# *Media Technology*

Prof. Dr.-Ing. Andreas Schrader

## Solution of Assignment 2

Solution 2.1

***Entropy*** of a system in general defines the amount of order contained. For random sources it is a means for the uncertainty of the events generated as symbols of a certain probability. The higher the order (structured system), the lower is the entropy. In general systems (like the universe), the entropy is constantly growing towards total chaos (infinite entropy). In digital random sources, the entropy is limited and can be computed by the set of symbol probabilities as defined by Shannon.

***Redundancy*** describes the distance of a system from its perfect state (total chaos). It describes the remaining structures. For a random source, the redundant part is the amount of data unnecessary to describe the information content. Entropy encoding or source encoding eliminates the redundant parts of the message (due to regular patterns or self-similarities) to increase the compression ratio.

***Irrelevancy*** is a subjective measure of unnecessary information from the perspective of the receiver. For multimedia compression, irrelevancy is defined as the part of the data that can not be perceived by the human aural and visual perception system. In lossy compression schemes, this is used to increase the compression ratios by not storing or transmitting the irrelevant parts. Even higher compression ratios can be achieved by reducing also some of the relevant parts of the data and tolerating some distortion.

Solution 2.2

Let's assume a Markoff process of zero order with the alphabet $A = \{a, b, c, d, r\}$ is the source of the following message: 'abracadabra'. The probability set of the source is not known and has to be estimated from the message itself.

(a) Estimate the probability of the alphabet symbols from the message.
Since we don't know the probabilities, we can only assume, that the number of occurrence $h(a_i)$ of each letter $a_i$ in the only message we know matches the probabilities $p(a_i)$ of the source. These occurrences are
$p(a_1='a') = h('a') = 5/11$, $p(a_2='b') = h('b') = 2/11$, $p(a_3='c') = h('c') = 1/11$, $p(a_4='d') = h('d') = 1/11$, $p(a_5='r') = h('r') = 2/11$.

(b) Calculate the entropy of the source from this estimation.
The entropy of the source can be determined by the following formula (see slide 274):

$$H(\omega) = -\sum_{i=1}^{5} p(a_i) \cdot ld(p(a_i))\ bit$$

Therefore

$$H(\omega) = -\frac{5}{11} \cdot ld(\frac{5}{11}) - 2 \cdot \frac{2}{11} \cdot ld(\frac{2}{11}) - 2 \cdot \frac{1}{11} \cdot ld(\frac{1}{11})$$

$$\approx 0.517 + 2 * 0.447 + 2 \cdot 0.314 \approx 2.039 \text{ bit}$$

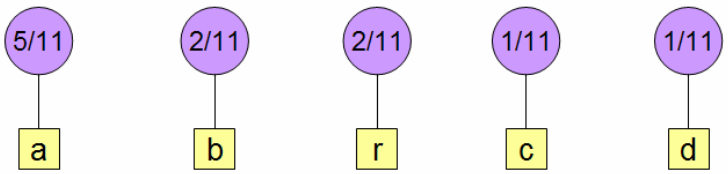
This means, we can't construct a code with an average code length less than 2.039 bit, if we assign a dedicated code for each symbol. In other words, whatever code we construct, we will need at least 11*2.039 = 22.429 bit to code the given message.

(c) Construct a Huffman code for the source. Is there just one code mapping table possible?

1. The set of source symbols are sorted in order of non-increasing probabilities

   Here are different ways of course, since there are some symbols with the same probability. The following sorted lists can be used:
   a, b, r, c, d – a, b, r, d, c – a, r, b, c, d – a, r, b, d, c
   They are leading to different Huffman codes, but the result will have the same average code length in each cases. Let's just use the first list in the following steps.

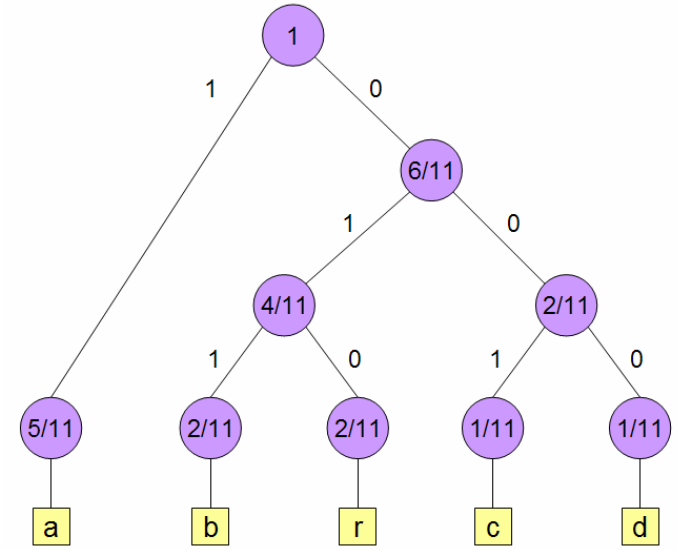
2. Each symbol is interpreted as the root of a tree




3. Two subtrees with the smallest probabilities are merged into a new subtree, which root element is assigned the probability sum. The left subtree is marked with 1, the right subtree is marked with 0
4. Step (3) is repeated until a single tree remains, containing all symbols and having a probability sum of 1

   Again, here we have several possibilities. We decide for merging c and d first, resulting in a subtree with a sum probability of 2/11. Afterwards we merge b and r giving another subtree with a sum probability of 4/11. Then we merge the two new subtrees and achieve another subtree with

probability 6/11. Finally we merge ‘a’ with that last subtree and finish the process.

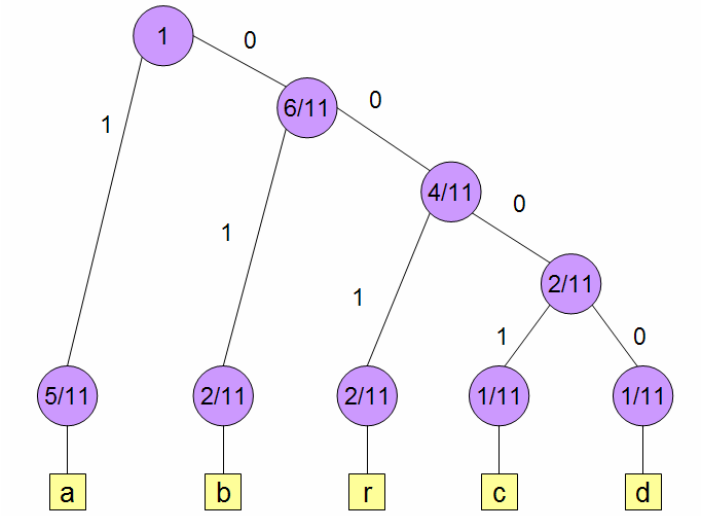



5. `The code of the leaves is given by the sequence of marks on the path from the root to the leaf`

We receive the following code:
c(‘a’) = ‘1’, c(‘b’) = ‘011’, c(‘c’) = ‘001’, c(‘d’) = ‘000’, c(‘r’) = ‘010’

Of course, also other mapping tables would be possible. One simple alternative would be to invert ‘0’s and ‘1’s in the code above, resulting in
c(‘a’) = ‘0’, c(‘b’) = ‘100’, c(‘c’) = ‘110’, c(‘d’) = ‘111’, c(‘r’) = ‘101’

Other codes could be constructed using another order in the combination of subtrees. For example, instead of merging ‘b’ and ‘r’ in the second step above, we could have merged ‘r’ with the subtree of (‘c/d’). This would result in the following code tree:



and the following code:
c(‘a’) = ‘1’, c(‘b’) = ‘01’, c(‘c’) = ‘0001’, c(‘d’) = ‘0000’, c(‘r’) = ‘001’

(d) What is the average code length of your code?
The average code word length can be determined using the formula below (see slide 261):

$$L(C) = -\sum_{i=1}^{5} p(a_i) \cdot l(a_i)\ bit$$

Since we have l($a_1$=’a’) = 1 bit , l($a_2$=’b) = 3 bit, l($a_3$=’c’) = 3 bit, l($a_4$=’d’) = 3 bit and l($a_5$=’r’) = 3 bit, we can compute

$$L(C) = \frac{5}{11} \cdot 1 + 2 \cdot \frac{2}{11} \cdot 3 + 2 \cdot \frac{1}{11} \cdot 3\ bit \approx 2.091\ \text{bit}$$

Regarding the entropy calculated above, we have a redundancy of

$$R(C) = L(C) - H(\omega) = 2.091 - 2.039 = 0.052\ \text{bit}$$

(e) Code the message using your code table. How long is the resulting bit sequence?

When we use the code word table from (c), the message will be coded as

C(‘abracadabra’) = ‘1 011 010 1 001 1 000 1 011 010 1’

The resulting bit sequence is therefore 23 bits long, which matches our expectation of having 11 symbols using 2.091 bit on average.

If we would use a normal ASCII coding table (8 bit per symbol), we would have used 88 bits. Therefore, our code has a compression ratio of about 1: 3.826. In other words, the message could be stored or transmitted using only about 26% of the original space. But we must transmit the Huffman table in addition, which lowers this effect, especially on very small messages like the one above.