

Politechnika Śląska  
Wydział Informatyki, Elektroniki i Informatyki

# Fundamentals of Computer Programming

Assignment number: 6  
Project's topic: Huffman

---

|             |                          |
|-------------|--------------------------|
| author:     | Michał Grochowski        |
| instructor: | dr inż. Tomasz Garbolino |
| year:       | 2021/2022                |
| lab group:  | Friday, 11:45-13:15      |
| deadline:   | 2022-02-21               |

---



## 1 Project's topic

Implement a program that compresses files with the Huffman's method. The program is run in command line with switches:

-i input file

-o output file

-m mode: c – compress, d – decompress

-d dictionary file (created in compression, used in decompression)

## 2 Analysis of the task

Compression of the data has been something that revolutionized digital media. Compression algorithms are used to reduce the size of a file while ensuring that files can be fully restored to their original state. There is a very wide range of algorithms that can be used to perform lossless data compression. The simplest known data compression algorithm is the Run-Length Encoding (RLE). Unfortunately, it has some drawbacks (quite significant ones) - there are no repeated characters, and the size of output data can be two times more than the size of input data. To eliminate this problem Huffman Coding was used. Huffman coding algorithm was introduced by David Huffman in 1952.

The main purpose of information theory is to convey information in as few bits as possible. Huffman's coding has the same aim and is largely successful. The algorithm stands on creating a tree in the form of a leaf node and its children which has a probability of the frequent appearance of the characters. So, Huffman algorithm has more efficiency in file compression with greater compression ratio.

In the era of the internet, the transmission of data depends on time. More data takes more time to transmit. Huffman coding reduces the data size thus data transmission time is significantly reduced. It is the basis of many data compression and encoding schemes. mainly because it is a lossless data conversion.

This Algorithm is used for compressing / decompressing data - reducing its size without losing any information. It consists on two major steps:

1. Generating a Huffman tree from the input characters
2. Traversing the Huffman tree to assign codes to the characters (encoding)

For decompression, the Huffman tree must be reconstructed.

So the final objectives are:

- Compressing data without losing data
- Statistical coding - more frequently used symbols should have shorter code words
- Information transmission in fewest possible bits

## 2.1 Data structures

In my project implementation I have used the priority queue and the binary tree as data structures.

Binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.

A priority queue is an abstract data-type in which each element additionally has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority.

## 2.2 Space requirements and time target

Space requirement:  $O(n)$  The algorithm creates a tree with a maximum of  $2n - 1$  elements.

Time requirement:  $O(n \log n)$  The time requirement for adding to the stack is  $O(\log n)$  and is done  $n$  times in the worst case.

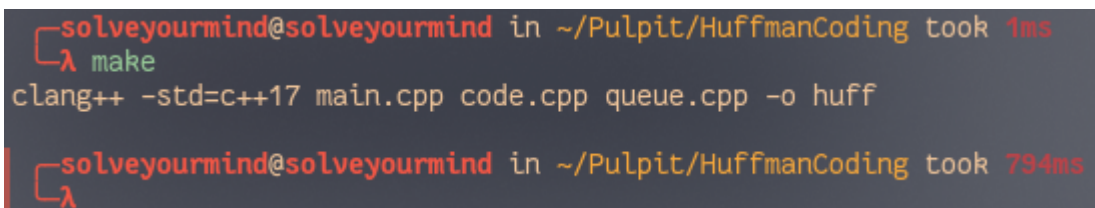
## 2.3 General assumptions

The program gets the data to be compressed or decompressed into the stdin and outputs the corresponding data to the stdout.

## 3 External specification

My script implementation is a command line program. For the purpose of preparing my solution, I have learned how to use 'make' so it is incipiently included. Therefore, to generate the "huff" executable file you must use the command:

```
$ make
```



```
solveyourmind@solveyourmind in ~/Pulpit/HuffmanCoding took 1ms  
λ make  
clang++ -std=c++17 main.cpp code.cpp queue.cpp -o huff  
solveyourmind@solveyourmind in ~/Pulpit/HuffmanCoding took 794ms  
λ
```

Program gleans two different options:

–encode

or

–decode

Furthermore it requires names of input and output files. The input file declaration stands without a parameter and output file is preceded by a “-o” flag, eg:

```
$ ./huff –encode input_to_encode.txt -o encoded_output.txt  
$ ./huff –decode encoded_input.txt -o decoded_output.txt
```

Program called without any parameters prints a short manual:

```
solveyourmind@solveyourmind in ~/Pulpit/HuffmanCoding  
λ ./huff  
Possible script options:  
--encode  
--decode  
  
Usage example:  
./huff --encode sample_text_file.txt -o encoded_sample_text_file.txt  
./huff --decode encoded_sample_text_file.txt -o decoded_sample_text_file.txt
```

Program called with incorrect input file name prints an error message:

```
solveyourmind@solveyourmind in ~/Pulpit/HuffmanCoding took 1ms  
λ ./huff --encode sample_text.txt -o sample_output.txt  
failed to open the input file!
```

Otherwise, it displays a short manual.

## 4 Internal specification

The program is implemented with a structural paradigm. User interface is separated from the program's logic.

### 4.1 Internal code description

The program accepts the input file in the manner defined above. Calculates the frequency of occurrences. On this basis creates a binary tree - "huffman tree" - and huffman code from these tree(traverse the tree), encodes the tree data, after rewriting the bits to bytes, writes the result to the output file.

When decoding, the previously encoded file is opened - the bytes are converted into bits, the binary tree is regenerated on the basis of which the data is decoded. Restoring the compressed data to its original form is based on a data storage algorithm.

---

### 4.2 Description of types and functions

Description of types and functions included in the target files.

## 5 Testing

The program has been tested with various types of files. An empty input file does not cause failure - an empty output file is created.

1 of the sample test provided is included into solution directory in img folder. However, the script did not encounter any problems with slightly larger files.

## 6 Deficiencies according to my assessment

- obtaining problems with large (or at least very large) files
- could be more responsive to error situations
- can reserve a significant amount of memory

## 7 Raw work report

Weekly rep1:

To do and at progress:

- Topic selected
- Collecting needed materials

What have i learned:

- Creating a makefile
- How huffman's algorithm works and it's activities on a general level

Weekly rep2:

To do and at progress:

- Heap implementation started
- Huffman implementation started. Characters (bytes) are counted and entered into heap
- Collecting more needed informations

What have i learned:

- Huffman's operation at a deeper level
- Making a heap with an array

Weekly rep3:

To do and at progress:

- Combining huffman into a pile
- Huffman coding done

What have i learned:

- Explored the knowledge of compression algorithms
- Delved into the huffman algorithm

Weekly rep4:

To do and at progress:

- Looking for ideas of optimization
- Code debugging and making it more clean

What have i learned:

- Deeping with knowledge about c++ bugs and solving them

Weekly rep5:

To do and at progress:

- own priority queue
- management error - blank text breaks the program

What have i learned:

- Heapifs can be different

Weekly rep6:

To do and at progress:

- Implemented Queue works in the same way as `std::priority_queue`
- Implemented Queue is as fast as `std::priority_queue`



- started restoring compressed data by implementing data storage

What have i learned:

- You should stay away from bits unless you are doing something embedded

Weekly rep7:

To do and at progress:

- Saving compressed data
- Compressed data decompression completed
- My own rudimentary vector implementation

What have i learned:

- Bits
- C++ templates

## Sources for acquiring knowledge

- [1] [https://en.wikipedia.org/wiki/Huffman\\_coding](https://en.wikipedia.org/wiki/Huffman_coding)
- [2] <http://bigocheatsheet.com>
- [3] <https://riptutorial.com/cplusplus/example/16851/sharppragma-once>
- [4] <https://stackoverflow.com/questions/759707/efficient-way-of-storing-huffman-tree>
- [5] <https://www.geeksforgeeks.org/little-and-big-endian-mystery/>
- [6] <https://www.youtube.com/watch?v=B3y0RsVCyrw>
- [7] <https://www.geeksforgeeks.org/heap-sort/>
- [8] <http://cslibrary.stanford.edu/110/BinaryTrees.html>
- [9] <https://www.youtube.com/watch?v=XDxLEUgVDMM>
- [10] Opus Magnum c++11, Jerzy Grębosz
- [11] Introduction to Algorithms, Cormen Thomas H.

Project link: <https://github.com/ViciousCircle-Github/Huffman-Coding>