Immagine che contiene testo, emblema, simbolo

Descrizione generata automaticamente

**Homework 4 – Lossless Compression**

Mobile Programming and Multimedia

Gabriel Rovesti – 2103389

Table of Contents

[2 Assignment 3](#_Toc166158023)

[3 LZW Algorithm 3](#_Toc166158024)

[4 Huffman Algorithm 4](#_Toc166158025)

# Assignment

*Encode the string:*

*abcabcabcabcffffffffffff000000000000ffffffffffffffffffffffff*

*using:*

1. *the LZW algorithm and*
2. *choose an algorithm between Shannon-Fano and Huffman.*

*Compare the two results in terms of compression ratio.*

# LZW Algorithm

Let’s first start considering the LZW algorithm on the string given:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Here, an explanation step by step based on the algorithm code taken from the [slides](https://stem.elearning.unipd.it/pluginfile.php/803212/mod_resource/content/3/mmdata.pdf).

# Huffman Algorithm

In this section, the Huffman algorithm is chosen and applied, with the following table describing each symbol, occurrences and the encoding, obtained looking at the tree obtained ( for left children, for the right children), hence considering the total number of those:

|  |  |  |  |
| --- | --- | --- | --- |
| *Symbol* | *N. of occurrences* | *Code* | *N. of bits* |
| A | 4 | 1110 | 16 |
| B | 4 | 1111 | 16 |
| C | 4 | 110 | 12 |
| F | 36 | 0 | 36 |
| 0 | 12 | 10 | 24 |

Immagine che contiene calligrafia, disegno, Carattere

Descrizione generata automaticamenteThe corresponding tree is represented here representing the encoding is shown here:

The algorithm is bottom-up, so we start from the lowest-occurrences nodes, in this case , forming a new node as sum. Given the tree would be unbalanced, the character is then summed subsequently, forming a sum node of . Continuing this way, we sum all occurrences of the nodes, reaching the root of .

We then compute how many bits are occupied, considering this is computed multiplying the number of occurrences with how many bits the single code occupies: