afcs
ALGEBRAIC FOUNDATIONS of COMPUTER SCIENCE
lectures by Ferucio Laurentiu Tiplea

Semigroups and Monoids

Huffman Codes: Examples

Prof.dr. Ferucio Laurențiu Țiplea

Spring 2022

Department of Computer Science
"Alexandru Ioan Cuza" University of Iași
Iași 700506, Romania

e-mail: ferucio.tiplea@uaic.ro

Outline

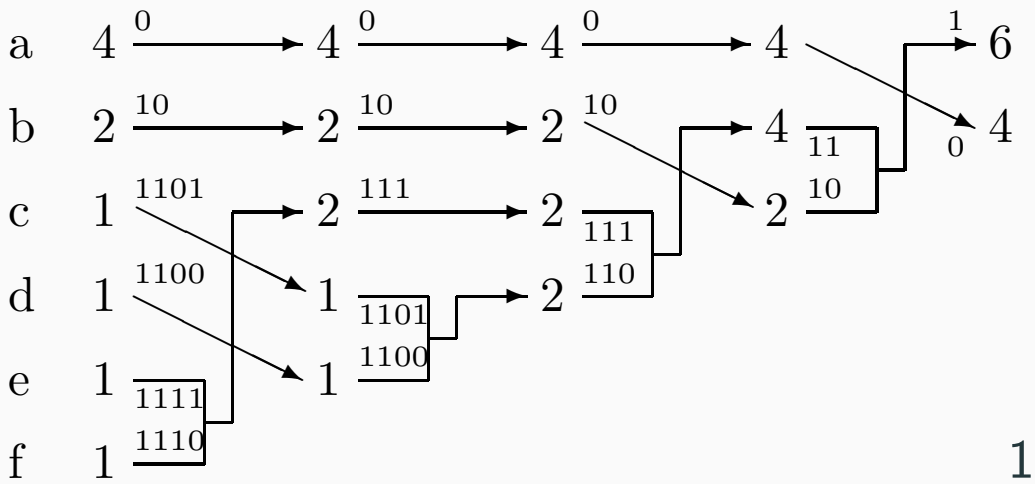
Introduction

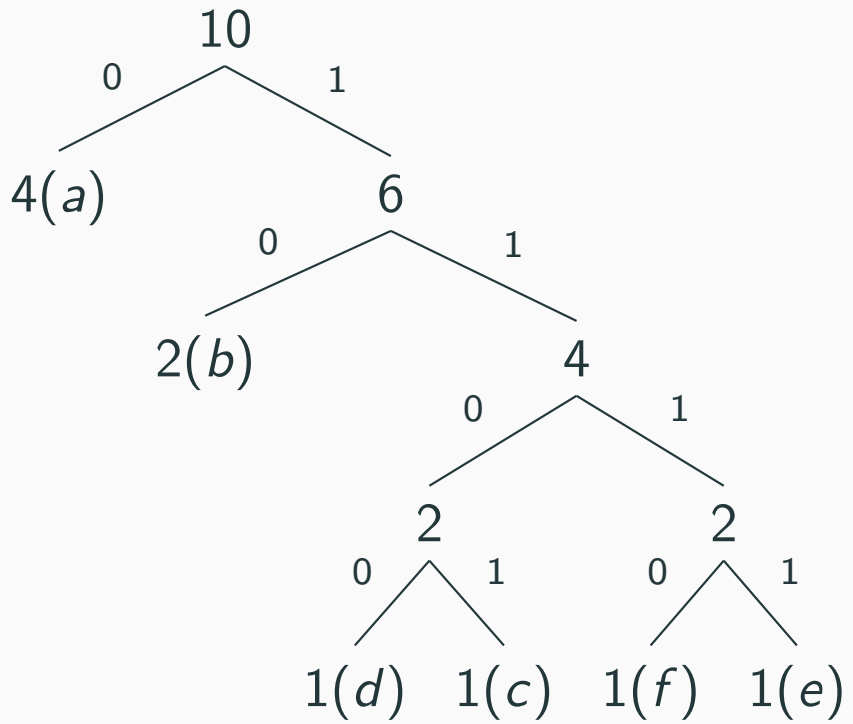
About this slide package

This slide package aims to help you, by a few examples, in understanding Huffman codes. Use them together with the course lesson on Huffman codes!

Reducing the information sources in the lesson from the course was done by a tree-like diagram horizontally written. However, we prefer to write it vertically on the slides in this package (see the next slide as an example).

Equivalent tree representations of Huffman codes





Examples of static Huffman encoding and decoding

Static Huffman codes: encoding

Encoding procedure:

1. Parse the text;
2. Collect symbols' frequencies;
3. Reduce the information source in successive steps;
4. Compute the H_code;
5. Encode the text and submit

$$\langle \text{ H\_code\_info} \# \text{encoded\_text} \rangle$$

H_code_info should give sufficient information so that the decoder can recover the H_code and decode the encoded text. Usually, this information may be:

- Symbols' frequencies (this is easy because the frequencies are integers – it will add just a few hundred bytes to the encoded text);
- The H_code itself (this can be inconvenient because the code words have variable lengths);
- The H_tree (this can be an acceptable solution with a compact representation of the tree).

Static Huffman codes: encoding

Remarks on reordering the source:

1. Each source reduction step requires ordering it! In practice, two main techniques are employed:
   - using a priority queue – complexity $\mathcal{O}(n \log n)$;
   - using two queues – complexity $\mathcal{O}(n)$ if the initial source is ordered.

   In both cases, n is the number of symbols in the text;
2. The order of symbols in the text is irrelevant to computing the H_code!

Two examples of computing the Huffman code are in order, one based on a priority queue and the other based on two queues. The notation is self-explanatory. For example, $9(a)$ means 9 occurrences of the symbol a.

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using a priority queue to reduce the source

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using a priority queue to reduce the source

$9(b)$ $4(c)$ $4(a)$ $3(e)$ $2(d)$

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using a priority queue to reduce the source

$9(b)$ $4(c)$ $4(a)$ $3(e)$ $2(d)$

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using a priority queue to reduce the source

$9(b)\ 4(c)\ 4(a)\ 3(e)\ 2(d)$

$9(b)\ 5(d,e)\ 4(c)\ 4(a)$

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using a priority queue to reduce the source

$9(b)\ 4(c)\ 4(a)\ 3(e)\ 2(d)$

$9(b)\ 5(d,e)\ 4(c)\ 4(a)$

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using a priority queue to reduce the source

$9(b)\ 4(c)\ 4(a)\ 3(e)\ 2(d)$

$9(b)\ 5(d,e)\ 4(c)\ 4(a)$

$9(b)\ 8(c,a)\ 5(d,e)$

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using a priority queue to reduce the source

$9(b)\ 4(c)\ 4(a)\ 3(e)\ 2(d)$

$9(b)\ 5(d,e)\ 4(c)\ 4(a)$

$9(b)\ 8(c,a)\ 5(d,e)$

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using a priority queue to reduce the source

$9(b)\ 4(c)\ 4(a)\ 3(e)\ 2(d)$

$9(b)\ 5(d,e)\ 4(c)\ 4(a)$

$9(b)\ 8(c,a)\ 5(d,e)$

$13((d,e),(c,a))\ 9(b)$

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using a priority queue to reduce the source

$9(b)\ 4(c)\ 4(a)\ 3(e)\ 2(d)$

$9(b)\ 5(d,e)\ 4(c)\ 4(a)$

$9(b)\ 8(c,a)\ 5(d,e)$

$13((d,e),(c,a))\ 9(b)$

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using a priority queue to reduce the source

$9(b)\ 4(c)\ 4(a)\ 3(e)\ 2(d)$

$9(b)\ 5(d,e)\ 4(c)\ 4(a)$

$9(b)\ 8(c,a)\ 5(d,e)$

$13((d,e),(c,a))\ 9(b)$

$22(b,((d,e),(c,a))$

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using a priority queue to reduce the source

$9(b)\ 4(c)\ 4(a)\ 3(e)\ 2(d)$

$9(b)\ 5(d,e)\ 4(c)\ 4(a)$

$9(b)\ 8(c,a)\ 5(d,e)$

$13((d,e),(c,a))\ 9(b)$

$22(b,((d,e),(c,a))$

$$\dfrac{(\underset{0}{b},\ \dfrac{(\underset{0}{d},\underset{1}{e})}{1},\ \dfrac{(\underset{0}{c},\underset{1}{a})}{1})}{root}$$

Computing the Huffman code with a priority queue

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using a priority queue to reduce the source

$9(b)\ 4(c)\ 4(a)\ 3(e)\ 2(d)$

$9(b)\ 5(d,e)\ 4(c)\ 4(a)$

$9(b)\ 8(c,a)\ 5(d,e)$

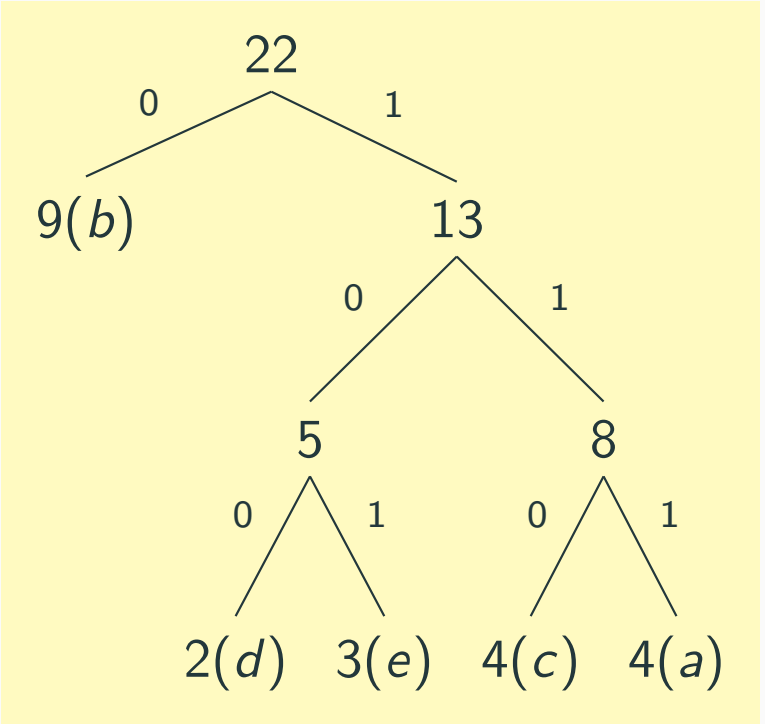
$13((d,e),(c,a))\ 9(b)$

$22(b,((d,e),(c,a))$

$$(\underset{0}{\underline{b}},((\underset{0}{\underline{d}},\underset{1}{\underline{e}}),(\underset{0}{\underline{c}},\underset{1}{\underline{a}})))$$

$$\dfrac{\dfrac{\dfrac{\underline{b}}{0}\quad\dfrac{\underline{d}\ \underline{e}}{0\ 1}}{0}\quad\dfrac{\dfrac{\underline{c}\ \underline{a}}{0\ 1}}{1}}{1}$$

root



To encode the text, parse it and replace each symbol by its code as shown in the tree!

Computing the Huffman code with two queues

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Computing the Huffman code with two queues

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using two queues to reduce the source

Computing the Huffman code with two queues

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using two queues to reduce the source

L1: $2(d)$ $3(e)$ $4(c)$ $4(a)$ $9(b)$

L2: $\emptyset$

Computing the Huffman code with two queues

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using two queues to reduce the source

L1: ~~$2(d)$ $3(e)$~~ $4(c)$ $4(a)$ $9(b)$

L2: $5(d, e)$

Computing the Huffman code with two queues

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using two queues to reduce the source

L1: ~~$2(d)$~~ ~~$3(e)$~~ $4(c)$ ~~$4(a)$~~ $9(b)$

L2: $5(d, e)$ $8(c, a)$

Computing the Huffman code with two queues

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using two queues to reduce the source

L1: ~~$2(d)$ $3(e)$~~ $4(c)$ ~~$4(a)$~~ $9(b)$

L2: ~~$5(d, e)$ $8(c, a)$~~ $13((d, e), (c, a))$

Computing the Huffman code with two queues

Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using two queues to reduce the source

L1: ~~$2(d)$~~ ~~$3(e)$~~ $4(c)$ ~~$4(a)$~~ ~~$9(b)$~~

L2: ~~$5(d, e)$~~ ~~$8(c, a)$~~ ~~$13((d, e), (c, a))$~~

$22(b, ((d, e), (c, a)))$

Computing the Huffman code with two queues

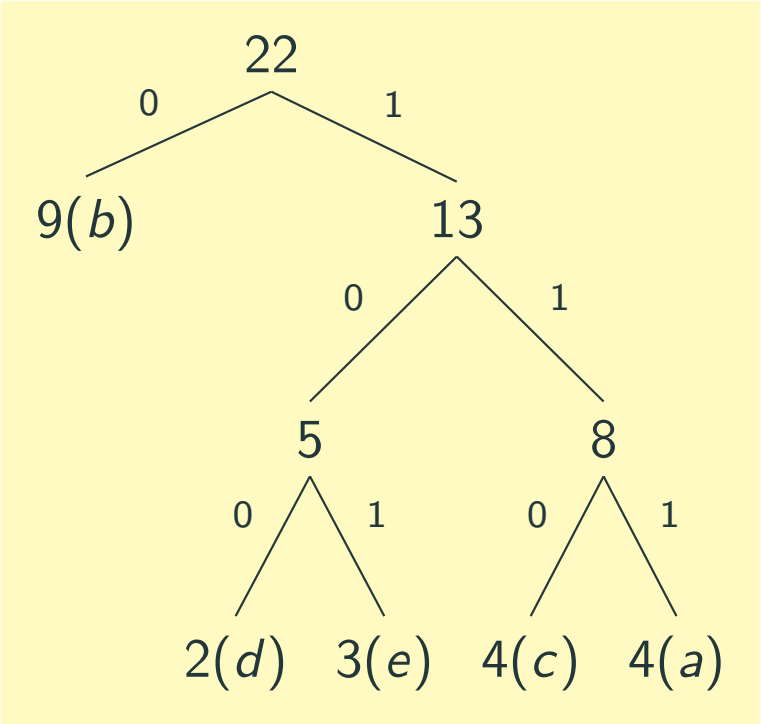
Any text with the symbols' frequencies: $9(b)$, $4(c)$, $4(a)$, $3(e)$, $2(d)$

Using two queues to reduce the source

L1: ~~$2(d)$ $3(e)$~~ $4(c)$ ~~$4(a)$~~ ~~$9(b)$~~

L2: ~~$5(d,e)$ $8(c,a)$~~ ~~$13((d,e),(c,a))$~~

$22(b,((d,e),(c,a)))$



To encode the text, parse it and replace each symbol by its code as shown in the tree!

Static Huffman decoding

Decoding procedure:

1. recover the H_code (H_tree) from H_code_info;
2. decode – this is a straightforward process. Start from the tree's root and move down according to the bit read from the code sequence until a leaf is met. Decode then the bit sequence by the leaf. Start the process again at the root with the next bit.

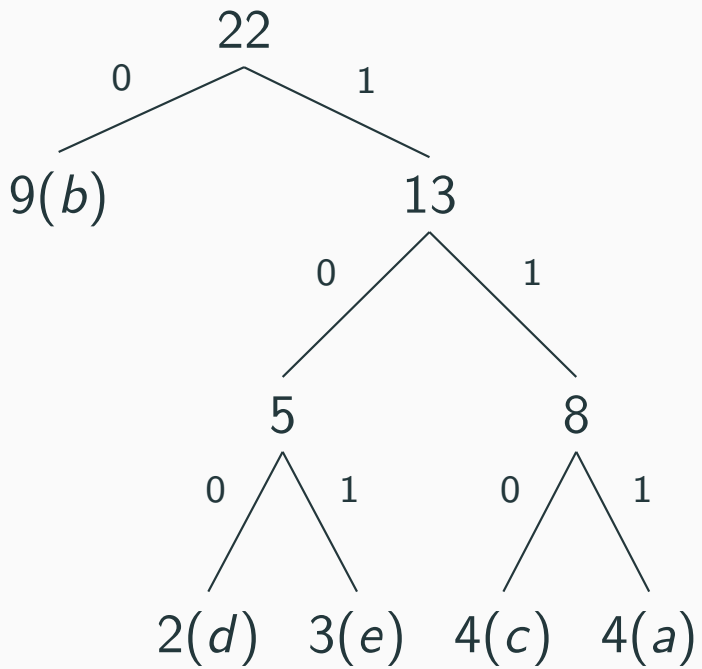
Static Huffman decoding

$$\langle\ \text{H\_code\_info} \,\#\, \text{encoded\_text}\ \rangle$$

Static Huffman decoding

$$\langle \text{ H\_code\_info } \# \text{ encoded\_text } \rangle$$

H_code_info



code sequence

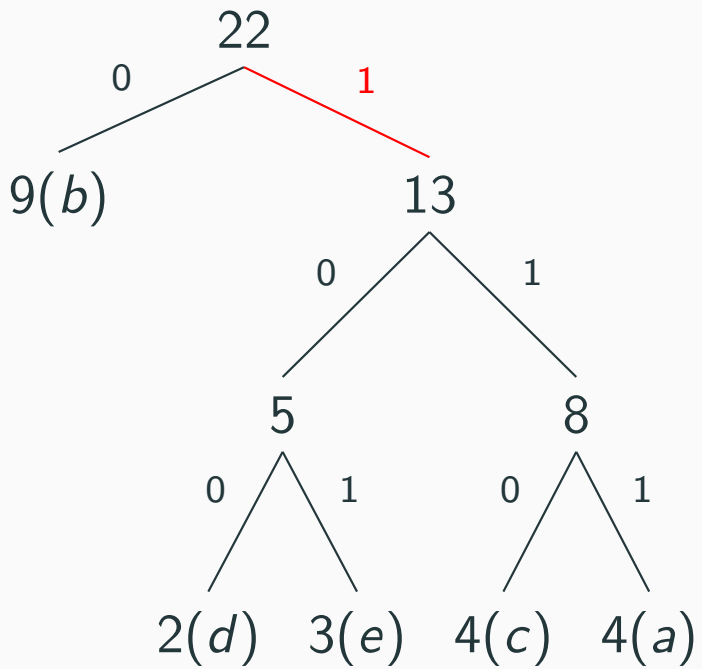
$100111110\cdots$

text:

Static Huffman decoding

$$\langle \text{ H\_code\_info } \# \text{ encoded\_text } \rangle$$

H_code_info



code sequence

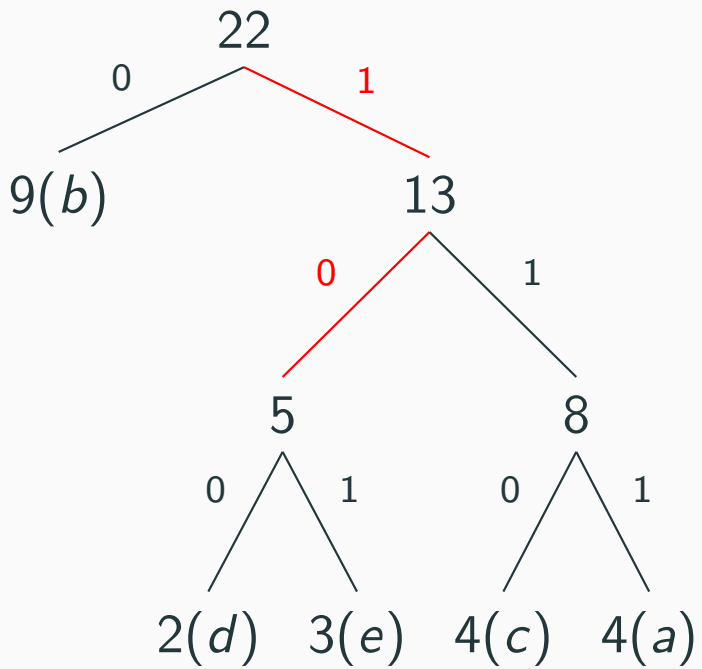
$100111110\cdots$

text:

Static Huffman decoding

$$\langle \text{ H\_code\_info } \# \text{ encoded\_text } \rangle$$

H_code_info



code sequence

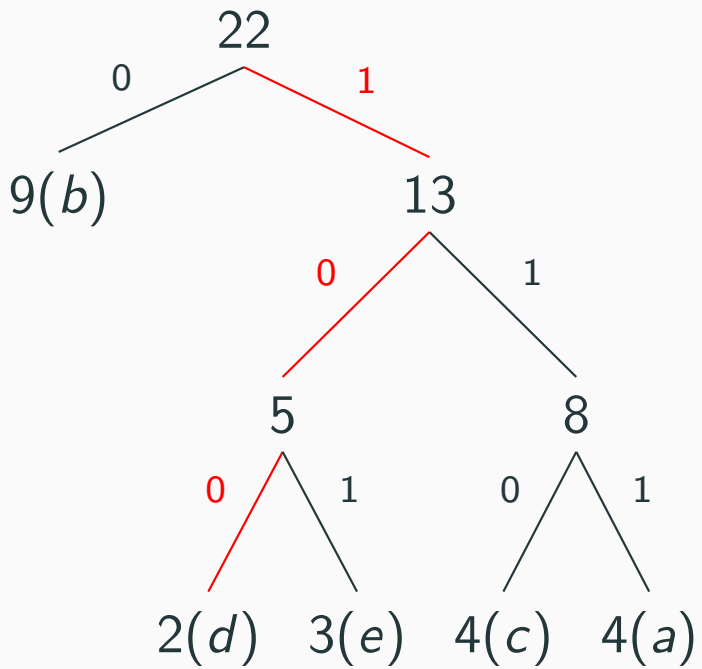
$10\,0111110\cdots$

text:

Static Huffman decoding

$$\langle\ \text{H\_code\_info}\ \#\ \text{encoded\_text}\ \rangle$$

H_code_info



code sequence

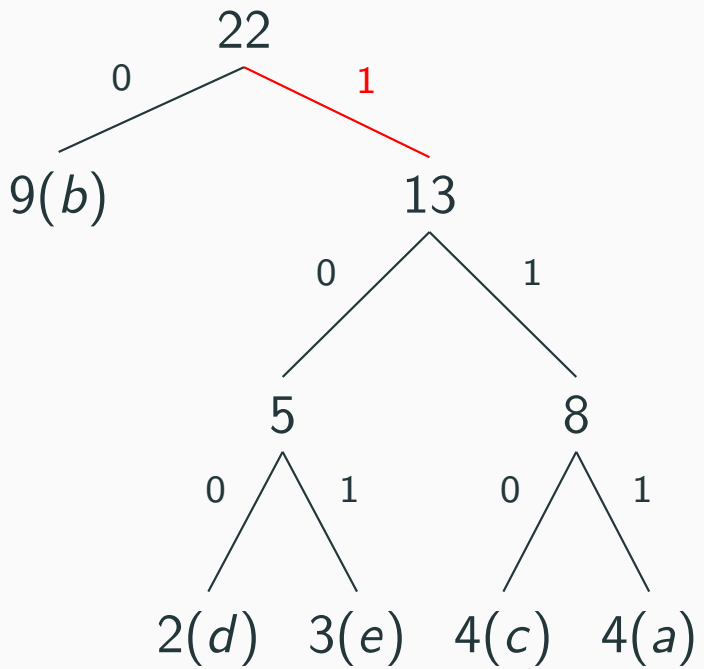
$100111110\cdots$

text: d

Static Huffman decoding

$$\langle\ \text{H\_code\_info}\ \#\ \text{encoded\_text}\ \rangle$$

H_code_info



code sequence

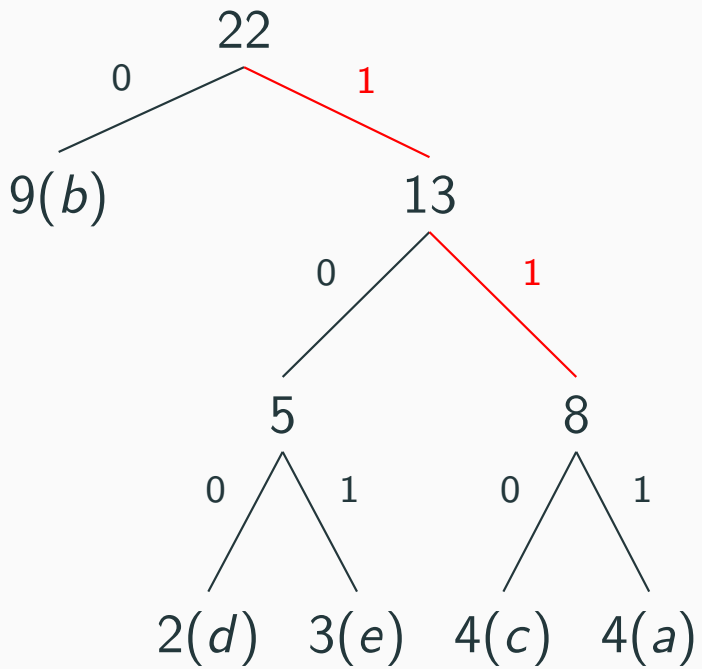
$100111110\cdots$

text: d

Static Huffman decoding

$$\langle\ \text{H\_code\_info}\ \#\ \text{encoded\_text}\ \rangle$$

H_code_info



code sequence

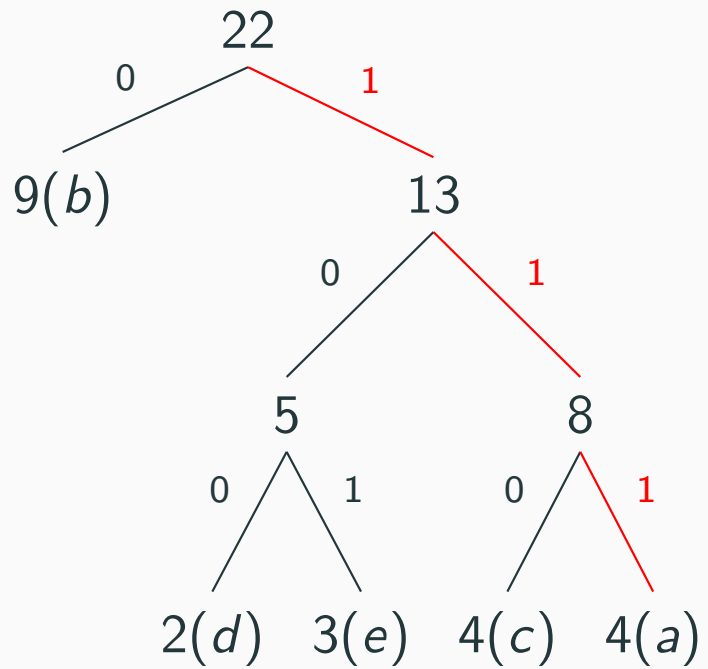
$100111110\cdots$

text: d

Static Huffman decoding

$$\langle\ \text{H\_code\_info}\ \#\ \text{encoded\_text}\ \rangle$$

H_code_info



code sequence

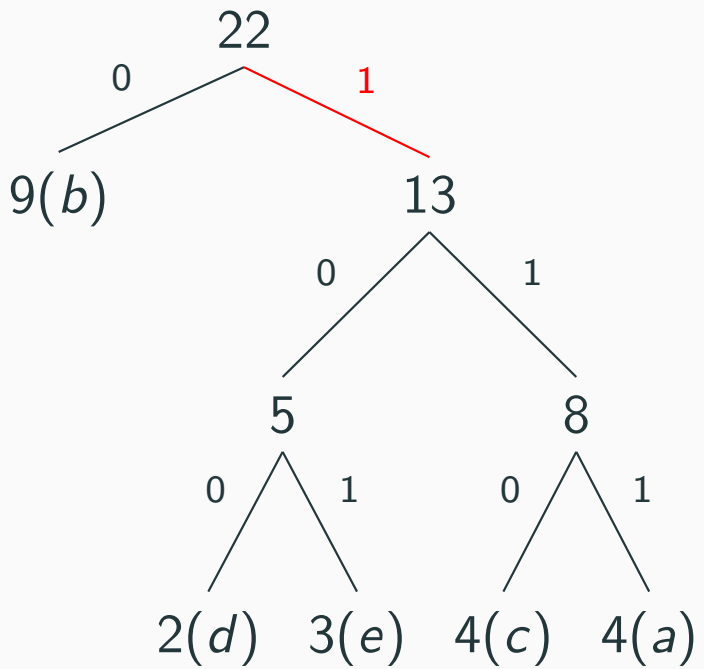
$100111110\cdots$

text: da

Static Huffman decoding

$$\langle\ \text{H\_code\_info}\ \#\ \text{encoded\_text}\ \rangle$$

H_code_info



code sequence

$100111110\cdots$

text: da

... and so on!

Examples of dynamic Huffman encoding and decoding

Dynamic Huffman codes: encoding and decoding

Encoding procedure:

1. choose an initial Huffman code (tree), denoted h_0, such as:
   - a Huffman code of the entire alphabet (each occurrence of each symbol will be encoded);
   - a Huffman code with just one "empty node" (the first occurrence of each symbol will be unencoded);
2. parse the text and for each symbol, do the following:
   - if the current symbol is the $(i+1)$st symbol in the text, where $i \geq 0$, encode it by h_i;
   - update h_i with the new symbol. The result is h_{i+1}.

Decoding procedure: the decoder simply mirrors the encoder steps.

The fundamental issue is how to update the Huffman tree!

Faller-Gallager-Knuth's algorithm

1. Main idea: Faller, 1973 [2];
2. Sibling property: Gallager, 1978 [3];
3. Other improvements: Knuth, 1985 [4].

The final algorithm is known as the FGK algorithm.

The initial Huffman tree:

1. Consists of only one node whose weight will always be 0. The code of this node (which is initially the empty code) is called the escape code.
2. This node is used to insert a new symbol (which is not yet in the tree). The first occurrence of this symbol is encoded by the escape code followed by the unencoded symbol.

The sibling transformation

The sibling transformation applied to symbol a and tree h_i consists of:

1. compare a to its successors in the tree (from left to right and from bottom to top). If the immediate successor has frequency $k+1$ or greater, where k is the frequency of a in h_i, then the nodes are still in sorted order and there is no need to change anything. Otherwise, a should be swapped with the last successor which has frequency k or smaller (except that a should not be swapped with its parent);
2. increment the frequency of a (from k to $k+1$);
3. if a is the root, the loop halts; otherwise, the loop repeats with the parent of a.

The FGK algorithm: encoding

Text to be encoded

Huffman code

Text: `abccd`

Encoded text:

The FGK algorithm: encoding

Read the first character

Text: abccd

Encoded text:

Current Huffman code

0

The FGK algorithm: encoding

Encode the first character

Text: abccd

Encoded text: a

Current Huffman code

0

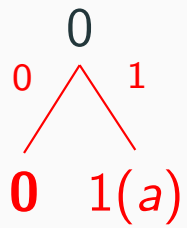
The FGK algorithm: encoding

First character encoded

Text: abccd

Encoded text: a

Update Huffman code



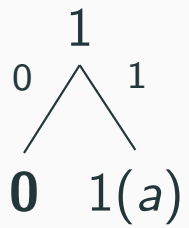
The FGK algorithm: encoding

First character encoded

Text: abccd

Encoded text: a

Updated Huffman code



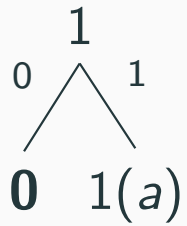
The FGK algorithm: encoding

Read the second character

Text: abccd

Encoded text: a

Current Huffman code



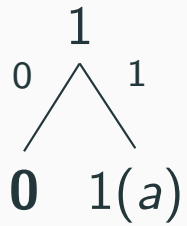
The FGK algorithm: encoding

Encode the second character

Text: abccd

Encoded text: a0b

Current Huffman code



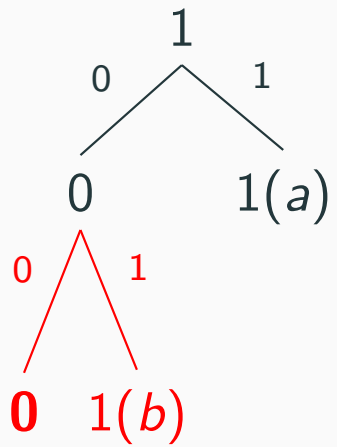
The FGK algorithm: encoding

Second character encoded

Text: abccd

Encoded text: a0b

Update Huffman code



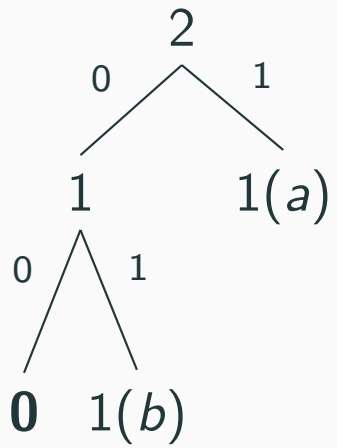
The FGK algorithm: encoding

Second character encoded

Text: abccd

Encoded text: a0b

Updated Huffman code



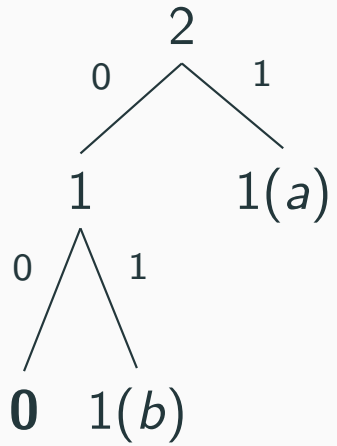
The FGK algorithm: encoding

Read the third character

Text: abccd

Encoded text: a0b

Current Huffman code



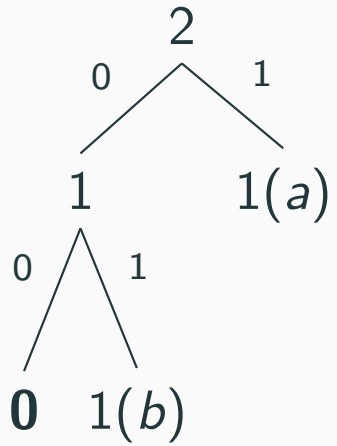
The FGK algorithm: encoding

Encode the third character

Text: abccd

Encoded text: a0b00c

Current Huffman code



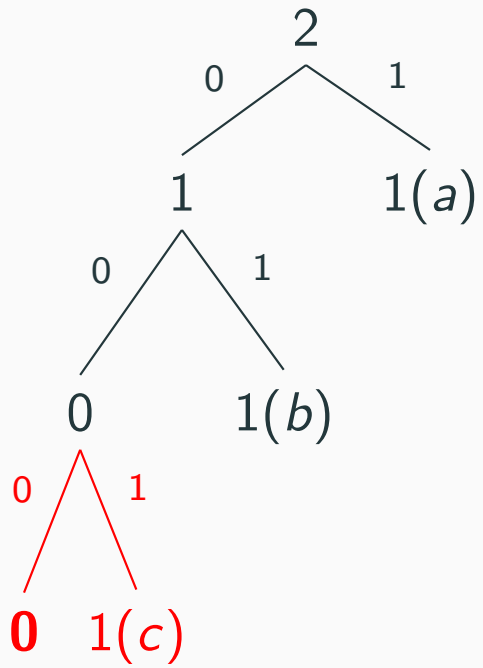
The FGK algorithm: encoding

Third character encoded

Text: abccd

Encoded text: a0b00c

Update Huffman code



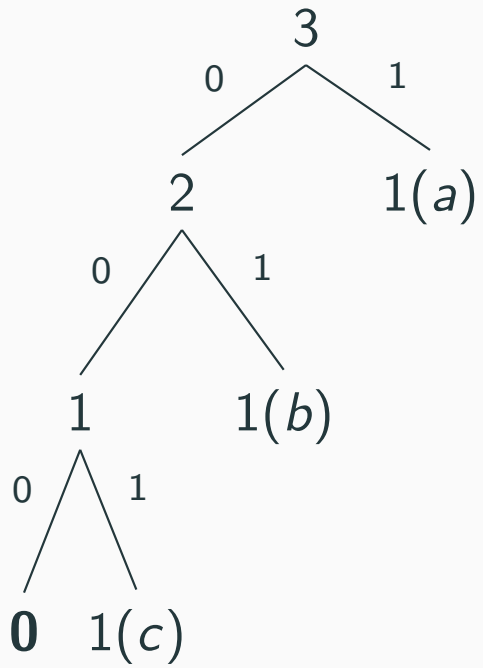
The FGK algorithm: encoding

Third character encoded

Text: abccd

Encoded text: a0b00c

Update Huffman code



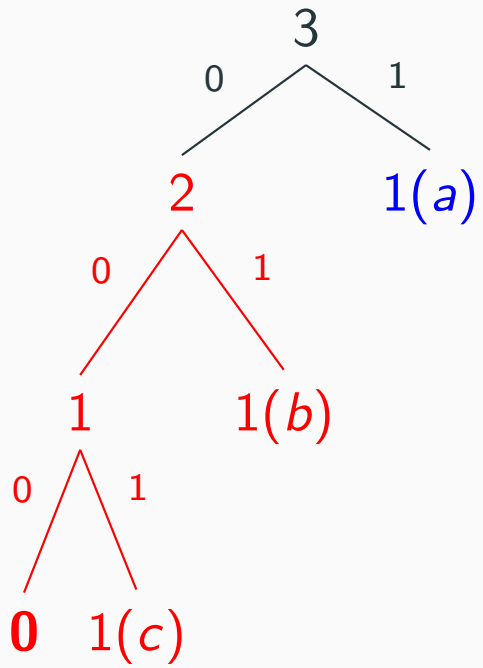
The FGK algorithm: encoding

Third character encoded

Text: abccd

Encoded text: a0b00c

Update Huffman code



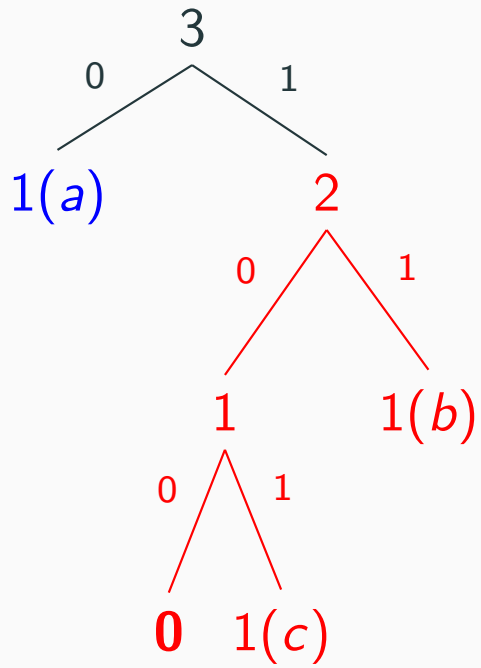
The FGK algorithm: encoding

Third character encoded

Text: abccd

Encoded text: a0b00c

Update Huffman code



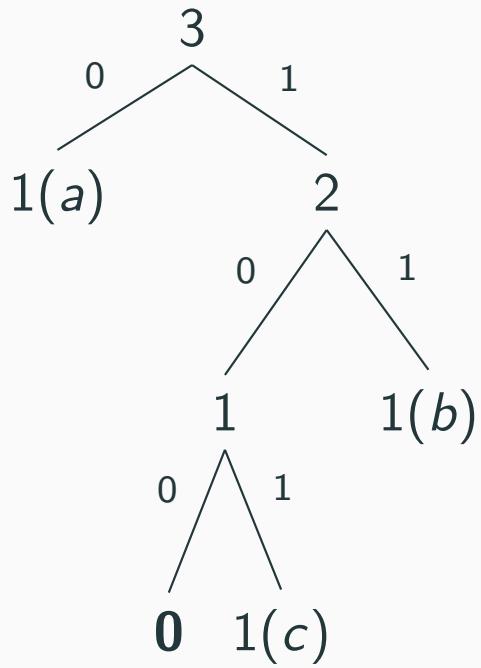
The FGK algorithm: encoding

Third character encoded

Text: abccd

Encoded text: a0b00c

Updated Huffman code



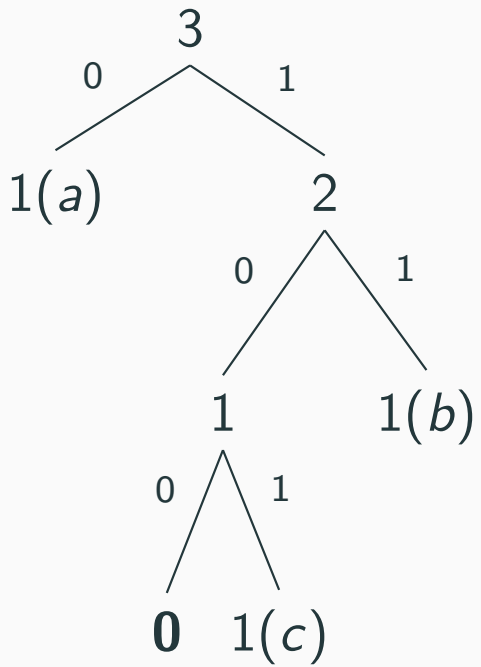
The FGK algorithm: encoding

Read the fourth character

Text: abccd

Encoded text: a0b00c

Current Huffman code



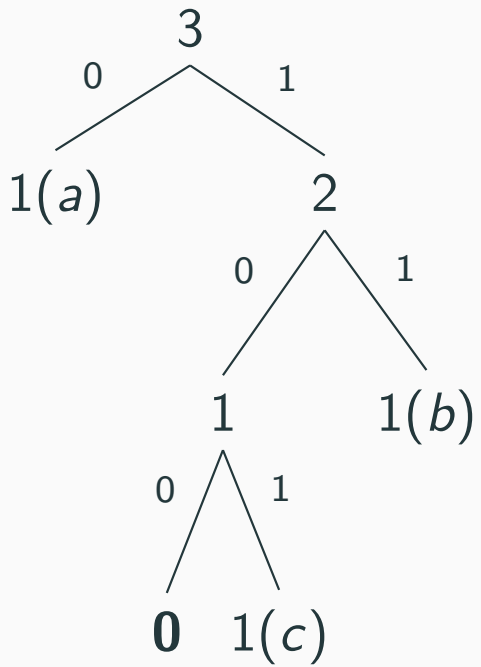
The FGK algorithm: encoding

Encode the fourth character

Text: abccd

Encoded text: a0b00c101

Current Huffman code



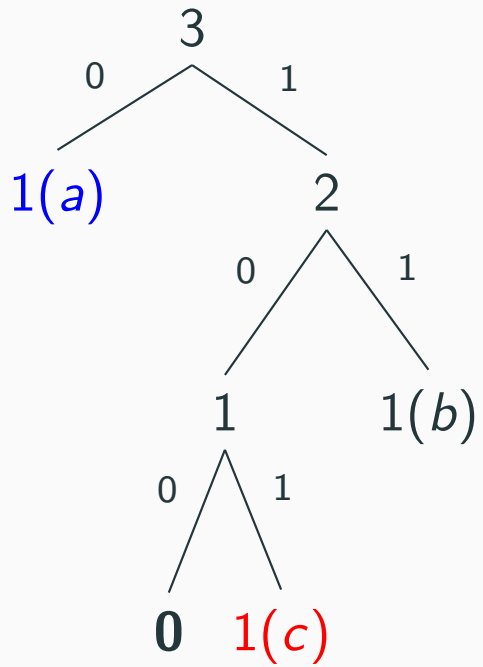
The FGK algorithm: encoding

Fourth character encoded

Text: abccd

Encoded text: a0b00c101

Update Huffman code



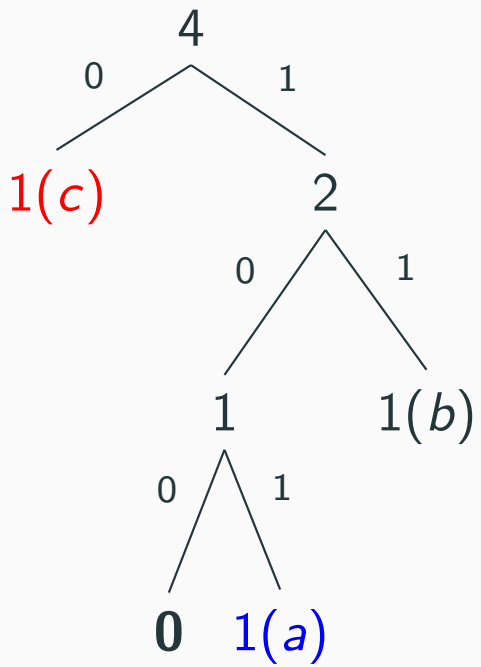
The FGK algorithm: encoding

Fourth character encoded

Text: abccd

Encoded text: a0b00c101

Update Huffman code



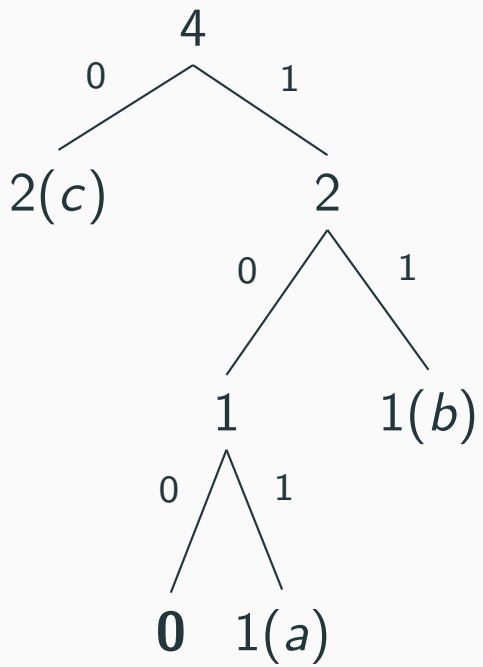
The FGK algorithm: encoding

Fourth character encoded

Text: abccd

Encoded text: a0b00c101

Updated Huffman code



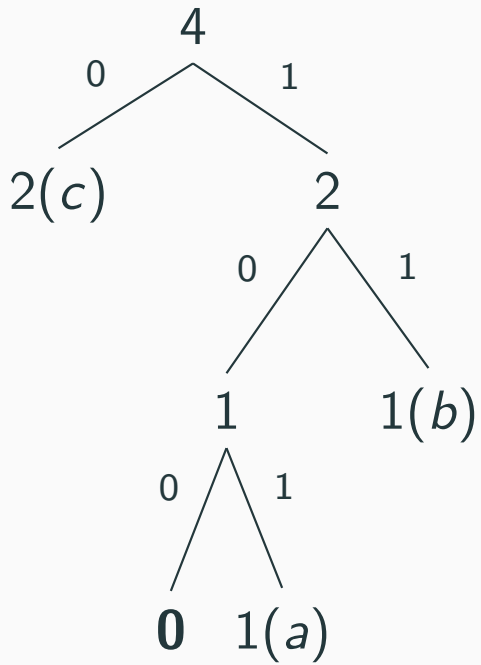
The FGK algorithm: encoding

Read the fifth character

Text: abccd

Encoded text: a0b00c

Current Huffman code



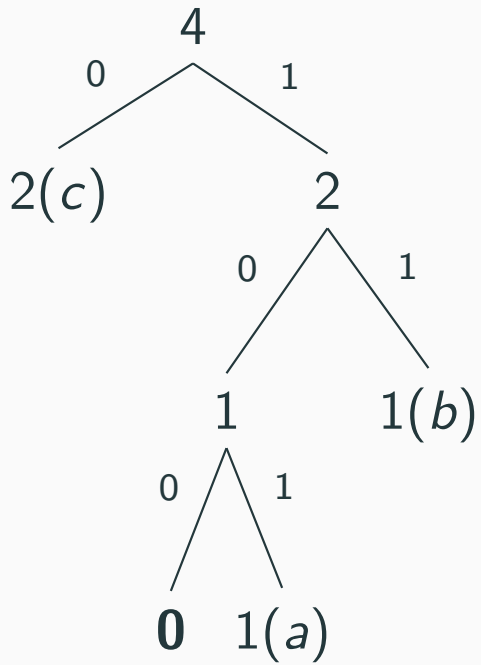
The FGK algorithm: encoding

Encode the fifth character

Text: abccd

Encoded text: a0b00c101100d

Current Huffman code



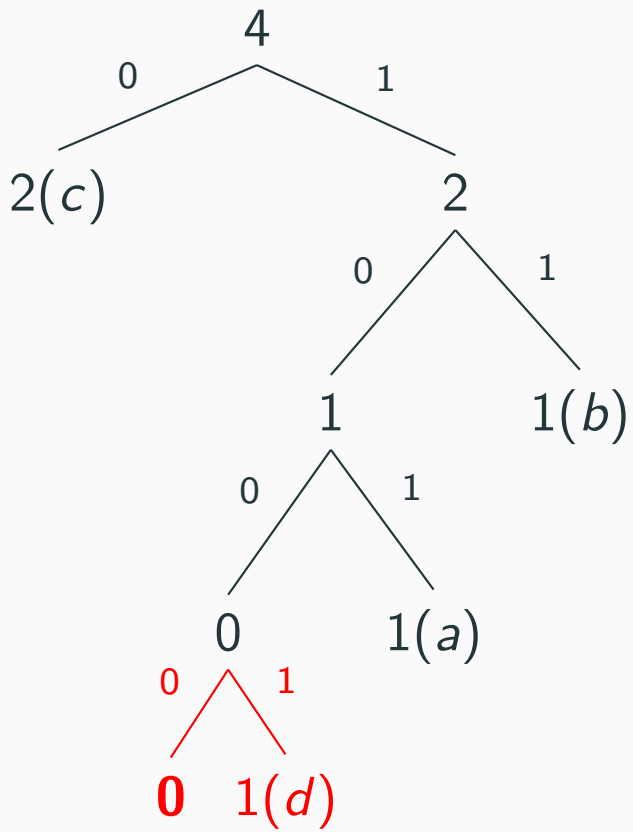
The FGK algorithm: encoding

Fifth character encoded

Text: abccd

Encoded text: a0b00c101100d

Update Huffman code



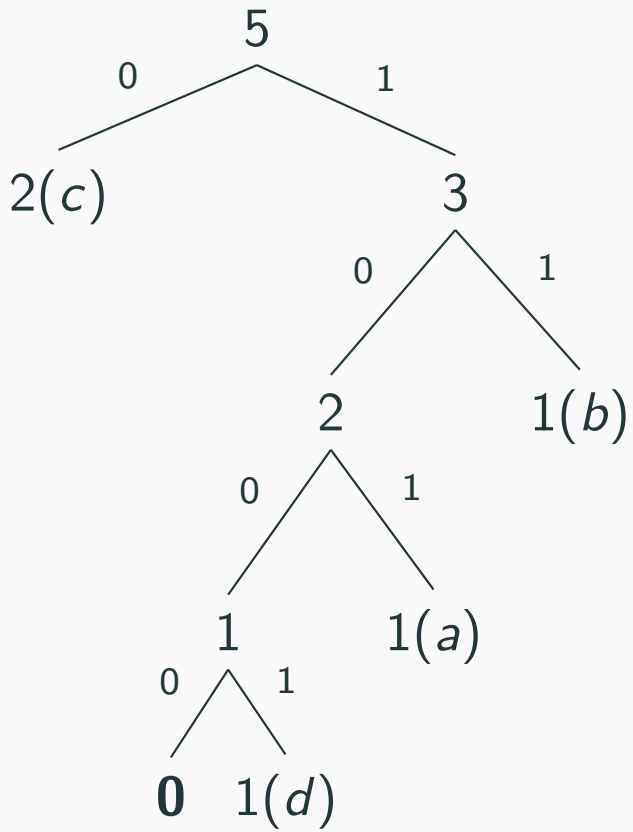
The FGK algorithm: encoding

Fifth character encoded

Text: abccd

Encoded text: a0b00c101100d

Update Huffman code



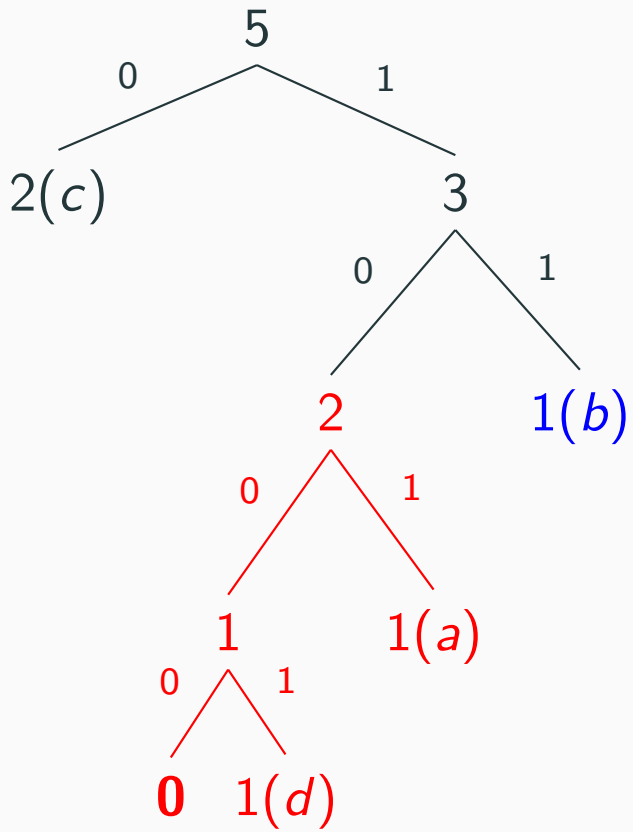
The FGK algorithm: encoding

Fifth character encoded

Text: abccd

Encoded text: a0b00c101100d

Update Huffman code



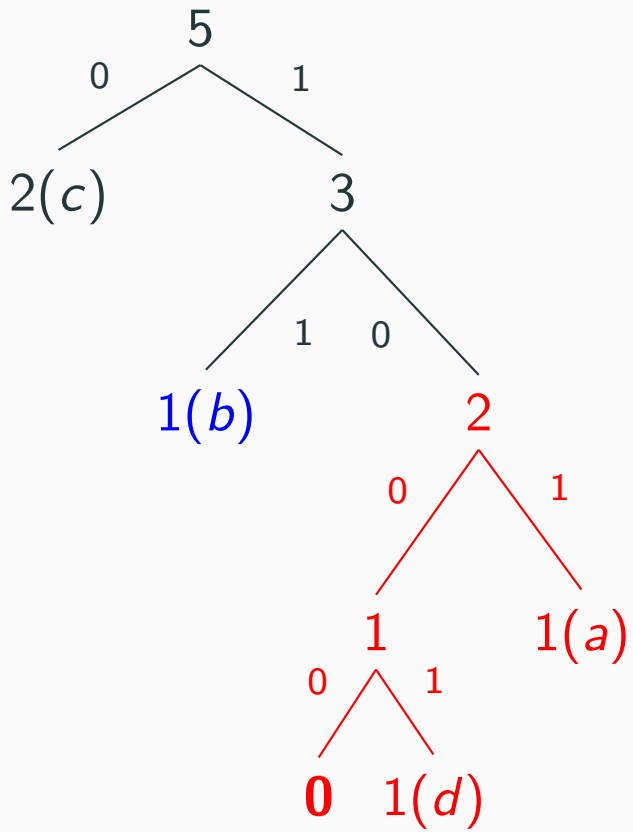
The FGK algorithm: encoding

Fifth character encoded

Text: abccd

Encoded text: a0b00c101100d

Update Huffman code



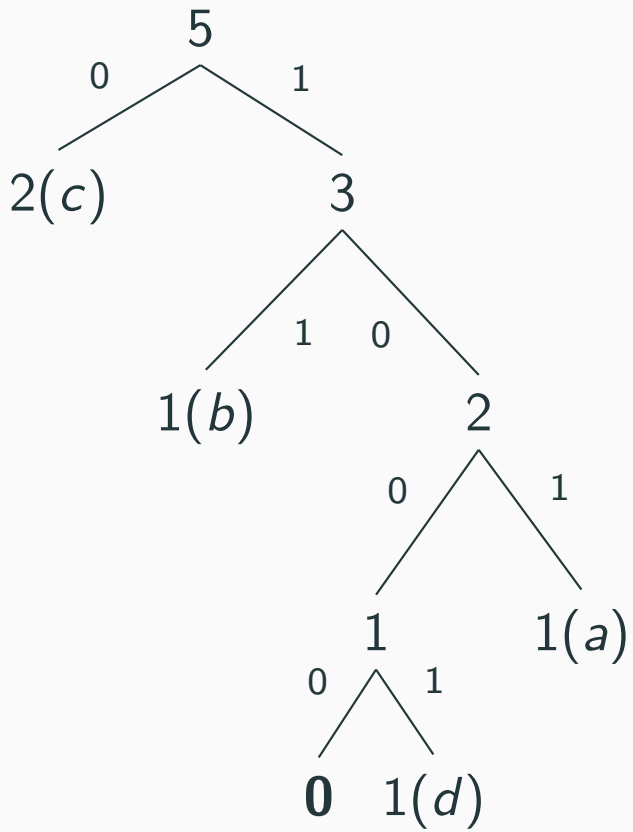
The FGK algorithm: encoding

Fifth character encoded

Text: abccd

Encoded text: a0b00c101100d

Updated Huffman code



Vitter's algorithm

1. Jeffrey Scott Vitter, 1987 [5];
2. Designed to minimize the average codeword length $\sum_{a \in A} |h(a)|\pi(a)$, the total codeword length $\sum_{a \in A} |h(a)|$, and the maximum codeword length $max_{a \in A}|h(a)|$.

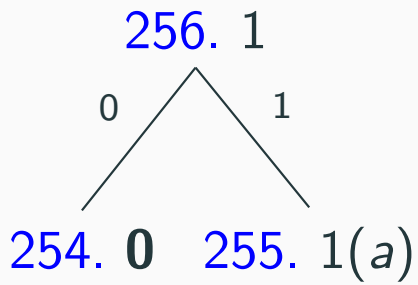
Key elements to be described:

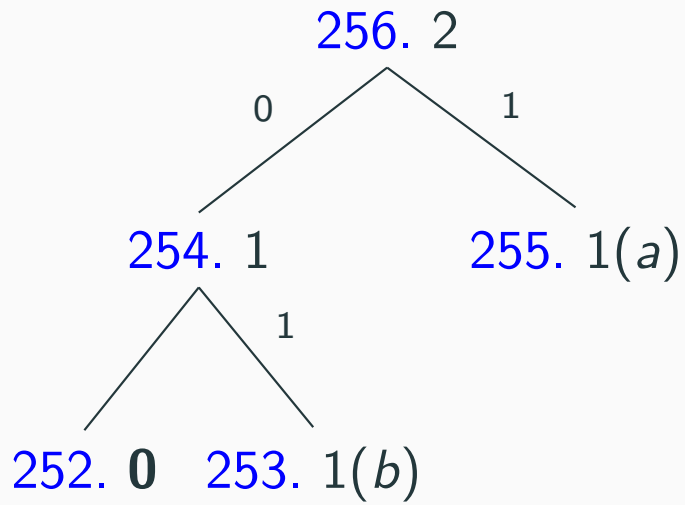
1. Implicit numbering;
2. Invariant;
3. Block (of leaves and internal nodes);
4. Updating procedures.

Vitter's algorithm: implicit numbering

An implicit numbering is just a numbering scheme that numbers the tree nodes in increasing order on levels from left to right and bottom to top.

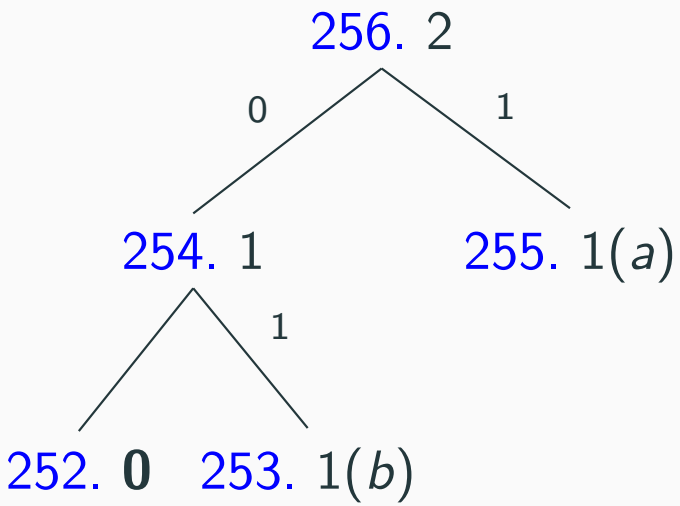
The implicit numbering does not change except for the insertion of new nodes that should have smaller numbers than those in the old tree!



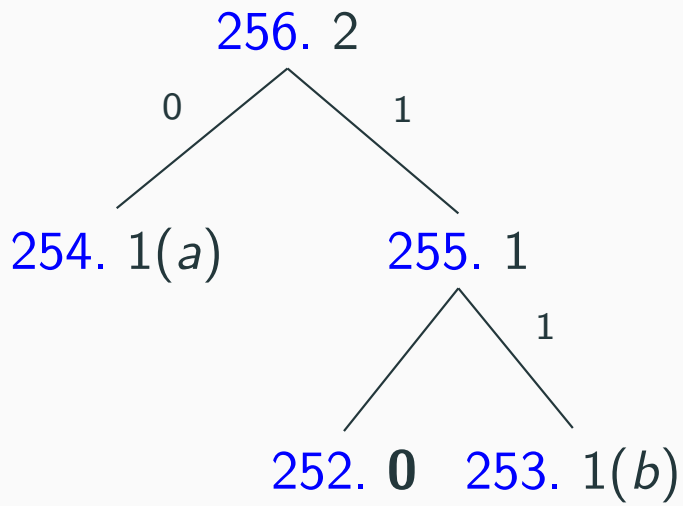


Vitter's algorithm: invariant

Invariant: For each weight w, all leaves of weight w precede (in the implicit numbering) all internal nodes of weight w.



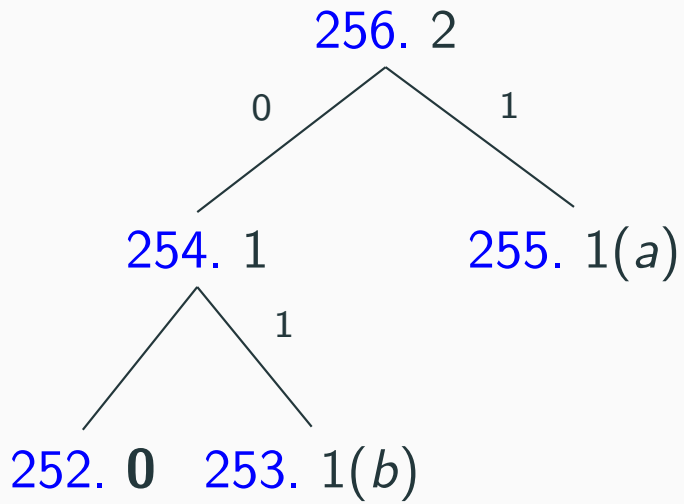
Does not satisfy the invariant!



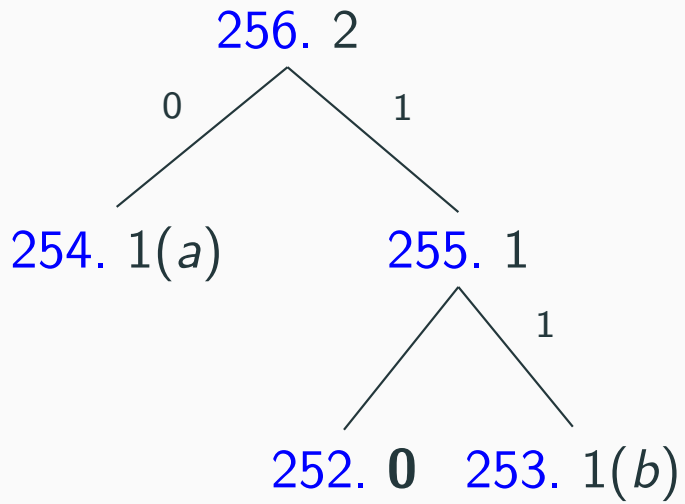
Satisfies the invariant!

Vitter's algorithm: block

A block is a maximal sequence of nodes, in the implicit order, of the same weight and type. The element with the highest number is the block leader. There are two types of blocks: blocks of leaves and blocks of internal nodes.



Two blocks of leaves, $\langle 253 \rangle$ and $\langle 255 \rangle$, and one block of internal nodes, $\langle 254 \rangle$, all of weight 1.



One block of leaves, $\langle 253, 254 \rangle$, and one block of internal nodes, $\langle 255 \rangle$, all of weight 1.

Vitter's algorithm: tree updating

The tree updating procedure is based on the following concepts:

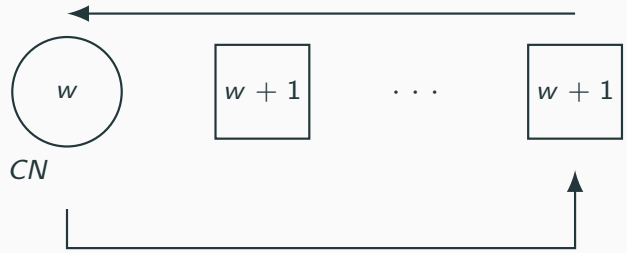
1. Current_node (CN): a node to be processed. It may be any node, including the **0**-node;
2. Leaf_to_increment (LtI): a leaf to be incremented;
3. Slide_&_Increment: slide a node over a block. There are two procedures:
   - 3.1 slide a leaf over a block of internal nodes;
   - 3.2 slide an internal node over a block of leaves;
4. New or existing symbol insertion.

Vitter's algorithm: Slide_&_Increment

Sliding a leaf

Leaf's weight is incremented, and CN is changed!

Sliding an internal node



Internal node's weight is incremented, and CN is changed!

Vitter's algorithm: inserting a new symbol

The procedure to add a new symbol a is the following:

1. CN is the **0**-node;
2. Add two children to CN: a **0**-node as the left children and a leaf for a as the right children;
3. LtI becomes the right children of CN;
4. While CN is not the root do Slide_&_Increment(CN);
5. Slide_&_Increment(LtI).

The procedure to add an existing symbol a is the following:

1. CN is the existing node of a;
2. Interchange CN with the leader of its block;
3. If the new node if the sibling of the **0**-node then it becomes LtI and its parent is the new CN;
4. While CN is not the root do Slide_&_Increment(CN);
5. Slide_&_Increment(LtI).

Vitter's algorithm: encoding

Text to be encoded

Huffman code

Text: `abcc`

Encoded text:

Vitter's algorithm: encoding

Read the first character

Text: abcc

Encoded text:

Current Huffman code

0

Vitter's algorithm: encoding

Encode the first character

Text: abcc

Encoded text: a

Current Huffman code

0

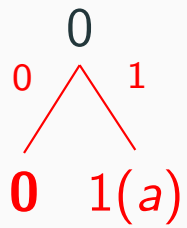
Vitter's algorithm: encoding

First character encoded

Text: abcc

Encoded text: a

Update Huffman code



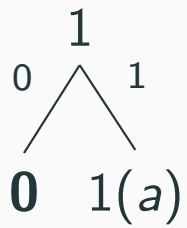
Vitter's algorithm: encoding

First character encoded

Text: abcc

Encoded text: a

Updated Huffman code



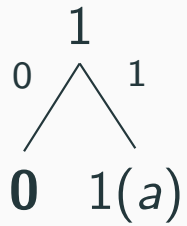
Vitter's algorithm: encoding

Read the second character

Text: abcc

Encoded text: a

Current Huffman code



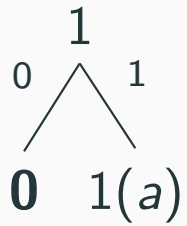
Vitter's algorithm: encoding

Encode the second character

Text: abcc

Encoded text: a0b

Current Huffman code



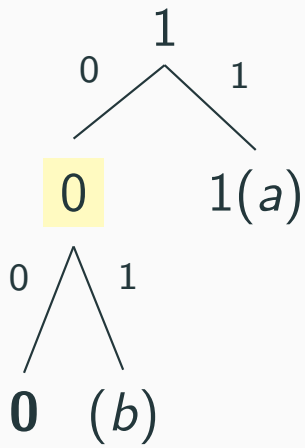
Vitter's algorithm: encoding

Second character encoded

Text: abcc

Encoded text: a0b

Update Huffman code



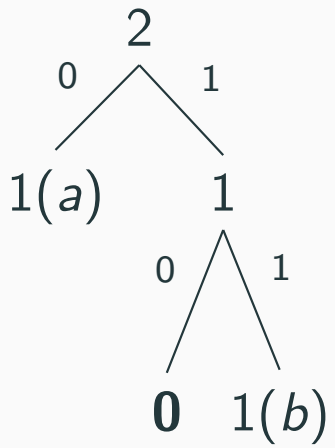
Vitter's algorithm: encoding

Second character encoded

Text: abcc

Encoded text: a0b

Updated Huffman code



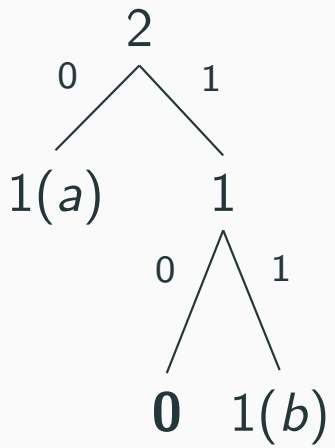
Vitter's algorithm: encoding

Read the third character

Text: abcc

Encoded text: a0b

Current Huffman code



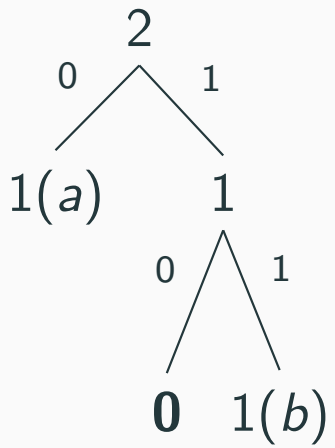
Vitter's algorithm: encoding

Encode the third character

Text: abcc

Encoded text: a0b10c

Current Huffman code



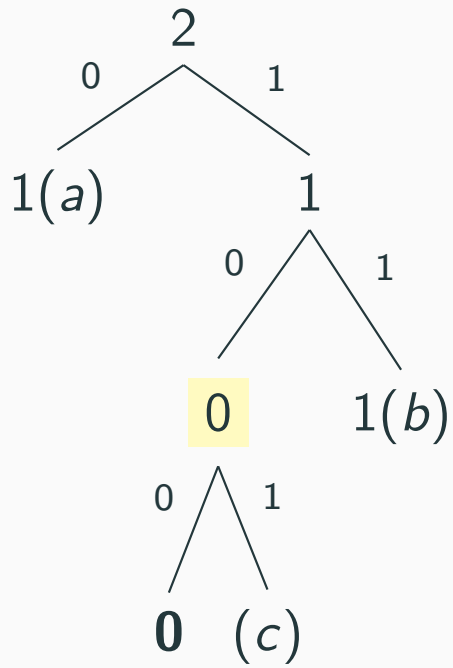
Vitter's algorithm: encoding

Third character encoded

Text: abcc

Encoded text: a0b10c

Update Huffman code



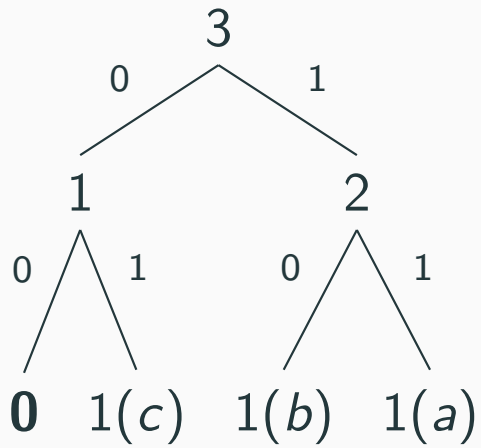
Vitter's algorithm: encoding

Third character encoded

Text: abcc

Encoded text: a0b10c

Updated Huffman code



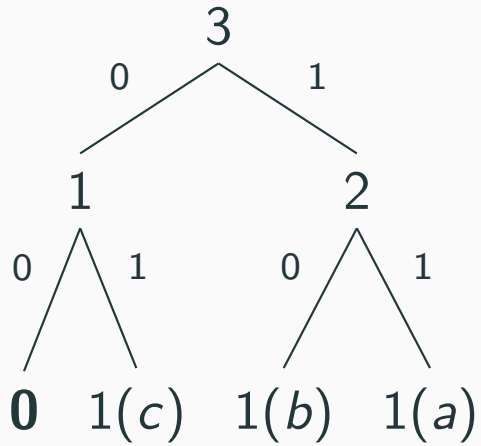
Vitter's algorithm: encoding

Read the fourth character

Text: abcc

Encoded text: a0b10c

Current Huffman code



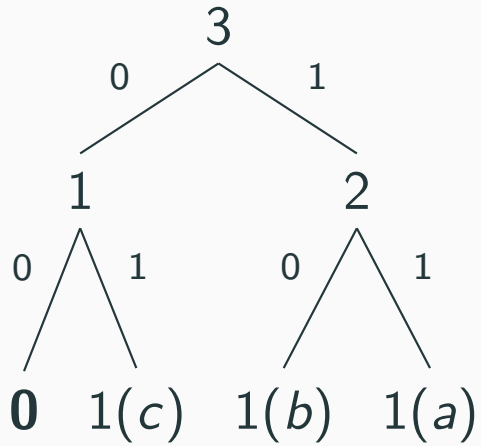
Vitter's algorithm: encoding

Encode the fourth character

Text: abcc

Encoded text: a0b10c01

Current Huffman code



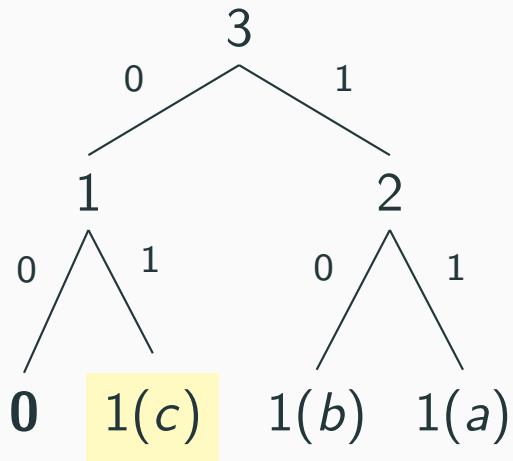
Vitter's algorithm: encoding

Fourth character encoded

Text: abcc

Encoded text: a0b10c01

Update Huffman code



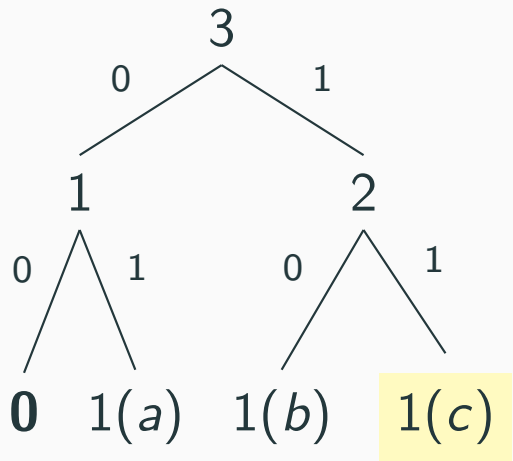
Vitter's algorithm: encoding

Fourth character encoded

Text: abcc

Encoded text: a0b10c01

Update Huffman code



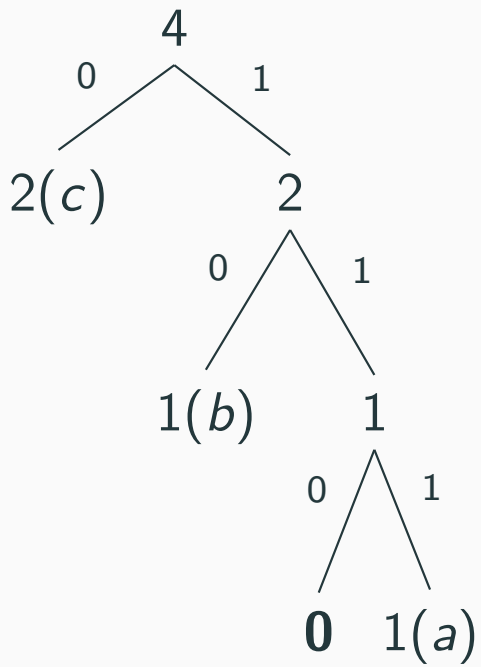
Vitter's algorithm: encoding

Fourth character encoded

Text: abcc

Encoded text: a0b10c01

Updated Huffman code



Reading

References

[1] Ferucio Laurențiu Țiplea. *Algebraic Foundations of Computer Science*. "Alexandru Ioan Cuza" University Publishing House, Iași, Romania, second edition, 2021.

[2] Newton Faller. An adaptive system for data compression. In *7th Asilomar Conference on Circuits, Systems and Computers*, pages 593–597. IEEE, 1973.

[3] Robert Gallager. Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, 24(6):668–674, 1978.

[4] Donald E Knuth. Dynamic Huffman coding. *Journal of Algorithms*, 6(2):163–180, 1985.

[5] Jeffrey Scott Vitter. Design and analysis of dynamic huffman codes. *J. ACM*, 34(4):825–845, Oct 1987.