[[1]](#footnote-2)

Demonstration of Huffman Coding  
(June 2014)

Jarosław Kieć, Mateusz Blaszkiewicz, Electronics & Telecommunications 3rd Year,  
 AGH University of Science and Technology

Huffman coding as one of the topics of Digital Communications course have been undertaken as end of term project assignment. The *Python 2.7.3* script language has been chosen altogether with python *Qt4* package for GUI implementation and *pydot* for graphical visualization.   
Those tools has been used to present and demonstrate basic functionality and efficiency of genuine Huffman coding algorithm, referred in the application as *bottom-up* method.

Huffman coding, information theory, python, pyQt, Qt4, pydot.

# INTRODUCTION

By the means of implemented features user can easily load source/message strings of symbols into our application, generate *binary source tree*, obtain source visualization, encode and decode message. Due to the implementation of two source generation methods, source’s basic parameter indicators and easiness of symbol input one can quickly check efficiency of Huffman coding altogether with comparison to suboptimal *top-down* method. Visualization of *binary source tree* is provided by *pydot* package which is actually the interface for *Graphviz’s* Dot language. Source and message data as text can be created, edited and also loaded/saved in files.

# Configuration and Execution

User needs to install following packages:

1. *Python 2.7.3* package – provided by default with modern linux OS.
2. *python-qt4* as python implementation of popular Qt4 graphical user interface API.
3. *python-pydot*  for visualization.
4. *python-numpy* for as a logarithm with base of 2 method.

To configure your linux based system to run the project script one needs to run in *terminal* command in super user mode: *apt-get install* together with mentioned package names.

To run our application one needs two scripts: *huff\_algorithm.py* (API) and *HC\_demo.py* (GUI), their functionality may be described as follows:

1. *huff\_algorithm.py* provides classes and methods for Huffman encoding/decoding, binary source tree construction, graphical interpretation source.
2. *HC\_demo.py*  used to run application, provides graphical user interface, actually is modified script generated by *pyuic4* from *Qt Designer* file.

In order to run project script one needs to:

1. Have mentioned files in one directory,
2. Open *terminal* and access that directory
3. Run command ‘*python* *HC\_demo.py’*

# Usage Instructions

Application GUI is composed of three main parts:

1. Top menu bar with source/message data file manipulation menus.
2. Left functional part called *toolbox* (former Qt toolbox class) with two text edit widgets as symbol input for source and message. *Toolbox* also contains push buttons to initiate source tree generation, encoding and to switch between source input (*Toolbox’*s text edit widget and *Input Box* in the right winged GUI part). There are also two checkboxes one to zoom in visualization in tab 2 (right part), second to separate symbols in encoded message – good to visualize variable code length. Indicators of entropy, coding efficiency and weighted average code length are also present here. Last but not least is the presence of combobox to specify generation method: *top-down* (by default) or *bottom-up*.
3. Right winged part with *Input Box* for source designation with two button to add/remove symbols in this method of input, three text non-editable areas to show current source symbols and their probability, show encoded and decoded message. On the other tab the current generated binary source tree is visualized.

Functionality hints:

1. Source/message input does not accept all ASCII codes, especially those recognized as special. Limitations exist due to fact that the strings and *pyDot* uses those.   
   All alphabet letters are accepted.
2. Input Box input has its restriction according to total (summary) probability limit equal 1.0. User can lock (checkbox) the symbol’s probability. If the total limit is reached further incrementation of probability will result in decrementation of unlocked symbol situated below until it reaches 0.
3. Message box has its character limitations as well. What is more error information is showed when symbol not predicted by source is encountered.
4. To perform your first source tree generation one needs to type in on the left side some source text, push ‘*Create Encoding Dict.*’. Then with the same characters fill next text editable area and hit ‘*Start Encoding*’. After successful operation right non-editable text area should be filled also second tab should be filled with graphical visualization of source tree.

# Summary

Final remarks:

1. Things we would like to do is to organize classes and functions in scripts, describe their functionality; random data generation and visualize encoding dictionary.
2. All task from 3.0 mark criteria have been fulfilled: Correct operation of the code construction function and the encoding and the decoding routines; measurement of the source's entropy, coding efficiency, average weighted code length.
3. Most tasks from 3.5 mark criteria have been fulfilled: Ability to specify the details of the message source as well as an input sequence of symbols; ability to encode manually provided messages, and whole files. Visualization of the code tree; random data generation have not been implemented.
4. Lack of 4.0 mark criteria: implementation of symbol grouping; evaluation of the impact on coding efficiency.
5. Lack of 4.5 mark criteria: implementation of a generalized Huffman code for an arbitrary M­ary alphabet.

References

1. D.J.C. MacKay, “Data Compression,” in *Information Theory, Inference, and Learning Algorithms, 4*th ed. Cambridge

1. [↑](#footnote-ref-2)