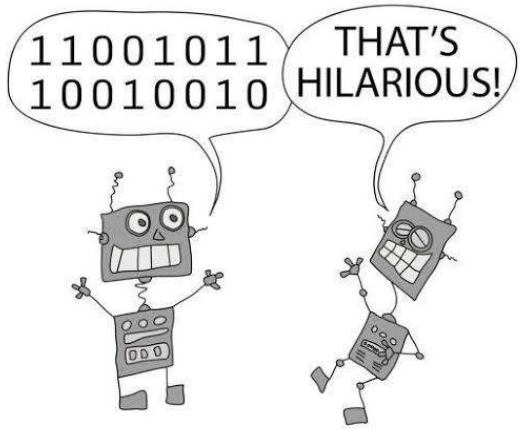
Analysis & Design of Algorithms



COURSE PROJECT: HUFFMAN TREE

Brought to you by: Farah Walid Abdelsalam

222307

INTRO
SECTION A
SECTION B
SECTION C
SECTION D
SECTION E
SECTION F

Quick Access

What is Huffman Tree?

- **The Huffman tree** is the binary tree structure used in **Huffman coding**, which is the algorithm for compressing and decompressing data by assigning variable-length codes to symbols based on their frequencies/percentages.

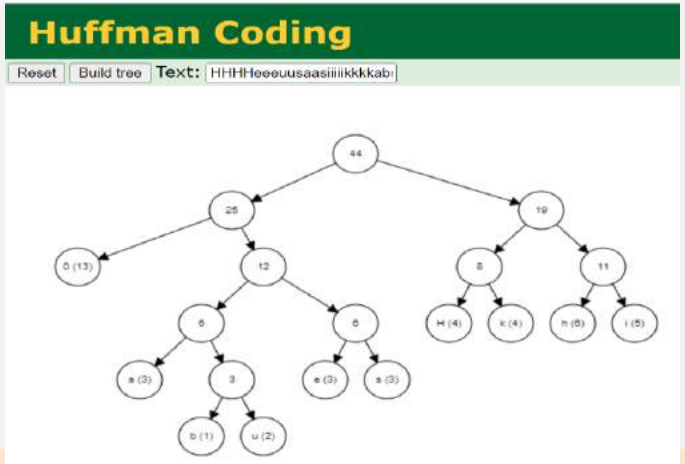
- The more frequent symbols appear in a message the shorter the code that represents it, and vice versa. The tree structure is essential for efficiently encoding and decoding the data.

-Aim of this algorithm:
Our goal here is to minimize the overall length of the encoded message.

Understanding the problem

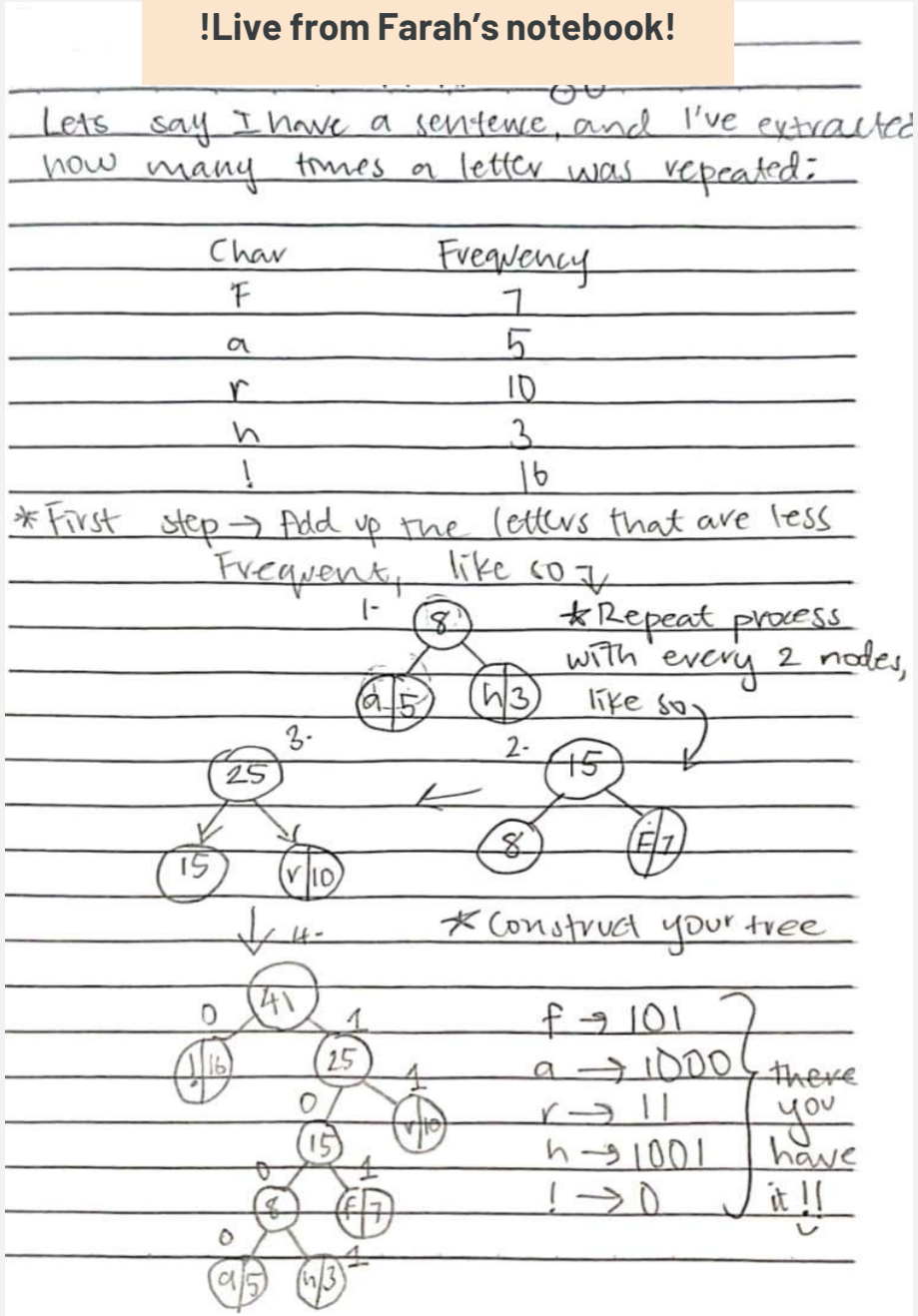
The problem:

compressing the data to save space, where the characters in the message provided are assigned codes (0->left node and 1->right node) based on the frequency of their occurrence.



Check out this website to try different examples of how the structure of the Huffman tree is:
https://cmps-people.ok.ubc.ca/ylucet/DS/Huffman.html

!Live from Farah's notebook!



Strategy used

Key strategy used in the Huffman tree:

- Greedy strategy-> It makes locally optimal choices at each step to achieve an optimal solution. This is done by merging the two nodes with the lowest frequencies at each step of building the Huffman tree.

Example

E=0.4 , T=0.3 , A=0.1 , I=0.1 , N=0.1

Code words → e=1, t=01, a=000, i=0010, n=0011

Avg bit per character → [0.4x1] + [0.3x2] + [0.1x3] + [0.1x4] + [0.1x4] = 1.7 bits //

Assume fixed length is <3> → compression ratio $\left(\frac{3-1.7}{3}\right) * 100 = 43.33\%$

The Algorithm

HUFFMAN PSEUDOCODE:

```
Function MinHeapNode(data, freq)
    node = Create a new MinHeapNode
    node.data = data
    node.freq = freq
    node.left = null
    node.right = null
    return node

Function HuffmanCodes(data[ ], freq[ ], size)
    minHeap = Create an empty MinHeap with
comparison function compare
    For i from 0 to size - 1
        Insert MinHeapNode(data[i], freq[i]) into
minHeap
While size of minHeap is not 1
left = minHeap.top(), minHeap.pop()
        right = minHeap.top(), minHeap.pop()
        top = MinHeapNode('$', left.freq + right.freq)
        top.left = left, top.right = right
        Insert top into minHeap

    Call printCodes(minHeap.top(), "")

Function printCodes(root, str)
    If root is null
        Return
    If root.data is not equal to '$'
        Output root.data + ": " + str
    Call printCodes(root.left, str + "0")
    Call printCodes(root.right, str + "1")

Function main()
    Call HuffmanCodes({'a', 'b', 'c', 'd', 'e', 'f'}, {5, 9, 12, 13,
16, 45}, 6)
    Return 0
```

Implementation

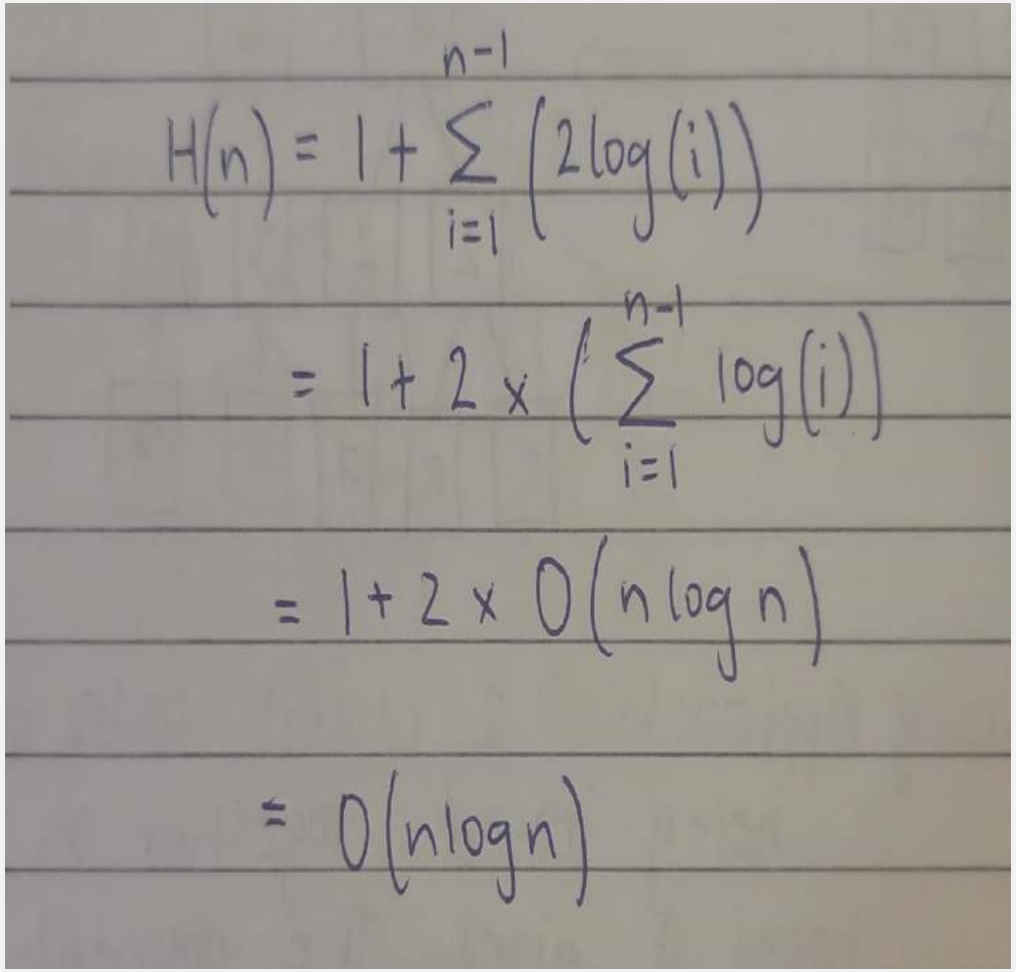
```
1   // C++ program for Huffman Coding
2   #include <iostream>
3   #include <queue>
4   using namespace std;
5
6   struct MinHeapNode {
7
8       char data;
9       unsigned freq;
10      MinHeapNode *left, *right;
11
12      MinHeapNode(char data, unsigned freq)
13      {
14          left = right = NULL;
15          this->data = data;
16          this->freq = freq;
17      }
18  };
19  struct compare {
20      bool operator()(MinHeapNode* l, MinHeapNode* r)
21      {
22          return (l->freq > r->freq);
23      }
24  };
```

Click here to open the C++ code for the Huffman coding:
https://the-algorithms.com/playground?id=2425

Analysis, Space, and Time Complexity

Mathematical analysis of the code

C(n) = 1 + \sum_{i=1}^{n-1} (2 * \log(i))
= 1 + 2 * (\sum_{i=1}^{n-1} \log(i))
= 1 + 2 * O(n * \log(n))
= O(n * \log(n))



INTRO
SECTION A
SECTION B
SECTION C
SECTION D

Analysis, Space, and Time Complexity

- The space complexity of this algorithm is broken down into the following:

->Tree construction- the tree is constructed using pointers so the space complexity, in this case, is O(n)

Overall space complexity:

O(n)= O(n)

- The time complexity of this algorithm is broken down into the following:

->Tree construction- number of iterations is proportional to the number of elements in the list, giving O(n log n)
->Displaying the tree- displaying the nodes takes O(n)

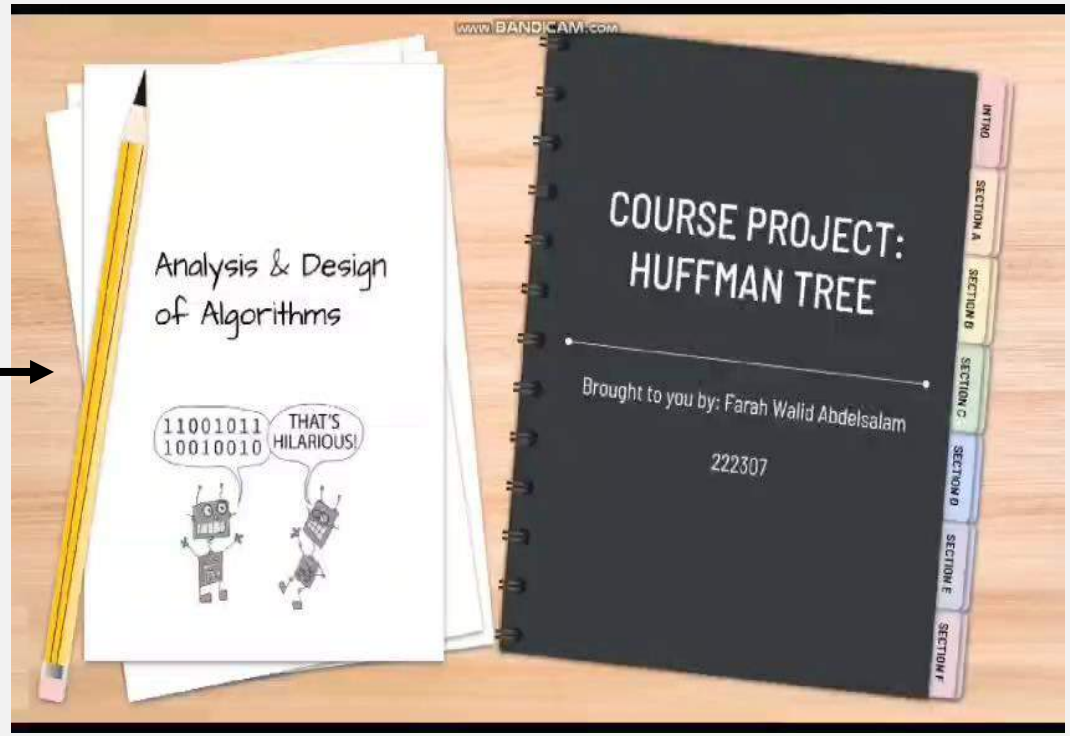
Overall time complexity:

O(n log n)+O(n)= O(n log n)

INTRO
SECTION A
SECTION B
SECTION C
SECTION D
SECTION E

Explanation Video

Watch this video for a better understanding of the HUFFMAN algorithm.



INTRO
SECTION A
SECTION B
SECTION C
SECTION D
SECTION E
SECTION F

INTRO

SECTION A

SECTION B

SECTION C

SECTION D

SECTION E

SECTION F

Thank you!

MATERIALS USED FOR THIS PROJECT:

- GITHUB LINK:

- https://github.com/Felixaleed/AlgorithmsProject/blob/main/README.md

-POWERPOINT TEMPLATE:

* SlidesMania.com

-INSPIRATION FOR HOW THE ALGORITHM WAS EXPLAINED WAS TAKEN FROM:

- LECTURE NOTES AND PROFESSOR ISLAM ELSHAARAWY
- https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/

HUGE THANKS TO:

Template created by: SlidesMania.Com