Algorithm Design – Programming Assignment

Objective

Implement Huffman coding compression and decompression in Python. Write two scripts: one for compression and another for decompression.

Guidelines

1. Use Python and feel free to leverage common libraries like Numpy, Scipy, Pandas, etc...
2. Do *not* use pre-existing functions that implement Huffman code or significant parts of it.
3. Ensure the code is *well-documented*, *readable*, and adheres to the *specified guidelines*.
4. You need to submit a zip file named `ID1_ID2.zip` that contains both of your codes (`ID1_ID2_decompression.py` & `ID1_ID2_compression.py`)

---

1. Compression Code

Write a Python script, named `ID1_ID2_compression.py`, to compress a given `txt` file using Huffman coding.

Input & Output

- The main function takes a `txt` file (e.g., `'Alice_in_wonderlands.txt'`) as input. You can use the provided file as a practice example. You can assume that the input text does not contain any digits.
- Outputs the compressed file, along with the decoded Huffman tree, into the script's file directory in a *single* `txt` file.

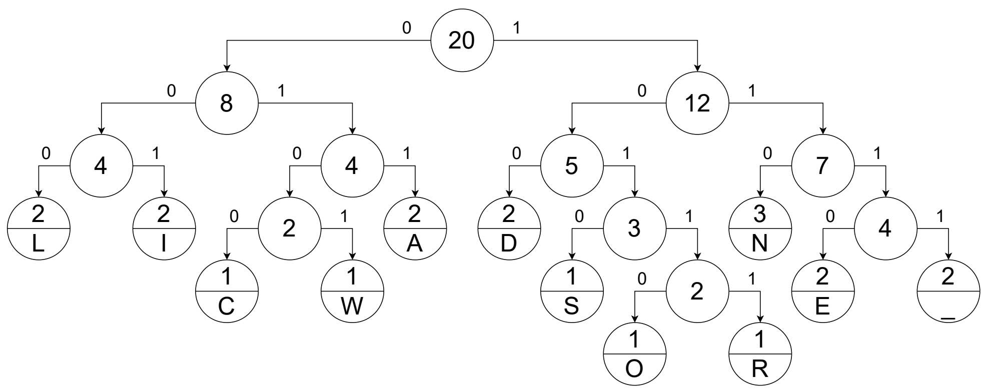
Note

Huffman code words are binary, and writing to a `txt` file is done using `'char'` type. Hence you would need to convert the binary sequence to groups of 8 bits such that each group corresponds to some `'char'`, for example, the sequence $\{0, 1, 0, 0, 0, 0, 0, 1\}$ corresponds to the 65-th ASCII character which is the char `'A'`. However, some sequences correspond to dubious characters such as `'DEL'`, that will remove the previous characters. These challenges are left for you to overcome. (e.g, you can use some other mapping schemes)

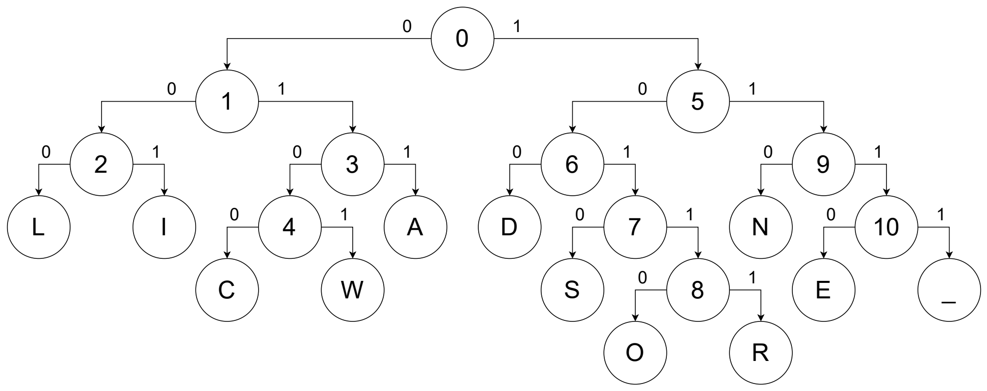
Decoding Huffman tree

In order to decompress, you are required to send the Huffman tree alongside the compressed file. To do so you need to add a preorder and inorder traversals of the Huffman tree into your compressed file.

For example, let us say this is our Huffman tree of the phrase `ALICE_IN_WONDERLANDS`



In order to decode this tree, you would first need to convert the 'frequency' nodes into unique valued nodes. (You can not decode a tree from preorder and inorder traversal if there are duplicate values). For example, you could map each such node as an integer like so:



Then the traversal will be

- Inorder= $\{L, 2, I, 1, C, 4, W, 3, A, 0, D, 6, S, 7, O, 8, R, 5, N, 9, E, 10, \_\}$
- Preorder= $\{0, 1, 2, L, I, 3, 4, C, W, A, 5, 6, D, 7, S, 8, O, R, 9, N, 10, E, \_\}$

To not confuse integers of the text, and integers of inner nodes, you can assume that the original file does not contain any integers. You are required to add those traversals into your compressed file, such that your compressed file will look like this:

```
ID1_ID2_compressed.txt
    Compressed sequence ⏎
    Inorder sequence ⏎
    Preorder sequence
```

⏎ is the newline symbol. Your compressed sequence could be multiple lines, but it's left for you to overcome.

Running Example

```
> python ID1_ID2_compression.py Alice_in_wonderland.txt
```

Creates a file named `ID1_ID2_compressed.txt` containing the compressed text file alongside the decoded Huffman tree, in the script's file directory.

2. Decompression Code

Write a Python script, named `ID1_ID2_decompression.py`, to decompress a previously compressed `txt` file. You would rebuild the Huffman tree from the traversal and then be able to decompress correctly the rest of the compressed file.

Input & Output

- Takes the compressed `txt` file (`ID1_ID2_compressed.txt`) as input.
- Creates a file named `ID1_ID2_decompressed.txt` that holds the decompressed text, matching the original file (i.e. in our example `ID1_ID2_decompressed.txt == Alice_in_wonderland.txt`).

Running Example

```
> python ID1_ID2_decompression.py ID1_ID2_compressed.txt
```

Creates a file named `ID1_ID2_decompressed.txt` containing the decompressed text, in the script's file directory.

Tip

In the compression file, to make the problem easier. You could start by sending out the compressed text as usual, but instead of the decoded Huffman tree, you can send the codeword of each character. Once you can successfully decompress the file, you will send the decoded Huffman tree, and not the codeword of each character.

This way, you would know where is your error. If you do things simultaneously, it would be difficult to know if your Huffman tree decoding/encoding scheme is incorrect or your translation of bits into character (or both).