# Laboratory work 10

Author: Dávid Kromka

**Algorithm Description - Task 1: Redundant Encoding**

The **redundant\_encode** function takes a message represented by symbols and encodes it using a redundant code. The redundant code is constructed based on the binary representation of the symbols and a specific encoding rule.

1. **Initialization:**
   * Initialize an empty string **redundant\_code** to store the encoded message.
   * Set **prev\_value** to 0, which will be used to keep track of the previous value during encoding.
2. **Encoding Loop:**
   * Iterate through each symbol in the input **message**.
   * For each symbol:
     + Convert the binary representation of the symbol to an integer (**int(symbol, 2)**).
     + Calculate the current value using the formula **(prev\_value + int(symbol, 2)) % 2**.
     + Convert the current value to a string and append it to the **redundant\_code**.
     + Update **prev\_value** with the current value for the next iteration.
3. **Return Result:**
   * Return the final **redundant\_code** representing the encoded message.

Obrázok, na ktorom je text, písmo, snímka obrazovky

Automaticky generovaný popis

**Algorithm Description - Task 2: Huffman Table Construction**

The **huffman\_table** function constructs a Huffman coding table based on a set of symbols and their corresponding probabilities.

1. **Node Creation:**
   * Create a list of nodes, where each node represents a symbol and its probability.
   * Each node is initially assigned a **left** and **right** child as **None**.
2. **Huffman Tree Construction:**
   * While there is more than one node in the list:
     + Sort the nodes based on their probabilities.
     + Take the two nodes with the lowest probabilities, create a new parent node with their combined probability, and update the list of nodes.
3. **Code Assignment:**
   * Recursively traverse the Huffman tree to assign binary codes to each symbol.
   * The left child represents appending '0' to the code, and the right child represents appending '1'.
4. **Huffman Table Construction:**
   * Create a dictionary (**huffman\_table**) where symbols are keys and their corresponding Huffman codes are values.
5. **Return Result:**
   * Return the constructed **huffman\_table**.

