Datastructuren

Data Structures

Hendrik Jan Hoogeboom
Mark van den Bergh

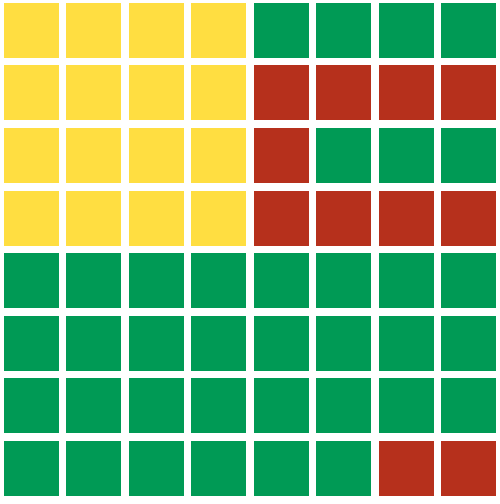
Informatica – LIACS
Universiteit Leiden

najaar 2024

Table of Contents I

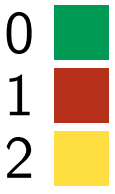
Contents

GIF & LZW

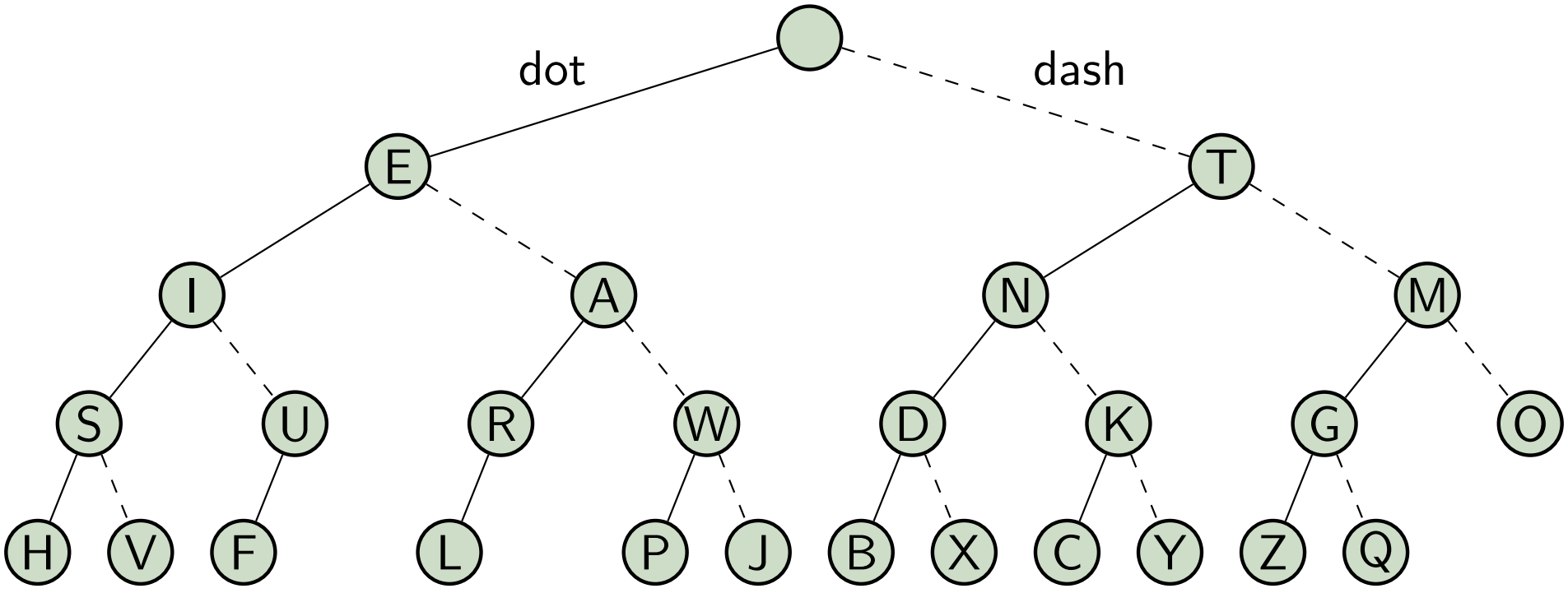


2	2	2	2	0	0	0	0
2	2	2	2	1	1	1	1
2	2	2	2	1	0	0	0
2	2	2	2	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1

colour table



Morse ☒

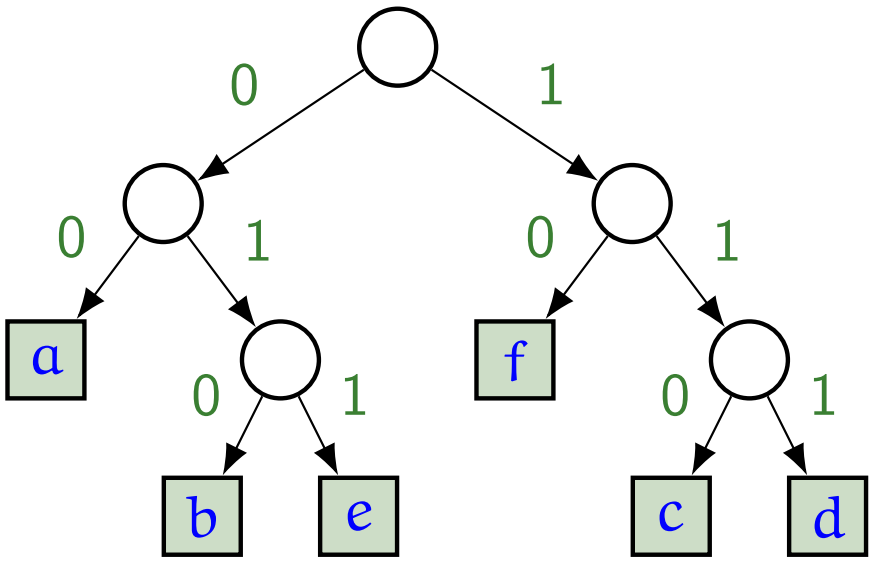


byoxo — Are you trying to crawl out of our deal?

tks om bv cu 73 — thanks old-man bon-voyage see-you best regards

text compression: Huffman vs. LZW

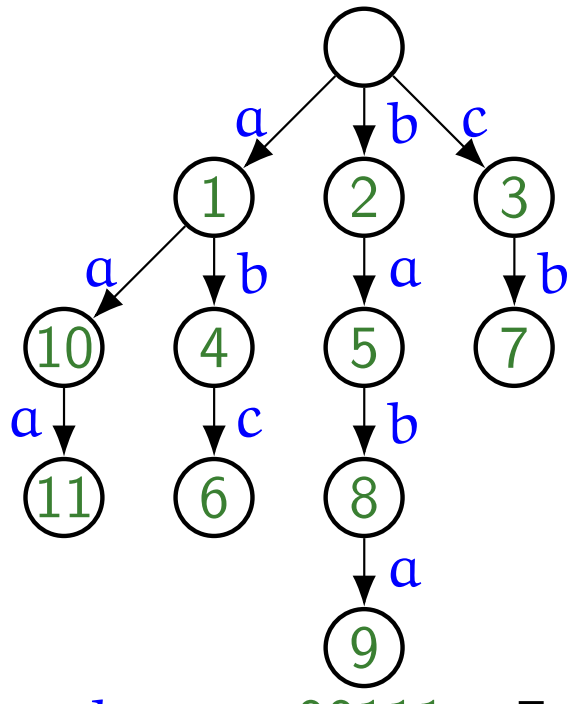
prefix code



$e \mapsto 011$

$f \mapsto 10$

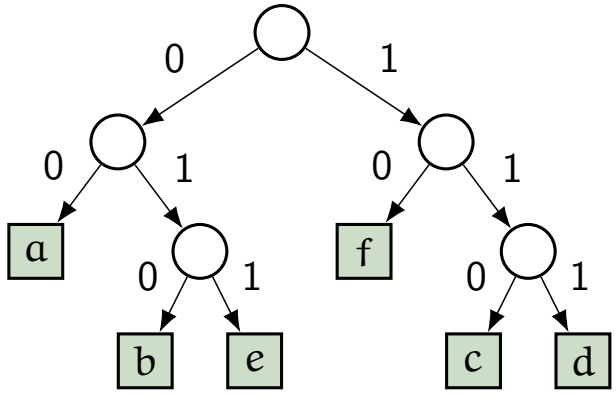
trie

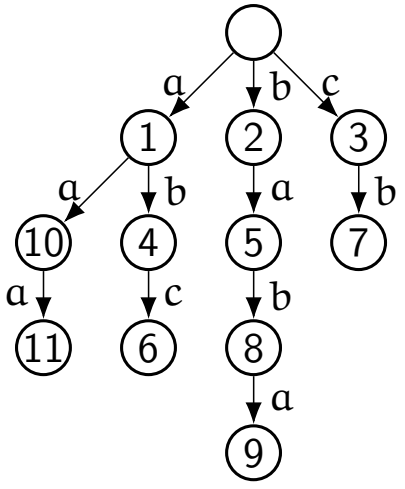


$cb \mapsto 00111$ 7

$aaa \mapsto 01011$ 11

Huffman vs. LZW coding





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

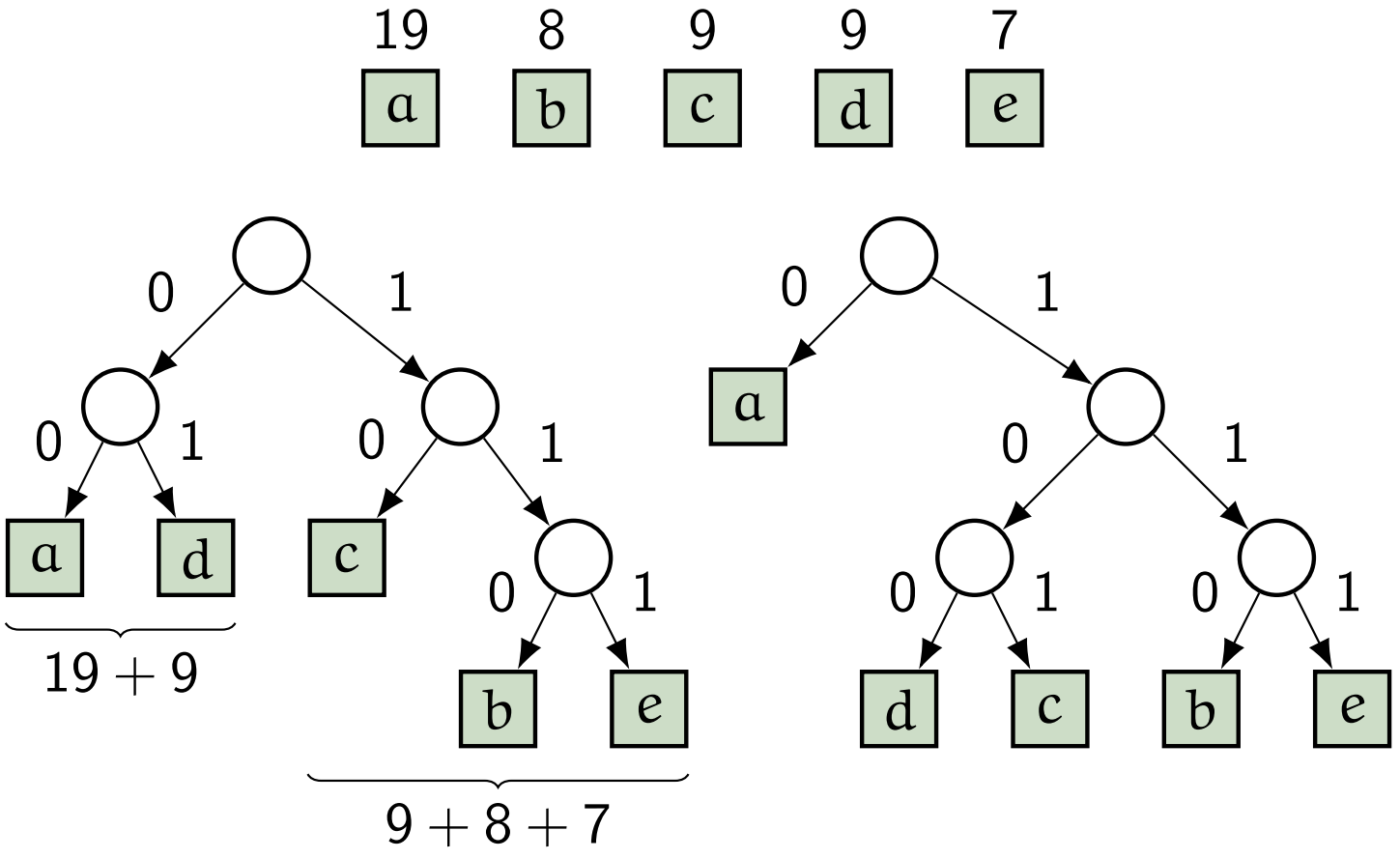
self learning
variable string to fixed length
trie
- letters along edges
- code in node
decoder learns too

Contents

Huffman (1952)

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

Shannon-Fano



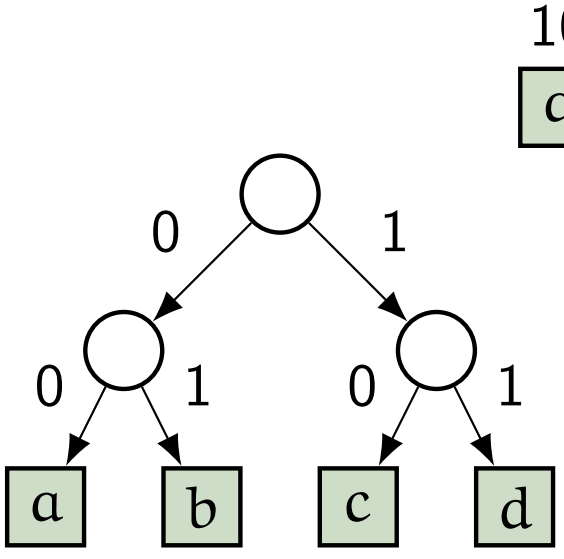
$$2 \cdot (19 + 9 + 9) + 3 \cdot (8 + 7) = 119 \text{ vs. } 1 \cdot 19 + 3 \cdot (9 + 9 + 8 + 7) = 118$$

Huffman

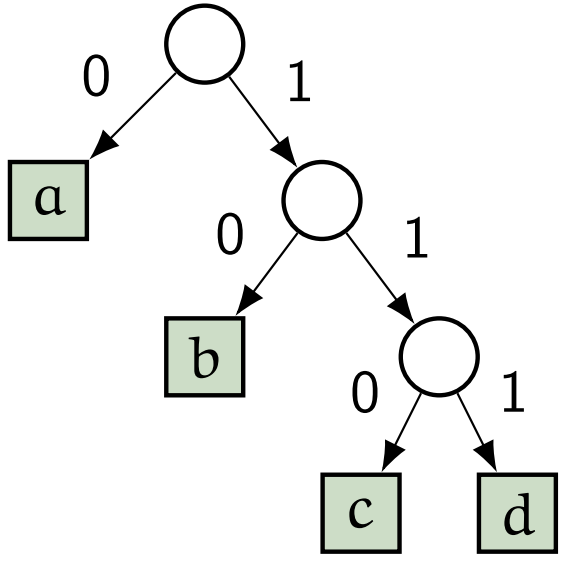
Huffman

```
// 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
```

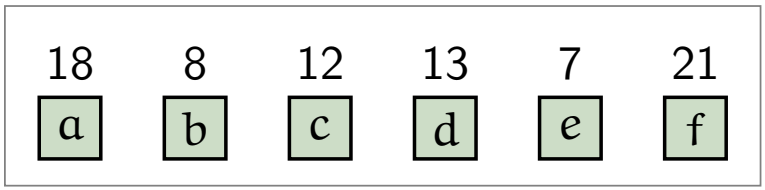
keuzes, keuzes, . . .

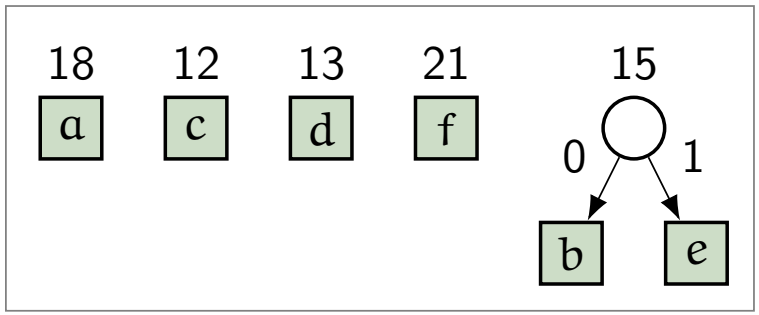


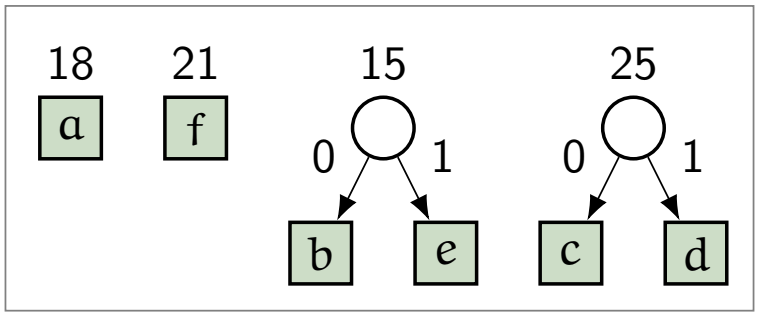
$2 \cdot 10 + 2 \cdot 5 + 2 \cdot 5 + 2 \cdot 5 = 50$

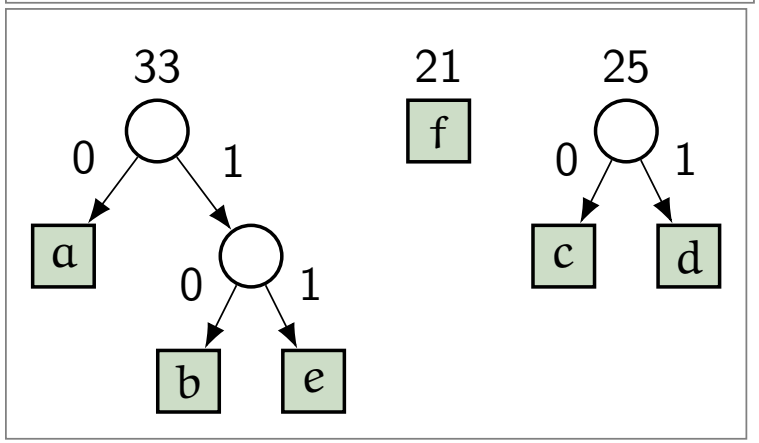


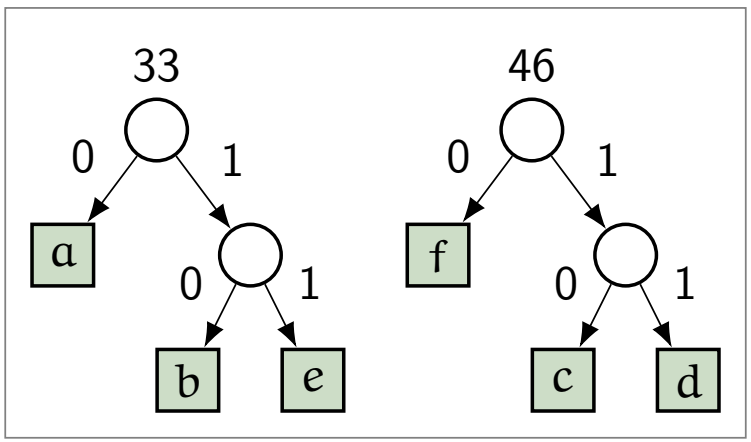
$1 \cdot 10 + 2 \cdot 5 + 3 \cdot 5 + 3 \cdot 5 = 50$

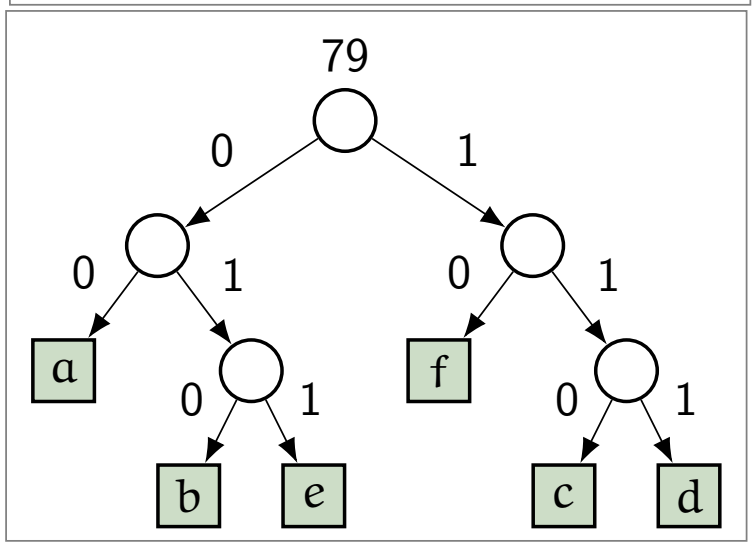

18 8 12 13 7 21
a b c d e f


18 12 13 21 15
a c d f
0 1
b e

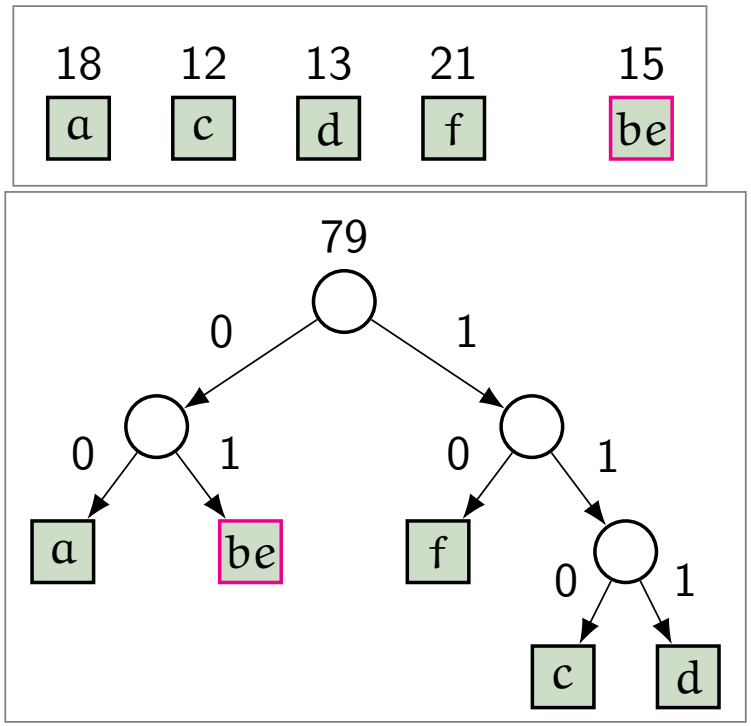

18 21 15 25
a f
0 1
b e
0 1
c d

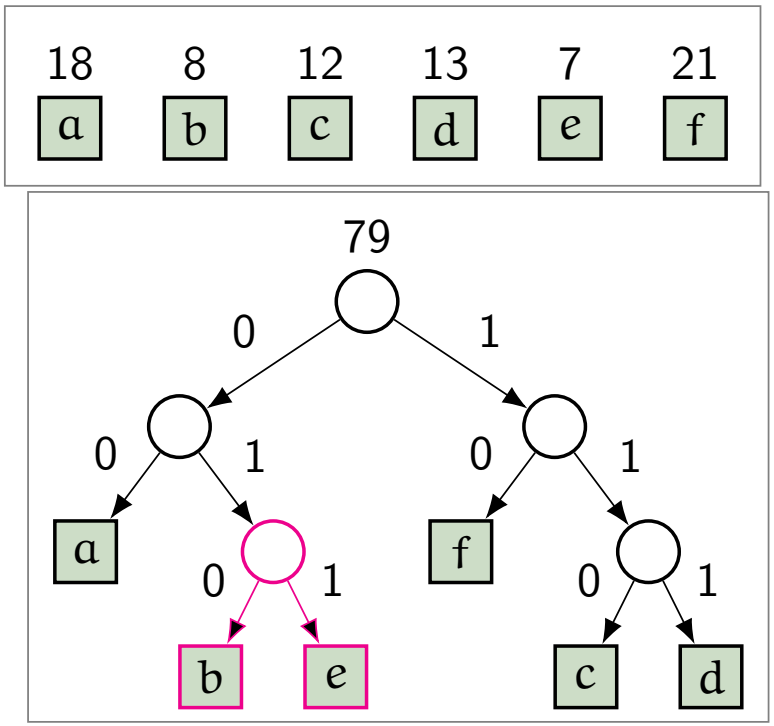

33 21 25
0 1
a
0 1
b e
f
0 1
c d


33 46
0 1
a
0 1
b e
0 1
f
0 1
c d


79
0 1
0 1
a
0 1
b e
0 1
f
0 1
c d

recursive / correctness





Contents

Ziv-Lempel & Welch (1977, 1984)

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

Ziv-Lempel & Welch

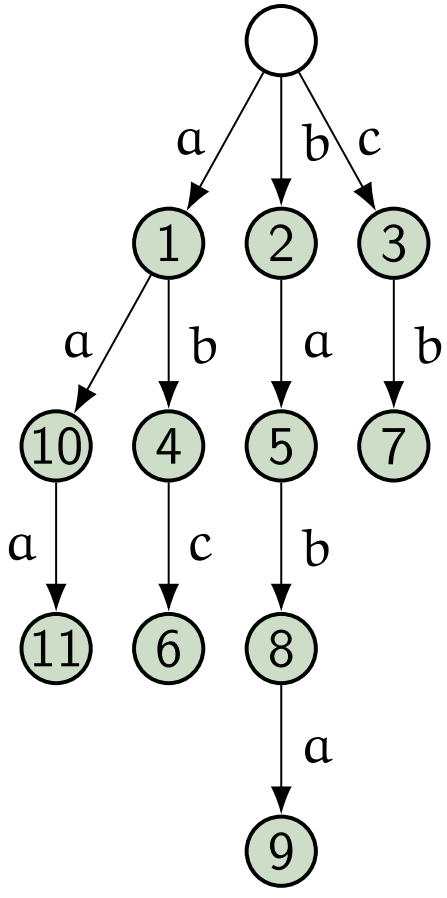
ZLW compression

```
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
```

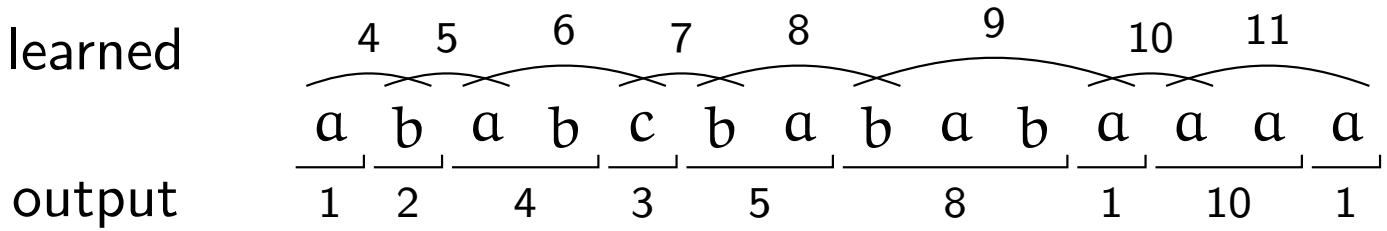
input abab cbab abaa aa

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



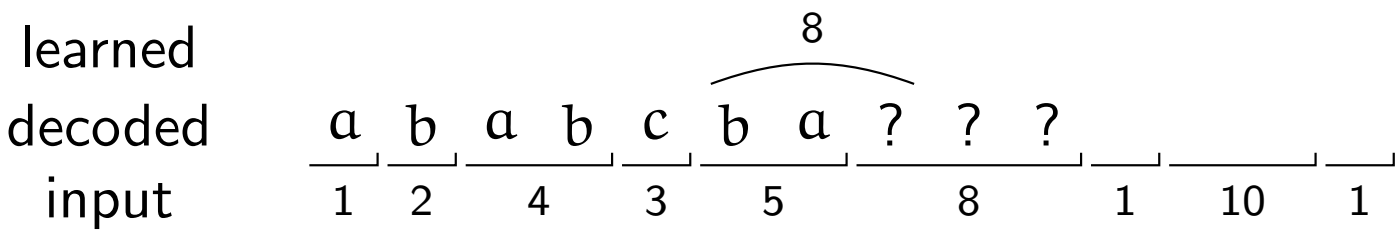
all codes used

1. continue “as is”, freeze 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



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 & Welsh - decompression

Decoding 1 2 4 3 5 8 1 10 1.

code	text	new codes	
		$1, 2, 3 \mapsto a, b, c$	initialization
1	a		we learn the new code one step late
2	b	$4 \mapsto ab$	last text + first letter
4	ab	$5 \mapsto ba$	
3	c	$6 \mapsto abc$	
5	ba	$7 \mapsto cb$	
8	bab	$8 \mapsto bab$	the new code is too late! is of the form last text (ba) + first (b)
1	a	$9 \mapsto baba$	
10	aa	$10 \mapsto aa$	too late again
1	a	$11 \mapsto aaa$	

Contents

truukje

$$\text{MISSISSIPPI} \quad \mapsto \quad \text{SSMP-PISSIII} = \text{S}^2\text{MP-PIS}^2\text{I}^3$$

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											
S_1	I_1											
S_2	I_2											
M_1	I_3											
P_1	I_4											
-	M_1											
P_2	P_1	-	M_1	I_3	S_4	S_2	I_2	S_3	S_1	I_1	P_2	P_1	I_4
I_1	P_2											
S_3	S_1											
S_4	S_2											
I_2	S_3											
I_3	S_4											
I_4	-											

end.