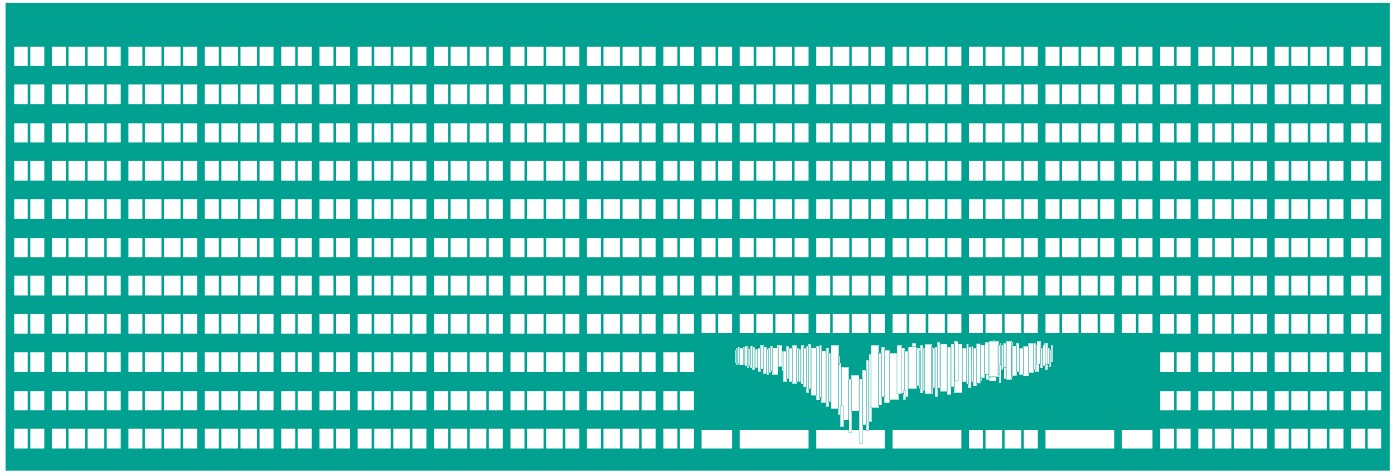
VŠB TECHNICKÁ UNIVERZITA OSTRAVA

VSB TECHNICAL UNIVERSITY OF OSTRAVA



www.vsb.cz

Statistical Compression

Part I

Michal Vasinek

VSB – Technical University of Ostrava

name.surname@vsb.cz

March 28, 2023

Overview

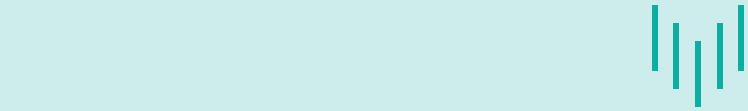
- Shannon-Fano Coding
- Kraft Inequality
- Optimal Codes
- Bounds on the Optimal Code Length
- Huffman Coding

Opakování z minula

Prefixový kód - otázka 1 bod

Co to znamená, že je kód prefixový?

Fanův algoritmus - Motivace



- V roce 1948 Claude Shannon publikoval svůj článek, kde zadefinoval entropii.
- Na stejném problému pracoval i Robert Fano, došel ke stejnému výsledku, ale o rok později. Odvození konceptem binárních otázek.
- Shannonův článek má 96500 citací.
- Fanův článek 320 citací.

Otázka doby

Jak navrhnout prefixový kód, který bude pro dané rozdělení pravděpodobnosti symbolů optimální?

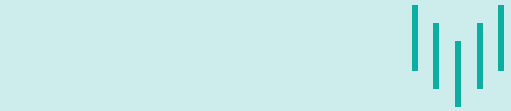
K zamyšlení

- Nechť máme následující pravděpodobnosti symbolů: $p(a) = 0.35$, $p(b) = 0.15$, $p(c) = 0.16$, $p(d) = 0.17$, $p(e) = 0.17$.
- Chtěli bychom sestavit kód tzv. shora dolů, kód můžeme reprezentovat binárním stromem, v listech budou symboly, čtením hodnot 0,1 od kořene k listu obdržíme kódové slovo symbolu v listu.

Otázka za 2 body

Libovolně rozdělte množinu symbolů na dvě podmnožiny, jedné podmnožině přiřaďte 0 druhé 1. Proces opakujte dokud nemáte jednoprvkové množiny. Jak byste rozdělili symboly do podmnožin?

Fanovo kódování - příklad



1. Setřiďte symboly od nejčetnějšího po nejméně četný.
2. Rozdělte setřizené symboly na dvě množiny, tak aby rozdíl součtu pravděpodobností symbolů v obou množinách byl minimální. První podmnožině přiřaďte 0, druhé podmnožině 1.
3. Opakujte bod dva na každé podmnožině s více, než jedním prvkem.
4. Algoritmus funguje velmi dobře, ale např. pro $p(a) = 0.35$, $p(b) = 0.15$, $p(c) = 0.16$, $p(d) = 0.17$, $p(e) = 0.17$ nevygeneruje optimální prefixový kód.

Fanovo kódování

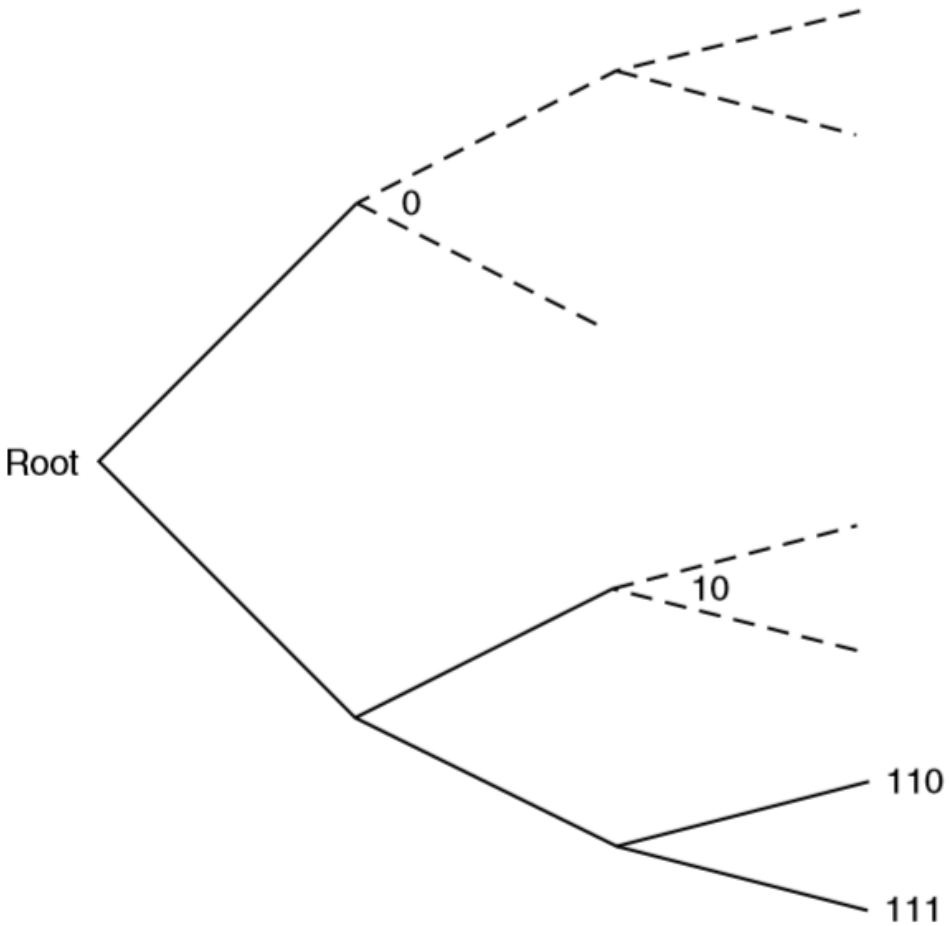
a_i	$p(a_i)$	1	2	3	4	Code
a_1	0.36	0	00			00
a_2	0.18		01			01
a_3	0.18	1	10			10
a_4	0.12		11	110		110
a_5	0.09			111	1110	1110
a_6	0.07				1111	1111

Figure: Fano coding - source: https://stackoverflow.com/questions/43357381/shannon-fano-coding-algorithm-strange-behaviour-on-larger-sets

Smutný příběh génia - Fanovo kódování

- Existují dva odlišné algoritmy Shannonův a Fanův, nicméně v literatuře se Fanův algoritmus označuje termínem Shannon-Fanovo kódování
- V roce 1952 byl Robert Fano profesorem na MIT, kde zadal svému doktorskému studentovi Davidu Huffmanovi práci na téma komprese dat.
- Huffman navrhl a dokázal techniku pro vytvoření optimálního prefixového kódu jako seminární práci!
- Huffmanův algoritmus má téměř 10 tis. citací a je součástí mnoha dodnes používaných formátů GZIP, JPEG a dalších.
- `https://www.youtube.com/watch?v=rAL1g2FdWxE`

Kraft Inequality



- Branches of the tree represent symbols of the codeword.
- Each codeword is represented by a leaf on the tree and the path from the root yields symbols of the codeword.
- No codeword is a prefix of other codeword.
- Codes = { 0, 10, 110, 111 }

Figure: Code tree for the Kraft inequality, Cover and Thomas, Elements of Information Theory, p. 108.

Kraft Inequality

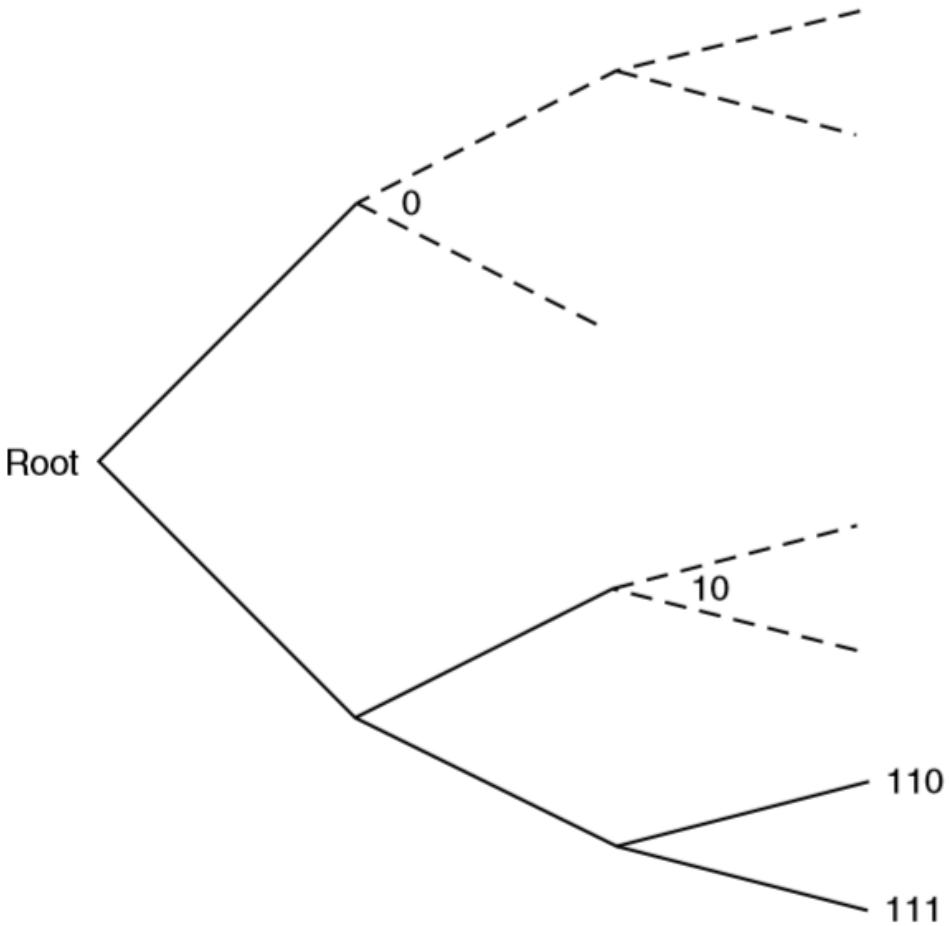


Figure: Code tree for the Kraft inequality, Cover and Thomas, Elements of Information Theory, p. 108.

- Let l_{max} be the length of the longest codeword.
- Consider all nodes (not only codewords) of the tree at level l_{max}.
- For instance with $l_{max} = 3$, there are 8 such nodes.
- Codeword at level i has $2^{l_{max}-l_i}$ descendants at level l_{max}.
- For instance node 11 at level 2 has $2^{3-2} = 2$ descendants in l_{max}.
- Descendant sets in one level are disjoint.

Kraft Inequality

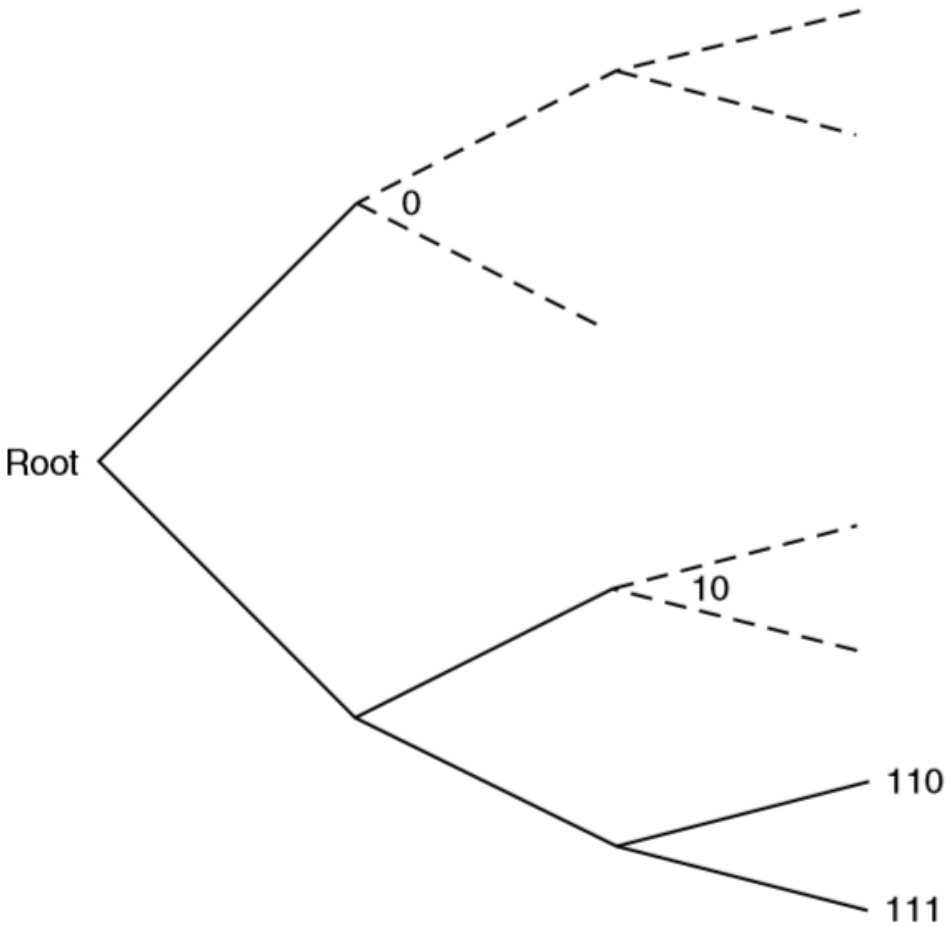


Figure: Code tree for the Kraft inequality, Cover and Thomas, Elements of Information Theory, p. 108.

- Sum of all descendants at l_{max} over all codewords (only) is smaller or equal to $2^{l_{max}}$:

$$\sum_{codewords} 2^{l_{max}-l_i} \leq 2^{l_{max}}$$

- After simplification:

$$\sum_{codewords} 2^{-l_i} \leq 1 \qquad (1)$$

Kraft Inequality

Theorem

Kraft inequality For any prefix code over an alphabet of size D, the codeword lengths $l_1, l_2, \ldots, l_m$ must satisfy the inequality:

$$\sum_i D^{-l_i} \leq 1 \tag{2}$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an prefix code with these word lengths.

Optimal Codes

- $l_1, l_2, \ldots, l_m$ be code lengths for symbols $c_1, c_2, \ldots, c_m$, whose probabilities are $p_1, p_2, \ldots, p_m$.
- The average (expected) codeword length L is given by:

$$L = \sum p_i l_i$$

- For instance: let $l_1 = 1, l_2 = 2, l_3 = 2$ and $p_1 = 0.5$, $p_2 = 0.3$ and $p_3 = 0.2$, then:

$$L = 0.5 * 1 + 0.3 * 2 + 0.2 * 2 = 1.5 \text{ bits}$$

- We now consider the problem of finding the prefix code with the minimum expected length.

Minimum expected codeword length

Minimize:

$$L = \sum p_i l_i$$

over all integers $l_1, l_2, \ldots, l_m$ satisfying:

$$\sum D^{-l_i} \leq 1$$

The problem can be solved analytically using Lagrange multipliers.

Optimal Codes

- For the moment we neglect the constraint on integer values of l_i and we assume equality in Kraft inequality.
- We form Lagrange multiplier[1]:

$$J = \sum p_i l_i + \lambda \left(\sum D^{-l_i} - 1 \right)$$

- Differentiate to obtain minimum:

$$\frac{\partial J}{\partial l_i} = p_i - \lambda D^{-l_i} \log_e D$$

- Set derivative to 0:

$$D^{-l_i} = \frac{p_i}{\lambda \log_e D}$$

- Substituting it into Kraft inequality we obtain:

$$\lambda = \frac{1}{\log_e D}$$

[1] see: `https://en.wikipedia.org/wiki/Lagrange_multiplier`

- Sofar we have:

$$D^{-l_i} = \frac{p_i}{\lambda \log_e D}$$

$$\lambda = \frac{1}{\log_e D}$$

- Which implies:

$$p_i = D^{-l_i}$$

- Hence, the optimal codeword length l_i^*:

$$l_i^* = -\log_D p_i$$

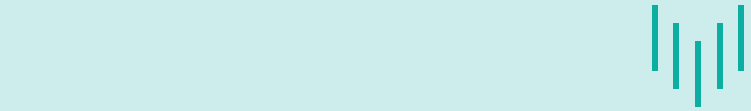
Optimal Codes

- Optimal code L^*:

$$L^* = \sum p_i l_i^* = -\sum p_i \log_D p_i = H_D(X)$$

- l_i^* can be here arbitrary real number.
- Optimal code is achieved only when all $-\log_D p_i$ are integers.

Bounds on the Optimal Code Length



- Optimal codeword length $-\log_D p_i$ can be arbitrary real number!
- We have to round it up to obtain integer codeword length:

$$l_i = \left\lceil \log_D \frac{1}{p_i} \right\rceil$$

- Split into inequalities:

$$\log_D \frac{1}{p_i} \leq l_i < \log_D \frac{1}{p_i} + 1$$

On board (2pts)

Multiply by p_i and sum over all i the following inequality:

$$\log_D \frac{1}{p_i} \leq l_i \leq \log_D \frac{1}{p_i} + 1$$

Optimal code length

Optimal prefix code length is never worse than $H_D(X) + 1$

Length of the optimal prefix code

$$H_D(X) \leq L \leq H_D(X) + 1 \tag{3}$$

Requirements on optimal prefix code

- (1) No codeword is a prefix of other codeword - **prefix property**.
- (2) Let $p(i)$ be a probability of i-th symbol and let l_i be the associated codeword length, alphabet size be N, then:

$$p(0) \geq p(1) \geq p(2) \geq \cdots \geq p(N)$$

$$l(0) \leq l(1) \leq l(2) \leq \cdots \leq l(N)$$

Bonus

- Suppose the input alphabet Σ, the size of the alphabet be N.
- Further asssume $p(i) \geq p(j)$, $i < j$.

On board (1pt)

Draw a tree for any prefix code, where the code for the N-th symbol is longer than the code for the $(N-1)$-th symbol.

Requirements on optimal prefix code

- Consider codeword for $(N-1)$-th symbol:

$$C_{N-1} = B_1 B_2 \ldots B_{l(N-1)}$$

it contains $l(N-1)$ binary digits B_j.

- (3) To select a codeword for N-th symbol it makes no sense to construct a codeword longer than $l(N-1)$ digits as the shortest possible codeword can be obtained by the last bit negation in C_{N-1}. For instance:

$$C_{N-1} = B_1 B_2 \ldots 0$$

and

$$C_N = B_1 B_2 \ldots 1$$

Requirements on optimal prefix code

- Let C be a set of codewords for symbols from the input source alphabet Σ.
- Suppose that there is a binary sequence $S = B_1 B_2 \ldots B_M$, so that $M < l(N)$ and S is not a prefix of any other codeword.
- That means that we can replace codeword for N-th symbol and reduce the average code length => then C is not an optimal prefix code.
- (4) Summary: Each possible sequence of less than $l(N)$ binary digits must be used either as a codeword or must have one of its prefixes used as a codeword.

Construction of optimal prefix code

- Let the alphabet of the source be Σ, the size of the alphabet be $\sigma = N$, and $p(\Sigma[i]) \geq p(\Sigma[i+1])$.
  1. Select two the least probable symbols $\Sigma[N-1], \Sigma[N]$, by (3) we know that the two symbols codewords differs only in the last bit.
  2. Remove the two symbols from Σ and form an ensemble of the two symbols $(\Sigma[N-1], \Sigma[N])$, the probability of the ensemble will be the sum of the two symbols probabilities:

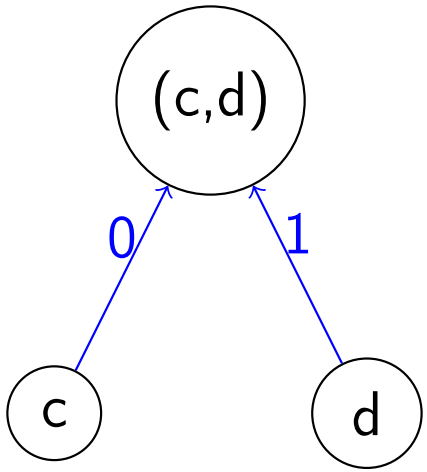
$$p((\Sigma[N-1], \Sigma[N])) = p(\Sigma[N-1]) + p(\Sigma[N])$$

  3. Put the ensemble back to Σ and if neccessary rearrange symbols so that $p(\Sigma[i]) \geq p(\Sigma[i+1])$.
  4. If $\sigma > 1$ then continue with Step 1 otherwise stop.

Example

- Alphabet: $\Sigma_0 = \{a, b, c, d\}$.
- Probabilities: $p(a) = 0.4$, $p(b) = 0.25$, $p(c) = 0.25$, $p(d) = 0.1$.

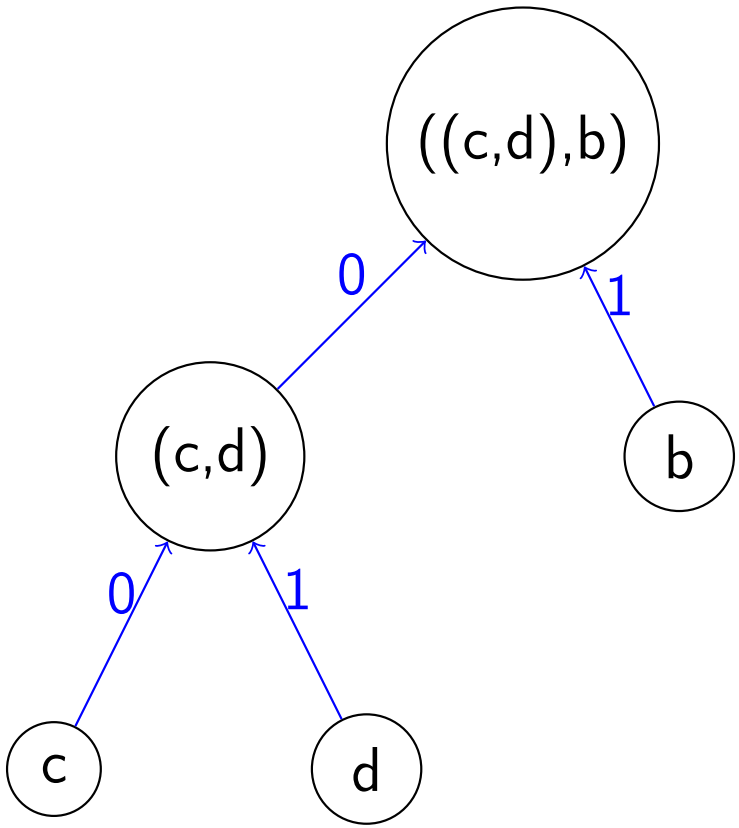
Form an ensemble: (c, d) with $p(c, d) = 0.35$:



Example

- Alphabet: $\Sigma_1 = \{a, (c, d), b\}$.
- Probabilities: $p(a) = 0.4$, $p((c, d)) = 0.35$, $p(b) = 0.25$.

Form an ensemble: $((c, d), b)$ with $p((c, d), b) = 0.6$:

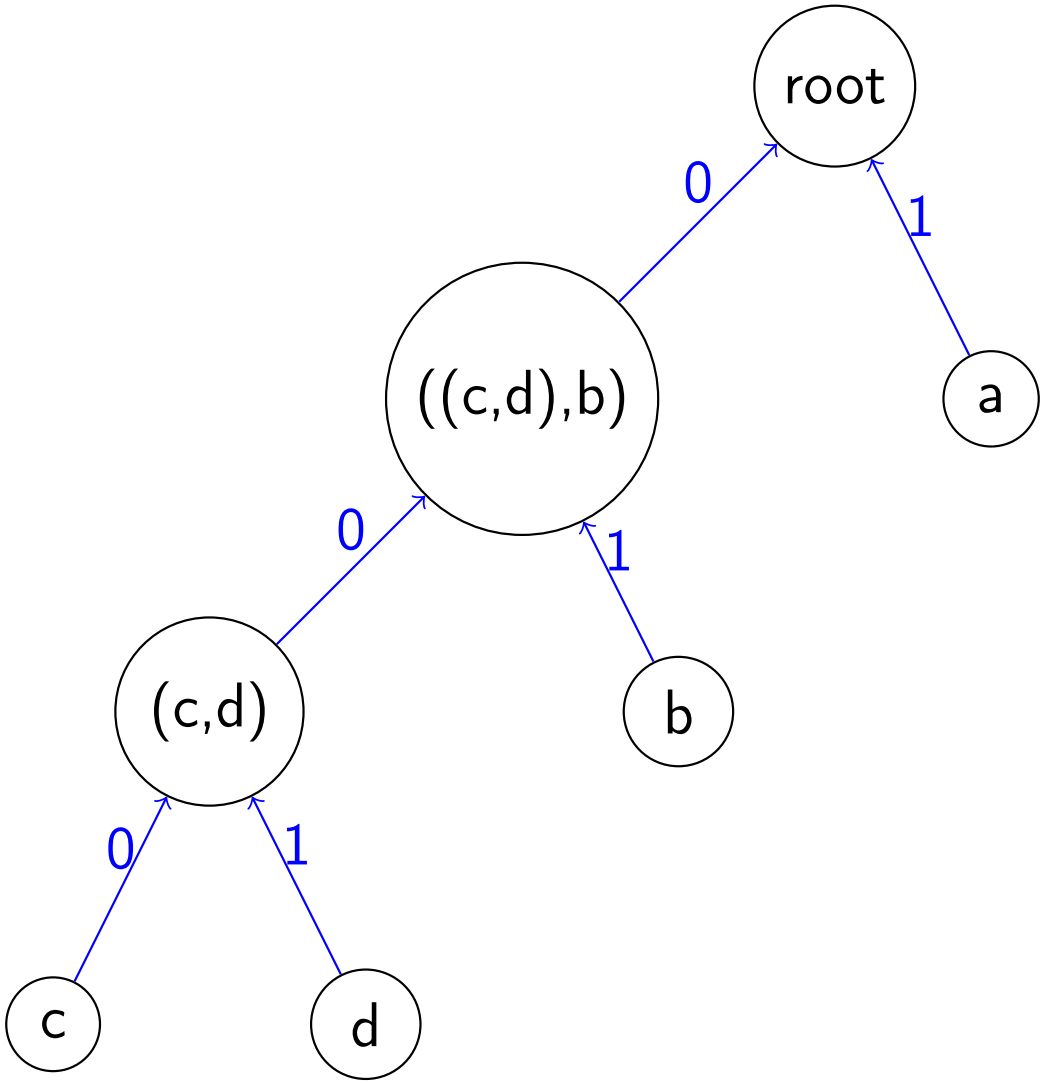


Example

- Alphabet: $\Sigma_2 = \{a, ((c, d), b)\}$.
- Probabilities: $p(((c, d), b)) = 0.6$, $p(a) = 0.4$.

Form an ensemble: $(((c, d), b), a)$ with $p(((c, d), b), a) = 1$.

Example



Example - codewords

- Read binary digits from root to leaves to obtain codewords.
- Conditions (1),(2),(3) and (4) are met.
- The average codeword length:
  $\hat{L} = 0.4 + 0.25 * 2 + 0.25 * 3 + 0.1 * 3 = 1.95$ bits.
- The entropy: $H(X) = 1.86$ bits.

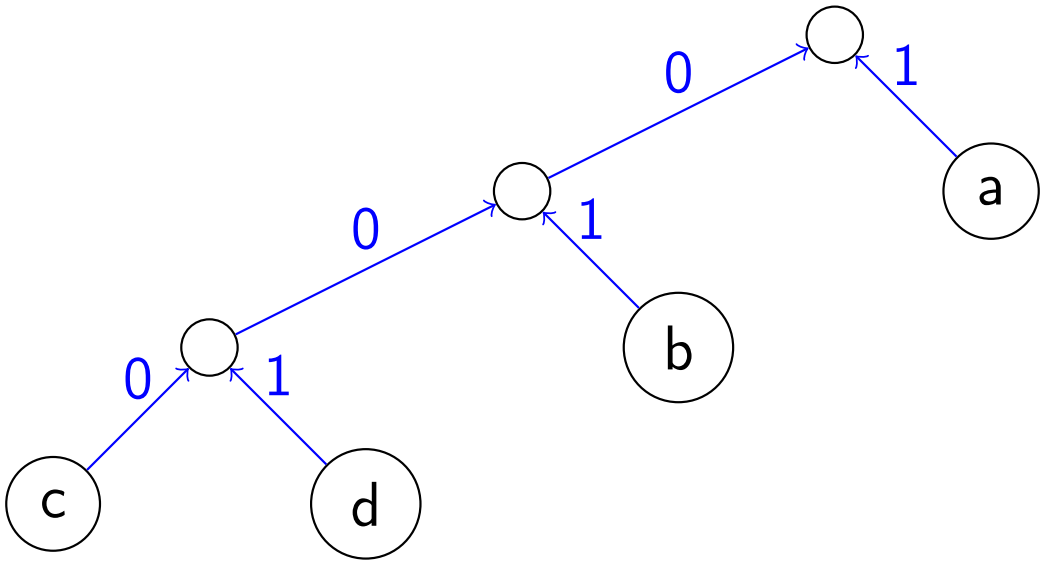
Symbol	Probability	Codeword
a	0.4	1
b	0.25	01
c	0.25	000
d	0.1	001

Huffman coding

- The method for construction of optimal prefix code is called Huffman coding.
- Developed in 1952 by David Huffman.
- Encoding:
  1. Build a codeword tree (Huffman tree).
  2. Encode each symbol by coresponding codeword.
- Decoding:
  1. Build a codeword tree (Huffman tree).
  2. Start from the root node. Read bits and navigate through tree. Once the algorithm is in a leaf node => output coreponding symbol and repeat the step.

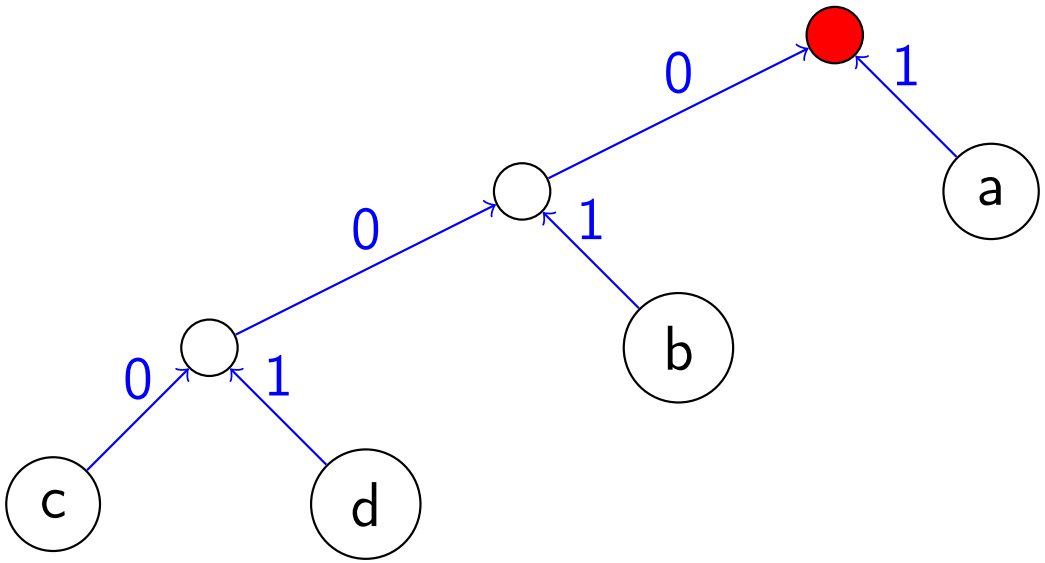
Encoding - example

- We encode the sequence $s = adac\ldots$.
- Compressed sequence: $C(s) = 1\ 001\ 1\ 000\ \ldots$.



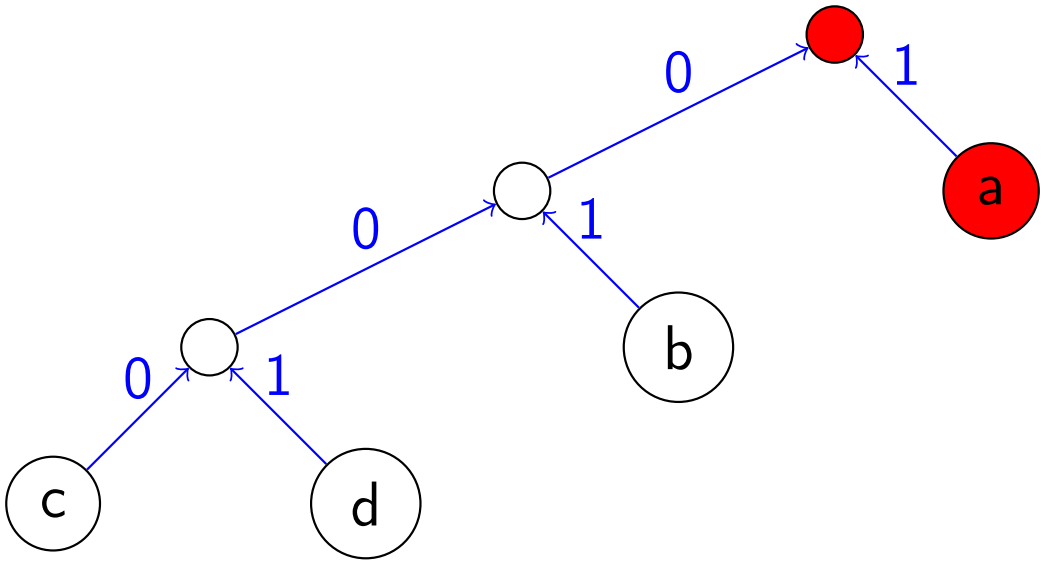
Decoding - example

- Decoding sequence: $C(s) = 10011000\ldots.$



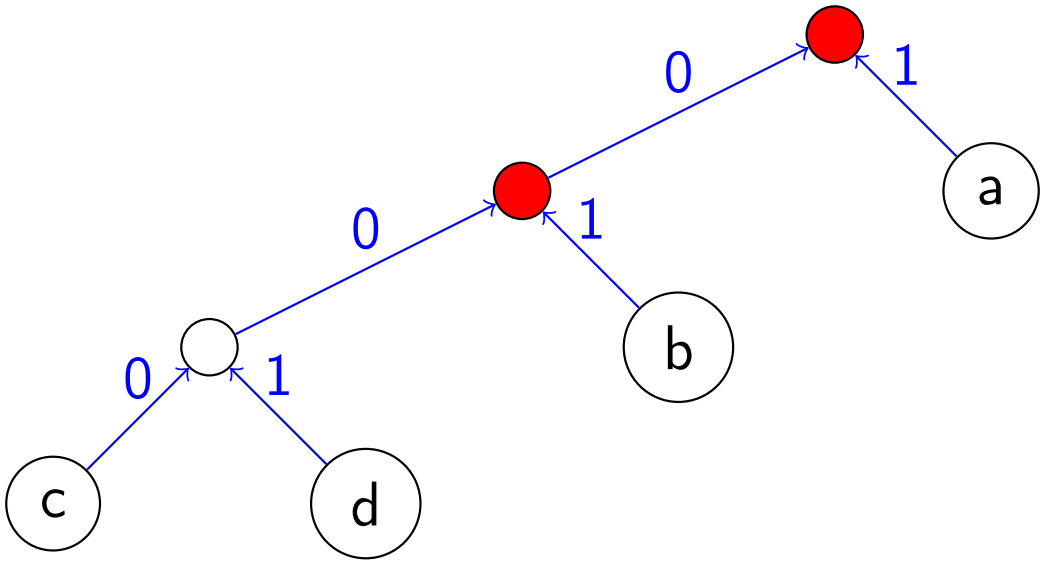
Decoding - example

- Decoding sequence: $C(s) = 10011000\ldots.$
- Read the first bit: $C(s) = \mathbf{1}\ 0011000\ldots.$
- Original message: a



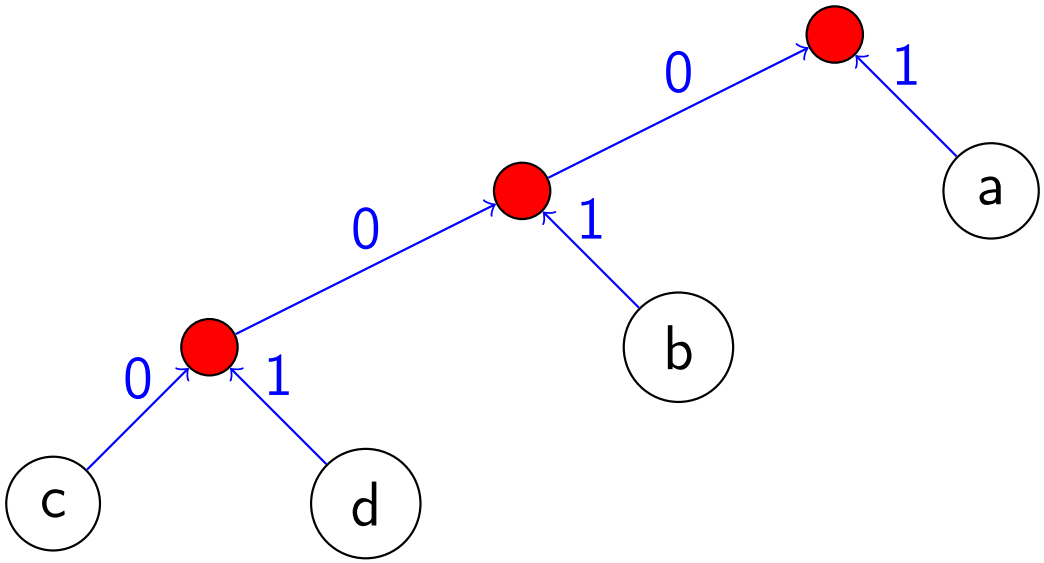
Decoding - example

- Decoding sequence: $C(s) = 10011000\ldots.$
- Read the first bit: $C(s) = 1\ \mathbf{0}\ 011000\ldots.$
- Original message: a



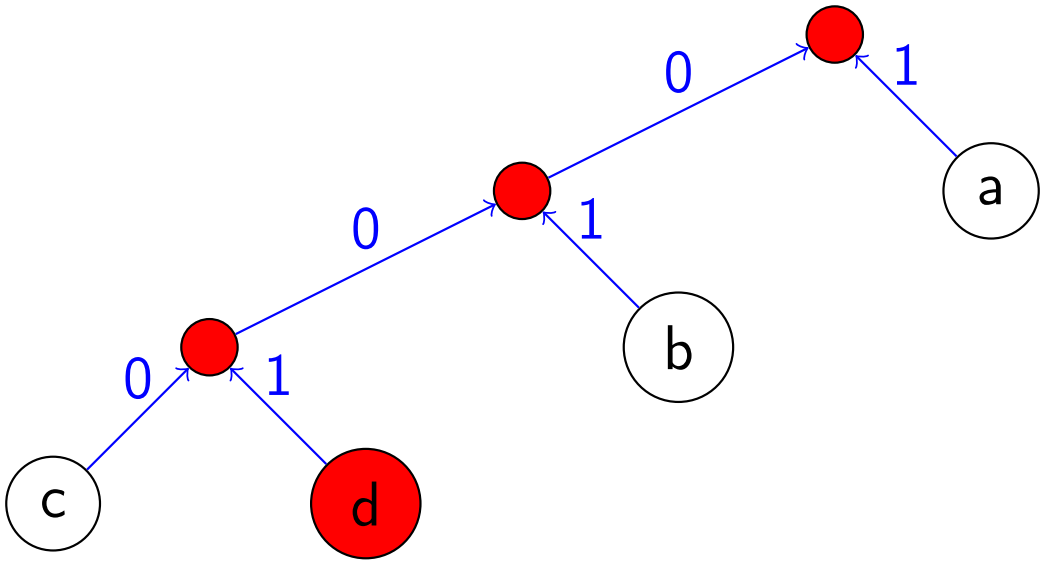
Decoding - example

- Decoding sequence: $C(s) = 10011000\ldots.$
- Read the first bit: $C(s) = 10\ \mathbf{0}\ 11000\ldots.$
- Original message: a



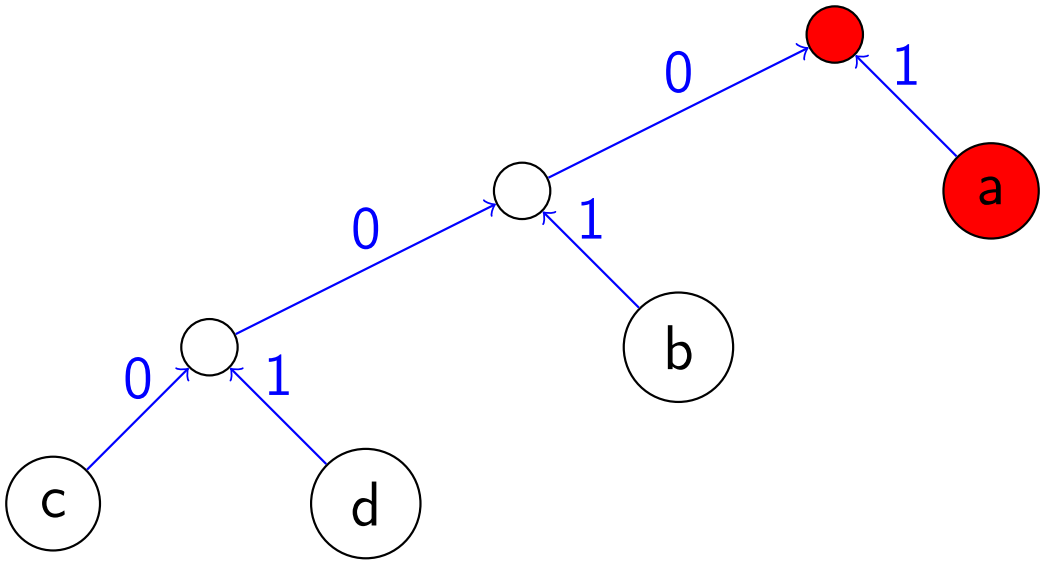
Decoding - example

- Decoding sequence: $C(s) = 10011000\ldots.$
- Read the first bit: $C(s) = 100\ \mathbf{1}\ 1000\ldots.$
- Original message: ad



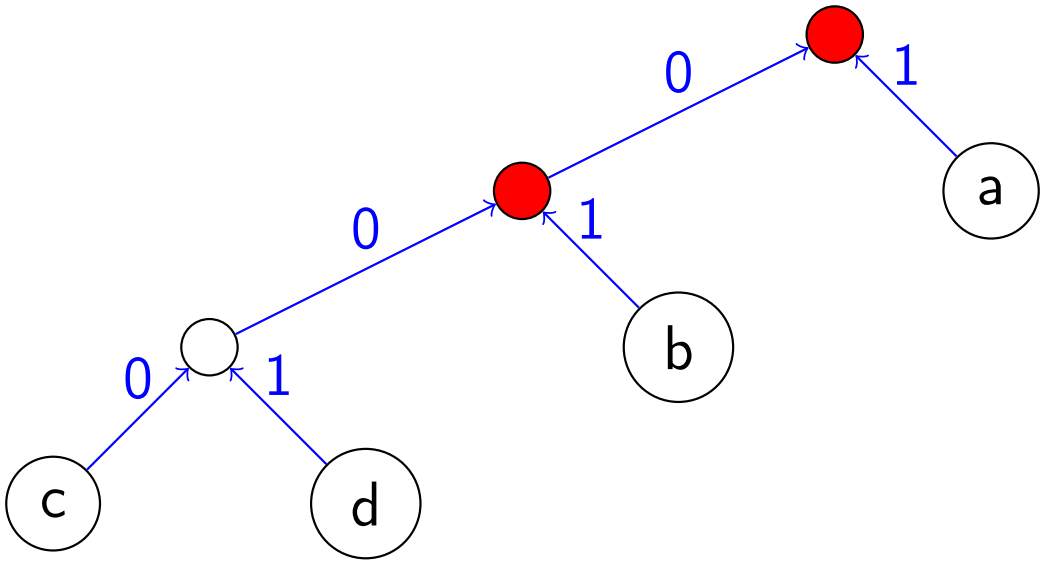
Decoding - example

- Decoding sequence: $C(s) = 10011000\ldots.$
- Read the first bit: $C(s) = 1001\ \mathbf{1}\ 000\ldots.$
- Original message: ada



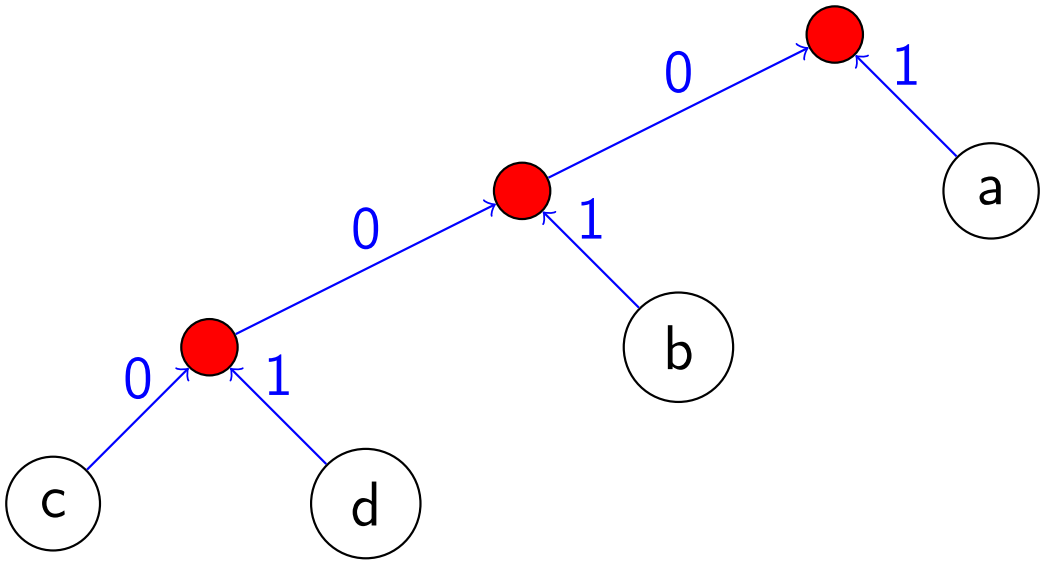
Decoding - example

- Decoding sequence: $C(s) = 10011000\ldots.$
- Read the first bit: $C(s) = 10011\ \mathbf{0}\ 00\ldots.$
- Original message: ada



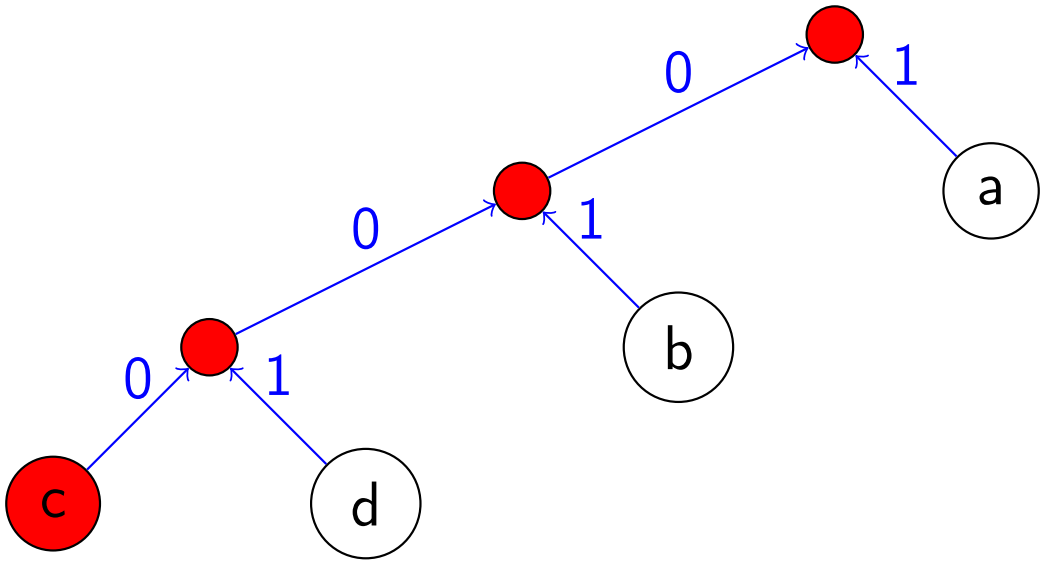
Decoding - example

- Decoding sequence: $C(s) = 10011000\ldots.$
- Read the first bit: $C(s) = 100110\ \mathbf{0}\ 0\ldots.$
- Original message: ada

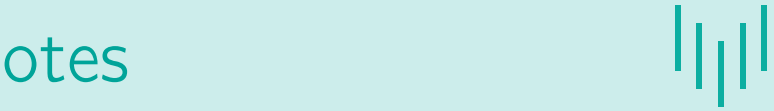


Decoding - example

- Decoding sequence: $C(s) = 10011000\ldots.$
- Read the first bit: $C(s) = 1001100\ \mathbf{0}\ \ldots.$
- Original message: adac



Huffman Coding - implementation notes



- Encoded file should contain a frequency model so that the decoder is able to reconstruct the Huffman tree. Usually encoded in a compressed file header as a table of (symbol,frequency) pairs.
- Encoding: after the Huffman tree completion, codewords should be read out and stored in separate data structure.
- Decoding: never write single symbol to file! The io related tasks (disk read/write) are by far the slowest operations in your computer. Always decode into temporary buffer and once full then flush on hard drive.
- Encoding - decoding - use frequencies (integers) not probabilities (floats, doubles)

Huffman Coding - applications

- Huffman codes are very efficient and extremely fast to process.
- Code redundancy (Gallager 74) $R = L - H \leq p1 + 0.086$ bits.
- They serve as backend to many compression methods.
- ZIP (part of deflate algorithm), GZIP, BZIP2
- JPEG, PNG, MP3 - storage of quantized values.
- Word2vec - hierarchical softmax
- Optimal search tree for objects with different probabilities

Thank you for your attention

Michal Vasinek

VSB – Technical University of Ostrava

name.surname@vsb.cz

March 28, 2023

VSB TECHNICAL
UNIVERSITY
OF OSTRAVA