Lww Algorithm

Implementation

This Python code defines a class called LzwAlgorithm that implements the lzw compression algorithm. The class has three methods:

- `encode` method takes a string text as input and returns a tuple containing a list of integers and a dictionary. The list of integers represents the compressed text, and the dictionary maps the substrings to their corresponding integers.
- `decode` method takes a list of integers lst and a dictionary dictionary as input and returns the decoded string.

```
from typing import Tuple, List, Dict

class LzwAlgorithm():
    """
    LzAlgorithms class
    """
    @staticmethod
    def encode(text: str) -> Tuple[List[int], Dict[str, int]]:
        """
        Encodes the input string `text` using the Lempel-Ziv-Welch 
algorithm.

        Args:
            text: str - The input string to be encoded.

        Returns:
            Tuple[List[int], Dict[str:int]]: A tuple containing the 
list of
        integers that represents the compressed `text`, and a 
dictionary
        that maps the substrings to their corresponding integers.

        Example:
        >>> lzw_encode('ABBABABBAABABA')
        ([0, 1, 1, 2, 2, 4, 5, 4], \
    {'A': 0, 'B': 1, 'AB': 2, 'BB': 3, 'BA': 4, 'ABA': 5, 'ABB': 6, 
'BAA': 7, 'ABAB': 8})
        """
        dictionary = dict()
        idx = 0
        result = []
        for letter in sorted(text):
            if letter not in dictionary:
                dictionary[letter] = idx
                idx += 1
        founded_letters = text[0]
        for i in range(1, len(text)):
            l = founded_letters + text[i]
            if l in dictionary:
                founded_letters = l
            else:
                result.append(dictionary[founded_letters])
                founded_letters = text[i]
                dictionary[l] = idx
                idx += 1
        if founded_letters:
            result.append(dictionary[founded_letters])
        return result, dictionary

    @staticmethod
    def decode(lst: List[int], dictionary: Dict[str, int]) -> str:
        """
        Decodes the compressed sequence of integers
        `lst` using the dictionary `d` generated by
        the Lempel-Ziv-Welch algorithm.

        Args:
            lst: List[int] - The list of integers to
            be decoded.
            dictionary: Dict[str:int] - The dictionary generated
            by the Lempel-Ziv-Welch algorithm.

        Returns:
            str - The decoded string.

        Example:
        >>> lzw_decode([65, 66, 67, 68, 69], {'A': 65, 'B': 66, 'C':
67, 'D': 68, 'E': 69})
        'ABCDE'
        """
        result = ""
        for num in lst:
            result += list(dictionary.keys())
[list(dictionary.values()).index(num)]
        return result
```

How it is working

```
#Create text for encoding
text =
'abarfkoeflepfepkaijdiefjeopqjsndajndjahajhajdahajfhdjeokfeofepp'
#Get encoded list and dictionary from encode() function
encoded, dictionary = LzwAlgorithm.encode(text)
print(encoded)
#Compare initial text with decoded one
LzwAlgorithm.decode(encoded, dictionary) == text
```

```
[0, 1, 0, 14, 4, 8, 11, 3, 4, 9, 3, 12, 4, 26, 8, 0, 6, 7, 2, 6, 23,
7, 3, 11, 12, 13, 7, 15, 10, 2, 0, 7, 44, 7, 0, 5, 46, 51, 33, 50, 46,
4, 5, 2, 37, 11, 8, 28, 11, 28, 12, 12]
```

```
True
```

Testing

```
import sys

def read_data(file) -> str:
    """
    Read data from file function

    Args:
        file: str - Path to input file
    Returns:
        str - read data from file in string format
    """
    with open(file, 'r', encoding='utf-8') as f:
        return "\n".join([i.strip() for i in f.readlines()])

def get_lzw_compressed_bytes(compressed_data):
    """
    Compress data using LZW algorithm and return the compressed data
as bytes.

    Args:
        compressed_data: list - List of integers representing the
compressed data.

    Returns:
        compressed_bytes: bytes - Compressed data as bytes.
    """
    output_bytes = bytearray()

    # Convert the list of integers to bytes
    for code in compressed_data:
        output_bytes.extend(code.to_bytes(2, byteorder='big'))

    return output_bytes

data = read_data('text_files/small_text.txt')
encoded, dictionary = LzwAlgorithm.encode(data)
compressed = get_lzw_compressed_bytes(encoded)
comparison_persantage = round((1 -
sys.getsizeof(compressed)/sys.getsizeof(data))*100, 2)
print(f"file: {sys.getsizeof(data)/1000000}mb")
print(f"compressed: {sys.getsizeof(compressed)/1000000}mb")
print(f"comparison persantage: {comparison_persantage}%")
```

```
file: 0.002384mb
compressed: 0.00147mb
comparison persantage: 38.34%
```

```python
data = read_data('text_files/middlesize_text.txt')
encoded, dictionary = LzwAlgorithm.encode(data)
compressed = get_lzw_compressed_bytes(encoded)
comparison_persantage = round((1 -
sys.getsizeof(compressed)/sys.getsizeof(data))*100, 2)
print(f"file: {sys.getsizeof(data)/1000000}mb")
print(f"compressed: {sys.getsizeof(compressed)/1000000}mb")
print(f"comparison persantage: {comparison_persantage}%")
```

```
file: 0.013542mb
compressed: 0.006015mb
comparison persantage: 55.58%
```

```python
data = read_data('text_files/big_text.txt')
encoded, dictionary = LzwAlgorithm.encode(data)
compressed = get_lzw_compressed_bytes(encoded)
comparison_persantage = round((1 -
sys.getsizeof(compressed)/sys.getsizeof(data))*100, 2)
print(f"file: {sys.getsizeof(data)/1000000}mb")
print(f"compressed: {sys.getsizeof(compressed)/1000000}mb")
print(f"comparison persantage: {comparison_persantage}%")
```

```
file: 0.06777mb
compressed: 0.027797mb
comparison persantage: 58.98%
```

```python
data = read_data('text_files/verybig.txt')
encoded, dictionary = LzwAlgorithm.encode(data)
compressed = get_lzw_compressed_bytes(encoded)
comparison_persantage = round((1 -
sys.getsizeof(compressed)/sys.getsizeof(data))*100, 2)
print(f"file: {sys.getsizeof(data)/1000000}mb")
print(f"compressed: {sys.getsizeof(compressed)/1000000}mb")
print(f"comparison persantage: {comparison_persantage}%")
```

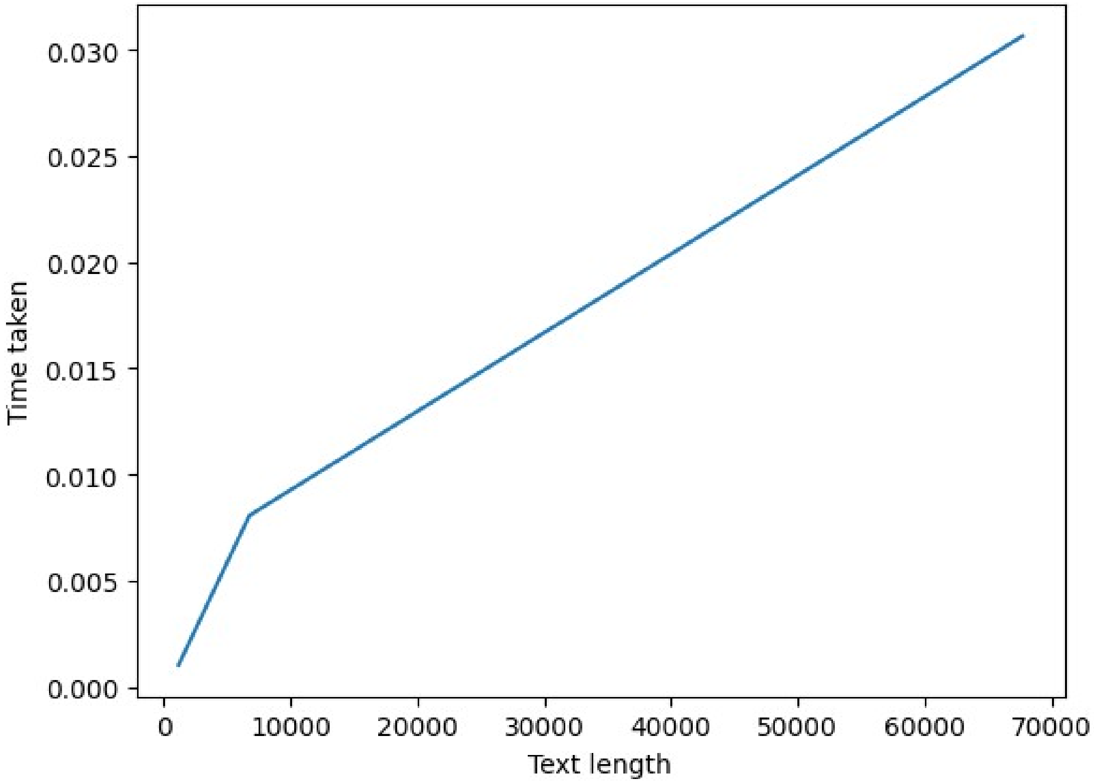
```
file: 16.321808mb
compressed: 0.128494mb
comparison persantage: 99.21%
```

```python
#import libraries for testing an algorithm
import matplotlib.pyplot as plt
import time
```

```python
#List of file pathes which are containing text for testing
file_pathes = ['text_files/small_text.txt',
'text_files/middlesize_text.txt', 'text_files/big_text.txt']
```

```python
def test_lzw(files: List[str]):
    time_taken = []
    length = []
    for file_path in files:
        with open(file_path, 'r', encoding='utf-8') as file:
            data = file.read()
            length.append(len(data))
            start = time.time()
            LzwAlgorithm.encode(data)
            end = time.time()
            time_taken.append(end - start)
    plt.xlabel('Text length')
    plt.ylabel('Time taken')
    x = length
    y = time_taken
    plt.plot(x, y)

test_lzw(file_pathes)
```



Conclusion

The Lempel-Ziv-Welch (LZW) algorithm is a popular lossless data compression algorithm that is widely used in various applications, including image and video compression, file compression, and network protocols. It is particularly effective for compressing data with repetitive patterns, such as text files, DNA sequences, and images with regions of uniform color.

LZW algorithm works best on data with high redundancy, such as text files, because it can exploit the repeating patterns in the data to achieve high compression ratios.