

Text Indexing

Lecture 04: Text-Compression

Florian Kurpicz

The slides are licensed under a Creative Commons Attribution-ShareAlike 4.0 International License © ⓘ ⓘ: www.creativecommons.org/licenses/by-sa/4.0 | commit 224e27c compiled at 2022-11-21-13:01



<https://pingo.scc.kit.edu/651997>

Recap: Suffix Array and LCP-Array

Definition: Suffix Array [GBS92; MM93]

Given a text T of length n , the **suffix array** (SA) is a permutation of $[1..n]$, such that for $i \leq j \in [1..n]$

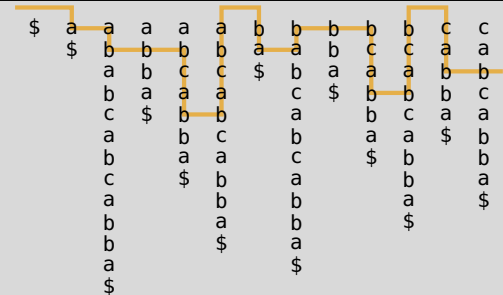
$$T[SA[i]..n] \leq T[SA[j]..n]$$

Definition: Longest Common Prefix Array

Given a text T of length n and its SA, the **LCP-array** is defined as

$$LCP[i] = \begin{cases} 0 & i = 1 \\ \max\{\ell: T[SA[i]..SA[i] + \ell) = \\ T[SA[i - 1]..SA[i - 1] + \ell)\} & i \neq 1 \end{cases}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
LCP	0	0	1	2	2	5	0	2	1	1	4	0	3



Why Compression

Types of Compression

- lossy compression
 - ⓘ audio, video, pictures, ...
- lossless compression
 - ⓘ audio, **text**, ...

- only interested in lossless compression
- faster data transfer
- cheaper storage costs
- “compress once, decompress often”

Types of Text-Compression

- entropy coding ⓘ compress characters
- dictionary compression ⓘ compress substrings
- ...

This Lecture

- measure compressibility
- different compression algorithms
 - ⓘ both types
- space/time requirements of compression algorithms
- make use of known concepts

k -th Order Empirical Entropy [KM99] (1/2)

Definition: Histogram

Given a text T of length n over an alphabet of size σ , a histogram $Hist[1..\sigma]$ is defined as

$$Hist[i] = |\{j \in [1, n]: T[j] = i\}|$$

Definition: 0-th Order Empirical Entropy

Given a text T of length n over an alphabet $\Sigma = [1, \sigma]$ and its histogram $Hist$, then

$$H_0(T) = (1/n) \sum_{i=1}^{\sigma} Hist[i] \lg(n/Hist[i])$$

- $T = \text{abbaaacaaba\$}$
- $n = 12$
- $Hist[\text{a}] = 7$
- $Hist[\text{b}] = 3$
- $Hist[\text{c}] = 1$
- $Hist[\text{\$}] = 1$
- $H_0(T) = (1/12)(7 \lg(12/7) + 3 \lg(12/3) + 1 \lg(12/1) + 1 \lg(12/1)) \approx 1.55$

k -th Order Empirical Entropy (2/2)


Given a text T over an alphabet Σ and a string $S \in \Sigma^k$, T_S the concatenation of all characters that occur in T after S in text order

- $T = \text{abcdabceabcd}$
- $S = \text{abc}$
- $T_S = \text{ded}$

Definition: k -th Order Empirical Entropy

Given a text T of length n over an alphabet $\Sigma = [1, \sigma]$ and its histogram $Hist$, then

$$H_k = (1/n) \sum_{S \in \Sigma^k} |T_S| \cdot H_0(T_S)$$

-  **PINGO** can we describe a property of H_k

Example for k -th Order Empirical Entropy [Kur20]

Name	σ	n	H_0	H_1	H_2	H_3
Commoncrawl	243	196,885,192,752	6.19	4.49	2.52	2.08
DNA	4	218,281,833,486	1.99	1.97	1.96	1.95
Proteins	26	50,143,206,617	4.21	4.20	4.19	4.17
Wikipedia	213	246,327,201,088	5.38	4.15	3.05	2.33
SuffixArrayCC	n	137,438,953,472	$37 (= \lg n)$	0	0	0
RussianWordBased	29 263	9,232,978,762	10.93	—	—	—

- does not measure repetitions well
- there are other measures

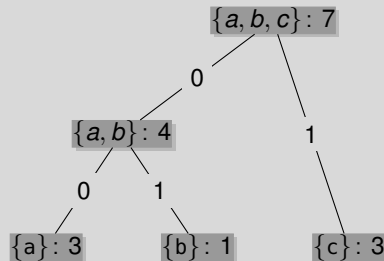


Huffman Coding [Huf52]

- idea is to create a binary tree
- each character α is a leaf and has weight $Hist[\alpha]$
- create node for two nodes **without parent** with smallest weight
- give new node total weight of children
- repeat until only one node without parent remains

- label edges:
 - left edge: 0
 - right edge: 1
- path to children gives code for character


$T = cbcacaa$




- codes are variable length and prefix-free
- tree/dictionary needed for decoding

Canonical Huffman Coding


- start with Huffman codes, code word 0, and length 1
- to get canonical code for current length, then add 1 to code word
- to update length add 1 and append required amount of zeros to code word

- all codes of same length are increasing
- required for Huffman-shaped wavelet trees
-  will be discussed in a later lecture

-  **PINGO** what are some advantages of canonical Huffman codes?


Continue From Last Slide

- length 1: c
- length 2: a, b
- start with 0 → code for c
- add 1 and append 0
- 10 → code for a
- add 1
- 11 → code for b

- still variable length and prefix-free
- instead of tree only require lengths' of codes and corresponding characters 

Shannon-Fano Coding [Fan49; Sha48]

- given a text T of length n over an alphabet Σ and its histogram $hist$
- each character $\alpha \in \Sigma$ receives a code of length $l_\alpha = \lceil \lg \frac{n}{Hist[\alpha]} \rceil$

- show that there always exists such a code
- assume a complete binary tree of depth $l_{\max} = \max_{\alpha \in \Sigma} l_\alpha$ with all free nodes
- left edges labeled 0, right edges labeled 1
- characters ordered by frequency ($l_1 \geq l_2 \geq \dots \geq l_\sigma$)
- assign characters the leftmost free node
- mark all nodes above and below as non-free 

Proof there are enough free nodes (Sketch)

- a code l_α marks $2^{\ell_{\max} - l_\alpha}$ nodes
- total number of marked leaves is

$$\begin{aligned}
 \sum_{\alpha \in \Sigma} 2^{\ell_{\max} - l_\alpha} &= 2^{\ell_{\max}} \sum_{\alpha \in \Sigma} 2^{-l_\alpha} \\
 &= 2^{\ell_{\max}} \sum_{\alpha \in \Sigma} 2^{-\lceil \lg \frac{n}{Hist[\alpha]} \rceil} \\
 &\leq 2^{\ell_{\max}} \sum_{\alpha \in \Sigma} 2^{-\lg \frac{n}{Hist[\alpha]}} \\
 &= 2^{\ell_{\max}} \sum_{\alpha \in \Sigma} \frac{Hist[\alpha]}{n} \\
 &= 2^{\ell_{\max}}
 \end{aligned}$$

Optimality of Both

- H_0 gives average number of bits needed to encode character
- $nH_0(T)$ is lower bound for compression **without context**

- one can show that no fixed-letter code can be better than Huffman **not in this lecture**
- Shannon-Fano codes can be slightly longer than Huffman
- even Shannon-Fano achieves H_0 -compression

Proof (Sketch)

- let T be a text of length n over an alphabet Σ with histogram $Hist$
- let T_{SF} be the Shannon-Fano encoded text
- average length of encoded character is

$$\begin{aligned}
 (1/n)|T_{SF}| &= (1/n) \sum_{\alpha \in \Sigma} Hist[\alpha] \lceil \lg \frac{n}{Hist[\alpha]} \rceil \\
 &\leq \sum_{\alpha \in \Sigma} \frac{Hist[\alpha]}{n} (\lg \frac{n}{Hist[\alpha]} + 1) \\
 &= \sum_{\alpha \in \Sigma} \frac{Hist[\alpha]}{n} \lg \frac{n}{Hist[\alpha]} + \sum_{\alpha \in \Sigma} \frac{Hist[\alpha]}{n} \\
 &= H_0(T) + 1
 \end{aligned}$$

Problem with the Previous Approaches

aa
aa
aa
aa
aa
aa
aa
aa
aa
aa
aa

- does not work well with repetitions
- better encode $605 \times a$

Lempel-Ziv 77 [ZL77]

Definition: LZ77 Factorization

Given a text T of length n over an alphabet Σ , the **LZ77 factorization** is

- a set of z factors $f_1, f_2, \dots, f_z \in \Sigma^+$, such that
- $T = f_1 f_2 \dots f_z$ and for all $i \in [1, z]$ f_i is
- single character not occurring in $f_1 \dots f_{i-1}$ or
- longest substring occurring ≥ 2 times in $f_1 \dots f_i$

$T = \text{abababbbbaba\$}$

- | | |
|----------------|---------------|
| ■ $f_1 = a$ | ■ $f_4 = bbb$ |
| ■ $f_2 = b$ | ■ $f_5 = aba$ |
| ■ $f_3 = abab$ | ■ $f_6 = \$$ |

$T = \underbrace{aaa \dots aa}_{n-1 \text{ times}} \$$

- $f_1 = a$
- $f_2 = \underbrace{aaa \dots aa}_{n-2 \text{ times}}$
- $f_3 = \$$

Representation of Factors

- factors can be represented as tuple

$$(\ell_i, p_i)$$

- $\ell_i = 0$
 - factor is a single character
 - encode character in p_i
- $\ell_i > 0$
 - factor is a length- ℓ_i substring
 - $f_i = T[p_i..p_i + \ell_i]$

$T = \text{abababbbbaba\$}$

- $f_1 = \text{a} = (0, \text{a})$
- $f_2 = \text{b} = (0, \text{b})$
- $f_3 = \text{abab} = (4, 1)$
- $f_4 = \text{bbb} = (3, 6)$
- $f_5 = \text{aba} = (3, 1) = (3, 3)$
- $f_6 = \text{\$} = (0, \$)$

- finding the right-most reference is hard



Previous and Next Smaller Values (1/2)

Definition: Previous and Next Smaller Value Arrays


Let $A[1..n]$ be an integer array, then

- $PSV[i] = \max\{j \in [1, i) : A[j] < A[i]\}$
- $NSV[i] = \min\{j \in (i, n] : A[j] < A[i]\}$

In the Context of SA


- close to the suffix in SA
- longest possible common prefix
- before the suffix in text order

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
PSV	0	0	0	3	3	3	6	3	8	8	8	11	11
NSV	2	3	∞	5	6	8	8	∞	10	11	∞	13	∞
LCP	0	0	1	2	2	5	0	2	1	1	4	0	3

-  **PINGO** how fast can we compute NSV/PSV?

Previous and Next Smaller Values (2/2)

- both arrays can be computed in linear time
- consider the PSV array
 - ⓘ NSV works analogously
- prepend $-\infty$ at index 0

- follow already computed values
- nothing in between can be *PSV*
- compare each element at most twice
- compute *PSV* and *NSV* in $O(n)$ time
- example on the board 

Function ComputePSV(*SA with $-\infty$*):


```

1  |  for  $i = 1, \dots, n$  do
2  |       $j = i - 1$ 
3  |      while  $j \geq 1$  and  $SA[i] < SA[j]$  do
4  |           $j = PSV[j]$ 
5  |           $PSV[i] = j$ 
6  |  return PSV
  
```


NSV, PSV, and RMQ

Recap: Range Minimum Queries

- for a range $[\ell..r]$, return **position** of smallest entry in an array in that range
- query time: $O(1)$ using $O(n)$ space
- can be used to compute the *lcp*-value of any two suffixes using the *LCP*-array

- use all arrays to find lexicographically closest suffixes
- that occur before current suffix in text order 


	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	b	a	b	c	a	b	c	a	b	b	a	\$
<i>SA</i>	13	12	1	9	6	3	11	2	10	7	4	8	5
<i>PSV</i>	0	0	0	3	3	3	6	3	8	8	8	11	11
<i>NSV</i>	2	3	∞	5	6	8	8	∞	10	11	∞	13	∞
<i>LCP</i>	0	0	1	2	2	5	0	2	1	1	4	0	3

LZ77 Factorization using *SA*, *ISA*, *LCP*, *NSV*, *PSV*, and *RMQs*

Function *LZ77*(*SA*, *ISA*, *LCP*, *RMQ*, *PSV*, *NSV*):

```

1  |  pos = 1
2  |  while pos ≤ n do
3  |  |  psv = SA[PSV[ISA[pos]]]
4  |  |  nsv = SA[NSV[ISA[pos]]]
5  |  |  if lcp(pos, psv + 1) > lcp(pos + 1, nsv) then
6  |  |  |  ℓ = lcp(pos, psv + 1) and p = psv
7  |  |  |  else
8  |  |  |  |  ℓ = lcp(pos + 1, nsv) and p = nsv
9  |  |  |  if ℓ = 0 then p = pos
10 |  |  |  new factor (ℓ, T[pos])
11 |  |  pos = pos + max{ℓ, 1}
  
```

■ bring your own example 

LZ77: Running Time

Lemma: LZ77 Running Time

The LZ77 factorization of a text of length n can be computed in $O(n)$ time

Proof (Sketch)

- $SA, LCP, PSV, NSV, RMQ_{LCP}$ can be computed in $O(n)$ time
- for each text position only $O(1)$ time

Lempel-Ziv 78 [ZL78]

Definition: LZ78 Factorization

Given a text T of length n over an alphabet Σ , the **LZ78 factorization** is


- a set of z factors $f_1, f_2, \dots, f_z \in \Sigma^+$, such that
- $T = f_1 f_2 \dots f_z$, $f_0 = \epsilon$ and for all $i \in [1, z]$
- if $f_1 \dots f_{i-1} = T[1..j-1]$, then f_i is the longest prefix of $T[j..n]$, such that

$$\exists k \in [0, i), \alpha \in \Sigma \cup \{\$\}: f_k = f_i \alpha$$

$T = \text{abababbbbaba\$}$

- $f_1 = a$
- $f_2 = b$
- $f_3 = ab$
- $f_4 = abb$
- $f_5 = bb$
- $f_6 = aba$
- $f_7 = \$$
- $T = \text{abababbbbaba\$}$

LZ78 Factorization using a Dynamic Trie

- use dynamic trie to hold computed factors
- our fastest easy to use dynamic trie is?
- using arrays of fixed size 

$T = abababbbba\$$

■ $f_1 = a$

■ $f_2 = b$

■ $f_3 = ab$

■ $f_4 = abb$

■ $f_5 = bb$

■ $f_6 = aba$

■ $f_7 = \$$

LZ78 Factorization in Linear Time

Lemma:

The LZ78 factorization of a text of length n can be computed in $O(n)$ time

Proof (Sketch)

- search each character of the text at most once (in the trie)
- insert each character of the text at most once (in the trie)

Sliding Window

- memory usage of the LZ78 factorization very high **i** using arrays of fixed size does not help
- consider only a sliding window of the text
- only factors in the window are found
- space/compression rate trade-off
- used in practice

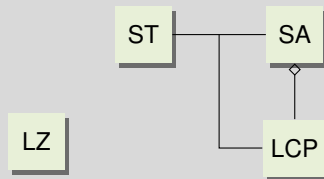
Conclusion and Outlook

This Lecture

- different compression methods for texts
- entropy coding
- dictionary compression

- LZ77 and LZ78 have been generalize, improved, and combined: 🧠 a lot!
- LZ77
 - LZSS, LZB, LZR, LZH, . . .
- LZ78
 - LZO, LZJ, LZM, LZMW, LZJ, . . .

Linear Time Construction



Next Lecture

- easy to compress index

Bibliography I

- [Fan49] Robert M. Fano. *The Transmission of Information*. Massachusetts Institute of Technology, Research Laboratory of Electronics, 1949.
- [GBS92] Gaston H. Gonnet, Ricardo A. Baeza-Yates, and Tim Snider. “New Indices for Text: Pat Trees and Pat Arrays”. In: *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, 1992, pages 66–82.
- [Huf52] David A. Huffman. “A Method for the Construction of Minimum-Redundancy Codes”. In: *Proceedings of the IRE* 40.9 (1952), pages 1098–1101. DOI: [10.1109/JRPR0C.1952.273898](https://doi.org/10.1109/JRPR0C.1952.273898).
- [KM99] S. Rao Kosaraju and Giovanni Manzini. “Compression of Low Entropy Strings with Lempel-Ziv Algorithms”. In: *SIAM J. Comput.* 29.3 (1999), pages 893–911. DOI: [10.1137/S0097539797331105](https://doi.org/10.1137/S0097539797331105).
- [Kur20] Florian Kurpicz. “Parallel Text Index Construction”. PhD thesis. Technical University of Dortmund, Germany, 2020. DOI: [10.17877/DE290R-21114](https://doi.org/10.17877/DE290R-21114).

Bibliography II

- [MM93] Udi Manber and Eugene W. Myers. “Suffix Arrays: A New Method for On-Line String Searches”. In: *SIAM J. Comput.* 22.5 (1993), pages 935–948. DOI: [10.1137/0222058](https://doi.org/10.1137/0222058).
- [Sha48] Claude E. Shannon. “A mathematical theory of communication”. In: *Bell Syst. Tech. J.* 27.3 (1948), pages 379–423. DOI: [10.1002/j.1538-7305.1948.tb01338.x](https://doi.org/10.1002/j.1538-7305.1948.tb01338.x).
- [ZL77] Jacob Ziv and Abraham Lempel. “A Universal Algorithm for Sequential Data Compression”. In: *IEEE Trans. Inf. Theory* 23.3 (1977), pages 337–343. DOI: [10.1109/TIT.1977.1055714](https://doi.org/10.1109/TIT.1977.1055714).
- [ZL78] Jacob Ziv and Abraham Lempel. “Compression of Individual Sequences via Variable-Rate Coding”. In: *IEEE Trans. Inf. Theory* 24.5 (1978), pages 530–536. DOI: [10.1109/TIT.1978.1055934](https://doi.org/10.1109/TIT.1978.1055934).