Name: Ahaan Desai
Roll no.: 231070016
Batch: Batch A (1-20)

DAA Lab Experiment 5

Aim

1. To implement the fractional knapsack algorithm for goods considering weight, value and shelf life.
2. To compress text, html, pdf, doc using huffman coding algorithm.

Program

1. Fractional Knapsack

```python
import pandas as pd

class Item:
    def __init__(self, w, v, l):
        self.weight = w
        self.value = v
        self.shelf_life = l

    def __lt__(self, other):
        return (
            (self.value / (self.weight * self.shelf_life))
            < (other.value / (other.weight * other.shelf_life))
        )

class Reader:
"""Class to read csv file of items with value, weight and shelf life."""
    def __init__(self, file):
        self.df = pd.read_csv(file)
        self.weights = self.df['Weight'].to_list()
        self.values = self.df['Value'].to_list()
```

```
        self.shelf_lifes = self.df['Shelf Life'].to_list()

    def get_items_array(self):
       """Converting items in dataframe to an array of tuples."""
        items = []
        for w, v, l in zip(self.weights,  self.values, self.shelf_lifes):
            items.append(Item(w, v, l))

        return items

def fractional_knapsack(items, W):
       """Implementation of knapsack algorithm to find maximum profit among
items."""
    items.sort(reverse = True)
    total_value = 0
    for item in items:
        if item.weight <= W:
            W -= item.weight
            total_value += item.value
        else:
            fraction = W / item.weight
            total_value += item.value * fraction
            break

    return round(total_value, 2)

def process_files(file):
    reader = Reader(file)
    items = reader.get_items_array()
    W = 200
    total_value = fractional_knapsack(items, W)
    print(f"Total value for file {file}: {total_value}")

if __name__ == '__main__':
    for i in range(1, 6):
        process_files(f"Knapsack Data/couriergoods{i}.csv")
```

2. Huffman Coding

Python

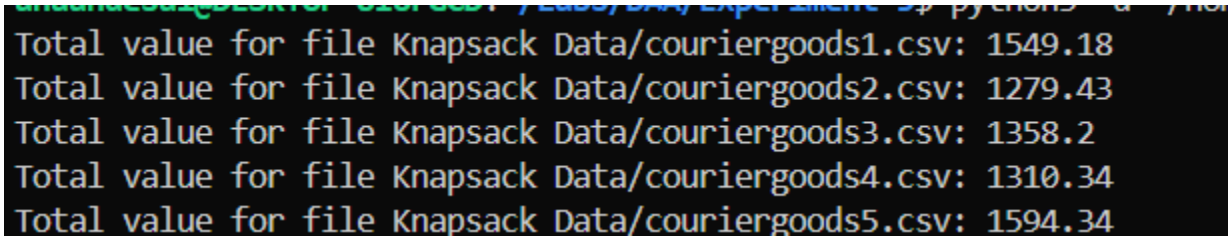
```
import heapq
from collections import Counter
from PyPDF2 import PdfReader


class Node:
    def __init__(self, freq, symbol, left=None, right=None):
        """
        Initialize a node in the Huffman tree.
        """
        self.freq = freq
        self.symbol = symbol
        self.left = left
        self.right = right
        self.huff = ''

    def __lt__(self, nxt):
        """
        Compare two nodes based on frequency.
        """
        return self.freq < nxt.freq


class HuffmanTree:
    def __init__(self, data):
        self.root = None
        self.data = data
        self.mapping = {}

        self.frequencies = Counter(data)
        self.huffman_encode(list(self.frequencies.keys()),
list(self.frequencies.values()))
        self.get_codes()

    def huffman_encode(self, chars, freq):
        """
        Build the Huffman tree based on the character frequencies.
        """
        nodes = [Node(freq[x], chars[x]) for x in range(len(chars))]
        heapq.heapify(nodes)
        while len(nodes) > 1:
```

```
            left = heapq.heappop(nodes)
            right = heapq.heappop(nodes)
            left.huff = 0
            right.huff = 1

            newNode = Node(left.freq + right.freq, left.symbol + right.symbol,
left, right)
            heapq.heappush(nodes, newNode)

        self.root = nodes[0]

    def get_codes(self, node=None, val=''):
        """
        Generate the Huffman codes for all characters.
        """
        if node is None:
            node = self.root
        newVal = val + str(node.huff)
        if node.left:
            self.get_codes(node.left, newVal)
        if node.right:
            self.get_codes(node.right, newVal)
        if not node.left and not node.right:
            self.mapping[node.symbol] = newVal

    def compress(self):
        """
        Compress the data using the generated Huffman codes.
        """
        res = ''
        for char in self.data:
            res += self.mapping[char]
        return res

    def decompress(self, string):
        """
        Decompress the given Huffman encoded string.
        """
        res = ''
        node = self.root
        for bit in string:
            if bit == '0':
                node = node.left
            else:
```

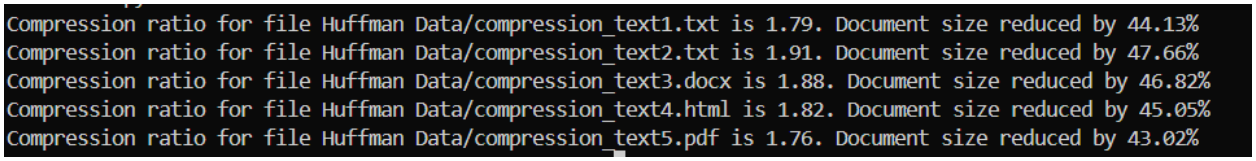
```
                node = node.right

            if not node.left and not node.right:
                res += node.symbol
                node = self.root

        return res

def compress_file(file):
    """
    Compress the contents of a given file using Huffman encoding.
    """
    filename = file
    extension = file.split('.')[-1]
    if extension == 'pdf':
        with open(file, 'rb') as file:
            reader = PdfReader(file)
            data = ''
            for page in reader.pages:
                data += page.extract_text()
    else:
        with open(file, 'r') as file:
            data = file.read()

    huffman_generator = HuffmanTree(data)
    original_data_length = len(data) * 8
    encoded_data = huffman_generator.compress()
    encoded_data_length = len(encoded_data)
    decoded_data = huffman_generator.decompress(encoded_data)

    assert data == decoded_data

    print(f"Compression ratio for file {filename} is
{round(original_data_length / encoded_data_length, 2)}. "
          f"Document size reduced by {round((1 - encoded_data_length /
original_data_length) * 100, 2)}%")


if __name__ == "__main__":
    compress_file("Huffman Data/compression_text1.txt")
    compress_file("Huffman Data/compression_text2.txt")
    compress_file("Huffman Data/compression_text3.docx")
    compress_file("Huffman Data/compression_text4.html")
    compress_file("Huffman Data/compression_text5.pdf")
```

Output

1. Fractional Knapsack



2. Huffman Coding



Conclusion

1. We have implemented the fractional knapsack algorithm for goods, while considering their value, shelf life and weight. Goods with higher value , lower shelf life and lower weight are given higher priority.

2. We have compressed documents of different file types using Huffman Coding. We have achieved file size reduction of 43%-47% of original file size.