Proiect Procesarea Semnalelor

ROCKZIP

Studenți: Kerberos – Cibotari Augustin, Licu Mihai-George,
Stanciu Alexandra-Andreea, Stătescu Relu
Grupa 462

Cuprins

Introducere..... 3

I. Istoric gzip & Deflate..... 3

II. Implementare.....4

1. Implementarea simplificată a Deflate.....4

1.1 Abordare generală.....5

1.2 Generarea body-ului folosind Deflate.....5

1.3 Manipulare simplificată a blocurilor..... 6

2. LZSS și Huffman coding cu package-merge..... 6

2.1 LZSS..... 7

2.2 Huffman coding cu package-merge..... 8

3. Structurile și conținutul blocurilor..... 8

3.1 Blocuri necomprimate (0b00)..... 9

3.2 Blocuri de coduri de prefixe personalizate (0b10)..... 10

III. Rezultate..... 12

1. Caragiale.txt - Volumul “Nuvele” Ion Luca Caragiale..... 12

2. Kennedy.xls - Spreadsheet Excel cu informatii economice..... 12

3. Samba.tar - Codul susrsa a Samba 2-2.3.....13

4. Shakespeare.txt - The Complete Works of William Shakespeare. 13

Concluzie..... 13

Bibliografie..... 14

Introducere

Compresia datelor reprezintă o provocare în sistemele moderne de comunicare și stocare digitală. Transmiterea eficientă a unor cantități mari de date prin rețele necesită minimizarea utilizării lățimii de bandă, în timp ce stocarea locală impune constrângeri stricte privind cerințele de spațiu. Abordarea acestor probleme necesită utilizarea algoritmilor de compresie a datelor, concepuți pentru a păstra informațiile relevante, eliminând în același timp redundanțele. Un exemplu proeminent de astfel de tehnici se regăsește în formatul GNU ZIP (gzip), care utilizează algoritmul loseless de compresie a datelor Deflate.

Această lucrare descrie un program de tip CLI, Rockzip, conceput pentru a imita și extinde funcționalitățile utilitarului tradițional gzip. Utilizând algoritmul Deflate, Rockzip îmbină algoritmul de compresie LZ77/LZSS cu Huffman coding, oferind o înțelegere practică și aprofundată a strategiilor de compresie a datelor.

Secțiunile următoare vor diseca arhitectura și funcționalitățile lui Rockzip, aprofundând specificul proceselor de compresie/decompresie, complexitatea algoritmului Deflate și fiabilitatea Huffman coding în cadrul utilitarului. Această investigație este completată de o analiză comparativă cu alte utilitare standard.

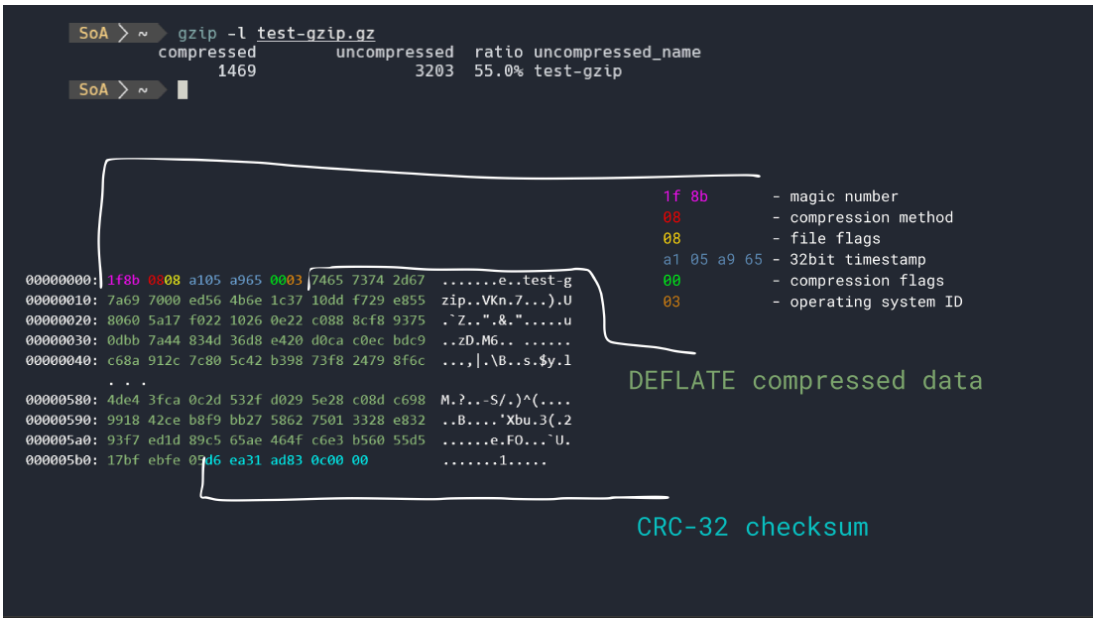
Această lucrare își propune să clarifice complexitatea și eficiența compresiei de date, folosind Rockzip ca mijloc de înțelegere a compresiei de date loseless.

I. Istoric gzip & Deflate

Gzip, un acronim pentru GNU zip, este un format de fișier și un program software recunoscut pe scară largă, utilizat pentru comprimarea și decomprimarea fișierelor. Gzip este bine cunoscut pentru nivelul său ridicat de eficacitate și eficiență. El face acest lucru prin utilizarea algoritmului Deflate, care combină algoritmul LZ77 (per standard, dar implementarea actuală folosită este un derivativ al acestuia, LZSS-Lempel–Ziv–Storer–Szymanski) cu prefix codes (teoretic, dar toate implementările actuale folosesc Huffman coding). Această combinație garantează un mecanism de compresie puternic, realizând un echilibru între rata de compresie și utilizarea resurselor de calcul.

Un fișier gzip este organizat metodic, cuprinzând un header, un body și un footer. Headerul, reprezentat printr-un identificator distinct, definește formatul fișierului și conține metadate, cum ar fi marcajele de timp și eventualele flag-uri suplimentare. Datele comprimate sunt încapsulate în body, demonstrând eficiența algoritmului Deflate. LZSS reduce redundanța prin înlocuirea cazurilor repetate de date cu indicatoare către o singură apariție a acestor date. Huffman coding îmbunătățește acest proces prin codificarea modelelor frecvențe folosind secvențe de biți mai scurte, reducând astfel dimensiunea totală.

Footerul conține verificările de integritate, care constau într-o sumă de control CRC32 și în lungimea datelor originale, necomprimate. Acest framework riguros garantează compresia datelor și protejează integritatea datelor în timpul procesului de decompresie.



II. Implementare

1. Implementarea simplificată a Deflate

Obiectivul nostru a fost de a proiecta o implementare Deflate simplă și ușor de înțeles. Pentru a atinge acest obiectiv, am luat câteva decizii diferite în comparație cu specificația Deflate tradițională.

1.1 Abordare generală

Ideea centrală din spatele implementării noastre constă în funcția Compress, care efectuează operațiile necesare pentru a pregăti datele de intrare pentru compresie. Procesul include încărcarea unui singur bloc întreg în memorie, scrierea header-urilor, generarea body-ului cu ajutorul DeflateEncoder și, în final, scrierea footer-ului.

Pentru a simplifica implementarea, am decis să nu despărțim datele de intrare în multiple blocuri. Deși este potențial risipitoare pentru inputuri mai mari, această abordare facilitează raționamentul asupra codului, reduce preocupările legate de citirea sincronă și buffering și permite o iterație rapidă în timpul fazelor de testare și de debugging.

1.2 Generarea body-ului folosind Deflate

Ne bazăm pe clasa DeflateEncoder din modulul utils.py pentru a genera body-ul comprimat. În Deflate, există trei tipuri de blocuri (patru, dar unul este rezervat):

- 0b00 (date necomprimate): Dacă lungimea datelor este mai mică de 128 de octeți, le stocăm ca un bloc necomprimat.
- 0b01 (Date comprimate cu coduri de prefix predefinite): Unele date sunt comprimate cu ajutorul unor coduri de prefix predefinite pe care orice decoder le poate genera pe cont propriu. În implementarea noastră, însă, sărim peste utilizarea codurilor de prefix predefinite și trecem direct la coduri de prefix personalizate.
- 0b10 (Date comprimate cu coduri de prefix): Caracterele și secvențele matchuite sunt comprimate folosind coduri de prefix personalizate generate cu ajutorul algoritmului package-merge aplicat datelor de intrare. Deoarece decoderele au nevoie să cunoască aceste coduri generate pentru a decoda, ele trebuie sa fie la randul lor comprimate printr-o alta serie de coduri numite CL codes și adaugate headerelor blocului.

Înainte de această fază, au fost deja scrise header-urile arhivei, folosind următoarea structură:

- Magic bytes: 0x1F 0x8B indică formatul gzip.

- Metoda de compresie: 0x08 indică algoritmul de compresie (în cazul nostru Deflate).
- Flag-uri: 0x00, deoarece nu folosim nicio caracteristică opțională.
- Timestamp: 0xTT 0xTT 0xTT 0xTT, ora curentă reprezentată ca timestamp Unix.
- Flag-uri suplimentare: În prezent neutilizate, setate la 0x00.
- Indicator de sistem de operare: Setat la 0x05, reprezentând Atari.

În ceea ce privește footer-ul, el este formatat astfel:

- CRC: O verificare de redundanță ciclică pe 32 de biți calculată cu ajutorul bibliotecii *binascii*. Această valoare confirmă validitatea datelor decomprimate.
- Lungimea datelor originale: Lungimea originală a fișierului înainte de compresie, reprezentată ca un 32-bit unsigned integer.

1.3 Manipulare simplificată a blocurilor

O diferență notabilă între implementarea noastră și specificația Deflate standard este gestionarea blocurilor. Implementarea noastră utilizează un singur bloc mare. Deflate standard împarte de obicei datele în mai multe blocuri și le procesează separat, în funcție de cea mai bună potrivire determinată de algoritm.

Cu toate acestea, abordarea noastră simplificată are unele implicații. Atunci când avem de-a face cu date de intrare lungi, crește probabilitatea de a găsi multe secvențe consecutive de potrivire, ceea ce duce la o eficiență redusă a compresiei din cauza capacității limitate de a diviza datele în unități mai mici. Cu toate acestea, utilizarea unui singur bloc menține implementarea noastră relativ simplă și ușor de înțeles.

2. LZSS și Huffman coding cu package-merge

În această secțiune, analizăm cei doi algoritmi esențiali utilizați în compresia Deflate: LZSS și Huffman coding cu package-merge. Acești algoritmi formează structura de bază a Deflate.

2.1 LZSS

LZSS (Lempel-Ziv-Storer-Szymanski) este un algoritm de compresie a datelor bazat pe dicționare care funcționează prin înlocuirea secvențelor recurente de date cu referințe la apariții anterioare. Algoritmul LZSS din Deflate utilizează două tipuri de “ferestre”: sliding window și lookahead buffer.

Sliding window conține datele procesate recent, în timp ce buffer-ul de așteptare conține datele care urmează să fie procesate. În timpul funcționării, algoritmul caută cea mai lungă secvență din cadrul “ferestrei glisante” care se potrivește cu începutul bufferului de așteptare. Odată găsită, acesta înlocuiește secvența corespunzătoare din bufferul de așteptare cu o referință la exemplul anterior, comprimând efectiv datele.

Implementarea noastră utilizează LZSS pentru a găsi secvențe recurente în cadrul datelor de intrare și pentru a efectua compresia de bază a datelor.

```
Total time: 4.06041 s
File: mumbojumbo.py
Function: find_longest_match at line 112

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   112                                           @profile
   113                                           def
find_longest_match(window, lookahead):
   114    124743      33104.1      0.3      0.8
longest_match_length = None
   115    124743      28198.2      0.2      0.7
longest_match_distance = None
   116    124743      60096.4      0.5      1.5      window =
bytes(window)
   117    124743      44541.6      0.4      1.1      lookahead =
bytes(lookahead)
   118
```

```
   119   3843578    2679730.0      0.7     66.0      while (index
:= window.rfind(lookahead)) == -1:
   120   3718835    1043728.7      0.3     25.7          lookahead
= lookahead[:-1]
   121
   122    124743      34191.8      0.3      0.8      if index !=
-1:
   123    124743      37070.6      0.3      0.9
longest_match_length = len(lookahead)
   124    124743      39892.1      0.3      1.0
longest_match_distance = len(window) - index
   125
   126    124743      59851.8      0.5      1.5      return
longest_match_length, longest_match_distance
```

2.2 Huffman coding cu package-merge

Huffman coding este o altă componentă esențială a compresiei Deflate. Funcția sa principală este de a atribui coduri de lungime variabilă diferitelor simboluri în funcție de frecvența lor în datele de intrare. Simbolurile mai puțin frecvente primesc coduri mai lungi, în timp ce simbolurilor mai frecvente li se atribuie coduri mai scurte, ceea ce duce la o scădere generală a numărului mediu de biți necesari pentru a reprezenta datele.

În Deflate, codificarea Huffman este combinată cu un algoritm package-merge care generează coduri de prefix personalizate pentru Literals and Lengths (LL) și Distance (D). Această abordare modificată îmbunătățește eficiența compresiei și reduce cantitatea de metadate necesare pentru a descrie codurile.

Algoritmul package-merge ia în considerare pachete de simboluri grupate în funcție de frecvențele lor. Începând cu pachetele cu frecvența cea mai mică, acesta fuzionează în mod repetat pachetele cu valori de frecvență apropiate până când se ajunge la numărul dorit de simboluri. Acest proces ajută la crearea unei distribuții echilibrate a frecvențelor simbolurilor, ceea ce duce la o compresie mai eficientă.

Deși implementarea noastră utilizează o abordare mai simplă de tratare a blocurilor, aceasta aplică în continuare codificarea Huffman cu package-merge pentru a crea coduri de prefix personalizate pentru datele comprimate.

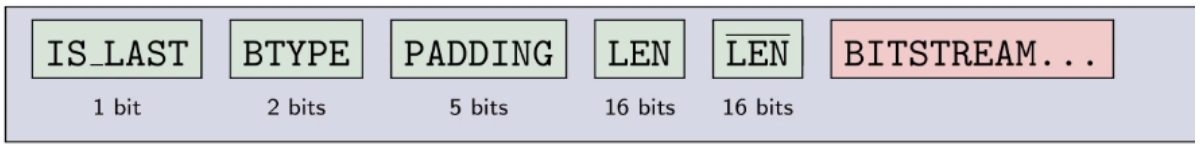
3. Structurile și conținutul blocurilor

În această secțiune, vom discuta structura și conținutul blocurilor din implementarea noastră simplificată Deflate, inclusiv blocurile necomprimate și blocurile de cod de prefixare personalizate. De amintit ca la inceputul fiecarui block exista un bit de control setat la 1 daca acest bloc este ultimul din secventa de deflate sau 0 in caz contrar, iar in urma fiecarui bloc se ataseaza simbolul special 256 care identifica finalul blocului.

3.1 Blocuri necomprimate (0b00)

Un bloc necomprimat conține date brute, necomprimate. În implementarea noastră, dacă lungimea datelor este mai mică de 128 de octeți, le stocăm ca bloc necomprimat.

În continuare este prezentată o descompunere a structurii blocurilor necomprimate:

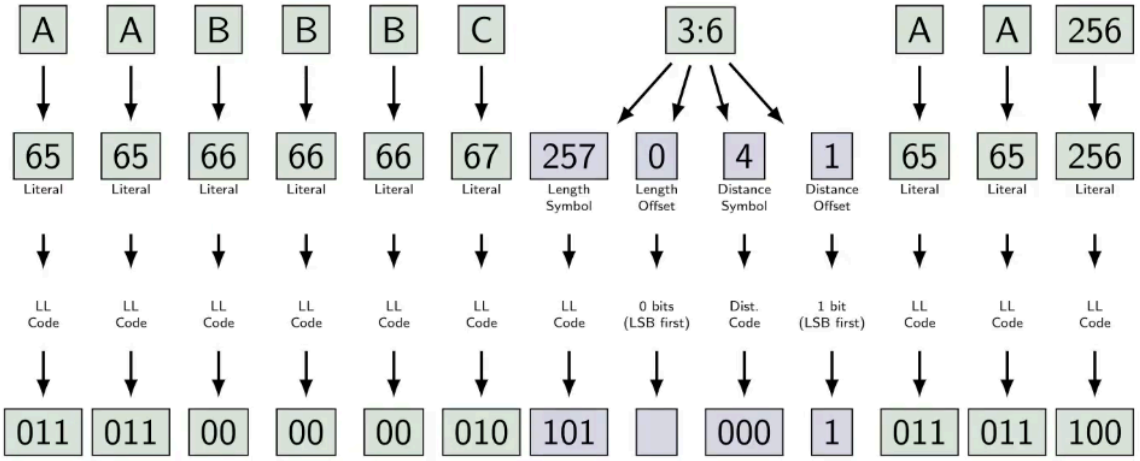


Block Type: Setat la 0b00, reprezentând blocurile de coduri necomprimate.

Padding (0b00000): Zerouri suplimentare sunt inserate după date pentru a ajunge la o limită de octeți. Acest lucru asigură că lungimea body-ului este un multiplu de 8 octeți, așa cum cere standardul Deflate.

Lungime (0xLL 0xLL): Lungimea blocului indică cantitatea de date care trebuie citită de către decompresor.

Lungime negată (NOT 0xLL 0xLL): O copie negativă a lungimii este adăugată la bloc, acționând ca o sumă de control pentru a se asigura că decompressorul interpretează corect lungimea datelor.

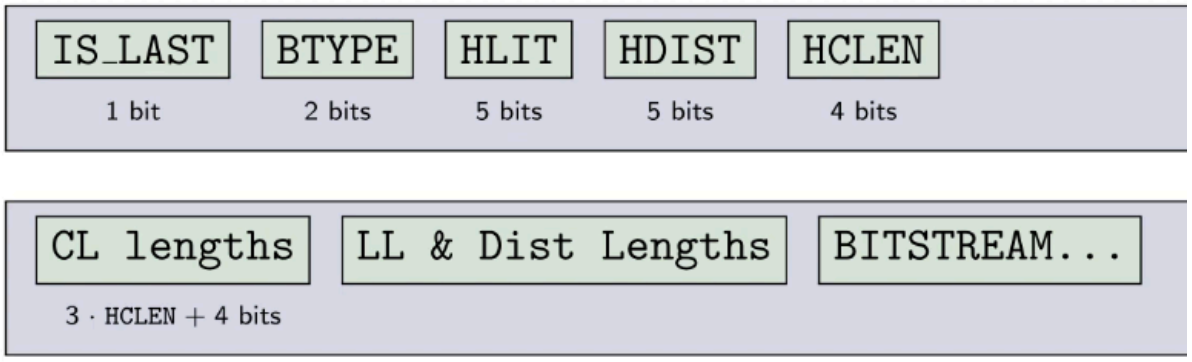


După scrierea datelor necomprimate ale blocului, aplicăm padding pentru a asigura o aliniere corectă.

3.2 Blocuri de coduri de prefixe personalizate (0b10)

Blocurile de cod de prefixare personalizate utilizează arbori Huffman personalizați generați din datele de intrare. În implementarea noastră, folosim un algoritmul package-merge pentru a crea acești arbori pentru simbolurile Literals and Lengths (LL) și Distance (D).

În continuare, analizăm mai îndeaproape structura blocurilor de cod prefix personalizat:



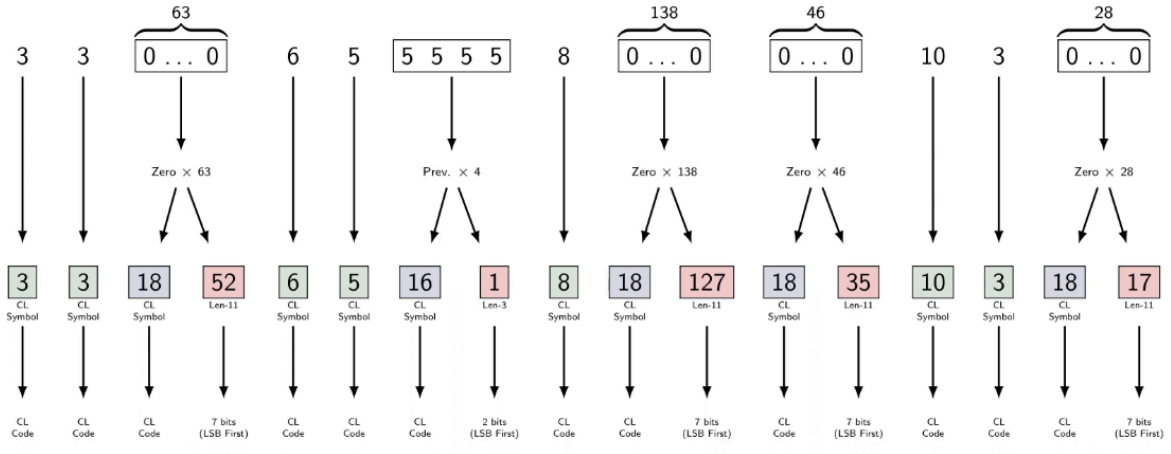
Tipul de bloc: Setat la 0b10, reprezentând blocurile de coduri de prefix personalizate.

HLIT, HDIST, HCLEN: valori pentru a seta cate coduri de Literals+Lengths, Distances si Code Lengths dorim sa folosim, aceste headere optimizează spațiul

necesar pentru a codifica simbolurile ce prezintă datele, ca un mecanism de RLE prin faptul că poate scapa de secvențe lungi de caractere similare.

CL lengths: sunt lungimile celor maxim 19 coduri ce dorim sa le folosim pentru Code Length encoding.

LL & Dist Lengths: sunt lungimile codurilor care descriu fluxul de date.

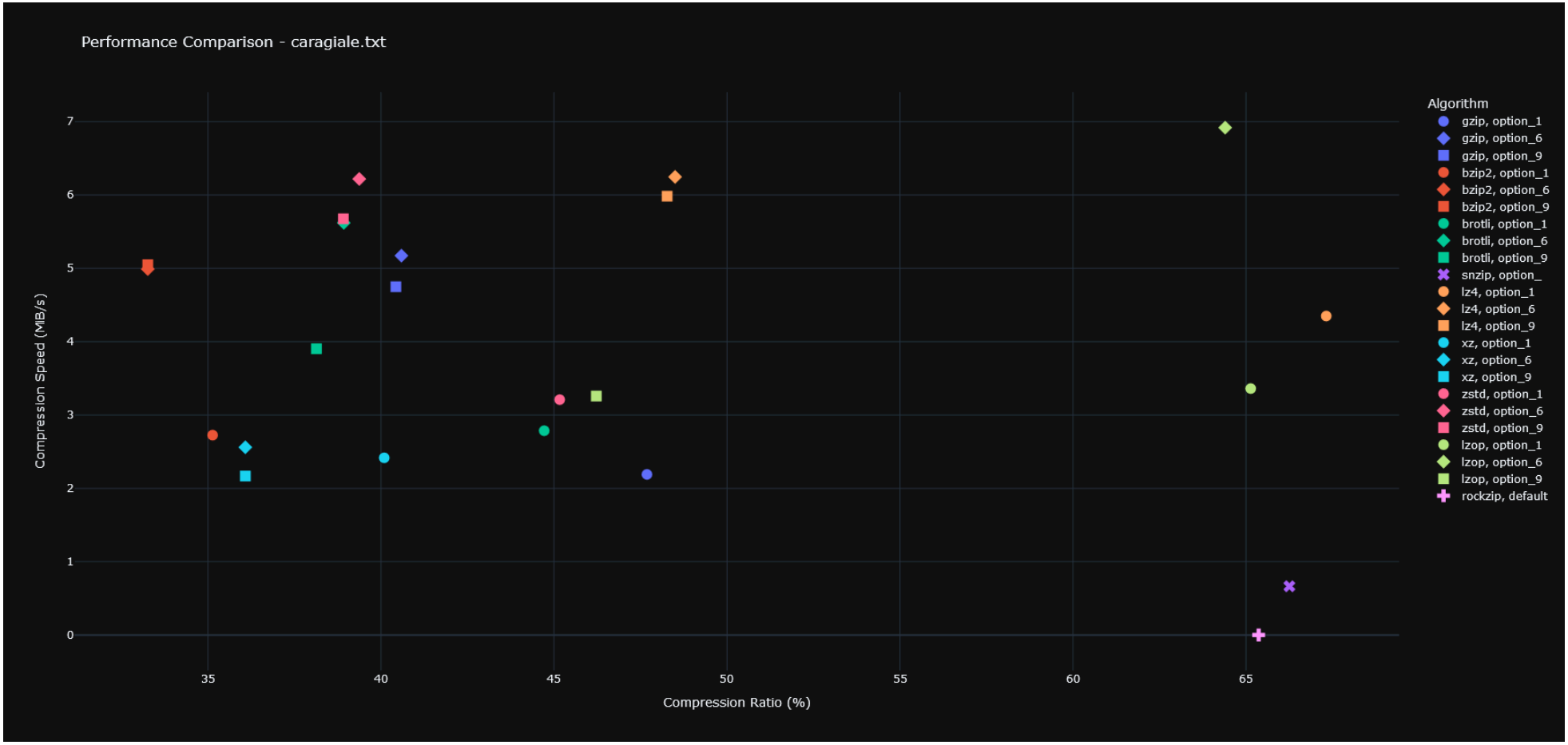


Generarea blocurilor de cod de prefix personalizate implică crearea de arbori Huffman utilizând algoritmul de package-merge și apoi codificarea secvențelor de intrare utilizând arborii personalizati rezultate din recalculare.

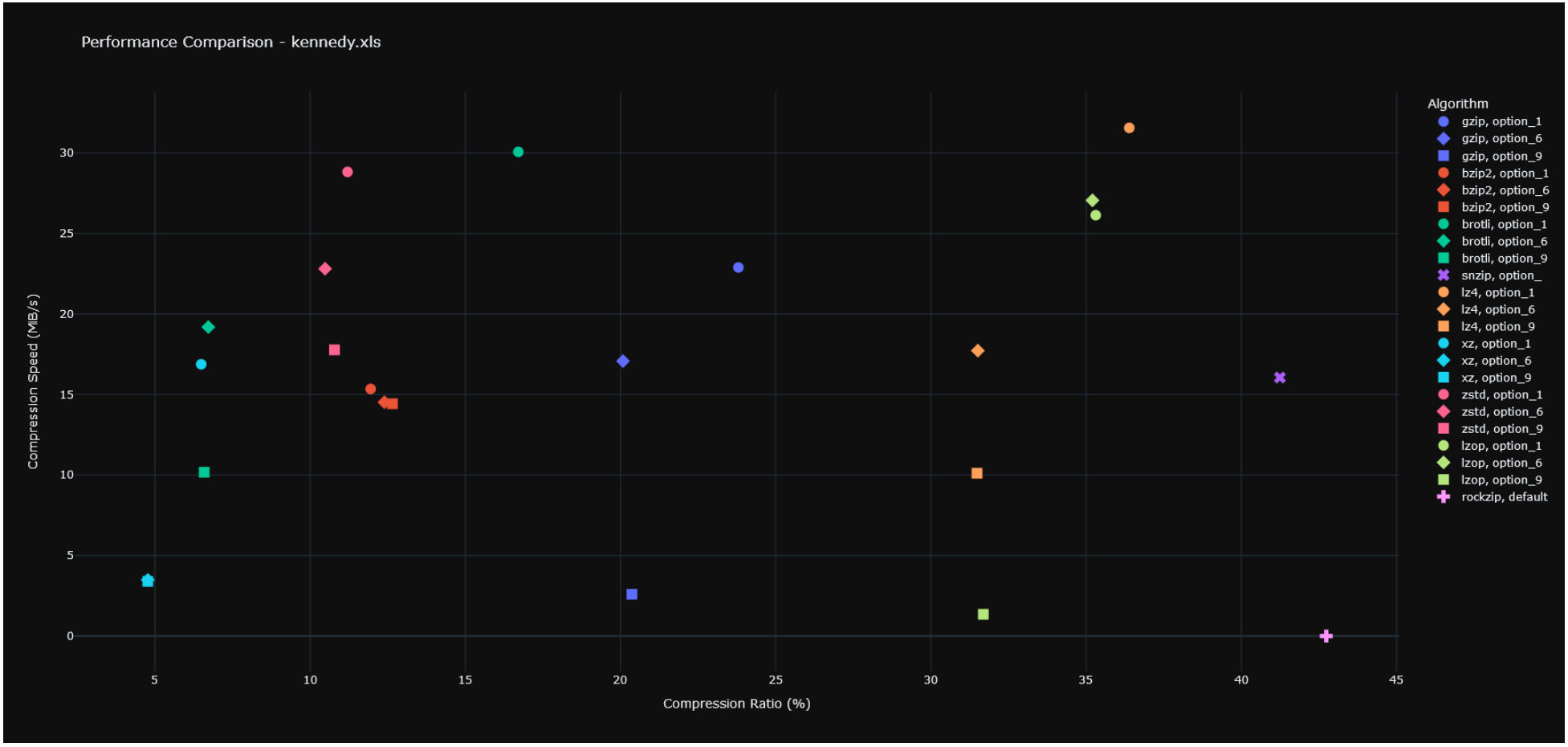
Reamintim că, după scrierea datelor blocului de cod prefix personalizat, aplicăm padding pentru a asigura o aliniere corespunzătoare, în conformitate cu standardul Deflate.

III. Rezultate

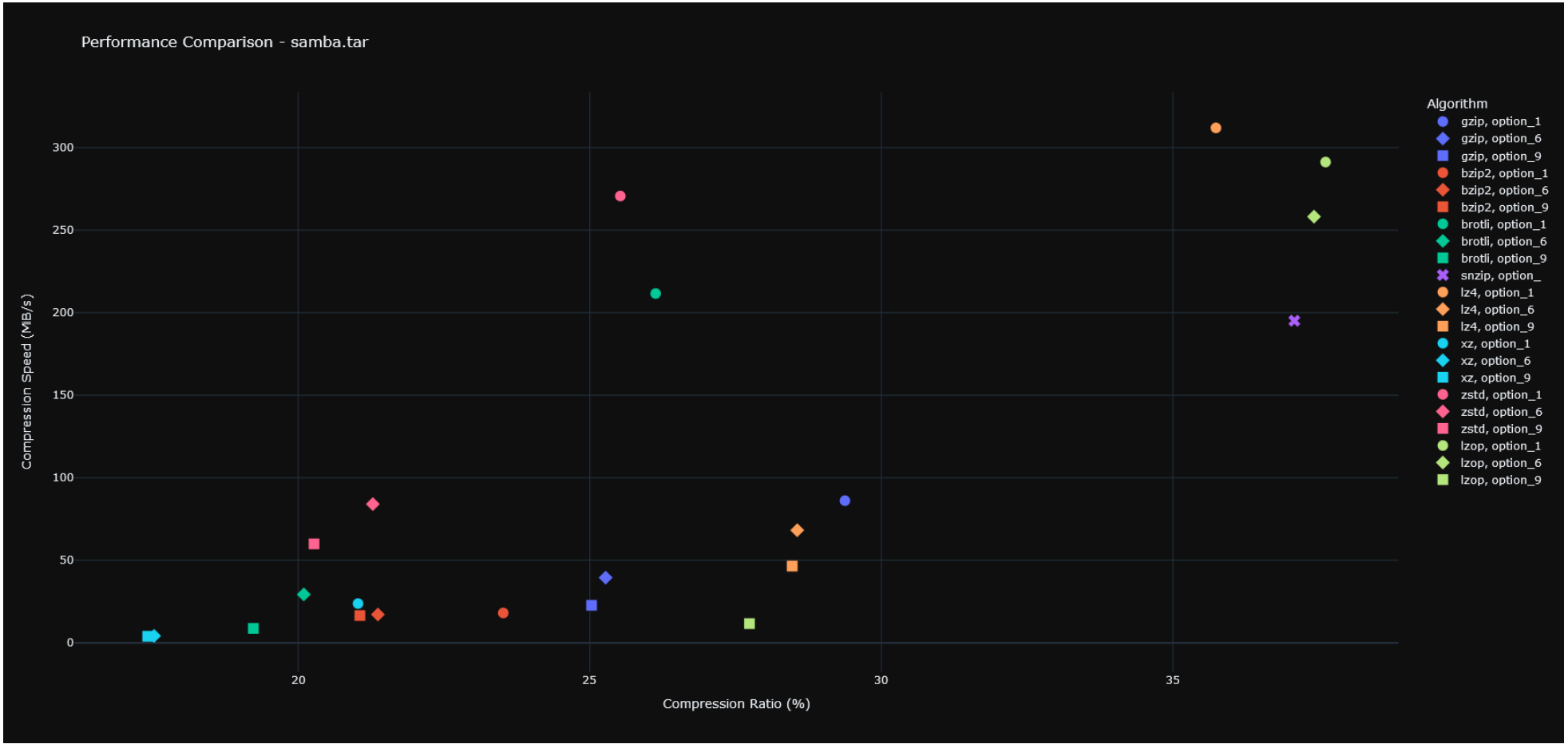
1. Caragiale.txt - Volumul “Nuvele” Ion Luca Caragiale



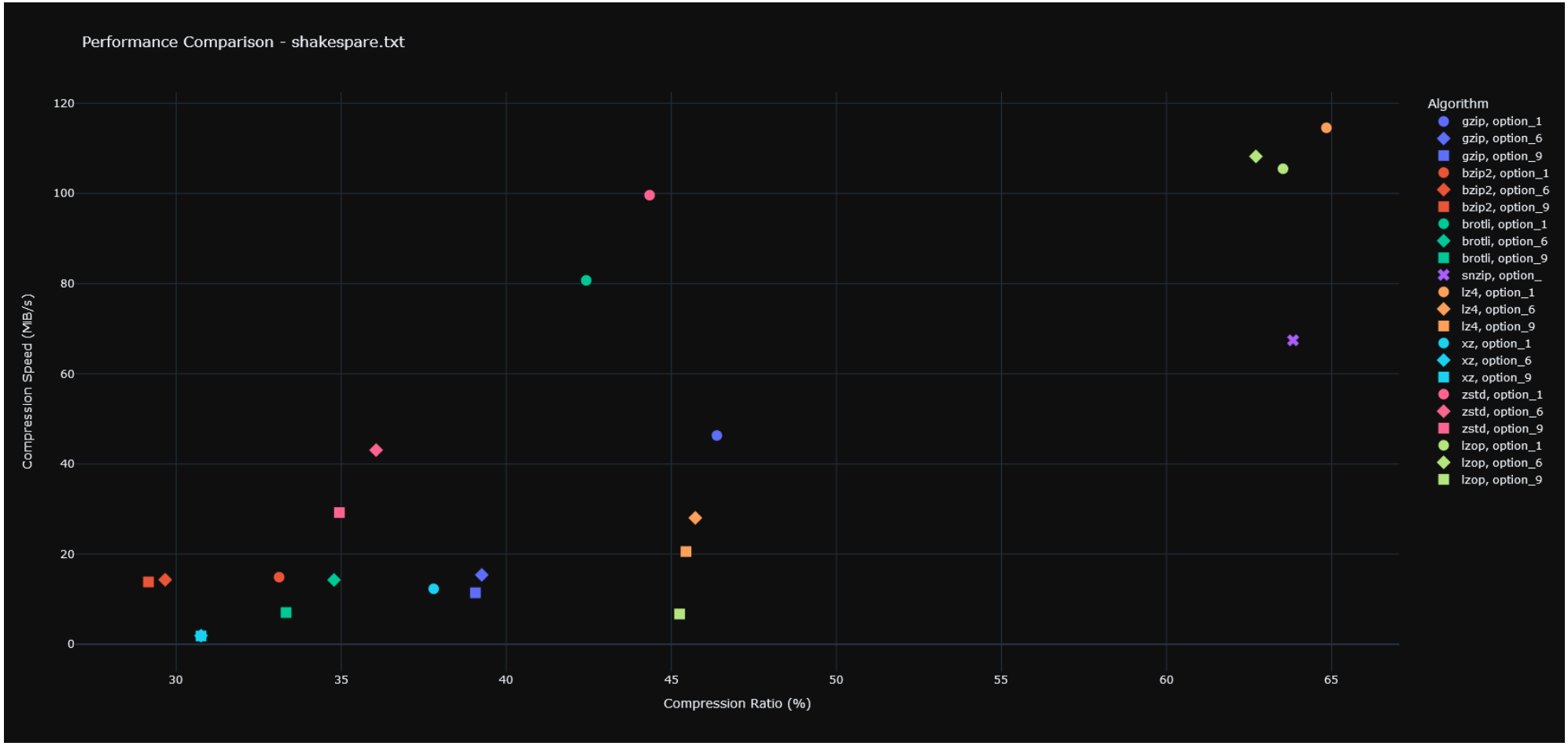
2. Kennedy.xls - Spreadsheet Excel cu informatii economice



3. Samba.tar - Codul susrsa a Samba 2-2.3



4. Shakespeare.txt - The Complete Works of William Shakespeare



Concluzie

Această lucrare explorează tehnicile de compresie a datelor Deflate, și anume LZSS și Huffman coding cu package-merge. Este prezentată o implementare simplificată a algoritmului Deflate, caracterizată de un bloc mare singular. Blocurile de codare cu prefix personalizat cu arbori Huffman demonstrează abilitățile generative acordate de algoritmii de tip package-merge.În general, investigația noastră a fost avantajoasă în extinderea înțelegerii Deflate și a tehnicilor de compresie a datelor în general.

Bibliografie

- https://datatracker.ietf.org/doc/html/rfc1952
- https://datatracker.ietf.org/doc/html/rfc1951
- https://www.youtube.com/watch?v=oi2lMBBjQ8s
- https://youtu.be/SJPvNi4HrWQ
- http://www.codersnotes.com/notes/elegance-of-deflate/