]):

message\_certainty = 0

has\_required\_words = True

# Counts how many words are present in each predefined message

for word in user\_message:

if word in recognised\_words:

message\_certainty += 1

# Calculates the percent of recognised words in a user message

percentage = float(message\_certainty) / float(len(recognised\_words))

# Checks that the required words are in the string

for word in required\_words:

if word not in user\_message:

has\_required\_words = False

break

# Must either have the required words, or be a single response

if has\_required\_words or single\_response:

return int(percentage \* 100)

else:

return 0

def check\_all\_messages(message):

highest\_prob\_list = {}

# Simplifies response creation / adds it to the dict

def response(bot\_response, list\_of\_words, single\_response=False, required\_words=[]):

nonlocal highest\_prob\_list

highest\_prob\_list[bot\_response] = message\_probability(message, list\_of\_words, single\_response, required\_words)

# Responses -------------------------------------------------------------------------------------------------------

response('Hello!', ['hello', 'hi', 'hey', 'sup', 'heyo'], single\_response=True)

response('See you!', ['bye', 'goodbye'], single\_response=True)

response('I\'m doing fine, and you?', ['how', 'are', 'you', 'doing'], required\_words=['how'])

response('You\'re welcome!', ['thank', 'thanks'], single\_response=True)

response('Thank you!', ['i', 'love', 'code', 'palace'], required\_words=['code', 'palace'])

# Longer responses

response(long.R\_ADVICE, ['give', 'advice'], required\_words=['advice'])

response(long.R\_EATING, ['what', 'you', 'eat'], required\_words=['you', 'eat'])

best\_match = max(highest\_prob\_list, key=highest\_prob\_list.get)

# print(highest\_prob\_list)

# print(f'Best match = {best\_match} | Score: {highest\_prob\_list[best\_match]}')

return long.unknown() if highest\_prob\_list[best\_match] < 1 else best\_match

# Used to get the response

def get\_response(user\_input):

split\_message = re.split(r'\s+|[,;?!.-]\s\*', user\_input.lower())

response = check\_all\_messages(split\_message)

return response

# Testing the response system

while True:

print('Bot: ' + get\_response(input('You: ')))

lab 6

**import pandas as pd**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**from sklearn.preprocessing import** LabelEncoder, StandardScaler

**from sklearn.model\_selection import** train\_test\_split, GridSearch

CV

**from sklearn.metrics import** accuracy\_score, classification\_repor

t, confusion\_matrix

**import warnings**

warnings.filterwarnings('ignore')

%**matplotlib** inline

data = pd.read\_csv('Hr.csv')

data.shape

data.head()

data.info()

dept = data.iloc[:,[5,27]].copy()

dept\_per = dept.copy()

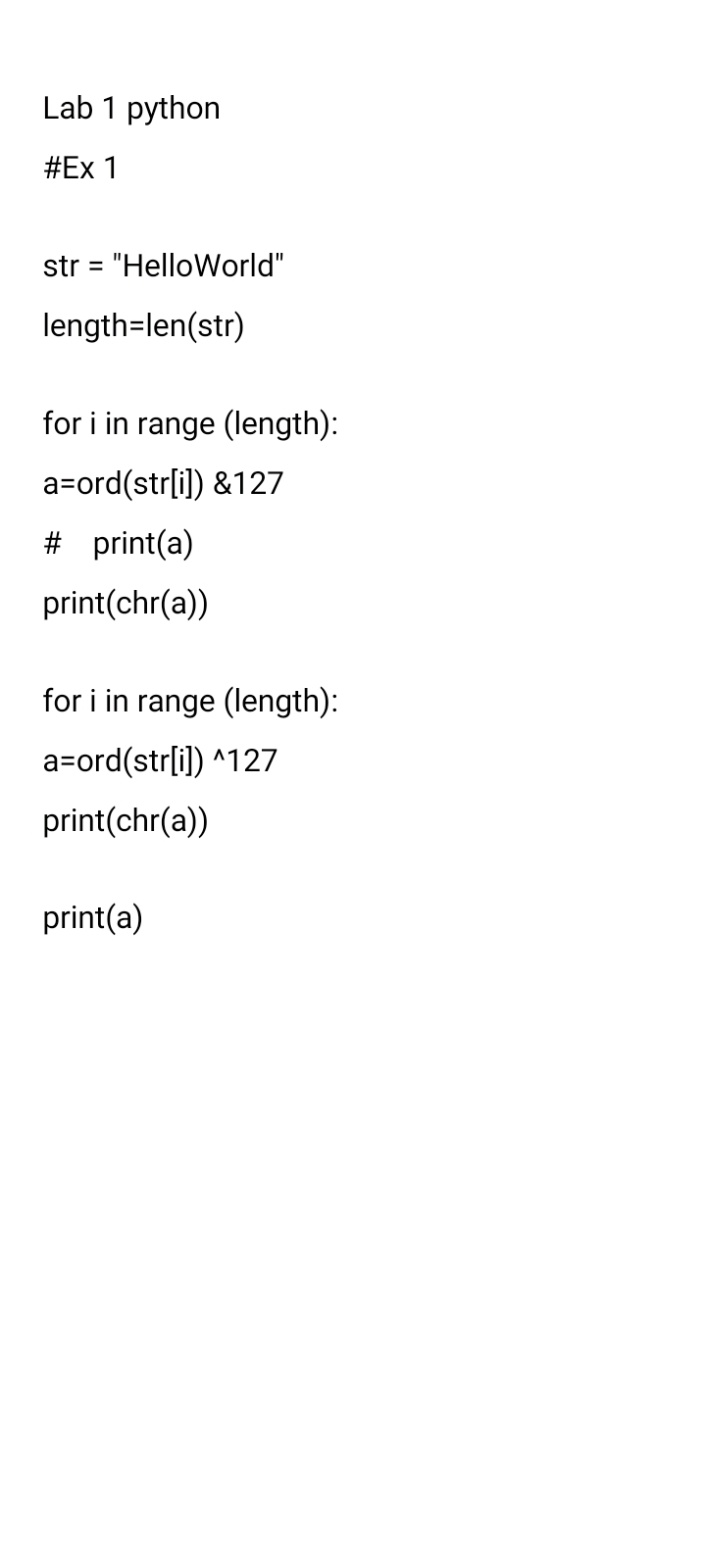
dept\_per.groupby(by='EmpDepartment')['PerformanceRating'].mean()

plt.figure(figsize=(10,4.5))

sns.barplot(dept\_per['EmpDepartment'],dept\_per['PerformanceRatin

g'])

data.corr()

data.drop(['EmpNumber'],inplace=True,axis=1)

**def** encryptRailFence(text, key):

rail **=** [['\n' **for** i **in** range(len(text))]

**for** j **in** range(key)]

dir\_down **= False**

row, col **=** 0, 0

**for** i **in** range(len(text)):

**if** (row **==** 0) **or** (row **==** key **-** 1):

dir\_down **= not** dir\_down

rail[row][col] **=** text[i]

col **+=** 1

**if** dir\_down:

row **+=** 1

**else**:

row **-=** 1

result **=** []

**for** i **in** range(key):

**for** j **in** range(len(text)):

**if** rail[i][j] **!=** '\n':

result.append(rail[i][j])

**return**("" . join(result))

**def** decryptRailFence(cipher, key):

rail **=** [['\n' **for** i **in** range(len(cipher))]

**for** j **in** range(key)]

dir\_down **= None**

row, col **=** 0, 0

**for** i **in** range(len(cipher)):

**if** row **==** 0:

dir\_down **= True**

**if** row **==** key **-** 1:

dir\_down **= False**

rail[row][col] **=** '\*'

col **+=** 1

**if** dir\_down:

row **+=** 1

**else**:

row **-=** 1

index **=** 0

**for** i **in** range(key):

**for** j **in** range(len(cipher)):

**if** ((rail[i][j] **==** '\*') **and**

(index **<** len(cipher))):

rail[i][j] **=** cipher[index]

index **+=** 1

result **=** []

row, col **=** 0, 0

**for** i **in** range(len(cipher)):

**if** row **==** 0:

dir\_down **= True**

**if** row **==** key**-**1:

dir\_down **= False**

**if** (rail[row][col] **!=** '\*'):

result.append(rail[row][col])

col **+=** 1

**if** dir\_down:

row **+=** 1

**else**:

row **-=** 1

**return**("".join(result))

**if** \_\_name\_\_ **==** "\_\_main\_\_":

print(encryptRailFence("attack at once", 2))

print(encryptRailFence("information security ", 3))

print(encryptRailFence("defend the east wall", 3))

print(decryptRailFence("atc toctaka ne", 2))

print(decryptRailFence("dnhaweedtees alf tl", 3))

RSA

import math

def gcd(a, h):

temp = 0

while(1):

temp = a % h

if (temp == 0):

return h

a = h

h = temp

p = 3

q = 7

n = p\*q

e = 2

phi = (p-1)\*(q-1)

while (e < phi):

if(gcd(e, phi) == 1):

break

else:

e = e+1

k = 2

d = (1 + (k\*phi))/e

# Message to be encrypted

msg = 12.0

print("Message data = ", msg)

# Encryption c = (msg ^ e) % n

c = pow(msg, e)

c = math.fmod(c, n)

print("Encrypted data = ", c)

m = pow(c, d)

m = math.fmod(m, n)

print("Original Message Sent = ", m)

deffie

-

<html>

<head>

<title>Diffie-HellmanKey Exchange</title>

</head>

<body>

<h2>Diffie-HellmanKey Exchange</h2>

<hr>

<script>

function power(a, b, p)

{ if (b== 1)

return a;

else

return((Math.pow(a, b)) % p);

}

// Driver code

var P, G, x, a, y, b, ka, kb;

P = 11;

document.write("The value of P:" + P + "<br>");

G = 7;

document.write("The value of G:" + G + "<br>");

a = 4;

document.write("The private key a for Alice:" +

a + "<br>");

x = power(G, a, P);

b = 3;

document.write("The private key b for Bob:" +

b + "<br>");

y = power(G, b, P);

ka = power(y, a, P); // Secret key for Alice

kb = power(x, b, P); // Secret key for Bob

document.write("Secret key for the Alice is:" +

ka + "<br>");

document.write("Secret key for the Bob is:" +

kb + "<br>");

</script>

</body>

</html>

md5

