Lecture 9

Universal Compression with Lempel-Ziv compression

Recap

- Notion of conditional entropy & entropy rate.
- Context-based adaptive arithmetic coding.

Recall, entropy rate:

$$H(\mathbf{U}) = \lim_{n\to\infty} H(U_{n+1}|U_1, U_2, \ldots, U_n) = \lim_{n\to\infty} \frac{H(U_1, U_2, \ldots, U_n)}{n}$$

Today

Study *universal* compressors - a scheme that does *well* on **any** stationary input without prior knowledge of the source distribution.

As part of this - explore one of the most common schemes used in practical compressors!

Universal compressor

Consider a compressor C that works on arbitrary length inputs and has length function $l(x^n)$.

> **Definition: Universal Compressor**
> C is universal if
>
> $$\lim_{n\to\infty} \frac{1}{n} E[l(X^n)] = H(\mathbf{X})$$
>
> for any stationary ergodic source.

So a single compressor C is asymptotically optimal for every stationary distribution without prior knowledge of the source distribution!

Thinking in terms of universal predictors

- Recall from last lecture that a compressor induces a distribution via it's length function: $\hat{p}(x^n) = 2^{-l(x^n)}$.
- A universal compressor's $\hat{p}$ approximates any stationary distribution arbitrarily closely as n grows!
- In particular a universal compressor is a universal predictor!

Thinking in terms of universal predictors

- Recall from last lecture that a compressor induces a distribution via it's length function: $\hat{p}(x^n) = l(x^n)$.
- A universal compressor's $\hat{p}$ approximates any stationary distribution arbitrarily closely as n grows!
- In particular a universal compressor is a universal predictor!

All this needs to be rigorously formulated, e.g., see the reference below, talk to Tsachy, and take EE 376C!

Ref: M. Feder, N. Merhav and M. Gutman, "Universal prediction of individual sequences," in IEEE Transactions on Information Theory, vol. 38, no. 4, pp. 1258-1270, July 1992, doi: 10.1109/18.144706.

Lempel-Ziv universal algorithms

- LZ77: in gzip, zstd, png, zip, lz4, snappy
- LZ78
- LZW (Lempel-Ziv-Welch) (LZ78 variant): in linux compress utility, GIF
- LZMA (Lempel–Ziv–Markov chain algorithm) (LZ77 variant): 7-Zip

References:

1. **LZ77:** Ziv, Jacob, and Abraham Lempel. "A universal algorithm for sequential data compression." IEEE Transactions on information theory 23.3 (1977): 337-343.
2. **LZ78:** Ziv, Jacob, and Abraham Lempel. "Compression of individual sequences via variable-rate coding." IEEE transactions on Information Theory 24.5 (1978): 530-536.
3. **LZW:** Welch, Terry A. "A technique for high-performance data compression." Computer 17.06 (1984): 8-19.

LZ77 algorithm

Simple idea: Replace repeated segments in data with pointers and lengths!

Around the World

Song by Daft Punk

Lyrics

Around the world, around the world
Around the world, around the world
Around the world, around the world
Around the world, around the world
Around the world, around the world
Around the world, around the world
Around the world, around the world
Around the world, around the world

Around the world, around the world
Around the world, around the world

LZ77 parsing example

ABBABBABBCAB

Unmatched literals	Match length	Match offset
-	-	-
-	-	-
-	-	-

LZ77 parsing example

A[B]BABBABBCAB

Unmatched literals	Match length	Match offset
AB	1	1
-	-	-
-	-	-

LZ77 parsing example

[ABBABB]ABBCAB

Unmatched literals	Match length	Match offset
AB	1	1
-	6	3
-	-	-

LZ77 parsing example

ABBABB[AB]BCAB

Unmatched literals	Match length	Match offset
AB	1	1
-	6	3
C	2	4

LZ77 unparsing example

Unmatched literals	Match length	Match offset
AB	1	1
-	6	3
C	2	4

Decoded:

LZ77 unparsing example

Unmatched literals	Match length	Match offset
AB	1	1
-	6	3
C	2	4

Decoded: ABB

LZ77 unparsing example

Unmatched literals	Match length	Match offset
AB	1	1
-	6	3
C	2	4

Decoded: ABBABBABB

LZ77 unparsing example

Unmatched literals	Match length	Match offset
AB	1	1
-	6	3
C	2	4

Decoded: ABBABBABBCAB

LZ77 parsing

Pseudocode:

```
For input sequence x[0], x[1], ...

Suppose we have parsed till x[i-1].

- Try to find largest k such that for some j < i
  x[j:j+k] = x[i:i+k]

- Then the match length is k and the match offset is i-j

  [note that the ranges j:j+k and i:i+k are allowed to overlap]

- If no match found, store as literal.
```

LZ77 unparsing

Pseudocode:

```
At each step:

- First read any literals and copy to output y.

- To decode a match with length l and offset o.
  - If l < o:
    - append y[-o:-o+l] to y
  - Else:
    // Need to be more careful with overlapping matches!
    - For _ in 0:l:
      - append y[-o] to y
```

Decompression is very fast since it just involves copying!

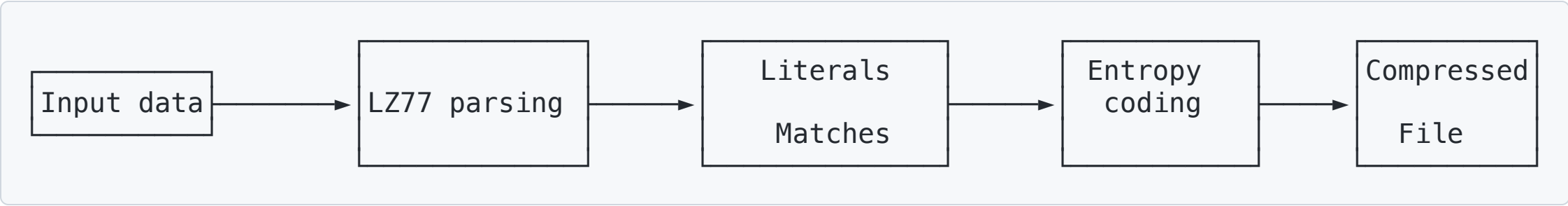
Quiz question

Apply the above parsing and unparsing algorithms for the following:

1. Parse **AABBBBBBBAABBBCDCDCD**.
2. Unparse the below table (note that this parsing was generated using a different parser than the one described above!):

Unmatched literals	Match length	Match offset
AABBB	4	1
-	5	9
CDCD	2	2

Encoding step



- Need to encode the literals, match lengths and match offsets.
- Implementations (gzip, zstd, etc.) differ in the approach.
- Typically use Huffman coding/ANS with some modifications to optimize for real-life data.
- More on this later today and in next lecture!

LZ77 universality proof idea

Question:

Consider iid sequence $\ldots, X_{-2}, X_{-1}, X_0, X_1, X_2, \ldots$

If $X_0 = a$, what is the expected value of the T such that $X_{-T} = a$ and $X_{-s} \neq a$ for $0 < s < T$? [in words, expected time we last saw a]

LZ77 universality proof idea

Question:

Consider *iid* sequence $\ldots, X_{-2}, X_{-1}, X_0, X_1, X_2, \ldots$

If $X_0 = a$, what is the expected value of the T such that $X_{-T} = a$ and $X_{-s} \neq a$ for $0 < s < T$? [in words, expected time we last saw a]

Hint: What is the mean of the geometric distribution?

Hint: How many times does a occur in a sequence of length n? How often it must repeat?

LZ77 universality proof idea

Question:

Consider *iid* sequence $\ldots, X_{-2}, X_{-1}, X_0, X_1, X_2, \ldots$

If $X_0 = a$, what is the expected value of the T such that $X_{-T} = a$ and $X_{-s} \neq a$ for $0 < s < T$? [in words, expected time we last saw a]

Hint: What is the mean of the geometric distribution?

Hint: How many times does a occur in a sequence of length n? How often it must repeat?

Answer: $E[t] = \frac{1}{P(a)}$

LZ77 universality proof idea

Generalize the above property to stationary ergodic processes.

> **Kac's Lemma**
> Let $\ldots, X_{-2}, X_{-1}, X_0, X_1, X_2, \ldots$ be a stationary ergodic process and let $R_n(X_0, \ldots, X_{n-1})$ be the recurrence time (last time $X_0, \ldots, X_{n-1}$ occurred before index 0). Given that $(X_0, \ldots, X_{n-1}) = x_0^{n-1}$, we have
>
> $$E[R_n(X_0, \ldots, X_{n-1})] = \frac{1}{p(x_0^{n-1})}$$

So you are likely to have seen x_0^{n-1} with a match offset of $\frac{1}{p(x_0^{n-1})}$.

LZ77 universality proof idea

- Can encode the match offset $\frac{1}{p(x_0^{n-1})}$ using close to $\log_2 \frac{1}{p(x_0^{n-1})}$ bits using an appropriate uint coder (e.g., check out the Elias Delta code in SCL which encodes a positive integer n in roughly $\log_2 n + 2 \log_2 \log_2 n$ bits).

LZ77 universality proof idea

- Can encode the match offset $\frac{1}{p(x_0^{n-1})}$ using close to $\log_2 \frac{1}{p(x_0^{n-1})}$ bits using an appropriate uint coder (e.g., check out the Elias Delta code in SCL which encodes a positive integer n in roughly $\log_2 n + 2 \log_2 \log_2 n$ bits).
- Match length and literal contribution is negligible!

LZ77 universality proof idea

- Can encode the match offset $\frac{1}{p(x_0^{n-1})}$ using close to $\log_2 \frac{1}{p(x_0^{n-1})}$ bits using an appropriate uint coder (e.g., check out the Elias Delta code in SCL which encodes a positive integer n in roughly $\log_2 n + 2 \log_2 \log_2 n$ bits).
- Match length and literal contribution is negligible!
- $$E[l(X^n)] \approx E[\log_2 \frac{1}{p(x_0^{n-1})}] = H(X^n)$$

LZ77 universality proof idea

- Can encode the match offset $\frac{1}{p(x_0^{n-1})}$ using close to $\log_2 \frac{1}{p(x_0^{n-1})}$ bits using an appropriate uint coder (e.g., check out the Elias Delta code in SCL which encodes a positive integer n in roughly $\log_2 n + 2 \log_2 \log_2 n$ bits).
- Match length and literal contribution is negligible!
- $$E[l(X^n)] \approx E[\log_2 \frac{1}{p(x_0^{n-1})}] = H(X^n)$$
- $$\lim_{n\to\infty} \frac{1}{n} E[l(X^n)] \approx \lim_{n\to\infty} \frac{1}{n} H(X^n) = H(\mathbf{X})$$

LZ77 universality proof idea

- Can encode the match offset $\frac{1}{p(x_0^{n-1})}$ using close to $\log_2 \frac{1}{p(x_0^{n-1})}$ bits using an appropriate uint coder (e.g., check out the Elias Delta code in SCL which encodes a positive integer n in roughly $\log_2 n + 2 \log_2 \log_2 n$ bits).
- Match length and literal contribution is negligible!
- $$E[l(X^n)] \approx E[\log_2 \frac{1}{p(x_0^{n-1})}] = H(X^n)$$
- $$\lim_{n\to\infty} \frac{1}{n} E[l(X^n)] \approx \lim_{n\to\infty} \frac{1}{n} H(X^n) = H(\mathbf{X})$$

For a more detailed and rigorous proof, check out Cover and Thomas chapter 13 or A. D. Wyner and J. Ziv, "The sliding-window Lempel-Ziv algorithm is asymptotically optimal," in Proceedings of the IEEE, vol. 82, no. 6, pp. 872-877, June 1994, doi: 10.1109/5.286191.

LZ77 universality proof idea

Asymptotic theory doesn't fully explain the excellent performance in practice: the idea of finding matches is just very well-matched to real-life data which is not always modeled easily as a kth order Markov process.

LZ77 parsing on real data - examples

Let's look at how matches look in practice and how the match lengths and offsets are typically distributed.

We use the LZ77 implementation in SCL for this purpose.

LZ77 parsing on real data - examples

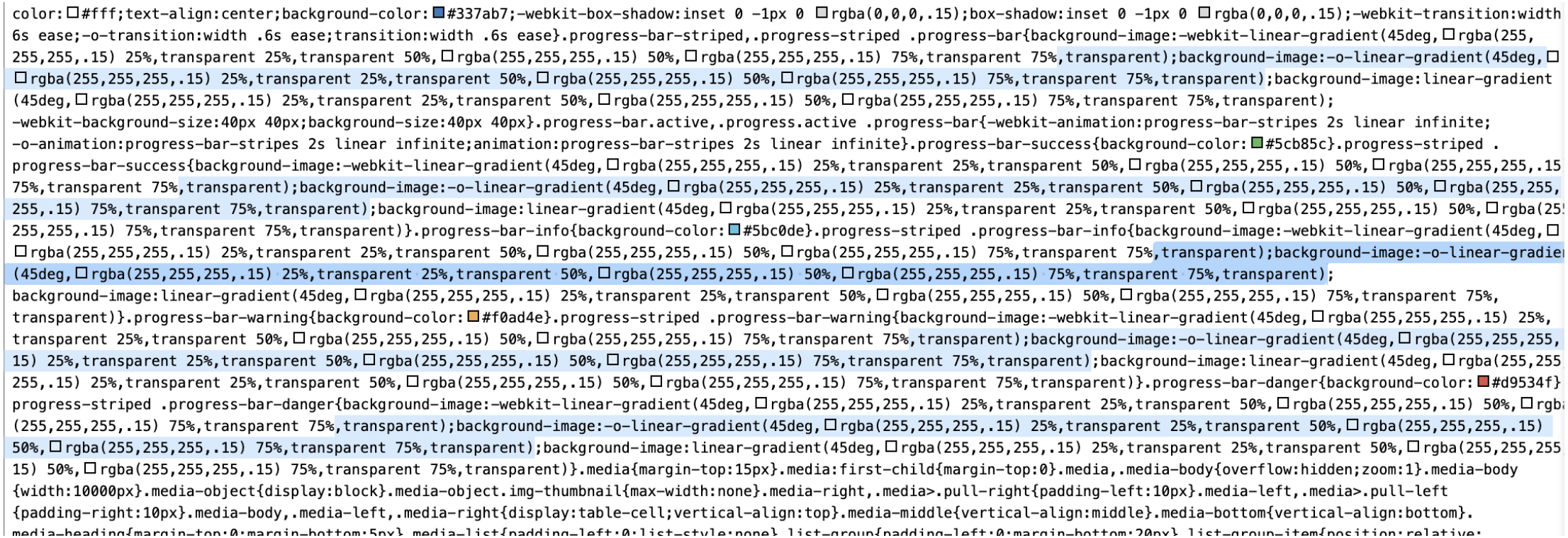
Long matches:

```
`Beautiful Soup, so rich and green,
Waiting in a hot tureen!
Who for such dainties would not stoop?
Soup of the evening, beautiful Soup!
Soup of the evening, beautiful Soup!
    Beau--ootiful Soo--oop!
    Beau--ootiful Soo--oop!
Soo--oop of the e--e--evening,
    Beautiful, beautiful Soup!

`Beautiful Soup!  Who cares for fish,
Game, or any other dish?
Who would not give all else for two p
ennyworth only of beautiful Soup?
Pennyworth only of beautiful Soup?
    Beau--ootiful Soo--oop!
    Beau--ootiful Soo--oop!
Soo--oop of the e--e--evening,
    Beautiful, beauti--FUL SOUP!'
```

LZ77 parsing on real data - examples

Long matches:



LZ77 parsing on real data - examples

Far off matches (150 KB apart) [`pleasure`]:

First page:

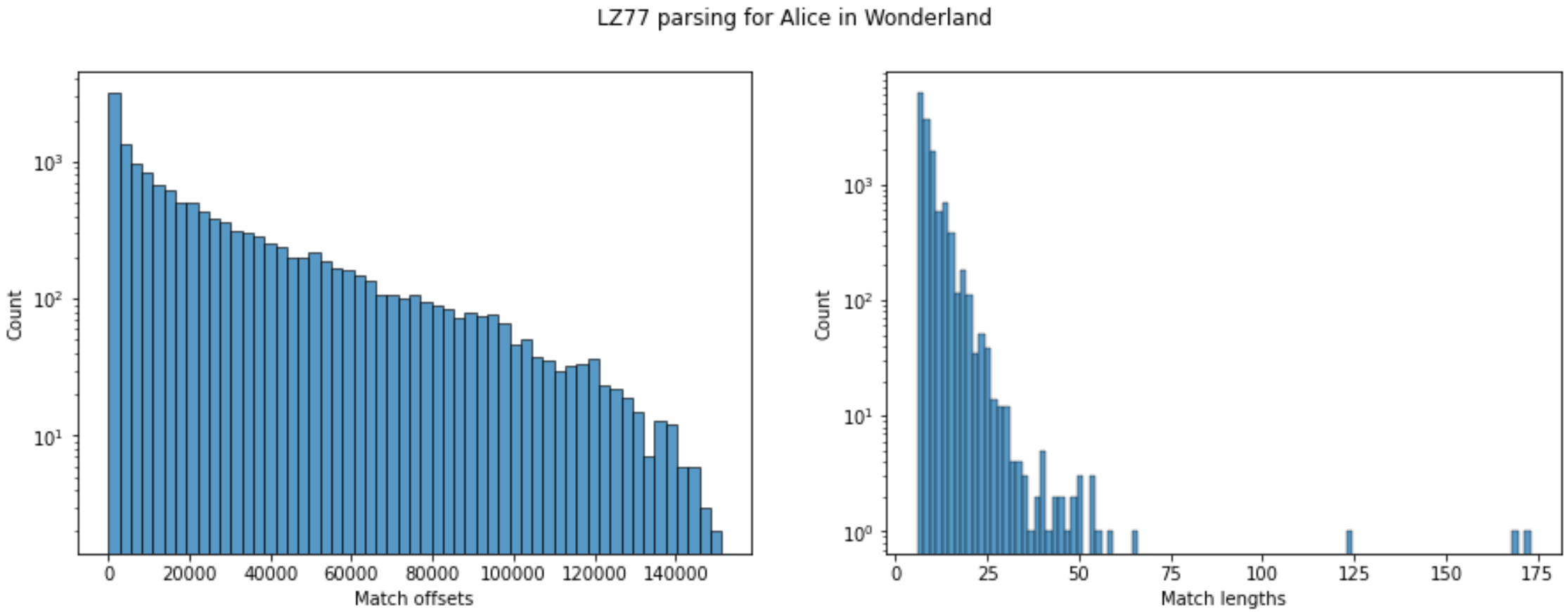
```
So she was considering in her own mind (as well as she could,
for the hot day made her feel very sleepy and stupid), whether
the pleasure of making a daisy-chain would be worth the trouble
of getting up and picking the daisies, when suddenly a White
Rabbit with pink eyes ran close by her.
```

Last page:

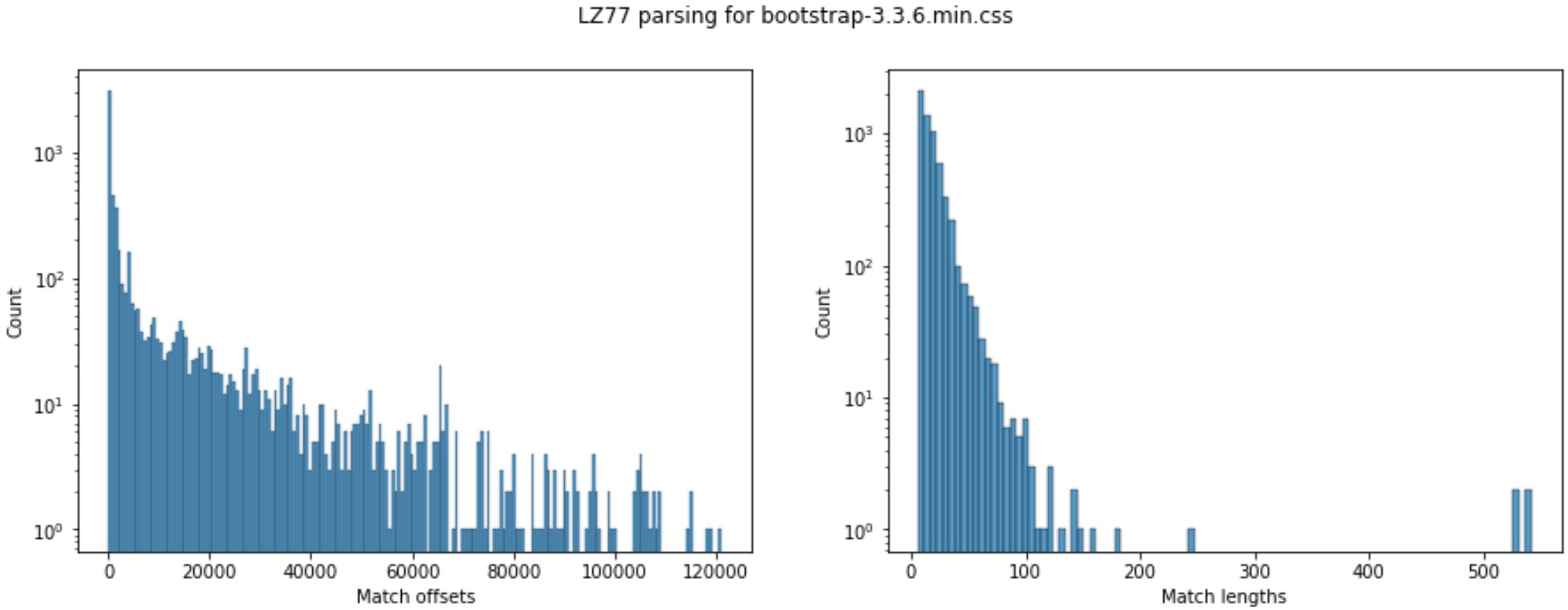
```
, and make THEIR eyes bright and eager
with many a strange tale, perhaps even with the dream of
Wonderland of long ago:  and how she would feel with all their
simple sorrows, and find a pleasure in all their simple joys,
remembering her own child-life, and the happy summer days.

                                THE END
```

LZ77 parsing on real data - examples



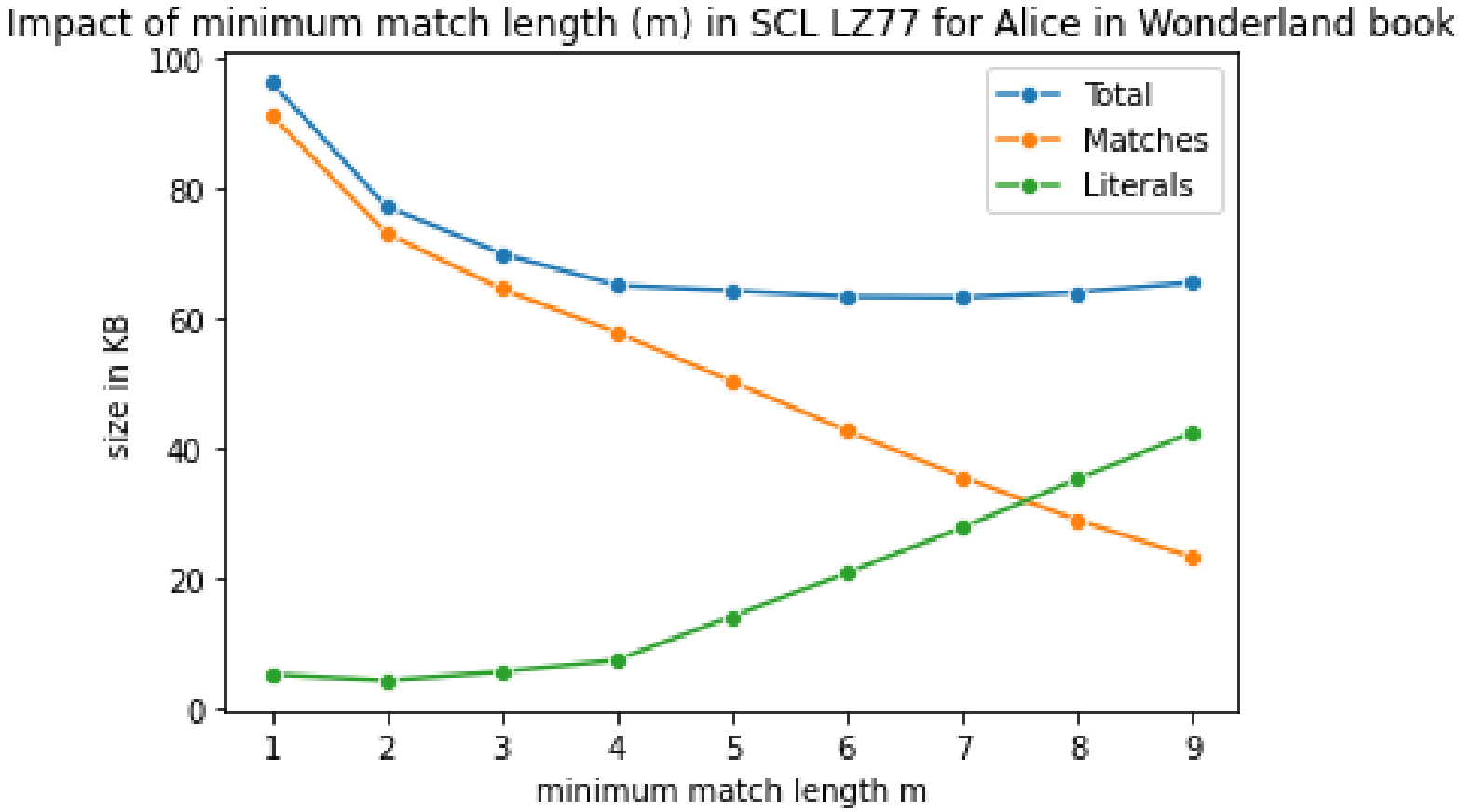
LZ77 parsing on real data - examples



Practical considerations

- How to find matches?
  - Fix minimum match length and index in hash table (check out the chained hash table for an optimized implementation!)

Practical considerations



Practical considerations

- How to find matches?
  - Fix minimum match length and index in hash table (check out the chained hash table for an optimized implementation!)
- Do I need to keep infinite past memory?!

Practical considerations

- How to find matches?
  - Fix minimum match length and index in hash table (check out the chained hash table for an optimized implementation!)
- Do I need to keep infinite past memory?!
  - Use sliding window - only find matches in past 10s of KBs (gzip) to multiple MBs (zstd) window.

Practical considerations

- How to find matches?
  - Fix minimum match length and index in hash table (check out the chained hash table for an optimized implementation!)
- Do I need to keep infinite past memory?!
  - Use sliding window - only find matches in past 10s of KBs (gzip) to multiple MBs (zstd) window.
- Is finding the longest match at every step optimal?

Practical considerations

- How to find matches?
  - Fix minimum match length and index in hash table (check out the chained hash table for an optimized implementation!)
- Do I need to keep infinite past memory?!
  - Use sliding window - only find matches in past 10s of KBs (gzip) to multiple MBs (zstd) window.
- Is finding the longest match at every step optimal?
  - No. Sometimes a literal costs less than encoding a very short match.
  - Can find a longer match at the next position if we sacrifice to get a literal/shorter match at this step.
  - The tradeoffs depend on how the entropy encoding works.

Entropy coding

Unmatched literals	Match length	Match offset
AABBB	4	1
-	5	9
CDCD	2	2

encoded as

Entropy coding

```
literals = AABBBCDCD
```

and

Literal counts	Match length	Match offset
5	4	1
0	5	9
4	2	2

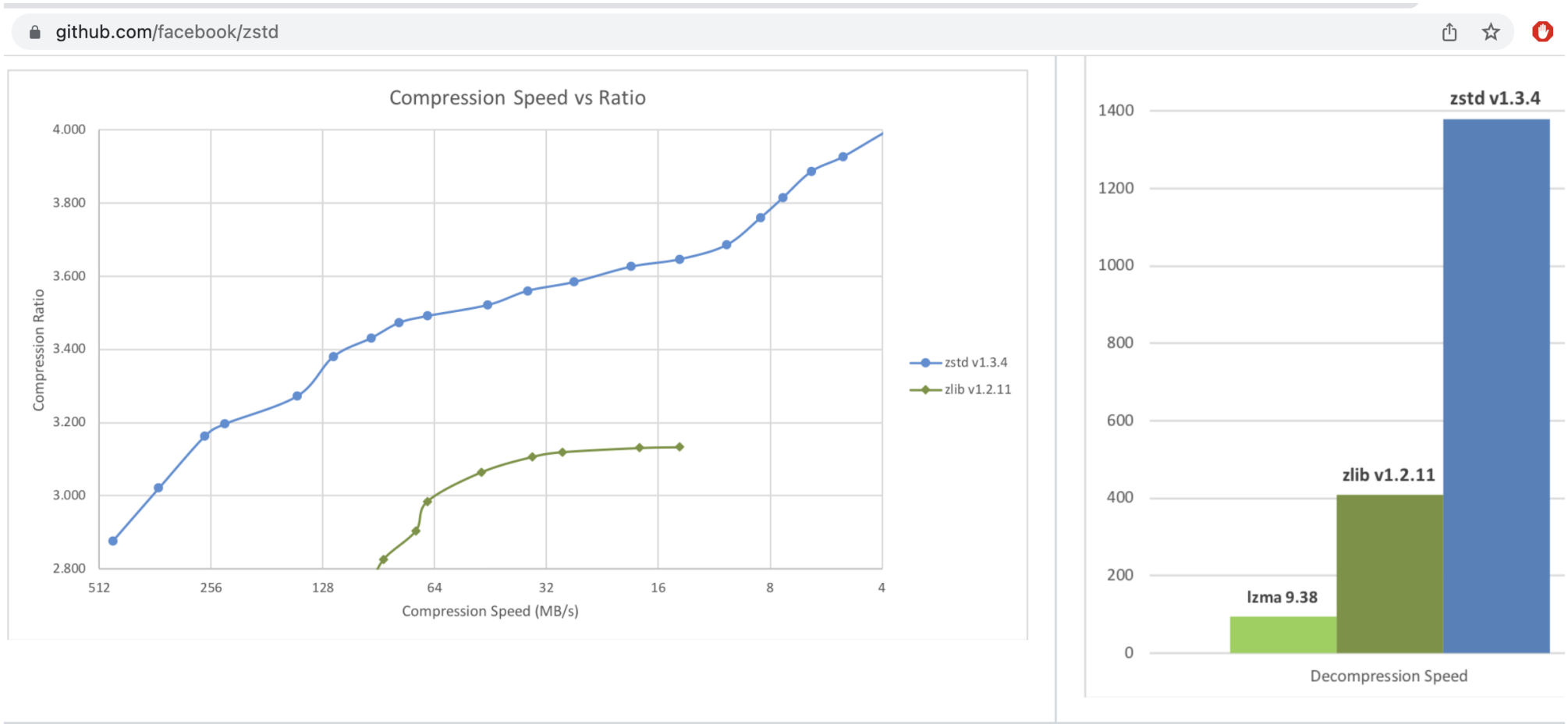
Entropy coding

Each of streams encoded using various entropy coding approaches.

- Huffman - dynamic/static
  - zstd - Huffman only for literals
- ANS/arithmetic coding
- Universal integer coders
- For very high speeds, skip entropy coding! (LZ4, Snappy)

Practical considerations

- Parsing strategy, window size, entropy coding matters a lot in determining speed and memory usage.



That's it for now!

- Next lecture - Yann Collet: author of zstd, lz4, FSE (implementation of tANS)
- We didn't talk about LZ78 and LZW - similar core ideas but slightly different tree-based parsing method
  - For LZ78, possible to prove very powerful universality results, including non-asymptotic ones!
  - In particular can show that LZ78 gets compression rate within $O(\frac{k}{\log n} + \frac{\log \log n}{\log n})$ of the optimal kth order model for any sequence.
  - Learn more in EE 376C.