

Datastructure

Data Structures

Hendrik Jan Hoogeboom

Informatica – LIACS
Universiteit Leiden

najaar 2023

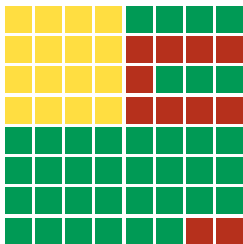
Table of Contents I

- | | | | |
|---|------------------------|----|------------------|
| 1 | Basic Data Structures | 6 | B-Trees |
| 2 | Tree Traversal | 7 | Graphs |
| 3 | Binary Search Trees | 8 | Hash Tables |
| 4 | Balancing Binary Trees | 9 | Data Compression |
| 5 | Priority Queues | 10 | Pattern Matching |

Contents




- 9 Data Compression
 - Huffman Coding
 - Ziv-Lempel-Welch
 - Burrows-Wheeler ☒

GIF & LZW

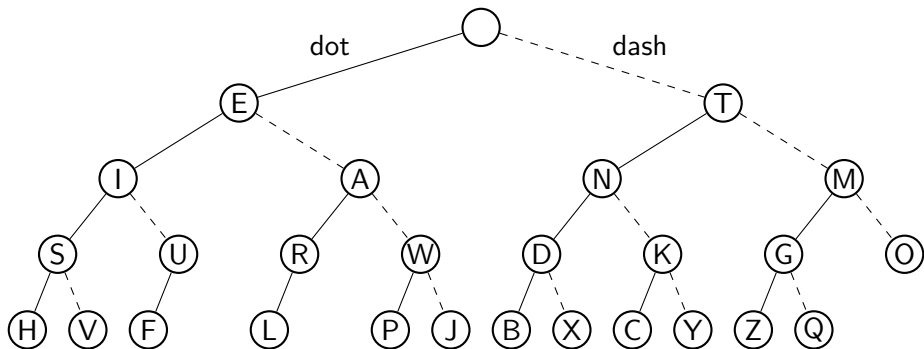


2	2	2	2	0	0	0	0
2	2	2	2	1	1	1	1
2	2	2	2	1	0	0	0
2	2	2	2	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1

colour table

0	
1	
2	

Morse ☒



byoxo

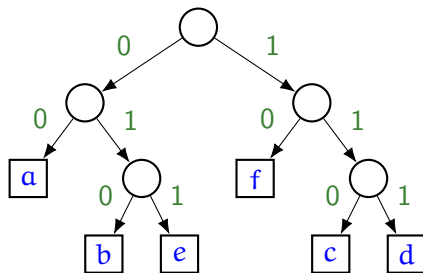
tko om bv cu 73

Are you trying to crawl out of our deal?

thanks old-man bon-voyage see-you best regards

text compression: Huffman vs. LZW

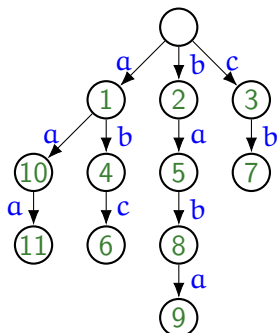
prefix code



e \mapsto 011

f \mapsto 10

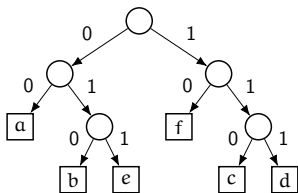
trie



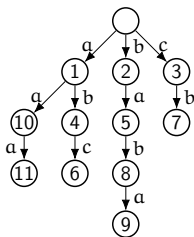
cb \mapsto 00111 7

aaa \mapsto 01011 11

Huffman vs. LZW coding



frequencies given
 single letter to variable bits
 prefix code-tree
 - letters as leafs
 - bits left/right
 store code



self learning
 variable string to fixed length
 trie
 - letters along edges
 - code in node
 decoder learns too

Contents

- 9 Data Compression
 - Huffman Coding
 - Ziv-Lempel-Welch
 - Burrows-Wheeler ☒

Huffman (1952)

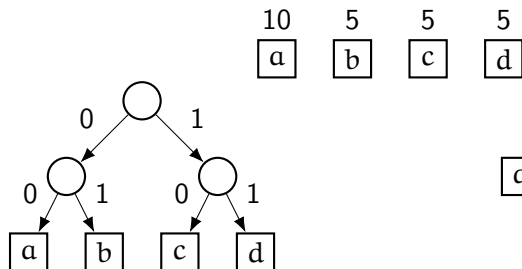
- variable length code for single letters
 $a_1, \dots, a_n \in \Sigma \mapsto w_1, \dots, w_n \in \{0, 1\}^*$
- based on character frequencies (known in advance)
 f_1, \dots, f_n
- optimal expected code length (for prefix code)
$$\sum_{i=1}^n f_i \cdot |w_i|$$
- code has to be known by decoder

Huffman

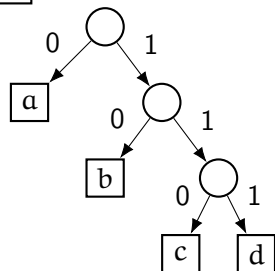
Huffman

```
// initialize:  
for each input letter a tree with that letter,  
    and its frequency  
repeat  
    take two trees of minimal frequencies  
    join these as children in a new tree,  
        with combined frequency  
until one tree left
```

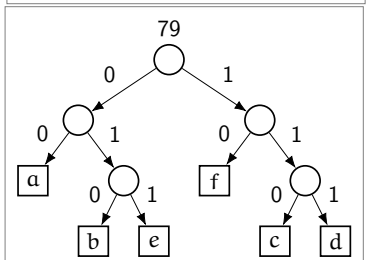
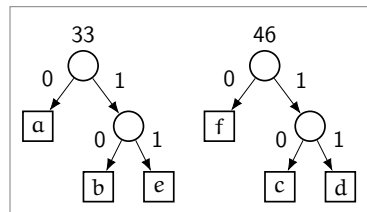
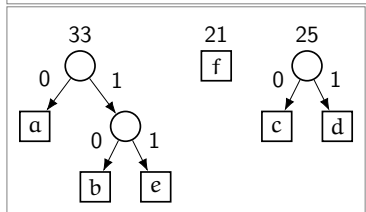
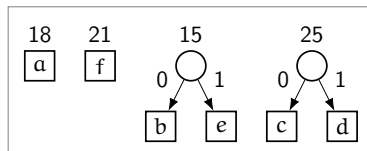
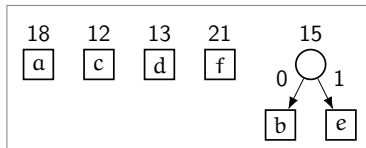
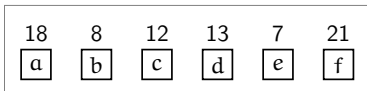
keuzes, keuzes, ...



$$2 \cdot 10 + 2 \cdot 5 + 2 \cdot 5 + 2 \cdot 5 = 50$$



$$1 \cdot 10 + 2 \cdot 5 + 3 \cdot 5 + 3 \cdot 5 = 50$$



David Albert Huffman (1925–1999)

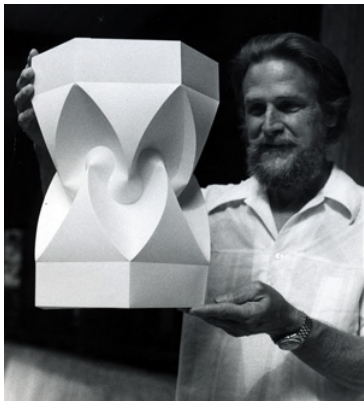


photo: 1978, UCSC
(maa.org)

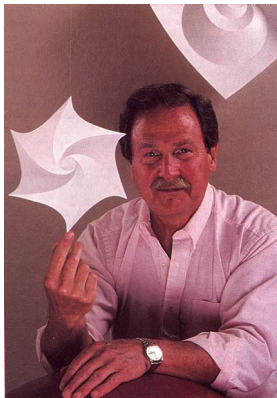


photo: 1991, Matthew Mulbry
(SciAm / huffmancoding.com)

Contents

- 9 Data Compression
 - Huffman Coding
 - Ziv-Lempel-Welch
 - Burrows-Wheeler ☒

Ziv-Lempel & Welch (1977, 1984)

- fixed length code for repeating patterns in input
 $x_1, \dots, x_n \in \Sigma^* \mapsto w_1, \dots, w_n \in \{0, 1\}^k$
- strings x_i learned while reading input
- code is also learned by decoder and does not have to be transmitted

Ziv-Lempel & Welch

ZLW compression

```
initialize dictionary with single characters
```

```
w = "";
```

```
while ( not end of input )
```

```
do read character c
```

```
    if w+c exists in the dictionary
```

```
        then w = w+c;
```

```
    else
```

```
        add w+c to the dictionary;
```

```
        output the code for w;
```

```
        w = c;
```

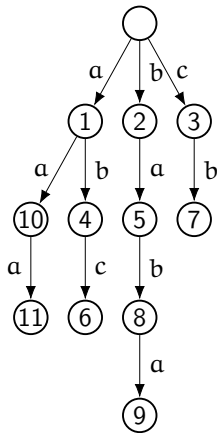
```
    fi
```

```
od
```

```
output the code for w
```

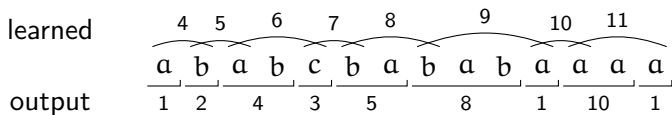
input ababcbababaa aa

w	c	dict?	out	new code
	a	✓		
a	b	×	1	4 ↦ ab
b	a	×	2	5 ↦ ba
a	b	✓		
ab	c	×	4	6 ↦ abc
c	b	×	3	7 ↦ cb
b	a	✓		
ba	b	×	5	8 ↦ bab
b	a	✓		
ba	b	✓		
bab	a	×	8	9 ↦ baba
a	a	×	1	10 ↦ aa
a	a	✓		
aa	a	×	10	11 ↦ aaa
a	⊥		1	



0	(end)
1	a
2	b
3	c
4	ab
5	ba
6	abc
7	cb
8	bab
9	baba
10	aa
11	aaa

overview



all codes used

- 1 continue "as is", fix dictionary.
- 2 add one bit to code space, and continue.
- 3 trim code tree, by removing leaves.

code too late

decoder: learns new code one step later than original encoder

here: 8 needs to be decoded, not yet in dictionary

learned											
decoded	a	b	a	b	c	b	a	?	?	?	
input	1	2	4	3	5	8			1	10	1

when new code is too late:

it is of the form 'previous output' + 'its first letter'

ZLW decompression

ZLW decompression

```
initialize dictionary with single characters

read first code in variable prev and output str(prev)
while ( not end of input )
do read w
    if w exists in the dictionary
    then output str(w)
        add to dict: str(prev) + firstchar(str(w))
    else
        // special case
        output str(prev) + firstchar(str(prev))
        add to dict: str(prev) + firstchar(str(prev))
    fi
    prev = w
od
output the code for w
```

Ziv-Lempel & Welch - decompression

Decoding 1 2 4 3 5 8 1 10 1.

code	text	new codes
		1, 2, 3 \mapsto a, b, c
1	a	
2	b	4 \mapsto ab
4	ab	5 \mapsto ba
3	c	6 \mapsto abc
5	ba	7 \mapsto cb
8	bab	8 \mapsto bab
1	a	9 \mapsto baba
10	aa	10 \mapsto aa
1	a	11 \mapsto aaa

1, 2, 3 \mapsto a, b, c initialization

we learn the new code one step late

last text + first letter

the new code is too late! is of the
form last text (ba) + first (b)

too late again

Contents

- 9 Data Compression
 - Huffman Coding
 - Ziv-Lempel-Welch
 - Burrows-Wheeler ☒

truukje

MISSISSIPPI \mapsto SSMP-PISSII = S²MP-PIS²I³

MISSISSIPPI.

rotate

1		M I S S I S S I P P I -
2		I S S I S S I P P I - M
3		S S I S S I P P I - M I
4		S I S S I P P I - M I S
5		I S S I P P I - M I S S
6		S S I P P I - M I S S I
7		S I P P I - M I S S I S
8		I P P I - M I S S I S S
9		P P I - M I S S I S S I
10		P I - M I S S I S S I P
11		I - M I S S I S S I P P
12		- M I S S I S S I P P I

alphabetize, last column

8		I P P I - M I S S I S S
5		I S S I P P I - M I S S
2		I S S I S S I P P I - M
11		I - M I S S I S S I P P
1		M I S S I S S I P P I -
10		P I - M I S S I S S I P
9		P P I - M I S S I S S I
7		S I P P I - M I S S I S
4		S I S S I P P I - M I S
3		S S I S S I P P I - M I
6		S S I P P I - M I S S I
12		- M I S S I S S I P P I

decode

12 1

S₁ I₁S₂ I₂M₁ I₃P₁ I₄- M₁P₂ P₁I₁ P₂S₃ S₁S₄ S₂I₂ S₃I₃ S₄I₄ -- M₁ I₃ S₄ S₂ I₂ S₃ S₁ I₁ P₂ P₁ I₄

end.