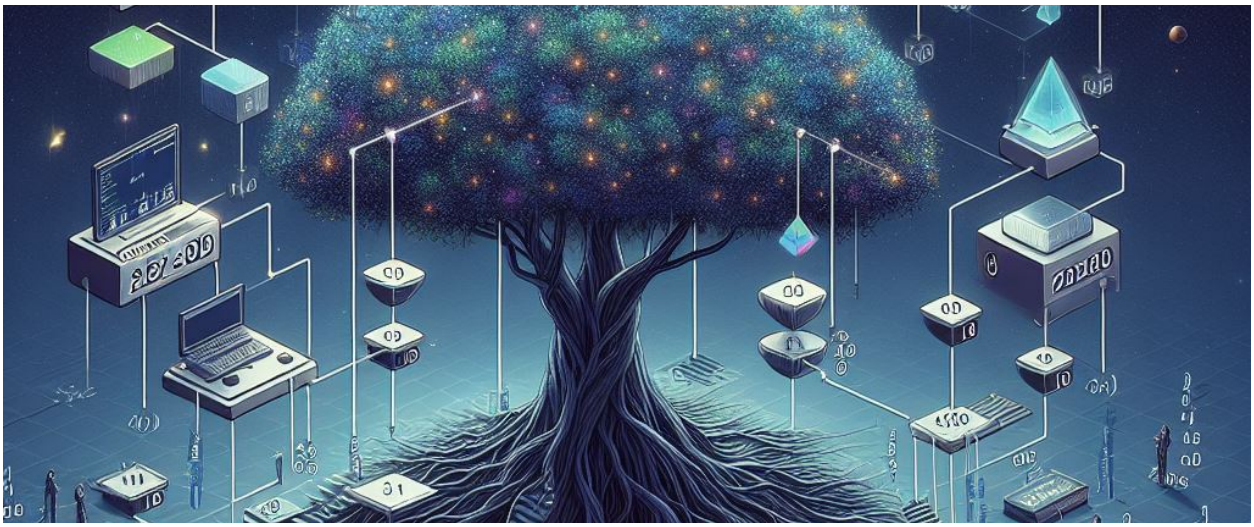


Huffman Algorithm

1. Introduction:

The Huffman Algorithm, developed by David A. Huffman in 1952, is a widely used algorithm for lossless data compression. This report aims to provide a thorough understanding of the Huffman Algorithm.

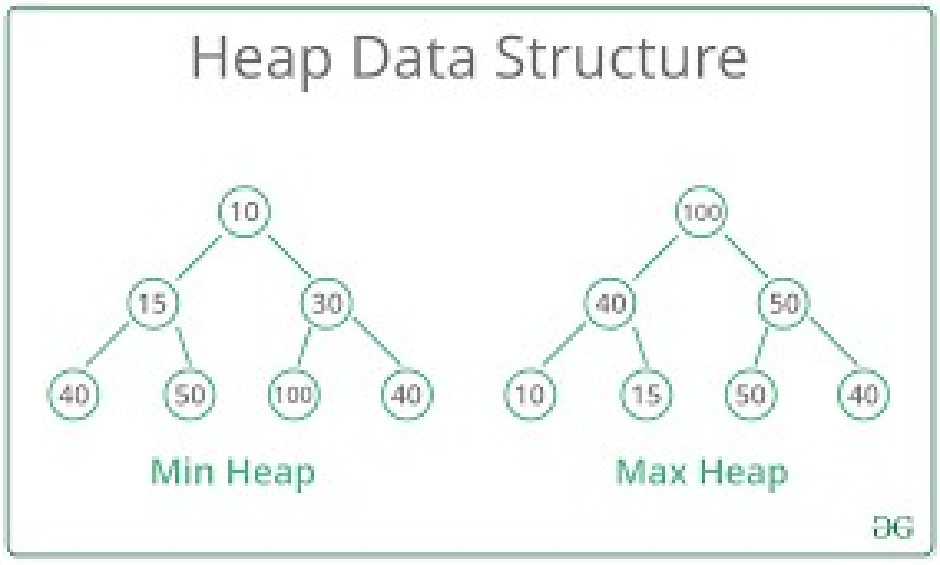
2. Definition:

The Huffman Algorithm is a greedy algorithm, that can be implemented by **Priority Queue, Max Heap, OR Min Heap** **(Here we Used Min-Heap)**, The Huffman Algorithm is used for encoding and compression of data(Text can be encoded by representing each character by a bit string (code)). It creates variable-length codes for input characters based on their frequencies(how often they appear in the text), with shorter codes assigned to more frequently occurring characters. This enables efficient storage and transmission of data, as common characters are represented by shorter codes, reducing overall file size. But before diving into the Huffman Coding we need to know what is the **Heap**.

What is a Heap?

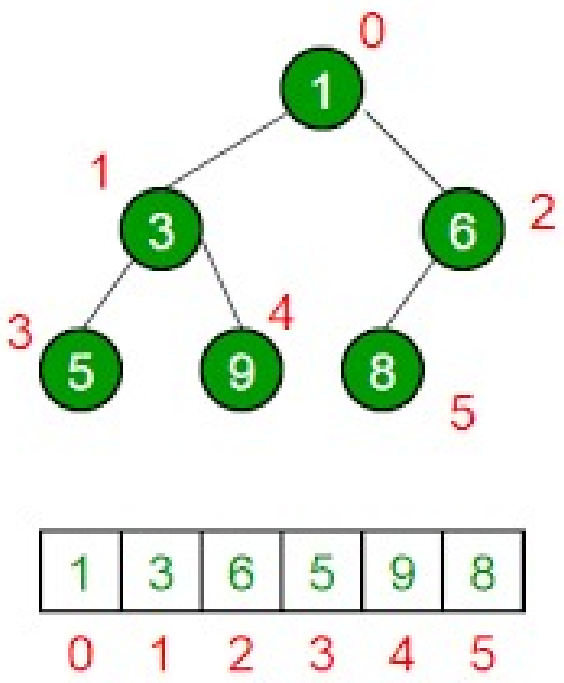
A heap is a complete binary tree, which can be classified as a min heap or a max heap based on its properties. In a min heap, the value of each node is less than or equal to its children's values, with the minimum value at the root. Conversely, in a max heap, each

node's value is greater than or equal to its children's values, with the maximum value at the root.



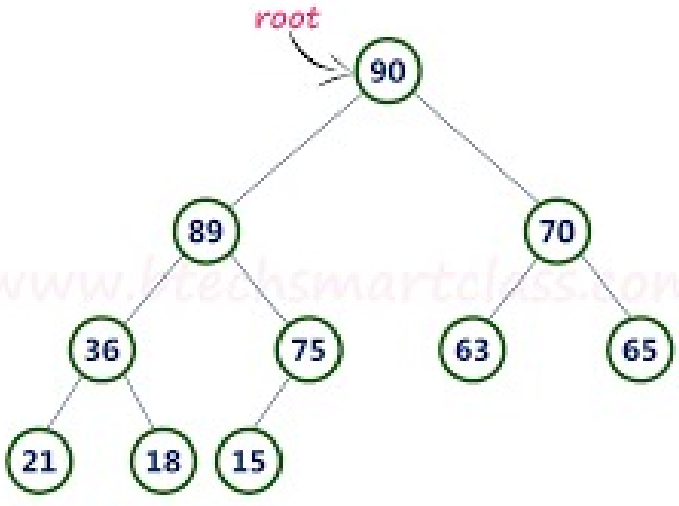
Min Heap:

A Min Heap makes sure the smallest number is always at the top of the family tree. When you add a new number to the family, you put it at the bottom and let it climb up until it finds its correct spot. When you take out the smallest number (which is always at the top), you move the last number to the top and let it fall down to where it fits.



Max Heap:

A Max Heap is the opposite. The biggest number is at the top. It works like a Min Heap, but instead of climbing up, the big numbers fall down from the top when they are added, and the biggest number is always the first to be taken out.



Huffman Coding and Min Heap:

Huffman Coding is like making a secret code where you want to use the shortest codes for the most common words. Min heaps are perfect for this because they always keep the smallest numbers at the top. Huffman Coding takes the two smallest numbers and combines them, so it's faster to always have them ready at the top, which is what Min Heaps do.

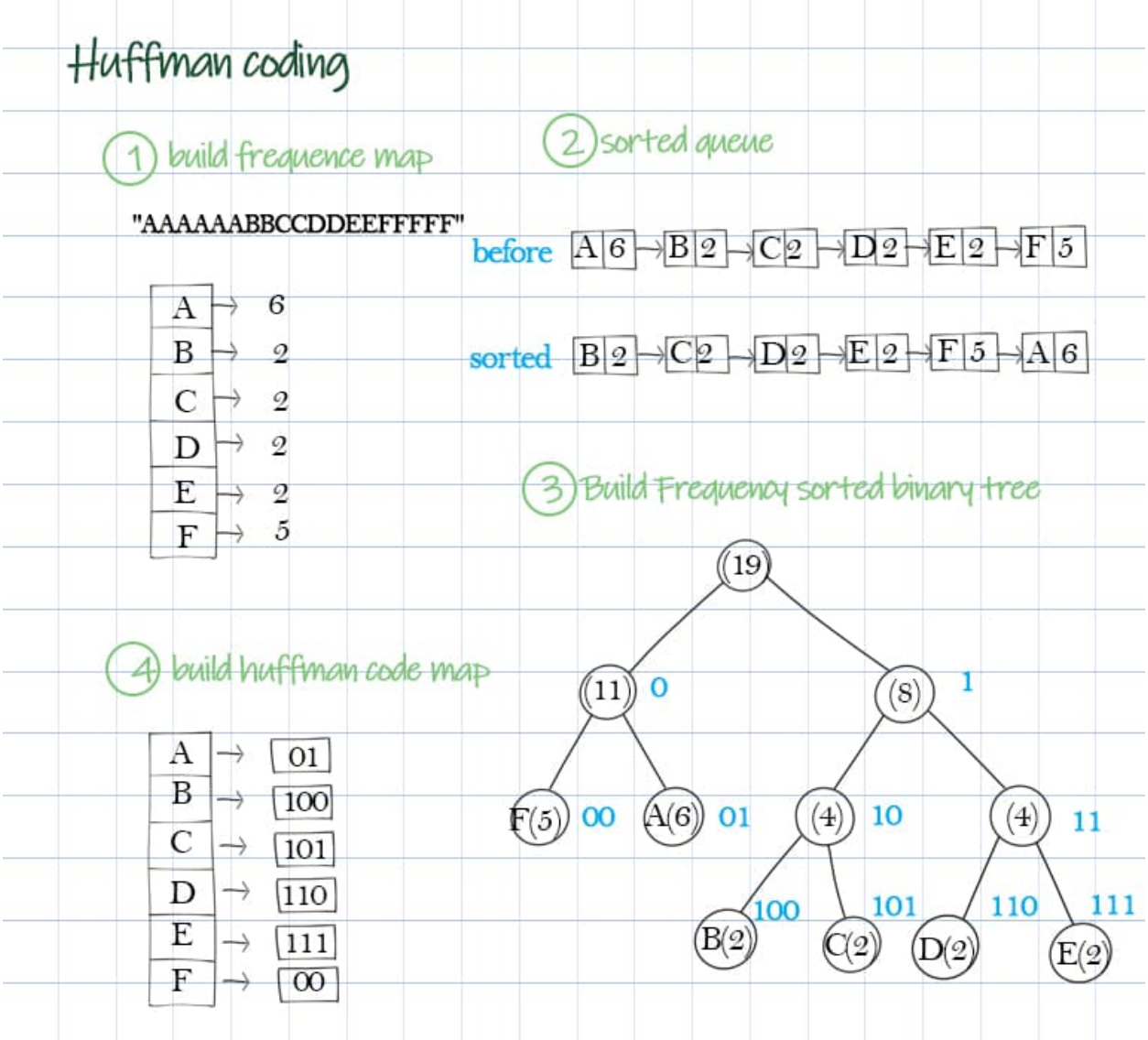
Why Min Heap is Better Than Max Heap for Huffman Coding:

Min Heaps are better for Huffman coding because they make it easy to grab the smallest numbers quickly. With Max Heaps, finding the smallest numbers would take longer because they are at the bottom, which would make the secret code take longer to make.

Think of Heaps as a smart way to keep numbers in a family tree where you can quickly find the smallest or biggest number. For making secret codes with Huffman coding, Min Heaps are like the fast lane because they keep the smallest numbers right where you need them.

Advantages of the Huffman algorithm include:

1. It's a lossless compression technique, meaning no data is lost during the compression process.
2. It's efficient and fast.
3. It results in a variable-length encoding scheme where frequently occurring characters are assigned shorter codewords, leading to more efficient compression.



3. Key Components of the Huffman Algorithm:

3.1 Frequency Table:

- The algorithm begins by constructing a frequency table that records the occurrence of each character in the input data.

3.2 Priority Queue:

- A priority queue OR ”Min-Heap” is employed to store the frequencies of characters in a sorted manner, with the least frequent characters having higher priority.

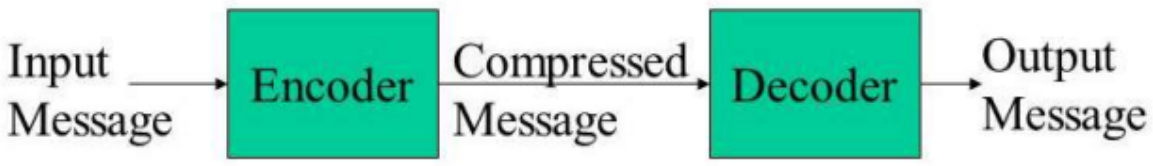
3.3 Building the Huffman Tree:

- The algorithm builds a binary tree, known as the Huffman Tree, using a bottom-up approach. It repeatedly combines the two least frequent characters into a new internal node until a single root node is formed.

3.4 Assigning Codes

- Once the Huffman Tree is constructed, the algorithm traverses the tree to assign binary codes to each character. Left and right movements represent binary '0' and '1', respectively.

3.5 Decoding & Encoding:



- The encoding process involves creating a binary tree where each leaf node corresponds to a character in the input data. Each character is then encoded as the sequence of bits on the path from the root to the corresponding leaf node.
- The decoding process is the reverse of encoding. The compressed data is traversed from the root of the tree to a leaf node, reconstructing the original data.

Decoding In Huffman:

1. start from the root.
2. read the encoded data one bit at a time.
3. branch according to the bits encountered in the input.
4. Once the bits read match a code for a character, write out the character and start collecting
   bits again.

4 Example:

Consider the input "ABRACADABRA." The frequency table and resulting Huffman Tree might look like this:

```
Character  Frequency
    A          5
    B          2
    R          2
    C          1
    D          1
```

The Huffman Tree:

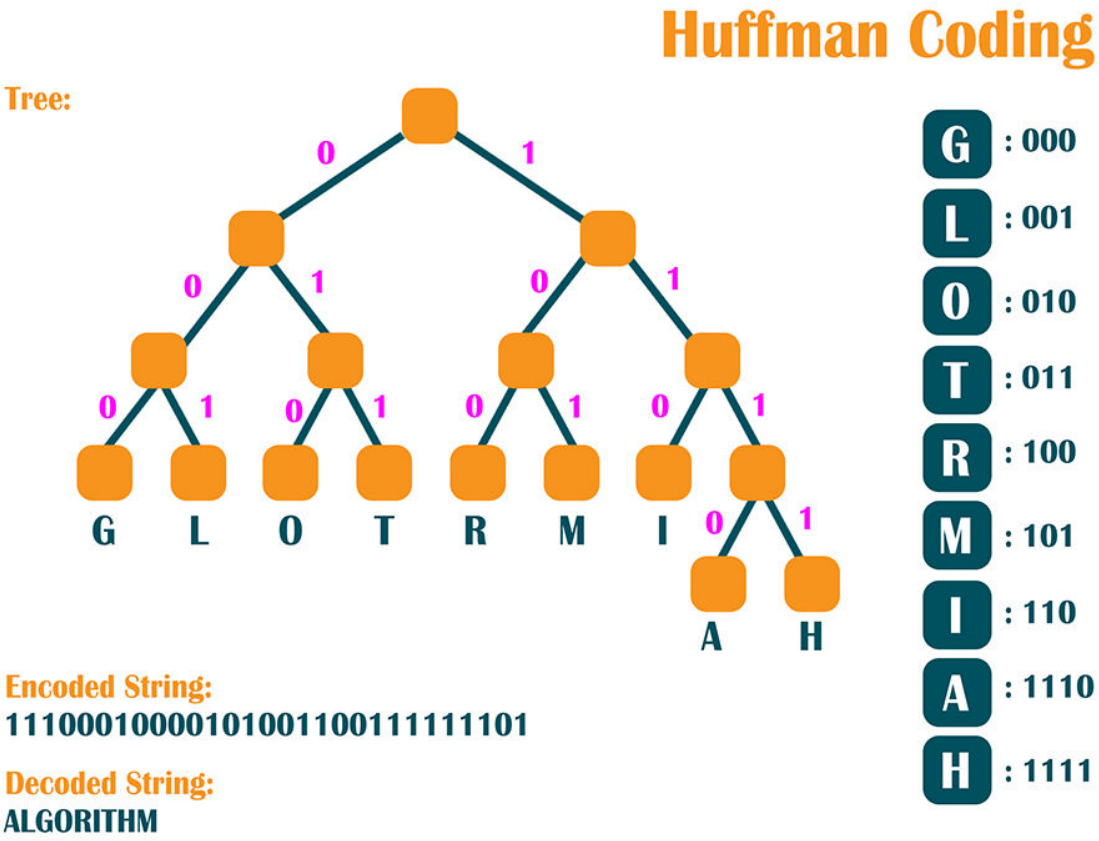
```
   (11)
  /    \
 A(5)  (6)
      /   \
    (3)   R(3)
    /  \
  B(1) C(2)
```

The codes for each character would be:

- A: 0
- B: 10
- R: 110
- C: 1110
- D: 1111

The encoded message for "ABRACADABRA" would be "01011001110011110101100."

Another example:



6. Compression and Decompression:

- The Huffman Algorithm achieves compression by replacing variable-length codes with fixed-length codes. During decompression, the original data is reconstructed using the Huffman Tree.

7. Applications:

- Huffman coding is widely used in file compression formats such as ZIP, gzip, and JPEG for image compression.

8. Resources:

The resources I relied on throughout this project included various YouTube videos and articles. Check out the links:

https://www.programiz.com/dsa/huffman-coding

https://www.youtube.com/watch?v=HqPJF2L5h9U

https://www.youtube.com/watch?v=2DX9BwRYBKM

https://www.youtube.com/watch?v=iiGZ947Tcck

https://www.geeksforgeeks.org/min-heap-in-javascript/

9. Conclusion:

The Huffman Algorithm stands as a fundamental technique in data compression, providing an efficient way to represent information. Its application extends to various domains, contributing to the optimization of storage and transmission resources.

Students:

Ahmed Alawneh, 202111391

Asem Sawafta, 202111479

Haytham Alawneh, 202111541