**Binary File Compression: Huffman Encoding**

By Vasavi Vempati

11/21/2019

**Table of Contents**

1. Introduction 2

1.1. Document Outline 3

1.2. Document Description 5

1.2.1. Introduction 5

1.2.2. System Overview 5

2. Design Considerations 5

2.1. Assumptions and Dependencies 6

2.2. General Constraints 6

2.3. Goals and Guidelines 6

2.4. Development Methods 7

3. Architectural Strategies 7

4. System Architecture 7

4.1. Subsystem Architecture 8

5. Policies and Tactics 9

6. Detailed System Design 10

6.1. Classification 10

6.2. Definition 10

6.3. Responsibilities 10

6.4. Constraints 10

6.5. Composition 10

6.6. Uses/Interactions 10

6.7. Resources 11

6.8. Processing 11

6.9. Interface/Exports 11

6.10. Detailed Subsystem Design 11

7. Glossary 12

8. Bibliography 12

# Introduction

The following is a detailed description of all the technologies used in order to create my binary file compression web application. The language I chose to write the application in is Python 3.8. I chose this language because I liked some of the comparison and iteration techniques available in this version of python. I also chose to use the python web micro-framework Flask because it is very lightweight, easy to set up, and easy to test with. In addition to Flask, I used Flask-Bootstrap for UI manipulation and design. Finally, I used Anaconda to create a Conda environment to manage all my dependencies so that the libraries I used would not conflict with those installed in my main development environment. I chose not to deploy to web-server because I believe my application is simple enough that it will be fairly easy to run locally.

## Document Description

Below I will describe the approach I took to solving this file compression problem, the technologies I used, and the significance of the output produced by my program in relation to the sample data provided. I will also describe how to set up and run my application locally after cloning the repository from GitHub (<https://github.com/vvempati/huffmanEncoding>) .

### Introduction

* The purpose of this document is to explain my approach to file compression via Huffman encoding.
* The scope of this document will cover launching the application and explaining the code written.
* This document is intended for anyone who wants to learn about Huffman encoding and why it is a great lossless compression technique.
* I am using Python 3.8, Flask 1.1.1, Flask-Bootstrap 3.3.7.1
* I used this guide: <https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/>

### System Overview

This software system consists of a Python web application deployed using the python web micro-framework: Flask. The application consists of a folder called ‘app’ which contains all the pertinent classes and files. The ‘app’ folder consists of three classes ‘\_\_init\_\_.py’, ‘routes.py’, and ‘huffman.py’. The first class ‘\_\_init\_\_.py’ imports all the required dependencies and creates instances of Flask and Bootstrap that are needed for the rest of the application. The second class ‘routes.py’, describes the three different URI’s (uniform resource identifies) related to the homepage (‘/index’), the compression page (‘/compress-data’), and the final output page(‘/return-files’). The third class ‘huffman.py’ details the file parsing, creation of the Huffman binary tree in the form of a min-heap and finally compressing the file input into a smaller size using the Huffman binary tree. The ‘app’ folder consists of three sub-folder called ‘outputs’, ‘templates’, uploads’. The ‘outputs’ folder contains the output file from the compression algorithm. The ‘templates’ folder contains the HTML UI templates for displaying the data on the index, compression, and output pages. The ‘uploads’ folder contains the input binary files uploaded by the user for compression.

# Design Considerations

## Assumptions and Dependencies

Assumptions:

* + Related software or hardware
  + Operating systems
  + End-user characteristics
  + Possible and/or probable changes in functionality

## General Constraints

Describe any global limitations or constraints that have a significant impact on the design of the system's software (and describe the associated impact). Such constraints may be imposed by any of the following (the list is not exhaustive):

* Hardware or software environment
* End-user environment
* Availability or volatility of resources
* Standards compliance
* Interoperability requirements
* Interface/protocol requirements
* Data repository and distribution requirements
* Security requirements (or other such regulations)
* Memory and other capacity limitations
* Performance requirements
* Network communications
* Verification and validation requirements (testing)
* Other means of addressing quality goals
* Other requirements described in the requirements specification

## Setup and Deployment

Describe any goals, guidelines, principles, or priorities which dominate or embody the design of the system's software. Such goals might be:

* The KISS principle ("Keep it simple stupid!")
* Emphasis on speed versus memory use
* working, looking, or "feeling" like an existing product

For each such goal or guideline, unless it is implicitly obvious, describe the reason for its desirability. Feel free to state and describe each goal in its own subsubsection if you wish.

## Conclusion

Briefly describe the method or approach used for this software design. If one or more formal/published methods were adopted or adapted, then include a reference to a more detailed description of these methods. If several methods were seriously considered, then each such method should be mentioned, along with a brief explanation of why all or part of it was used or not used.