A3-Huffman-Coding

Stuart Thiel

May 27, 2023

Introduction

This assignment is worth **5%** of your grade. This is an **individual assignment** (so individual, I put *your* student ID on it) and you should not share your assignment with anyone else. The **assignment is due June 16th at 11:59 pm, Montreal time**.

All submissions must go through Moodle. I like EAS, but the new VPN rules suck.

Q1) Huffman Coding Question

1.1 General Instructions

This assignment involves understanding and implementing Huffman coding trees. You will be asked to discern the strategies used to generate two provided Huffman coding trees and then construct your own corresponding tree.

The priority queue utilized in this assignment is stable, but its initial configuration is unknown. It can be one of the following:

- Ordered based on the first occurrence in the sample data file used to build the frequencies.
- Ordered based on ASCII code.
- Ordered in reversed ASCII code.

By comparing with the example graphs provided, you should be able to identify variations in the tree construction.

1.2 Programming Task

Your main task is to develop a program named `HCDriver.java`. This program should define a single class named HCDriver, which is part of the `ca.concordia.algos.huffman` package. You might need to create additional supporting classes, and these should be included in the same package.

Your program will accept two command-line arguments, which are strings:

1. The filename of the source file used to construct your Huffman coding tree.
2. Either 'encode' or 'decode'.

Depending on the second argument, your program should either encode or decode the input it receives:

1. If 'encode' is chosen, your program should accept a line of text via standard input and encode it using the Huffman coding tree.

2. If 'decode' is chosen, your program should accept a sequence of zeros and ones via standard input and decode it using the Huffman coding tree.

Please ensure that your program correctly handles both encoding and decoding processes.

1.3 Additional Requirements

Your program should convert all input to lowercase. Even though your haiku might contain capital letters, the DebugRunner expects all input to be in lowercase to allow for smaller trees.

You will be given two examples for this assignment and will be marked on a third one. Please note that the program takes a filename as input, so you should not hardcode filenames or file content in your program.

In addition to the programming task, you will also need to answer three theory questions related to the assignment.

1.4 Theory Questions

Please provide your answers to the theory questions in comments at the top of your HCDriver.java file. Your responses should not exceed 100 words each. The questions are:

1. What is the purpose of using a priority queue in Huffman coding, and how does it help to generate an optimal code?
2. How does the length of a Huffman code relate to the frequency of the corresponding symbol, and why is this useful for data compression?
3. What is the time complexity of building a Huffman code, and how can you optimize it?

1.5 Evaluation

The zip file for this assignment includes not only the assignment PDF but also a `DebugRunner.class` file, a `config.xml` file and two sample haikus. The `DebugRunner` can be executed using Java to provide feedback on your code. The `config.xml` file serves as a partial grading tool, allowing you to assess your progress in the assignment.

After considering feedback from previous assignments this semester, it appears that the most straightforward method of utilizing `DebugRunner` is by copying the contents of your `bin` folder into the directory containing `DebugRunner.class`, `config.xml`, and the sample haikus. Once done, you can execute the command line demonstrated in the examples to run `DebugRunner`.

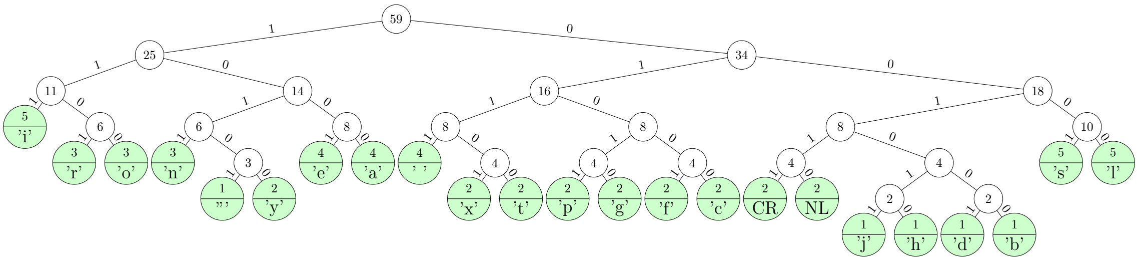
The programming component constitutes 60% of your grade, with a part of these marks coming from the example test suite provided and the rest from a mystery test set. The remaining 40% of your grade will be derived from your responses to the theory questions.

Example 1

`haiku1.txt` has this text:

```
astrophysicist
exploring galaxies far
Jocelyn Bell's find
```

It should build a Huffman Coding Tree like this:



If you type: *java ca.concordia.algos.huffman.HCDriver haiku1.txt encode*
>explorin

The output would look like:
100101101010110000110011011111011

Subsequently running *java ca.concordia.algos.huffman.HCDriver haiku1.txt decode*
>100101101010110000110011011111011

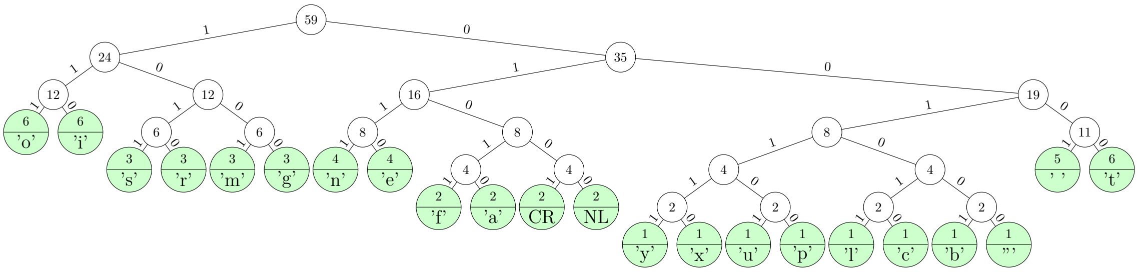
The output would look like:
explorin

Example 2

`haiku2.txt` has this text:

```
time complexity
measuring sorting's effort
big O notation
```

It should build a Huffman Coding Tree like this:



If you type: *java ca.concordia.algos.huffman.HCDriver haiku2.txt encode*

```
>big O no
```

The output would look like:

```
0010011101000000111100010111111
```

Subsequently running *java ca.concordia.algos.huffman.HCDriver haiku2.txt decode*

```
>0010011101000000111100010111111
```

The output would look like:

```
big o no
```

1.6 Guidance

Understanding Huffman Coding

This assignment centers around a core topic: Huffman Coding. It is crucial that you comprehend this concept thoroughly before proceeding with the tasks at hand.

Huffman coding is a specific type of data encoding algorithm used for lossless data compression. It uses a variable-length code table for encoding a source symbol where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol.

In particular, pay close attention to the following components of Huffman coding:

- **Priority Queue:** This data structure is vital for building the Huffman Tree. The nodes of the tree, each representing a symbol and its frequency, are added to the queue in order of frequency. In this particular example, you should write your own Priority Queue and you should ensure that it is stable.
- **Huffman Tree:** A binary tree where each node represents a symbol and its frequency. The process of generating this tree is iterative. In each iteration, two nodes with the smallest frequencies are removed from the priority queue, combined to form a new node (their frequencies added together), and this new node is added back into the queue. This process repeats until there is only one node left in the queue - the root of the Huffman Tree.

- **Optimal Encoding:** Once the Huffman Tree is constructed, it's used to generate the Huffman Code. This is an optimal prefix code, meaning that no character's encoding is a prefix of another's, and characters that occur more frequently have shorter codes than those that occur less frequently.

Understanding these elements will be instrumental for successfully completing this assignment. Remember to thoroughly review the underlying theory before you start coding.

Consulting Resources

During your journey through this assignment, don't hesitate to turn to various resources available to you whenever you're unsure or encounter difficulties. However, you should not be looking for existing code answers to this problem or similar problems. The goal is to program it on your own. You may still look up API references and simple tasks (like reading from a file).

- **Textbooks:** Your textbooks may provide valuable insights or explanations about the concepts covered in this assignment.
- **Online Resources:** There are countless educational platforms, forums, and tutorials available online that may assist you in understanding Huffman coding better.
- **Instructor, TAs & Classmates:** Never underestimate the value of collaborative learning. If you're stuck, reach out to your instructor or discuss with your TA or classmates. Please remember to keep the discussions high-level and avoid sharing explicit solutions to preserve academic integrity.

Remember, asking for help is not a sign of weakness, but a step towards greater understanding. Also note that helping others is an opportunity to learn to think about something you already know in a new way.

Time Management

This assignment is complex and demands a significant amount of time and attention. Therefore, it is recommended to approach it with good time management.

- **Start Early:** The sooner you start, the more time you will have to handle unexpected challenges and thoroughly test your solution.
- **Allocate Sufficient Time:** Make sure you allow adequate time not only for the implementation but also for understanding the underlying concepts and debugging your solution.
- **Test Thoroughly:** Rigorous testing is as important as implementation. Allocating time for this will ensure that your solution works under various scenarios and edge cases. You may extend your DebugRunner config.xml or write your own tests to enhance this process.

Remember, time is of the essence, and managing it wisely can be the key to successfully completing this assignment.

Grading Criteria

The grading for this assignment will be as follows:

- 36% of your grade will be based on the output of the DebugRunner on your submitted code. This means that the final version of your program should work correctly and produce the expected output.
- 24% of your grade will be based on our evaluation of your code quality. This includes organization, readability, style, and adherence to coding best practices.

The remaining 40% of your grade will be awarded based on your responses to the theory questions. These questions are intended to evaluate your comprehension of Huffman coding, its practical application, and the associated data structures. Every question is purposefully framed to gauge your understanding of various aspects of Huffman coding, spanning from practical implementation to theoretical fundamentals.

Your answers should succinctly illustrate your grasp of the core principles, your capacity to employ these concepts in real-world situations, and your recognition of how these concepts impact Huffman coding's efficiency and effectiveness. Ensure that your explanations are precise and to-the-point, fitting within the 100-word limit.

While it's important to provide a concise answer, do not forget the objective of these questions is not merely to test your memory of facts. Instead, they aim to probe your deep understanding, along with your ability to analyze, evaluate, and apply Huffman coding's principles. Remember, clarity and depth of understanding are more valuable than volume in this context.

Q2) Theory 1

What is the purpose of using a priority queue in Huffman coding, and how does it help to generate an optimal code?

Q3) Theory 2

How does the length of a Huffman code relate to the frequency of the corresponding symbol, and why is this useful for data compression?

Q4) Theory 3

What is the time complexity of building a Huffman code, and how can you optimize it?