

LZW Encoding-Decoding

Lempel-Ziv-Welch (LZW) is a universal lossless data compression algorithm. Your task is to implement both the encoding and decoding functions for LZW compression.

The LZW algorithm works by building a dictionary of sequences during compression. It starts with all single-character sequences in the dictionary (ASCII characters 0-255), then adds new sequences as it encounters them.

Encoding (LZW Compressor)

We treat the dictionary as a mapping `string -> code`.

1. Initialize the dictionary with all single-byte ASCII characters:

$$\text{dict}[\text{string}] = \text{code}, \quad \text{for } \text{code} = 0, 1, \dots, 255$$

2. Let `next_code` = 256.

3. Let current string `w` = "" (empty).

4. For each input character `c`:

- Let `wc` = `w + c`.
- If `wc` is in the dictionary, set `w` = `wc`.
- Otherwise:
 - Output the code `dict[w]`.
 - Add `wc` to the dictionary with code `next_code`, then increment `next_code` += 1.
 - Set `w` = `c`.

5. After the loop, if `w` is not empty, output `dict[w]`.

Decoding (LZW Decompressor)

We treat the dictionary as a mapping `code -> string`.

1. Initialize the dictionary with all single-byte ASCII characters:

$$\text{dict}[\text{code}] = \text{string}, \quad \text{for } \text{code} = 0, 1, \dots, 255$$

2. Let `next_code` = 256.

3. Read the first code `k`. Let `w` = `dict[k]`. Output `w`.

4. For each subsequent code `k`:

- If `k` exists in the dictionary, let `entry` = `dict[k]`.
- Otherwise, set `entry` = `w + w[0]` (previous string + its first character).
- Output `entry`.
- Add a new dictionary entry:

$$\text{dict}[\text{next_code}] = w + \text{entry}[0]$$

- Increment `next_code` += 1.
- Set `w` = `entry`.

Input Format

For encoding: A string containing ASCII characters.

For decoding: A list of integers representing the encoded output.

Output Format

For encoding: A list of integers representing the compressed codes.

For decoding: The original decompressed string.

Constraints

- $1 \leq \text{length of input string} \leq 5 \times 10^4$
- Input string contains only ASCII characters (codes 0-255)
- Dictionary codes start from 256 onwards for new sequences

Sample Input

```
"TOBEORNOTTOBEORTOBEORNOT"
```

Sample Output

```
Encoded: [84, 79, 66, 69, 79, 82, 78, 79, 84, 256, 258, 260, 265, 259, 261, 263]
Decoded: TOBEORNOTTOBEORTOBEORNOT
Original length: 24
Encoded length: 16
```

Implementation

Goal: Implement the following two functions:

```
def lzw_encode(input_string: str) -> list[int]:
    dictionary = {chr(i): i for i in range(256)}
    next_code = 256
    return ...

def lzw_decode(encoded_input: list[int]) -> str:
    dictionary = {i: chr(i) for i in range(256)}
    next_code = 256
    return ...

if __name__ == "__main__":
    original_string = "TOBEORNOTTOBEORTOBEORNOT"
    encoded = lzw_encode(original_string)
    print("Encoded:", encoded)
    decoded = lzw_decode(encoded)
    print("Decoded:", decoded)
    assert original_string == decoded
    print("Original length:", len(original_string))
    print("Encoded length:", len(encoded))
```

Scoring

The grading for this problem is split between the two functions:

- **Encoding (60%):** If your `lzw_encode` function produces the correct output, you will receive 60% of the points for each test case.
- **Decoding (40%):** If your `lzw_decode` function produces the correct output, you will receive 40% of the points for each test case.

Warnings

- You are **NOT ALLOWED** to use any external compression libraries
- You must implement the LZW algorithm from scratch
- Your implementation should handle standard ASCII characters (codes 0-255)

13 Nov 2025:

THE ENTIRE INTERNET

