

The Implementation and Comparison of the BCCBT Data Compression Algorithm

Aiden Taylor - B.Sc. in Computer Science
Noah Pinel - B.Sc. in Computer Science
Ty Irving - B.Sc. in Computer Science

Apr. 11th, 2023

- 1 Quiz Questions
- 2 Pseudocode
- 3 Example
 - Complete Binary Trees and Frequencies
 - Encoding and Decoding
- 4 Test Results
 - Factors
 - Test Result Graphs
 - Summary
- 5 Conclusion
- 6 Q&A

1. What specific type of Binary Tree is used in the implementation of the BCCBT Data Compression Algorithm? Describe at least one discussed property of this type of Binary Tree.
2. Given the specific binary tree needed for the BCCBT algorithm, where the tree's nodes correspond to symbols in an arbitrary alphabet Σ . Denote a symbol ϕ , such that ϕ is in our arbitrary alphabet Σ . Note, ϕ is **NOT** the root of the tree. How is the bit code generated for the symbol ϕ ?
3. Does the BCCBT Data Compression Algorithm make use of the frequency/probability of each unique symbol in the source file? Explain why or why not.

Bit Code Complete Binary Tree (BCCBT) PSEUDOCODE OF THE BCCBT ALGORITHM

ENCODING

```
Get the frequency of each symbol from the input stream
Set the frequency table to the frequency of each symbol
Create a complete binary tree using the frequency table
Set the bit codes according to where the symbols are in the tree
While more symbols to read from the input stream
    Read one symbol from the input stream
    Write the symbol's bit code to the bit code stream
    Write the length of the bit code to the level stream
Compress the level stream with a lossless algorithm
Write the frequency table to the output stream
Write the compressed level stream and the bit code stream to the output stream
```

DECODING

```
Read the frequency table from the input stream
Create a complete binary tree using the frequency table
Read the compressed level stream from the input stream
Uncompress the compressed level stream
Read the bit code stream from the input stream
While more levels to read from the level stream
    Read one level from the level stream
    Read level bits from bit code stream
    Find the symbol in the complete binary tree using the level and the bit code
    Write the symbol to the output stream
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec

Properties of Complete Binary Trees:

- All levels are completely full except possibly the lowest level.
- Filled from left to right at each level. i.e. Tree leans left.
- The number of nodes at level n is 2^n .



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec

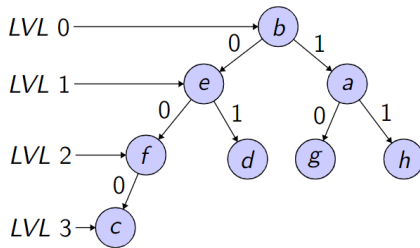
Step 1: Find the frequency Table

Given an alphabet $\Sigma = \{a, b, c, d, e, f, g, h\}$ where each symbol has the following frequency

Symbol	Frequency
b	55
e	37
a	32
f	26
d	19
g	9
h	7
c	4

Step 2: Generate Complete Tree and Get Bit Codes

Symbol	Frequency
b	55
e	37
a	32
f	26
d	19
g	9
h	7
c	4



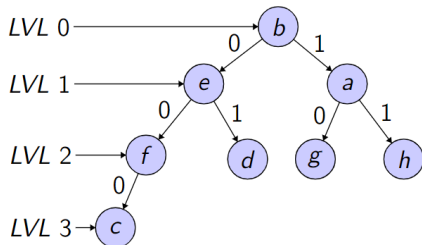
Wait, these bit codes are not uniquely decodable, how do we fix it?

BIT CODES							
a	b	c	d	e	f	g	h
1	NULL	000	01	0	00	10	11

Step 3: Making Uniquely Decodable Strings

The solution, append the symbols level to its bit code. The bit strings are now uniquely decodable.

Ex) Say we want to encode the string 'feed', encoding would look like this: [2]00[1]0[1]0[2]01, where the number between the [] specifies the level in the tree.



BIT CODES							
a	b	c	d	e	f	g	h
1	NULL	000	01	0	00	10	11

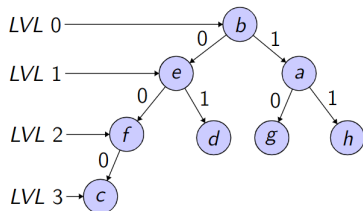
Step 4: Decoding an Encrypted String

Encrypted String: [1]0[2]01[2]10[1]0

We first look at [1]. This is telling us the symbol is in LVL 1. The next bit, 0, tells us how we should walk the tree, in this case to the left once. We arrive at symbol e. Iterating through this n times:

- ① [1]0 \rightarrow e
- ② [2]01 \rightarrow d
- ③ [2]10 \rightarrow g
- ④ [1]0 \rightarrow e

[1]0[2]01[2]10[1]0 \Rightarrow 'edge'

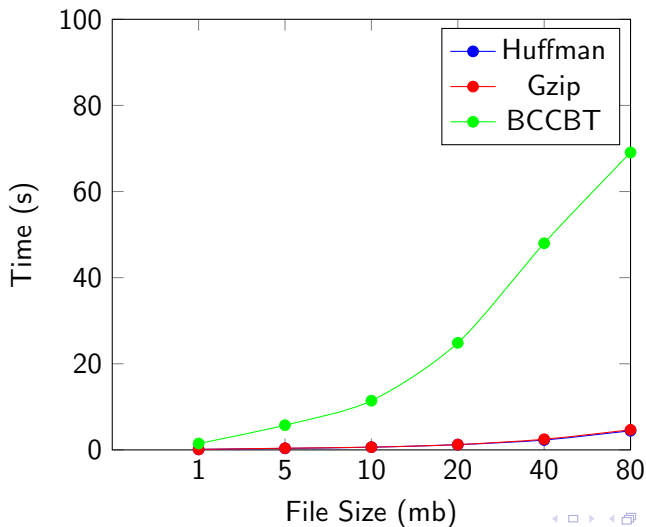


BIT CODES							
a	b	c	d	e	f	g	h
1	NULL	000	01	0	00	10	11

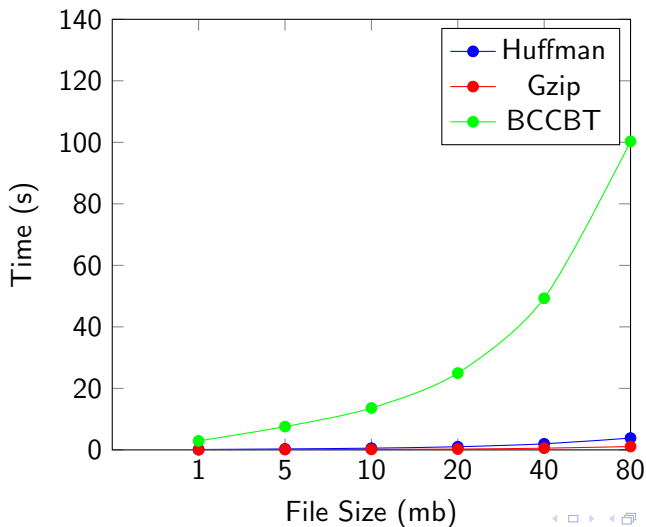
Some factors we will be using to analyze and compare these algorithms are as follows:

- ① Compression Time
- ② Decompression Time
- ③ Saving Percentage = $\frac{\text{Original File Size} - \text{Compressed File Size}}{\text{Original File Size}}$
- ④ Compression Ratio = $\frac{\text{Compressed File Size}}{\text{Original File Size}}$

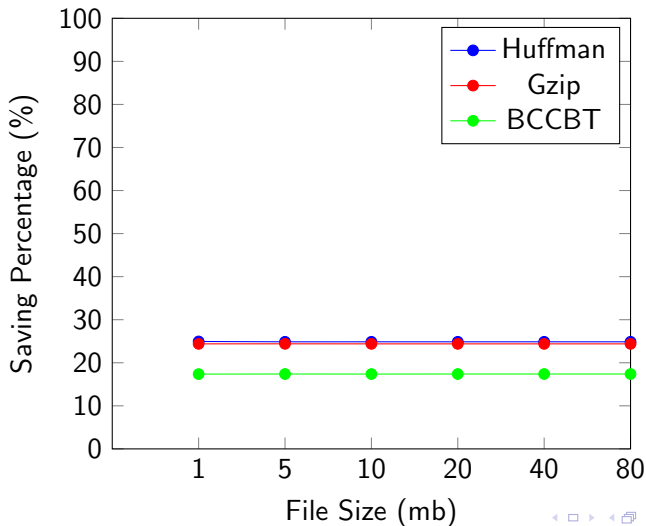
Comparison of Compression times



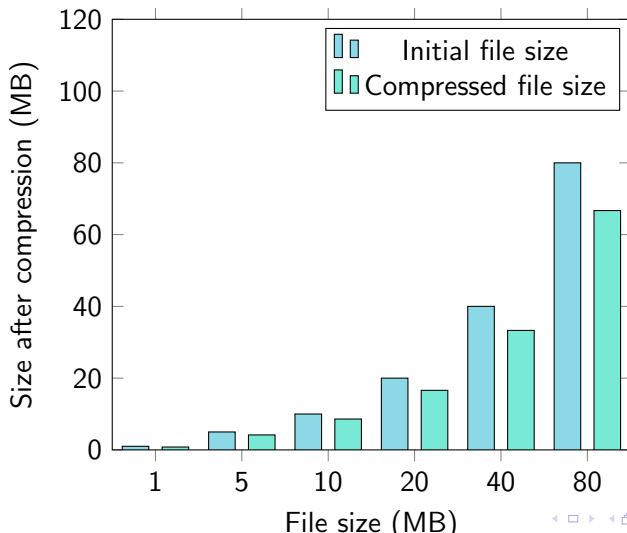
Comparison of Decompression times



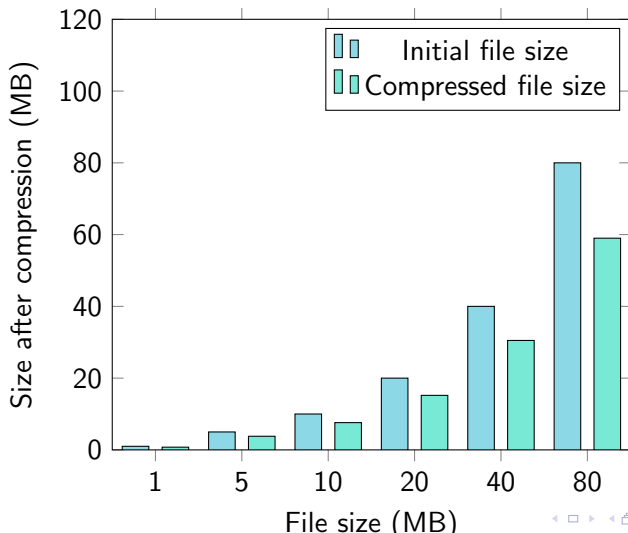
Comparison of Saving Percentage



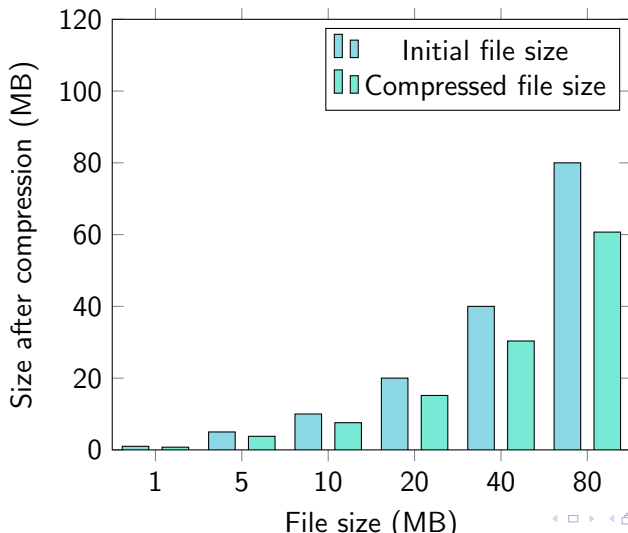
BCCBT Compression Sizes



Gzip Compression Sizes



Huffman Compression Sizes



Result Summary

Huffman and GZip both performed better than the BCCBT algorithm especially during compression and decompression times. The main factor behind this run time for the compression and decompression time is how it was implemented in our code with traversing the tree, this algorithm also requires a lot of memory space for large files.

- BCCBT was slightly behind GZip and Huffman for saving percentages however the gap between them did not grow.
- BCCBT fell behind heavily in compression and decompression times as files sizes got larger.

Ty should probably add a slide to wrap up what he just talked about, before we say we are done rn the transition does not look natural

Thank you for listening! Any questions?