Source Coding - Creating Huffman Codes

Information Theory Lab 7

Objective

Understand binary Huffman coding by implementing an application in C for creating Huffman codes.

Theoretical notions

See lecture notes for details on the Huffman coding algorithm

Exercises

1. Study the structures and functions defined in the following files, in order to understand their purpose.
    - `huffman.h` : header file for Huffman coding functions
    - `huffman.c` : source file for Huffman coding functions
    - `bitmacros.h` : header file for bitwise operation macros
2. Write a C program that creates a Huffman code from an input data file. The program shall be called as follows:

    ```
    HuffmanCode.exe input.txt code.dat
    ```

    - The arguments are:
        - `input.txt`: the input file, from which the code is created
        - `code.dat`: the output file containing the Huffman code created (known as the “codebook” file). It shall contain a vector of 256 elements of the `CODE32BIT` structure type also used in the previous laboratories.

- The program will follow the following steps:
  - Include the accompanying header files
  - Declare a vector with 256 elements of the `CODE32BIT` structure type
  - Read the input file and compute the probabilities of every character, just like it was done in lab L02.
  - Create the Huffman code with the provided functions, in sequence:
    - initialize the Huffman tree structure
    - set the probabilities of every character
    - create the tree with `make_huffman_tree()`
    - create the codeword vector with `make_codewords()` and `to_new_codewords()`
  - Display the codewords for all characters
  - Save the codeword vector to the output file

3. Check the displayed codewords. Is it an instantaneous code or not?

Program design

- All the basic Huffman-related functions are already declared in `huffman.c` and defined in `huffman.c`. You must only create the main program and call the Huffman functions.

- A node in the Huffman tree is of a structure type `Node`, which contains:

  - the probability value
  - the assigned message (character / byte), or 0 if it is an internal node
  - the index for the parent node (or −1 if the node has no parent)
  - two indices for the left and right child nodes (or −1 if none)

- All the nodes are stored in a global array `tree` of max size 512. Each node will be identified by its index in the array. The parent/left/right indices of a node are the indices in this array of the corresponding nodes.

- The procedure of constructing the Huffman tree is split into smaller steps, each done in a separate function which acts on the global array:

  - `init_huffman_tree()`: initializes the array with default values
  - `set_prob()`: sets the probabilities of each character
  - `find_two_minima()`: returns the indices of the two nodes with least probability
  - `make_parent()`: creates a parent node for two other nodes, setting the parent/left/right indices for the affected nodes
  - `count_roots()`: returns the number of nodes that have no parent

- The tree is created step by step inside the function `make_huffman_tree()` which performs:

    - While there is more than one root:
        - get the nodes with least probability
        - create a parent for them

- After the tree is created, the codewords are obtained in a function `make_codewords()`, which fills the vector `codebook` with the codewords (arrays of integers).

- The codebook can be be converted to the more efficient bitwise structure with `to_new_codewords()`

Final questions

1. TBD
2. TBD