Marburg University

Department of Computer Science
Prof. Dr. Sebastian Wild

Date: 2025-12-05
Version: 2025-12-05 17:41

Sheet 8 for Effiziente Algorithmen (Winter 2025/26)

Hand In: *Until 2025-12-12 18:00, on ILIAS.*

Problem 1

30 points

a) Specify the encoding for the text `TIN␣TIPTIP` using the Lempel-Ziv-Welch encoding. Provide the table (after entry 127) as your calculation method, analogous to the lecture.

s	#(s)
␣	32
...	...
I	73
...	...
N	78
...	...
P	80
...	...
S	83
T	84
...	...

b) Decode the numbers [65 65 66 129 67 132 131] which were encoded using Lempel-Ziv-Welch. Provide the step-by-step decoding and the table (after entry 127) as a workaround, analogous to the lecture.

s	#(s)
A	65
B	66
C	67
D	68
...	...

Problem 2

40 points

a) Let $T = T[0..9) = \texttt{ABBACBAAA}$ be the input text with the alphabet $\Sigma = \{\texttt{A}, \texttt{B}, \texttt{C}\}$.

Use the Move-To-Front transformation on the input with the initial list $Q = [\texttt{A}, \texttt{B}, \texttt{C}]$.

Show the state of Q after each step and the output. If present, mark each run in the output and explain how it is generated.

b) Use the *inverse* Burrows-Wheeler transform on the encoded text `OOO$DOND` to obtain the original text. The following holds: $\texttt{\$} < \texttt{D} < \texttt{N} < \texttt{O}$, where `$` marks the end of the string.

Problem 3

20 + 30 points

a) Show that in our general stochastic sequence framework, for every input string $X_0X_1 \ldots X_n = \texttt{\$}$, we have $m \leq \lg\big(1/(P_{0,X_0}P_{1,X_1} \ldots P_{n,X_n})\big) + 2$.

b) Design an algorithm that generates a sequence of n perfectly uniform, random trits, but receives only (perfectly uniform) random bits as input. Your algorithm should require at most lg(3) random bits per trit for large n. Justify the correctness of your solution.

Hint: Use arithmetic coding.

Problem 4

10 + 20 + 30 + 10 + 10 points

In this exercise, you will create a step-by-step proof-of-concept implementation of a compression pipeline in Java. This exercise (exceptionally) does not focus on the runtime of your implementation; in particular, you do not need to strive for maximum efficiency in computing the transformations.

For simplicity, we will also represent the output as a `String` containing the characters `0` and `1`.

Create a class `Compression` in which you implement the following methods.

Code must be submitted (in addition to your description) as separate, compilable `.java` file!

a) Implement a method `String eliasGammaCode(int i)` which calculates the Elias Gamma Code for an integer > 0 as presented in the lecture.

b) Implement a method `int[] moveToFront(String text)` which implements the Move-To-Front transformation from the lecture for a string.

c) Implement a method `String burrowsWheelerTransform(String text)` that applies the Burrows-Wheeler transformation from the lecture to a string. Choose a suitable letter as the *End-of-Text Character* to mark the end of the text and clearly indicate this in your solution (e.g., as a comment).

d) Implement a method `String compress(String text)` that processes the text as follows:

   Burrows Wheeler Transformation → Move-To-Front Transformation → Elias Gamma Code

   Note:

   - Since the Elias Gamma code was only handled for $n > 1$, you can increment the number by 1 before inputting it. For decoding, you would correspondingly decrement the result by 1.
   - Combine the results of the Elias Gamma Code to obtain a final result.

e) Choose your favorite book from Project Gutenberg (alternatively, your least favorite book, or any book of your choice), which is available in *Plain Text UTF-8* format. Indicate to the tutor which book you have chosen (submit a link or a manageable file size). Apply the compression pipeline to the book in a `main` method. To simplify the process, you may slightly modify the text after reading it, particularly regarding special characters, if you encounter problems with your *End-of-Text Character*. Specify the compression rate and provide a rationale.

Hint: Decoding is not required, but can be useful for detecting errors, especially for the Burrows Wheeler transformation.