DAA ASSIGNMENT 5

NAME:- Raunak Thanawala

Assignment:- Fractional Knapsack and Huffman Encoding

Batch:- CE Group C

AIM:-

1. Consider a XYZ courier company. They receive different goods to transport to different cities. Company needs to ship the goods based on their life and value. Goods having less shelf life and high cost shall be shipped earlier. Consider a list of 100 such items and the capacity of a transport vehicle is 200 tons. Implement an Algorithm for the fractional knapsack problem.
2. Download books from the website in html, text, doc, and pdf format. Compress these books using Hoffman coding technique. Find the compression ratio.

CODE:-

1. Fractional Knapsack:

"""

Name: Raunak Thanawala

Experiment: Fractional Knapsack

Batch: C

Registration Number: 231070051

"""

import pandas as pd

def read\_csv(filepath):

data = pd.read\_csv(filepath)

if data.empty:

raise ValueError("EMPTY FILE")

if data.isnull().any().any():

raise ValueError("INVALID SIZE OF COLUMNS")

for col in data.columns[2:]:

if not data[col].apply(lambda x: isinstance(x, int) and x > 0).all():

raise ValueError("INVALID INPUT")

return data.to\_dict(orient='records')

def knapsack(items, capacity):

for item in items:

item['value\_index'] = item['Cost'] / (item['ShelfLife'] \* item['Capacity'])

items = sorted(items, key=lambda x: x['value\_index'], reverse=True)

total\_cost = 0

current\_capacity = 0

for item in items:

if current\_capacity >= capacity:

break

if item['Capacity'] + current\_capacity <= capacity:

total\_cost += item['Cost']

current\_capacity += item['Capacity']

else:

fraction = (capacity - current\_capacity) / item['Capacity']

total\_cost += fraction \* item['Cost']

current\_capacity = capacity

return total\_cost

def main():

filepath = "C:/Users/ASUS/Desktop/Coding/DAA/DAA LAB5/knapsack\_data\_5.csv"

capacity = 200

items = read\_csv(filepath)

cost = knapsack(items, capacity)

cost = round(cost, 2)

print(

f"The Maximum Cost in Transport vehicles such that Items have Max Cost, "

f"Min Capacity and Min Shelf Life is {cost}"

)

if \_\_name\_\_ == '\_\_main\_\_':

main()

1. Huffman Encoding:

"""

Name: Raunak Thanawala

Experiment: Huffman Encoding

Batch: C

Registration Number: 231070051

"""

import heapq

import os

from collections import Counter

from PyPDF2 import PdfReader

from bs4 import BeautifulSoup

from docx import Document

class Node:

def \_\_init\_\_(self, char, freq):

self.char = char

self.freq = freq

self.left = None

self.right = None

def \_\_lt\_\_(self, other):

return self.freq < other.freq

def calculate\_compression\_ratio(input\_file, output\_file):

if os.path.exists(input\_file) and os.path.exists(output\_file):

return (os.path.getsize(input\_file) \* 8) / os.path.getsize(output\_file)

return 0

def build\_huffman\_tree(text):

frequency = Counter(text)

priority\_queue = [Node(char, freq) for char, freq in frequency.items()]

heapq.heapify(priority\_queue)

while len(priority\_queue) > 1:

left = heapq.heappop(priority\_queue)

right = heapq.heappop(priority\_queue)

merged = Node(None, left.freq + right.freq)

merged.left = left

merged.right = right

heapq.heappush(priority\_queue, merged)

return priority\_queue[0] *# Return the root of the tree*

def generate\_codes(root, code="", codebook=None):

if codebook is None:

codebook = {}

if root is not None:

if root.char is not None:

codebook[root.char] = code

generate\_codes(root.left, code + "0", codebook)

generate\_codes(root.right, code + "1", codebook)

return codebook

def encode\_file(input\_file, output\_file, codebook):

with open(input\_file, 'r') as f:

text = f.read()

encoded\_text = ''.join(codebook[char] for char in text)

padding = 8 - len(encoded\_text) % 8

encoded\_text = f"{padding:08b}" + encoded\_text + '0' \* padding

byte\_array = bytearray(int(encoded\_text[i:i + 8], 2) for i in range(0, len(encoded\_text), 8))

with open(output\_file, 'wb') as f:

f.write(byte\_array)

return output\_file

def extract\_text\_from\_pdf(file\_path):

text = ""

with open(file\_path, 'rb') as f:

reader = PdfReader(f)

for page in reader.pages:

text += page.extract\_text() or ''

return text

def extract\_text\_from\_html(file\_path):

with open(file\_path, 'r') as f:

soup = BeautifulSoup(f, 'html.parser')

return soup.get\_text()

def extract\_text\_from\_docx(file\_path):

doc = Document(file\_path)

text = "\n".join([para.text for para in doc.paragraphs])

return text

def read\_file(file\_path):

if file\_path.endswith('.pdf'):

return extract\_text\_from\_pdf(file\_path)

elif file\_path.endswith('.html'):

return extract\_text\_from\_html(file\_path)

elif file\_path.endswith('.docx'):

return extract\_text\_from\_docx(file\_path)

elif file\_path.endswith('.txt'):

with open(file\_path, 'r') as f:

return f.read()

else:

raise ValueError("NOT PROPER FORMAT OF FILE")

def huffman\_encoding(input\_file, output\_file):

text = read\_file(input\_file)

if not text.strip():

raise ValueError("EMPTY FILE")

huffman\_tree = build\_huffman\_tree(text)

codebook = generate\_codes(huffman\_tree)

encode\_file(input\_file, output\_file, codebook)

compression\_ratio = calculate\_compression\_ratio(input\_file, output\_file)

return codebook, compression\_ratio

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

input\_file = 'C:/Users/ASUS/Desktop/Coding/DAA/DAA LAB5/file5.xls'

output\_file = 'output.huff'

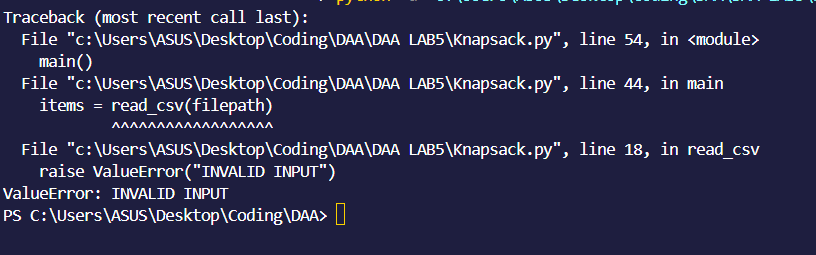
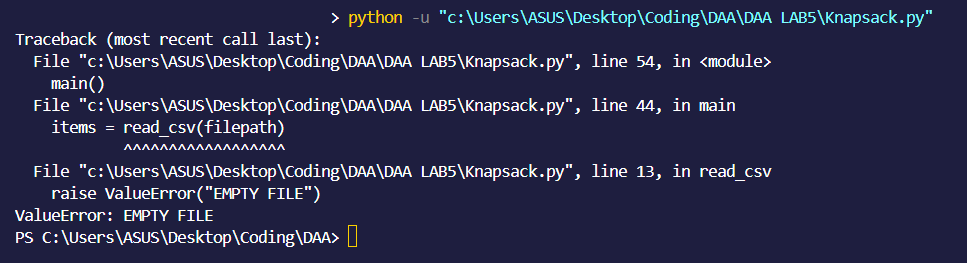
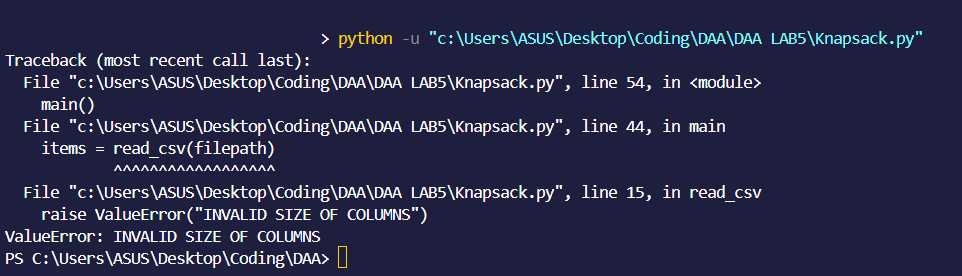
codebook, compression\_ratio = huffman\_encoding(input\_file, output\_file)

print("Huffman Codes for all characters:", codebook)

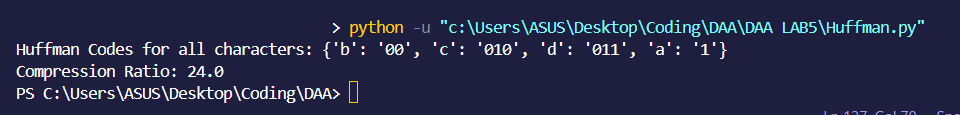
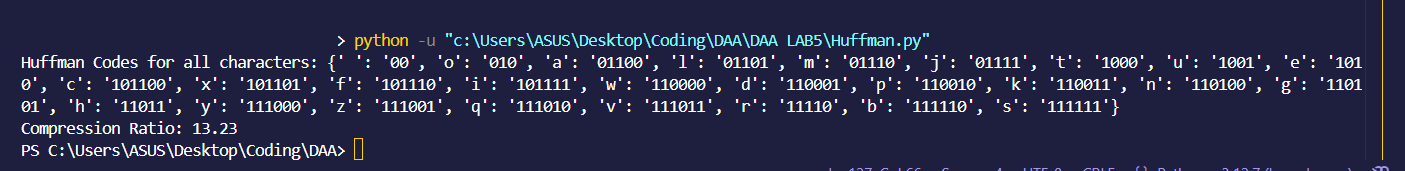
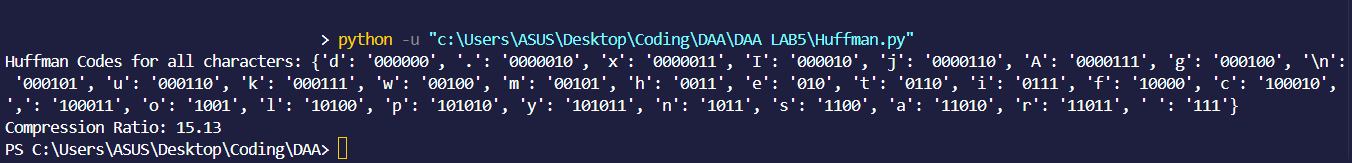
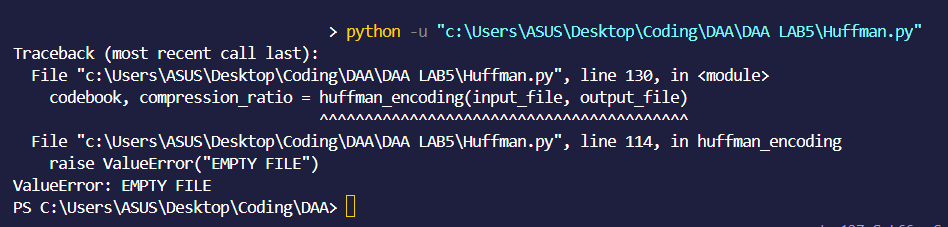
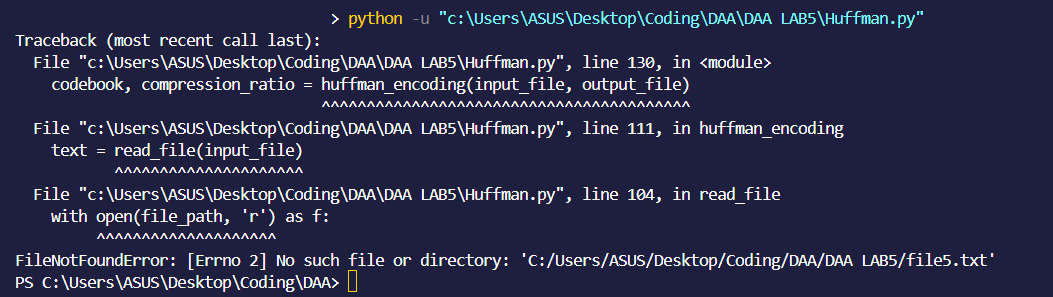
print("Compression Ratio:", round(compression\_ratio, 2))

TESTCASES IN CODE :-

FRACTIONAL KNAPSACK:

1. 
2. 
3. 
4. 
5. 

HUFFMAN ENCODING:

1. 
2. 
3. 
4. 
5. 

CONCLUSION:

So in this experiment we have Written Pseudo Code Algorithms for Fractional Knapsack problem and Huffman Encoding problem with Inputs and Outputs given outside of the algorithms.

We have also written 5 Test Cases each for fractional knapsack and huffman encoding where half of them are Positive and half of them are Negative. The testcases are written by giving all the inputs and then showing the ideal output.

Then we wrote a program following the python pep8 style guide such that the ideal outputs were obtained in the program from the testcases.

We then executed the Program to give the Outputs for the given test cases.

We have also found out the Time Complexity of Fractional Knapsack and Huffman Encoding for their Brute force and Greedy Approaches with their Mathematical Analyses.