

Huffman Encoder

CS21120 Assignment



May 12, 2016

Jakub Janas – jaj48

Contents

[1. Introduction 2](#_Toc468747055)

[1. Huffman encoding scheme 2](#_Toc468747056)

[2. System design 3](#_Toc468747057)

[3. System implementation and operation 4](#_Toc468747058)

[4. Time and space complexity 4](#_Toc468747059)

[5. Conclusion 4](#_Toc468747060)

# Introduction

In this assignment we were to implement the Huffman encoding scheme in Java programming language. This scheme is a very efficient encoding scheme which is about 66% more efficient than the most popular American Standard Code for Information Interchange (ASCII).

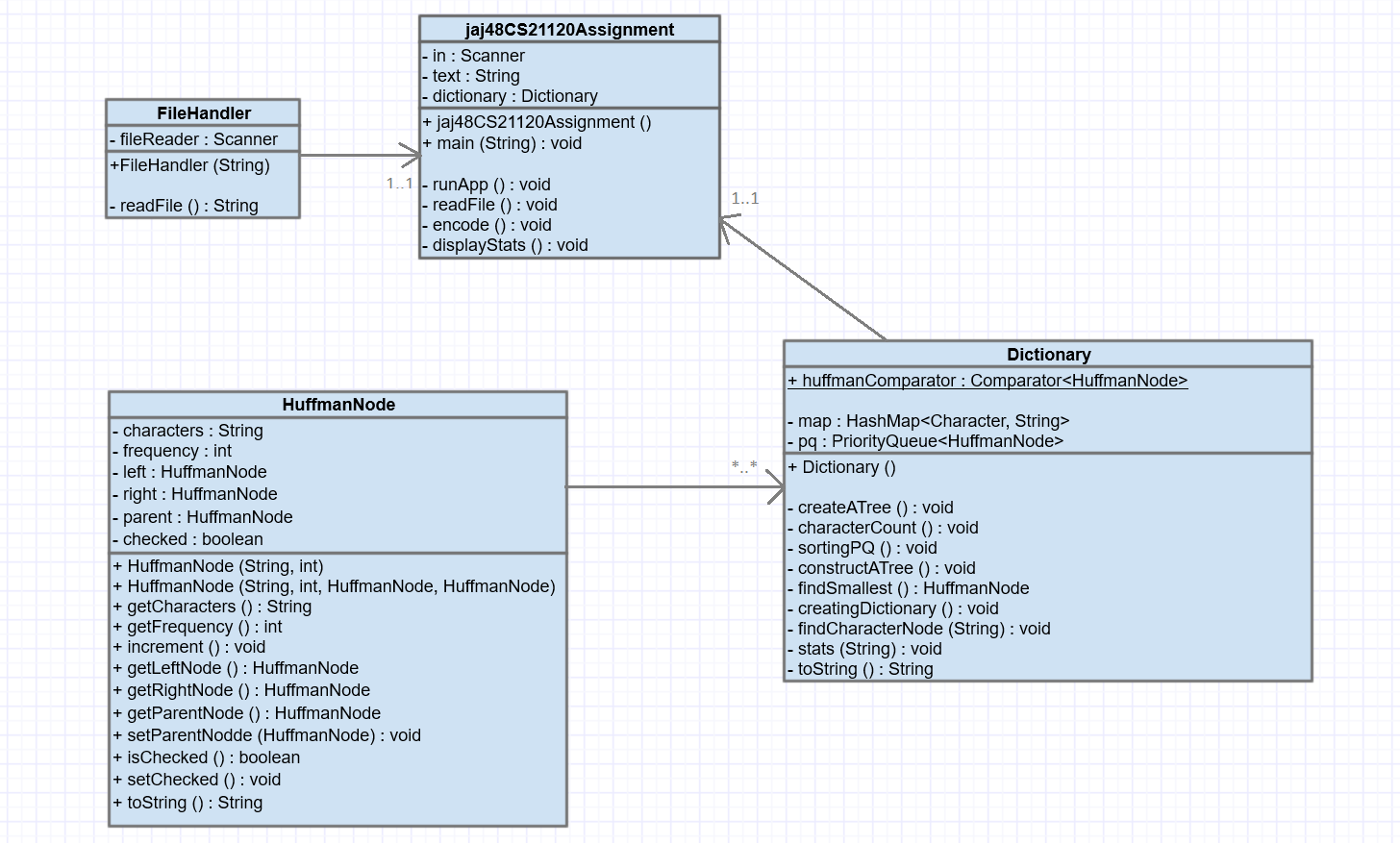
## Huffman encoding scheme

Every piece of data is encoded in a different way. It is based on the idea that the most common character should have the shortest code. In this way we can shorten the overall code by about 66%, meaning there will be only 33% of the characters of the usual ASCII encoding as a product of this encoding scheme. There are four stages to create a “dictionary” for every character:

* Count the frequency of every character that appears in the data
* Sort it in descending order
* Build the binary tree: the bottom row are the basic characters that you already have from the previous step, find 2 smallest frequencies and add them together creating a new element, repeat that step until you run out of elements to add together
* Traverse the tree for every character and generate the code by adding 0s and 1s: 0 if the lower element in the path is on the left side and 1 if the element is on the right size

After that you have a “dictionary” ready to encode the data. In my opinion I have succeeded in implementing that system correctly and my program provides all the specified information.

# System design



My program consists of four classes. Jaj48CS21120Assign is my application class. It has 3 instance variables: the scanner for taking input (just for getting the file path), string for holding the contents of the file and dictionary which is an object that encodes and holds the dictionary for the file. The application is also instantiating a FileHandler object which open a FileReader stream and reads in the contents of the file into a single string. It is then returned to application and the passed to the Dictionary object for encoding. The encoding is carried out in four steps:

* Counting the frequency of the characters and putting it into the hashmap
* Sorting by putting the characters and frequency into a priority queue, with a custom comparator that sorts in descending order, elements are of class type HuffmanNode
* Creating a binary tree, by adding the two smallest frequencies and creating a new HuffmanNode with a string of the characters from the children and sum of frequencies, repeating until the smallest frequency is the root
* Traversing the binary tree from bottom to the top, generating a binary code for each character by adding 0 if the current node is on the left of the parent node and 1 if on the right

While traversing a tree I am also storing different depths of the tree. This is necessary to calculate the average depth of the tree. After that the only thing that is left to do are the statistics. The uncompressed size is just the number of characters multiplied by 8 and the compressed size is each character’s frequency multiplied by the number of characters in it’s binary code and added together. Then the height which is the biggest depth of the tree, and number of nodes which is just size of the binary tree.

# System implementation and operation

# Time and space complexity

# Conclusion