Facultatea de
Matematică şi Informatică
Universitatea din Bucureşti

FLZW: Implementare LZW pentru compresia datelor

Mușat Fabian

Ianuarie 2025

Cuprins

1 Introducere

Algoritmul LZW (Lempel-Ziv-Welch) este un algoritm de comprimare a datelor fără pierderi, utilizat pe scară largă datorită eficienței și simplității sale. A fost dezvoltat în 1984 de Terry Welch ca o îmbunătățire a algoritmului LZ78. Acesta stă la baza multor formate de fișiere populare, cum ar fi GIF și TIFF, și este utilizat în diverse aplicații informatice.

2 Descrierea algoritmului

LZW este un algoritm adaptiv ce construiește un dicționar dinamic în timpul procesului de comprimare. Conceptul acesta de a construi un dicționar dinamic revine din ideile algoritmului LZ78 si dovezile eficienței acestuia. Inițial in scenariul descris in 1984 de către Welch, LZW era folosit pentru a coda date text reprezentate ASCII deci pe 8 biți in coduri de lungime fixa pe 12 biți. Lucrul cu coduri de lungime fixa poate fi eficient in cazuri specifice, dar de cele mai multe ori creșterea numărului de biți de la început nu este o idee atât de bună. Astfel, o idee imediata este să începi cu un număr minim de biți, si să-l crești după nevoie. O altă îmbunătățire ar fi să incluzi un cod special ce-ți indică nevoia de a reseta dicționarul.

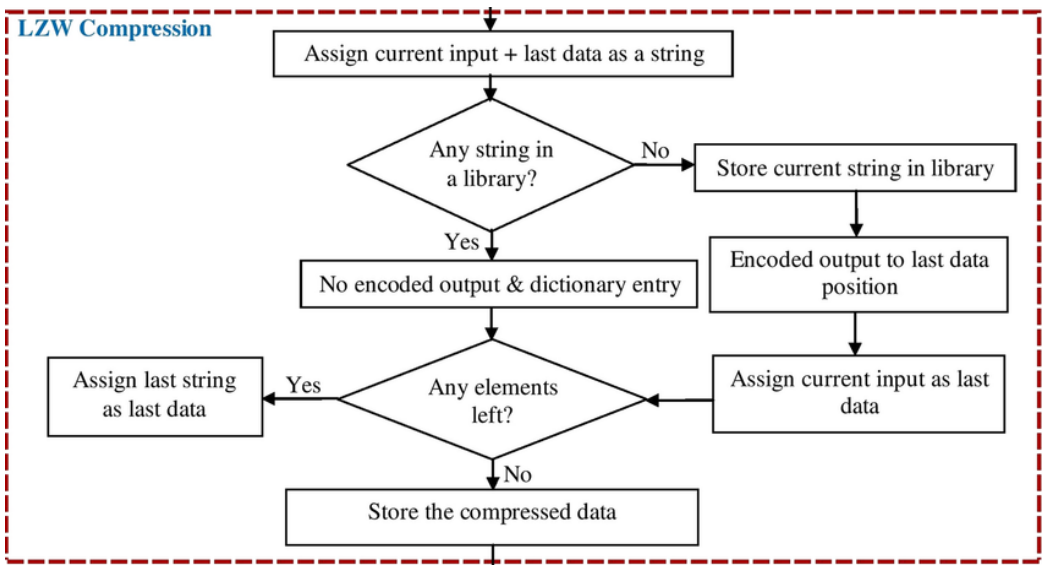

Figura 1: Flowchart pentru algoritmul LZW

2.1 Inițializare

Inițializarea depinde destul de mult de contextul in care este aplicat algoritmul. Astfel, un algoritm general ar putea inițializa dicționarul cu toate secvențele de lungime 1. Această inițializare este bună și pentru compresia datelor text, deoarece sunt reprezentate ASCII, iar dicționarul conține inițial toate secvențele de lungime 1 (bytes).

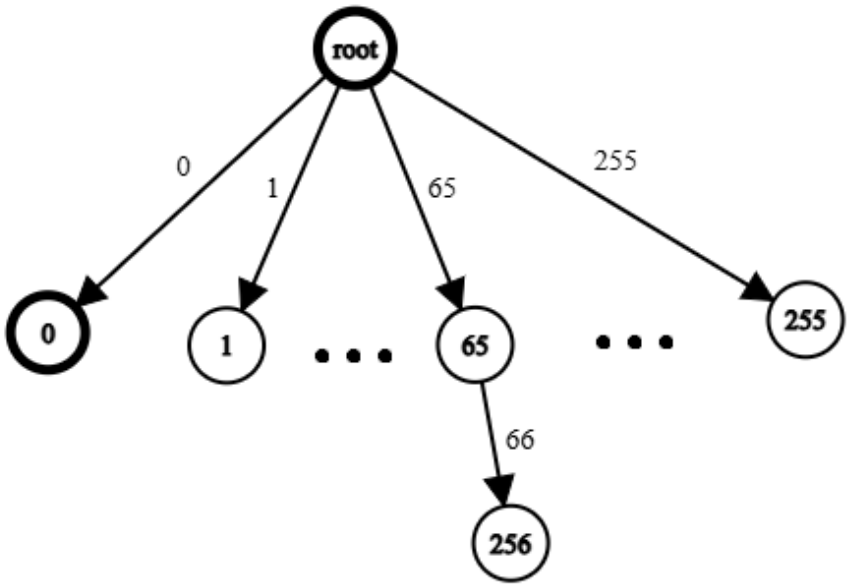

Figura 2: Dicționarul după câțiva pași

2.2 Encoding

Algoritmul de codificare descris pe scurt ar fi:

1. Se inițializează dicționarul cu secvențele de lungime 1.
2. Găsește cel mai lung șir **W** din dicționar ce se potrivește cu inputul curent.
3. Afișează indexul șirului **W** și continuă.
4. Adaugă **W** concatenat cu următorul simbol din input in dicționar.
5. Repetă începând cu pasul 2.

Cel mai greu pas in această etapă este cel în care trebuie sa găsești cel mai lung șir (2), deoarece trebuie făcut eficient. FLZW urmează pseudocodul descris in acest curs [3]. Această tehnică construiește șirul curent denumit *working* și menține și un alt șir *augmented* ceea ce simplifică pasul la doar a căuta unul din șiruri la un anumit pas. Pentru a găsi valoarea indexului asociat unui șir (dacă există) FLZW folosește o trie, organizată sub forma unui dicționar cu chei șiruri de bytes. Decizia am luat-o pentru că structura are o complexitate bună pentru a accesa valoarea indexului si anume O(n) unde n este lungimea șirului, deci O(1) considerând că majoritatea șirurilor sunt bounded. Un dicționar obișnuit poate fi implementat mai eficient, dar accesul are aceeași complexitate teoretica (puțin mai slabă, fiind O(1) amortizat).

2.3 Decoding

Procesul de decodificare folosește același dicționar și pentru a nu fi nevoie să fie transmis, îl creează in timpul rulării algoritmului.

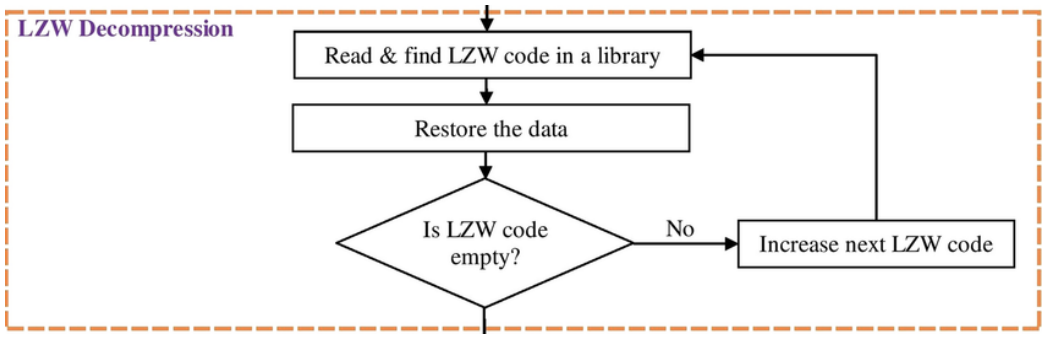

Figura 3: Flowchart pentru algoritmul LZW

1. Se inițializează dicționarul cu secvențele de lungime 1.
2. Verifică dacă simbolul curent se află în dicționar.
   (a) Da?
      i. Afișează secvența **W** pentru simbolul curent.
      ii. Concatenează secvența afișată anterior cu primul simbol din **W**.
   (b) Nu?
      i. Concatenează secvența afișată anterior cu primul simbol din ea, secvența obținută o notăm cu **V**.
      ii. Adaugă **V** în dicționar și afișează.
3. Repetă începând cu pasul 2.

FLZW folosește 2 structuri, o trie pe post de dicționar și un array pentru indexarea secvențelor după indexul dat, pentru a fi constant în timp.

3 Avantaje și dezavantaje

3.1 Avantaje

- **Implementare simplă.**
- Eficiență ridicată pentru datele cu redundanță mare și pentru datele cu tipare frecvente.
- Nu necesită stocarea dicționarului în prealabil.
- Eficiență nelimitată în teorie, deoarece cu un dicționar *infinit* algoritmul poate surprinde tipare de orice lungime.

3.2 Dezavantaje

- Performanță **redusă** pentru datele fără redundanță.
- Dimensiunea dicționarului poate crește rapid.

4 Aplicații

Algoritmul LZW este utilizat într-o varietate de domenii, incluzând:

- Compresia imaginilor: formatele GIF și TIFF.
- Compresia textului și a imaginilor din PDF-uri.
- Arhivarea fișierelor: unele utilitare de compresie folosesc LZW ca parte a algoritmului lor. Vechiul utiliar de pe unix *compress* folosește LZW, *ncompress* pe Linux.
- Transmisii de date: utilizat pentru optimizarea lățimii de bandă.

5 Exemplu practic

Considerăm următorul șir de intrare: `banana`.

5.1 Initializare

Dictionarul initial va fi: {0x00: 0, 0x01: 1, .. 0x41: 'A', .., 0xff: 255}. Codul urmator valabil este: 256, iar numarul de biti initial este 9. Notam sirul curent := *working*, caracterul curent := *cur* si urmatorul sir := *augmented*.

5.2 Encoding

- working = '', cur = b, augmented = 'b', 'b' se afla in dictionar.
- working = 'b', cur = a, augmented = 'ba', 'ba' nu se afla in dictionar, deci il adaugam cu indexul 256, iar urmatorul index va fi 257 etc. si afisam b.
- working = 'a', cur = n, augmented = 'an', 'an' nu se afla in dictionar, deci il adaugam cu indexul 257 si afisam a.
- working = 'n', cur = a, augmented = 'na', 'na' nu se afla in dictionar, deci il adaugam cu indexul 258 si afisam n.
- working = 'a', cur = n, augmented = 'an', 'an' este in dictionar.
- working = 'an', cur = a, augmented = 'ana', 'ana' nu este in dictionar, deci il adaugam cu indexul 259 si afisam 257.
- working = 'a', cur = '' (am ajuns la final), deci afisam ce a ramas a.

Deci sirul de bytes final va fi `ban[257]a`.

5.3 Decoding

Dictionarul initial va fi acelasi ca la initializare. Acum avem `ban[257]a`. Folosim tot working, cur, si augmented si verificam daca simbolul curent este in dictionar, daca da afisam secventa, daca nu construim dictionarul si afisam ce stim. Astfel:

- working = '', cur = b, augmented = 'b', exista in dictionar, afisam b.
- working = 'b', cur = a, augmented = 'ba', a exista deci afisam a, 'ba' nu exista deci il adaugam cu indexul 256.
- working = 'a', cur = n, augmented = 'an', n exista deci afisam n, 'an' nu exista deci il adaugam cu indexul 257.
- working = 'n', cur = 257, augmented = 'n257', dar exista in dictionar cu secventa an deci working = 'n', cur = a, augmented = 'na', afisam a, 'na' nu exista in dictionar deci il adaugam cu indexul 258.
- working = 'a', cur = n, augmented = 'an', afisam n, 'an' exista deci continuam.
- working = 'an', cur = a, augmented = 'ana', afisam a, 'ana' nu exista deci il adaugam cu indexul 259.

Deci sirul final va fi `banana`.

6 Implementare

Desi algoritmul suna extrem de usor de implementat, exista cateva parti ale algoritmului care trebuie implementate cu grija pentru a nu strica eficienta / complexitatea prea mult. FLZW foloseste cateva structuri esentiale:

1. Dictionarul - reprezentat de o Trie.
2. Secventele - stocate intr-un Array.
3. Bitstream - un stream de biti pentru lucrul usor la nivel de biti.

Implementarea are 2 parti: partea de teste automate, implementarea propriu-zisa. Partea de teste cuprinde cateva unit-tests ce verifica stabilitatea implementarii pentru: Trie, Bitstream, dar si pentru compresia si decompresia de date text de marimi diferite pana la 1.6 MiB.

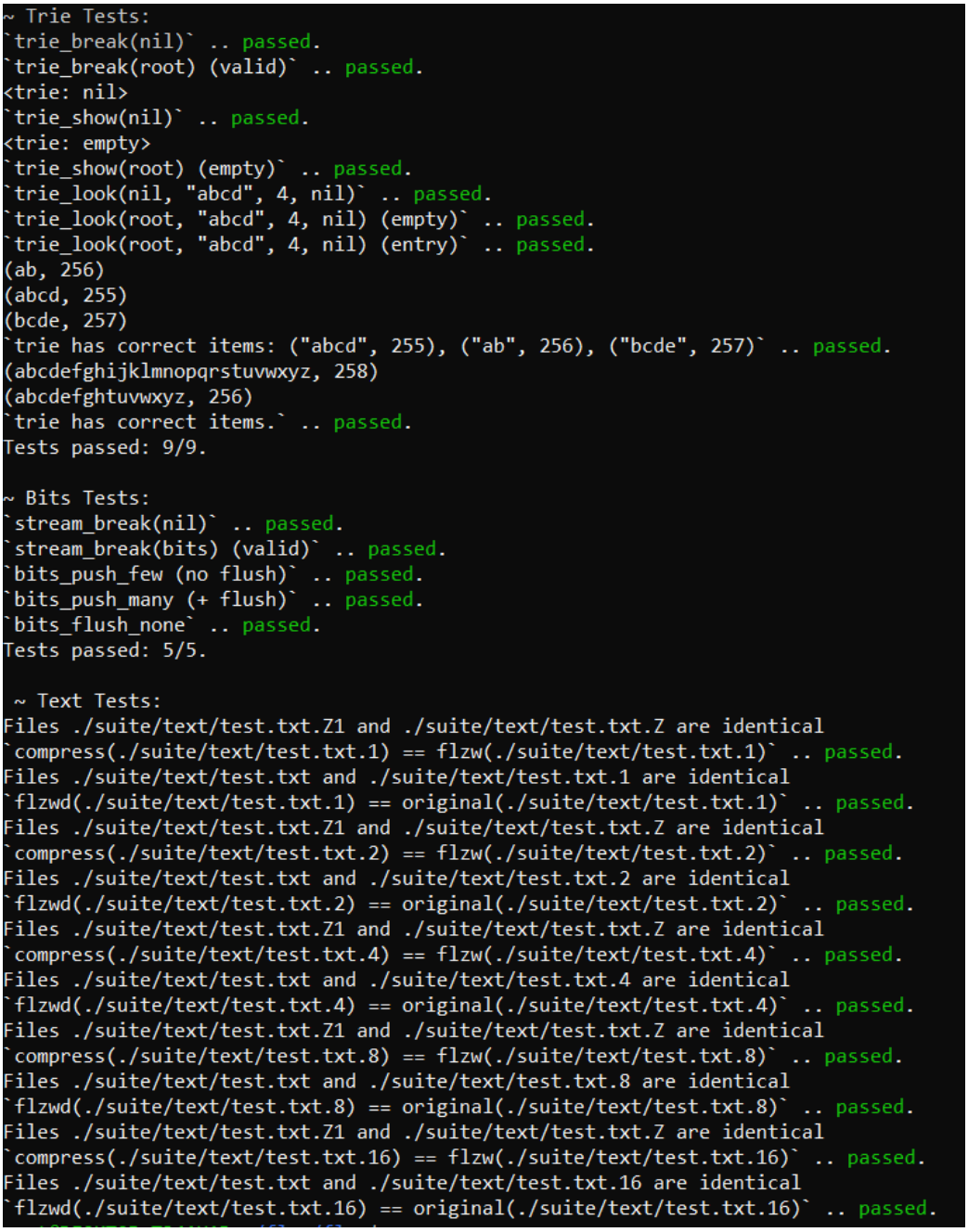

Figura 4: Rularea testelor

Implementarea este impartita in Trie, Bitstream si LZW, fiecare prezentand un API usor de folosit.

```
int flzw_decode(const byte *, const size_t, const flzw_config,
    ByteStream *);
int flzw_compress(const char *, const char *, const flzw_config)
    ;
int flzw_encode(const byte *, const size_t, const flzw_config);
int flzw_decompress(const char *, const char *);
void flzw_break(int);
void flzw_make(int);
```

Partea dificila la compresie consta in emiterea bitilor codificati, FLZW suporta formatul .Z specific unix, dar functiile decode si encode, functioneaza pe orice stream de biti LZW. Dupa ce bytes sunt codificati folosind LZW, deoarece codurile au lungimi variabile (9 $\rightarrow$ N) biti, acestea trebuie impartite cumva in bytes si stocate. Metoda standard este:

1. Inversezi bitii codurilor.
2. Imparti in grupuri de 8 biti.
3. Inversezi bitii in grupurile de 8 rezultate.
4. Acest sir de bytes este rezultatul final.

Acest proces este extrem de error-prone si este si partea dificila din decodificare, deoarece trebuie mai intai sa rulezi procesul invers pentru a obtine codurile codificate, iar apoi sa rulezi algoritmul LZW.

FLZW (de)comprima fisierele in chunk-uri de 16 MiB pentru a nu folosi prea multa memorie in timpul rularii algoritmului, acest lucru si implementarea nu cache-friendly a Trie-i face ca FLZW sa nu fie cel mai rapid, dar este extrem de simplu de folosit.

6.1 Benchmarks

Timpul de executie este decent, dar exista o diferenta intre timpii de compresie si cei de decompresie, din cauza nevoii de a transforma bitstream-ul in simboluri codate, iar apoi rularea propriu-zisa a algoritmului LZW.

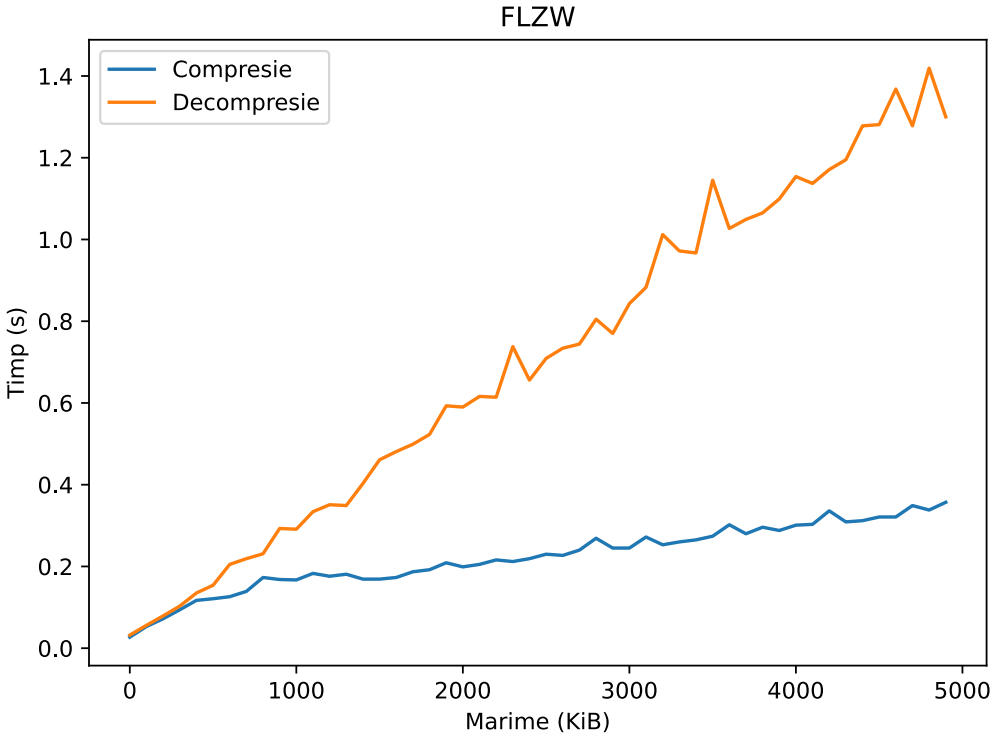

Figura 5: Timpii de rulare

Compression ratio-ul este extrem de bun pe date text, mai ales pe cele ce se repeta mult, insa si algoritmul este implementat cu acest scop, dictionarul este initializat astfel incat sa comprime cat mai bine datele text, de orice natura. Spre exemplu un fisier de 4.8 MiB comprimat cu FLZW are marimea de 1.1 MiB, iar comprimat cu GZip are marimea 1.21 MiB.

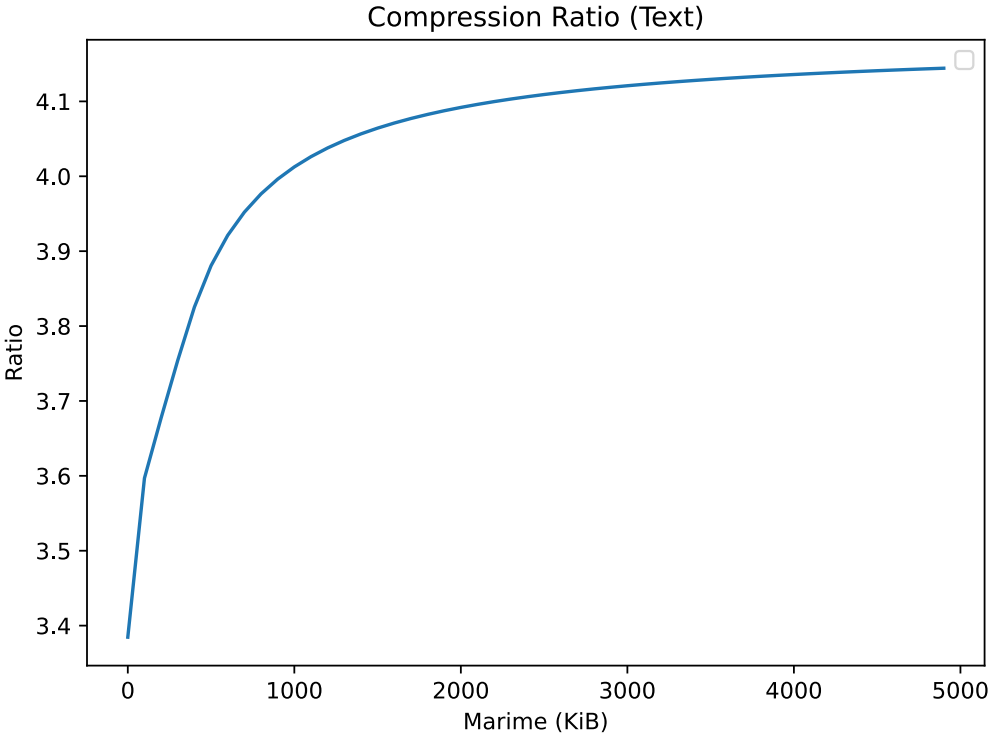

Figura 6: Compression ratio

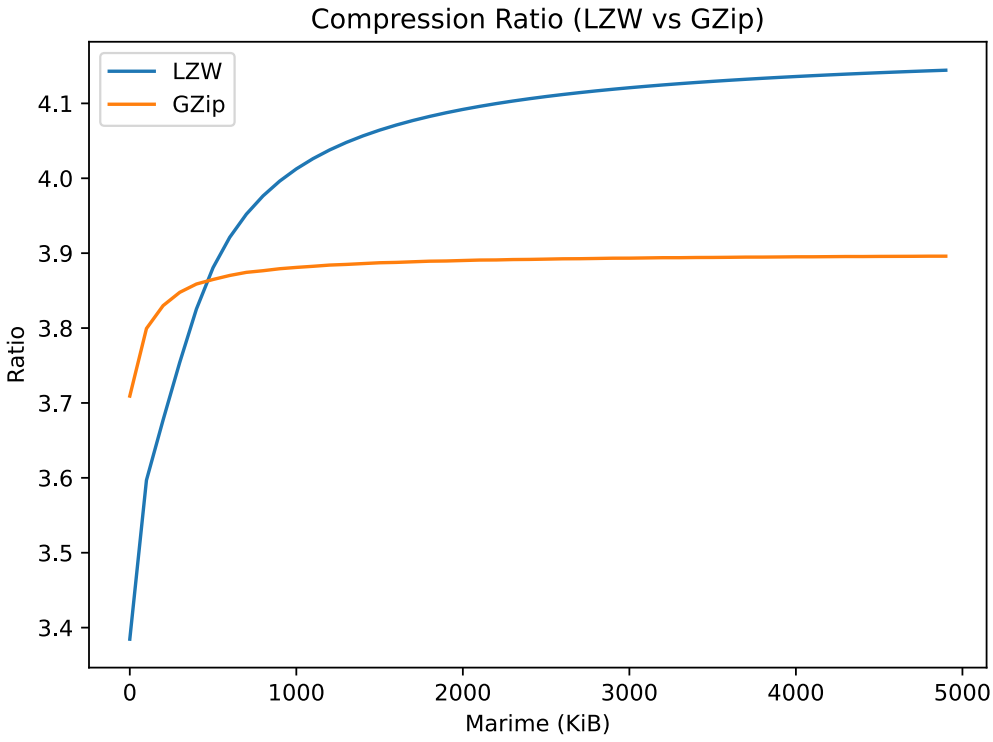

Figura 7: Comparatie cu GZip (DEFLATE)

7 Concluzii

Algoritmul LZW rămâne o metodă fundamentală de comprimare datorită adaptabilității și eficienței sale. Deși unele metode moderne îl depășesc în anumite cazuri, LZW este încă relevant în multe aplicații practice datorită simplității sale. Această proprietate permite implementări rapide, deci crește folosința in aplicații low-level, transmitere de date serial, prin rețea etc.

Bibliografie

[1] Terry A. Welch. *A Technique for High-Performance Data Compression*. IEEE Computer, 1984.

[2] Wikipedia. *Lempel–Ziv–Welch Algorithm*. Wikipedia, 2024.

[3] Bill Bird. *LZW Course*. Youtube, LZW.