

## Compression

- Scenario: communicating over "expensive" line.
- Goal: zip function, $f: \Sigma \rightarrow\{0,1\}^{*}$.
- Lossless Reconstruction: easily invertible (i.e., $f^{-1}$ ).
- Size Efficiency: good compression (i.e., $\mid f($ text $)|\ll| t e x t \mid$ ).
- Impossible to perfect both, why?
- $f$ must be injective.
- If some strings are "zipped" others must "expand".
-But..
- Do we care for all the strings?
- What's special about of strings of interest?


## Huffman Coding

## Huffman Coding

- Main idea: chars in human text do not distribute uniformly.
- Use corpus to build compression function $H: \Sigma \rightarrow\{0,1\}^{*}$
- Frequent letters $\rightarrow$ short encoding.
- Scheme:
- Alice: compress message with $H(m s g)$.
- Bob: decompress message (with $H^{-1}$ ).
- Which corpus?

Corpus


## Huffman Coding - Stages



## Huffman Coding - Forest Stage

## Pseudocode:

build forest from corpus
Init forest with frequencies nodes
while $\mid$ forest $\mid>1$ :
Extract 2 min trees: t1, t2
Union $\mathrm{t}^{\prime}=\mathrm{t} 1+\mathrm{t} 2$
Put t' in forest

## Huffman Coding - Forest Stage

## Pseudocode:

build forest from corpus
Init forest with frequencies nodes
while |forest| > 1:
Extract 2 min trees: t1, t2
Union $\mathrm{t}^{\prime}=\mathrm{t} 1+\mathrm{t} 2$
Put $\mathrm{t}^{\prime}$ in forest
corpus = 'aaaabbcdddeee'
Huffman Coding - Forest Stage
Pseudocode:
build forest from corpus
Init forest with frequencies nodes
while $\mid$ forest $\mid>1$ :
Extract 2 min trees: t1, t2
Union $\mathrm{t}^{\prime}=\mathrm{t} 1+\mathrm{t} 2$
Put $\mathrm{t}^{\prime}$ in forest

corpus = 'aaaabbcdddeee'
Huffman Coding
Pseudocode:
build forest from corpus
Init forest with frequencies nodes
while $\mid$ forest $\mid>1$ :
Extract 2 min trees: t1, t2
Union $\mathrm{t}^{\prime}=\mathrm{t} 1+\mathrm{t} 2$
Put $\mathrm{t}^{\prime}$ in forest

## Huffman Coding - Extract Code


corpus = 'aaaabbcdddeee'

Pseudocode: color edges
Left=0, right=1
Define $H(c):=$ path $($ root $\rightarrow c)$

- $H\left({ }^{\prime} c^{\prime}\right)=000$
- $H\left({ }^{\prime} b^{\prime}\right)=001$
- $H\left({ }^{\prime} d^{\prime}\right)=01$
- $H\left({ }^{\prime} e^{\prime}\right)=10$
- $H\left({ }^{\prime} a^{\prime}\right)=11$


## Huffman Coding



Claim: Chars $\leftrightarrow$ leaves.
Proof: Induction on |forest|.

Claim: Huffman Code is prefix free*. Proof: If not, $H(x)=H(y)+\cdots$ but then $x$ is descendant of $y \ldots$

Why is this important?

Claim: Each is either a leaf or have two children.

- $H\left({ }^{\prime} c^{\prime}\right)=000$

Compressing and decompressing

- $H\left({ }^{\prime} b^{\prime}\right)=001$
- $H\left({ }^{\prime} d^{\prime}\right)=01$

Alice: $H$ ('babd') = 0011100101
Bob: 0011100101

- $H\left({ }^{\prime} e^{\prime}\right)=10$
- $H\left({ }^{\prime} a^{\prime}\right)=11$
- 0?
- 00?
- 001? b

Bob: 0011100101

- 1?
- 11? a

Bob: 0011100101...

- $H\left({ }^{\prime} c^{\prime}\right)=000$

Compressing and decompressing
Alice: $H$ ('babd') = 0011100101
Why does it work?
Imagine:

- $f\left({ }^{\prime} a^{\prime}\right)=1$
- $f\left({ }^{\prime} b^{\prime}\right)=01$
- $f\left({ }^{\prime} c^{\prime}\right)=011$

Alice: 'c' $\rightarrow 011$
Bob: $011 \rightarrow{ }^{\prime}$ ba'
$H$ is prefix free! $f$ is not.

## Prefix Free and Uniquely Decodable

Let $f: \Sigma \rightarrow\{0,1\}^{*}$ be a compression scheme

- We say $f$ is Prefix Free if $f(x)$ is never a prefix of $f(y)$.
- We say $f$ is Uniquely Decodable if for any string $m$ $\in\{0,1\}^{*}$ there is at most 1 message $\sigma_{1} \ldots \sigma_{k} \in \Sigma^{k}$ such that $f\left(\sigma_{1} \ldots \sigma_{k}\right)=m$.
- Said differently, $f$ 's codes are invertible.

Claim: PF $\rightarrow$ UD
Other direction?

## Prefix Free and Uniquely Decodable



## Optimality

Claim: Among all codes $C$ in which each character is encoded separately, Huffman is optimal!

- Optimal $:=$ minimizes $\sum_{a_{i} \in \Sigma}\left(\left|f\left(a_{i}\right)\right| \cdot w_{i}\right)$
(i.e., minimizing the length of the encoded corpus).
- Where -
- $f\left(a_{i}\right)$ is the length of the encoding of $a_{i}$.
- $w_{i}$ is the number of appearances of $a_{i}$ in the corpus.
- Interesting when text distributes as corpus.


## Example 1

- Draw Huffman tree for a corpus with $|\Sigma|=2^{n}$, where each character appears once ( $\forall i$ : appear $\left(a_{i}\right)=1$ ).

Shortest encoding $=$ longest encoding $=n$


## Example 2

- Draw Huffman tree for a corpus with $|\Sigma|=n$, where the $i$-th character appears $2^{i}$ times $\left(0 \leq i \leq n-1\right.$ : appear $\left.\left(a_{i}\right)=2^{i}\right)$



## Example 2

- Draw Huffman tree for a corpus with $|\Sigma|=n$, where the $i$-th character appears $2^{i}$ times $\left(0 \leq i \leq n-1\right.$ : appear $\left.\left(a_{i}\right)=2^{i}\right)$



## Compression ratio

```
def compression_ratio(text, corpus):
    d = generate_hcode(build_huffman_tree(char_count(corpus)))
    len_compress = len(compress(text, d))
    len_ascii = len(ascii2bit_stream(text)) #len(text)*7
    return len_compress/len_ascii
```

- compression_ratio(text="ab", corpus="ab")
$\rightarrow 1 / 7$
- len_compress = 2
- len_ascii = 14
- compression_ratio(text="hello", corpus="a"*100 + \{ch | ch $\in$ ascii $\})$ $\rightarrow 8 / 7$
- Huffman tree for corpus "a"*100 + ascii:

For most characters: len=8

## Exam question: Alternative Trees

- Definition: Two Huffman trees are alternative if they can be generated from the same corpus but have different multiset of lengths.
- "Multiset of length" - the lengths of codewords: sorted (|H('a')|,|H('b')|, ...)
- Example: corpus = 'abcdd'
- Lengths: ( $2,2,2,2$ ) and ( $1,2,3,3$ )



## Alternative Trees

- Show two alternative trees for corpus = 'abcdee'


Lengths: $(3,3,3,3,1)$ and ( $3,3,2,2,2$ )

## Alternative Trees

- Prove/disprove: if corpus frequencies are unique, no alt. trees.
- Disprove.
- Key observation: weights are unique initially, but not necessarily throughout the algorithm execution!
corpus = 'aa bbb cccc ddddd'



## Exam Question (2019aa)

נתון קורפוס שמכיל את התווים a,b,c,d,e. שכיחויות התווים בקורפוס הינך: wa $, w_{b}, w_{c}, w_{d}, w_{e}$ בהתאמה. כל
השכיחויות שלמות וגדולות ממש מ-0. על סמך השכיחויות הללו, בונים עץ האפמן ע"י האלגוריתם שלמדנו. בכל תת-סעיף נתון מבנה של עץ ועליכם לסמן את התנאים על ערכי השכיחויות שבהכרחה יגרמו לבניית עץ בעל מבנה
 (ורק במקרה כזה) עליכם גם להסביר, ולכלול בהסבר אחד משניים: ציון תנאים אחרים עבורם בהכרח מתקבל עץ כזה, או נימוק מדוע זה לא ייתכן. שימו לב שיתכנו מספר תשובות נכונות עבור כל עץ ועליכם לסמן את כולן.

## Exam Question (2019aa)



$$
\begin{aligned}
& w_{a}>w_{b}>w_{c}>w_{d}>w_{e} \quad \text {.a } \\
& w_{a}=\frac{1}{2} w_{b}=\frac{1}{4} w_{c}=\frac{1}{8} w_{d}=\frac{1}{16} w_{e} \quad . \mathrm{b} \\
& w_{a}=w_{b}=w_{c}, \quad w_{d}>3 w_{a}, \quad w_{e}>2 w_{d} \quad . C \\
& \mathrm{w}_{\mathrm{a}}=\frac{1}{2} w_{b}=\frac{1}{3} w_{c}=\frac{1}{4} w_{d}=\frac{1}{5} w_{e} \quad . \mathrm{d} \\
& \text { אף אחד מהנ"ל לא מתאים עבור עץ זה }
\end{aligned}
$$

$$
\begin{aligned}
& \text { • א', דוגמא נגדית: משקלים 1, 2, 3, 4, } 5
\end{aligned}
$$

$a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$ • ג', בכון: סדר החיבור הוא ת d ביחד ומשקל e גדול מפעמיים a, b, c ביוון שמשקל d גדול מ-d d d d d d ולכן גדול מ-a, b, c, d ביחד. - ד', דוגמא נגדית: כמו מקודם.

## Exam Question (2019aa)



$$
\begin{aligned}
& w_{a}=\frac{1}{2} w_{b}=\frac{1}{4} w_{c}, \quad w_{d}=\frac{1}{16} w_{e} \quad \text {.a } \quad . \\
& w_{a}>w_{b}>2 w_{c}>4 w_{d}>8 w_{e} \quad . \quad \text { b } \\
& w_{a}=w_{b}=w_{c}, \quad w_{d}=w_{e}>10 w_{a} \quad \text {.c } \\
& w_{a}=w_{b}=w_{c}, \quad w_{d}=w_{e}>3 w_{a} \quad . d \\
& \text { אף אחד מהנ"ל לא מתאים עבור עץ זה }
\end{aligned}
$$

- לא ניתן לבנות את העץ הנ"ל.
- בעץ האפמן לכל צומת יש 0 או 2 בנים. - בפרט, לא ייתכן בעץ האפמן צומת עם בן אחד (כמו בעץ שבתמונה).


## Exam Question (2019aa)



$$
\begin{aligned}
& w_{a}=\frac{1}{2} w_{b}<\frac{1}{4} w_{c}=\frac{1}{4} w_{d}=\frac{1}{5} w_{e} \quad \text {.a . } \\
& \mathrm{w}_{\mathrm{a}}=\frac{1}{10} w_{b}=\frac{1}{20} w_{c}=\frac{1}{25} w_{d}=\frac{1}{30} w_{e} \quad . \mathrm{b} \\
& w_{a}>w_{b}>w_{c}, \quad w_{d}<w_{e} \quad . C \\
& w_{a}=w_{b}=w_{c}<w_{d}=w_{e} \quad . \mathrm{d} \\
& \text { אף אחד מהנ"ל לא מתאים עבור עץ זה } \\
& \text { נימוק, אם סימנתם את תשובה e: }
\end{aligned}
$$

• • א', דוגמא נגדית: משקלים 1, 2, 80, 80, 100
$w_{a}+w_{b}+w_{c}>w_{d}, w_{e}$ •', בכון: $w_{a}+w_{b}<w_{c}$ וג
• • ג', דוגמא נגדית: 1, 2, 4, 8, 16
• • •', דוגמא נגדית: 1, 1, 1, 10, 10

## Lempel-Ziv Compression

## LZ Compression

- Main idea: human text is repetitive.
- text ='abcxyabc' $\rightarrow$ code = 'abcxy', go back 5 chars and repeat 3 chars.
- Succinct: ['a', 'b', 'c', 'x', 'y', [5, 3]] $\leftarrow$ Intermediate representation
- Binary encoding:
- $c \rightarrow \operatorname{bin}(\operatorname{ord}(c))$
- [back,rep] $\rightarrow \operatorname{bin}(b a c k), \operatorname{bin}(r e p)$
- Binary decoding (From binary to intermediate rep.)
- 100110101?
- $\operatorname{chr}(100)+\operatorname{chr}(110)+\operatorname{chr}(101)$
- $\operatorname{chr}(100)+\operatorname{chr}(110101)$


## Problem

- $\operatorname{chr}(100110)+\operatorname{chr}(10)+\operatorname{chr}(1)$
- What about repetitions?


## LZ Compression - Solution

- Step 1: fix length
- char $\rightarrow$ ascii(char) in exactly 7 bits
- [back, rep] $\rightarrow 000$... bin(back) +000 ... $\operatorname{bin}(r e p)$
- Requires log |max back| + log |max rep| bits
- Step 2: add indicator to distinguish between char and repetition
- char $\rightarrow 0$ + ascii(char)
- Requires 8 bits
- $[$ back, rep $] \rightarrow \mathbb{1}+000 \ldots \operatorname{bin}(b a c k)+000 \ldots \operatorname{bin}(r e p)$
- Requires $1+\log \mid$ max back| + log |max rep| bits
- If $|\max \operatorname{back}|=|\max r e p|=n$, each rep. takes $O(\log n)$ bits
- Choose $\mid \max$ back $|,|\max r e p|=O(1)$ Why?


## LZ Compression

- We chose $\log |\max b a c k|=12, \log |\max r e p|=5$.
- Thus, we pay 18 bits for this suffix.
- A repetition of 1-2 chars is bad $(8<18)$.
- And 3? 4? ...?
- In class we saw how to find these "good" repetitions.


## More examples

- "abcab" $\rightarrow$ ['a', 'b', 'c', 'a', 'b']
- "abcabcdabc" $\rightarrow$ ['a', 'b', 'c', [3,3], 'd', [4, 3]]
- "a"*10 $\rightarrow$ ['a', [1,9] ]
- "a"*40 $\rightarrow$ ['a', [1, 31], [1, 8]]


## Compression ratio

```
def lz_ratio(text):
    inter1, bits, inter2, text2 = process(text)
    return len(bits)/(len(text)*7)
```

- Iz_ratio("hello") $\rightarrow$ 8/7
- Intermediate rep. ['h', 'e', 'l', 'l', 'o']
- len_compress $=5 \cdot 8+0 \cdot 18$
- len_ascii $=5 \cdot 7$
- Iz_ratio("hello"*2) $\rightarrow$ 0.82857...
- Intermediate rep. ['h', 'e', 'l', 'l', 'o', [5,5]]
- len_compress $=5 \cdot 8+1 \cdot 18$
- len_ascii = $10 \cdot 7$


## LZ performance

- Empirically, LZ performs very well over human text.
- In fact, still in use (WinZip, 7zip).


## Exam questions (2020bb)

- For a string $S$ let $\ell_{L Z}(S)$ denote the bit-length of the compressed string $L Z(S)$.
- We assume the presented parametrization.
- Prove/disprove: for any string $S, \ell_{L Z}(S)==\ell_{L Z}(S[::-1])$
- False! $S=$ "aaaabaaac"
- Recall: A repetition of 1-2 chars is not taken ( $8,16<18$ ).
- Key observation: the repetition "aaa" can be compressed after reading "aaaa" but not before it.


## Exam questions (2020bb)

- For a string $S$ let $\ell_{L Z}(S)$ denote the bit-length of the compressed string $L Z(S)$.
- We assume the presented parametrization.
- Prove/disprove: for any string S, $L_{L Z}(S)<L_{L Z}(S+$ "a")
- False! S = "aaaa"
- Key observation: the last "a" can be "absorbed" in a repetition.


## Exam questions (2020bb)

- For a string $S$ let $\ell_{L Z}(S)$ denote the bit-length of the compressed string $L Z(S)$.
- We assume the presented parametrization.
- Prove/disprove: for any strings $S, R L_{L Z}(S+R)=L_{L Z}(S)+L_{L Z}(R)$
- False! S = R = "aa"
- Key observation: S+R can contain a new repetition.

